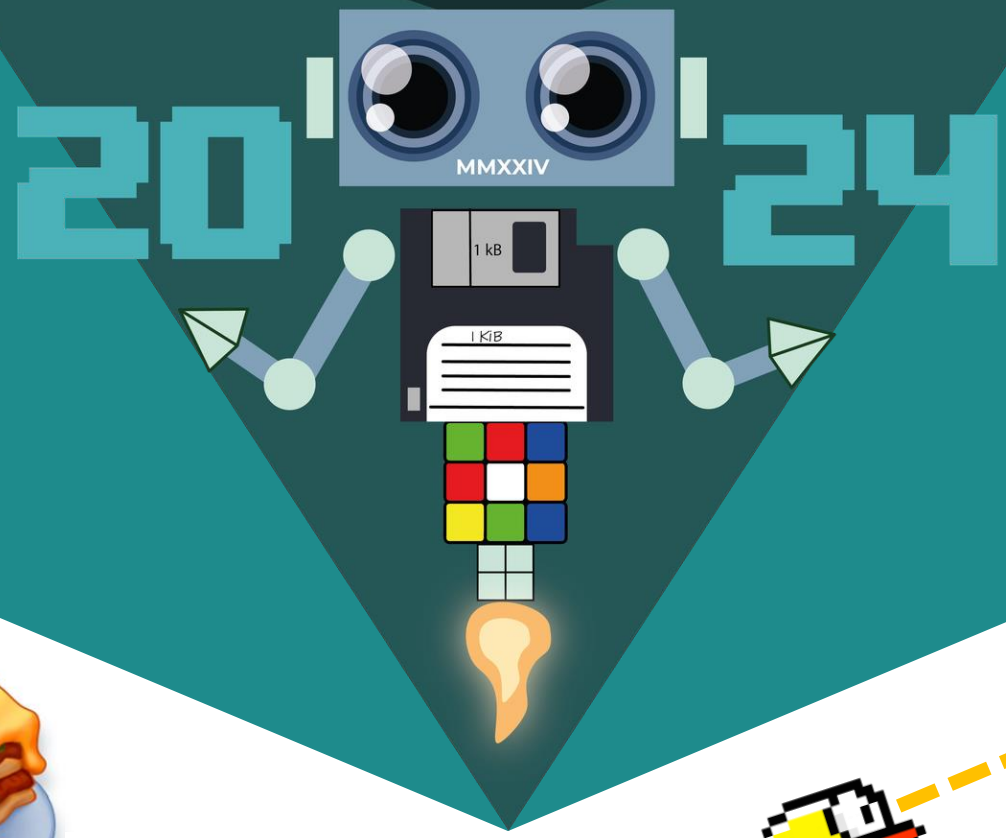
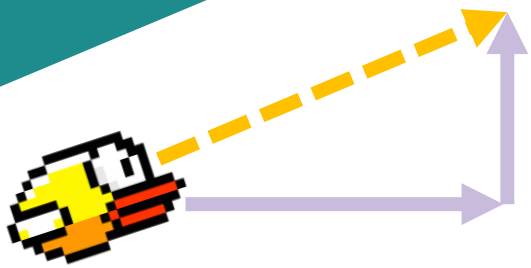
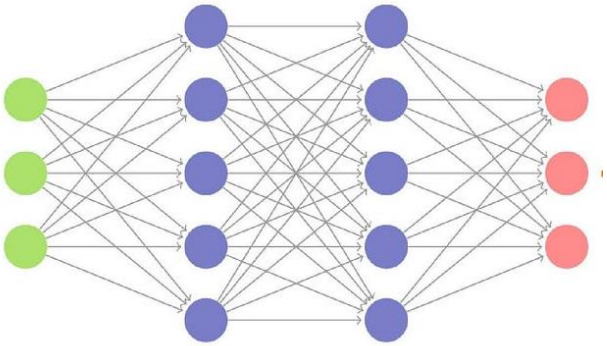
PROGRAMMERINGSKOLLO

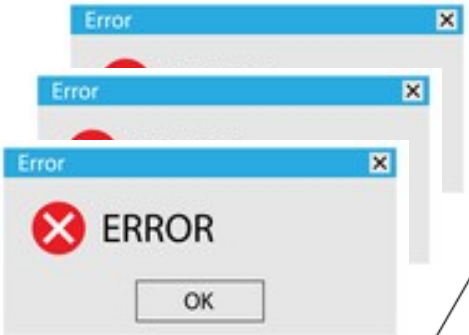


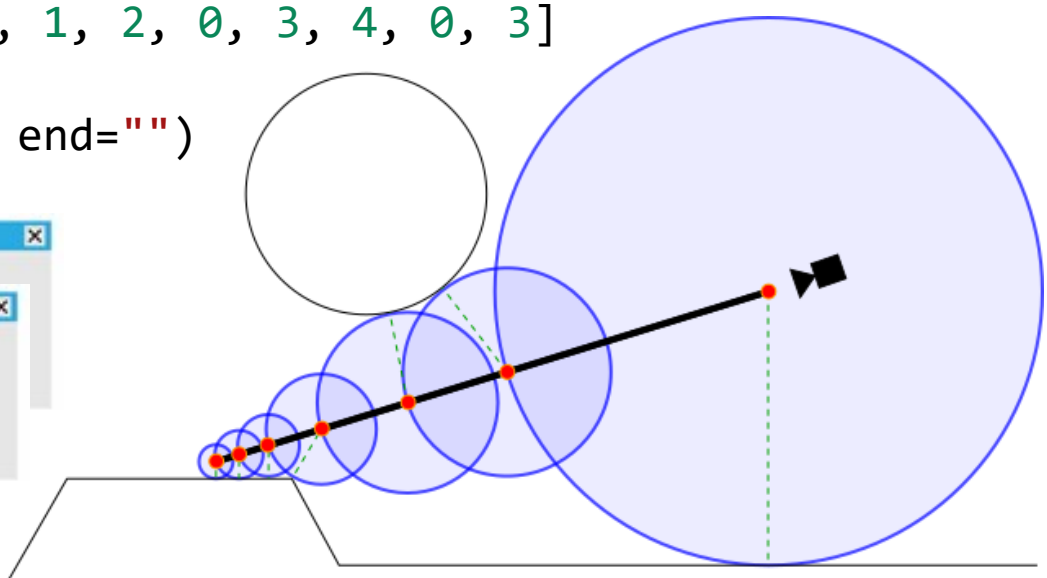




Inte en macka

```
a = ["kollo", "Programmering", "s"," är kul!\n", ”Matte"]
b = [1, 3, 4, 3, 1, 2, 0, 3, 4, 0, 3]
for i in b:
    print(a[i], end="")
```





Programmeringskollo 2024

Lektionsmaterial

J. Andersson, T. Glimmerfors, E. Lundell, F. Löfgren, H. Zhang, I. Ängquist

Linköping, juli 2024

Ada

1 Introduktion till Python I 4

2 Introduktion till Python II 8

3 Chattklient och API 12

4 API fortsättning 16

5 Tävlingsprogrammering I. Simuleringar 20

6 Tävlingsprogrammering II. Brute Force 24

1. Introduktion till Python I

16:e juli

1.1 Datatyper

Det här är några sätt man kan deklarera variabler med olika datatyper:

```
x = 5                                       # Integer
y = 4.2                                     # Float
z = "Text"                                  # String
bl = True                                   # Bool
lst = [9, False, "mango", ["!", 3]]         # List
dct = {"key1":"value1", "key2":"value2"}    # Dictionary
st = {"banan", 20, "kiwi"}                  # Set
tpl = (5, "tre", 9)                         # Tuple
```

1.2 Symbolhanterande

Testa att lagra följande värden i olika variabler, och kör koden. Vad händer?

1. Integer + Integer?

```
a = 3 + 5
print(a)
b = 2
print(a+b)
```

2. String + String?

```
a = "ost" + "macka"
print(a)

b = "milk"
c = "shake"
print(b+c)
```

3. Integer + String?

```
a = 9 + "smörgåsar"
print(a)
```

4. String * Integer?

```
a = "Echo" * 3
print(a)
```

5. Integer * Bool? (Testa även Integer + Bool!)

```
a = 9 * False
print(a)
```

6. List * Integer

```
a = [1, 2, 3] * 4
print(a)
```

7. List + List

```
a = [3, 1, 4] + [1, 5, 9]
print(a)
```

1.3 Indexering

8. Vad händer när man kör koden nedan? Gör några egna gissningar först innan ni kör koden. Reflektera sedan varför det blev som det blev.

```
x = "Hello world"
print(x[7])
print(x[1:3])
print(x[-2])
print(x[3:])
print(x[:]) # När vill man använda detta?
print(x[:1])
print(x[:-3])
print(x[-5:])
```

9. Vad händer när man kör koden nedan med $print(a)$ efter varje rad? Gör några egna gissningar först innan ni kör koden. Reflektera sedan varför det blev som det blev.

```
a = [2,3]
a[1] = 5
# <-- Vad händer ifall vi printar a här? Testa!
a[1] = [3,4,5]
# <-- Vad händer ifall vi printar a här? Testa!
a[2] = [7]
a[2:] = [7]
a[2:] = [7,8,9,10]
a[2:] = [3,4]
a[:-3] = [0]
```

10. Vad innebär indexen i och j när man skriver a[$i : j$]?

1.4 Indata/Input

För att ta in input från en användare kan man använda funktionen:

```
input()
```

Ifall man vill att programmet skriver ett meddelande innan den frågar om input, kan man göra så här:

```
namn = input("Skriv ditt namn här: ")
print("Hej " + namn + "!")
```

11. Skriv ett program som ber användaren om 2 tal, utför ett par beräkningar och sedan skriver ut svaret.

1.5 Funktioner

Ibland nöjer vi oss inte med de inbyggda funktionerna i programmeringsspråket man använder, och då kan vi definiera våra egna funktioner:

```
def add(a, b):
    return a + b
```

12. Implementera någon funktion som tar ett eller flera argument och returnerar ett lämpligt svar. Några förslag:

1. En funktion som kan räkna ut antal bananer du kan köpa ifall du har X kronor, och varje banan kostar 5 kr.
2. En funktion som konverterar antal grader i Fahrenheit till Celsius, eller tvärtom. (Googla gärna hur man räknar ut Celsius med hjälp av Fahrenheit!)
3. En funktion som konverterar längder i fot och tum till centimeter.

1.6 Omvandling mellan datatyper

13. Hur kan man konvertera en variabel mellan string, integer och float? Vilka begränsningar finns?
Studera följande kod:

```
a = "120"
b = 360
print(a + str(b))
print(int(a) + b)
```

14. Skriv ett program som läser in 2 heltal från en användare och skriver ut summan.

15. Skriv ett program som läser in 2 heltal från en användare och räkna antalet 1or i summan.
Till exempel finns det 4 st 1or i talet 11034151.

1.7 Extra: Klasser

Python används ofta för att skriva objektorienterad kod. Detta innebär att man kan skriva egna klasser, med egna medlemsfunktioner. Implementera en egen klass! Ett exempel finns nedanför:

```
class Bus:
    def __init__(self, ID, color, passengers):
        self.ID = ID
        self.color = color
        self.passengers = passengers

    def add_passengers(self, n):
        self.passengers += n

    def remove_passengers(self, n):
        self.passengers -= n

    def info(self):
        print('Bus ID: ' + self.ID + '\n' + 'Color: ' + self.color + '\n' +
            'Current number of passengers: ' + str(self.passengers))

bus1 = Bus('ABC123', 'green', 5)
bus1.info()
bus1.add_passengers(3)
bus1.info()
```

1.8 Fler instruktioner och exempel

https://docs.python.org/3/tutorial/introduction.html

2. Introduktion till Python II

17:e juli

2.1 If-satser

I programmering är det bra att kunna hantera olika sorters fall.

1. Testa runt med följande kod, och svara på frågorna.

```
person = input("Vad heter du? ")

if person == "Fredrik":
    print("Hej Fredrik! Koden är 512")
elif person[0] == "F":
    print("Oj, ditt namn börjar på samma bokstav som Fredriks namn!")
else:
    print("Hej", person, "!")
```

(a) Vad gör *if*, *elif* och *else*?

(b) Vad händer om du byter ut *elif* mot *if*?

Notera att man använder 2 likamedstecken (==) **i if-satsen för att kolla ifall 2 objekt är samma.**

2.2 While-loopar, Break och Continue

När man vill repetera en liknande process i kod ett antal gånger, så brukar man använda sig av loopar. Andra anledningar till att man använder loopar är för att spara tid och för att minska misstag.

2. Testa runt med följande kod, och svara på frågorna.

```
a = 0
while a < 10:
    print(a)
    a += 1
```

(a) Hur många gånger loopas print-funktionen?

(b) Vilka tal printas?

(c) Vad innebär olikheten efter *while*?

3. Testa runt med följande kod, och svara på frågorna.

```
a = 0
while True:
    if a == 10:
        continue
    elif a == 20:
        break
    else:
        a += 1
    print(a)
```

(a) Hur många gånger loopas print-funktionen?

(b) Vilka tal printas?

(c) Testa runt och ändra på siffrorna för att lista ut vad *continue* och *break* gör.

2.3 For-loopar

En annan vanlig loop är en For-loop.

4. Testa runt med följande kod, och svara på frågorna.

```
a = ["Harry", "Elias", "Tobias", "Fredrik", "Joshua", "Ivar"]
for namn in a:
    print(namn)
```

(a) Hur många gånger körs print-funktionen här? Vad beror det på?

(b) I vilken ordning printas namnen?

(c) Vad händer om vi loopar igenom en sträng istället för en lista?

```
for x in "Mattekollo":
    print(x)
```

5. Testa runt med följande kod, och svara på frågorna.

```
for number in range(5):
    print(number)
```

(a) Hur många gånger loopas print-funktionen? Alltså, hur många gånger kommer variablen *number* att printas?

(b) Vilka tal printas?

(c) Vad är *range*(5)?

(d) Skapa en for-loop som printar de 5 första jämna talen.

(e) Printa ut namnen i listan genom att använda:

```
a = ["Harry", "Elias", "Tobias", "Fredrik", "Joshua", "Ivar"]
for i in range(len(a)):
    ... (här ska DU skriva kod!)
```

2.4 Sets

Förutom listor finns även en annan vanlig och användbar datastruktur, och det kallas för set. Det fungerar som en "mängd" i matematik.

2.5 Lägga till element i listor och sets

Att lägga till och ta bort element i listor och sets använder sig av olika sorters funktioner i Python. Till exempel:

```
frukter = ["banan", "kiwi"]
frukter.append("druvor")
frukter.pop()    #Tar bort det sista elementet
frukter.pop(1)   #Vad innebär siffran här? Testa runt och skriv ut grejer!

harrys_favoritsiffror = {1,1,1,1,2,2,2,2,0}
harrys_favoritsiffror.add(7)
harrys_favoritsiffror.remove(1)
```

Testa skriva ut variablerna för att se vad som finns i dem efter varje funktion!

2.6 Övningar listor

6. Skapa en lista med alla kvadrattal mellan 1 och 100.

7. Skapa en lista med alla tal mellan 1 och 100 som innehåller siffran 6.

8. Utgå från listan ['Björn', 'Hund', 'Katt', 'Häst']. Lägg till ett utropstecken till alla strängar, och skriv ut strängarna.

9. Utgå från listan ['Björn', 'Hund', 'Katt', 'Häst']. Filtrera bort strängar som inte innehåller bokstaven 'n'.

2.7 Övningar sets

10. Givet en lista av positiva heltal där det kan finnas dubbletter, skriv en kod som kan printa ut alla **unika** tal som uppstår i listan. (Tips! Använd sets.)

11. Givet en lista av positiva heltal, skriv en kod som räknar ut antalet möjliga summor man kan skapa genom att addera 2 tal från listan.

2.8 Ordböcker/Dictionary

Ordböcker fungerar som en set, men för varje element/nyckel i ordboken så är den kopplad med ett specifikt värde.

```
svenska_till_engelska = dict()
svenska_till_engelska["apelsin"] = "orange"
svenska_till_engelska["jordgubbe"] = "strawberry"
```

2.9 Övningar ordböcker

12. Skapa en dictionary vars nycklar är talen från 1 till 15 (båda inkluderas) och värdena är kvadraten av nycklarna.

13. Skapa en dictionary med alla orden i denna delfråga där orden är nyckeln och ordens längd är dess värden.

2.10 Problemlösning!

14. Lös följande uppgift: https://projecteuler.net/problem=9

15. https://projecteuler.net/problem=14

16. https://projecteuler.net/problem=25

Fler problem finns här: https://projecteuler.net/archives

3. Chattklient och API

18:e juli

Denna lektion går igenom hur man kan göra en chattklient och hur man kan använda ett enkelt API (Application Programming Interface - sätt för olika program att prata med varandra). Vi kommer att lära oss hur man skickar data till en server och hur man kan hantera datan man får tillbaka. Vi kommer att använda oss av en server som vi har satt upp på `https://chat.eliaslundell.se`. Om du öppnar hemsidan i din webbläsare kan du se alla meddelanden som har skickats hittills.

Hämta meddelanden

1. Först vill vi hämta alla meddelanden från servern. Detta gör vi genom att skicka ett GET-anrop till `/posts` med följande kod:

```
import requests

address = "https://chat.eliaslundell.se"

meddelanden = requests.get(address + "/posts").json()
print(meddelanden)
```

Vi får alla meddelanden i en dictionary som innehåller mycket information, till exempel vem som har skrivit meddelandet, när det skrevs, vilken användare som har skrivit det och vad meddelandet innehåller.

2. När vi skriver ut detta är det lite svårläst, vi använder oss av en `for`-loop för att göra en snyggare utskrift.

```
import requests

address = "https://chat.eliaslundell.se"

meddelanden = requests.get(address + "/posts").json()
for meddelande in meddelanden:
    print(f"{meddelande['userid']}: {meddelande['message']}")
```

3. Dock ser vi inte vem som har skrivit meddelandet. Allt vi ser är olika siffror som motsvarar olika användares ID. Som tur är kan vi få reda på alla användare och vilka ID de har genom att skicka ett GET-anrop till `/users`.

```
användare = requests.get(address + "/users").json()
print(användare)
```

4. Vi vill nu matcha ihop vilken användare som har vilket namn med följande kod:

```
import requests
```

```
address = "https://chat.eliaslundell.se"

användare = requests.get(address + "/users").json()
meddelanden = requests.get(address + "/posts").json()

for meddelande in meddelanden:
    namn = "okänd"
    for anv in användare:
        if anv['id'] == meddelande['userid']:
            namn = anv['name']
            break
    print(f"{namn}: {meddelande['message']}")
```

5. Nu börjar det se bra ut, men något du kanske märkt är att den skriver ut meddelande i omvänd ordning mot vad vi vill. I en chatt är oftast det senaste/nyaste meddelandet längst ner. För att ändra ordningen kan vi använda oss av `.reverse()`-metoden på listan.

```
meddelanden.reverse()
```

Skriva meddelanden

> Här behöver du en TOKEN för att kunna skicka meddelanden. Fråga en ledare så får du en. Spara den sedan i en variabel högst upp i koden:
>
> ```
> TOKEN = "[DIN TOKEN HÄR]"
> ```

6. Nu vill vi skicka meddelanden. Det gör vi genom att skicka ett POST-anrop till `/post`. Denna måste innehålla meddelandet vi vill skicka, samt vår hemliga TOKEN, så att servern vet att det är just du som har skickat meddelandet.

```
meddelande = input("Meddelande att skriva: ")
requests.post(address + "/post", json = {"token": TOKEN, "message":
    meddelande})
```

Meny

7. Nu får användaren inget val utan måste först läsa alla meddelanden och skriva ett svar. För att lösa detta kan vi fråga användaren vad den vill göra:

```
while True:
    val = input("Vad vill du göra (läs, skriv, avsluta)? ")
    if val == "läs":
        ...
    elif val == "skriv":
        ...
    elif val == "avsluta":
        break
```

Placera sedan koden för att ladda in och printa ut alla meddelanden under ”läs” och all kod för att skicka ett meddelande under ”skriv”.

Ladda alla meddelanden (paginerat API)

8. Nu när chatten förhoppningsvis har börjat få många meddelanden i sig har du kanske märkt att den bara visar de senaste 10 meddelanderna. Det är ganska vanligt att APIer endast skickar lite data åt gången, speciellt om det finns väldigt mycket information. För att ladda in äldre meddelanden kan vi lägga till en ”page” parameter.

```
meddelanden = requests.get(address + "/posts", params = {"page": 0})
```

Om parametern är 0 får vi meddelande nr 1-10, om den är 1 får vi meddelande nr 11-20 osv. Om användaren väljer att läsa meddelanden, skriv först ut de första 10 och fråga sedan om användaren är färdig, eller om hen vill se äldre meddelanden. Detta kan t.ex. göras med följande kod:

```
    page = 0
    while True:
        ... # koden som läser in och skriver ut meddelanden
        val = input("Vill du se fler meddelanden (ja, nej)? ")
        if val == "ja":
            page += 1
        else:
            break
```

9. Nu läser vi sida för sida, gör så att den automatiskt läser in alla meddelanden som finns, och se till att avbryta när den är färdig så den inte frågar servern efter fler sidor i all oändlighet.

10. Ibland vill man prata med bara en person, skriv ut en lista med alla användare på servern, låt din använadare välja en av de och skriv sedan endast ut meddelanden som den personen har skickat.

Extrauppgift

11. Kryptera dina meddelanden med ett ceasar krypto och deshiffrera alla meddelanden som du läser in. Dina meddelanden är nu krypterade, alla andra kan se vad du har skickat, men kan inte läsa texten. Du och dina kompisar kan skicka meddelanden normalt till varandra.

4. API fortsättning

20:e juli

Igår gick vi igenom ett enkelt API mot en chattserver. Denna lektion kommer gå igenom mer komplexa APIer. Bland annat hur man kan läsa in väderdata och väderprognoser och hur man kan generera text och bilder med OpenAI.

Väder

1. Först testar vi att läsa in väderdata. Vi kan med följande kod få ut temperaturen i Linköping just nu.

```
import requests

WEATHER_KEY="4016360a2ee049889ee213659241507"
response = requests.get("http://api.weatherapi.com/v1/current.json?" \
                        f"key={WEATHER_KEY}&q=Linköping&aqi=no")
data = response.json()

print(data["current"])
print(str(data["current"]["temp_c"]) + " grader celsius")
```

Uppdatera koden och skriv utöver temperaturen även ut luftfuktigheten för Linköping. Skriv ut temperaturen och luftfuktigheten (här får du leta i `data['current']` för att hitta rätt nyckel) för din hemmastad (välj valfri annan stad om du bor i Linköping).

2. Med `api.weatherapi.com` kan vi även få en prognos för framtiden. Hämta en väderleksrapport 3 dagar framåt via forecast-endpointen.

```
response = requests.get("http://api.weatherapi.com/v1/forecast.json?" \
                f"key={WEATHER_KEY}&q=Linköping&days=3&aqi=no&alerts=no")
data = response.json()
print(data)
```

Som du ser när du skriver ut data har den väldigt mycket information i sig. Vi får en prognos för varje timmar de kommande tre dagarna. Detta är lite mycket data, så vi börjar med att ta ut endast en mätpunkt för varje dag.

```
for day in data["forecast"]["forecastday"]:
    print(day["hour"][12])
```

Skriv ut `avgtemp_c`, `daily_chance_of_rain` och `conditions['text']` för klockan 12 varje dag. Skriv även ut den högsta och lägsta temperaturen som kommer inträffa under de kommande 3 dagarna.

OpenAI

3. Härnäst ska vi använda oss av OpenAI's GPT-4o model för att generera text. Det går att använda `requests`, men OpenAI har gjort ett pythonbibliotek för att underlätta både att skicka och hämta data. Importera paketet med `pip install openai`. Vi kan sedan skicka en prompt med följande kod:

```
from openai import OpenAI

client = OpenAI(
    api_key = "sk-proj-gJSh5Y7loGR1vg1tMPmbT3BlbkFJzAGqaJv05KvB1qbpbiSq"
)

chat_completion = client.chat.completions.create(
    messages=[
        {
            "role": "user",
            "content": "Vad heter Danmarks huvudstad?"
        }
    ],
    model="gpt-4o",
)
print(chat_completion.choices[0].message.content)
```

Notera hur vi hanterar svaret. Istället för `data['nyckel']` som vi har gjort med `requests` kan vi nu skriva `data.nyckel`.

4. Kombinera väderprognosen med OpenAI GPT. Skicka in temperaturen, risken för regn och väderförhållandet för varje dag och be GPT göra en väderleksrapport i vanlig svenska istället för massa siffror. Inkludera även min- och maxtemperaturen i prognosen.

5. Vi kan använda en annan tjänst från OpenAI, Dall-E 3, för att generera bilder. Använd följande kod och sätt in den väderbeskriving som du fick från förra uppgiften för att visualisera den.

```
from openai import OpenAI
import requests
client = OpenAI(
    api_key = "sk-proj-gJSh5Y7loGR1vg1tMPmbT3BlbkFJzAGqaJv05KvB1qbpbiSq"
)

response = client.images.generate(
  model="dall-e-3",
  prompt="En dramatisk bild som visar: [...].", # Sätt in väderbeskrivingen här
  size="1024x1024",
  quality="standard",
  n=1,
)

image_url = response.data[0].url
```

```python
print(image_url)
response = requests.get(image_url)

if response.status_code == 200:
    with open("vader.png", "wb") as file:
        file.write(response.content)
        print("File downloaded successfully!")
else:
    print("Failed to download the file.")
```

Exprimentera med prompten och skapa andra bilder. Gör en logga för Mattekollo och Programmeringskollo 2025. Gör en bild som föreställer den mystiske onsken.

6. GPT-4o kan inte bara generera text och bilder utan vi kan även skicka bilder till den som den kan beskriva och analysera. Ladda ner en bild från internet och spara den som `image.png` brevid din -python-fil.

```python
from openai import OpenAI
import base64

client = OpenAI(
  api_key = "sk-proj-gJSh5Y7loGR1vg1tMPmbT3BlbkFJzAGqaJv05KvB1qbpbiSq"
)

def encode_image(image_path):
  with open(image_path, "rb") as image_file:
    return base64.b64encode(image_file.read()).decode('utf-8')

with open("image.png", "rb") as image_file:
  image_description = client.chat.completions.create(
    model="gpt-4o",
    messages=[
      {
        "role": "user",
        "content": [
          {
            "type": "text",
            "text": "Create a description of the attached image."
            },
          {
            "type": "image_url",
            "image_url": {
              "url": f"{encode_image('image.png')}",
              },
            },
          ],
        }
    ],
    max_tokens=300)

  print(image_description.choices[0].message.content)
```

7. Kombinera dessa två tjänster för att generera nya högupplösta bilder för brädspelet Codenames. Ladda ner bilder från spelet från `v5.nu/wpb8`.

Figur 4.1: Codenames bilder

a) Börja med att göra en loop och låt GPT-4o beskriva varje bild. Skriv ut denna i terminalen. Ändra prompten till GPT-4o tills du är nöjd med beskrivningarna, hur påverkas den om du ber GPT-4o vara specifik med många detaljer eller endast ha övergripande drag?

b) Skicka sedan in dessa beskrivningar till Dall-E 3 (som i uppgift 5). Testa lägga till extrainstruktioner, som att alla bilder ska se ut som att de kommer från en gammal bok, eller att de ska vara målade med vattenfärg.

8. Skapa en chat med GPT-4o. Låt användaren skriva frågor med `input()` och skicka dessa till GPT-4o. Skriv ut svaren du får från GPT-4o med `print()`. För att GPT-4o ska kunna se historiken i chatten måste man ha med de i historiken.

```
messages = ""
while True:
    messages += "Human: " + input() + "\n"

    # generera svar från GPT-4o, se problem 3.

    print(chat_completion.choices[0].message.content)
    messages += "AI: " + chat_completion.choices[0].message.content + "\n"
```

Extrautmaningar/Större uppgifter

Här finns förslag på några större uppgifter som man kan lösa med OpenAIs modeller. Välj den du tycker verkar mest kul/utmanande och börja med den.

9. Prompta fram en AI som har koll på ett område, säg kvantmekanik, och hjälper dig att lära dig mer om området genom att ställa frågor till dig och lära dig om området på din nivå.

10. Skapa ett äventyrsspel där du går runt och interagerar med agenter som kan prata med dig. Du kan ha olika prompts för olika agenter, för att forma deras personligheter. Spelet ska vara textbaserat så du får olika val i hur du ska röra dig runt, till exempel ”Vill du gå till mataffären eller kyrkan?” och där kan du träffa på olika personer.

5. Tävlingsprogrammering I. Simuleringar

21:a juli

1. **Frodiga Fält Från Fredrik (FFFF)**

Fredrik satt på sitt kontor och tittade ut genom fönstret, han funderade på livet och vad som egentligen är viktigt. Ska han verkligen spendera resten av sitt liv framför en datorskärm och skriva programmeringskod? Det låter rätt meningslöst.

Det var dags för en förändring och Fredrik bytte han ut sitt tangentbord och mus mot en kratta och spade och startade ett trädgårdsföretag!

Efter flera års hårt arbete utomhus har Fredrik nu biceps som Van Damme och äger det främsta trädgårdsföretaget i hela Sverige. Han har precis haft tur nog att förhandla fram ett stort avtal för att så rektangulära gräsmattor åt godsägare.

Varje avtal specificerar storleken på de gräsmattor som behöver sås, samt kostnaden för frön per kvadratmeter. Hur mycket behöver han spendera på frön?

Körningsexempel:

```
Kostnad för frön per kvadratmeter? 20
Antal gräsmattor? 3
Bredd på grässmatta 1? 2
Längd på gräsmatta 1? 3
Bredd på grässmatta 2? 4
Längd på gräsmatta 2? 5
Bredd på grässmatta 3? 5
Längd på gräsmatta 3? 6
Svar: 1120 kronor

Kostnad för frön per kvadratmeter? 0.75
Antal gräsmattor? 2
Bredd på grässmatta 1? 2
Längd på gräsmatta 1? 3.33
Bredd på grässmatta 2? 3.41
Längd på gräsmatta 2? 4.567
Svar: 16.68 kronor
```

2. **A Furious Cocktail**

I Minecraft kan drycker (potion) skapas och drickas. När en dryck dricks, appliceras motsvarande dryckseffekt på spelaren under en viss tid. ”A Furious Cocktail” är namnet på bedriften att lyckas få alla dryckeseffekter samtidigt. Detta är en utmanande prestation, eftersom det tar tid att dricka varje dryck, och endast en dryck kan drickas åt gången.

Antag att det finns N unika drycker i din inventory, som var och en har en aktiv tidsperiod på $t_1, \ldots, t_N$ sekunder. Om det tar T sekunder att dricka en dryck, är det möjligt att ha alla N dryckseffekter applicerade samtidigt? Observera att

om en dryckseffekt tar slut samtidigt som du avslutar att dricka en annan dryck, räknas inte dryckerna som applicerade samtidigt.

