Västra Götalandsregionen

2022-07-07

Code week 2022: Självstudiematerial

En introduktion till AI-tekniken övervakad maskininlärning med enkel Python-programmering

VÄSTRA GÖTALANDSREGIONEN

Innehåll

Kul att du vill lära dig lite om praktisk AI och specifikt maskininlärning!

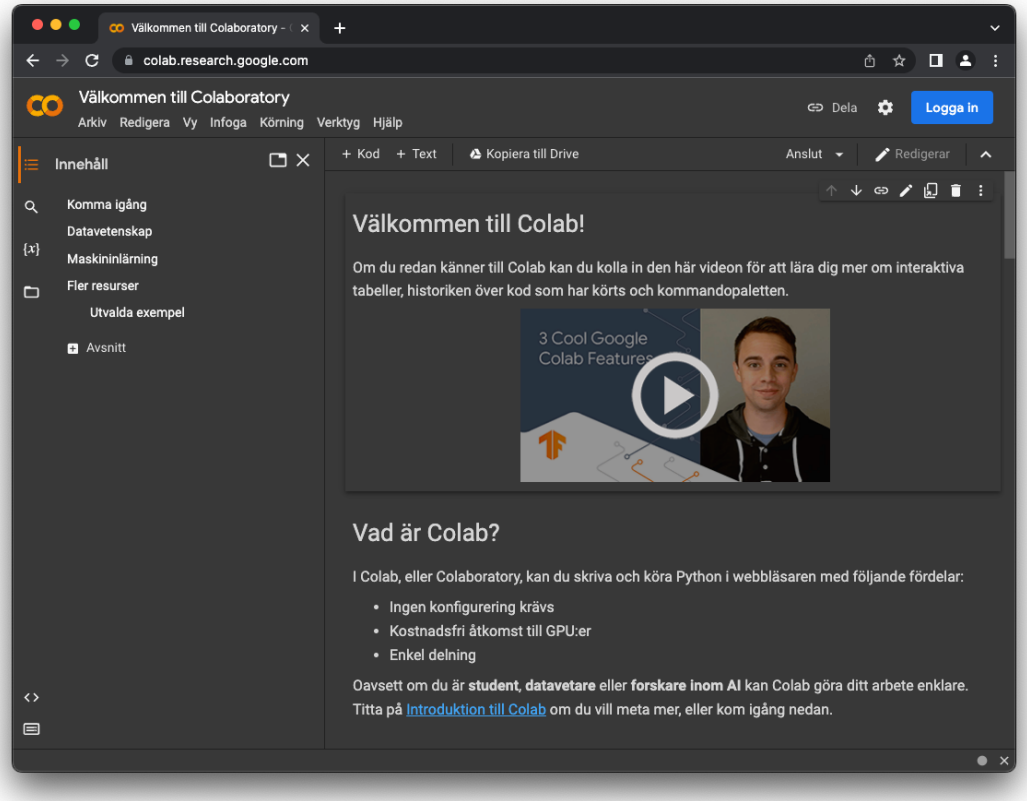
Om den tekniska miljön du behöver

Enklast är att använda nättjänster för de tekniska övningarna, många har en gratisversion som duger gott. Exempel på sådana är Google Colab[1], Python Anywhere[2] och datalore.io[3] Den mer komplicerade versionen är att installera saker på din dator. Om du väljer att installera saker på din dator så är det enklast att installera Anaconda[4] och sedan funktionen Jupyter Notebook.

I det här materialet kommer vi utgå från Google Colab eftersom det går snabbt att komma igång med.

Steg 1: Logga in eller skapa ett gratiskonto hos Google Colab

Börja med att i webbläsaren gå till följande adress: https://colab.research.google.com



Figur 1: Uppe i högra hörnet kan du logga in. På samma plats skapar du ett nytt konto.

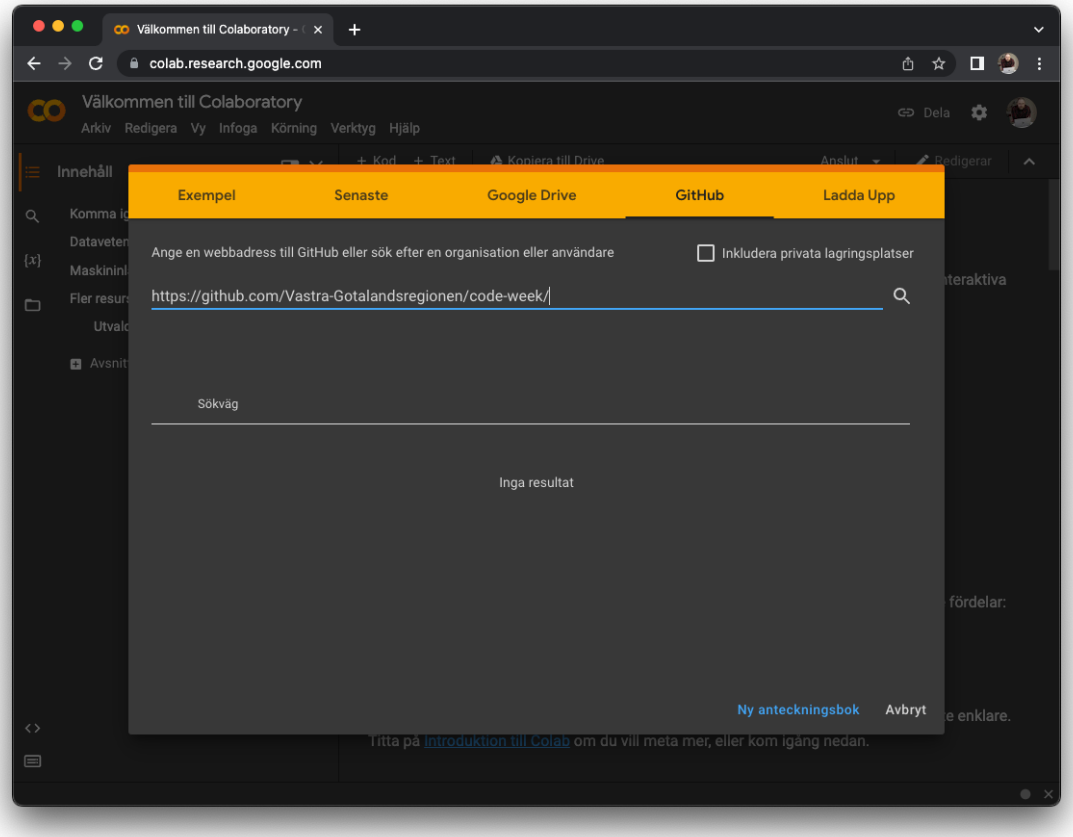
[1] https://colab.research.google.com
[2] https://www.pythonanywhere.com
[3] https://datalore.io
[4] https://anaconda.org

Om du redan har ett Google-konto via jobb eller skola kan du logga in med det användarkontot. Det upptäcker du om du skriver in den e-postadressen vid inloggning.

