Linnæus University

Python för yrkesverksamma

Tobias Andersson Gidlund

Agenda 👋

- Om kursen
  - Syfte, mål och framtid
- Vilka är ni och vad vill ni?
- Programmering, vad är det?
- Verktyg att använda

Om kursen

- Kursen är en introduktion till programmering i Python för yrkesverksamma
- Det innebär att vi kommer att ge er grunden för att kunna programmer i Python
- Men, för att verkligen förstå hur det fungerar så behöver ni också jobba med det
  - Kommer därför att vara några inlämningar
- Arbetet ni gör diskuterar vi även vid de träffar vi har

Syfte

- Grundsyftet är givetvis att få en förståelse för hur Python fungerar
- Det utökade syftet är att ge er tillräcklig grund att stå på för att kunna läsa maskininlärning
  - På grund av det, så är syftet vinklat något mot att kunna ta till sig ML senare
- Det innebär att vi kommer att titta på bibliotek (tillägg) för Python som senare kan användas för ML
  - Det gäller visualiering, beräkningar och liknande

Förändringar från förra omgången

- Det här är andra gången kursen ges
  - Första där man faktiskt kan få poäng för den
- Vi har lyssnat på återkopplingen från förra omgången och har gjort lite ändringar
  - Tempot upplevdes som för högt – vi har skalat ned det något
  - Grunderna togs upp för snabbt – vi har mer av grunderna under längre tid
  - Övningsuppgifter och projektuppgift kom för sent – det ska vara klart nu och tillgängligt på Moodle

Moodle

- Vi använder Moodle för att kommunicera med er
- Ni kommer åt Moodle via ert studentkonto
  - Eller gå till https://moodle.lnu.se och logga in med ert studentkonto
  - Direktlänken är https://moodle.lnu.se/course/view.php?id=68183
- Där kommer det att finnas föreläsningsanteckningar och inspelningar av föreläsningarna
- Vi publicerar nyheter och information via Moodle till er, så det är bra att kolla in lite då och då

Innehåll

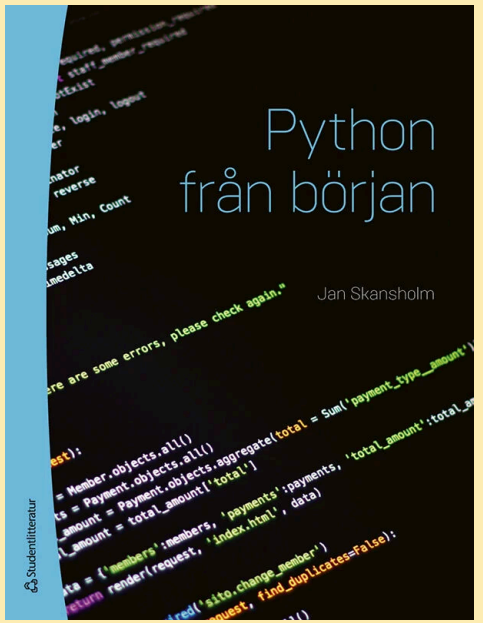
- De lärare ni kommer träffa är:
  - Tobias Andersson Gidlund
  - Alexander Gustafsson
- Grunderna i Python och programmering kommer att ges av Tobias medan Alex kommer att titta på mycket av det övriga
- Föreläsningarna spelas in och läggs sedan upp på Moodle

Innehåll i mer detalj

- Introduktion och grundläggande programmering
- Grunderna i Python (variabler, sekvens, selektion, iteration)
- Mer om Python (text, listor, funktioner, lite OO)
- De sista bitarna om Python (felhantering, I/O, set och dict)
- Data management (Pandas, NumPy)
- Visualisering (Matplotlib)
- Molnet, klient-server (bibliotek)
- Maskininlärning
- Avslutning på plats med presentation av slutprojekt

Literatur

- Obligatorisk kurslitteratur är **Python från början** av Jan Skansholm
  - Det är också den som ligger till grund för mycket av de första föreläsningarna
- Det finns mängder med bra litteratur och mycket på internet för den som vill veta mer
  - Eftersom det rör sig om grunderna för Python så fungerar det mesta
  - För det material som ligger utanför grunderna, får ni föreläsningsanteckningar samt annan information av lärarna



Python från början

Från kursplanen

- Kursplanen är indelad i tre delar
- *Kunskap och förståelse*
  - Förklara grundläggande programspråkskonstruktioner såsom variabler, typer, styrande satser och funktioner,
  - förklara egenskaper hos grundläggande datastrukturer, samt exemplifiera hur och när de bör användas, samt
  - förklara när externa bibliotek bör användas för visualisering, programmering mot molnet och maskininlärning.