Körningsexempel:

```
Antal drycker? 5
Antal sekunder det tar att dricka en dryck? 1
Tid dryck 1 är aktiv? 1
Tid dryck 2 är aktiv? 2
Tid dryck 3 är aktiv? 3
Tid dryck 4 är aktiv? 4
Tid dryck 5 är aktiv? 5
Svar: Ja, det går!

Antal drycker? 3
Antal sekunder det tar att dricka en dryck? 2
Tid dryck 1 är aktiv? 4
Tid dryck 2 är aktiv? 3
Tid dryck 3 är aktiv? 2
Svar: Nej, det går inte.

Antal drycker? 3
Antal sekunder det tar att dricka en dryck? 2
Tid dryck 1 är aktiv? 2
Tid dryck 2 är aktiv? 3
Tid dryck 3 är aktiv? 5
Svar: Ja, det går!
```

3. Snöskottning

Scott har som många andra haft problem under vintertiden med allt snö. Scotts hus måste skottas ofta, men han tycker själv att skotta varje dag är för ofta. Du ska hjälpa Scott att sätta upp ett skott-schema för de kommande N dagarna. Till din hjälp har du en detaljerad väderleksprognos, så du vet exakt hur många cm det snöar eller smälter varje dag (innan kvällen). När det ligger minst H cm snö en kväll så är det dags att skotta, och när man skottar så försvinner all snö. Från början är det ingen snö alls.

Indata

Första raden innehåller antalet dagar $1 \leq N \leq 100$ och andra raden snötröskeln $1 \leq H \leq 1000$. Därefter följer N heltal mellan -100 och 100 (inklusive). Dessa tal beskriver hur många cm snö det kommer under var och en av de kommande N dagarna. Ett negativt värde betyder att denna mängd smälter istället (men snötäcket kan naturligtvis aldrig bli mindre än 0).

Utdata

Utdatat ska bestå av ett heltal, antalet kvällar Scott måste skotta.

Körningsexempel:

```
Antal dagar? 10
Snögräns? 27
Snöförändringar? 5 -7 8 19 -20 22 8 26 -15 14
```

```
Scott behöver skotta minst 2 gånger
```

4. **Robotar**

Fredrik gillar skattjakter, jättemycket. Och robotar, faktiskt ännu mer. Nu har han programmerat en robot att följa en skattkarta.

På skattkartan är varje ruta markerad med pilar för att visa åt vilket håll roboten ska gå från den rutan.

Roboten börjar alltid i den ruta som befinner sig längst upp till vänster på skattkartan, och följer därefter pilarna. I labyrinten finns det två olika mål: ett batteri, samt en läskig elstöt. Det kan också hända att skattkartan leder runt roboten i en oändlig cykel av rutor så den aldrig når ett mål.

Kan du hjälpa roboten att avgöra vilket mål den når, eller om den kommer gå runt i all oändlighet.

Indata

Följande tecken förekommer i skattkartan:

- ”<” – ruta med vänsterpil,
- ”>” – ruta med högerpil,
- ”v” – ruta med nedåtpil,
- ”^” – ruta med uppåtpil,
- ”B” – rutan batteriet befinner sig på,
- ”E” – rutan elstöten befinner sig på.

Roboten börjar på den första rutan i den första raden av skattkartan. Skattkartan är konstruerad så att roboten aldrig kommer lämna labyrinten när den följer pilarna.

Utdata

Ditt program ska skriva ut vad som händer för roboten.

Körningsexempel:

```
vB<
vE^
>>^
Roboten kommer gå till Batteriet.
v>>v
>^Ev
vv^v
B<^<
Roboten får en elstöt.
v<E
>^B
>>^
Roboten kommer gå runt, runt, runt för evigt.
vv<<<<<
```

```
vv>>v^^
vv^<vB<
vE^v<<^
v>^v^<^
>^<>>>^
Roboten kommer gå till Batteriet.
```

5. **Tågväxeln**

Innan Valla Folkhögskola öppnade fanns endast en tågstation på den här platsen. Det finns kvar spår och räls bakom hästarna på ett järnvägsmuseum.

Tågstationens föreståndare, Viveka, hade jobbat där i många år. Hennes jobb gick ut på att styra stationens manuella tågväxel. Det fanns bara två tåglinjer, antingen till Norrköping eller till Motala. Tågen gick periodiskt med m respektive n minuters mellanrum på de två linjerna, med första avgång m respektive n minuter efter midnatt. Alla tåg åkte alltså ut från stationen åt samma håll men delades sedan upp på två olika spår med hjälp av en växel.

Viveka fick lön baserat på hur många gånger hon måste ändra växeln på en dag. Arbetsgivaren undrade hur många gånger som Viveka måste ändra på växeln under ett helt dygn (dvs 1440 minuter). Tågen avgår alltså bara under minuterna 00:00 till 23:59.

Viveka skulle ändra växeln enligt reglerna:

1. Om ett tåg ska avgå, och växeln är fel inställd, måste Viveka ändra växeln till rätt spår.
2. Om båda tåg ska avgå samma minut, så avgår först det tåg som växeln är inställd för, och sedan ska Viveka ändra växeln till det andra tågets spår.

I början var växeln inställd på spåret för det tåg som avgår först.

Skriv ett program som beräknar hur många gånger som Viveka måste ändra växeln under ett helt dygn.

Körningsexempel

```
Talet m ? 450
Talet n ? 180
Antal växlingar: 5

Talet m ? 500
Talet n ? 1000
Antal växlingar: 1

Talet m ? 719
Talet n ? 720
Antal växlingar: 2

Talet m ? 7
Talet n ? 4
Antal växlingar: 410
```

6. Tävlingsprogrammering II. Brute Force

22:a juli

1. **Permutationer**

   a) Skriv ett program som skriver ut alla möjliga strängar, av längd 3, som man kan bilda med bokstäverna 'a', 'b' och 'c'. Det ska bli 27st strängar, varför?

   b) Modifiera programmet i a) så att varje bokstav bara får förekomma en gång. Det ska bli 6st strängar, varför?

   c) Skriv ut alla permutationer av strängen "EMMA".

   d) Modifiera programmet i c) så att 'M' bara används en gång.

2. **Send more money**

   Ersätt varje bokstav med en unik siffra mellan 0 och 9 så att uträkningen i figur 6.1 stämmer. Notera att talen inte kan börja med 0, för i så fall skulle man kunna strunta i att skriva ut första bokstaven.

```
  SEND
+ MORE
------
 MONEY
```

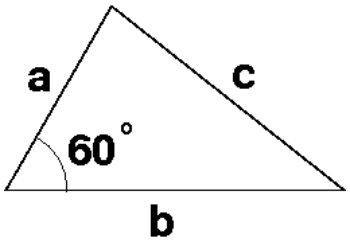
Figur 6.1

3. **Lagomvinklade trianglar**

   En lagomvinklad triangel är vad vi i denna uppgift kallar en triangel där minst en av vinklarna är exakt 60 grader. De lagomvinklade trianglarna känner sig ofta förbisedda jämfört med de mycket mer kända rätvinkliga trianglarna (så kallat mindervinkelkomplex), trots att de lagomvinklade också har en snygg formel för sina sidlängder:

   $c^2 = a^2 + b^2 - ab$

   Skriv ett program som skipar lite rättvisa i detta triangeldrama genom att fråga efter ett tal N (mellan 1 och 100) och sedan skriva ut hur många lagomvinklade trianglar det finns vars sidor är heltal i intervallet 1 till N.



Figur 6.2: Ett exempel på en lagomvinklad triangel med sidlängderna $a = 5, b = 8$ och $c = 7$.

Körningsexempel

```
Talet N ? 25
Antal trianglar: 35

Talet N ? 70
Antal trianglar: 112
```

4. Uppställning

Figur 6.3: Raden med barn i exemplet sedda bakifrån.

På idrotten så ville en grupp med 8 deltagare utmana idrottsledaren Ellen förmåga att tänka. Utan att hon ser dem ställer de upp sig på en rad. Sen räknar vart och en av dem hur många av de barn som står till vänster om honom/henne som är längre än han/hon själv, och sedan likadant med dem som står till höger. Var och en skriver ner dessa antal på en lapp som de ger till Ellen efter att ha frångått uppställningen. Deras enkla uppmaning till Ellen var att tala om i vilken ordning de stod.

Ett exempel med fem barn visas i figur 6.3 ovan. A har ett längre barn (D) till vänster om sig och två (C och E) till höger. B har tre längre barn till vänster om sig och ett till höger. C har ett längre barn till vänster om sig men inget till höger och så vidare. Informationen på lapparna kan sammanfattas så här:

Barn	Vänster	Höger
A	1	2
B	3	1
C	1	0
D	0	0
E	2	0

Tyvärr klarade Ellen inte nöten utan måste i hemlighet smyga iväg och skriva ett datorprogram som löser uppgiften. Hon kan förutsätta att alla barn är olika långa och att de inte har gjort något misstag när de skrev lapparna. Intressant nog finns det aldrig mer än en lösning.

Körningsexempel

```
Barn A, vänster ? 2
Barn A, höger ? 0
Barn B, vänster ? 0
Barn B, höger ? 1
Barn C, vänster ? 0
Barn C, höger ? 6
Barn D, vänster ? 2
Barn D, höger ? 3
Barn E, vänster ? 0
Barn E, höger ? 0
Barn F, vänster ? 3
Barn F, höger ? 1
Barn G, vänster ? 2
Barn G, höger ? 1
Barn H, vänster ? 6
Barn H, höger ? 1

Uppställningen: C B E D G F H A
```

5. Klockan

Figur 6.4

Om någon frågar hur mycket klockan är, svarar de flesta ”kvart över fem”, 15:29 eller något liknande. Vill man göra det lite svårare så kan man annars svara med vinkeln mellan tim- och minutvisaren, eftersom man ur denna information entydigt kan bestämma klockslaget. Dock är det många människor som är ovana vid detta sätt att ange tider, så det vore bra att ha ett datorprogram som översätter till ett mer normalt format. Du ska skriva ett sådant program.

Vi förutsätter att vår klocka saknar sekundvisare och endast visar ett helt antal minuter (det vill säga: båda visarna hoppar framåt bara på hel minut). Vinkeln avläses genom att utgå från timvisaren och sedan mäta hur många grader medurs minutvisaren ligger (se figur 6.4). För att undvika decimaler anges vinkeln i tiondels grader (så att 85.5 grader skrivs som 855). Detta tal är alltid ett heltal mellan 0 och 3595 (inklusive) och är, som en följd av att endast hela minuter visas, alltid delbart med 5.

Programmet ska fråga efter en vinkel och sedan skriva ut tiden i vanligt digitalformat, alltså h:mm eller hh:mm, beroende på antalet timmar. Vi förutsätter att det är morgon, så alla tider ska ligga mellan 0:00 och 11:59 (inklusive).

Två körningsexempel

```
Vinkel ? 855
Klockan är 1:21

Vinkel ? 3140
Klockan är 3:08
```

II

Babbage

1 Introduktion till Python 29

2 Spelprogrammering I 33

3 Spelprogrammering II 37

4 Problemlösning med Python 39

5 Inbyggda funktioner i Python 42

6 Hashning 46

1. Introduktion till Python

16:e juli

Uppvärmningsuppgifter

1. Skriv ett program som skriver ut alla jämna tal i intervallet $[10, 80]$.

2. Skriv ett program som beräknar vilken tuple (2 tal i varje) i en lista som har störst produkt. T.ex. tuplen $(4, 8)$ har en produkt som är 32.

```
pairs = [(1, 3), (5, 7), (-5, -1), (-9, -10), (-5, 4), (5, 100)]
...
```

3. Skriv ett program som först läser in 5 heltal som användaren skriver in, sen beräknar vilket tal som är näst störst och skriver ut det.

   Tips! Använd:

```
input().split()
```

Sets och dictionaries

4. Skriv ett program som med hjälp av ett set skriver ut alla unika element det finns i en lista.

```
x = [1, 2, 3, 1, 2, 1, 1, "Hello", "Hello again", "Hello"]
...
print(number_of_unique_elements)
```

5. Skriv ett program som med hjälp av en dictionary beräknar hur många gånger varje element i en lista förekommer. Skriv ut varje element i listan samt hur många gånger det förekommer.

```
x = [1, 2, 3, 1, 2, 1, 1, "Hello", "Hello again", "Hello"]
...
```

6. Skriv ett program lagrar namnen samt åldrar på personer i en dictionary, och sen skriver ut åldern på varje person som användaren skriver in.

```
names = ["Alice", "Bob", "Charlie", "David", "Erik", "Fredrik"]
ages = [14, 20, 25, 30, 84, 12]
... # Skapa en dictionary här

for i in range(5): # Exempel på att köra programmet för flera namn.
    name = input()
    ... # Skriv ut åldern på personen som man fick från input
```

Rekursiva funktioner

7. Skriv en rekursiv funktion som beräknar fakulteter. Vi utgår ifrån definition:

$$0! = 1$$

och för alla heltal $x > 0$:

$$x! = x \cdot (x-1)!$$

8. Skriv en rekurisv funktion som beräknar fibonacci-talen. Lägg även in en variabel som håller koll på hur många rekursiva anrop som gjordes. Skriv ut både fibonaccitalet och antalet rekursiva anrop som gjordes. Finns det nåtт intressant mönster i antalet anrop som behövs?

Inbyggda funktioner

Det finns en del bra funktioner att känna till i python, som gör våra liv mycket lättare ibland. Här kommer ett par exempel. Ifall nånting är nytt så får ni gärna testa koden för att se vad som händer. Det rekommenderas att printa värdena på variabler för att se vad som händer i varje steg ifall man är förvirrad.

Det finns så klart många fler inbyggda funktioner än de vi tar upp här, men det sparar vi till ett annat tillfälle ;)

```
numbers = [5, 9, 10, 3]
contains_2 = 2 in numbers # detta blir False
contains_5 = 5 in numbers # detta blir True

print(contains_2)
print(contains_5)
print(contains_2 + contains_5) # Vad skriver det här ut? Varför?


# Byt från t.ex. list till set på enkelt sätt
x = [3, 4, 5]
y = set(x)
# Detta funkar även för andra klasser!
# Tuple till lista:
x = (3, 10, 20)
y = list(x)
# range till lista:
x = range(10)
y = list(x)
# str till lista
x = "This is a string"
y = list(x)

# Sätt ihop alla strängar i en lista:
words = ["This", "is", "a", "sentence"]
join_string = " " # Testa med lite olika värden här och se vad som händer.
sentence = join_string.join(words)
print(sentence)
```

```
numbers = [10, 30, 5, 2, 3, 100]
max_number = max(numbers) # Hittar största talet i numbers
max_index = numbers.index(max_number) # Vilket index talet ligger på
mean_number = sum(numbers) / len(numbers)
# Fråga: kan man använda dessa funktioner på saker som inte är heltal? Ifall
    ni fyller numbers med till exempel stränger, vad händer då?
```

9. Som ni såg ovan så finns det många klasser som man kan konvertera direkt mellan. Använd dessa för att skriva ut alla bokstäver som förekommer minst en gång i en sträng.

```
word = input()
...
unique_letters = ...
print(unique_letters)
```

10. Testa följande kod. Som ni lär inse så har Tobias skrivit en bugg i den här koden som gör att programmet kan krascha. Hitta buggen och fixa den så att programmet fungerar, helst utan att ändra på massor i koden.
Programmet ska först läsa in rad text från input som vi antar består enbart av siffror och mellanslag, slå ihop alla tal och sen beräkna ifall talet är delbart med 7.
Ifall ni kör fast så är återigen ett tips att printa variabler för att se att ni har koll på exakt vad som händer.

```
x = input()
s = set()
for c in x:
    s.add(c)
if " " in s:
    s.remove(" ")
x = int("".join(s))

if x%7 == 0:
    print("Talet är delbart med 7!")
else:
    print("Talet är inte delbart med 7!")
```

Flyttal är onda (eller?)

11. Testa att köra följande kod:

```
x = 0.1
print(x)
y = 0.3
print(x * 3 - y)
```

Vad fick ni för resultat? Varför hände det?

12. Vad blir det här?

```
0.1 + 0.2
```

Vad skriver Python ut? Varför då?

13. Ett exempel på när flyttal kan ställa till med problem: följande program är tänkt att skriva ut alla multiplar av 0.3 som är mindre än eller lika med 12. Men om ni testkör det så märker ni att nånting blir fel. Hur kan ni fixa programmet så att rätt saker skrivs ut.

```
x = 0
while x <= 12:
    print(x)
    x += 0.3
```

14. Som vi har sett så kan det bli lite fel ibland om man har decimaltal. Tänk er nu följande problem: ni vet redan att ett tal är delbart med 7 och vill skriva ut vad talet blir ifall ni dividerar det med 7. Ni tänker kanske då att ni borde undvika problem eftersom allting är heltal. Men vad är skillnaden mellan de två följande printsatserna.

```
x = int(input())
print(x // 7)
print(round(x / 7)) # Använder round för att slippa .0 i slutet.
```

Som ni kanske förstår efter det här så kan flyttal ge lite oväntade resultat ibland. Därför är det viktigt att man är försiktig i vissa sammanhang, och det kan ibland löna sig att hålla sig till heltal.

2. Spelprogrammering I

17:e juli

> OBS! För denna lektion behöver du installera `pygame`. Testa att köra `pip install pygame` i en terminal, eller fråga en ledare om hjälp.

`Pygame` kan man använda för allt från att göra allt från spel till simuleringar. Idag kommer vi göra en Flappy-Bird klon. Börja med att skriva av följande kod för att starta ett fönster. `clock.tick(FPS)` försäkrar oss att koden kör 60 gånger i sekunden:

```
import pygame as pg

WIDTH = 1280
HEIGHT = 720

FPS = 60

pg.init()
screen = pg.display.set_mode([WIDTH, HEIGHT])
clock = pg.time.Clock()

running = True
while running:
    # ---- EVENTS ----
    for event in pg.event.get():
        if event.type == pg.QUIT:
            running = False

    # ---- RENDER ----
    screen.fill((0, 0, 0)) # Fyll skärmen med svart

    pg.display.update() # Visa för användaren

    clock.tick(FPS)

pg.quit()
```

1. Testa köra och se vad som händer!
2. Testa vad som händer om du ändrar på siffrorna i `screen.fill` till:

```
screen.fill((255, 0, 0))
screen.fill((0, 255, 0))
screen.fill((0, 0, 255))
```

3. Försök göra skärmen
   a) Gul

b) Lila

c) Orange

Rita saker på skärmen

Det är nu dags att rita saker på skärmen! Pygame har en rad funktioner för att rita olika former, kvadrater, cirklar, linjer, mm.

```
pg.draw.rect(screen, (red, green, blue), (x, y, width, height))
pg.draw.circle(screen, (red, green, blue), (x, y), radius)
pg.draw.line(screen, (red, green, blue), (x1, y1), (x2, y2), width)
```

(Fler funktioner hittar ni här: `https://www.pygame.org/docs/ref/draw.html`).

4. Skapa variabler som heter `spelare_x` och `spelare_y`. Rita en cirkel på dessa koordinater och placera cirkeln mitt på skärmen. Detta kommer bli vår spelare. Cirkeln måste ritas efter att särmen fylls med svart men innan den uppdateras.

5. Rita en rektangel i övre högra hörnet. Denna ruta kommer vi använda för att visa poängen.

Styra med tangentbordet

Skapa en variabel som heter som styr spelarens position (i y-led). Denna kan sedan uppdateras när spelaren trycker ner en tangent med följande kod:

```
for event in pg.event.get():
    ...
    if event.type == pg.KEYDOWN:
        if event.key == pg.K_SPACE:
            print("Mellanslag trycktes ner")
            spelare_y += 50
    if event.type == pg.KEYUP:
        if event.key == pg.K_SPACE:
            print("Mellanslag släpptes")
            spelare_y -= 25
```

Hastighet och gravitation

6. Vi har redan en cirkel som flyttar sig när man trycker ner mellanslag. Skapa en nya variabel som heter `hastighet_y` och förflytta spelaren varje frame genom att köra följande kodstycke varje frame:

```
spelare_y += hastighet_y
```

Låt nu också hastigheten förändras genom att addera gravitation till den varje frame. Definiera först en gravitationsvariabel och uppdatera hastigheten med koden

```
hastighet_y += gravitation
```

7. Skapa ett rör (förslagsvis en grön rektangel) som börjar på höger sida av skärmen. Skapa två variabler som heter `pipe_x` och `pipe_y` som betecknar rörets övre vänstra hörn. Låt `pipe_y` vara slumpad varje gång spelet startar. Detta kan du göra med följande kod:

```
import random

pipe_y = random.randint(0, HEIGHT)
```

8. Förändra `pipe_x` för att kontinuerligt förflytta röret åt vänster med en konstant hastighet. Flytta tillbaka röret till höger sida igen är den har nått väster sida av skärmen.

9. Testa ifall spelaren kolliderar med rören. Vi börjar med att skapa en funktion som kan avgöra om två rektanglar överlappar eller inte. Den tar in åtta parametrar: x, y, bredd och höjd för respektive rektangel och returnerar `True/False` beroende på om rektanglarna kolliderar.

```
def are_colliding(x1, y1, w1, h1, x2, y2, w2, h2):
    if x1 + w1 < x2 or x1 > x2 + w2:
        return False
    if y1 + h1 < y2 or y1 > y2 + h2:
        return False
    return True
```

Använd denna funktion för att avgöra om spelaren och rören krockar med varandra. Om de krockar kan du sätta `running = False` för att avsluta spelet.

10. Lägg slutligen till rör som kommer från taket och följer rören på golvet så att spelaren måste träffa mellanrummet för att överleva. Implementera kollision för dessa rör också.

Tips/Info

Man kan definiera vilka färger man vill använda högst upp i dokumentet. Dessa är sedan enkla att använda i resten!

```
BLACK = (0, 0, 0)
RED = (255, 0, 0)
GREEN = (0, 255, 0)
BLUE = (0, 0, 255)
PURPLE (255, 0, 255)
WHITE = (255, 255, 255)
```

Ibland är man inte intresserad av om en tangent just trycktes ner, utan endast om den är nedtryckt eller inte. Det kan göras med `get_pressed()` som returnerar en lista med booleans (True/False), en för varje tangent. Denna lista kan indexeras med `K_` för att få om en viss tangent är nere eller inte.

```
if pygame.key.get_pressed()[pg.K_SPACE]:
    print("Mellanslag är nedtryckt")
```

Extrautmaning

Om du hinner färdigt får du utveckla spelet fritt, här är några förslag:

- Lägg till fler rör på skärmen samtidigt.
- Höj svårhetsgraden genom att låta rören gunga upp och ner
- Lägg till mynt spelaren kan plocka upp för extrapoäng
- Lägg till powerups, t.ex. en bubbla som gör spelaren odödlig, eller tillåter spelaren att skjuta på rören.

3. Spelprogrammering II

18:e juli

> Detta är en fortsättning på Spelprogrammering 1. Gör först klart de uppgifter från förra lektionen som du inte hann bli färdig med. Fortsätt sedan med dessa om du hinner.

Poäng och text

1. Skapa en poäng-variabel. För varje rör som spelaren lyckas passera utan att krocka in i ska du öka poängen med 1. Antingen när väggen försvinner till vänster på skärmen eller när spelaren precis paserat genom.

2. Spelaren vill såklart se poängen! För att rita text i pygame använder man följande kod:

```
pg.font.init()
font_size = 36
font = pg.font.Font(None, font_size)

while running:
    ....
    text_surface = font.render(str(poäng), True, (255, 255, 255))

    screen.blit(text_surface, (X, Y)) # Kopiera texten från text_surface till
        skärmen på positionen X, Y. Välj X, Y så att den hamnar ovanför vår
        score-rektangel.
```

Notera att denna kod renderar texten varje frame vilket är onödigt om texten inte har ändrats, då räcker det att spara `text_surface`.

Bilder

Fram tills nu har vi bara ritat former i olika färger, men vill man göra snyggare spel kan man ladda in bilder och använda istället. Bilden laddas först in och sparas i en surface som sedan kan placeras på skärmen.

```
# Ladda in bilden, om det är en png med transparent bakgrund använd
    .convert_alpha()
image = pygame.image.load("image.png").convert()
image = pygame.transform.scale(image, (BREDD, HÖJD)) # Skala om bilden till
    rätt storlek.
```

Sedan kan vi placera bilden på skärmen med `screen.blit(image, (X, Y))`.

3. Hitta en bild på en fågel (eller något annan valfritt) och byt ut cirkeln som nu är spelare.

Meny

4. Nu avslutas spelet när man dör, vilket är tråkigt. Istället för att sätta `running = False` när spelaren dör kan du skapa en ny variabel som avgör om spelaren är död eller inte. Om spelaren är död ska spelaren sluta röra på sig. Rita en menu/ruta ovanpå spelet där du skriver `"Game Over"`, hur mycket poäng spelaren fick och en `"restart"`-knapp. Vi kommer lägga till funktionalitet till denna knapp senare.

5. Introducera en high-score variabel som sparar högsta intjänade poängen hittills. Lättast är att låta denna återställan varje gång man startar om spelet.

   Om du vill spara poängen när man startar om spelet kan du skriva highscoren till en fil när spelet stängs.

```
with open("highscore.txt", "w+") as f:
    f.write(str(highscore))
```

   Och sedan läsa in filen i början av spelet.

```
from os import path

if path.exists("highscore.txt"):
    with open("highscore.txt", "r+") as f:
        highscore = int(f.read())
else:
    highscore = 0
```

6. Sist vill vi göra så att `"restart"`-knappen fungerar. Klick med musen reagerar vi på genom ett event.

```
for event in pg.event.get():
    ...
    if event.type == pg.MOUSEBUTTONDOWN:
        if event.button == 1: # 1 - Vänsterklick, 3 - Högerklick
            click_x, click_y = event.pos
            print(f"Left mouse button was pressed at x={click_x}, y={click_y}")
```

   Om användaren klickar med musen på `"restart"`-knappen ska spelet starta om. Återställ spelarens position, poängen samt alla rörs position. Testa om menyn känns bättre om du ersätter `pg.MOUSEBUTTONDOWN` med `pg.MOUSEBUTTONUP`.

Extrauppgifter

- Gör spelet vackrare! Lägg t.ex. in en mark eller moln som langsamt rör sig över himeln. Gör kanske rören och `"Game Over"`-menyn snyggare eller skapa en animation när fågeln dör.
- Låt användaren välja mellan flera olika bilder för spelaren.
- Byt håll på gravitationen ibland.
- Gör någon av gårdagens extrauppgifter - eller något helt eget!

4. Problemlösning med Python

20:e juli

Struktur

Alla problem som man löser med programmering kan delas in i följande 3 steg:

- Läsa in data.
- Bearbeta data.
- Formatera utdata och skriva ut svaret.

I Python läser man in datan med hjälp av funktionen

```
input()
```

Denna funktion läser in exakrt **1 rad** av indata från terminalen, och sparar den raden som en sträng.

1. Skriv ett program som läser in 2 rader av input, och sedan skriver ut andra raden först, och därefter den första raden.

Exempel:

Indata:

```
jag gillar
bananer
```

Utdata:

```
bananer
jag gillar
```

Ibland vill du dela in strängen som du läser in i mindre bitar, till exempel om det finns tre heltal på samma rad. Då vill man i de flesta fallen separera på dessa tre heltal.
Man kan använda sig av funktionen `split()`, som delar upp en sträng där det finns mellanslag, och lägger in alla element i en lista.

```
s = input().split()
```

2. Skriv en kod som läser in tre heltal på samma rad. Skriv ut summan av dessa tre heltal.

3. Skriv en kod som läser in ett okänt antal heltal på samma rad. Skriv ut summan av alla dessa tal.

Man kan lägga till en sträng i `split()` för att klargöra vad det är man vill dela upp strängen på. T.ex. ifall indatan består av flera namn separerade med komma-tecken, vill man använda `input().split(",")`.

4. Skriv en kod som läser in ett antal namn som är separerade med dollar-tecknet, och skriv ut varje namn på en ny rad.
Exempel:

Indata:

```
Harry$Elias$Tobias$Fredrik$Joshua$Ivar
```

Utdata:

```
Harry
Elias
Tobias
Fredrik
Joshua
Ivar
```

Ibland vill man skriva ut svaret på ett specifikt sätt, och då finns många användbara funktioner i Python som kan hjälpa till med det:

```
#join() kan sammansätta en lista av strängar och separerar de med en given
    sträng.
s = ",".join(["A","B","C"])
print(s)


# print() kan skriva ut flera element på samma rad, och separera varje element
    med mellanslag.
print(2,3,"ett ord")


# Man kan specificera i print-funktionen vad som ska printas direkt efter.
    Default är att det blir en ny rad.
print("ost", end = "")
print("bågar")
```

Fler uppgifter

Nu kommer några uppgifter på hemsidan Kattis! Skapa ett konto och lös uppgifterna. (Kattis är väldigt noggrann kring hur indatan och utdatan ska se ut!)

5. https://open.kattis.com/problems/tolvunarfraedingartelja
6. https://open.kattis.com/problems/hipphipphurra
7. https://open.kattis.com/problems/ostgotska
8. https://open.kattis.com/problems/planetaris
9. https://open.kattis.com/problems/drinkingsong
10. https://open.kattis.com/problems/harshadnumbers

Ibland vet man inte hur mycket indata det finns, och det kan vara väldigt jobbigt. Men man kan använda sig av en while-loop och en try-catch som samlar in sig indata tills något går fel (till exempel att det är slut på indata).

```
while True:
    try: # Testa att göra följande:
        n = int(input())
    except: # Oj nånting i det vi testade gick fel, gör det här istället:
        break

    ...
```

11. https://open.kattis.com/problems/closestsums
12. https://open.kattis.com/problems/savingdaylight

Blandade uppgifter

13. https://open.kattis.com/problems/champernowneverification
14. https://open.kattis.com/problems/npuzzle
15. https://open.kattis.com/problems/irepeatmyself
16. https://open.kattis.com/problems/delimitersoup
17. https://open.kattis.com/problems/guess
18. https://open.kattis.com/problems/meddelande
19. https://open.kattis.com/problems/siffersumma
20. https://open.kattis.com/problems/klippaprotein

5. Inbyggda funktioner i Python

21:e juli

Att läsa in input

```
# Läsa in en rad input:
line = input() # Allt blir till en sträng, inklusive mellandslag osv.

# Läsa in ett heltal som står självt på en rad:
number = int(input())

# Läs in en lista med ord:
words = input().split()

# Läs in en lista med heltal:
numbers = list(map(int, input().split()))
```

Att printa saker

```
# För att printa ut en sak (kan vara antingen en sträng, int, flyttal osv):
variable = ...
print(variable)


# För att printa ut en lista med saker (på en rad):
# Antingen:
for elem in list:
    print(elem, end = " ") # Det blir ett mellanrum mellan alla saker i listan.
print() # Gör så att framtida saker skrivs ut på en ny rad.

# Eller:
print(*list)
```

Inbyggda saker i python

Här kommer vi att gå igenom en del funktioner som finns inbyggda i python. Dessa funktioner är väldigt vanliga i programmaring, då det förkortar koden med mycket, men så länge man kan syntaxen, så är det fortfarande läsbart.