Har du inget konto via jobb eller skola kan du fortfarande ha ett privat, exempelvis för Googles mejltjänst Gmail, dessa inloggningsuppgifter går också bra på Colab.

I annat fall behöver då skapa ett nytt konto. Om du inte redan är på https://colab.research.google.com skriver du in den adressen i webbläsaren och börjar med att klicka på länken “Logga in” uppe i högra hörnet och följa guiden.

Steg 2: Gör en kopia av startpaketet



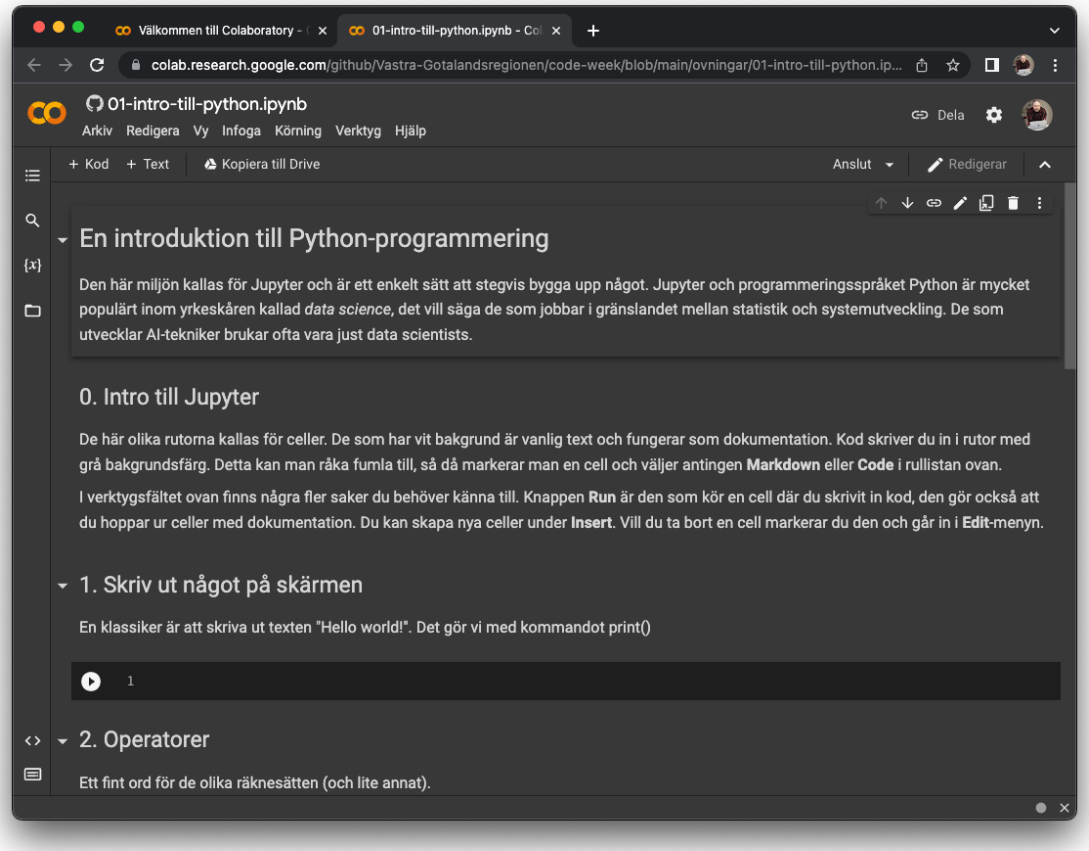
Figur 2: Skriv in vad från Github du vill kopiera.

Det finns ett startpaket att utgå ifrån. Det är särskilt lämpligt om du är oerfaren när det gäller programmering, inte har använt notebooks tidigare eller vill ha ett facit att tjuvkika på vid behov.

Kopiera filer från startpaketet till ditt konto så här:

1. Se till att du är inne på Google Colab (https://colab.research.google.com)
2. Aktivera fliken som heter ”Github”. Se figur 2.
3. Skriv in https://github.com/Vastra-Gotalandsregionen/code-week/ och tryck på Enter.
4. Leta upp första filen, ”ovningar/01-intro-till-python.ipynb”

5. Klicka på ikonen längst ut till höger på raden. En fyrkant med en pil riktad upp till höger.
6. Nu öppnas en ny flik i din webbläsare med din första övningsfil.



Figur 3: En notebook kan se ut så här. Notera play-knappen längst till vänster i det svarta fältet. Klickar man på den så körs koden man skrivit in.

Steg 3: Starta din notebook för första gången

Nu har du din kopia av övningsfilen på ditt eget konto.

Startpaketet har vi hämtat från Microsofts öppna kodtjänst som heter Github. I det här fallet är det kod vi tagit fram till Code week, men du kan alltid utforska vad mer du kan hitta. På samma plats hittar du även ett presentationsmaterial till denna övning. Se mer på: https://github.com/Vastra-Gotalandsregionen/code-week/

Du har två olika sorters filer i startpaketet. Dels de som ligger i mappen ”ovningar” och är dina övningsfiler när du följer med en föreläsning med detta material eller studerar detta material självständigt. Dessutom har du en mapp som heter “facit”. Där ligger filer med samma filnamn som övningsfilerna, men dessa är färdiga och kan fungera som ett facit för dig om du kör fast.

För att starta din första notebook:

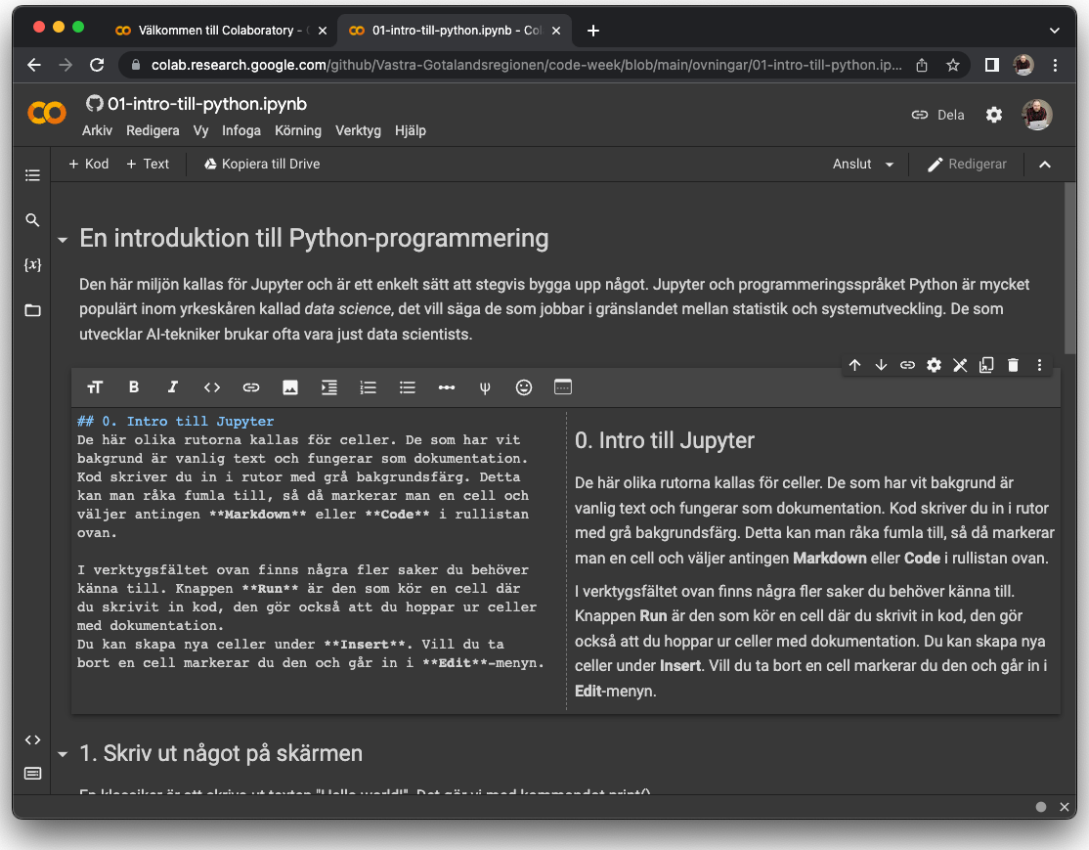
- Se till att du har samma sak på din skärm som du ser på Figur 3.

Du har en så kallad Jupyter Notebook[5] framför dig, framöver enbart kallad notebook.

Notebooks är en utvecklarmiljö du kan köra direkt i din webbläsare. I det här fallet körs din kod i Googles molntjänst Colab för enkelhetens skull men det skulle lika gärna kunna vara på din arbetsplats, skola eller egen dator.

Man blandar kod (i de mörkare rutorna/cellerna) och dokumentation (i ljusare rutor/celler) för att enkelt strukturera upp det man vill ha gjort på ett begripligt sätt.

En notebook är uppdelad i så kallade celler. En cell är en ruta vars bakgrundsfärg är antingen ljus eller mörk. En ljus cell innehåller dokumentation, i en grå cell skriver vi in programmeringskod.



Figur 4: Dubbelklicka på de ljusare cellerna och lär dig lite om dokumentationsspråket Markdown.

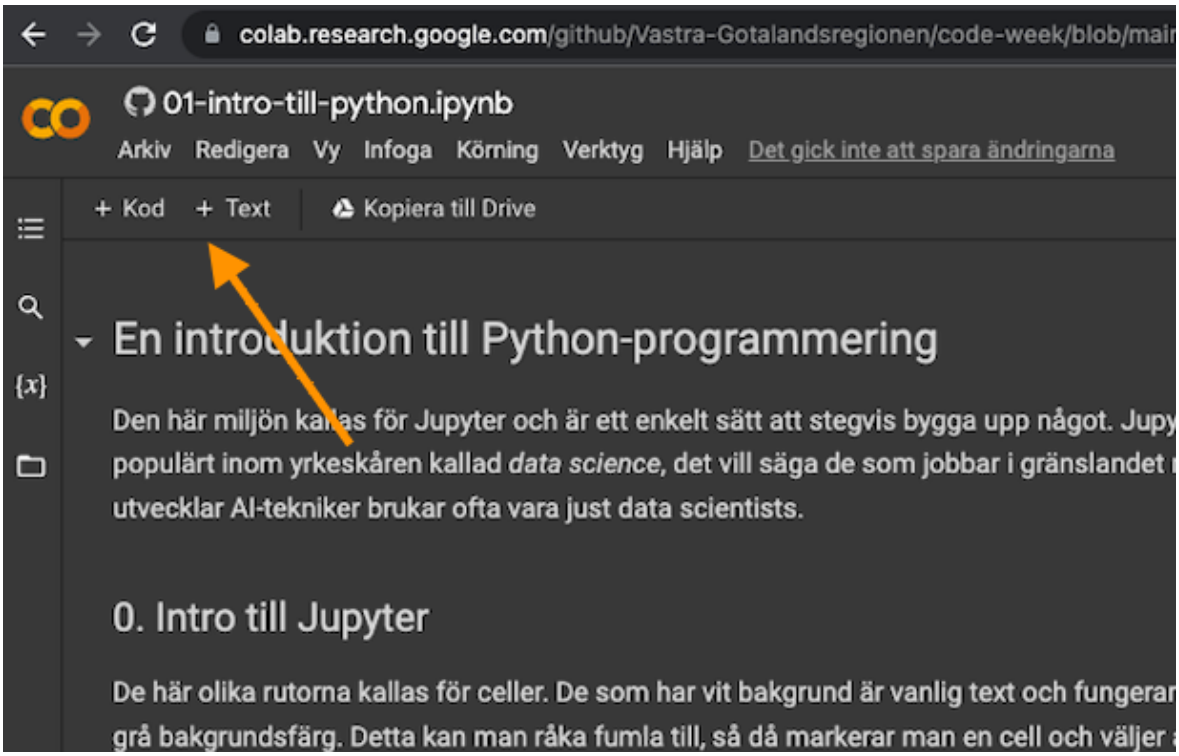
Om du dubbelklickar på en ljus cell kommer den gå över till redigeringsläge. De ljusa cellernas innehåll skrivs med ett dokumentationsspråk som heter Markdown[6]. Man behöver inte lägga någon energi på det, se det mer som ett snabbt sätt att skriva anteckningar intill din kod.

För att komma ur redigeringsläge klickar du på knappen “Stäng Markdown-redigerare” ute till höger, illustrerad med ikonen med överstruken penna.

[5] https://jupyter.org
[6] https://sv.wikipedia.org/wiki/Markdown

Om du vill provköra en snutt kod du skrivit i en mörk ruta, då trycker du på “play-knappen” till vänster i din mörka cell eller tangentbordskombinationen Ctrl + Enter (eller CMD + Enter på en Mac).

Eftersom du kopierat den här övningsfilen från en tredjepart kommer du (åtminstone hos Google Colab) få en varning om att Google inte ansvarar för vad som händer när du kör koden. Det är ingen fara. Vi kommer inte göra några skumma grejer, så du kan klicka på ”Kör ändå”.



Figur 5: För lägga till nya celler kan du välja på text (dokumentation) eller kod.