Mer från kursplanen

- *Färdighet och förmåga*
  - skapa och implementera en lösning till ett givet problem i programmeringsspråket Python,
  - implementera givna algoritmer för att lösa kända typer av problem,
  - använda bibliotek för visualisering av data, samt
  - skapa och implementera enklare program som använder sig av maskininlärning

Det sista av målen

- *Värderingsförmåga och förhållningssätt*
  - resonera kring olika strategier för problemlösning och felsökning, samt
  - resonera kring, och motivera användandet av maskininlärning som en komponent när system utvecklas
- Tillsammans ska det här ge er en bra grund att stå på samt god förståelse för hur programmering med Python fungerar

Mer om er

Och vad vill ni då... 😁

- Ni är ju trots allt den viktigaste delen av kursen
- Så, vilka är ni?
  - Vad har ni för förväntningar?
  - Vad kan ni sedan innan?

Programmering 💻

Programvaruutveckling

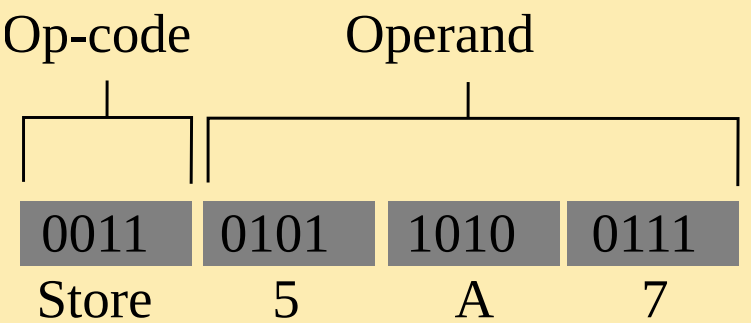
- Datorer är egentligen ganska dumma maskiner
  - De gör bara det vi säger åt dem, men de gör också exakt det vi säger åt dem
- Programmering är arbetet/konsten att göra så att datorn gör det vi vill
  - Återigen, den gör bara det vi säger åt den men exakt det
- Så har det varit sedan de första datorerna, men sättet man programmerar på i dag skiljer sig från då
- Datorn i sig förstår bara instruktioner för just den uppsättning av processorer som den har

Maskinkod

- Den lägsta nivån är att ange instruktionerna på samma språk som processorn
- Det kallas för *maskinkod*
- Maskinkod är helt enkelt en sekvens av 1 or 0
- De representerar olika instruktioner (kallade Op-code) till vilka man skickar vilket register och vilken minnesposition som ska användas
  - Register är ett litet minne nära processorn som används vid de direkta beräkningarna
- Vilka op-coder som gör vad och vilka register man har tillgång till beror helt på processorn

Exempel

- Nedan visar vi ett enkelt exempel (fiktivt) på hur det kan se ut
- I det här fallet är det en sekvens av 16 bitar



- Op-coden ”0011” innebär att ett värde ska lagras
- Därefter kommer vilket register man ska hämta värdet från (i det här fallet 5)
- De två sista talar om vilken minnesposition som värdet ska lagras i

Behov av högre nivåer

- Det är ganska uppenbart att det blir besvärligt att skriva långa program i endast maskinkod
- För att förenkla introducerades *assembleringsspråk* ganska snart
- I stället för op-coder så används engelska ord (eller förkortningar) och minnesadressering sker med hexadecimala tal i stället
- Sekvensen av 16 bitar i exemplet tidigare skulle kunna bli:

```
MOV    5, A7
```

- Språket är fortfarande väldigt hårt kopplat till processorn
- Det går dock betydligt fortare att programmera med assembleringsspråk än tidigare

Hello world i assembler...

- Följande är ett exempel på hur man kan skriva ut ”Hello world!” med hjälp av assemblerspråk för en x86-processor

```
section         .text
global          _start
_start:
    mov edx, len
    mov ecx, msg
    mov ebx, 1
    mov eax, 4
    int 0x80
    mov eax, 1
    int 0x80
section         .data
    msg         db "Hello world!", 0xa
    len         equ $ -msg
```

Ännu högre!

- Även om assembler är *enklare* än ren maskinkod, så är det inte enkelt
- I takt med att allt fler ”behövde” lära sig att skriva kod, så behövdes lättre språk
  - Och också språk som inte var lika hårt knutna till processorn
- De språk som då utvecklades kallas *högnivåspråk* och bland dem finns Python
- Man använder (oftast) engelska ord och uttryck samt vanliga matematiska beräkningar

Mer om högnivåspråk

- En omskrivning av exemplet skulle kunna bli något så enkelt som:

```
c = a + b
```

- Det gör det också möjligt att fokusera på problemet att lösa snarare än tekniska detaljer
- Nackdelen är att det inte direkt går att köra, det måste *översättas* till just den dator man sitter vid
  - Alla språk utom just maskinkod (så även assembler) måste översättas
- Jämfört med assembler så är det mycket svårare att översätta ett högnivåspråk

Översättning via kompilering

- Om man vill översätta kod hela vägen till maskinkod så kallas det *kompilering*
  - Programkoden, kallad *källkod*, läser in instruktionerna och översätter det till motsvarande maskinkod
  - Om kompilatorn (programmet som översätter) hittar fel så meddelas det och programmet kan inte kompileras
  - Kompilering kan ta tid eftersom det är mycket som ska göras
- Exempel på språk som kompileras är C, C++ och Rust
- När programmet väl är kompilerat så går det fort att köra det (*exekvera* det)