Lambdafunktioner

Lambdafunktioner är en typ av pythonfunktion, men som inte skrivs med ordet *def* för att definiera den. Det gör att man kan använda den lokalt, och göra koden lite mer kompakt och läsbar.

```
# Exempel på hur man kan spara en lambdafunktion och sedan använda den
```

```
foo = lambda x: x ** 2
five_squared = foo(5)

# Samma med en vanlig funktion:
def foo(x):
    return x ** 2
five_squared = foo(5)

# Lambda funktioner kan även ta in flera argument:
division = lambda numerator, denominator: numerator / denominator
quotient = division(3,2)
```

Det är ofta användbart att använda lambdafunktioner tillsammans med de funktioner som vi kommer gå igenom nu. Försök att använda lambdafunktioner om ni kan. Men ifall det känns svårt så kan ni hålla er till vanliga funktioner.

map

Nu ska vi använda funktionen map. map tar in 2 parametrar: först en funktion och sen en iterable. Dvs någonting som går att iterera genom. Det kan vara t.ex. en lista, tuple, dictionary och liknande. Ett exempel såg ni tidigare:

```
numbers = list(map(int, input().split()))
```

Det som skickas in här är funktionen *int*, och det kommer att appliceras på varje sträng som kommer som del av *input().split()*. Som returvärde får vi ett map-objekt. Därför måste vi göra om det till den datastruktur som vi vill använda istället. I det här fallet så gör vi därför om det till en lista.

1. Skriv kod som använder map för att skapa en lista som innehåller kvadraterna av alla talen i en annan lista.
2. Skriv kod som använder map som applicerar följande på varje tal i listan: ifall talet är jämnt, kvadrera det. Annars, ersätt det med None.

List comprehensions

List comprehensions är väldigt användbara i python. Det är på vissa sätt väldigt likt att använda lambda, men syntaxen är lite annorlunda. Ibland kan det vara lättare att använda lambda, och ibland tvärt om. Därför är det bra att känna till båda metoderna.

```
# Exemepel på en list comprehension som kvadrerar alla tal i en lista:
x = [1, 2, 3, 4]
y = [elem ** 2 for elem in x] # Blir: [1, 4, 9, 16]

# Exempel på en list comprehension som bara behåller jämna tal:
x = [1, 2, 3, 4]
y = [elem for elem in x if elem % 2 == 0] # Blir [2, 4]

# Exempel där man gör båda saker samtidigt.
x = [1, 2, 3, 4]
y = [elem ** 2 for elem in x if x % 2 == 0] # Blir [2, 16]
```

3. Skriv en list comprehension som lägger till bokstaven s i slutet av varje sträng i en lista.

4. Skriv en list comprehension som skapar en lista av tuples av längd 26, där tuple på index i i den resulterande listan är lika med $(i, bokstav_i)$, där $bokstav_i$ är den bokstav som kommer på plats i i alfabetet.

5. Översätt föregående lösning till en dictionary comprehension, som mappar varje bokstav i alfabetet till vilken plats den har i alfabetet.

```
# Här är ett annat exempel på en dictionary comprehension:
x = {i : i ** 2 for i in range(10)} # Skapar en dictionary som innehåller de
    10 första kvadrattalen.
```

Kodgolf

Kodgolf går ut på att skriva så kort kod som möjligt, (för det mesta räknat som antalet tecken koden består utav). Nu ska vi försöka att göra det för lite exempel. Försök gärna att använda funktioner så som map, list comprehensions osv då det ofta hjälper för att göra koden kortare.

6. Skriv ett program som läser in en lista av heltal från input, och sedan kollar hur många av talen som är lika med deras index i listan. Printa svaret så att det syns när man kör programmet.

7. Skriv ett program som för varje rad av input skriver ut den lägsta bokstaven som finns i raden och även den högsta bokstaven som finns.

```
# Ifall man till exempel skriver detta som en rad:
>>> Hejsan det här är lite text.
# Så ska erat program skriva ut:
>>> a ä
```

8. Skriv ett program som för varje rad av input skriver ut andelen ord som innehåller bokstaven x.

```
# Exempel:
>>> Det här är ett par ord. Vissa innehåller x så som xylofon men vissa gör
    inte det.
# Bör skriva ut (2 / 17):
>>> 0.11764705882352941
```

Kattisuppgifter

9. https://open.kattis.com/problems/squarepeg
10. https://open.kattis.com/problems/lastfactorialdigit
11. https://open.kattis.com/problems/flexible

12. Ni ska nu försöka förkorta eran kod i tidigare uppgifter så mycket som möjligt. Ifall ni kör fast eller har frågor så är det alltid ok att fråga ledarna om tips. Det finns nämligen ofta många knep som man kan använda för att förkorta sin kod.

Clash Of Code

13. — **EXTRA.** Grattis! Du har tagit dig igenom alla upggifter på dagens lektion.

Ifall du har tid över får du skapa ett konto på https://www.codingame.com/multiplayer/clashofcode. Harry kommer att skapa ett rum där alla kan köra kodgolf mot varandra.

Ett bra sätt att faktiskt bli bättre på kodgolf är att man kodgolfar tillsammans med andra personer! Oftast lär man sig många nya trick som man kan använda.

6. Hashning

22:a juli

Under dagens lektion kommer ni att få lära er om hashning. Vi kommer att börja med vad det är för nått, hur man kan hasha någonting själv, och sen undersöka ett par vanliga områden där det används.

Vad är hashning? Varför använder man det?

En hashfunktion är en funktion som tar in ett objekt som input och ger ett heltal som output. Viktigt är att funktionen ger samma output varje gång den anropas med samma input. Dvs ifall man till exempel beräknar hashvärdet av strängen `"hej"` en gång och får 45 som output, så måste man få 45 nästa gång man beräknar hashvärdet av samma sträng.

Ett vanligt exempel på hashning är att spara lösenord. Istället för att spara vad ditt lösenord är i en stor databas så kan man spara hashen. På så sätt kan man jämföra ifall man skrev in rätt lösenord genom att se ifall det hashar till det sparade värdet. Hashar det till ett annat värde, så kan man vara säker på att man skrev in fel lösenord. Och hashar det till samma värde så kan man vara ganska säker på lösenordet är rätt. Eftersom lösenordet inte faktiskt sparas nånstans, så blir det myckte svårare för hackers att stjäla dina lösenord. De kanske hackar sig in i ett företags databas och ser en hashen av ditt lösenord, men hur ska de veta vad ditt lösenord var.

Notera att även om man skriver in fel lösenord så kan det råka hashas till samma värde. Så ifall man får samma hashvärde så kan man alltså inte vara säker på att lösenordet är rätt. Men om hashfunktionen har bra spridning på outputen så blir det väldigt osannolikt att man råkar få samma hash för olika lösenord.

Från exemplet så förstår ni alltså kanske att det finns ett par viktiga egenskaper med hashfunktioner:

- Den bör vara deterministisk - annars blir ju hashvärdet bara som ett slumpat tal, och det vill vi inte ha.
- Fördelningen av hashfunktionens outputvärde bör vara så nära uniformt som möjligt. På så sätt ökar man säkerheten av hashen.
- Det bör vara svårt att hitta en invers till hashfunktionen. T.ex. ska det vara nästan omöjligt att lista ut vad lösenordet är enbart utifrån hashvärdet, även ifall man vet exakt hur hashfunktionen fungerar.

1. Skriv en funktion som tar in en sträng s som argument, och beräknar en hash av strängen. Hashen bör beräknas enligt följande formel, eller liknande:

$$\sum \text{ascii}(s_i) \times m^i \text{mod} p$$

m kan ni välja ett värde på, men det bör vara större än det största värdet som ett ascii tecken kan ha. Varför tror ni? p bör vara ett primtal då detta fungerar

väldigt bra jämfört med ickeprimtal. Ju större primtal ni väljer desto bättre borde hashfunktionen bli. Men allt över 10^9 brukar räcka för bra för våra syften. Testa vad ni får för hashvärde på ett par olika strängar. De bör bli olika (om ni inte har väldigt otur).

2. Skriv ett program som går igenom en lista med strängar och jämför hur många par av strängarna som har samma hash. Eran lösning bör enbart beräkna hashfunktionen en gång per sträng.

3. Bygg vidare på förra lösningen, men beräkna nu hur många par av strängar som är identiska. Dvs ifall de har samma hash, så kontrollera ifall de är lika med varandra också.

Hashning för dictionaries

Under tidigare lektioner använde vi dictionaries lite grann, men vi har inte gått igenom varför de fungerar. Hur kan man så snabbt slå upp någonting i en dictionary? Jo, en dictionary i python är implementerad som en hashmap - en datastruktur som använder sig utav en hashfunktion för att vara snabb. När man exempelvis vill stoppa in en sträng i en hashmap så görs följande. Först hashas strängen till ett värde h. Sen stoppar man in strängen i en stor lista på index h. Nästa gång man vill leta upp strängen kan man allstå beräkna hash funktionen och se ifall det finns någonting på index h i den interna listan. Allt det här görs automatiskt av pythons dict klass ifall man stoppar in någonting som strängar, heltal osv. Men vad händer ifall man vill stoppa in en egen klass?

För följande uppgifter, använd er utav den här koden, och modifiera den för att utföra det som står i uppgifterna.

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

4. Skriv ett program som skapare ett par *Person* object och använd de som nycklar i en dictionary. Stoppa in samma objekt, flera gånger (uppdatera värdet), och även ett annat *Person* objekt med samma `name` och `age`. Uppdateras det värde du förväntade dig? Undersök hur stor dictionaryn blir med *len*. Beroende på hur ni gör det här så kanske ni märker att längden blir större än förväntat. Hur kommer det sig? Testa att använda funktionen *hash*. Vad är det som händer?

5. Istället för att använda objekten som de är i en dictionary så kan vi beräkna hashen av objektet och använda det istället. Skriv ett program som stoppar in hasharna av personer som nycklar i en dictionary. Beräkna dessa hashar med funktionen ni skrev tidigare.

6. Förra problemet är egentligen lite av en fusklösning. Man kan göra så, men det är inte rekommenderat. Istället kan man implementera funktionen `__hash__`, som då kommer att användas automatiskt när klassen ska hashas. Lägg till `__hash__` och återgå till tidigare uppgifter. Fungerar det nu?

Prefix hashning/intervalhashning

Vi ska nu undersöka hur man kan hasha strängar på mer avancerade sätt. Det finns en mängd med strängproblem som kan lösas med hashning genom att hasha en del av strängen.

Exempel: Säg att vi har en sträng så som `"Hallå eller"` och vill beräkna hashen av slutet av strängen. Dvs enbart `"eller"` delen. Det naiva sättet att göra detta är att skapa en ny sträng som bara innehåller det vi är intresserade av, och hasha den strängen. Ifall vi bara kommer att göra detta en gång så är det en ok lösning. Men säg att vi har en gigantisk sträng och vill beräkna hashen av massor av delar av strängen. Då skulle en sån lösning kunna bli väldigt långsam.

Vad kan vi göra istället? Medans vi beräknar hashen av hela strängen kan vi spara hashen utav alla prefix. Eftersom vi implicit räknar ut det medans vi räknar ut hashen för hela strängen så är det lika bra att spara undan. Vi kan sedan ta hashen av ett prefix minus hashen av ett kortare prefix för att få hashen av ett intervall. Dock ifall vi bara gör så här så finns det ett litet problem. Kan ni lista ut vad?

7. Skriv ett program där ni sparar hashen för varje prefix av en sträng. Detta bör göras genom att bara loopa över strängen en gång. Spara undan hashen av varje prefix i en lista.

Säg nu att vi vill jämföra hashen av ett intervall med en sträng. Då kan vi göra följande:

```
# To compare the hash of a string ‘s‘, and the hash of an interval [a, b) of
    another string.
prefix_hashes = ...
hash_of_interval = prefix_hashes[b] - prefix_hashes[a]
hash_of_string = hash_fn(s)
if (hash_of_string * (m ** a)) % p == hash_of_interval:
    print("They match")
```

8. Studera koden ovan och försök att förstå den. Varför måste vi ta med `m ** a`?

9. Beräkna hashen av ett intervall genom att göra det som beskrevs ovan. Skriv nu ett program som avgör ifall en sträng innehåller en annan sträng med hjälp av hashning. Detta kan ju så klart göras med pythons `in` funktion, men det är inte tillåtet för denna uppgift. Det är ju kul att implementera ett hyffsat snabbt sätt att göra det själv.

Palindromuppgifter

Ett exempel på nått man kan göra genom att använda prefixhashning är att avgöra ifall ett interval av en sträng är ett palindrom eller inte. Detta kan vi göra på följande sätt:

- Beräkna prefixhashar för en sträng.
- Beräkna prefixhashar av samma sträng, fast baklänges.
- Jämför hasharna av intervallet genom att använda både prefixhasharna. Var försiktig då du kommer att behöva multiplicera strängarna med någon potens av `m`, så

som vi förklarade i uppgiften under dagens lektion.

10. Skriv en funktion som avgör hur många delsträngar av en sträng som är palindrom. Använd er av beskrivningen ovan för att försöka åstadkomma detta.

III

Church

1 PyTorch 51

2 Pink, fluffy unicorns dancing on rainbows! 56

3 C++ (Fredriks grupp) 65

4 C++ (Tobias grupp) 79

5 Git 96

6 Bashscripting 101

7 Fina bilder med shadertoy 111

8 Algoritmer I. Segmentträd 118

9 Parallellprogrammering i C++ 123

10 Algoritmer II. Nåbarhet och MST 129

11 Webscraping med Python 137

1. PyTorch

17:e juli

Under dagens lektion kommer vi att gå igenom hur man kan träna ett neuralt nätverk i Python med hjälp av Pytorch. Vi kommer att träna ett neuralt nätverk att känna igen handskrivna siffror från det välkända MNIST datasetet. För att åstadkomma detta behövs inte mycket kod, särskilt ifall man använder sig utav många inbyggda pytorch-funktioner. Vi kommer dock inte att göra detta. Då det är kul att lära sig saker, kommer ni att implementera vissa av dessa saker själva, så att ni får en bättre förståelse över hur det egentligen fungerar.

Intsallera Pytorch

Pytorch finns som ett Pythonpaket och går lätt att installera på sin dator. Om ni har en dator som kör Linux eller MacOS så räcker det med att skriva `pip3 install torch` i terminalen (förutsatt att man har pip installerat först, vilket brukar finnas). Ifall ni har en windows dator, kan det vara lite svårare att komma igång, och vi rekommenderar därför att ni kör på en av skoldatorerna.

Ett lätt sätt att verifiera att PyTorch har installerats korrekt är att köra ett pythonprogram med raden `import torch`. Ifall det fungerar så är ni redo att börja koda.

Vi kommer även att använda oss utav ett par andra bibliotek under lektionen. För dagens lektion behöver ni även ladda ner torchvision, vilket ni gör genom att köra `pip3 install torchvision`.

Tensorer

Innan vi börjar med att träna ett neuralt nätverk så är det bra att ha lite kunskap om tensorer, som används lite överallt av pytorch. En tensor kan man se som en n-dimensionell matris. Har man en 1-dimensionell tensor så är det i princip en vektor. Är det en 2-dimensionell tensor så är det en matris, men det kan även ha fler dimensioner än så.

Här är ett par exempel på att skapa tensorer. Testa att printa ett par av de här alternativen för att se vad de gör.

```
import torch

a = torch.tensor([[1,2],[3,4]])
b = torch.zeros(5)
c = torch.zeros((5, 6))
d = torch.arange(10)
e = torch.ones(8)
```

Vill man använda en tensor, så kan använda den ungefär som om det vore en lista. Dvs ifall man har en 1-dimensionell tensor funkar:

```
import torch
a = torch.arange(5)
print(a[2])
```

Har man en flerdimensionell tensor kan man skriva:

```
import torch
# Skapa en tensor med 6 rader med siffrorna 0-3
a = torch.tensor([list(range(4) for _ in range(6)])
single_row = a[2]
single_entry = a[2, 3]
two_rows = a[1:3]
```

Notera att man alltså kan indexera med ett komma mellan varje dimension, och behandla varje dimension som om det vore en lista.

Varje tensor som man skapar har en viss storlek, som man kan se genom att skrive `variable_name.shape`, och det går även att manipulera storleken lätt, exempelvis så här:

```
import torch
a = torch.arange(24)
print(a.shape, a)
b = a.reshape((4, 6))
print(b.shape, b)
c = b.unsqueeze(0)
print(c.shape, c)
d = c.squeeze()
print(d.shape, d)
```

1. Skapa en två dimensionell tensor som har 5 rader och 6 kolumner. Tensorn bör innehålla talen 0-5 på första raden, 6-11 på andra raden osv.

   Tips: `reshape` är en användbar metod.

2. Skriv ut de mellersta två kolumnerna av matrisen du skapade i förra uppgiften. Dvs kolumnerna med index 2 och 3.

   Tips: indexering/slicing

Det finns även massor med andra användbara funktioner för att manipulera tensorer. Ett tips när man håller på med deep learning är att se till att storleken på sina tensorer är korrekt. En vanlig bugg är nämligen att man skapar tensorer som till exempel har omvänd storlek (t.ex. (5, 6) istället för (6, 5)) eller att den har fel antal dimensioner (t.ex. (1, 5, 6) istället för (5, 6)). Därför tycker jag att det ofta är värt att printa storleken av tensorer så att man kan se till så att de är rätt.

Förberedning av träningsdatan

För att träna ett neuralt nätverk måste vi ladda ner datasetet och läsa in det i programmet. Detta kan vi lätt åstadkomma med i princip en rad kod, då det redan finns i torchvision biblioteket:

```
import torchvision
image_directory = 'images/'
dataset = torchvision.datasets.MNIST(image_directory,
                                     download=True,
                                     transform=torchvision.transforms.ToTensor())
```

Första gången ni kör det här kommer datasetet att sparas ner till mappen som ni gav som `image_directory`. Nerladdningen kommer dock bara att ske första gången. Följande gånger ni kör programmet kommer bilderna att läsas in från datorn istället.

Dataset innehåller en hel del information om datasetet. Det viktigaste, som vi kommer att använda oss utav är `dataset.data` - en tensor som innehåller all bilddata, och `dataset.targets` - en tensor med vad varje bild är för typ av siffra.

3. Skapa två partitioner av datasetet: en partition som vi kommer att använda för att träna nätverket, och en partition som vi kommer att använda för att säkerställa att nätverket inte har överfittat till träningsdatan. Träningspartitionen brukar bestå utav en majoritet av datan, men exakt hur mycket är upp till er.

   Tips: Det kan vara lätt att loopa över alla exempel i datasetet och lägga de i en temporär lista. Sen på slutet kan man skapa konvertera listan till en tensor.

4. Skapa en klass som representerar ditt neurala nätverk. Här ser ni ett exempel på hur klassen kan se ut:

```
class Model(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.forward_layer = ...

    def forward(self, x):
        return self.forward_layer(x)
```

Tills allting är på plats räcker det med att modellen har ett enda lager. Typiskt sett brukar en modell ha flera lager (ibland väldigt många), men det kan ni lägga in senare, när allting fungerar om ni vill. Ett exempel på ett lager ni kanske vill använda är `torch.nn.Linear`.

`forward` funktionen går att anropa väldigt lätt genom att skriva anropa modellen som om det vore en funktion. T.ex:

```
model = Model()
input_data = ... # Bilddatan från datasetet.
output = model(input_data)
```

5. Loopa över träningsdatasetet och träna modellen på en bild åt gången. Exempel:

```
training_data = ...
training_targets = ...
model = ...
optimizer = torch.optim.SGD(model.parameters())
for image_data, target in zip(training_data, training_targets):
    optimizer.zero_grad()
    output = model(image_data)
    loss = ...
    loss.backward()
```

Som ni ser så finns det ett par saker som ni måste lägga till. Dels så skapar vi ett `optimizer` objekt som är ansvarig för att uppdatera nätverkets vikter. I början av varje iteration så måste vi se till att anropa `optimizer.zero_grad()` för att derivatorna som beräknades i förra iterationen inte ska räknas med igen i den här iterationen. Sist gör vi en forward pass genom att anropa modellens forward funktion, beräknar lossen, och kör `loss.backward()` som uppdaterar parametrerna i nätverket.

6. Lägg till kod så att ni under träningen kan se hur stor loss modellen har. Ett bra sätt är att printa medelvärdet av de n senaste iterationerna och göra det med jämna mellanrum. Starta en träning och se hur lossen förändras. Förhoppningsvis ser ni att lossen går ner med tiden. Om inte, så kan det vara ett tecken på att det finns nån bugg i koden.

7. Lägg till kod som utvärderar modellen på valideringsdatasetet. För detta kommer vi inte att beräkna en loss som vi gjort tidigare. Istället kommer vi endast köra forward funktionen, och jämföra modellens output med korrekt svar. Hur många procent av gångerna gissar modellen på samma sak som det korrekta svaret efter att ni har tränat modellen?

8. Spara modellen till en fil och läs in den igen. På så sätt kan ni stänga av programmet och fortsätta träna modellen igen, utan att behöva börja om från början. Det här kan spara massor med tid.

   Tips: kolla upp `torch.save`, `torch.load`.

Förbättringar

Nu när vi har skrivit ihop alla de viktigaste delarna av programmet, så är det dags att försöka förbättra modellen. Målet är att den ska få så hög precision som möjligt på valideringssettet. Precisionen påverkas av massor av faktorer, och nu ska vi ta en titt på vissa av de faktorerna.

9. Modifiera koden så att vi loopar igenom datasettet flera gångar. Att vi går igenom hela datasettet kallas för en epoch, och ofta räcker det inte bara med en epoch. Gör så att modellen tränar i ett par epocher. Efter varje epoch, printa precisionen på valideringssettet. Detta kan hjälpa med att avgöra hur många epocher som är rimligt att ha. När precisionen slutar att gå ner så behöver vi inte träna längre.

10. Testa att lägga till ett par till lager i nätverket, särskilt ifall ni bara har ett lager för tillfället. Exakt hur många lager som är bäst och hur stora de bör vara är inte uppenbart. Det kan vara värt att testa ett par olika alternativ för att se vad som ger bäst resultat.

11. Hittills har vi bara tränat på en bild åt gången. En förbättring som vi kan använda oss utav är att träna med en batch åt gången. Dvs att modellen samtidigt tränar på ett antal bilder. Modifiera träningsloopen så att modellen tar in *batch_size* stycken bilder åt gången, där ni väljer ett rimligt värde på *batch_size*. Ofta brukar man använda en tvåpotens, men man måste inte göra det.

Extrauppgifter

12. Rita egna siffror och se ifall nätverket gissar rätt på vad de liknar. Ni kan antingen rita bilderna i t.ex. paint, gimp osv, eller fota på ett papper. För detta så måste ni även skriva ihop kod för att läsa in bilderna och konvertera de till rätt format. Dvs svartvitt, rätt storlek, och som en tensor.

13. Skriv ihop ett program som visualiserar vilka siffror som nätverket gör fel på. Finns det något gemensamt för de bilderna som nätverket gör fel på? Eller är det ganska slumpartat?

14. Experimentera med att applicera transformer på bilderna under träningen. Syftet med detta är att få in mer variation i datasettet, för att försöka få en högre precision på valideringssettet. Det finns en mängd transformer i `torchvision.transforms` som går att använda. Bland annat skulla man kunna testa att rotera bilderna med en liten, slupmad vinkel, lägga till en blurreffekt på vissa bilder, eller att flytta vissa bilder lite i sidled/höjdled.

15. Läs på om olika optimizers och hur de fungerar. Ett kännt exempel är `torch.optim.Adam`.

Final boss

Ifall ni känner er sugna på en utmaning så får ni testa att implementera ännu mer från scratch. Dvs hela forward propagation- och backpropagationberäkningarna, uppdatering av nätverkets parametrar etc.

2. Pink, fluffy unicorns dancing on rainbows!

18:e juli

Nycklar (enbart giltiga under mattekollo):

```
export OPENAI_API_KEY=sk-proj-gJSh5Y7loGR1vg1tMPmbT3BlbkFJzAGqaJv05KvB1qbpbiSq
export ELEVENLABS_KEY=a1c5312ae2f15b26c163d111ef9123ed
export WEATHER_KEY=4016360a2ee049889ee213659241507
```

Några klonade röster:

```
Ulf: pSli88HXKG7P1vmOo65c
Fredriks gammelmormor: cpOePYrTUb1rxarrwSw9
Kungen: VTwSMNYT9lnv1okNWZMz
```

2.1 Några exempel:

2.1.1 Hur man anropar ett väder-API ”rått”

```python
import requests
import os

key = os.environ["WEATHER_KEY"]
response =
    requests.get(f"http://api.weatherapi.com/v1/current.json?key={key}&q=Linköping&aqi=no")
data = response.json()

print(str(data["current"]["temp_c"]) + '\N{DEGREE SIGN}')
```

2.1.2 Hur man anropar ett API med ett Python-bibliotek

```python
import os
from openai import OpenAI

client = OpenAI()

chat_completion = client.chat.completions.create(
    messages=[
        {
            "role": "user",
            "content": """Vad heter Danmarks huvudstad?"""
        }
    ],
    model="gpt-4o",
)
print(chat_completion.choices[0].message.content)
```

2.1.3 Få en beskrivning av en bild (via requests)

```
import base64
import requests
import os

# Function to encode the image
def encode_image(image_path):
  with open(image_path, "rb") as image_file:
    return base64.b64encode(image_file.read()).decode('utf-8')

# Path to your image
image_path = "bild.png"

# Getting the base64 string
base64_image = encode_image(image_path)

api_key = os.environ["OPENAI_API_KEY"]
headers = {
  "Content-Type": "application/json",
  "Authorization": f"Bearer {api_key}"
}

payload = {
"model": "gpt-4o",
"messages": [
  {
  "role": "user",
  "content": [
    {
      "type": "text",
      "text": "Can you describe the attached image in one sentence? What does
          it depict? You don't have to start the sentence with \"The image
          depicts\" or similar."
    },
    {
      "type": "image_url",
      "image_url": {
        "url": f"{base64_image}"
      }
    }
  ]
  }
],
"max_tokens": 300
}
response = requests.post("https://api.openai.com/v1/chat/completions",
    headers=headers, json=payload)
response = response.json()

print(response["choices"][0]["message"]["content"])
```

2.1.4 Läsa in alla kort i spelet Codenames

Figur 2.1: Ord i Codenames

Ändra på exempel 2.1.3 så du laddar in en annan bild och uppdatera text-prompten.

```
image_path = "ord.jpg"
[...]
"text": "Can you read these cards and write a list of all 25 words that
    appears on the cards?"
```

2.1.5 Skapa en bild på en blå gris

```
from openai import OpenAI
import requests
client = OpenAI()

response = client.images.generate(
  model="dall-e-3",
  prompt="A blue pig.",
  size="1024x1024",
  quality="standard",
  n=1,
)

image_url = response.data[0].url
print(image_url)
```

```
response = requests.get(image_url)

if response.status_code == 200:
    with open("bla_gris.png", "wb") as file:
        file.write(response.content)
        print("File downloaded successfully!")
else:
    print("Failed to download the file.")
```

2.1.6 Ändra lite på en bild (undvika copyright problem)

```
from openai import OpenAI
import requests
client = OpenAI()

response = client.images.create_variation(
  model="dall-e-2",
  image=open("katt_och_hund.png", "rb"),
  n=1,
  size="1024x1024"
)

image_url = response.data[0].url
response = requests.get(image_url)

if response.status_code == 200:
    with open("new_image.png", "wb") as file:
        file.write(response.content)
        print("File downloaded successfully!")
else:
    print("Failed to download the file.")
```

2.1.7 Transkribera vad jag säger från en fil

```
from openai import OpenAI
client = OpenAI()

audio_file= open("fredrik_snackar_skit.mp3", "rb")
transcription = client.audio.transcriptions.create(
  model="whisper-1",
  file=audio_file
)
print(transcription.text)
```

2.1.8 Använda openAI:s talsyntes

```
from openai import OpenAI
import pyaudio

client = OpenAI()

player_stream = pyaudio.PyAudio().open(format=pyaudio.paInt16, channels=1,
    rate=24000, output=True)


with client.audio.speech.with_streaming_response.create(
    model="tts-1",
    voice="shimmer",
    speed=0.8,
    response_format="pcm",
    input="Hejsan, hur är läget? Jag skulle vilja prata mer med dig!",
) as response:
    #response.stream_to_file("output.mp3")
    for chunk in response.iter_bytes(chunk_size=1024):
        player_stream.write(chunk)
```

2.1.9 Använda elevenlabs klonade röster

```
# Import necessary libraries
import requests # Used for making HTTP requests
import json # Used for working with JSON data
import os

CHUNK_SIZE = 1024 # Size of chunks to read/write at a time
XI_API_KEY = os.environ["ELEVENLABS_KEY"] # Your API key for authentication
VOICE_ID = "VTwSMNYT9lnv1okNWZMz" # ID of the voice model to use
TEXT_TO_SPEAK = "Hej, ursäkta, jag behöver tränga mig förbi här. Jag är
    faktiskt kung över konungariket Sverige." # Text you want to convert to
    speech
OUTPUT_PATH = "output.mp3" # Path to save the output audio file

# Construct the URL for the Text-to-Speech API request
tts_url = f"https://api.elevenlabs.io/v1/text-to-speech/{VOICE_ID}/stream"

# Set up headers for the API request, including the API key for authentication
headers = {
    "Accept": "application/json",
    "xi-api-key": XI_API_KEY
}
# Set up the data payload for the API request, including the text and voice
    settings
data = {
    "text": TEXT_TO_SPEAK,
    "model_id": "eleven_multilingual_v2",
```

```
    "voice_settings": {
        "stability": 0.5,
        "similarity_boost": 0.8,
        "style": 0.0,
        "use_speaker_boost": True
    }
}
# Make the POST request to the TTS API with headers and data, enabling
    streaming response
response = requests.post(tts_url, headers=headers, json=data, stream=True)

# Check if the request was successful
if response.ok:
    # Open the output file in write-binary mode
    with open(OUTPUT_PATH, "wb") as f:
        # Read the response in chunks and write to the file
        for chunk in response.iter_content(chunk_size=CHUNK_SIZE):
            f.write(chunk)
    # Inform the user of success
    print("Audio stream saved successfully.")
else:
    # Print the error message if the request was not successful
    print(response.text)
```

2.2 Anropa API:er

1. Väder-API tjänsten har också en endpoint som ger kommande matcher. Skriv en kod som hämtar en ort och skriver ut kommande matcher i fotboll i regionen. Använd en loop som skriver ut alla matcher. Be användaren om en ort som skickas med i anropet.

   http://api.weatherapi.com/v1/sports.json?key=XXX&q=Linköping

Figur 2.2: AI-genererad bild av en fotboll

2.3 Generera text

2. Skicka in lite frågor från en frågesport och be chatGPT besvara dem. Kanske kan ta Qube-frågorna om svenska språket? Välj en specifik frågetyp (personligen gillar jag ”Kluriga synonymer”). Ge gärna ett par, fyra, fem exempel av en frågetyp först, så GPT förstår hur frågetypen fungerar. Varje exempel består alltså av både frågan och rätt svar. Försök skriva en bra prompt som klarar av att svara rätt på många frågor.