När du senare vill skapa dina egna notebooks kommer du skapa dina egna celler (genom verktygsmenyn + Kod eller + Text). Kom då ihåg att kod skrivs in i mörka celler, dokumentation i ljusa celler.

Intro till programmering – på två minuter

Vill du instruera en dator behöver du programmera den. Programmering är alltså att sammanställa en mer eller mindre lång lista med instruktioner du vill få utfört.

En utmaning med att programmera är att datorn gör precis som den blir tillsagd. Det kan låta som en bra sak, och så länge du skriver "rätt" programmeringskod är det bra. Men det gör att du arbetar på maskinens villkor, på ett språk som är mer anpassat efter maskinens behov än dina.

Det finns många olika programmeringsspråk. De olika språken har olika styrkor och svagheter. Några vanliga programmeringsspråk i Sverige är C#, Java, C++ och PHP. Just när det gäller hantverket runt maskininlärning är det vanligt att man använder språket Python.

Intro till maskininlärning – på fem minuter

Klassisk programmering kallas oftast för systemutveckling eller computer science i akademisk miljö. När det gäller maskininlärning är det i stället datavetare eller data science. Det är alltså en skillnad på dator och data, där man historiskt har instruerat datorer genom programmering och att en datavetare idag i stället fokuserar på den data som datorn behöver för att "programmera sig själv".

Maskininlärning kan alltså förenklat förklaras som att man hjälper en sorts maskin att bygga upp erfarenheter baserat på information man matar den med. Dessa data kan vara att erbjuda mängder med bilder som antingen föreställer äpplen eller apelsiner. Det maskinen till sist lär sig är vilka egenskaper (kallas features inom maskininlärning) som är typiska för ett äpple och vad som är typiskt för en apelsin.

Övervakad maskininlärning

Övervakad maskininlärning (supervised machine learning) betyder att man tillsammans med informationen berättar vilka bilder som föreställer äpplen och vilka som föreställer apelsiner. Det kallas för att informationen är annoterad, det finns alltså en lärare som hjälper maskinen att lära sig. Maskinen kommer ganska snart att inse att en slät yta och en liten mörk pinne som sticker ut är gemensamt för många äpplen och att en brandgul och gropig yta brukar den "se" på en apelsin.

Poängen med detta är att senare kunna skicka en för maskinen helt okänd bild och be maskinen säga om den tror det föreställer ett äpple eller en apelsin. Den här maskinen skulle alltså ha en snäv kunskap endast om två sorters frukter, samt att den inte har mer kunskap än de bilder den sett under sin träningsfas.

Oövervakad maskininlärning

Till skillnad mot versionen som är övervakad har man ingen pedagogisk lärarfunktion i oövervakad maskininlärning (unsupervised machine learning). Maskinen måste alltså på egen hand se mönster och försöka lista ut vad det är för information. Det kan vara fynd som att hitta avvikelser i siffror, som att en person haft en period av avvikande pulsdata, eller regelbundenheter. Maskinen kan lätt upptäcka sådant i en enorm mängd med siffror, men då den inte är övervakad kan den inte beskriva **vad** det är den hittat eller ge en trolig förklaring bakom avvikelsen.

Ett sätt att se det på är att man med oövervakad maskininlärning är öppen för vilka fynd som finns i en mängd data, medan man med övervakad maskininlärning redan bestämt sig mer exakt vad man är ute efter.

Det finns fler varianter av maskininlärning vi inte kommer gå in på. Googla på reinforcement learning, transfer learning eller deep learning för att läsa mer.

Övningar

Du behöver ha skapat ett konto hos Google för att kunna köra övningarna. Om du inte gjort det så se tidigare avsnitt.

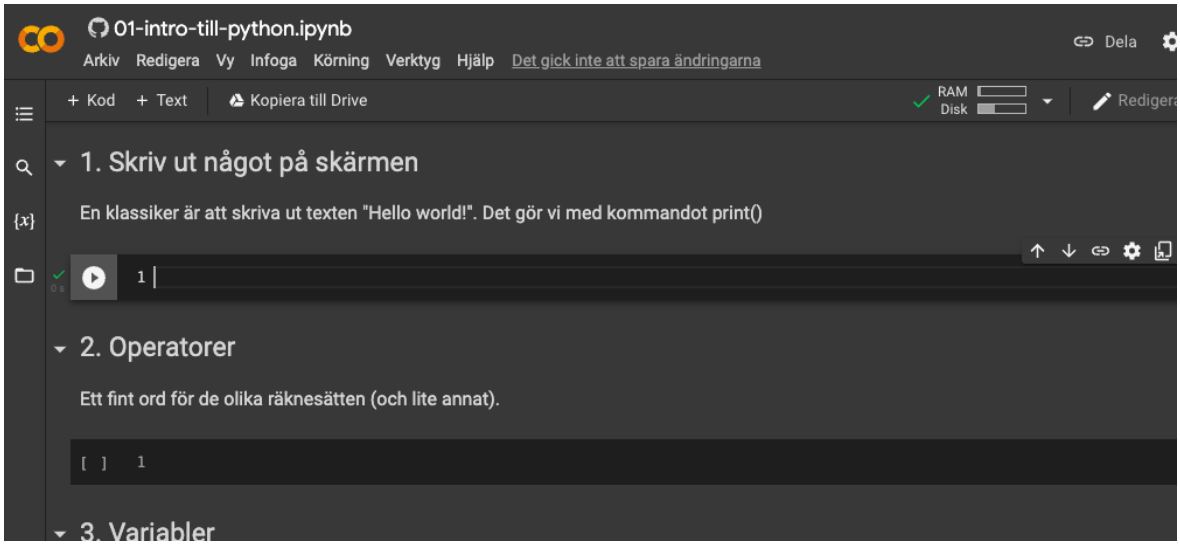
Övning 1: grundläggande programmering

Det finns ett antal grundkunskaper man behöver ha lite koll på. Vi kommer gå igenom dem i denna övning för att i nästa övning fokusera på maskininlärning.

Om du inte redan har den första notebooken öppen så öppna den nu, den som heter "ovningar/01- intro-till-python.ipynb". Om något inte verkar fungera kan det vara värt att börja om. Eftersom det är en gratistjänst kan man dessvärre inte räkna med helt felfri drift, då borde man skaffa sig en betald tjänst i stället.

Du kommer i respektive notebook få lite vägledande information i de ljusa cellerna, men de är främst där för att hjälpa dig hitta rätt mörk cell.

1.1 Hejsan världen



Figur 6: För att kunna skriva kod ställer du markören i ett mörkt fält.

Klicka i det första mörka fältet. Nu kommer markören blinka där och du ska också få en ram som visuellt visar var i din notebook du är aktiv.
Den kod vi ska skriva är klassisk. Vi ska göra minsta möjliga program och det brukar gå ut på att få maskinen att säga hej till världen - Hello world!