Översättning via interpretering

- Eftersom det tar lite tid och kräver en hel del beräkningar att kompilera så är en del språk tolkade, eller *interpreterade*
  - Det var speciellt vanligt i hemdatorer under åttiotalet, till exempel Commodore 64
- Det innebär att ett program läser och utför instruktionerna rad för rad i stället
- Tyvärr är det då inte möjligt att hantera fel förrän de uppstår och programmet kan därför krascha oväntat
- Programmen exekveras också långsammare än för kompilerade språk
- Exempel på interpreterade språk är Python, Basic och JavaScript

Översättning via virtuella maskiner

- Ett mellanting är att köra program via en *virtuell maskin*
- Då kompileras koden till ett speciellt *byte code*-format som är en mellanform
- När programmet ska köras så tolkas det genom att översättas till maskinkod via den virtuella maskinen
  - Det här gör programmen lika portbara som interpreterade men nästan lika snabba som kompilerade
- Exempel på språk som kör på en virtuell maskin är Java och C#

Paradigmer

- En stor skillnad från att ligga nära hårdvaran är att kod i ett högnivåspråk kan skrivas på många olika sätt
- Det är inte bara att de har olika namn på instruktionerna (*syntax*), de kan också vara från olika *paradigmer*
- Det innebär att olika språkfamiljer har olika sätt att i grunden se hur problem ska lösas
- För att ytterligare komplicera saker så kan ett språk tillhöra flera paradigmer

Olika paradigmer

- Den vanligaste paradigmen är den *imperativa* (även kallad *procedurella*)
- Det är det synsätt som är närmast algortimtänkandet
  - En sekvens av instruktioner som manipulerar data för att ge ett resultat
- Ett annat sätt att angripa programmering är *deklarativt*
  - Programmeraren beskriver problemet i sig och programmeringsspråket har generella sätt att lösa det på
  - Svårigheten är att beskriva problemet på ett sådant sätt att det kan lösas med de givna, inbyggda, algoritmerna

Andra paradigmer

- Språk kan också vara *funktionella*
  - Bygger på den matematiska funktionen, dvs att en viss indata ska ge ett visst resultat
  - Program byggs genom att sätta ihop en mängd sådana funktioner
- Den sista paradigmen som bör tas upp är den *objektorienterade*
  - Här är program uppdelade i autonoma entiteter som kommunicerar genom att skicka meddelanden till varandra
  - Det är vanligt att själva koden för att utföra logik i sig är imperativ, men sätts ihop objektorienterat

Multiparadigm

- I princip alla programmeringsspråk i dag är *multiparadigmspråk*
- Det innebär att de tar in element från flera olika paradigm
- Oftast har de en grund, men lägger till andra
  - Python är i grunden imperativt men kan användas objektorienterat och vissa saker kan även göras med funktionell programmering
- Vissa språk är multiparadigmspråk från grunden, medan andra har byggt på under årens lopp

Python 🐍

- Python är alltså ett interpreterat språk men det har också flera delar som är skrivna i C
  - Det gör att till exempel beräkningar och datastrukturer kan bli nästan lika snabba om kompilerade språk
- Som nämdes tidigare är det ett multiparadigmspråk med rötterna i den imperativa paradigmen
- Första versionen av Python kom redan 1991, så det är inte ett så nytt språk
- Det utvecklades först av Guido van Rossum och ja, namnet på språket är en referens till *Monty Python*

Mer om Python

- Språket är i sig känt för att vara förhållandevis enkelt att ta till sig
- Det utvecklas kontinuerligt och är inne på version 3.x
  - I dag utvecklas det av en mängd utvecklare ledda av en styrkommittée
- En anledning till att det har tagit fart den senaste tiden är att det används mycket inom maskininlärning och AI
  - En stor mängd verktyg har utvecklats för att göra det möjligt
  - Eftersom språket i sig är enkelt att lära sig, så behövs inte samma djupa kunskap för att använda det

Zen of Python

- För att språket ska fortsätta att vara lättillgängligt så finns en grundsyn kallad *Zen of Python*
- Den har följande riktlinjer:
  - Beautiful is better than ugly
  - Explicit is better than implicit
  - Simple is better than complex
  - Complex is better than complicated
  - Readability counts
- Trots det finns det tillfällen när nya egenskaper läggs till språket som komplicerar det hela...

Vad gör Python unikt?