Figur 2.3: Frågekort från Qube

3. Hämta en väderleksrapport 3 dagar framåt via forecast-endpointen och skicka in det till GPT för att be den formulera en väderleksrapport på vanlig svenska istället för massa data.

Eventuellt får du göra lite pre-processing, som att bara ta med `avgtemp_c`, `daily_chance_of_rain` och `conditions` för dagen, inte för varje timme per dag.

http://api.weatherapi.com/v1/forecast.json?key=XXX&q=Linköping&days=3&aqi=no&alerts=no

4. Använd codenames orden (som du läste in i exempel 2.1.4) och skriv en prompt som grupperar ihop så många ord som möjligt från en lista, samtidigt som den undviker att nämna ett ord som ens påminner om orden i andra listan.

5. Prompta fram en AI som har koll på ett område, säg kvantmekanik, och hjälper dig att lära dig mer om området genom att ställa frågor till dig och lära dig om området på din nivå. Eventuellt kan du ju använda requests biblioteket för att hämta faktaunderlag från wikipedia eller andra källor.

6. Skapa ett äventyrsspel där du går runt och interagerar med agenter som kan prata med dig. Du kan ha olika prompts för olika agenter, för att forma deras personligheter. Spelet ska vara textbaserat så du får olika val i hur du ska röra dig runt, till exempel ”Vill du gå till mataffären eller kyrkan?” och där kan du träffa på olika personer. Använd olika prompts som styr handlingen och vad du kan göra, och andra prompts för personerna du möter. På sikt kanske du kan koppla på grafik i spelet, men det är nog mer ett projekt för fredagens projektdag!

2.4 Analysera bilder

7. Beskriv vad som finns i codenames bilderna. Gör en loop som beskriver varje bild och skriver ut det i terminalen.

Figur 2.4: Codenames bilder

8. Använd beskrivningarna av bilderna i uppgift 7 och spara texterna i en array. Sen ska programmet fråga efter ett ord och en siffra N. Programmet (mha en openAI-prompt) ska gruppera ihop N antal beskrivningar som matchar ordet. Programmet ska skriva ut vilka bilder som den anser att passar till ordet (upp till N beskrivningar).

2.4.1 Generera bilder

9. Skapa snygga HD bilder utifrån beskrivningarna vi skapade i uppgift 7.

10. Använd edits-endpointen `https://api.openai.com/v1/images/edits` och skicka med en mask för att lägga till en hatt på en katt.

Figur 2.5: Katt med hatt

2.5 Transkribera tal (hörsel)

11. Vem orkar titta på en 15 minter lång video i dagens samhälle??? Ingen!! Ladda ned en youtube video som mp3 (använd t.ex. `https://ytmp3s.nu/`) och skicka in det för att transkribera hela videon. Skicka sedan transkriptionen till chatGPT för att sammanfatta det och ta ut höjdpunkterna ur videon. Hur lång tid tog det för AI:n att lyssna igenom 15 minuter video? AI är överlägsen oss på att ta till sig information. Om du inte vet vad för video kan du väl ta nån om Redstone i minecraft? Typ `https://www.youtube.com/watch?v=ooL9nVQA6qU`

12. Använd mikrofonen och transkribera vad jag säger live.

13. Översätt transkriptionen av det jag säger i realtid till spanska, ryska eller nått annat språk.

2.6 Generera röster (tal)

14. Låt en talsyntes (t.ex. OpenAI) läsa upp sammanfattningen av youtube-videon. Testa lite olika röster, t.ex. alloy, echo, fable, onyx, nova, eller shimmer.

15. Låt användaren välja ett språk (tyska, franska, ryska etc.) och översätt sammanfattningen med hjälp av en lämplig prompt till chatGPT till valt målspråk. Sedan ska OpenAIs röst läsa upp den på ett annat språk.

16. Låt kungen läsa upp dagens väderleksrapport, eller prognosen vi skapade i uppgift 3.

17. Koppla ihop talsyntes med röstigenkänning och gör en chatt-robot som kan prata med dig. Kan göras ganska enkelt med långsam svarshastighet, men också mer komplicerat genom att streama ljud upp till transkriberingen, streama text till GPT och streama svartstext till en talsyntes. Vad händer om någon stream går snabbare än någon annan? Ska man lägga in utfyllnadsord som ”Hmmm”, ”eeeeee” eller ”öööööhh”?

18. Fredrik har alltid önskat ha längre konversationer med Alina, just nu är det enda ryska Fredrik kan säga ”Ojski Bojski”. Översätt det man säger i datorns mikrofon till ryska och läs upp det med en talsyntes, så Fredrik kan prata mer med Alina.

19. Skapa en AI som läser in en PDF-fil, kanske en forskningsartikel (Min favoritartikel är ”Attention is all you need”: `https://arxiv.org/pdf/1706.03762`) och låt AI:n förklara innehållet för dig med en behaglig röst. Om du vill kan du även lägga till så att AI:n lyssnar på dig och låter dig ställa frågor om artikeln.

20. Skapa en AI som läser av spelplanen av codenames med hjälp av din webbkamera eller en bild du laddar in. Sen ska programmet läsa upp ett ord och en siffra som indikerar hur många kort som ordet refererar till, inget annat. Använd t.ex. statsministerns röst.

21. Skapa ett äventyrsspel där du går runt och interagerar med agenter som kan prata med dig, inte bara i text utan också med hjälp av rösten.

3. C++ (Fredriks grupp)

18 juli

I alla exempel nedan betyder `[...]` att du ska ersätta det med en eller flera rader egen kod.

3.1 Variabler

1. **Personlig information**

   Skapa variabler för namn, ålder, och favoritfärg. Be användaren mata in dessa värden och skriv ut dem i ett välkomstmeddelande.

```
#include <iostream>
using namespace std;

int main() {
    string name, color;
    int age;

    cout << "Vad heter du? ";
    [...]
    cout << "Hur gammal är du? ";
    [...]
    cout << "Vad är din favoritfärg? ";
    [...]

    cout << "Hej " << name << "! Du är " << age << " år gammal och din
        favoritfärg är " << color << "." << endl;
    return 0;
}
```

2. **Valutakonvertering**

   Skapa ett program som konverterar svenska kronor (SEK) till euro (EUR). Använd en variabel för växelkursen och låt användaren mata in ett belopp i SEK.

```
#include <iostream>
using namespace std;

int main() {
    double sek, eur, rate = 0.087;

    cout << "Mata in belopp i SEK: ";
    cin >> sek;
    [...]
    cout << sek << " SEK motsvarar " << eur << " EUR." << endl;
    return 0;
}
```

3.2 If-satser

3. Gissa Fredriks robot

Be användaren gissa vad Fredriks favoritrobot heter. Om användaren gissar rätt, skriv ut ett gratulationsmeddelande.

```
#include <iostream>
using namespace std;

int main() {
    string guess;

    cout << "Gissa vad Fredriks favoritrobot heter: ";
    cin >> guess;

    [...]
        cout << "Grattis! Du gissade rätt!" << endl;
    [...]
        cout << "Tyvärr, fel gissning. Fredriks favoritrobot är R2D2." << endl;
    [...]
    return 0;
}
```

4. Jämför tal

Be användaren mata in två tal och skriv ut vilket som är störst, minst eller om de är lika.

```
#include <iostream>
using namespace std;

int main() {
    int num1, num2;

    cout << "Mata in två tal: ";
    cin >> num1 >> num2;

    [...]
        cout << num1 << " är större än " << num2 << "." << endl;
    [...]
        cout << num1 << " är mindre än " << num2 << "." << endl;
    [...]
        cout << num1 << " är lika med " << num2 << "." << endl;
    [...]
    return 0;
}
```

5. **Ålderskontroll**

Skapa ett program som kollar om användaren är myndig (över 18 år) eller inte.

```
#include <iostream>
using namespace std;

int main() {
    int age;

    cout << "Mata in din ålder: ";
    cin >> age;

    [...]
        cout << "Du är myndig." << endl;
    [...]
        cout << "Du är inte myndig." << endl;
    [...]
    return 0;
}
```

3.3 Switch-satser

6. **Filmgenre rekommendation**

Be användaren mata in en siffra för att välja en filmgenre (1-Action, 2-Komedi, 3-Drama, 4-Skräck). Använd en `switch`-sats för att rekommendera en film inom den genren.

```
#include <iostream>
using namespace std;

int main() {
    int genre;
    cout << "Välj en filmgenre (1-Action, 2-Komedi, 3-Drama, 4-Skräck): ";
    cin >> genre;
    [...]
            cout << "Vi rekommenderar: Die Hard" << endl;
    [...]
            cout << "Vi rekommenderar: The Hangover" << endl;
    [...]
            cout << "Vi rekommenderar: The Shawshank Redemption" << endl;
    [...]
            cout << "Vi rekommenderar: The Conjuring" << endl;
    [...]
            cout << "Ogiltig genre." << endl;
    [...]
    return 0;
}
```

7. **Interaktiv miniräknare**

Skapa ett program som ber användaren mata in två tal och en operation (addition, subtraktion, multiplikation, division) och sedan utför operationen.

```
#include <iostream>
using namespace std;

int main() {
    double num1, num2;
    char op;

    cout << "Mata in två tal: ";
    cin >> num1 >> num2;
    cout << "Mata in en operation (+, -, *, /): ";
    cin >> op;

    switch (op) {
        case '+':
            cout << "Resultat: " << num1 + num2 << endl;
            break;
        [...]
        default:
            cout << "Ogiltig operation." << endl;
    }
    return 0;
}
```

Glöm inte att hantera division med noll, så inte programmet kraschar.

3.4 While-loopar

8. **Gissa talet med Fredrik**

Skapa ett program där användaren ska gissa ett tal mellan 1 och 100 som Fredrik tänker på. Använd en `while`-loop för att fortsätta tills användaren gissar rätt.

```
#include <iostream>
using namespace std;

int main() {
    int secret = 42; // Fredriks hemliga tal
    int guess;

    cout << "Gissa Fredriks hemliga tal (mellan 1 och 100): ";
    cin >> guess;

    [...]
        if (guess < secret) {
            cout << "För lågt! Försök igen: ";
        } else {
            cout << "För högt! Försök igen: ";
```

```
        }
        cin >> guess;
    [...]

    cout << "Rätt! Fredriks hemliga tal var 42." << endl;
    return 0;
}
```

9. Räkna ned från 10

Använd en `while`-loop för att skriva ut tal från 10 till 1 och sedan skriv ut ”Liftoff!”.

```
#include <iostream>
using namespace std;

int main() {
    int i = 10;

    [...]

    cout << "Liftoff!" << endl;
    return 0;
}
```

10. Räkna och skriv ut summan

Be användaren mata in ett tal och använd en `while`-loop för att räkna summan av alla tal från 1 till det inmatade talet.

```
#include <iostream>
using namespace std;

int main() {
    int n, sum = 0, i = 1;

    cout << "Mata in ett tal: ";
    cin >> n;

    [...]

    cout << "Summan av alla tal från 1 till " << n << " är " << sum << "." <<
        endl;
    return 0;
}
```

3.5 For-loopar

11. Multiplikationstabellen

Skriv ut multiplikationstabellen för ett tal som användaren matar in.

```
#include <iostream>
using namespace std;

int main() {
    int num;

    cout << "Mata in ett tal för att få multiplikationstabellen: ";
    cin >> num;

    [...]
        cout << num << " * " << i << " = " << num * i << endl;
    }
    return 0;
}
```

12. Fibonacciserie

Skriv ut de första 10 talen i Fibonacciserien.

```
#include <iostream>
using namespace std;

int main() {
    int n1 = 0, n2 = 1, n3, i;

    cout << n1 << " " << n2 << " ";

    for (i = 2; i < 10; ++i) {
        [...]
    }
    return 0;
}
```

13. Skriv ut varje bokstav i ett namn

Be användaren mata in sitt namn och skriv ut varje bokstav i namnet på en ny rad.

```
#include <iostream>
using namespace std;

int main() {
    string name;

    cout << "Mata in ditt namn: ";
    cin >> name;

    for (int i = 0; i < name.length(); i++) {
```

```
        [...]
    }
    return 0;
}
```

14. Break och continue

Skapa en loop som skriver ut alla tal från 1 till 20, men hoppar över tal som är delbara med 3 (använd `continue`) och avslutar loopen om talet är större än 15 (använd `break`).

```
#include <iostream>
using namespace std;

int main() {
    for (int i = 1; i <= 20; i++) {
        [...]
            continue;
        [...]
            break;
        [...]
        cout << i << endl;
    }
    return 0;
}
```

3.6 Do-while

15. Validerad inmatning

Be användaren mata in ett tal mellan 1 och 10. Använd en `do-while`-loop för att säkerställa att inmatningen är giltig.

```
#include <iostream>
using namespace std;

int main() {
    int number;
    [...]
        cout << "Mata in ett tal mellan 1 och 10: ";
        cin >> number;
        if ([...]) {
            cout << "Ogiltigt tal, försök igen." << endl;
        }
    } while ([...]);

    cout << "Tack! Du matade in ett giltigt tal: " << number << endl;
    return 0;
}
```

16. Menyval

Skapa en enkel meny med flera alternativ. Använd en `do-while`-loop för att fortsätta visa menyn tills användaren väljer att avsluta.

```
#include <iostream>
using namespace std;

int main() {
    int choice;

    [...]
        cout << "1. Säg hej" << endl;
        cout << "2. Säg adjö" << endl;
        cout << "3. Avsluta" << endl;
        cout << "Välj ett alternativ: ";
        cin >> choice;

        switch (choice) {
            [...]
        }
    [...]

    return 0;
}
```

17. Enkel gissningslek

Skapa ett program där användaren ska gissa ett tal mellan 1 och 10 tills de gissar rätt. Använd en `do-while`-loop.

```
#include <iostream>
using namespace std;

int main() {
    int secret = 7; // Det hemliga talet
    int guess;

    [...]

    cout << "Grattis! Du gissade rätt." << endl;
    return 0;
}
```

3.7 Funktioner

18. Hälsning med namn

Skapa en funktion som tar ett namn som parameter och skriver ut en hälsning med namnet. Använd funktionen i ett program.

```
#include <iostream>
using namespace std;

void greet(string name) {
    [...]
}

int main() {
    string name;

    cout << "Vad heter du? ";
    cin >> name;

    greet(name);
    return 0;
}
```

19. Grundläggande funktioner

Skapa en funktion som tar två tal som parametrar och returnerar deras summa. Använd funktionen i ett program som ber användaren mata in två tal och sedan skriver ut summan.

```
#include <iostream>
using namespace std;

[...]

int main() {
    double num1, num2;

    cout << "Mata in två tal: ";
    cin >> num1 >> num2;

    cout << "Summan är: " << add(num1, num2) << endl;
    return 0;
}
```

20. Primtalskontroll

Skapa en funktion som tar ett heltal som parameter och returnerar true om talet är ett primtal, annars false. Använd funktionen i ett program.

```
#include <iostream>
using namespace std;

[...]

int main() {
    int num;

    cout << "Mata in ett tal: ";
    cin >> num;

    if (isPrime(num)) {
        cout << num << " är ett primtal." << endl;
    } else {
        cout << num << " är inte ett primtal." << endl;
    }
    return 0;
}
```

21. Enkel robotstyrning

Skapa en enkel simulering där en robot rör sig på en 2D-yta. Använd `while`-loopar och if-satser för att styra robotens rörelser baserat på användarens inmatning.

```
#include <iostream>
using namespace std;

int main() {
    int x = 0, y = 0;
    char move;

    cout << "Styr roboten (w: upp, s: ned, a: vänster, d: höger, q: avsluta): ";
    while (cin >> move) {
        [...]

        switch (move) {
            [...]
        }
        cout << "Robotens position: (" << x << ", " << y << ")" << endl;
    }
    return 0;
}
```

22. AI för att gissa ett tal

Implementera ett program där datorn gissar ett tal som användaren tänker på, och använd binärsökning (med hjälp av if-satser) för att snabbt hitta rätt tal.

```
#include <iostream>
using namespace std;

int main() {
    int low = 1, high = 100, guess;
    char response;

    cout << "Tänk på ett tal mellan 1 och 100 så kommer datorn att gissa det."
        << endl;

    while (low <= high) {
        guess = low + (high - low) / 2;
        cout << "Är talet " << guess << "? (h: högre, l: lägre, j: ja): ";
        cin >> response;

        if (response == 'j') {
            cout << "Datorn gissade rätt!" << endl;
            break;
        }
        [...]
    }
    return 0;
}
```

3.8 Matteuppgifter

23. Primtal

Skapa en funktion som tar ett heltal som parameter och returnerar true om talet är ett primtal, annars false. Använd funktionen i ett program som skriver ut alla primtal mellan 1 och 100.

```
#include <iostream>
using namespace std;

[...]

int main() {
    for (int i = 1; i <= 100; i++) {
        if (isPrime(i)) {
            cout << i << " ";
        }
    }
    cout << endl;
    return 0;
}
```

24. Fibonaccital

Skapa en funktion som beräknar det n:te Fibonacci-talet och använd den för att skriva ut de första 20 Fibonacci-talen.

```
#include <iostream>
using namespace std;

[...]

int main() {
    for (int i = 0; i < 20; i++) {
        cout << fibonacci(i) << " ";
    }
    cout << endl;
    return 0;
}
```

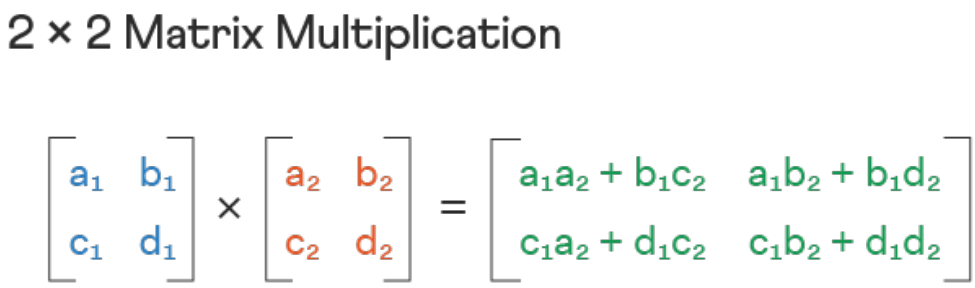
25. Matte för Valentina och Alina

Skapa en funktion som löser en andragradsekvation och använd den i ett program som hjälper Valentina att förklara konceptet för Alina.

```
#include <iostream>
#include <cmath>
using namespace std;

void solveQuadratic(double a, double b, double c) {
    [...]
        cout << "Rötterna är: " << x1 << " och " << x2 << endl;
    [...]
        cout << "Rötterna är lika: " << x << endl;
    [...]
        cout << "Ingen reell lösning." << endl;
    [...]
}

int main() {
    double a, b, c;

    cout << "För att lösa en kvadratisk ekvation (ax^2 + bx + c = 0), mata in
        värden för a, b, och c: ";
    cin >> a >> b >> c;

    solveQuadratic(a, b, c);
    return 0;
}
```

26. Multiplikation av två matriser

Skapa ett program som multiplicerar två matriser av storlek 2x2 och skriver ut resultatet.



Figur 3.1: Matrismultiplikation

```
#include <iostream>
using namespace std;

int main() {
    int a[2][2], b[2][2], c[2][2];

    cout << "Mata in elementen i den första matrisen (2x2): ";
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            cin >> a[i][j];
        }
    }

    cout << "Mata in elementen i den andra matrisen (2x2): ";
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            cin >> b[i][j];
        }
    }

    [...]

    cout << "Resultatet av matris-multiplikationen är: " << endl;
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            cout << c[i][j] << " ";
        }
        cout << endl;
    }
    return 0;
}
```

27. Tobias kartberäkning

Simulera ett enkelt navigationssystem där användaren matar in två koordinater och programmet beräknar och skriver ut riktningen mellan dem.

```
#include <iostream>
#include <cmath>
using namespace std;

[...]

int main() {
    double x1, y1, x2, y2;

    cout << "Mata in startkoordinater (x1 y1): ";
    cin >> x1 >> y1;
    cout << "Mata in slutkoordinater (x2 y2): ";
    cin >> x2 >> y2;

    cout << "Avståndet mellan punkterna är: " << calculateDistance(x1, y1, x2,
        y2) << " enheter." << endl;
    return 0;
}
```

4. C++ (Tobias grupp)

18 juli

Syftet med dagens lektion är att lära er så många coola saker om C++ som möjligt, under antagandet att ni redan har lite kunskap om språket. Hoppas ni gillar det :). C++ är ett språk som har massor med features. Många bra, och en del som är gamla och kanske bör undvikas. Vi kommer under dagens lektion bara att gå igenom features som vi tycker är intressanta/viktiga för att kunna programmera i C++, eller för att förstå saker bättre.

Uppgifterna här är uppdelade i olika delar. Ifall ni känner att ni redan vet massor om ett visst ämne så kan ni hoppa över det ämnet, men läs gärna igenom uppgifterna först för att se ifall det finns något ni inte kan. Vi har försökt att ge en hel del exempel, men ifall ni vill ha mer info om det som vi inte tar upp så är det bara att fråga eller googla er fram till mer information då det är omöjligt för oss att ta upp allting.

Att köra ett program

För att köra ett program skrivet i C++ måste man först kompilera det. Det kan man göra genom att skriva `g++ file.cpp -o output_name` i terminalen. Sedan kan man köra programmet genom att skriva `./output_name`.

Kompileringsflaggor

Det är väldigt vanligt att program kraschar, oavsett vilket språk man kodar i. Något som är svårt i C++ jämfört med en del andra språk är att förstå vad som har gått fel. Därför ska vi titta på ett exempelprogram som kraschar för att se ifall vi kan förstå det bättre.

1. För följande exempel, identifiera först vad koden gör och vad som kan göra att programmen kraschar. Kör sedan programmen och se vad ni får för output när programmet kraschar.

```
#include <iostream>
#include <vector>

int main()
{
    std::vector<int> v = {10,20,50};
    int n;
    std::cin >> n;
    for(int i = 1; i <= n; i++){
        std::cout << v[i] << std::endl;
    }
    return 0;
}
```

Exakt vad ni får för output kan variera beroende på bland annat vilken dator ni kör på och vilka kompileringsflaggor ni använder. Kompilerade ni programmet genom att skriva `g++ program.cpp -o output_name`, så fick ni antagligen en rad output där det står:

Segmentation fault (core dumped)

Det säger inte riktigt vad som orsakade problem, och kan generellt sett orsakas på många olika sätt. Ett sätt som vi kan minska risken att få dessa kryptiska meddelanden är att lägga till en kompileringsflagga: `-fsanitize=address`. Kompilera om programmet med samma kommando som innan, men lägg till den här flaggan. Om ni kör om programmet nu så lär ni se att ni får mycket mer text som beskriver vad som gick fel.

Bland annat finns en stack trace av var kraschen skedde. Men var nånstans i koden skedde kraschen? Det står med, men är väldigt kryptiskt. Vi kan lägga till ytterligare en kompileringsflagga för att se till så att det blir lättare att läsa det: `-g`. Lägg till `-g` och kompilera om igen. Nu bör ni se ett par radnummer som del av outputen. Det finns även en hel del annan information som är bra att förstå, men det är lite för mycket för att beskriva allt här. Fråga en ledare om hjälp så berättar de mer!

Vi rekommenderar att ni använder dessa flaggor varje gång ni testkör era C++ program. Men finns det någon nackdel med detta? Vad tror ni?

2. Testkör följande kod. Vad får ni för output. Försök först att komma fram till vad det borde bli och kör det sedan.

```
#include <iostream>
using namespace std;
int main()
{
    int x = 128;
    char c = x;
    cout << (int)c << endl;
}
```

3. Lägg till kompileringsflaggan `-Wconversion` och kompilera om programmet från förra uppgiften. Vad ser ni nu?

Som ni ser så kan man kontrollera vilka varningar som genereras när man kompilerar sin kod. Det är väldigt användbart att få dessa varningar då det signalerar att det kanske finns en bugg i ditt program. Det finns alldelles för många kompileringsflaggor för att gå igenom alla här. Men ett par vanliga och användbara flaggor är:

- `-Wall` - Står för warning all, dock sätter den faktiskt inte på alla varningar. Men den varnar för mycket saker.
- `-Werror` - Står för warning error. Det gör att alla varningar man får blir till errors istället, vilket gör att kompileringen failar. Tanken med detta är att tvinga programmerare att skriva kod som inte generar varningar, och på så sätt undvika buggar.
- `-O0` - Optimeringsnivå 0. Det finns fortfarande ett fåtal optimeringar som är påslagna, men det mesta är av. Detta är bra för debugging då koden gör i princip exakt det du skrev, rad för rad.
- `-O1, -O2, -O3` - Olika optimeringsnivåer. Ju högre tal desto mer opitemerar kompilatorn din kod, vilket leder till snabbare exekverbara filer.

Typer/storlekar av typer

I C++ måste man för det mesta vara explicit med typerna. Det finns massor av typer att välja mellan, och vad man vill använda beror på syftet. T.ex. för heltal så räcker det ofta med en int, så man behöver inte använde mer än så då. Men behöver man spara större heltal så kan man byta typ. Ett par av de vanligaste typerna som finns inbyggda är:

```
// Tekniskt sett så har följande typer inte samma storlek på alla system. Men
    på i princip alla system så gäller följande storlekar.
char // Heltalsvärde med 8 bitar.
short // Heltalsvärde med 16 bitar.
int // Heltalsvärde med 32 bitar.
long long // Heltalsvärde med 64 bitar.

float // Decimaltal som sparas som 32 bitar.
double // (Bättre än floats) Decimaltal. Sparas som 64 bitar i minnet.
long double // Decimaltal. Sparas som 80 bitar i minnet, men kanske inte finns
    på typ vissa windowssystem.

unsigned char
unsigned short
unsigned int
... // Att lägga till unsigned gör att det går att representera större
    positiva tal då det inte längre går att lagra negativa tal i datatypen.
    Dock är det ibland farligt att använda unsigned datatyper då det är lätt
    att råka ut för underflow.

char * // Tolkas ofta som en sträng, särskilt i C.

// Man kan även skapa arrays på stacken genom att skriva någonting i stil med:
int arr[5]; // Innehåller 5 ints.

// Man kan beräkna antalet bytes av en variabel/datatyp genom att skriva t.ex.:
sizeof(int) // == 4. För typer krävs en parantes.
sizeof arr // == 20. För variabler krävs det inte en parantes.
```

4. Vad händer ifall du skriver ut sizeof på en pekare?
5. Hur kan du beräkna längden på en array?

Man kan även använda auto ibland för att skippa att skriva ut typen. Då skriver man `auto x = ...;` och kompilatorn kommer automatiskt att lista ut vilken typ x ska ha. Detta används ibland, men bör inte överanvändas då det är bra att lätt kunna se vilken typ en variabel har. Det är dock användbart när man t.ex. skapar variabler som har typer som är väldigt långa att skriva, och som bara kommer att användas i ett väldigt litet scope.

Includes/använda flera filer

Ifall man vill använda sig utav funktioner som ligger i en annan fil så måste man inkludera en fil som innehåller en deklaration av den funktionen. Detta gör man genom att skriva t.ex.:

```
#include <iostream> // For things like cout, cerr etc.
#include "my_other_cool_file.hpp"
```

Man kan alltså använda antingen <> eller "" men enligt kompilatorn så betyder det samma sak. Det är dock ganska standard att man brukar använda <> för standardbiblioteket, och "" för egna filer.

Det som `include` egentligen gör är att är bara att copy pasta hela filens innehåll så att det finns tillgängligt. Därför skulle man kunna tro att saker lätt kan gå fel ifall man råkar inkludera samma fil många gånger. Men det kan man lätt lösa genom att skrive `#pragma once` i början av en fil. Det ser till så att en fil bara inkluderas första gången den försöker bli inkluderad.

Datastrukturer

Ni är förmodligen bekanta med en del datastrukturer redan, men här kommer ett par korta uppgifter där ni kan lära er att använda de vanligaste datastrukturerna.

T.ex.:

```
#include <vector>
#include <unordered_map>
#include <iostream>
using namespace std;
int main()
{
    vector<int> my_cool_vector; // Creates an empty vector.
    my_cool_vector.push_back(5);
    cout << my_cool_vector[0] << endl;

    unordered_map<int, int> my_cool_map; // Creates an empty unordered map.
    my_cool_map[5] = 7;
    my_cool_map[10] = 20;
    cout << my_cool_vector[10] << endl;
    coutt << my_cool_vector.count(8) << endl;
}
```

6. Skriv ett program där ni skapar en `vector` som innehåller strings. Skriv sedan en funktion som tar bort de strängar som innehåller bokstaven `'a'`.

7. Skriv en funktion som returnerar hur många unika element det finns i en `vector` av integers. Tips: för en bra tidskomplexitet, använd er utav antingen `std::set` eller `std::unordered_set`.

8. Skriv kod som loopar igenom en unordered_map, och skriver ut dess inehåll, genom att använda auto.

Mer avancerade datastrukturer

Det finns även många andra väldigt användbara datastrukturer som finns inbyggda i standardbiblioteket. Ett par exempel är: `std::map`, `std::unordered_map`, `std::queue`, `std::priority_queue`, `std::stack`, `std::bitset`, `std::pair`, `std::tuple`.

9. Använd en tuple för att returnera en sträng, en int och en float samtidigt från en funktion.

Pass by reference/pointer/value

När man skriver en funktion i C++, så väljer man vilka typer som argumenten har. Det finns tre huvudsakliga alternativ för detta, som vi kommer att gå igenom nu.

Det enda som ändrar på vilket sätt man skickar nått till en funktion är hur funktionen är deklarerad.

```
// Option 1:
// Calling this function will copy the whole vector, which can sometimes slow
    things down.
int some_cool_function(std::vector<int> vec);

// Option 2:
// To call this function you need to provide the pointer to the vector. It
    won't be copied.
int some_cool_function(std::vectior<int> *vec);

// Option 3:
// You call this function in the same way as option 1, but in this case no
    copy of the original vector is created. Instead, it is passed as a
    reference. This means that if you modify the vector in the function, the
    vector that was used to call the function is also modified.
int some_cool_function(std::vector<int> &vec);
```

Iteratorer

I C++ är det ganska vanligt att använda sig av iteratorer. Detta är i princip en klass som tillåter att man lätt kan stega igenom data på ett sekventiellt sätt. Till exempel ifall man vill loopa igenom en `map` så använder kan man sig utav en iterator:

```
map<string, string> var;
auto iter = begin(var);
while(iter != end(var)){
    string key = iter->first;
    string value = iter->second;
    iter++;
}
```

Iteratorer användes även väldigt ofta utan att man ens tänker på det. T.ex. som input till en del funktioner som vi kommer ta upp senare. Notera att pilsyntaxen är som att skriva en punkt på en variabel, fast man använder det för pekare och iteratorer.