Så nu skriver du det första programmet i den mörka rutan, nämligen:

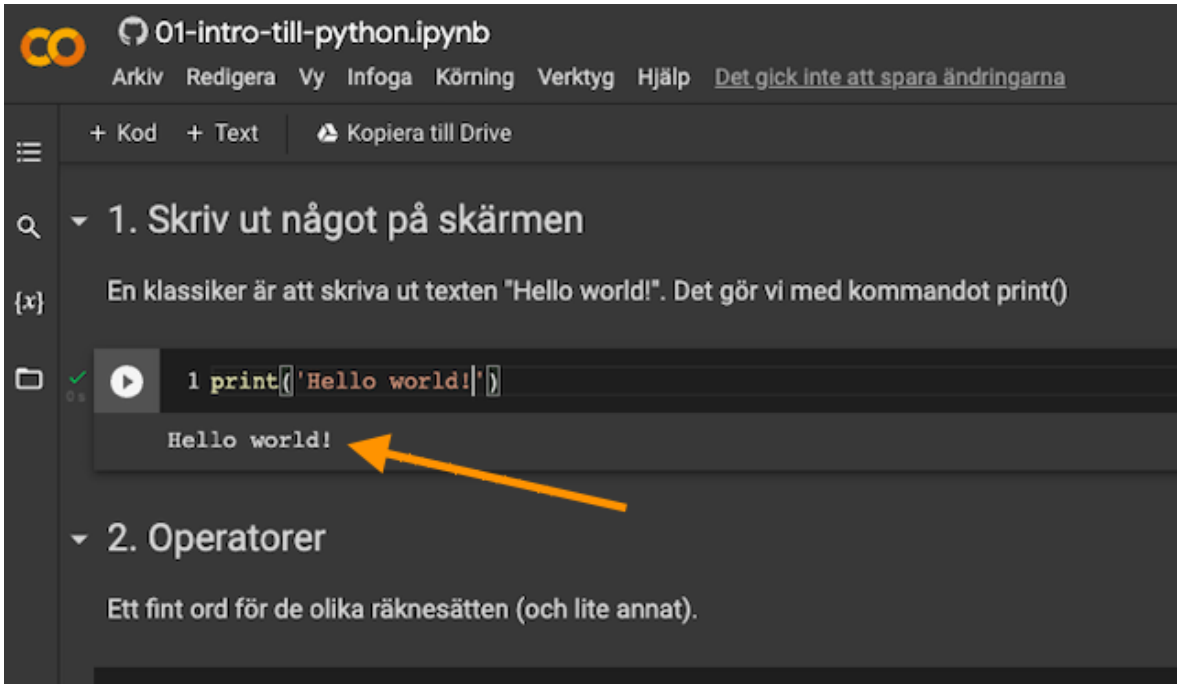
```
print("Hello world!")
```

Du säger åt maskinen att printa något, skriva ut på skärmen. Sen är det början på en parentes, ett citationstecken (eller en rak apostrof går också bra), lite text, ett till citationstecken (eller rak apostrof) och en avslutande parentes. Om du tycker det är svårt att hitta apostrof på tangentbordet går det lika bra med citationstecknet som du hittar på Shift + 2, det vill säga ".

Det du behöver förstå av denna rad med kod:

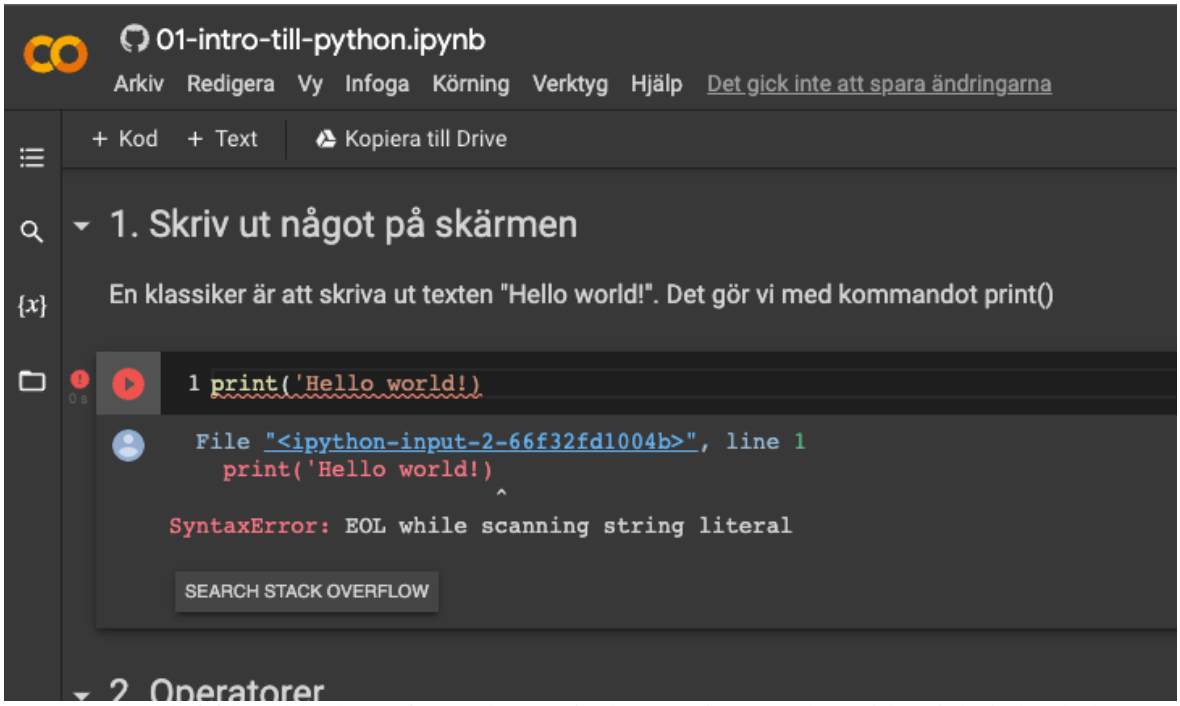
1. *print* är en inbyggd funktion i programmeringsspråket Python och dess uppgift är att hjälpa dig skriva något på skärmen.
2. Det som finns inom parentes är den information du vill ge *print* att jobba med.
3. Att texten *Hello world!* omges av citationstecken är för att berätta att det är just en textsträng. Vi kommer gå in mer på textsträngar och andra typer av data senare.
4. Citationstecknet hittar du på tangentbordet genom att hålla ned Shift-tangenten och trycka på siffran 2.
5. Det går lika bra att använda apostrof som citationstecken, så länge som du är konsekvent i ditt val.

Nu kan du klicka på Play-knappen i cellen vänstra kant, eller Ctrl + Enter på tangentbordet, för att köra denna cell med programmeringskod. Vad hände? Om du skrev rätt så skrevs texten *Hello world!* ut precis under din gråa ruta. Se nedan bild.



