Introduktion till Python

1 Inledning

Python är ett s.k. interpreterande högnivåspråk. Språket skapades 1990 av Guido van Rossum och är relativt lätt att använda. Koden är lättläst och språket används mer och mer. Det är också en bra ingång till andra språk. Python är fri programvara och utvecklas hela tiden. Python 3 är nu standard så man bör sträva efter att använda den versionen.

Python har en egen hemsida `https://www.python.org/`

Man kan läsa om Guido van Rossum och lite om bakgrunden och syftet med Python på wikipediasidan.

`https://en.wikipedia.org/wiki/Guido_van_Rossum`

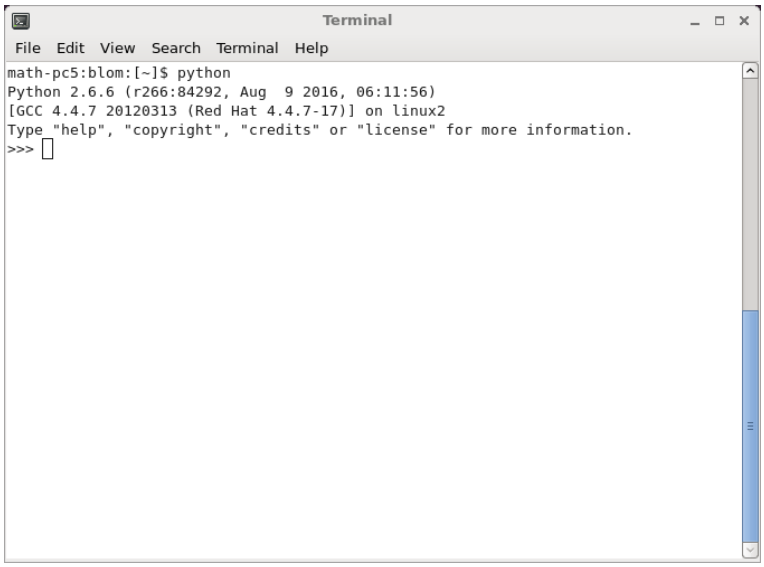
2 Starta Python

2.1 I Labsalen

Vid en WINDOWS-dator startar man Python genom att klicka på WINDOWS-ikonen allra längst ner till vänster och väljer alternativet Python (Man måste scrolla ner – i listan av program. Programmen är sorterade i bokstavsordning).

Vid en LINUX-dator öppnar man först ett terminalfönster. Man går in under Applications (högst upp till vänster) och väljer System Tools och sedan Terminal[1]. Sedan skriver man `python` i terminalfönstret. Det ser ut ungefär så här:

[1]Man kan också öppna ett terminalfönster genom att högerklicka med musen över den lila bakgrunden och välja alternativet Open Terminal i menyn som kommer upp.