10. Skapa en iterator som går igenom en vector. Stega igenom listan på ett sånt sätt att du printar var femte element i listan.

Lambdas

Lambdafunktioner är väldigt lika vanliga funktioner, förutom att de inte har något namn. Det gör det väldigt lätt att skicka det som input till en funktion som tar in en funktion som input, men man kan även spara det i ev variabel. Här är ett exempel på hur man skapar en lambdafunktion som tar in en int och en string, och returnerar en int:

```
...
int main()
{
    auto cool_lambda_function = [](int x, std::string s){
        int return_value = ...;
        return return_value;
    };
}
```

Mycket här ser ut som en vanlig funktion förutom att det finns ett semikolon på slutet och att det finns ett mystiskt `[]` efter likhetstecknet. I den kan man specifiera vilka variabler lambdafunktionen ska komma åt från det yttre scopet. Just så som den är skriven här kan funktionen inte komma åt några variabler från main funktionen. Vill man att den ska göra det så går det att specifiera. Man kan till och med välja exakt vilka variabler lambdafunktionen får ha access till, och ifall det ska vara by value eller by reference. Ifall man vill att lambdafunktionen ska ha access till alla funktioner by reference i main funktionen så skriver man:

```
...
int main()
{
    auto cool_lambda_function = [&](int x, std::string s){
        int return_value = ...;
        return return_value;
    };
}
```

11. Skriv en lambdafunktion som kvadrera ett heltal.

12. Skriv en lambdafunktion som lägger till ett tal 5 gånger i en vector som är definierad i den yttre funktionen, utan att skicka in det som en parameter.

Standardbiblioteket och ett par användbara funktioner

Det finns massor av användbara funktioner i standardbiblioteket. Här kommer ett par av dessa:

```
#include <vector>
#include <algorithm>
int main()
{
    vector<int> x = ...;
    sort(begin(x), end(x)); // Takes an iterator to the first element and an
        iterator to the end of the thing that you are sorting. This allows you
```

```
        to do things like easily sorting part of the vector etc.
}
```

13. Sortfunktionen kan även ta in en tredje parameter: en funktion som avgör hur man ska sortera innehållet. Detta kan antingen vara en funktion som du har definierat på ett annat ställe i ditt program, eller en lambdafunktion.

Skriv kod som sorterar en vector av ints med hjälp av en lambdafunktion så att den sorteras i stigande ordning på absolutbeloppet av innehållet. Om du bara ska använda lambdafunktionen på ett ställe så behöver du inte mellanlagra den i en variabel.

14. Det finns en funktion som heter `reverse`. Den tar in två iteratorer precis som sort. Skriv ett program som vänder på en vector med hjälp av denna funktionen.

Ett par andra väldigt användbara funktioner är `lower_bound` och `upper_bound`. Dessa funktioner utför en binärsökning för att hitta det man letar efter i ett intervall. Eftersom det är en binärsökning så krävs det att datan är sorterad. Både `lower_bound` och `upper_bound` tar in tre argument: först två stycken iteratorer precis som med de tidigare funktionerna, och sedan det värde man söker efter.

15. Experimentera med att använda `lower_bound` och `upper_bound` för att binärsöka genom en sorterad lista. Försök att se ifall ni kan avgöra vad skillnaden är mellan dessa funktioner. Notera att returvärdet är en iterator till det värde som binärsökningen hittar.

Pekare

En pekare är en variable som är lika med en minnesadress. Det används överallt i C kod, och även en del i C++. Man kan använda pekare till massor av saker. T.ex. kan man skapa en array genom att använda pekare, eller så kan man skicka pekare till en funktion för att få något som liknar en referens.

```
int *arr = new arr[10]; // Creates a pointer that points to memory containing
    10 ints.
...
delete arr[]; // You need to do this in order to free memory. Otherwise you
    won't be able to reuse this memory.
```

16. Skapa ett program där du inuti en while-loop allokerar små arrayer utan att köra delete. Vad händer ifall du låter programmet köra i ett par sekunder/minuter?

Smart pointers

Det finns ett par problem med vanliga pekare. Dels så måste man alltid frigöra det minne som man allokerar, vilket är väldigt lätt att glömma. Sen ifall det finns flera delare av programmet som pekar på samma minne så är det svårt att veta vilket del av programmet som blir ansvarig för att frigöra minnet. Därför är det ofta bra att undvika vanliga pekare ifall man kan. Istället kan man använda smarta pekare:

```
#include <memory>
void f(std::shared_ptr<some_class> ptr){}

int main()
{
    std::shared_ptr<some_class> ptr1 = std::make_shared<some_class>();
    std::shared_ptr<some_class> ptr2 = ptr1;
    f(ptr2);
}
```

Shared pointers tillåter att det kan finnas flera pekare som pekar på samma minne men det kommer automatiskt att frigöra minne när alla pekarna går ur scopet. Dvs när ingenting längre pekar på minnet. På så sätt slipper man många av beskymrena kring pekare.

Det finns även `std::unique_ptr` och `std::weak_ptr`. Unique pointers tillåter endast att en variabel pekar på det minna som den är ansvarig för, och den frigör automatiskt minnet när den går ur scope. Unique pointers syfte är ofta att markera att programmet endast bör ha en referens till en variabel som har blivit allokerad på heapen.

17. Skapa en funktion `create_unique_pointer` som skapar en unique pointer till en int och returnerar den. Visa hur ownership överförs från en unique pointer till en annan.

18. Skapa en funktion `create_shared_pointer` som skapar en shared pointer till en int och returnerar den. Visa hur du kan ha flera variabler som alla pekar på samma sak.

Asserts

Om ditt program gör något väldigt fel brukar det bara krascha. Men om det gör något lagom fel finns det en risk att det istället fortsätter köra, men gör inte som det ska. Kommandot `assert` låter kolla för att något är sant, och om det är inte sant, så kraschar programmet. Exempel på rimliga användningsmönster:

```
void perform_op(int op_type)
{
    if (op_type == 0) do_0();
    else if (op_type == 1) do_1();
    else if (op_type == 2) do_2();
    else assert(false);
}
```

I en graf vill man ibland ha:

```
void add_edge(int a, int b)
{
    assert(a!=b);
    ...
}
```

I en funktion som hittar största värdet i ett intervall av en array:

```
void interval_max(int a, int b)
{
    assert(a<=b);
    ...
}
```

För vissa icketriviala lösningar kan man orsaka undefined behavior om den asserten inte håller. Då kan man alltså inte alls lita på att det ska krascha.

Ibland är det inte trivialt att avgöra om det gått fel i början av en funktion. Då kan man blanda in asserts under algoritmens körning, och om man märker att det går katastrofalt fel så avslutas allt.

Viktigt är däremot att assert endast ska användas som debugging-verktyg. Det ska exempelvis inte vara the last line of defense så att användares hemligheter inte läcks eller dylikt. Detta beror på att compilern kan välja att kompilera den till en no-op om man kör med -O2 eller högre. Detta betyder också att det är ytterst viktigt att inte skriva kod som har några sidoeffekter innuti en assert. Den koden kommer ju inte att köras med vissa kompileringsflaggor.

Lär dig lita på kompilatorn

C++ kompilatorn är riktigt bra på att optimera kod som du har skrivit. Det bästa du kan göra för att optimera din kod är att skriva en algoritm som är snabb. Men att optimera en rad i taget för att undvika vissa småsaker är ofta inte värt det. Ett vanligt exempel är att folk väljer att skriva `(x >> 1)` istället för `(x / 2)` då de tänker att det är snabbare. Saningen är dock att det kompileras till exakt samma kod. Så det är smartare att behålla mer lättläst kod och låta kompilatorn göra jobbet åt dig.

Det finns många andra exempel som kompilatorn kan optimera till bättre kod. Till exempel kan den unrolla korta for-loopar, och ersätta en komplicerad beräkning med en konstant.

Dock är det inte alltid så att kompilatorn lyckas att optimera kod, så ifall man verkligen vill veta vad kompilatorn klarar av så får man läsa på mer om det.

Ett sätt man kan se vad kompilatorn gör med ens kod är att kompilera ner koden till assembler. Det kan man exempelvis göra på `https://godbolt.org/`. Där kan man direkt se vilken del av koden som översätts till vad i assembler. Vi hinner inte att gå igenom exakt hur man ska tolka assembler under dagens lektion, men jobbar man mycket med C++ så är det värt att åtminstone kunna lite grann.

19. Gå in på godbolt.org och sa vad följande kod översätts till i assembler. Teste först utan att ha några särsklida kompileringsflaggor, och lägg sedan till flaggan -O2 som kommer se till att koden optimeras mer. Kan ni se någon skillnad i vad funktionerna kompileras till? Utan att förstå vad varje rad assembler gör, kan ni gissa varför det blir så stor skillnad?

```
int compiler_dislikes_this(int n)
{
    int ret = 0;
    for(int i = 0; i < n; i++){
```

```
        ret += i;
    }
    return ret;
}

int compiler_likes_this()
{
    int n = 1000;
    int ret = 0;
    for(int i = 0; i < n; i++){
        ret += i;
    }
    return ret;
}
```

Structs/Klasser

Precis som andra språk så finns det klasser i C++. Men speciellt för C++ är att det finns två olika val: `struct` och `class`. Den enda skillnaden i vad de egentligen betyder är att en structs innehåll är public per default vilket betyder att man får accessa alla variabler som inte explicit markeras som private eller protected. I en class är default att variabler är private istället.

Dock brukar structer och klasser användas på olika sätt. T.ex. enligt googles styleguide så används bara structs som en container för att spara data av olika typer och gruppera ihop dem. Klasser används för mer komplicerade saker, och där kan man lägga in metoder, arv och andra en massor av andra kul saker man kan tänka sig göra.

Här är ett exempel på hur en simpel klass kan se ut:

```
class Person {
private:
    // These variables are private. They can only be accessed from the methods
        inside the class. This prevents users from modifying things that they
        should not be modifying.
    std::string name;
    int age;

public:
    // The functions down here are public, meaning they can be called from
        other places in the code.

    // Constructor
    Person(const std::string &name, int age) : name(name), age(age)
    {
        std::cout << "Constructor called: " << name << " is created.\n";
    }

    // Destructor. This is called when the variable goes out of scope, or when
        you call delete.
    ~Person() {
```

```
        std::cout << "Destructor called: " << name << " is destroyed.\n";
    }

    // Method to display person's information.
    void displayInfo()
    {
        std::cout << "Name: " << name << ", Age: " << age << std::endl;
    }
};
```

20. Skriv ett klass för att representera en punkt i 2d. Skapa en vector som innehåller intanser av din klass, och sortera sedan denna vektor i stigande ordning, först baserat på x-koordinat, och sedan på y-koordinat ifall de har samma x-koordinat.

21. Tidigare gick vi igenom vector, och hur pekare/smarta pekare fungerar. Välj antingen `vector` eller `shared_ptr` och implementera den klassen från scratch, för en valfri typ (t.ex. bara för int). Eran version bör inte ha stöd för alla metoder som finns i `std::vector` eller `std::shared_ptr`, men ni bör se till att hålla koll på minnet internt.

Templates

Låt säga att du skriver ett program där du använder arrayer av tal och du behöver ofta garantera att alla är positiva. Då kanske du skriver:

```
void make_positive(vector<int> &numbers)
{
    for (size_t i = 0; i < numbers.size(); i++)
    {
        numbers[i] = abs(numbers[i]);
    }
}
```

Allt är frid och fröjd. Men en dag behöver du använda den med flyttal:

```
void make_positive(vector<double> &numbers)
{
    for (size_t i = 0; i < numbers.size(); i++)
    {
        numbers[i] = abs(numbers[i]);
    }
}
```

Men oj, nu ber chefen dig att ibland använda normala flyttal för att spara på minne. Blir det ännu en overload? Nej, nu får det vara nog. Vi kan nämligen skriva en version som funkar för alla datatyper:

```
template<typename T>
void make_positive(vector<T> &numbers)
{
    for (size_t i = 0; i < numbers.size(); i++)
```

```
    {
        numbers[i] = abs(numbers[i]);
    }
}
```

Denna tar alltså en implicit parameter T, vilket den listar ut från vektorn vi anropar funktionen med. T kan användas precis som någon annan typ, såsom int eller annat innanför funktionen.

Exempel 2: man kan även template:a klasser, vilket är hur i princip hela C++ standardbiblioteket funkar.

```
template<typename T>
struct Node
{
    T value;
    Node *next;
};
```

Vilket vi använder genom

```
Node<int> *root = new Node<int>();
```

22. Skriv en egen linked list-implementation som använder templates. Denna måste endast stödja att lägga till saker på slutet och få värdet av det i:te elementet.

Exempel 3: här kommer ett riktigt exempel som Joshua använt. Låt säga att du vill använda gcc:s datastruktur order statistics tree. Detta är en map som bland annat låter dig snabbt svara på "hur många nycklar har värde $\leq x$" i $O(log_2(N))$, där N är antalet nycklar. Om man använder vanliga std::map tar detta $O(N)$. Låt oss definiera denna, där vi har int som nyckel och värde.

```
#include <bits/extc++.h>
using namespace __gnu_pbds;
typedef tree<int, int, std::less<int>, rb_tree_tag,
    tree_order_statistics_node_update> indexed_map;
```

Här ser vi redan ett trick: för att slippa skriva hela det böset varje gång vi vill skapa en variabel av den typen kan vi använda typedef. Varje gång vi vill använda den räcker det nu med:

```
indexed_map map_example;
```

Men åh nej! Nu vill vi ändra våra nycklar till double. Ska vi copy-pastea den och skapa en ny version? Nej, vi kan templatea typedef:en!

```
template<typename T, typename U>
using indexed_map = tree<T, U, std::less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;
```

Och nu kan vi bara skriva

```
indexed_map<double,int> double_map;
```

Interagera med C kod

Ibland händer det att man har lite kod som är skriven i C och annan kod som är skriven i C++. Som tur är så är det lätt att anropa kod som ligger i en fil skriven i det andra språket. Det räcker egentligen med att man inkluderar header filen som deklarerar funktionerna man vill använda, men man måste dock se till att använda göra det på följande sätt:

```
extern "C" {
#include "my_cool_header_file.h"
}
```

Anledningen till att man måste skriva `extern "C"` är att C++ använder sig utav nånting som kallas för name mangling. Utan att gå in på för mycket detaljer så innebär det att kompilatorn inte lyckas lista ut vilken funktion den borde anropa. Att skriva `extern "C"` löser dock det problemet.

23. Testa att skriva en fil i C som innehåller en funktion som kvadrerar ett heltal, och en fil i C++ som anropar denna funktionen.

24. Testa att skapa en C-funktion som beräknar längden av en sträng. Skapa en `std::string` i C++ och anropa funktionen som du skapade i C. Hur kan du göra det, när `std::string` inte finns i C?

Compile-time template trickery

Rutinerade tävlingsprogrammerare har sedan länge tröttnat på skriva följande för att läsa input:

```
int a, b, c;
cin >> a >> b >> c;
```

Egentligen är det enda vi behöver specificera är typen och namnen på variablerna. En template:ad funktion kan (som tur är) inte skapa variabler där vi anropar den ifrån. Vi kan istället använda C++ preprocessor. Om vi exempelvis skriver

```
#define read(type, name) type name; cin >> name
```

Kan vi sedan skriva

```
read(int, a);
```

Notera att den inte tar typer på argumenten. Detta är för att preprocessorn inte vet att C++ finns: den gör i princip bara blind copy-pasting.

Men tänk om vi vill läsa in 2? 3? 4? Vi kan göra en version för varje antal med method overloading. Eller så kan vi göra häxkonst!

```
void readinput() {} // Recursion base case
template<typename T, typename... Args> void readinput(T& arg, Args&... args)
```

```
{
    cin >> arg;
    readinput(args...);
}
#define read(type, ...) type __VA_ARGS__; readinput(__VA_ARGS__);
```

__VA_ARGS__ innebär att vi tar godtyckligt många argument. Sedan anropar vi readinput med de.

Readinputs template tar som input en variable med godtycklig typ, och sen en lista med variabler som är samlade under Args. Då läser den först in variabeln och anropar sig själv med resten av argumenten. När den anropar sig själv extraheras ännu en gång den första variabeln ut ur listan, och den krymper med 1. Detta fortsätter tills den är tom och når base caset i toppen.

25. Försök förstå koden till fullo.

26. Implementera en print-funktion likt pythons, där du kan printa godtyckligt många argument av olika typer, mellanslagsseparerat. Använd bara templates.

27. Använd templates för att beräkna fakulteter vid kompileringstid.

28. Låt säga att du vill skapa en 4d-vektor av ints, där varje dimension har storlek a,b,c,d. Då måste du orioniskt skriva `vector<vector<vector<vector<int>>>> arr = \ vector<vector<vector<vector<int>>>>(a, vector<vector<vector<int>>>(b, vector<vector<int>>(c, vector<int>(d))));`
Förenkla detta med en template! Den ska ta typen av vectorn, och skapar den en till dimension för varje variabel, med motsvarande storlek. Så exempelvis ska ovanstående kunna ersättas med
`multi_init(int, a, b, c, d);`
Tips: du kan ha return type auto. std::decltype är väldigt användbar.

Operators

Ifall man har en egen klass, och man vill att den ska stöjda saker som till exempel addition, bitshifting, eller indexering, så tillåter C++ att man skriver vad som ska hända i såna fall. Detta gör man genom att definera en funktion precis som vanligt, men syntaxen är aningen annorlunda.

```
class Point {
private:
    int x, y;

    Point(int x, int y)
    {
        this->x = x;
        this->y = y;
    }

    Point operator+(Point other_point)
    {
        return Point(x + other_point.x, y + other_point.y);
```

```
    }
};
```

Man skriver alltså först typen som operatorn ska returnera följt av ordet operator, och sist den symbol som skrivs när man vill använda sig utav den funktionen. Ett par alternativ som finns är `*`, `+`, `-`, `/`, `>>`, `<<`, `[]`, `()`, `==`, `!=`, `%`, `+=`.

29. Skriv en klass för att representera bråktal. Erat program bör stödja åtminstone de fyra enkla räknesätten.

Cool overloading

Låt säga att du vill ha 2D-vektorer, men är för lat för att skriva en egen klass. std::pair<int,int> är nästan perfekt, men du vill kunna addera de. Då kan man lätt implementera detta, trots vi inte äger std::pair!

```
template <typename T, typename U> inline pair<T, U> operator+(const pair<T, U>
    l, const pair<T, U> r)
{
    return { l.first + r.first, l.second + r.second };
}
```

Tack vare att vi använder templates har vi nu definierat + för alla möjliga pairs.

Undefined behavior

I c++ finns det många skumma saker som kan hända. Skriver man exempelvis följande kod, så kommer man med vissa kompilatorflaggor att lyckas köra programmet:

```
int arr[5];
cout << arr[20] << endl;
```

Programmet skulle kunna krascha då vi går out of bounds i vår array, men det skulle också kunna vara så att vi skriver ut det som finns där i minnet. Läskigt!

Har man kod som gör nånting som i detta exempel så innebär det förmodligen att man har en bugg i sitt program eller att man har kodat nånting på ett dåligt sätt. Så oavsett, så bör man undvika detta.

Men, det som gör sånt här extra intressant i C++ är att det finns någonting som heter undefined behavior. Det är egentligen en samling med saker som är förbjudna att göra enligt språket, och ifall man gör det så få kompilatorn dessutom göra vad den vill med den koden. Exempelvis så är overflow på signed integers undefined behavior. T.ex.:

```
int x = 1000000;
x = x * x; // Invokes undefined behavior because of overflow.
```

Ofta får man höra att overflow är en sak som finns i C++ och att det gör så att en variabel *wrappar runt* - Tobias Glimmerfors. Men det är även så att det är undefined behavior. Kompilatorn kan alltså göra något annat med koden, och enligt standarden så är erat program ett ogiltigt C++ program ifall ni har undefined behavior. Så akta er för det.

30. Vad gör följande kod? Försök först att tänka ut vad som kommer hända. Testa

sedan att kompilera/köra den här koden med olika optimeringsnivåer. Testa -O0, -O1, -O2, och -O3.

```
int x = 2147483647 - 10; // 10 less than int max.
for(int i = 0; i < 20; i++){
    cout << x << endl;
    x++;
}
```

31. Förra problemet innehåller undefined behavior, så att nått skumt händer är faktiskt tillåtet. Skriver man sån kod så får man helt enkelt skylla sig själv då man skrev nånting som inte följer språkets standard. Men varför blir det så fel i förra uppfigten? Exeprimentera lite med koden i godbolt.org och se ifall ni kan lista ut vad kompilatorn försöker att göra.

Det finns många andra saker som är undefined behavior. T.ex. är det otillåtet att inkrementera en pekare så att den pekar utanför dett minne som den skapades för. Det är också otillåtet att casta pekare till andra typer och sedan avreferera de....

32. Läs på mer om saker som är undefined behavior online. Försök att hitta nånting obskyrt som Tobias inte känner till, och lär honom något :)

Mer om macron

Macros är egentligen bara copy paste av kod, och detta görs som del av ett så kallat preprocessing steg. Dvs innan koden faktiskt kompileras så händer detta.

Ett vanligt exempel som tävlingsprogrammerar gillar är:

```
#define rep(i,a,b) for(int i = (a); i < (b); i++)
```

Då kan man skriva `rep(i, 0, n)` för att loopa från 0 till och med n - 1, vilket förkortar mängden man behöver skriva.

Det finns en del viktiga saker man bör veta om macron ifall man vill skriva ett macro.

33. Vad skriver följande kod ut? Varför? Tänk igenom ditt svar först innan du testar.

```
#define push_back_twice(vec, x) vec.push_back(x); vec.push_back(x);
int main()
{
    vector<int> vec;
    int x = 5;
    push_back_twice(vec, x++);
    cout << vec[0] << ", " << vec[1] << endl;
}
```

34. Vad gör följande kod? Försök först att lista ut det själv.

```
#define is_equal(a, b) a == b

int main()
```

```
{
    cout << (is_equal(5, 5)) << endl;
    cout << (is_equal(0 || 2, 2)) << endl;
}
```

Som ni ser så måste man vara försiktig när man skriver macros. Ifall man skickar in ett argument till ett macro som har en sidoeffekt så kommer den sidoeffekten att upprepas varje gång det används innuti macrot, vilket man förmodligen inte vill. Detta kan undvikas genom att inte skicka in saker med sidoeffekter till macron.

Man bör också se till att lägga in paranteser runt variablerna man använder innuti macron, om man inte är säker på att det verkligen inte behövs. Alltså borde man egentligen skriva:

```
#define is_equal(a, b) (a) == (b)
```

På så sätt kommer förra exempelt att fungera mer så som man tänker sig att det borde.

Macros kan även skrivas lite som funktioner, där de exekverar flera påståenden. Detta behöver inte användas så mycket i C++, men är väldigt användbart i C ibland. T.ex.:

```
#define calc_dist(x, y, x2, y2) ({\
    typeof(x) dx = (x) - (x2);\
    typeof(x) dy = (y) - (y2);\
    sqrt(dx * dx + dy * dy);
})
```

Vill man skriva ett macro på flera rader så behöver man lägga in ett backslash i slutet av varje rad förutom den sista. I exemplet `calc_dist` så är det det på den sista raden som blir output när man använder macrot.

Ifall man vill skriva ett macro som inte returnerar nånting så brukar man skriva det på följande sätt:

```
#define some_cool_macro_name(vec, x) do {\
    /* Put some interesting code here */\
} while(0)
```

Det som är innuti do-while kroppen kommer bara att köras en gång eftersom det står `while(0)`, så då blir det som att köra en vanlig funktion. Att det inte finns något semikolon i slutet av macrot gör att man själv måste skriva det när man använder macrot, vilket leder till att macrot känns som att använda en funktion.

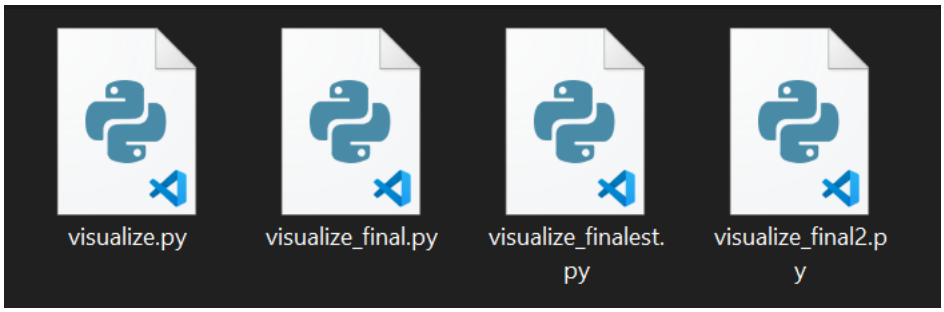
Det finns mycket mer man kan lära sig om macron, så det är bara att fråga ifall ni är intresserade.

Sammanfattning

Det finns massor av saker man kan lära sig om C++. Här har vi nu försökt att gå igenom massor av olika saker ytligt. Vill ni lära er mer om C++ rekommenderar vi att ni läser på mer om de features som ni tyckte verkade mest intressanta.

5. Git

20:e juli



När man arbetar med lite större projekt vill man ibland skapa backups av sin kod när den funkar. Detta är användbart om man exempelvis ska lägga till ny funktionalitet, men ifall allt går fel, så kan man gå tillbaka till när allt funkade. För nyare utvecklare brukar det se ut ungefär som ovan- man skapar kopior av filen. Det visar sig dock att det finns ett mycket mer strukturerat sätt att åstadkomma samma funktionalitet (och mer) tack vare versionshanterinssystemet Git!

Installation av Git

Vi kommer att utgå ifrån Windows. Om du använder Linux har du troligtvis redan fixat allt det här ändå. Om du använder Linux eftersom du lånar liu datorer, fråga Joshua om något av stegen inte funkar. Ladda först ner Git här (tryck på första länken högst upp). När den frågar dig om editor, använd vad som helst förutom Vim, exempelvis notepad eller Visual Studio Code. Tryck bara next på resten.

Github

Git är ett system som finns lokalt på din dator. Github låter dig ladda upp ditt Git-projekt till molnet, vilket både funkar bra som backup, men är även ett bra sätt att ha en delad kodbas mellan dig och kompisar/kollegor. Börja med att skapa ett konto på github.com. För att kunna ladda upp vår kod behöver vi bevisa för Github att ”vi är vi”. Vi kommer att göra detta med så-kallade SSH-nycklar. Starta först en kommandotolk (cmd). Detta kan göras genom att trycka Windows och sen söka cmd. Kör sedan

```
ssh-keygen
```

Tryck enter på file location, skriv sedan in ett lösenord (enter för inget lösenord, men det är rekommenderat att ha). Vi måste sedan ladda in ssh-nyckeln. Detta görs med

```
start-ssh-agent
```

Vi måste nu säga åt Github att lita på vår ssh-nyckel. Kör kommandot

```
type .ssh\id_rsa.pub
```

Den kommer nu skriva ut något som ser ut ungefär såhär:

```
ssh-rsa
    AAAAB3NzaC1yc2EAAAADAQABAAABgQC1eXetcnb23Gviz/OjMC7wkaWFCwQ72spbJji8ggGQn
Yz6r570exNTZQ8leCEusYUrjYlNDsxAfY2hx5EMFNff4HOdi1h8x+31Fky6+BTfLqBXj+My7uwEcQ
....= matis@DESKTOP-HP50M5G
```

Gå sedan in på Github, tryck på din profil högst uppe till höger. I menyn, klicka settings. I menyn som dök upp till vänster, klicka på SSH and GPG keys". Tryck sedan New SSH key"uppe till höger. Klistra sedan in den långa strängen som skrevs ut.

För att som allt funkar som det ska, testa att köra

```
ssh git@github.com
```

Om det skrivs ut ”Hi <namn>! You’ve successfully authenticated, but GitHub does not provide shell access.”, så vet du att allting funkar.

Som det är nu kommer du behöva köra start-ssh-agent varje gång du startar datorn. För att slippa det kan du göra följande: sök först efter powershell. Högerklicka på den och tryck ”Run as administrator”. Kör sedan

```
Start-Service ssh-agent
Set-Service ssh-agent -StartupType Automatic
```

På din vanliga command-line (den svarta), kör

```
ssh-add .ssh\id_rsa
```

För att testa att det funkar kan du starta om datorn köra

```
ssh git@github.com
```

Grattis! Du har nu klarat av den tråkigaste delen av den här lektionen. SSH-nycklar är användbart för mer än bara Git: exempelvis, om du någonsin skaffar en molnburk brukar man logga in med en SSH-nyckel.

Testa på Git

För att få testa på Git har vi skapat ett test-repository. Gå här och klicka ”fork” uppe till höger. Nu kommer du ha en kopia av repositoriet på ditt egna konto. Tryck sedan på den gröna knappen ”Code”, sen på ”SSH”. Kopiera den länken och kör:

```
git clone git@github.com:<username>/gitgood.git
```

(Där git@github-saken är det du kopierade). Öppna sedan mappen git-tutorial med din favorit-kodredigerare (exempelvis Visual Studio Code).

Nu kan vi förklara hur git funkar. Du arbetar med så kallade ”commits”. Dessa kan ses som checkpoints, och du kan gå tillbaka till alla gamla commits någonsin. Det är bra att namnge commiten med vad du förändrade sen den förra, exempelvis ”added

spinning gurka". Du kan skapa ett commit med ett visst namn såhär:

```
git add .
git commit -m "Commit message"
```

Se till att din command-lines path är i roten av projektet (något i stil med ...\git-tutorial).

1. Gå in i filen hej.py och ändra "name" till ditt namn. Ändra commitmeddelandet till något lämpligt.
2. Ladda upp dina förändringar på Github! Detta gör du med

```
git push
```

Den kommer att ge dig ett konstigt meddelande med ett kommande. Copy-pastea och kör det. Dubbelkolla sen att dina förändringar är synkade på webben.

3. Katten i bilden ser lite konstigt ut. Testa att byta ut den mot en annan kattbild. Skapa en commit med lämpligt meddelande och pusha sen.
4. Testa att köra filen klocka.py. Oj! Nu skapas en konstig temporärmapp __pycache__. Den vill vi inte ha med i commiten, då blir det svårt att se vad som faktiskt händer. Skapa filen .gitignore och lägg in __pycache__. Testa att commita. Voilá! Pycache tas inte med i commiten.
5. Ånej! Dina fingrar råkar slinta och du raderar kattbilden och kör sen

```
git add .
```

Inga problem! Kör bara

```
git reset --hard
```

och vips är katten tillbaks.