- En enkel syntax är en viktig del i att språket framstår som enkelt
- Indentering i stället för klamrar
  - För att gruppera kod använder många språk klamrar {} men Python kräver i stället indragna rader
  - Det gör koden både mindre full av ”konstiga” tecken och lättare att se och läsa
- Det finns en tydlig guide för hur saker ska göras, kallad *Python Enhancement Proposals* eller PEP
  - Där en är *PEP 8* - Style Guide for Python Code som beskriver hur Pythonkod ska se ut

PEP 8

- En viktig anledning till att det finns en stilguide är att man alltid ska känna igen sig i koden
- Exempel på sådant som rekommenderas här är:
  - Att indentering ska göras med mellanslag (inte tabb)
  - Indenteringen är fyra tecken bred
  - Tomrum runt literaler och matematiska tecken
  - Hur variabler ska namnges
- Givetvis finns det tillägg till olika editorer för att hjälpa dig som programmerare att göra rätt

Interaktivt läge

- Python kan också köras i ett så kallat *interaktivt läge*
  - På engleska *interactive mode* eller *read-eval-print loop* (REPL)
- Det läget fungerar som en möjlighet att skriva in och testa olika kommandon och uttryck
  - Utan att skapa en fil som man läser ifrån
- För att starta i interaktivt läge, skriver du `python3` (eller `py` i Windows) i en terminal
- Vid en prompt (`>>>`) skriver man helt enkelt in Pythonkod och trycker return för att utföra den

Exempel

- Ett kortare exempel på hur det kan se ut:

```
Python 3.11.7 (main, Dec 15 2023, 18:12:31) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 3 + 4
7
>>> print('May the Force be with you')
May the Force be with you
```

- För att avsluta, tryck `ctrl-d` (eller `ctrl-z` och return på Windows)

Var används Python?

- Som tidigare sagts så används Python mycket inom ML och AI
  - Till exempel med TensorFlow och PyTorch
- Det används också som *skriptspråk* för att utöka beteende hos program som Blender, 3ds Max, GIMP och Inkscape
- Python är en viktig komponent i MacOS och Linux (mycket i båda operativsystemen bygger på Python)
- Utbildningshård- och mjukvara har ofta kopplingar till Python (till exempel för Raspberry Pi)
- Givetvis även för att göra skrivbords- och webbapplikationer

Installera och använda Python

Installera Python

- Python kan installeras på många olika sätt
  - Kanske har du det redan installerat (vanligt på Linux och MacOS)
- Traditionellt installeras Python som ett program som körs via en terminal (kommandoprompt)
  - Det är då också vanligt att man använder någon texteditor eller IDE för att skriva programmen
- Det är också möjligt att köra Python via ett webbgränssnitt
  - Både lokalt eller via någon av alla de tjänster som finns
  - Då finns det ofta något sätt att skriva programmen direkt i webbläsaren

Installera från källan

- Det är naturligtvis möjligt att installera Python direkt från källan
  - Alla officiella filer finns via https://www.python.org/
- Fördelen att använda den här sidan är naturligtvis att man får den senaste versionen
- Det är dock en hel del manuellt arbete involverat i att installera Python så här
- Dessutom kan den eventuellt störa den installation som redan finns (om man kör MacOS eller Linux)

Installera Anaconda ⭐

- Det bästa sättet är förmodligen att installera en distribution av Python som heter *Anaconda*
  - Finns på https://www.anaconda.com/
  - Klicka på **Free Download** och välj din plattform
- Fördelen här är att man får en miljö som är skild från någon annan på datorn
- Det blir också enklare att installera olika paket
  - Det vill säga bibliotek för att utöka Python
- Sist men inte minst ger det oss en möjlighet att köra Python via en webbläsare (lokalt, mer senare)

Installera ytterligare paket

- En fördel med Anaconda är att en stor mängd av de paket, bibliotek och annat som behövs, redan är installerat
  - NumPy, Pandas, matplotlib med flera
- Dessutom är det enkelt att installera ytterligare paket grafiskt via *Anaconda Navigator*
  - Gå till **Environments** och sök efter ej installerade paket och välj att installera dem
  - Det går också att i terminalen använda `conda install` följt av paketnamnet
- Om man använder Python direkt från källan får man lära sig att använda `pip install` i ställlet och det kommer att behövas mer