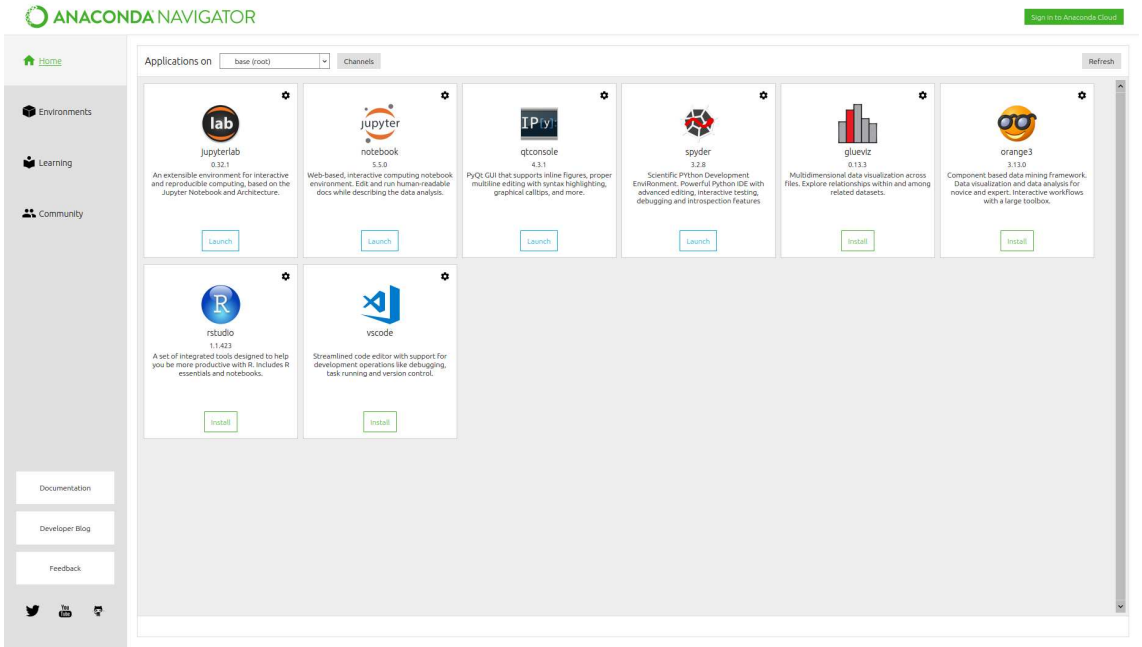
På sista raden står det `>>>`. Det är själva promten som visar att PYTHON är redo att ta emot kommandon.

Vi kommer standardmässigt att använda två paket som heter `NumPy` och `Matplotlib`, som gör att vi får tillgång till enkla matematiska funktioner, såsom `sin`, `cos`, `sqrt` (kvadratroten) etc, matematiska konstanter som t.ex. π samt möjlighet att rita grafer och andra figurer. Paketen finns inte installerade tillsammans med PYTHON 3 i labsalarna därför kommer vi istället att ladda hem PYTHON till egen dator.