6. Nu ska du få testa på branches. Kör

```
git checkout betterclock
```

Branches låter dig arbeta med flera versioner av projektet parallellt. Här har någon uppgraderat klockan. Vi vill nu få in dessa förändringarna i main-branchen. För att merge:a in i main kör du

```
git checkout main
git merge betterclock
```

Ånej! Merge conflicts? Detta beror på att Git inte kan avgöra vilka förändringar som ska behållas. Det får du avgöra. Gå in i klocka.py. Denna innehåller ganska tydligt avgränsat de två versionerna. Merge:a dessa till en på ett sätt du tycker är lämpligt. Ta bort allt bös så det går att köra filen. Kör sedan

```
git add .
```

```
git commit -m "Merged betterclock"
```

Grattis! Nu har du all kunskap du behöver för att kunna gnälla på lagkamrater över merge conflicts.

Merge conflicts är aldrig kul. Om två branches diverger för mycket ger man bara upp ibland och slänger den mindre viktiga branchen (har hänt i projekt jag varit med i). Ett vanligt sätt att minimera de är att arbeta i olika filer.

Inom industrin är det inte ovanligt att man inte har branches, utan att all kod pushas till main, men är inte ”påsatt” (typ, det finns en if (false) kring koden tills den är redo). Det finns mycket att läsa på om hur man ”borde” använda Git, men för er är det bäst att hålla det simpelt.

Om allt skiter sig kan ni i värsta fall radera foldern och klona det ni har i Github. Notera att ni förlorar allt som inte är pushat då.

Vanliga kommandon

- git init: gör en folder till ett git-repo.
- git add .: lägger till alla filer till nästa commit
- git commit -m <message>: skapar commit med meddelande <message>.
- git diff: visar vad du har förändrat. Bra att kolla innan du commitar, så du inte råkar commit något som pycache.
- git branch <name>: skapar en ny branch som heter <name>
- git checkout <name>: hoppa till branchen <name>.
- git push: ladda upp dina förändringar till Github.
- git pull: ladda ner förändringar från Github (om flera arbetar i samma repo)
- git merge <name>: mergear <name> in i main.
- git remote add origin <git-url>: låt säga att du börjat jobba i en mapp och vill göra det till ett github-repo. Kör först git init, sen lägg till git@github.com... url:en med det här kommandot.
- git bisect: Gör en binärsökning över commits där man kan leta efter vilken commit som orsakade en bugg.
- git cherry-pick: Låter en ta en commit från en annan branch och applicera den på den nuvarande branchen.

Tips and tricks

- Lägg allt på Git. Framtida arbetsgivare blir glada av att se bevis på att du är aktiv på fritiden. En ribba kan vara min github ;). (Den har hjälpt mig med att få en intership).

- Håll git repos små, annars blir allt långsamt. Råka inte commita stora filer, de är svåra att få bort ur historiken.
- Råka inte commita credentials. Läs antingen in de från en gitignore:ad fil eller sätt de som environment variables.
- Om du råkar göra någon av ovanstående går det att ta bort det från Git-historiken, men detta är väldigt farligt. Ta gärna hjälp av någon som kan Git isåfall.
- ChatGPT brukar vara ganska bra på Git. Var självklart lite försiktig så inte allting sprängs.

Utmaning

Gå in på https://github.com/Matistjati/bisect-chall och Klona repositoryt. Testa att köra code.py. I en okänd commit kom en bug in: om du skriver `MATTEKOLLO`, så skriver koden inte ut "Jag älskar programmering". Efter att buggen lades till så finns den i alla commits efter den.

7. Använd git bisect för att hitta den felaktiga commiten.

8. Printa s innan den körs och hitta vilken rad som är den felaktiga. Använd faktumet att du vet den felaktiga raden kod för att hitta den felaktiga commiten.

9. Åh nej! En katt slank in i https://github.com/Matistjati/edit-history. Ändra historiken så att katten aldrig fanns. Historiken av förändringar i code.py efter att katten lades till måste finnas kvar.

10. Orelaterat bonusproblem: skriv kod som printar `Jag gillar kanelbulle!` med hjälp av att stoppa in en massa ints i run. Mattefolket hjälper er gärna att förstå talteorin som gör det möjligt.

11. Kan jag behålla mina repos, men inte räkna de 256 commitsen på gröna väggen i Github? Jag har gärna kvar de, men känns lite bs att de räknas in i mitt commit-antal.

12. Skapa ett nytt githubkonto. Använd faktumet att du kan sätta custom tid på commits för att skriva `MATTEKOLLO` på den "gröna väggen".

6. Bashscripting

20 juli

Bash ("Bourne Again SHell"), är en kommandotolk eller skal (shell) som används i Unix-liknande operativsystem som Linux och macOS. Bash är både ett interaktivt skal, där användare kan skriva kommandon direkt för att interagera med operativsystemet, och ett skript-språk som kan användas för att automatisera olika uppgifter.

> Om du har MacOS eller Linux kan du använda din terminal. Om du har Windows kan du antingen installera WSL eller använda skoldatorerna.

Nedan är en lista med vanliga kommandon/program i bash. Kolla gärna igenom alla även om du är van bash-användare, så kommer du troligen hitta någon användbar funktion eller flagga du inte kände till innan.

Kommandon

Navigering i filsystemet

pwd - Print Working Directory

`pwd` skriver ut den fullständiga sökvägen för den aktuella arbetskatalogen.

```
$ pwd
/home/user/Documents
```

ls - List

`ls` skriver ut alla filer och kataloger i en specifik katalog.

```
$ ls
a.txt  Desktop  Documents  Downloads  Pictures  Videos
```

`ls` har även

```
$ ls -l # visa detaljerad information
total 15144
-rw-r--r-- 1 elias elias 15482880 Jul 19 12:31 a.txt
drwxr-xr-x 2 elias elias  4096 Jul 19 12:33 Desktop
drwxr-xr-x 2 elias elias  4096 Jul 19 12:33 Documents
$ ls -a # visa gömda filer (som börjar med .)
.  ..  .bashrc  .config
$ ls -lh # skriv ut filstorlekar i mänskligt läsbart format
total 15M
-rw-r--r-- 1 elias elias 15M Jul 19 12:31 a.txt
drwxr-xr-x 2 elias elias 4.0K Jul 19 12:33 Desktop
drwxr-xr-x 2 elias elias 4.0K Jul 19 12:33 Documents
$ ls -t # sortera efter ändringsdatum
$ ls -S # sortera efter filstorlek
```

```
$ ls -r # reversera sorteringsordningen (används ofta med '-t' för att visa ä
    ldst först)
$ ls -R # visar innehållet rekursivt för alla underkatalogen
$ ls /home/users/Documents # lista filer i Documents katalogen
```

cd - Change Directory

cd byter den aktuella arbetskatalogen.

```
$ cd /path/to/newdir # absolut sökväg
$ cd new/directory # relativ sökväg
$ cd .. # flytta till överordnade katalogen (t.ex. från
    /home/username/Documents till /home/username/)
$ cd ~ # byt till hemkatalogen (standard /home/username/)
$ cd - # byt till föregående katalogen
```

Man kan använda tab-komplettering för att automatiskt fylla i katalognamn.

Hantera filer och kataloger

touch - Skapa fil

```
$ touch fil.txt # skapa en tom fil
$ touch existerande_fil.txt # uppdatera tidsstämpel på existerande fil
$ touch fil1.txt fil2.txt fil3.txt # skapa flera filer
```

mkdir - Skapa katalog

```
$ mkdir katalog # skapa katalog
$ mkdir /home/username/Documents/projects # skapa 'projects' katalog med hela
    sökvägen
$ mkdir /home/username/Documents/projects -p # skapa 'projects' katalog och
    alla nödvändiga underkataloger
```

rm - Ta bort fil/katalog

```
$ rm fil.txt # ta bort fil.txt
$ rm -f fil.txt # tvinga borttagning av fil utan bekräftelse
$ rm -r katalog/ # ta bort en katalog och deras innehåll rekursivt
$ rm *.txt # * matchar ta bort alla .txt filer i katalogen
```

cp - Kopiera

```
$ cp fil.txt ny_fil.txt # kopiera en fil och ge den ett nytt namn
$ cp fil.txt /home/username/ # kopiera fil.txt in i /home/username/-katalogen
$ cp -r katalog/ /home/username/ # kopiera en hel katalog rekursivt
$ cp fil1.txt fil2.txt fil3.txt /home/username/Documents/ # kopiera tre filer
    in i Documents/ katalogen
```

mv - Flytta/döp om

```
$ mv fil.txt annat_namn.txt # byt namn på fil.txt
$ mv fil.txt /home/username/ # flytta in fil.txt i /home/username/-katalogen
$ mv katalog/ /home/username/ # flytta en hel katalog
$ mv *.txt /home/username/Documents/ # flytta alla .txt filer in i Documents/
    katalogen
```

Visning av filinnehåll

cat - Skriv ut filer till terminalen

```
$ cat fil.txt # skriv ut innehållet i fil.txt i terminalen
$ cat fil1.txt fil2.txt # skriv ut innehållet i fil1.txt och fil2.txt efter
    varandra
$ cat -n fil.txt # skriv ut innehållet i fil1.txt med radnummer
```

less - Öppna och navigera i stora filer

`less` låter dig bläddra genom filens innehåll utan att behöva ladda den helt i minnet. Du kan navigera med piltangenterna, trycka på 'q' för att avsluta, och använda olika andra tangenter för att söka och flytta runt i texten. För att söka kan man trycka '/' och sedan skriva in sökströngen. För att gå till nästa respektive föregående sökträff använder du 'n' och 'N'.

```
$ less fil.txt # öppna fil.txt
$ ls -l /etc | less # skriv ut allt innehåll i /etc och skicka det till less,
    istället för att skriva ut i terminalen
```

head/tail - Skriv ut början/slut av filer till terminalen

`head` och `tail` skriver ut början respektive slutet av en fil.

```
$ head fil.txt # skriv ut de första 10 raderna i fil.txt
$ head -n 5 fil.txt # skriv ut de första 5 raderna i fil.txt
$ head -n 2k fil.txt # skriv ut de första 2000 raderna i fil.txt
$ tail -f fil.txt # skriv ut de första 10 raderna, och skriv sedan
    kontinuerligt ut alla nya rader i filen
```

Textbearbetning

echo - Skriv ut text till terminalen

```
$ echo "Hello World!" # skriver ut "Hello World!" i terminalen
Hello World!
```

grep - Hitta text i filer

```
$ grep "error" logfile.txt # visar alla rader som innehåller strängen "error"
$ grep -E "[0-9]{3}-[0-9]{3} [0-9]{2} [0-9]{2}" contacts.csv # visar alla
    rader som matchar telefonnummer-regexen
```

```
$ grep -r "TODO" project/ # visar alla rader som har strängen "TODO" rekursivt
    i alla filer i project/ katalogen.
```

awk - Analysera och printa filer

`awk` är ett kraftigt textbehandlingsprogram som används för att manipulera och analysera textfiler med tabelldata.

```
$ cat data.txt # filen data.txt innehåller en tabell med id, namn och antal
    kvällsmackor respektive person har ätit på kollot
123 Fredrik 45
124 Elias  60
125 Tobias 50
$ awk '{print $2}' data.txt # skriver ut andra kolumnen
Fredrik
Elias
Tobias
$ awk '{sum += $3} END {print "Average:", sum/NR}' data.txt # skriver ut
    genomsnittliga antalet kvällsmackor per person. Kolla upp vad 'NR' betyder.
Average: 51.6667
$ awk '$1 >= 124 {print $2}' # skriv ut alla namn med id >= 124 (filtrera för
    rader)
elias
tobias
$ awk '/elias/ {print $1}' # skriv ut id för alla rader som har 'elias' i sig
124
```

`awk` är otroligt kraftfullt och mycket värt att söka upp mer information om man är intresserad.

sed - Redigera filer

`sed` (Stream EDitor) kan användas för att ersätta, redara eller stoppa in text

```
$ cat data.txt
När man ska avgöra vilken bokstav som är bra, till och med bäst.
Så är det ganska vanligt att man väljer ä.
Men det är bra,
om många rader innehåller bra ord.
$ sed 's/ä/ae/' data.txt # ersätt första instansen av 'ä' på varje rad med 'ae'
Naer man ska avgöra vilken bokstav som är bra, till och med bäst.
Så aer det ganska vanligt att man väljer ä.
Men det aer bra,
om många rader innehåller bra ord.
$ sed 's/ä/ae/g' data.txt # ersätt alla instanser av 'ä' på varje rad med 'ae'
Naer man ska avgöra vilken bokstav som aer bra, till och med baest.
Så aer det ganska vanligt att man vaeljer ae.
Men det aer bra,
om många rader innehåller bra ord.
$ sed '/väljer/d' data.txt # ta bort alla rader som innehåller ordet 'välja'
När man ska avgöra vilken bokstav som är bra, till och med bäst.
Men det är bra,
```

```
om många rader innehåller bra ord.
$ sed '/bra/a\Här stod det bra' data.txt # infoga en ny rad med texten 'Här
    stod det bra' efter varje rad som innhåller ordet 'bra'
När man ska avgöra vilken bokstav som är bra, till och med bäst.
Här stod det bra
Så är det ganska vanligt att man väljer ä.
Men det är bra,
Här stod det bra
om många rader innehåller bra ord.
Här stod det bra
```

Filhantering och informationsvisning

find - Hitta filer

```
$ find / -name example.txt # skriv ut alla filer med namnet 'example.txt' i
    hela filsystemet
$ find /var/log -type f -mtime +7 # hitta alla filer i /var/log som är äldre ä
    n 7 dagar
$ find /var/log -type f -name "*.log" -exec rm {} \; # hitta alla .log filer
    och radera de
```

I `-exec` ersätts '{}' med filnamnet och '\;' anger slutet på kommandot som ska köras.

du - Disk Usage

```
$ du # skriver ut totala utrymmet allt i den aktiva katalogen tar, rekursivt
$ du -h # -h skriver ut i ett mänskligt läsbart format
$ du -ah # -a inkluderar filer i listan över hur mycket utrymme det tar
$ du -ah --max-depth=1 . # skriver ut utrymmesinformation för allt i den
    aktiva katalogen, ej rekursivt
8.0K  ./Downloads
15M   ./a.txt
8.0K  ./Videos
4.0K  ./Desktop
4.0K  ./data.txt
0  ./.bashrc
4.0K  ./aoesnuh
4.0K  ./.config
0  ./fil.txt
4.0K  ./Documents
4.0K  ./Pictures
15M   .
```

df - Diskanvändning

```
$ df -h # skriver ut diskanvändning i mänskligt läsbart format
Filesystem     Size  Used Avail Use% Mounted on
dev            7.6G     0  7.6G   0% /dev
run            7.7G  1.5M  7.7G   1% /run
/dev/nvme0n1p5 121G 102G   14G  89% /
```

```
tmpfs          1.6G  128K  1.6G   1% /run/user/1000
```

stat - Visa statistik

```
$ stat fil.txt # skriv ut detaljerad information om fil.txt (fungerar även för
    kataloger)
  File: fil.txt
  Size: 0          Blocks: 0        IO Block: 4096  regular empty file
Device: 259,5 Inode: 5465905   Links: 1
Access: (0644/-rw-r--r--) Uid: ( 1000/ elias) Gid: ( 1000/ elias)
Access: 2024-07-19 13:15:01.023595013 +0200
Modify: 2024-07-19 13:15:01.023595013 +0200
Change: 2024-07-19 13:15:01.023595013 +0200
 Birth: 2024-07-19 13:15:01.023595013 +0200
$ stat --printf="Access: %x\nModify: %y\nChange: %z\n" fil.txt # skriv endast
    ut Access, Modify och Change tidsstämplar med --printf valet
Access: 2024-07-19 13:15:01.023595013 +0200
Modify: 2024-07-19 13:15:01.023595013 +0200
Change: 2024-07-19 13:15:01.023595013 +0200
```

Skripting

Skripting i bash är en enkelt och roligt sätt att automatisera saker på datorn. Man kan kombinera alla kommandon ovan, samt egna python-filer eller andra egenskrivna program. Påbörja din bash-skript resa genom att skapa en `.sh`-fil. Inled filen med en Shebang, `#!/bin/bash`, som informerar bash vilken interpertor som ska tolka filen. Man kan även ha en Shebang i python-filer (`#!/bin/python3`).

Grundläggande syntax

```
#!/bin/bash

# här är en kommentar, inleds med # precis som i python

my_var = "Hello World!" # skapa en variabel
echo $my_var # skriv ut variabelns innehåll

# Använd enkla (' ') eller dubbla ( " " ) citattecken för strängar. Dubbla
    citattecken tillåter variabelersättning.
single_quote_str='Detta är en sträng'
double_quote_str="Denna sträng innehåller en variabel: $my_var"

# If-sats
if [ $my_var == "Hello, World!" ]; then
    echo "Villkoret är uppfyllt."
else
    echo "Villkoret är inte uppfyllt."
fi
```

```
# For-loop
for i in {1..5}; do
    echo "Nummer $i"
done

# While-loop
count=1
while [ $count -le 5 ]; do # medans $count är <= 5
    echo "Räkna till $count"
    ((count++))
done

# Case-sats
case $1 in
    start)
        echo "Startar..."
        ;;
    stop)
        echo "Stoppar..."
        ;;
    *)
        echo "Ogiltigt kommando."
        ;;
esac

# Definitera med nyckelordet 'function' följt av namnet
function say_hello {
    echo "Hej, $1!" # skriv ut första parametern som ges till funktionen
}

say_hello "Användare"  # Anroppa funktionen med ett argument
```

Körning av skript

Nästa steg är att göra filen exekverbar. Detta görs med `chmod` kommandot, som ändrar på filens åtkomst behörigheter och speciella flaggor (historiskt modes, därav CHange MODes).

```
$ chmod +x skript.sh
```

Kör sedan filen med

```
$ ./skript.sh
```

I/O Omledning

>, >> (omdirigera output)

Tecknen > och >> används i Bash för att omdirigera output till en fil.

- Omdirigera output till en fil med >. Tecknet > används för att omdirigera output till en fil, vilket skriver över innehållet i filen om den redan existerar.

```
echo "Hej världen!" > output.txt
```

Detta kommando skriver strängen `"Hej världen!"` till filen `output.txt`. Om `output.txt` redan existerar, kommer dess innehåll att skrivas över.

- Append output till en fil med >>. Tecknet >> används för att lägga till output till slutet av en fil (append), utan att skriva över befintligt innehåll.

```
echo "Ytterligare rad" >> output.txt
```

Detta kommando lägger till strängen `"Ytterligare rad"` till slutet av filen `output.txt`. Om filen inte existerar, skapas den.

- Omdirigera output från ett kommando. Du kan också omdirigera output från olika kommandon och program. Här är ett exempel med `ls` som listar filer och mappar i en katalog.

```
ls -l > directory_listing.txt
```

Detta kommando omdirigerar den detaljerade listningen av katalogens innehåll till filen `directory_listing.txt`, skriver över filen om den redan existerar.

```
ls -l >> directory_listing.txt
```

Detta kommando lägger till den detaljerade listningen av katalogens innehåll till slutet av filen `directory_listing.txt`, utan att skriva över dess befintliga innehåll.

| (pipes, för att kedja ihop kommandon)

Pipes (|) i Bash används för att skicka utdata från ett kommando som indata till ett annat kommando.

```
$ grep "pattern" fil.txt | wc -l # filtrera ut rader som innehåller ett
    "pattern" och räkna dessa
$ sort fil.txt | uniq # sortera fil.txt och ta bort dubletter
$ du -ah /sökväg/till/katalog | sort -rh | head -n 10 # visa storleken för
    allt i /sökväg/till/katalog, sortera det med human-readable storlekar (-h)
    och reverse (-r) och skriv ut de 10 största filerna
```

tee

`tee` används för att läsa från standard input och sedan skriva resultatet både till standard output och till en eller flera filer.

```
$ ls -l | tee output.txt # skriv ut och spara resultatet till en fil
$ echo "Ny rad med data" | tee -a output.txt # lägg till (appendera)
    resultatet av ett kommando till en befintlig fil istället för att ersätta
    allt
echo "Data som ska sparas i flera filer" | tee file1.txt file2.txt file3.txt #
    skriv till flera filer samtidigt
```

Detta kan vara användbart om man vill spara data i en lång kedja av pipes, men också vill spara datan i mellanstegen, alternativt skapa en logfil men också behålla terminaloutput.

Cron

Cron är ett kraftigt verktyg för att schemalägga skript på datorn. Man kan enkelt köra skripter eller program var 5:e minut, den 3:e varje månad, varje måndag, m.m. Den konfigureras genom antingen `crontab -e` eller den globala `/etc/crontab`. Ett exempel på crontab:en som kör alla matcher på `mkbot.eliaslundell.se` är nedan:

Manualler och resurser

Alla kommandon har dokumentation som man kan finna i manualen. För att få fram manualen för ett visst kommando kör du `man [kommando]`. Denna är ofta väldigt bra.

```
# Example of job definition:
# .---------------- minute (0 - 59)
# |  .------------- hour (0 - 23)
# |  |  .---------- day of month (1 - 31)
# |  |  |  .------- month (1 - 12) OR jan,feb,mar,apr ...
# |  |  |  |  .---- day of week (0 - 6) (Sunday=0 or 7) OR
# |  |  |  |  |              sun,mon,tue,wed,thu,fri,sat
# *  *  *  *  * user-name command to be executed
*/5 *  * * *  root   /bin/bash -c "cd /root/mk-botchallenge && { time python3
    run_game.py ; } > /root/mk-botchallenge/log_run_game.log 2>&1"
```

Övningar

OBS! Det finns väldigt många sätt att göra saker i bash. Varken kommandona eller flaggorna och exemplen med kommandona ovan är på något vis komplett. Google:a och sök efter fler och effektivare lösningar på problemen nedan.

1. a) Skapa en katalog som heter programmeringskollo/ i din hemkatalog (där `username` är din användare på datorn). På linux är det (/home/username/), men på andra OS heter det lite annat.

   b) Skapa hela sökvägen `/home/username/programmeringskollo/programmering/church/dag6` med endast kommando.

   c) Skapa 10 tomma .txt filer i din nyligen skapta katalog ovan (kan du använda `{1..10}` för att underlätta på något sätt?)

   d) Skapa en till textfil med texten "Välkommen till programmeringskollo!".

2. Skapa ett skript som tar användarens favoritsiffra som en parameter och sedan räknar från favoritsiffran till $2 \times$ favoritsiffran, en siffra per sekund.

```
$ ./your_script.sh 23
23
# <1 sekunds paus>
```

```
24
# <1 sekunds paus>
25
# <1 sekunds paus>
26
.
.
.
46
```

3. — **Extra.** Sök upp `getopts` och använd detta för att göra det tydligare att man ska skriva in sin favoritsiffra i skriptet ovan.

> Ladda ner `files.zip` från Discord och extrahera alla filer.

4. a) Hur många bilder hittar du? Vilken bild är störst? Hur stor är den? (hint: den är större än 1mb)

   b) Hur många gånger förekommer ”error” i logfilen?

   c) Ersätt alla förekomster av ”error” med ”warning” och alla förekomster av ”warning” med ”error” i spexflix.log.

   d) Skriv ett skript/funktion som kan döpa om alla ”codenamesNN.png” i ordning efter storlek. Så att ”codenames1.png” är den minsta filen, ”codenames2.png” den näst minsta, etc

   e) Använd `awk` och `data/deltagare.tsv` (läser `awk` in datan rätt? Kolumnerna är separerade med tab ”`\t`”) (fritt att använda `sort, sed, m.f.` också) för att ta reda på:

      1) medelårskursen på Matte-/Programmeringskollo?
      2) hur många det är i varje mattegrupp?
      3) vilka rum som har 3 personer i sig?
      4) könsfördelningen på Matte-/Programmeringskollo?
      5) medellängden på alla deltagares namn på Matte-/Programmeringskollo?

   f) Skapa en fil med `mattedeltagare.tsv` och en med `programmeringsdeltagare.tsv`.

   g) Hitta en url för en unlisted video i `spexflix.log` och tjuvtitta på ett Kårspex.

5. Skapa ett skript som frågar efter användarens namn och skriver ut användbar information, t.ex.

```
$ ./mattekollo_info.sh Elias
Du bor i rum __ tillsammans med __ (och __).
Du är med i patrull __ som har __ antal deltagare.
Fun fact: På mattekollo är det __ personer som heter samma sak som du.
```

7. Fina bilder med shadertoy

21:a juli

Dator

Disclaimer: om du lätt blir illamående/åksjuk eller har epilepsi eller dylikt kanske den här lektionen inte är för dig. Prata med Joshua isåfall.

Gå in på shadertoy.com. För att avgöra om det bättre att använda din egen eller Liu:s dator, ta den med bäst fps på följande shader: **Rainforest** (sök efter den högst uppe till vänster). Även om du använder Lius dator sparas det du gjort, eftersom all kod finns på webben, kopplat till ett konto.

Skapa ett konto och tryck sen på new.

Also: den som skapar den finaste bilden vinner ett litet pris.

Vad gör koden

Funktionen mainImage körs en gång för varje pixel för att bestämma färgen på den pixeln. Färgvärden specificeras i RGBA, men istället för 0-255 är det 0-1 med 32-bitars flyttal.

```
void mainImage( out vec4 fragColor, in vec2 fragCoord )
{
    fragColor = vec4(vec3(0.4,0.0,0.8),1.0);
}
```

1. Testa att ändra färgen. Försök få en förståelse för hur du skapar olika färger.

Som input till vår kod får vi koordinaten av pixeln. Koordinatsystemet är [0, bredd på rutan) $\times$ [0, höjd på rutan]. Man vill alltid arbeta med normaliserade koordinater, dvs skalar vi om de till [0,1] $\times$ [0,1]. Detta är vad uv är. Test köra följande kod och testa både x och y.

```
void mainImage( out vec4 fragColor, in vec2 fragCoord )
{
    vec2 uv = fragCoord/iResolution.xy;

    fragColor = vec4(vec3(uv.y),1.0);
}
```

$(0,0)$ är alltså längst nere till vänster.

Mönster 1: OR

Vi kommer att färglägga varje pixel som en funktion av binary OR av positionen.

```
void mainImage( out vec4 fragColor, in vec2 fragCoord )
{
    vec2 uv = fragCoord/iResolution.xy;
    uv += iTime/10.;
    uv *= 1000.;
    float or = float(int(uv.x)|int(uv.y));

    fragColor = vec4(vec3(cos(or)),1.0);
}
```

2. Lek runt med koden! Testa att sätta y-komponenten av färgen till xor av koordinaten och z till and av koordinaterna.

Testa gärna att leka runt ännu mer! Ta typ cos(or*iTime/10.) som färg (epilepsivarning).

Vad shadern visar är att man kan skapa extremt intressanta och invecklade mönster med väldigt lite kod. Vi kommer att fortsätta på det mönstret med Mandelbrotsmängden.

Mönster 2: mandelbrot

Mandelbrotsmängden är en mängd komplexa tal som beter sig som en fraktal. Att vara en fraktal betyder att den har komplexa, oregelbundna mönster hur långt man än zoomar in (calculus' arch nemesis). På grund av hur fina det är finns det som folk som använder Mandelbrotsmängden som ett argument för att gud existerar.

För att avgöra vare sig en viss punkt C tillhör mängden eller inte, så itererar vi följande formel. Låt $Z_0 = (0,0)$.

$$Z_{n+1} = Z_n^2 + C$$

Och avgöra om denna någon gång blir godtyckligt stor. Det visar sig att om $|Z| \geq 2$, så garanteras den fly. Så hur implementerar man faktiskt allt det här? I praktiken implementerar man komplexa tal som en vektor med två element, och gör matte som om det var ett komplext tal. Då får vi

$$Z = a + i \cdot b$$

$$Z^2 = (a+ib)\cdot(a+ib) =$$

$$a^2 + iab + iab + (ib)^2 =$$

$$a^2 + 2iab - b^2 =$$

$$a^2 - b^2 + i(2ab)$$

Alltså implementeras Z^2 med `Z=vec2(Z.x*Z.x-Z.y*Z.y, 2*Z.x*Z.y)`. Ifall inget av det kändes logiskt, här har du kod:

```
void mainImage( out vec4 fragColor, in vec2 fragCoord )
{
    vec2 c = fragCoord/iResolution.xy;
    c = c * 2. - 1.; // Remap to [-1,-1] x [-1,-1]
    // correct for different aspect ratios
    c.x*=iResolution.x/iResolution.y;
    vec2 Z = vec2(0., 0.);

     int iterations = 0;
     for (; iterations < 64; iterations++)
     {
         Z = vec2(Z.x*Z.x-Z.y*Z.y, 2.*Z.x*Z.y) + c;
         if (length(Z)>2.) break;
     }

    fragColor = vec4(vec3(iterations<64),1.0);
}
```

Du kanske stör dig lite på hur ofta jag använder 2. istället för 2. Detta beror på att GLSL är jättekräset och låter dig inte omvandla int till float automatiskt. 2. är en förkortning för 2.0.

3. Just nu färglägger vi mandelbrotsmängden med svart/vitt. Kan vi skapa gråskala på något sätt?

4. Gråskala är tråkigt! Fixa färg som en funktion av antalet iterationer det tar för punkten att fly. Tips kan vara att testa sin och cos.

5. Jag lovade ju att det är spännande att zooma in. Zooma in med center $(-0.748974, -0.0635503)$. Du kommer kanske behöva köra lite mer än 64 iterations när du zoomat in en bit.

6. Stora områden av bilden blir vit (punkter som inte flyr). Kan du färglägga den inre delen på nåt kul sätt? (Tankar: alla där inne hamnar i cykler av olika längd. Försök uppskatta cykellängden för varje punkt? Vinkel av cykeln? Genomsnittlig längd? Nåt helt annat?)

(Optional) styra positionen med WASD

Det är tråkigt att bara se fraktalen på ett ställe. Tänk om du kunde styra position och zooma in med tangentbordet.

Det finns inget superlätt sätt att göra det. Först måste vi kunna läsa in om en knapp är nedtryckt. Lät säga att vi vill veta om W, med ASCII-kod 87 är nedtryckt. Då trycker vi på iChannel1 och sätter den till keyboard. Sen kör vi

```
texelFetch(iChannel1, ivec2(87,0),0).x;
```

Detta funkar, men ofta innebär en knapptryckning att vi flyttar vår position lite. Men hur kan vi flytta en position lite om vi inte har en variabel som persisterar över flera frames? För att ha detta måste vi använda en buffer. Sätt iChannel0 till buffer A. Tryck

sedan + högst uppe till vänster i kodeditorn och tryck buffer A. Lägg sedan in följande kod.

```
const int W = 87;
const int A = 65;
const int S = 83;
const int D = 68;

void mainImage( out vec4 p, in vec2 fragCoord )
{
    if (fragCoord==vec2(0.5))
    {
        p = texture(iChannel0,vec2(0));
        p.y+=texelFetch(iChannel1, ivec2(W,0),0).x*iTimeDelta*2.;
        p.x-=texelFetch(iChannel1, ivec2(A,0),0).x*iTimeDelta*2.;
        p.y-=texelFetch(iChannel1, ivec2(S,0),0).x*iTimeDelta*2.;
        p.x+=texelFetch(iChannel1, ivec2(D,0),0).x*iTimeDelta*2.;
    }
}
```

Notera att vi gångrar med iTimeDelta. Detta är så att man inte åker dubbelt så snabbt om man har 120fps istället för 60.