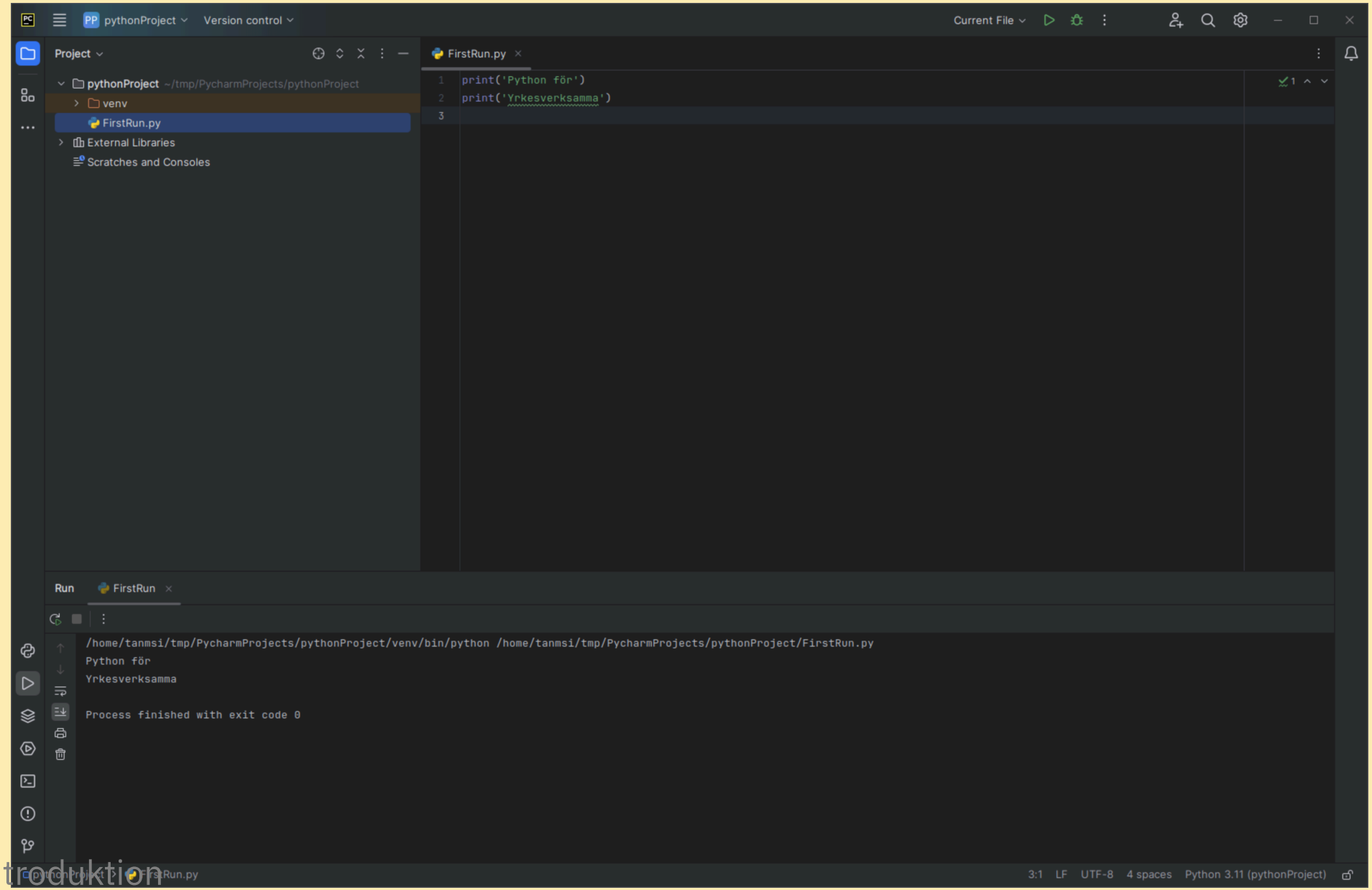
Texteditor eller IDE

- För att skriva sina program behöver man ett textverktyg som hjälper dig att skriva kod
  - Man kan givetvis göra det i vad som helst (Anteckningar) så länge programmet sparar ned en ren textfil
- En *texteditor* är ett avancerat verktyg för att hantera text och i viss utsträckning programmeringsprojekt
- En IDE (Integrated Development Environment) är ett riktigt avancerat textverktyg som fokuserar på programmeringsprojekt
- Det finns en stor mängd olika sådana verktyg, från riktigt enkla till väldigt avancerade
  - Med Python följer till och med en med, IDLE

PyCharm

- Ett exempel på en IDE som är utvecklad just för Python är *PyCharm* från JetBrains
  - Community Edition är gratis och finns på https://www.jetbrains.com/pycharm/
  - Gå till **Download** och se till att rulla ned till Community Edition
- Du behöver fortfarande ha Python installerat, men du får mycket hjälp med att skapa och hantera Pythonprojekt
- Ibland kan verktyg som det här kännas lite *för* stort...