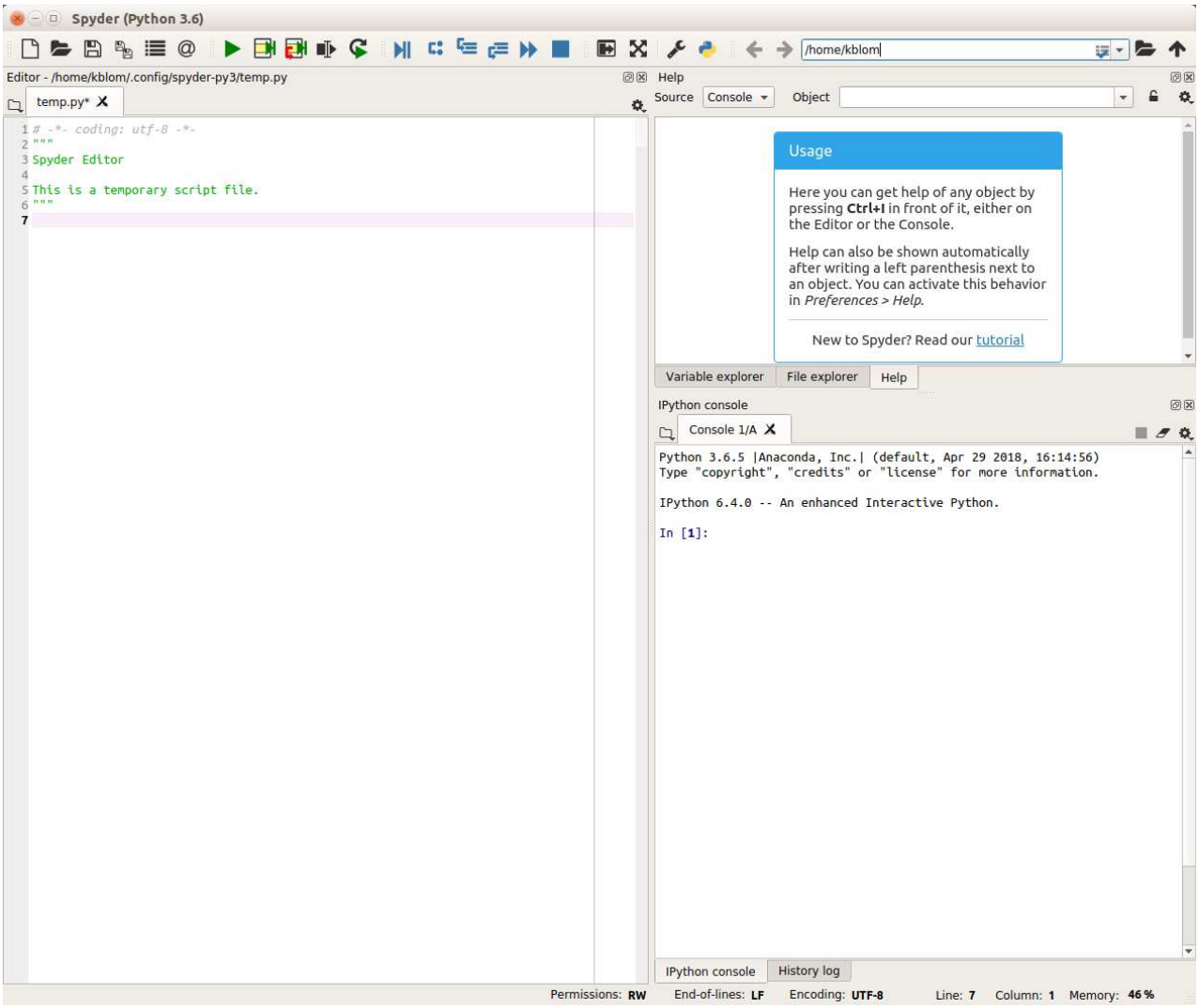
2.2 På egen dator

Ladda hem programmet ANACONDA från `https://www.anaconda.com/download/`. Vi kommer att använda oss av programmet SPYDER som vi når från ANACONDA-NAVIGATOR.

Starta först ANACONDA-NAVIGATOR. Vi får upp ett fönster som ser ut ungefär så här:



Välj sedan SPYDER (klicka på Launch). Arbetsmiljön i SPYDER kommer upp på skärmen.



Det vi ser är SPYDERs desktop eller arbetsmiljö. De olika fönstren i arbetsmiljön har namn. Fönstret till vänster heter Editor och fönstret längst ner till höger heter Console. Där kommer vi att skriva kommandon som PYTHON sedan utför. Fönstret uppe till höger heter Help och innehåller diverse information bl.a. om SPYDER och PYTHON.

3 En enkel beräkning och några grafer

Här följer några exempel så att vi snabbt kommer igång och ser lite resultat. Följ gärna med vid datorn och knappa in efter hand och se vad som händer.

Vi kommer att behöva konstanten π från paketet `NumPy` och så småningom också kommandon från paketet `Matplotlib` för att rita figurer. Det enklaste är att skriva raden

```
from pylab import *
```

allra först i Console. Det importerar både `NumPy` och `Matplotlib`. Tryck sedan på enter (↩).

Exempel 1. Beräkna volymen av ett klot med radien $r = 3$ cm. Volymen ges av $V = \frac{4}{3}\pi r^3$.