Du kan nu läsa in positionen i mainImage med

```
vec4 p = texture(iChannel0,vec2(0.));
```

p.xy kommer vara positionen.

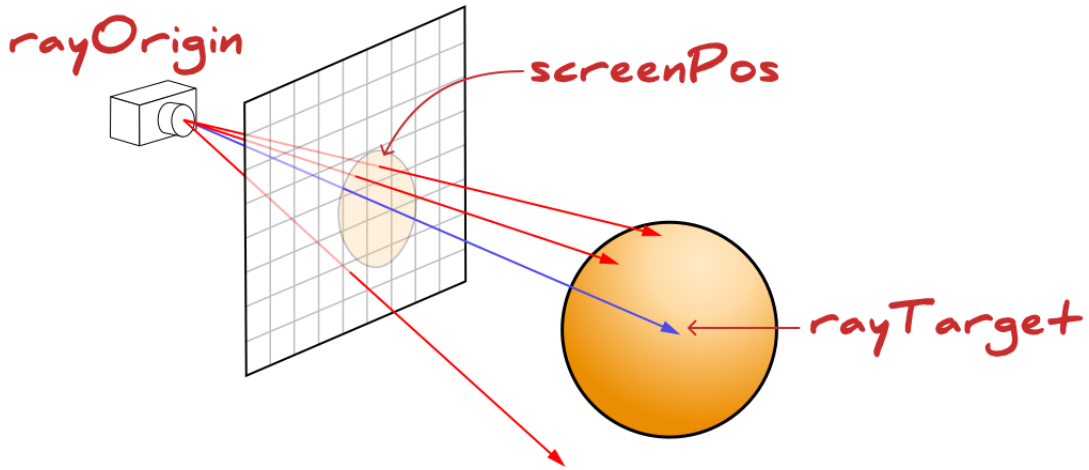
7. Använd positionen för att kunna röra dig i mandelbrotsmängden.

8. Implementera zoom. Notera att flytta är en additions-operation, medan zoom är multiplikativ. Börja utan timedelta.

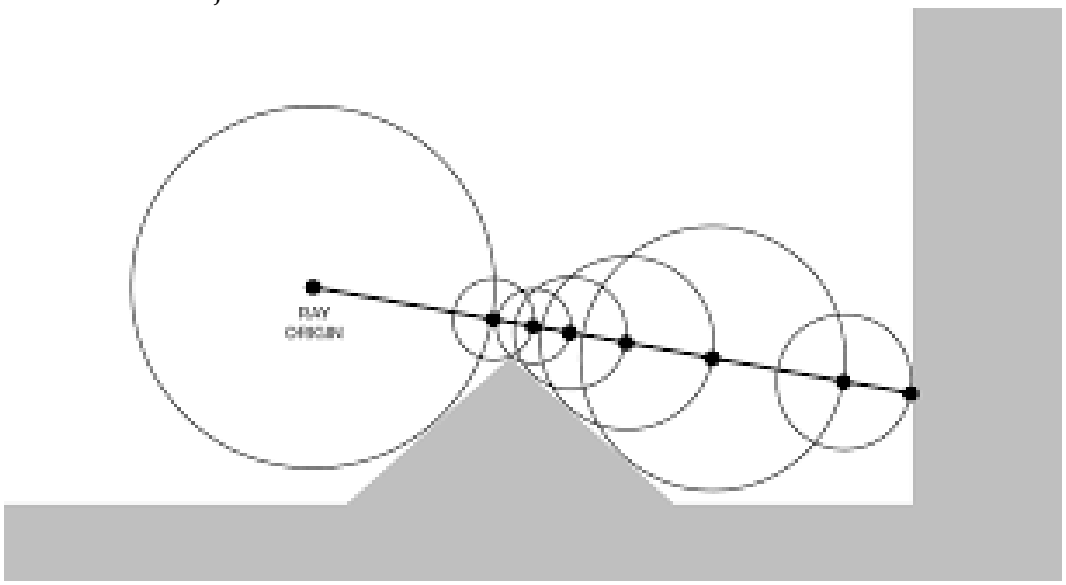
9. När du har zoom och rörelse, leta efter ett coolt ställe och visa för Joshua! Tips: höj antalet iterationer. Om du kan hitta ett område med en klass mönster han inte sett innan får du ett hemligt pris (inte jättestort). (Snälla var inte jobbig med den här).

3D raymarching

Vi kommer att implementera en form av raytracing som kallas raymarching. I denna skjuter vi en stråle från kameran för varje pixel.



Vad som skiljer raymarching från raytracing är att vi inte kommer beräkna exakt vart strålen skär objektet. Istället kommer vi att använda en funktion som ger avståndet mellan en viss punkt och objektet. Vi vet att vi kan stega åtminstone detta avståndet utan råka åka in i vårt objekt.



Vi kommer att börja med en sfär. I detta fallet är det trivialt- det är helt enkelt avstånd från mitten minus radien. Kul nog är sfär kanske det lättaste objektet inom raymarching, men bland de svårare med triangelbaserad rendering.

Koden undertill är lite lång, men inte vara för svår att förstå.

```
// Distance to sphere centered at sphere_center
float SphereDistance(vec3 p, vec3 sphere_center, float radius)
{
   return length(p-sphere_center) - radius;
}

// Combine multiple distances
float SceneDistance(vec3 p)
{
    return SphereDistance(p, vec3(0.), 1.);
}

// We have a line of the form ro+rd*t
// Does it hit the sphere?
vec3 RayTrace(vec3 ro, vec3 rd)
{

    float t = 0.0; // total distance of ray so far

    for (int i = 0; i < 100; i++)
    {
        vec3 position = ro + rd * t;
      float safe_dist = SceneDistance(position);

        // We've hit the sphere
        if (safe_dist < 0.001)
        {
            vec3 light = vec3(t)/2.;
            return light;
        }
        // Move forward by the safe distance
        t += safe_dist;

        if (t > 4.0) // we have passed the sphere
        {
            return vec3(0.0, 0.0, 0.0);
        }
    }

    // Backup if the loop somehow exits
    return vec3(0.0, 0.0, 0.0);
}



void mainImage( out vec4 fragColor, in vec2 fragCoord)
{
    vec2 uv = fragCoord/iResolution.xy * 2.0 - 1.0;
    uv.x *= iResolution.x / iResolution.y;
    // Rescale our coordinates
```

```
    vec3 origin = vec3(.5, 0.0, -2);
    vec3 direction = vec3(uv.xy, 1);
    vec3 color = RayTrace(origin, direction);

    fragColor = vec4(color.xyz, 1.0);
}
```

10. Förstå koden

11. Lek runt med funktionen SceneDistance. Ta min av avståndet till två sfärer. Max av avståndet?

12. Implementera någon av https://iquilezles.org/articles/distfunctions/.

13. Stoppa in en mer spännande distance estimator. Nämligen distance estimatorn till en fraktal! http://blog.hvidtfeldts.net/index.php/2011/09/distance-estimated-3d-fractals-v-the-mandelbulb-different-de-approximations/ (den första kodsnutten). (Läs gärna fler av bloggarna i den serien, de är väldigt bra).

14. Färglägg fraktalen finare, fixa en finare vinkel och kameraposition.

15. Implementera ljus. Du kan uppskatta ytnormalen med hjälp av http://blog.hvidtfeldts.net/in estimated-3d-fractals-ii-lighting-and-coloring/. Sen kan du använda Phongs ljusmodel. Sök **Phong** i shadertoy för att stjäla andras kod.

16. Fixa så att man kan röra sig runt i bilden. Utgå ifrån 2D-lösningen som beskrevs innan.

17. Avancerat: gör så att man kan titta runt och åker i riktningen man tittar.

8. Algoritmer I. Segmentträd

21:e juli

Summor i arrayer: långsam

Låt säga att du får en array med N heltal $A_0, A_1, \ldots, A_{N-1}$. Du vill svara på: givet L, R, hitta summan av $A_L, A_{L+1}, \ldots, A_R$. Med andra ord, beräkna

$$\sum_{L}^{R} A_i$$

1. Lös första subtasken ($N, Q \leq 1000$).
https://judge.progolymp.se/contests/rangesum/problems/rangesum1/sv.

Hur läser man input och skriver output? Standard I/O (samma som command line).

```
n, q = [int(i) for i in input().split()]
nums = [int(i) for i in input().split()]

for i in range(q):
    l, r = [int(i) for i in input().split()]
    s = 0
    # TODO: beräkna summan av nums[l], nums[l+1],...,nums[r]
    print(s)
```

Hur många operationer gör den här koden ungefär? I värsta fall innehåller varje fråga alla element i arrayen, alltså gör vi $O(N)$ operationer. Eftersom vi gör det Q gånger tar det $O(NQ)$ sammanalgt. Detta är snabbt nog för $N = Q = 1000$, men för $N = Q = 2 \cdot 10^5$ blir det $2 \cdot 10^5 \cdot 2 \cdot 10^5 = 4 \cdot 10^{10}$ operationer. En modern dator kan göra ca 10^9 operationer på en sekund, så detta är för långsamt för full poäng.

Kan vi göra lite extra arbete innan för att svara på varje fråga snabbare? Ja!

Vi kommer beräkna en ny array "pref". Låt oss kalla input-listan för nums. pref byggs upp som följande:

- pref[0]=$nums[0]$
- pref[1]=$nums[0] + nums[1]$
- pref[2]=$nums[0] + nums[1] + nums[2]$
- pref[3]=$nums[0] + nums[1] + nums[2] + nums[3]$
- ...

2. Vänta med att läsa vidare. Försök att komma på hur du kan använda pref för att beräkna summan snabbare.

Låt säga att vi vill hitta svaret för $L = 2$ och $R = 4$. Vår önskade summa är då $nums[2] + nums[3] + nums[4]$. Låt oss betrakta $pref[4]$. $pref[4] = nums[0] + nums[1] + nums[2] + nums[3] + nums[4]$

Alltså är $pref[4]$ nästan vad vi vill ha, den enda skillnaden är att $pref[4]$ också innehåller $nums[0] + nums[1]$. Om vi kan subtrahera bort dessa har vi alltså vår önskade summa! Men hur kan vi snabbt beräkna det vi ska subtrahera? Jo, detta är alltid ett annat prefix, $pref[1] = pref[0] + pref[1]$. Alltså, svaret för $L = 2, R = 4$ är $pref[4] - pref[1]$. I allmänhet gäller det att svaret för L, R är $pref[R] - pref[L-1]$.

Tänk också på vad som ska hända om $L = 0$. Tänk också på att summan kan överstiga 2^{31}, alltså behöver du använda long long i språk som C++. Tänk också på att du kan beräkna pref snabbt, eftersom $pref[i] = nums[i] + pref[i-1]$.

3. Lös hela problemet.
https://judge.progolymp.se/contests/rangesum/problems/rangesum1/sv.

Notera att detta är asymptotiskt optimalt, eftersom vi lägger lika mycket arbete på att läsa in indata som att beräkna saker.

Med uppdateringar

Vi gör det svårare. Nu ska vi hantera

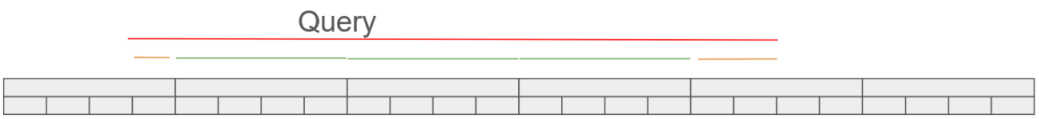
- Skriv ut summan $nums[L] + nums[L+1] + \cdots + nums[R]$ för givna L och R.
- Sätt $nums[k] = v$ för givna k och v.

Detta blir problematiskt för vår förra lösningen, eftersom ett element kan påverka nästintill hela prefixsumman. Så vi kan svara på varje fråga i $O(1)$, men det tar $O(N)$ att bygga om prefixsumman. Vi kanske kan få lite bättre balans mellan de två komplexiteterna?

Låt oss dela in vår lista i block av storlek 4. Vi beräknar summan av nums i varje block. Exempel: övre listan är blocken och den undre är nums.

1+3+2+1=7				4+3+7+1=15				2+9+2+3=16			
1	3	2	1	4	3	7	1	2	9	2	3

Hur kan vi använda denna struktur för att snabba upp frågor? Om vi använder block behöver vi bara gå igenom $\frac{N}{4}$ tal! För att se mer exakt, kolla på figuren nedan:



Låt säga att det röda intervallet är en fråga. Då kan vi svara på alla de gröna delarna med block. Men ändpunkterna av queryn kan inte svaras med block- här får vi naivt loopa. Sammanlagt tar det $\frac{n}{4}$ operationer för att svara på delen innanför block och $2 \cdot 4$ operationer för ändpunkterna. I allmänhet, om vi har block av storlek k tar det $O(\frac{n}{k} + 2k)$ operationer att svara på varje fråga.

4. Hitta det k som minimerar $O(\frac{n}{k} + 2k)$. Dvs, hur stora ska våra block vara för att svara på frågor så snabbt som möjligt? Om det är för svårt för allmänt N, gör det för $N = 10^5$ med valfri metod.

Uppdateringar är triviala: vi uppdaterar elementet i arrayen och blocket som den ligger i. Detta är alltså $O(1)$.

Detta ger ett ganska bra komplexitet, men vi har fortfarande obalans mellan tid av queries: vi har $O(\sqrt{N})$ per query och $O(1)$ per update. $O(\sqrt{N})$ är ofta för långsamt.

Det är även ganska pilligt att implementera detta felfritt, så vi avråder från att göra detta.

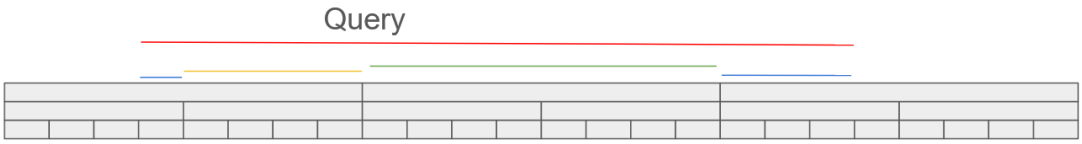
5. — **BONUSPROBLEM 1.** Kan du designa en datastruktur som låter dig svara på queries i $O(1)$ och tar $O(\sqrt{N})$ per uppdatering?

6. — **BONUSPROBLEM 2.** Kan du komma på ett annat sätt att få $\sqrt{N}$-komplexitet? Utgå ifrån lösningen till versionen utan uppdateringar och bygg om strukturen lite då och då.

Segmentträd

7. Är det möjligt att göra queries snababre i utbyte mot långsammare uppdateringar?

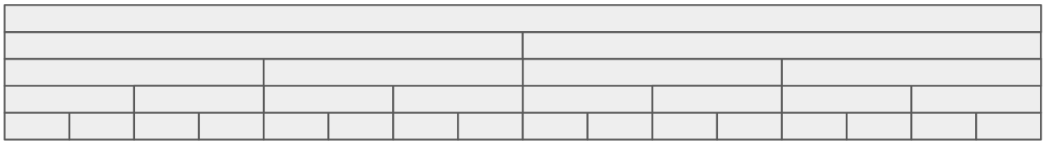
Spoiler: ja!



Den röda är hela queryn. Då delas den upp i 3 delar: den gröna i de största intervallen, de gula i de näst största och då blå i den minsta (ursprunliga arrayen). Varje uppdatering kräver nu 3 operationer, och queries har blivit lite snabbare. Att bestämma optimala lagerstorlekarna har för 3 lager är lite svårare.

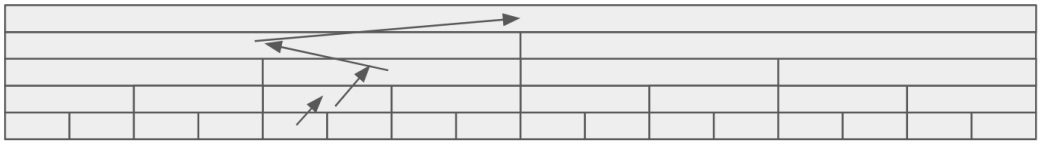
8. — **BONUSPROBLEM 3.** Givet $N = 10^5$, hitta optimala storleken av block 2 och 3.

Vad händer om vi tar detta till sin extrem och delar tills vi inte kan mer? Då får vi ett så-kallat segmentträd! I dessa låter vi första lagret (lagret längst ner) ha block-storlek 1 (ursprungliga arrayen), i lager 2 har varje block storlek 2, nästa lager storlek 4, osv.



Hur många lager kommer vår struktur att ha? Det är antalet gånger man kan dela N med 2 innan det blir 1. Detta är $log_2(N)$. Notera att $log_2(10^5) = 17$.

Hur ser då uppdatering och querying ut i denna struktur? Uppdateringar är simpla: vi startar på lagret längst ned, och uppdaterar summan av alla block som innehåller det elementet. Eftersom det finns $log_2(N)$ lager tar varje uppdatering $O(log_2(N))$ tid. Detta kan se ut som följande:

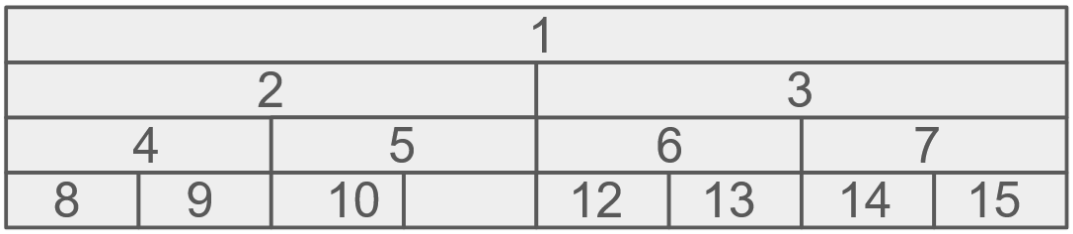


Att svara på queries är precis som innan: använd så stora intervall som möjligt för att täcka queryn. Detta tillåter oss att aldrig behöva ta fler än 2 av en specifik intervall storlek. Alltså tar det $O(log_2(N))$ att svara på en query.

Implementation

Att implementera 2 lager är redan pilligt, 3 lager blir hemskt. Om vi har $log_2(N)$ lager blir det väl jättepilligt? Faktiskt inte, det finns många trick för att förenkla implementationen.

För det första kommer vi tänka på det som ett träd, där varje block har de 2 blocken under den som barn. Dessutom kommer vi ge följande indexing till blocken:



Denna har egenskapen att ett block med index x har vänstra barn $2x$ och högra barn $2x+1$. Så vi kommer förvara alla block i en stor array indexerad med indexing som definierat ovan.

Vi kan även notera att om N inte är en tvåpotens blir det lite konstigt. Därför kommer vi padda nums med nollor tills det blir en tvåpotens genom att gånga N med 2. Hur många block kan det finnas totalt? Om $N = 2^k + 1$ kommer det paddas till $2N + 2 \approx 2N$. Varje lager har hälften så många block som det förra, så vi får $2N + N + N/2 + N/4 + ...$ noder. Det är välkänt att denna summa konvergerar till $4N$, så det är hur stor vår array som håller trädet blir.

Vi kan nu börja kolla på implementationen. Idén är att vi börjar vid roten (block 1) och håller koll på intervallet som vårt block spänner upp i ursprungliga arrayen.

```
n = int(1e5)

# Sätt nums[i]=v och uppdatera resten av trädet
def update(node, l, r, i, v):
    if l==r: # Intervallstorlek av 1, vi är i bottenlagret
        tree[node] = v
    else:
        mid = (l+r)//2
        # Kolla om elementet i bottenlagret ligger till det vänstra eller högra
            blocket undertill
        if i <= mid:
            update(node*2, l, mid, i, v)
        else:
            update(node*2+1, mid+1,r,i,v)
        tree[node] = tree[node*2]+tree[node*2+1]

def update_helper(i, v):
    # Börja på block 1, som spänner [0, n-1]
    update(1, 0, n-1, i, v)

# Vad är summan av nums[ql]+nums[ql+1]+...+nums[qr]
def query(node, l, r, ql, qr):
    if l > qr or r < ql: # Vårt intervall är helt utanför frågeintervallet
        return 0
    if l >= ql and r <= qr: # Vårt intervall är helt innanför frågeintervallet
        return tree[node]
    mid = (l+r)//2
    return query(node*2, l, mid, ql, qr) + query(node*2+1, mid+1, r, ql, qr)

def query_helper(ql, qr):
    return query(1, 0, n-1, ql, qr)
```

9. Försök förstå vad koden gör.

10. update tar "uppenbarligen" $O(log_2(N))$. Vad är komplexiteten av query? Exakt hur många block kan den besöka i värsta fallet?

11. Lös https://judge.progolymp.se/contests/rangesum/problems/rangesum2 med hjälp av koden.

12. https://open.kattis.com/problems/supercomputer

13. https://open.kattis.com/problems/megainversions

14. https://po.kattis.com/problems/skidor

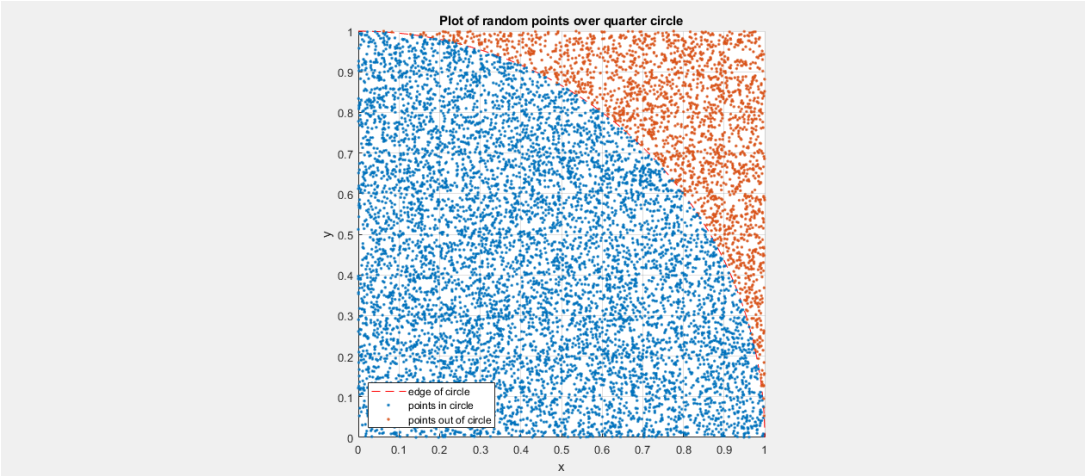
15. https://po.kattis.com/problems/kotid

16. https://po.kattis.com/problems/kotid

9. Parallellprogrammering i C++

22:a juli

För att demonstrera parallellprogrammering kommer vi att uppskatta π. Vi kommer använda följande setup:



Dvs, vi har en kvadrat som spänner $[0,1] \times [0,1]$ och en kvarts-cirkel med center $(0,0)$ och $r = 1$. Om vi slumpar en punkt uniformt slumpmässigt inom $[0,1] \times [0,1]$ är $P($ punkten är innanför cirkeln$) = \frac{area_{cirkel}}{area_{kvadrat}} = \frac{\pi}{4}$. Vi kan uppskatta sannolikheten genom att slumpa punkter, och därmed uppskatta π. Här är ett simpelt enkeltrådat program som gör detta:

```
#include <iostream>
#include <vector>
#include <random>
#include <chrono>
#include <thread>

using namespace std;

long long inside_circle = 0;
long long outside_circle = 0;
long long num_iterations = 1e6;

void monte_carlo_pi()
{
    random_device rd;
    mt19937 gen(rd());
    uniform_real_distribution<> dis(0.0, 1.0);

    long long local_count = 0;
    for (long long i = 0; i < num_iterations; ++i)
    {
        double x = dis(gen);
        double y = dis(gen);
        outside_circle++;
```

```
        if (x * x + y * y <= 1.0)
        {
            inside_circle++;
        }
    }
}

int main()
{
    monte_carlo_pi();
    double pi_estimate = 4.0 * inside_circle / outside_circle;

    cout << "Estimated Pi = " << pi_estimate << endl;

    return 0;
}
```

1. Testa att köra koden. Ungefär hur bra uppskattning får du om du höjer num_iterations?

Testa att höja num_iterations till 10^{11}, kör programmet och kolla i aktivitetshanteraren. På kolumnen "CPU" kommer den att säga ungefär 25%. Varför inte 100%? Det visar sig att nästintill alla moderan CPU:er har flera oberoende kodexekveringsenheter som kallas kärnor (ofta 4st). Ett program kan bara vanligtvis använda en kärna åt gången. Med hjälp av multitrådning kan vi använda flera samtidigt!

Multitrådning kan inte snabba upp alla program, exempelvis är följande kod väldigt troligtvis omöjlig att snabba upp med multitrådning:

```
int x=0;
for (int i = 0; i < int(1e9); i++)
{
    x = (x - 10) ^ 0x12763 << 0x12376;
}
```

Detta beror på att vi måste räkna ut alla x sekventiellt och kan inte använde de *oberoende* kodexekveringsenheterna. Vårt *pi*-program har däremot god chans att vinna prestanda med multitrådning- vi kan starta 4 trådar som alla kör monte carlo-koden och sedan kombinera dessa i slutet!

Ersätt main-funktionen med denna, som istället använder flera trådar:

```
int main()
{
    vector<thread> threads;

    int num_threads = 4;
    // Launch threads
    for (int i = 0; i < num_threads; ++i)
    {
        threads.push_back(thread(monte_carlo_pi));
    }
```

```
    // Wait for threads to finish
    for (auto& t : threads)
    {
        t.join();
    }
    double pi_estimate = 4.0 * inside_circle / outside_circle;

    cout << "Estimated Pi = " << pi_estimate << endl;

    return 0;
}
```

2. Testa att köra koden med samma num_iterations som innan. Är uppskattningen bättre eller sämre? Varför?

3. Jämför att köra med O1 och O2- flaggor till kompilatorn. Varför skiljer sig resultatet så stort?

Det visar sig att vi har fått en så-kallad race condition. Låt oss kolla noggrannare på följande del av koden:

```
for (long long i = 0; i < num_iterations; ++i)
{
    double x = dis(gen);
    double y = dis(gen);
    outside_circle++;
    if (x * x + y * y <= 1.0)
    {
        inside_circle++;
    }
}
```

mer specifikt,

```
outside_circle++;
```

Vad denna egentligen gör på processornivå är:

```
int outside_circle_value = outside_circle
outside_circle = outside_circle_value + 1;
```

Ser ni vad som kan gå fel om flera trådar kör den koden samtidigt? Låt säga att vi har två trådar, tråd A och B. Låt säga att följande händer sekventiellt:

- Tråd A läser och sparar outside_circle_value=10
- Tråd B läser och sparar outside_circle_value=10
- Tråd A skriver outside_circle=11
- Tråd B skriver outside_circle=11

Alltså skedde två stycken ++, men värdet ökade bara med 1 eftersom tråd B använde ett utdaterad värde av outside_circle. Det är lite slumpat om tråd A eller B "hinner" läsa eller skriva först (beror på vilken tråd som operativsystemet prioriterar, vilket är slumpartat). Allt detta leder till att vår kod ger en felaktig uppskattning av π.

Hur kan vi lösa det här? För det första måste vi identifiera den *kritiska sektionen* av vårt program- delen där tråden använder resurser som delas mellan flera trådar. I fallet av vårat program är det

```
outside_circle++;
if (x * x + y * y <= 1.0)
{
    inside_circle++;
}
```

Om vi kan garantera att som mest en tråd exekverar kod innanför det här blocket samtidigt är vårt problem fixat! För att säkerställa detta kommer vi använda så kallade lås. Ett lås kan antingen vara öppet eller stängt. Om tråd A låser låset och tråd B sedan försöker låsa det, så kommer tråd B tvingas vänta tills tråd A låser upp det. Genom att lägga till:

```
lock.lock()
outside_circle++;
if (x * x + y * y <= 1.0)
{
    inside_circle++;
}
lock.unlock()
```

Så har vi eliminerat race conditionen helt. I C++ använder vi mutex som lås. Lägg till följande i toppen

```
#include <mutex>

std::mutex monte_carlo_lock;
```

Och lägg till följande i monte_carlo_pi:

```
monte_carlo_lock.lock();
outside_circle++;
if (x * x + y * y <= 1.0)
{
    inside_circle++;
}
monte_carlo_lock.unlock();
```

Testa att köra programmet. Nu borde det skriva ut en ganska bra uppskattning av π.

Mer (eller mindre?) avancerad synkronisering

Ofta vill man bara använda lås om det är exakt lås man behöver. I fallet ovan var det vi egentligen sökte ett sätt att uppdatera en variabel utan race conditions. I C++ kan det också åstadkommas med hjälp av Atomic.

Ta koden utan locks och lägg till följande i toppen:

```
#include <atomic>

atomic<long long> inside_circle = 0;
atomic<long long> outside_circle = 0;
```

(Och ta bort de gamla inside_circle).

Nu är programmet fortfarande korrekt, samtidigt som du slipper oroa dig om att du hanterar låsen korrekt. I sina extremfall finns det risk att flera trådar väntar på olika lås cirkulärt, vilket kallas en deadlock. Om detta sker är det ofta kört för programmet. Genom att använda abstraktioner som atomics kan vi göra multitrådning foolproof. Vissa språk innehåller också trådsäkra datastrukturer, men C++ saknar dessa.

4. Inte synkronisering: använd multitrådning för att beräkna en prefix sum (det första ni gjorde igår).

Trådning för IO

Tänk dig att du vill hantera nån form av input i ditt program, men du vill inte att programmet pauser medan du väntar på inputen. En sak du kan göra då är att använda dig utav trådning: en tråd som hanterar input, och minst en tråd som gör det arbete som måste göras. På det sättet kan tråden som gör jobbet jobba på för fullt hela tiden. Sen kan tråden som sköter input se till att modifiera state:t av programmet. Så som att t.ex. lägga till jobb i en kö så att jobbet kan göras av en annan tråd.

Samma sak används en hel del i GUI:n. Man kan ha en tråd som hanterar mus- och tangentbordsinput, medans man har en annan tråd som sköter beräkningar som måste göras för att visa upp saker på skärmen varje frame.