Figur 7: När du kört din cell och allt fungerar som det ska kommer Hello world! dyka upp precis under din mörka cell.

Om du inte skrev rätt eller annat fel inträffade så kommer det stå ett felmeddelande direkt under den grå rutan. Dessa felmeddelanden är mer eller mindre kryptiska, men om man inte direkt listar ut vad man ska åtgärda så är det vanligaste att man kopierar felmeddelandet och googlar efter det. Ofta är det på webbplatsen StackOverflow.com man hittar svaret. Det här är vad många utvecklare lägger sin tid på - att felsöka.



Figur 8: Exempel på felmeddelande. Det finns en liten uppåtpil som markerar var den gick bet på din kod. Felet här är att det saknas en avslutande apostrof innan den avslutande parentesen.

Det finns många olika sorters fel och det är en överkurs för denna självstudie, men i ovanstående exempel anges *SyntaxError*. Det innebär att man gjort ett fel i hur man skrev programmeringsspråket. I det ovanstående exemplet förväntas man ha en avslutande apostrof innan en avslutande parentes. Därför klagar den på felaktig syntax.

När man rättat eventuella fel är det bara att köra cellen på nytt och se hur det går. Detta kallas för *trial and error*, man testar sig fram med saker man inte vet exakt hur man löser.

1.2 Operatorer

Operator är ett fint ord för de olika sätten att beräkna saker, som addition, subtraktion, division, med mera. Ofta menar man dock lite mer än det vi lärt oss under grundskolans matematik. Ett sådant exempel är hur man sammanfogar två textsträngar, så kallad konkatenering.

Här kommer lite beräkningar med hjälp av operatorer. Vi ber maskinen räkna ut 1 + 3, men också jobbigare beräkning vi helst slipper göra utan en dators hjälp. Raderna som börjar med en hashtag är en kommentar, det behöver du inte skriva men det är ett sätt att göra anteckningar i sin kod. Kommentarer används ofta på detta sätt för att förklara den kod som sedan följer.

```
# skriv ut summan av en addition
print(1+3)

# en multiplikation som är dryg i huvudräkning
print(78324687324676*12321)

# division
print(1/3)

# sammanfoga text, så kallad konkatenering
```

```
print("Hello" + "world" + "!")
```

Vi har tre matematiska exempel; addition, multiplikation, division. Vi avslutar med konkatenering, alltså att addera textsträngar vilket ger en lång textsträng.

1.3 Variabler

För att tillfälligt spara information kan man använda variabler. Det är som namngivna behållare du kan stoppa ner lite allt möjligt i. Tänk dig en hink du kan lägga lite vad du vill i, det kan vara grus, vatten eller glassbåtar.

En poäng med variabler är att din kod blir lättare att läsa, åtminstone om du namnger dina variabler på ett logiskt och beskrivande sätt. **Du får alltså namnge dina variabler nästan hur du vill, men det finns krav som att de inte kan innehålla mellanslag, specialtecken och kan inte börja med en siffra.**

Det tredje kodexemplet går ut på att räkna på omkretsen av en rektangel. Notera att vi inte skriver ut måttenheten efter respektive siffra, om vi skrev *9 cm* skulle datorn inte förstå eftersom vi blandar en siffra med bokstäver. Vi skriver i stället endast in siffran.

```
rektangel_y = 9
rektangel_x = 7

omkrets = (rektangel_y * 2) + (rektangel_x * 2)
print(omkrets)
```

Om du skriver av ovanstående kod exakt och kör cellen ska den svara med siffran *32*. Det vill säga omkretsen av rektangeln med sidornas mått 9 och 7.

rektangel_y och *rektangel_x* är de olika längder som rektangelns sidor har. Om man tar sidorna multiplicerat med sig själva och adderar resultaten har man fått rektangelns alla fyra sidor och vet hur långt det är runt.
Samma matematikregler du är van vid, som att uträkningar inom parentes körs först, fungerar när du programmerar. Ibland skriver man koden övertydlig mest för att göra den mer läsbar, även fast parenteser inte alltid behövs är det vanligt att hitta dem i alla fall.

1.4 Lista saker

Det finns flera olika sorters listor inom programmering. De är specialiserade för olika användningsområden, som att alltid vara sorterade eller kunna innehålla olika komplexa saker.

Poängen med en lista är dock självförklarande, man vill lista saker. Vi börjar med att skapa en tom lista, sedan lägger vi till två frukter.

```
favvo_frukt = list()

# lägga till något till en lista
favvo_frukt.append("Äpple")
favvo_frukt.append("Apelsin")
print(favvo_frukt)

print(type(favvo_frukt))
```

Här lägger vi två gånger till nya frukter till listan som heter *favvo_frukt*. Sedan skriver vi ut listans innehåll och till sist skriver vi ut vilken datatyp som variabeln/listan *favvo_frukt* är.

1.5 Importera funktioner och moduler

Programmeringsspråk har en massa funktioner man kan lära känna och ha nytta av. Dessa funktioner brukar kallas för dess standardbibliotek. Exempelvis kan man be om hjälp med saker som värdet Pi istället för att behöva skriva in decimalerna för hand. Värdet av Pi finns i en modul som heter 'math', det är inte konstigare än att vi importerar *'math'*.

```
import math

print("Pi är", math.pi)
```

Efter att vi importerat matematik-delen av Pythons standardbibliotek använder vi funktionen print() för att först skriva ut texten ”*Pi är*” och sedan anropar vi math-modulens underdel pi som då ger oss siffror tillbaka.

1.6 If-satser

Artificiell intelligens beskylls ibland för att enbart bestå av en massa villkor, if-satser alltså. Här ska vi sätta upp villkor. Nu börjar vi närma oss smarthet och inte bara ha en massa beräkningskraft. Dock är if-satser egentligen bara ett sätt att beskriva regler.

Men frågan är om du skulle kunna avslöja en "AI" som har miljontals färdiga svar på frågor?

Denna övning handlar om att plocka fram aktuell information, nämligen innevarande veckodag. Eftersom det är en maskin det handlar om så är veckodagarna numrerade snarare än att det är måndag till söndag det handlar om.

Precis som med exemplet tidigare med matematik-modulen och talet Pi så drar vi nytta av modulen för datumtid och importerar datum. Här stöter du för första gången på ett indrag i koden. Antingen gör du dessa med tabbtangenten eller tre mellanslag – det är viktigt att du enbart gör på det ena eller andra sättet. Datorer är kinkiga.

```
from datetime import date

veckodag = date.weekday(date.today())
# måndag är 0, tisdag är 1 -> söndag är 6
print("Nummer på veckodagen", veckodag)

if veckodag > 4:
   print("Nu är det helg!")
elif veckodag is 4:
   print("Nästan helg, det är ju fredag")
else:
   print("Nope, inte helg...")
```

Veckodagen är alltså en siffra mellan 0 och 6. Man börjar (nästan) alltid med siffran noll när det gäller data och olika sorters listningar av saker. Har maskinen du kör med svenska inställningar kan du räkna med att 0 står för måndag, men risken är att 0 är för söndag på grund av “kulturella inställningar”, men det är inget vi kommer gå djupare in på här.