Vi utför vår beräkning i Console. Först inför vi en variabel `r`, för radien, som vi ger värdet 3.

```
r=3
```

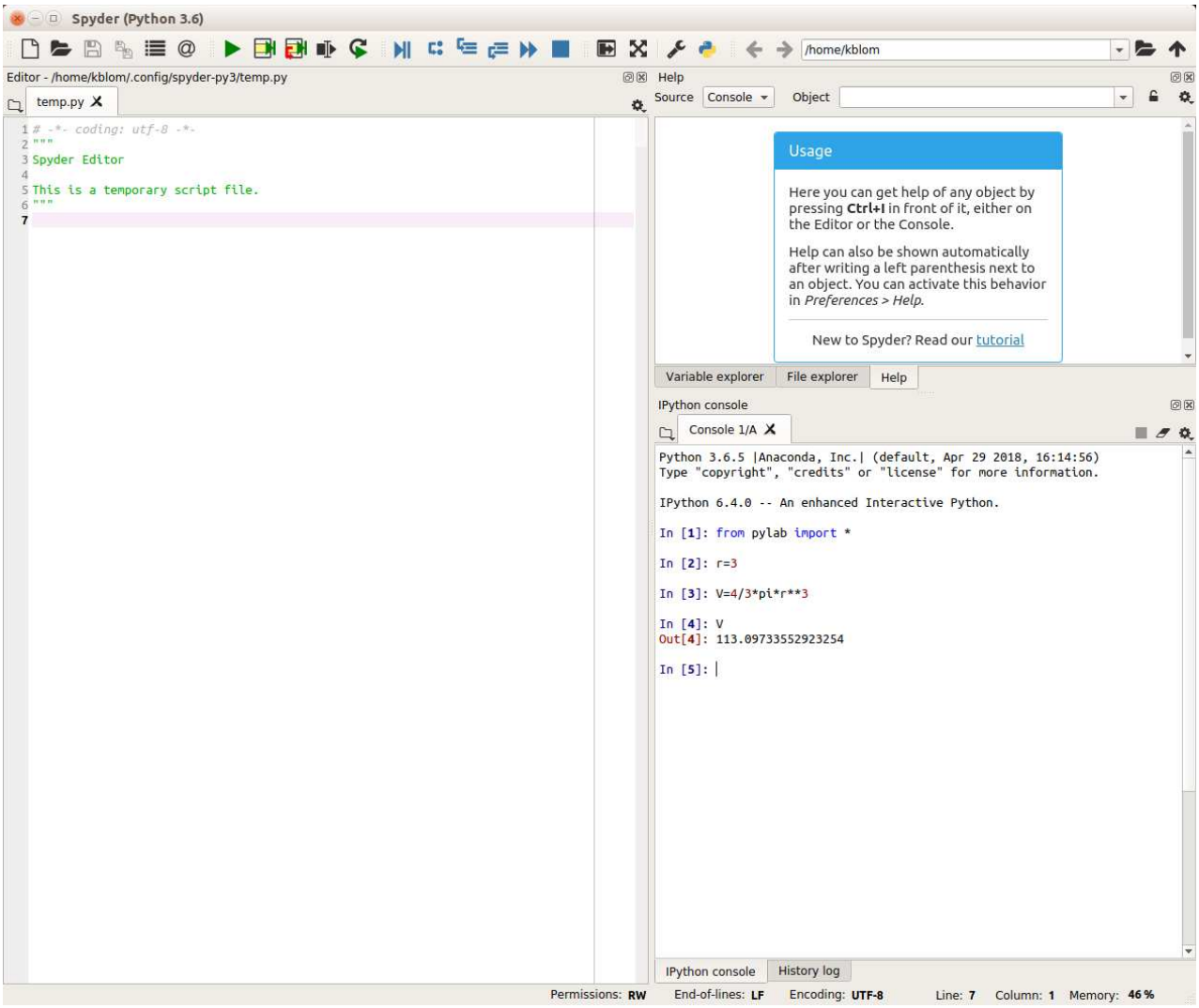
Därefter beräknar vi volymen enligt formeln (`pi` ger en approximation av konstanten π) och låter variabeln `V` få detta värde.

```
V=4/3*pi*r**3
```

De dubbla asteriskerna `**` används för upphöjt till i PYTHON.