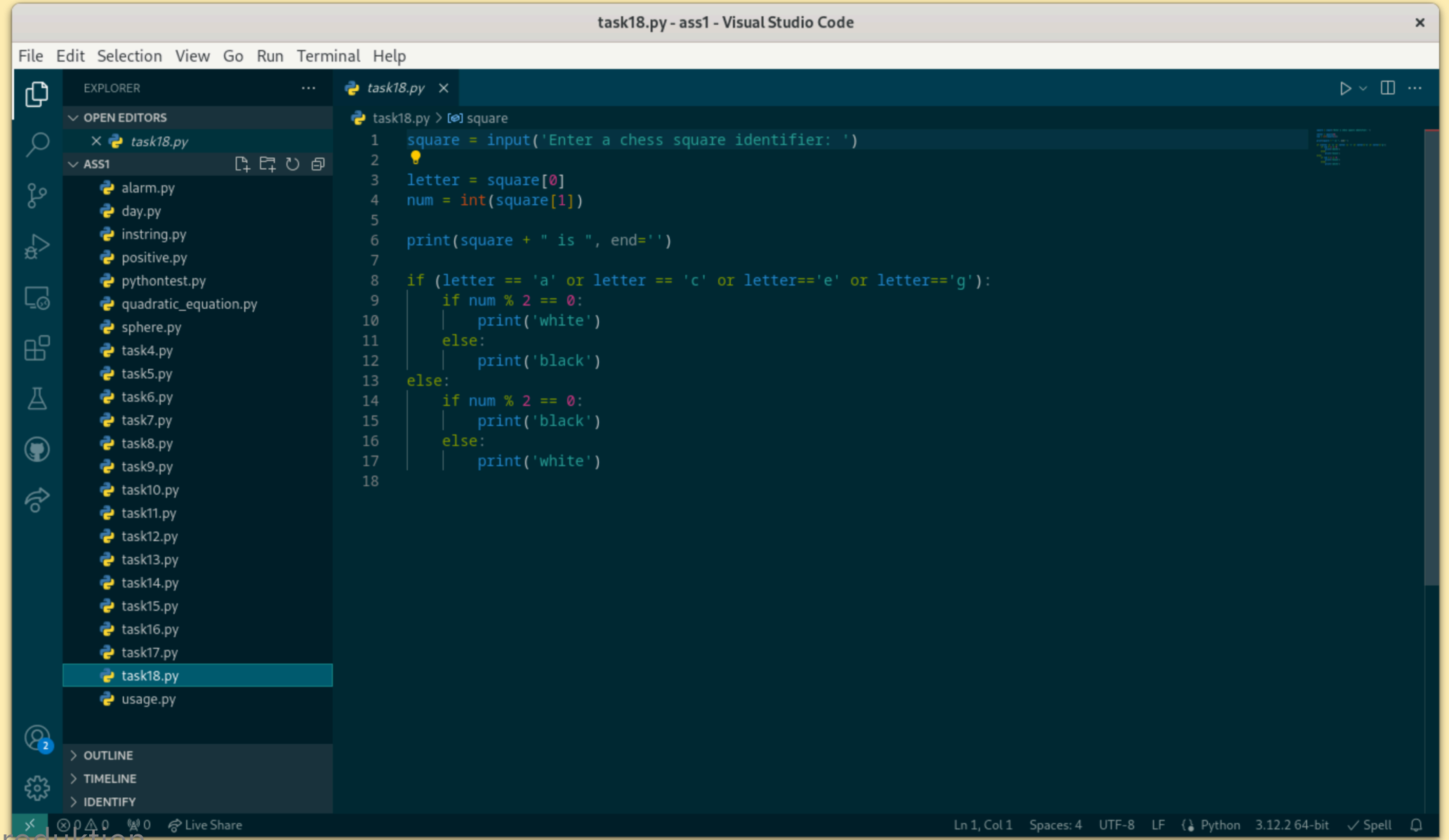
PyCharm som kör



Visual Studio Code ⭐

- En texteditor som nästan går hela vägen till en IDE är Microsofts *Visual Studio Code*
  - Som inte ska förväxlas med *Visual Studio* som är något annat...
  - Visual Studio Code hittar du på https://code.visualstudio.com/
  - Programmet är helt gratis
- Det har bra stöd för Python via så kallade *extensioner*
  - För just Python, see https://code.visualstudio.com/docs/languages/python
- Visual Studio Code är inte fullt lika tung som PyCharm, men har ändå fullt stöd för Python

VS Code som kör



Ett par saker att tänka på

- För båda verktygen gäller att Python i sig måste vara installerat först
  - Antingen via Anaconda eller något annat sätt
- Sedan måste man vara noga med att tala om att det är just den Python som du har installerat som används
  - För Visual Studio Code så tittar du nere i högra hörnet när du har en Pytonfil öppnad – klicka på versionen efter **Python**
  - I PyCharm får du frågan när du skapar ett nytt projekt
- Anledningen till det här är att det annars kan vara svårt att veta i vilken Python man har installerat paket och bibliotek

Köra i webbläsaren

- Som har nämnt så är det möjligt att köra Python direkt i en webbläsare också
- Det finns en stor mängd tjänster som tillhandahåller det, en del bra andra sämre
- Ofta behöver man kunna skapa ett konto och ladda upp eller ned filer, men ibland behöver man ett betalkonto för det
- Boken rekommenderar *PythonAnywhere* som kan fungera till en början, så snart man kommer till visualiseringar och liknande så är det för begränsat
  - När NumPy, Matplotlib eller liknande ska användas