Så vad är det för if-satser vi kodat? Först säger vi att om/if *veckodag* är mer än fyra, då vill vi skriva ut att "Nu är det helg!". *elif* betyder typ "eller om" och den kollar om *veckodag* är exakt 4, alltså inte mer än fyra som föregående, så skriver vi ut något annat. Siffran fyra representerar alltså fredag, fem för lördag och sex för söndag.

If-blocket avslutas med *else*. Man kan säga att här landar man alltid om inget av det tidigare varit sant. Om inget av ovanstående if/elif är sant så är det inte helg, det är måndag-torsdag.

Det finns alltså tre sammankopplade regler som gör detta kodblock dynamiskt. Först frågar vi om veckodagen är lördag eller söndag. Senare om det är fredag. Om inget av dessa stämmer kan vi konstatera att det varken är fredag eller helg.

Tänk dig hur du skulle fråga en människa för att lista ut vad det är för frukt hen har i sin väska. *Är den grön? Ja! Ok, är den oval? Nej! Jag tror det är ett grönt äpple.*

Samma system kan du fundera kring när det gäller triage, alltså en initial bedömning av hur illa däran en patient är. Eller det enklare LABC[7] (Livsfarligt läge – Andning – Blödning – Chock) som även vi som inte är vårdkunniga ska ha som stöd för att prioritera vad vi undersöker vid en krissituation. Det är som ett intervjumanus om du ringer 112 när du hittat någon liggandes på gatan.

F: Har du undanröjd eventuellt livsfarligt läge?
S: Ja, jag släpade personen bort från bilvägen upp på trottoaren!
F: Andas personen?
S: Ja!
F: Blöder personen?
S: Ja!

I detta fall stannar vi på blödningen och når aldrig *Chock* så som LABC:s "algoritm" är utformad. Vi behöver ta hand om blödningen eller åtminstone bedöma hur allvarlig den är innan vi kan gå vidare.

1.7 Slutligen, slingor, loopar, iterationer, kärt barn

För att sysselsätta en maskin med lite mer jobb kan man instruera den att göra något tills dess att den är färdig. Här följer ett exempel på det.

Vi skapar en lista med förnamn. Den där typen av parentes hittar du med Alt Gr intryckt och så trycker du på antingen 8 eller 9. Du på Mac använder Option-knappen och 8 eller 9. Tänk på att skriva alla namn på en och samma rad, annars kommer du få syntax error.

```
# en sorts loop är for-loopen, den matas med en lista av något

lista_med_namn = ["Nisse", "Greta", "Magdalena", "Harald", "Lily"]

for namn in lista_med_namn:
   print(namn)
```

[7] https://sv.wikipedia.org/wiki/LABC

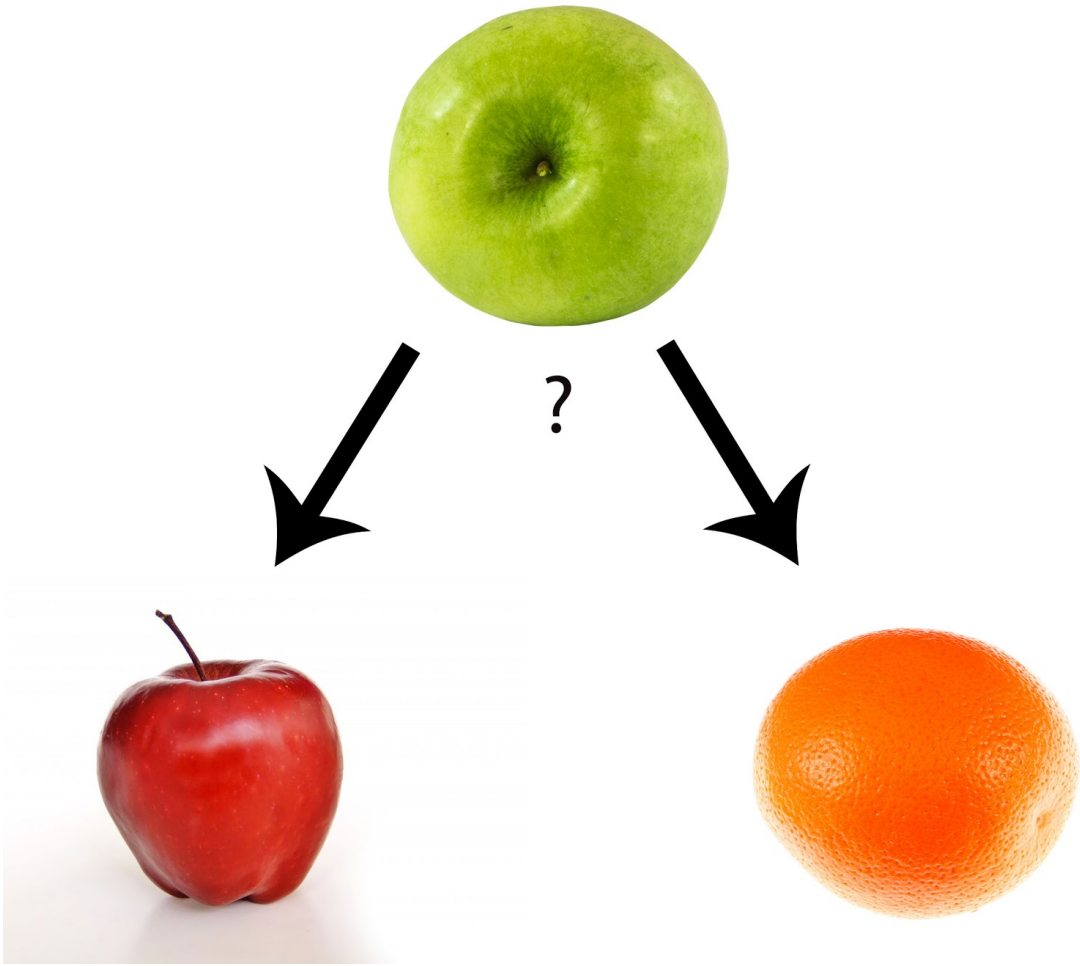
Den avslutande kodsnutten är en for-loop. Koden är lite bakvänd. Tänk på att vi kör raden print(namn) lika många gånger som det finns namn i listan *lista_med_namn*.

Så *namn* är alltså en variabel som första varvet i loopen innehåller *Nisse*, andra gången *Greta* och så vidare.
Det som kommer hända när du kör den här koden är att den skriver ut förnamnen vart och ett på egen rad. Först kommer *Nisse*.