Sedan skriver vi ut värdet på variabeln `V` genom att bara ange variabelnamnet.

```
V
```



Det vi har skrivit i Console står efter `In [1]:`, `In [2]:` osv. Det som står efter `Out[4]:` har skrivts av programmet.

Ett variabelnamn skall börja med en bokstav (`a-z`, `A-Z`), eller ett understrykningstecken (`_`) därefter får vi ha bokstäver (`a-z`, `A-Z`), siffror (`0-9`) och understrykningstecken (`_`). PYTHON skiljer på stora och små bokstäver. Det finns några sk. reserverade ord som man inte kan använda som variabelnamn eftersom PYTHON använder dessa till annat. (Om man råkar använda ett reserverat ord som variabelnamn svarar PYTHON med en felutskrift).

Vi kan se våra variabler och deras värden om vi klickar på fliken Variable explorer i Help-fönstret (fliken finns till vänster ovanför Console). Där ser vi också de matematiska konstanter vi laddade in när vi importerade `NumPy`. Dessa har definierats som variabler av PYTHON.

Uppgift 1. Beräkna arean av en cirkelskiva med radien $r = 4$ cm. Arean ges av $A = \pi r^2$.

Exempel 2. Rita grafen av $f(x) = \sin(x) + 0.3\sin(4x)$ för $0 \leq x \leq 4\pi$.

Först gör vi en lista eller radvektor `x` av x-värden mellan 0 och 4π, med funktionen `arange`