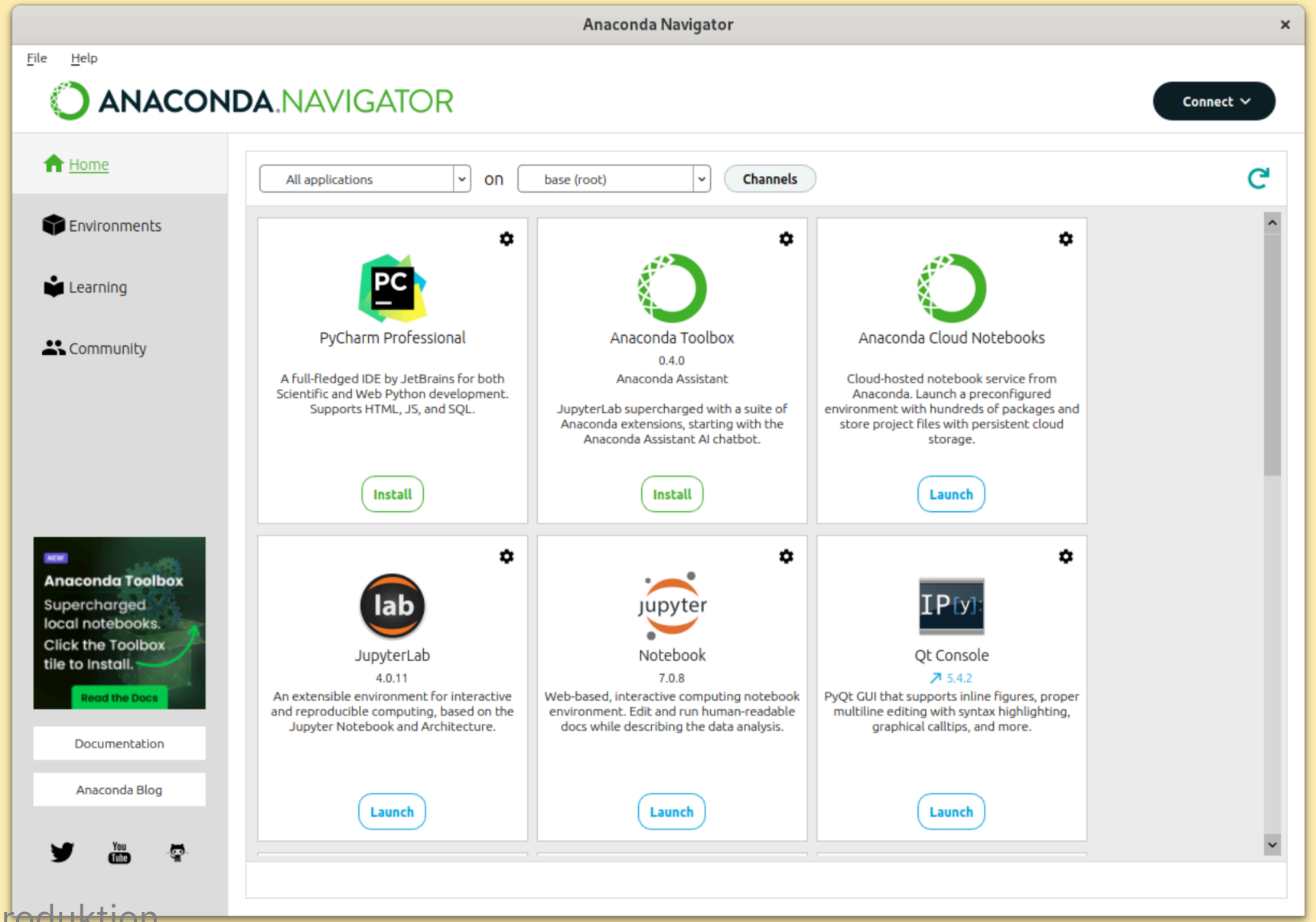
JupyterHub (Notebooks)

- Det mest använda sättet att köra Python i en webbläsare är genom *Jupyter*
  - Förr *Notebooks* (vilket fortfarande är formen), numera *Hub*
- Jupyter finns på https://jupyter.org/
  - Mjukvaran är gratis att ladda ned
- Det är också möjligt att testa via *Try it in your browser*
  - Tyvärr kan det då bli svårt med att hantera filer (man får ladda ned och upp)