Condition variables

Condition variables kan användas för att se till att en tråd i princip kan vakna till liv när nånting har skett. Titta på följande kod och tänk lite på varför den inte är så bra:

```
while(!other_thread_has_sent_signal()){
    this_thread::sleep_for(std::chrono::milliseconds(10));
}
do_work();
```

Det som koden försöker att uppnå är att den kan börja göra jobbet så fort som det är möjligt, men den lägger in en liten sleep så att tråden inte maxar cpu-användningen. Trots det så kommer ganska mycket datorkraft att gå åt, och det kommer även att bli en liten delay från att tråden kan göra jobbet, tills att den faktiskt gör jobbet. Hur kan vi förbättra dett? Jo vi kan använda oss utav `std::condition_variable`. Det finns ett par användbara funktioner som vi kan använda. Bland annat:

- `condition_var.wait`
- `condition_var.notify_one`
- `condition_var.notify_all`

5. Skriv ett program som använder två trådar. En tråd som är en worker tråd, som väntar på en signal från en annan tråd. Och en tråd som väntar på input från användaren. Så fort det finns input så bör den signalera för den andra tråden att det är dags att jobba lite. För detta problem det med att den skriver ut lite text till skärmen för att signalera att den faktiskt har tagit emot jobbet, men i mer verkliga scenarion hade jobbet faktiskt haft betydelse. Ni kan behöva google:a lite för att se exakt hur dessa funktioner används.

6. Skriv en kö klass som är trådsäker. Lägg till operationer som gör att ni kan skriva en worker tråd som väntar på att det ska läggas till någonting i kön, och poppar det så fort det finns tillgängligt.

När ska man använda multitrådning?

Innan du börjar fundera på multitrådning, går det att snabba upp programmets enkeltrådade prestanda? Om du kan byta till en effektivare algoritm kan du ofta se $\approx 100x$ prestanda. Om din algoritm är nästintill optimal, försök optimera konstantfaktorn, vilket kan ge $\approx 2 - 10x$ mer prestanda. Om du har tur kan du SIMD:a för $\approx 4x$ mer prestanda. Detta är väldigt oexakta siffror, men poängen är att multitrådning ofta inte är lättaste sättet att vinna prestanda. Dessutom introducerar det risker: om du inte analyserat race conditions väldigt noga finns risker att ditt program blir felaktigt. Ibland kan det vara en på 1000 att det går fel, så ibland kan det vara jättesvårt att debugga också. Ibland blir det inte ens så mycket snabbare på grund av att trådar spenderar väldigt mycket tid med att vänta på lås.

Men om du redan pressat den enkeltrådade prestandan, eller har en dator med orimligt många kärnor, kör för all del! Desto mer oberoende det är, desto bättre. Ett klassiskt exempel är grafik: att rendera varje pixel är nästintill oberoende alla andra pixlar. Ett annat kan vara matrismultiplikation för AI. Båda dessa är så viktiga applikationer av multitrådning att vi byggt processorer som har tusentals trådar: GPU:er!

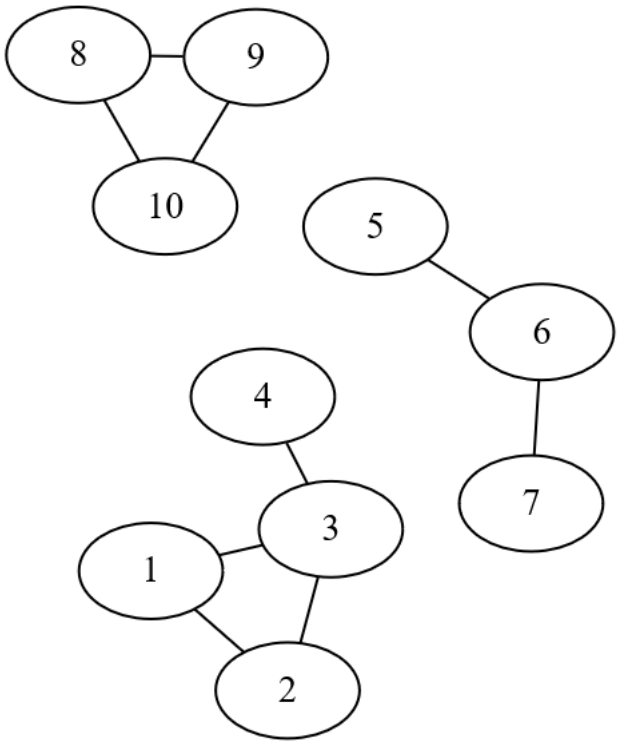
10. Algoritmer II. Nåbarhet och MST

22:e juli

Problem 1

Vi får givet en oriktad graf med 10^5 noder och kanter. Vi får följande sortens frågor:

- Kan vi ta oss från nod A till nod B genom att följa kanter?



Exempel på frågor och svar:

- Kan vi gå från nod 1 till nod 4? Svar: Ja.
- Kan vi gå från nod 1 till nod 5? Svar: Nej.
- Kan vi gå från nod 8 till nod 9? Svar: Ja.

Algoritm 1

Starta en DFS från nod A. Om vi någonsin besöker nod B, returnera att svaret är Ja. Om vi aldrig besöker nod B är svaret Nej.

```python
# number of nodes, edges and queries
n, m, q = [int(i) for i in input().split()]

edges = [[] for i in range(n)]
for i in range(m):
    a, b = [int(i) for i in input().split()]
    edges[a].append(b)
    edges[b].append(a)

def dfs(u, t, vis):
    # u=current node, t=target node, vis=list of visited nodes
    # TODO: complete dfs

for i in range(q):
    a, b = [int(i) for i in input().split()]
    # query: can we go from node a to node b
    vis = [False for j in range(n)]
    print("Yes" if dfs(a, b, vis) else "No")
```

1. Skriv färdigt dfs i koden ovan. Använd följande sample för att testa koden (grafen är samma som i bilden på förra sidan).

Sample:

```
10 9 5
1 2
1 3
2 3
3 4
5 6
6 7
8 9
8 10
9 10
1 2
1 4
1 5
3 5
8 9
```

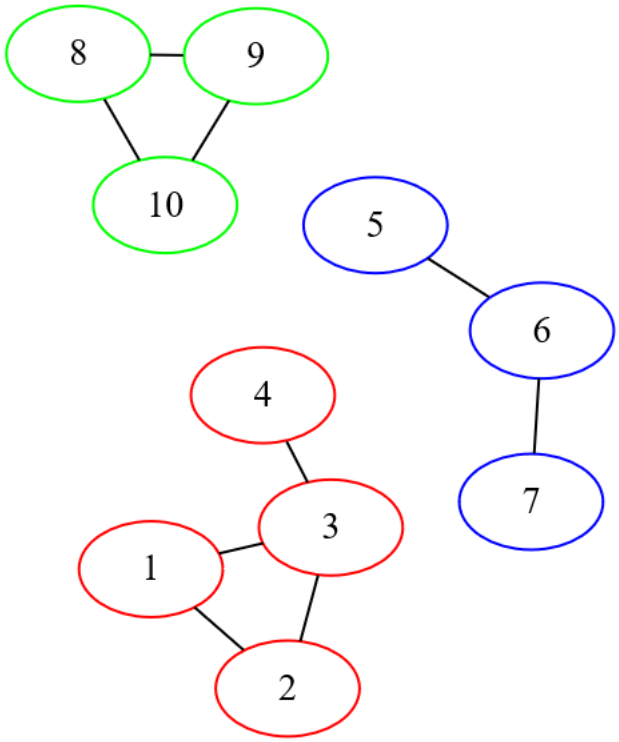
Svar:

```
Ja
Ja
Nej
Nej
Ja
```

Denna algoritmen är relativt långsam- om det finns M kanter och Q frågor kör den i $O(MQ)$.

Algoritm 2

Alla noder som kan nå varandra finns i så-kallade komponenter. Färglägg varje komponent med en unik färg. När vi får en fråga, svara om båda nodernas färg är samma. Om de har samma färg är svaret Ja, annars Nej. Denna algoritmen kör i $O(M+Q)$- mycket snabbare!



Praktiskt sett sker ”färglägger” vi noder genom att varje ”färg"är ett heltal.

2. Modifiera den förra koden så att den använder färgläggningsalgoritmen istället.

Detta är i princip optimalt- vi spenderar ungefär lika mycket tid på att göra arbete som att in indata.

Problem 2

Vi introducerar därmed ett nytt problem. Vi börjar med en tom graf och ska hantera 2 sorters “frågor”:

1. Lägg till en kant mellan nod A och B.
2. Finns det en väg mellan A och B?

(A och B kommer självklart variera mellan de olika frågorna). Om vi tillämpar färläggningsalgoritmen kan vi hantera frågor av typ 2 i $O(1)$ och frågor av typ 1 i $O(M)$ genom att köra om hela färläggningen. Detta känns dock onödigt - bara en väldigt liten del av

vår graf förändas faktiskt. Mer exakt vill vi nu ändra så att de två komponenterna som binder samman nod A och B ska ha samma färg. Alltså räcker det att välja en av de två komponenterna godtyckligt (bara om A och B har olika färg) och ändra dess färg till den andras.

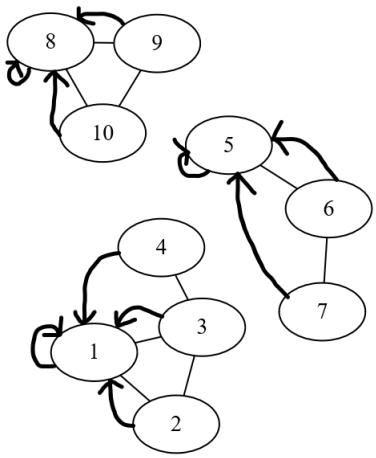
I värsta fallet kan detta fortfarande ta $O(M)$ per uppdatering. Hur? Detta kan ske om vi har en jättestor komponent som vi tillfrågas slå ihop med jättesmå komponenter om och om igen, och den stora färgas om varje gång.

Lite sidospår: ett sätt att lösa detta är att hålla koll på komponentstorlekar och färglägga den mindre komponenten när du ska merga 2 olika komponenter. Det är lite komplicerat varför, men detta orsakar som mest $O(M \log_2(M))$ sammanlagt över alla uppdateringar (även om en individuell uppdatering kan ta $O(M)$). Detta tricket brukar kallas smaller-to-larger merging. Ofta brukar man hålla någon annan struktur med lika många element som komponenten, vilket låter dig effektivt hålla koll på något. Det kan även vara användbart om du gör en DFS och vill hålla koll på information om alla dina barn (typ, hur många noder finns det varje djup). Detta kommer dock inte vara fokuset av dagens lektion.

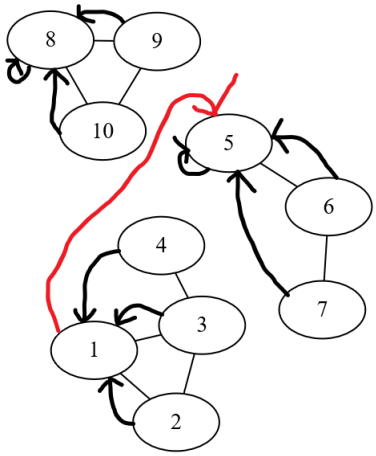
3. — **BONUS.** https://open.kattis.com/problems/almostunionfind

4. — **SUPERSVÅR BONUS.** https://open.kattis.com/problems/agario

Tillbaka till fokus. Vår struktur just nu är extremt lätt att querya, men svår att uppdatera. Vi kommer att ändra den lite: varje komponent kommer att ha en godtycklig "ledare". Istället för att varje nod har en färg, så pekar den på en ledaren. Alla noder som pekar på samma ledare är i samma komponent. Ledaren pekar på sig själv.



För att kolla om två noder är i samma komponent följer vi deras kanter och kollar om de hamnar på samma nod. Hur kan vi slå ihop komponenter? Ta ledaren av ena komponenten och ändra så att den pekar på den andra komponenten.



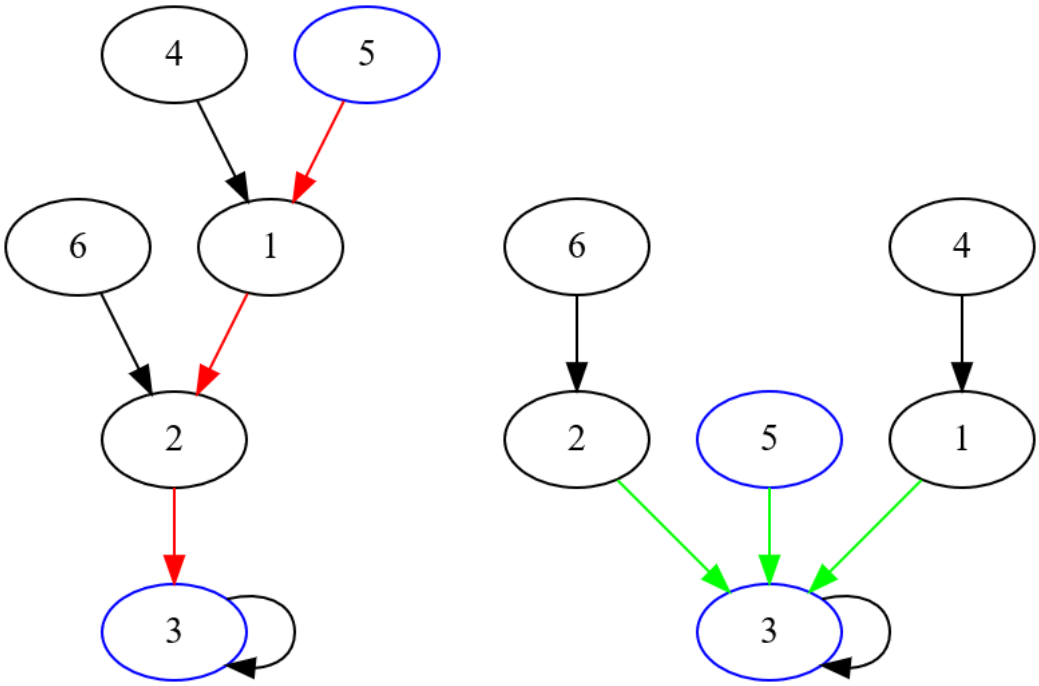
Hur vet vi nu att exempelvis nod 6 och 3 är i samma komponent? Nod 6 hittar ledaren 5. För nod 3 måste vi följa kanter tills vi till slut hamnar vid ledaren 5. Eftersom de har samma ledare är de i samma komponent!

Om vi alltid låter ledaren från den mindre komponenten peka på ledaren i den större av de 2 komponenterna i en merge, så får vi $O(\log_2(M))$ per uppdatering och query (längsta vägen kommer vara av längd $O(\log_2(M))$. Utan den kan det ta $O(M)$ per operation i värsta fallet.

Det finns faktiskt en optimering till som vi kan tillämpa. Kan du hitta den?

Om vi någonsin tillbes hitta ledaren för en viss nod, så kan vi lika gärna ändra förälderpekaren för alla noder på vägen, så att de pekar direkt på ledaren.

Låt oss titta på ett exempel. Anta att vi förfrågas om ledaren för nod 5. Alla kanter på vägen från 5 till 3 är rödmarkerade. I den andra bilden har de röda kanterna flyttats till att peka till nod 3 och är grönmarkerade.



Notera att detta minskar djupet på trädet - alltså blir strukturen bättre varje gång vi använder den. Vad är då komplexiteten? Väldigt snabb: $O(\alpha(N))$ per operation (query eller update). $O(\alpha(N))$ är inversen av den extremt snabbväxande Ackerman-funktionen. Definitionen av $\alpha(N)$ brukar inte vara särskilt exakt, men för $N \leq 10^9$ så är $\alpha(N) \leq 5$. Jag saknar bra notation för att beskriva det N där $\alpha(N) = 6$. $2^{2^{2^{\cdots}}}$ växer för långsamt för att kunna skriva ut talet med en rimlig mängd termer (läs mer: Knuth's pilnotation).

Här är en implementation:

```
# the pointers to the parent
par = [i for i in range(n)]
# component sizes
size = [1 for i in range(n)]

# what is the leader of a?
def find(a):
    if par[a]==a: # a is a leader
        return a
    par[a] = find(par[a]) # update a's leader
    return par[a]

# join node a and b by an edge
def merge(a, b):
    a = find(a)
    b = find(b)
    if a==b:
        return
    if size[a] < size[b]: # Small-to-large merge
        a,b=b,a
    par[b] = a
    size[a] += size[b]
```

Den här datastrukturen kallas för union find, och kan modifieras på olika sätt för att lösa många uppgifter.

5. https://open.kattis.com/problems/unionfind
6. https://open.kattis.com/problems/kastabroar
7. https://open.kattis.com/problems/andvag
8. https://open.kattis.com/problems/firetrucksarered

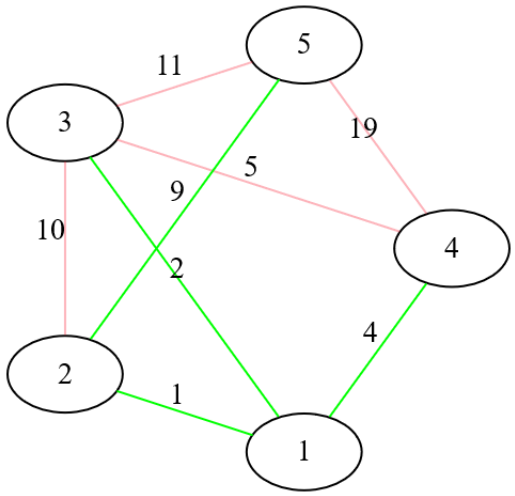
Ett exempel på en cool applikation av union find är att hitta ett minimum spanning tree (MST) av en graf.

Först måste vi definiera vad ett träd är. Om en graf har N noder, $N-1$ kanter och man kan nå varje nod från varje annan nod är det ett träd. En ekvivalent formulering är att man kan nå varje nod från varje annan nod, och det finns inga cykler.

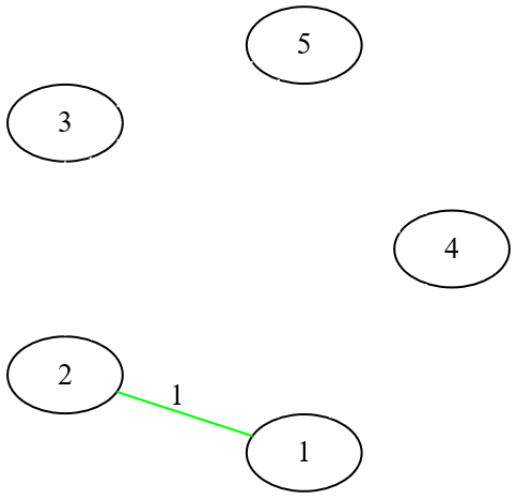
Problemet lyder: Du får givet en graf med N noder och M kanter. Varje kant har en vikt W mellan 1 och 10^9. Välj en delmängd av kanter som utgör ett träd (inga cykler) som innehåller alla N noder, där summan av vikterna av alla kanterna är så liten som möjligt.

Exempel: I följande graf är de gröna kanterna en del av MST:n, och de röda är inte

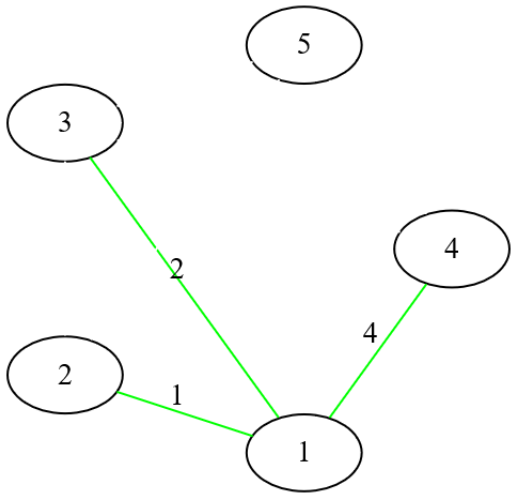
med.



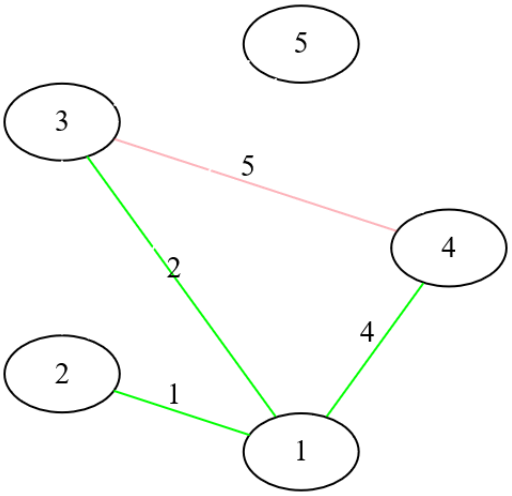
För att lösa problemet kommer vi att använda Kruskals algoritm. I denna algoritm sorterar vi först kanterna på deras vikter, så att de med minst vikt kommer först. Vi kommer att börja med en graf med endast noderna med inga kanter, och sedan lägga till kanter, så att vi gradvis bygger upp trädet. Betrakta nu första kanten i denna listan: uppenbarligen kommer denna att vara en del av MST:n. Så vi kan lägga till den.



Vi fortsätter...



När vi sedan försöker lägga till en kant mellan 3 och 4 märker vi att denna skapar en cykel! Eftersom vi inte vill ha cykler så kan inte alla kanter i cykeln finnas. Eftersom kanten vi försökte lägga till är dyrare än alla andra, så skippar vi att lägga till den. Och så vidare...



Den enda långsamma delen av den här algoritmen är att märka när cykler skapas. Så vi börjar med att sortera vikterna, och sedan får vi följande problem:

- Vi får givet nod A och B. Kommer det skapas en cykel mellan A och B om vi lägger till en kant mellan A och B?
- Om inte, lägg till en kant mellan A och B.

Kan vi lösa detta med union find? (Svar: Ja.)

9. https://open.kattis.com/problems/minspantree
10. https://po.kattis.com/problems/bandwidth
11. https://open.kattis.com/problems/redbluetree
12. https://open.kattis.com/problems/more10
13. https://open.kattis.com/problems/brokenminimumspanningtree

11. Webscraping med Python

23:e juli

Idag ska ni få lera er om webscraping. Detta innebär att använda kod för att läsa in en hemsida och extrahera data från den. Sedan kan man analysera datan för att ta reda på saker.

Installera bibliotek

Vi kommer att använda ett flertal bibliotek för att underlätta med scrapingen. Se till att ha Python installerat och kör sedan följande i terminalen:

```
pip3 install requests beautifulsoup4 matplotlib
```

Testa sedan att köra följande kod:

```
import requests

website_link = "http://motherfuckingwebsite.com/"

response = requests.get(website_link)

print(response.text)
```

Om du kör koden kommer du att få HTML-koden för hemsidan. Det är denna som beskriver hur hemsidan ska se ut. Allting du kan se på hemsidan finns i den här koden! Testa att gå in på hemsidan och försök se om du kan para ihop innehållet på den med delar av koden. Låt säga att vi vill få ut listan som motiverar varför det är en bra hemsida.

För att göra det behöver vi förstå hur HTML:en av sidan ser ut. Lättaste sättet att ta reda på detta är genom att trycka CTRL+SHIFT+C i din webläsare. Om du sedan klickar på någon del av hemsidan, så visas motsvarande delen av koden i fönstret till höger eller där nere.

Vi kan nu notera att listan vi letar efter ligger innanför en så kallad ul (lista). Vi kommer nu använda Beautifulsoup för att extrahera den från koden. Lägg till följande under resten av koden:

```
from bs4 import BeautifulSoup

soup = BeautifulSoup(response.content, "html.parser")
ul_elements = soup.find_all("ul")
print(ul_elements)
```

Om vi kör koden skriver den ut vår lista! Det är dock inte i strängar, utan konstiga Beautifulsoup-objekt. För att göra om det till en hederlig lista med Python-strängar väljer vi först den första ul:en ur resultatet (i teorin kan det finnas mer än en ul i sidan), och loopar sedan över varje li (ett element i en lista) i den:

```
reason_list = ul_elements[0]
```

```
list_strings = []
for element in reason_list.find_all("li"):
    list_strings.append(element.text)
print(list_strings)
```

I slutändan har vi kod som skriver ut listan.

Uppgifter

1. Extrahera alla titlar (h2 eller h3) i hemsidan och lägg de i en lista.
2. Extrahera alla strängar som dyker upp i hemsidan.

Etik/laglighet

I allmänhet är web scraping i sig inte olagligt. Att skicka så många förfrågningar till en sida att den går ner/blir långsam kan däremot vara olagligt. De flesta hemsidor blir nog inte *glada* över att man scrapear de, men så länge du håller dig till en rimlig volym requests (≈ 2/sekund) kommer ingen bry sig i praktiken.

Kodstil

Bry dig inte så mycket om att skriva ”bra” kod - i de flesta fall scrapear du hemsidan en gång, får svar på vad du vill veta, och slänger koden. Det är inte att rekommendera att skriva ett större program som bygger på scraping, eftersom hemsidorna kan ändra sin layout när som helst utan förvarning.

Varför?

Web scraping är mest användbart för att samla in stora volymer data som följer något standardformat. Det är inte värt att göra det om det går snabbare för hand eller om datan är för ostrukturerad. Det är också värt att nämna att vissa hemsidor ger villigt bort sin data genom API:er. Så länge de inte kostar är dessa bättre, eftersom det är mycket enklare.

Svårare exempel

Motherfuckingwebsite är ganska okomplicerad i sin struktur. Koden bakom moderna hemsidor är ofta väldigt komplicerad, vilket gör det svårare att få ut sin data. Vi ska nu visa hur man kan extrahera data ur en lite mer komplicerad hemsida, samt visualisera datan.

I de vanliga olympiska spelen finns ett spännande fenomen: länderna som håller i OS brukar vinna fler medaljer än åren innan och efter de hålls. Du kan läsa mer om fenomenet här. Vi vill nu svara på om samma fenom återfinns i tävlingsprogrammerings-VM (IOI). Vi kommer att använda följande hemsida:
https://stats.ioinformatics.org/. (Vi har tillåtelse att använda deras sida såhär. De ser det som ett stresstest).

Vårt mål är att ge programmet ett år, och sen plockar det automatiskt resultatet för landet som höll i IOI det året och några år före och efter.

Vi börjar med att hitta året som landet höll i IOI:

```
import requests
from bs4 import BeautifulSoup
from matplotlib import pyplot as plt

which_year = 2016

def get_host_name(year):
    data = requests.get("https://stats.ioinformatics.org/olympiads/")
    soup = BeautifulSoup(data.content, "html.parser")
    data = soup.find_all("div", {"class": "maincontent"})[0]
    inner_data = data.find_all("table")[0]
    for row in inner_data:
        columns = row.find_all("td")
        if len(columns) and str(year)==columns[1].text:
            return columns[3].text

host_name = get_host_name(which_year)
print(host_name)
```

Denna koden skapades genom att gradvis gå djupare och djupare ner i HTML:en med hjälp av att printa, ända tills datan vi vill åt nåddes.

Vi kan nu lägga till kod för att hämta resultaten för det landet ett visst år:

```
def get_results_for_year(year, country):
    data = requests.get(f"https://stats.ioinformatics.org/results/{year}")
    soup = BeautifulSoup(data.content, "html.parser")
    data = soup.find_all("div", {"class": "maincontent"})[0]
    inner_data = data.find_all("table")[0]
    results = []
    for row in inner_data:
        columns = row.find_all("td")
        if len(columns)<12:
            continue
        if columns[2].text == country:
            results.append(int(columns[0].text))
    return results

print(get_results_for_year(which_year, host_name))
```

Vi vill nu visualisera detta. Vi kommer att använda biblioteket matplotlib och dess funktion plot. Given en mängd punkter $P_1, P_2, \ldots, P_N$ ritar den linjerna $\overline{P_1P_2}, \overline{P_2P_3}, \ldots, \overline{P_{n-1}P_N}$. Tanken är att vi tar medelvärdet av resultaten för varje för år och drar linjer från år till år. Matplotlib vill få punkterna i form av en lista för alla x-koordinater och en för alla y-koordinater. Sen skapar den en punkt (x[i], y[i]) för varje $i < len(x)$.

```
placements = []
years = []

for i in range(2002, 2024):
    print(f"Fetching year {i}")
    years.append(i)
    avg = 0
    for placement in get_results_for_year(i, host_name):
        avg += placement
    placements.append(avg/4)

plt.plot(years, placements)
plt.show()
```

Uppgifter

3. Undersök om fenomenet återfinns i IOI? Dvs, tar värdlandet fler/bättre medaljer?

4. Medelvärdet är ett ganska dåligt sätt att beskriva ett lands prestation, eftersom en enda dålig prestation kan dra upp medelvärdet starkt. Testa några andra mått! Exempelvis medianen, bästa resultatet, geometriska medelvärdet eller harmoniska medelvärdet (googla!).

5. Finns det något bättre sätt att visualisera datan? Använd google och testa runt.

6. Notera att varje gång vi visualiserar ett land frågar vi om samma 19 sidor varje gång. Detta är både långsamt och elakt mot webservern. Försök cache:a de med requests-cache.

```
import requests_cache

requests_cache.install_cache('my_cache', backend='sqlite', expire_after=86400)
    # Cache expires after 1 day
```

Eget projekt

Här har ni förslag på hemsidor ni kan scrapea tillsammans med en frågeställning:

- (Joshua blir glad om du fixar den här): ta alla users med 3000+ rating, ge en lista på alla bloggar de skrivit som inte är en editorial. https://codeforces.com/ratings.
- Vem har snabbast lösning på största antalet problem på https://po.kattis.com/problems? (tryck på ett problem och sen statistics)
- Vilket livsmedel har bäst kalori eller protein per krona från https://www.coop.se/handla/?
- Träna AI? Google images för träningsdata. Överkurs: scrapea hela internet och bygg ChatGPT (oironiskt hur de flesta LLM:s kom till).

- Minsta antalet länkar du måste trycka på för att komma från någon sida till en annan i Wikipedia (blir väldigt många requests, så kommer ta en stund). Exempelvis ”Nuclear fusion” till ”Godzilla”.
- Statistiksida för ditt favoritspel. Kan man få ut intressant information?
- Scrapea hela https://www.hemnet.se/ för att skapa en bättre sökfunktion.
- Scrapea en nyhetssidas innehåll. Nu kan du läsa nyheterna utan reklam!
- Egen hemsida + frågeställning!

Efter LLM:s har det blivit mycket svårare att scrape:a sidor som Twitter eller Reddit, eftersom innehållet helt plötsligt har blivit värdefullt. Vi avråder er från att ens försöka; om ni scrapear en hemsida där ni har ett konto finns en risk att ni blir bannlysta.

Häftet sammanställer lektionsmaterial från samtliga ordinarie programmeringslektioner på Mattekollo och Programmeringskollo 2024. Lektionerna har hållts på tre nivåer: Ada (grundläggande), Babbage (medel) och Church (avancerad).

Material från mattelektionerna samt information om andra mattekollon och programmeringskollon hittar ni på http://www.programmeringskollo.se

Tack till alla ledare och deltagare!