Det var det, nu ska vi kolla på lite maskininlärning.

Övning 2: enkel övervakad maskininlärning

Nu ska vi testa lite maskininlärning. Det blir inte många rader kod, men desto mer abstrakt tänkande.
Konceptet är att mata maskinen med data den ska lära sig av. Den data maskinen tränar på kommer utgöra allt den vet om något och vara grunden för ett så kallat beslutsträd. Ett beslutsträd är ett sätt att sammanställa kunskap, det är inte lämpligt för alla problem men är en begriplig början för den som vill testa maskininlärning.



Figur 9: Beslutsträd är nyttiga för att klassificera saker. Som ifall en viss frukt är ett äpple eller apelsin.

Att lära en maskin genom exempel och erfarenheter, i stället för att manuellt mata in regler. Detta genom övervakad maskininlärning. Maskinen ska förutspå om en ny frukt är en apelsin eller ett äpple baserat på tidigare erfarenheter.

Nu ska vi öppna filen som heter:

- ovningar/02-supervised-machine-learning.ipynb

2.1 Samla in träningsdata

Egentligen behöver du bara skriva de tre raderna som inte är bortkommenterade med en brädgård/hashtagg som första tecken. Kommentarerna är bara där för att förklara hur vi skriver om våra data till nummer, vilket maskinen behöver.

Strax efter koden kommer du få respektive del förklarad. Det är inte så komplicerat egentligen.

```
from sklearn import tree

# features: vikt i gram, yta(1 för len, 0 för gropig)
# features = [[140, "len"], [130, "len"], [150, "gropig]]
features = [[140, 1], [130, 1], [150, 0], [170, 0]]

# labels = ["äpple", "äpple", "apelsin", "apelsin"]
# labels = 0 för äpple, 1 för apelsin
labels = [0, 0, 1, 1]
```

Den första insatsen är att importera scikit-learn. Det är en så kallad modul, den hjälper oss att genomföra maskininlärning utan att behöva skriva all kod från grunden.

Sedan skapar vi en variabel, *features*, som får en lista med fyra olika frukter. Den första frukten väger 140 gram och har en len yta, den andra väger 130 gram och har också en len yta. De två sista frukterna väger 150-170 gram och har en gropig yta. **Eftersom maskiner föredrar siffror skriver vi om ytans beskaffenhet; 1 får symbolisera len yta, 0 får symbolisera gropig yta.** Så nu har de fyra frukterna två egenskaper vardera som är siffror.

Sist ska vi berätta för maskinen vilka av dessa frukter som är äpplen och vilka som är apelsiner. Dessa uppgifter kallas för *labels*, du kan tänka på dem som att du sätter etiketter på respektive frukt. Genom att labels ens existerar och kan hjälpa maskinen att lära sig är denna kod övervakad maskininlärning. Om inga labels finns är det i stället oövervakad, då är det upp till maskinen själv att lista ut och gruppera dessa frukter på egen hand.

2.2 Träna upp klassificeraren och leta mönster i data

Vi skapar en ny variabel som vi döper till *klass*. Den får först datatypen att vara en beslutsträdsklassificerare. På rad två ger vi beslutsträdet den data den behöver för att bygga upp sin kunskapsmodell (beslutsträdet, alltså). *features* är våra fyra frukter från det tidigare kodblocket, *labels* är vår klassificering att två av frukterna är äpplen och två är apelsiner.

Ordet “fit” inom maskininlärning kanske kan översättas till svenska som att “passa in dessa data i din struktur”. Det är när vi anropar *klass.fit()* som maskinen sorterar upp sina intryck i ett beslutsträd. Den lär sig själv baserat på data, utan att uttryckligen ha programmerats med förbestämda regler.

Nu är ju dessa data väldigt banala jämfört med hur avancerade AI-system fungerar, inte minst för att vi har en begränsad världsbild med bara två olika frukter.

```
klass = tree.DecisionTreeClassifier()
klass = klass.fit(features, labels)
```

2.3 Predicera

Den återstående uppgiften är att förutsäga om en ny frukt är antingen ett äpple eller en apelsin – baserat på de två egenskaper maskinen nu lärt sig att tolka. Så vi skapar en ny variabel, *ny_frukt*, och säger att den nya frukten väger 180 gram och har slät yta, vilket blir en 1:a som ytans egenskap.

Sen anropar vi en funktion som heter *predict()* och ger den vår nya frukt. Maskinen känner bara till två frukter, den kallar dem för 0 och 1. Så om maskinen svarar att det är frukt 1 menar den apelsin och 0 är äpple.

En typisk if-sats, svaret är antingen det ena eller det andra.
Är prediktionen 1 skriver vi ut "Apelsin", är den något annat skriver vi ut "Äpple".

```
ny_frukt = [[180, 0]] # ny frukt, 180 gram och gropig yta

if 1 in klass.predict(ny_frukt):
   print("Apelsin")
else:
   print("Äpple")
```

Vi hade kunnat undvika att behöva maskininlärning för detta lilla exempel. Skillnaden hade varit att vi då fått skriva en if-sats som kollade på fruktens egenskap för yta. Om ytan var slät skulle den svarat att det var ett äpple, gropig yta skulle betyda apelsin. Det hade fungerat lika bra, men vad hade vi vunnit på det? Vi hade inte sparat särskilt mycket på mängden kod vi skriva.
Nackdelen skulle dock vara att koden blev mindre mångsidig, den hade inte lika bra hanterat att vi lagt till fler sorters frukter, eller nya egenskaper utöver vikt och yta. Så redan med ganska små datamängder kan principerna inom maskininlärning löna sig.

Problematisering av detta simpla exempel

Den här maskinen kan endast äpplen och apelsiner. Men låt säga att vi introducerar ännu en frukt, en som har samma lena yta som äpplet. Säg ett päron. Då kommer kunskapen inte att räcka till eftersom det eventuellt bara är vikten som skiljer ett päron från ett äpple. I det fallet skulle vi behöva ha mer data om frukterna. Det skulle exempelvis kunna vara färgen, men även äpplen kan vara lite gröna. Kanske om man tar in hur fruktens form är.

I vårt exempel med träningsdata är det dessutom en ordentlig obalans mellan storleken på siffrorna för en frukts vikt (i hundratals gram) och dess yta (binärt, 1 eller 0). Detta kan behöva balanseras beroende på hur man vill att beslutsträdet ska fungera. Den sysslan ingår i paraplybegreppet *feature engineering*[8].

Det här resonemanget är ett exempel på varför man ofta är intresserad av all data man kan komma över inom AI. Det är först efter en tid du vet vad du inte verkar behöva.

Det var det hela ☺

[8] https://en.wikipedia.org/wiki/Feature_engineering