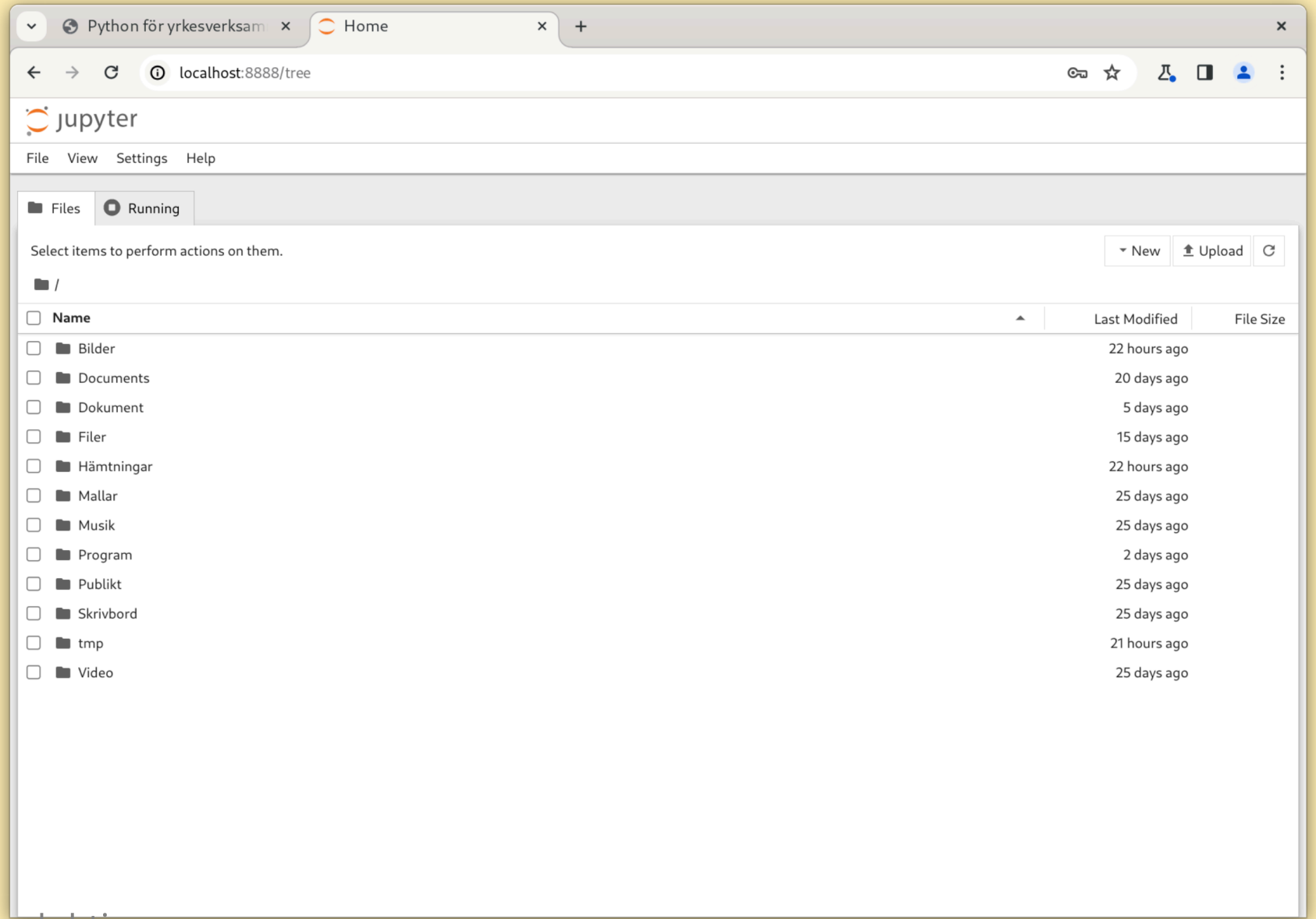
Anaconda to the rescue! ⭐

- Om man har installerat Python via Anaconda så får man faktiskt med en lokal installation av Jupyter Hub
- Starta **Anaconda Navigator** och under **Home** ska det finnas *JupyterHub*
  - Starta genom att klicka *Launch* och Jupyter kommer att öppnas i en webbläsare
- Du kommer få upp en lista över din hemkatalog, välj ett ställe att lägga dina filer och öppna en ny Jupyter Notebook

Jupyter i bilder (Navigator)



Jupyter i bilder (hemkatalog)



Jupyter i bilder

Python för yrkesverksam | Home | test

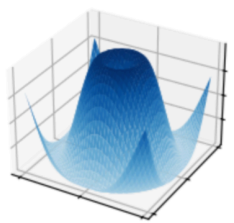
localhost:8888/notebooks/tmp/notebook/test.ipynb

jupyter test Last Checkpoint: 18 seconds ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

[1]:
```
import matplotlib.pyplot as plt
import numpy as np

from matplotlib import cm

plt.style.use('_mpl-gallery')

# Make data
X = np.arange(-5, 5, 0.25)
Y = np.arange(-5, 5, 0.25)
X, Y = np.meshgrid(X, Y)
R = np.sqrt(X**2 + Y**2)
Z = np.sin(R)

# Plot the surface
fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
ax.plot_surface(X, Y, Z, vmin=Z.min() * 2, cmap=cm.Blues)

ax.set(xticklabels=[],
       yticklabels=[],
       zticklabels=[])

plt.show()
```



[ ]:

Arbeta med Jupyter

- Jupyter fungerar lite annorlunda jämfört med att jobba i en editor
- Koden läggs in i celler som man sedan ”kör” genom att trycka på Play-ikonen
  - Den cell man vill köra måste vara markerad
  - Det går också att trycka `shift-enter` i cellen för att exekvera den
- Cellerna är sammankopplade, vilket betyder att om man t.ex. deklarerar en variabel i en cell så finns den i alla celler under

En möjlighet

- På Linnéuniversitetet har vi en egen Jupyter som kör på https://jypyter.lnu.se/
  - Notera den “intressanta” stavningen...
- Om ni skaffar ett studentkonto så kan vi lägga till er där så att ni kan använda den
  - Se först till att få ett studentkonto
  - Skicka sedan ett mail med användarnamnet till Tobias
  - Vänta lite
- Vanligtvis använder vi servern för våra andra introduktionskurser i Python

What’s next?

Nästa steg

- Det absolut viktigaste är att nu se till att man har tillgång till Python
- Den rekommenderade vägen är att installera Anaconda och använda Visual Studio Code
- Vi rekommenderar också att ni tittar på och testar Jupyter
  - Gärna via Anaconda
- Det rekommenderas att man skaffar boken, eller annan lämplig litteratur