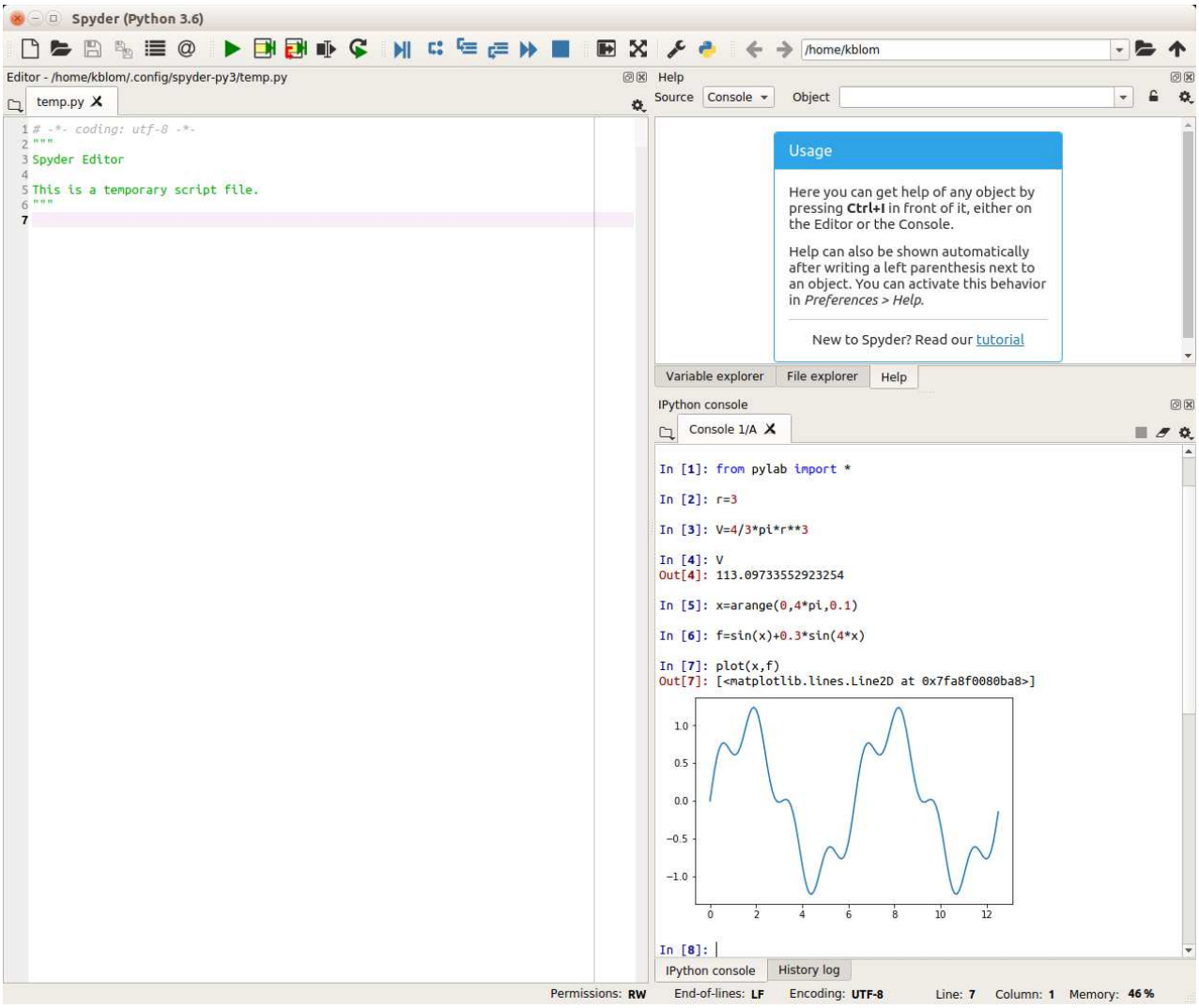
```
x=arange(0,4*pi,0.1)
```

som vi skriver i Console.

Närmare bestämt får vi värdena 0, 0.1, 0.2, 0.3, $\cdots$, 12.5, dvs. värden med start i 0, steget 0.1 och slut så nära upp mot 4π som möjligt.

Därefter gör vi en lista eller radvektor `f` med $f(x)$-värden för varje x-värde i `x` och ritar upp grafen med `plot`.

```
f=sin(x)+0.3*sin(4*x)
plot(x,f)
```



Vi kan använda uppåtpil (↑) för att komma till en rad vi skrivit tidigare Om vi vill kan vi gå längs raden med vänster- och högerpilarna (←), (→) och redigera raden. När raden ser ut som vi vill trycker vi på enter (↩).

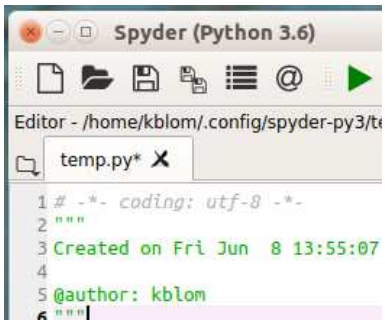
Uppgift 2. Rita grafen till $f(x) = \sin(x) + 0.3\sin(5x)$ över intervallet $0 \leq x \leq 4\pi$.

4 Pythonscript

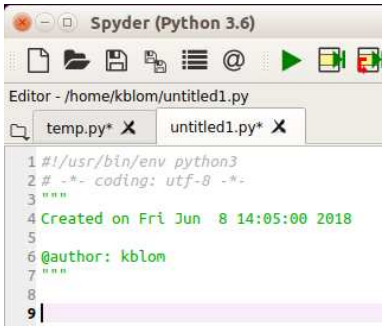
För att slippa skriva om sina kommandon, eller bläddra med uppåt- och nedåtpilar (↑), (↓) i Console så brukar man skriva ett script. Ett pythonscript är en textfil som innehåller det man skulle kunna skriva direkt i Console och som utförs när man klickar på den gröna pilen över Editorn.

Exempel 3. Rita graferna av $f(x) = \sin(x)$ och $g(x) = \sin(4x)$ för $0 \leq x \leq \pi$. Sätt rubrik och text på axlarna.

Vi skriver nu hela vårt program i Editorn och börjar med att öppna en ny skrivarea genom att klicka på symbolen som visar ett papper med invikt hörn (högst upp till vänster). Vi kan också klicka på krysset brevid temp.py.



Vi får en ny skrivyta i Editorn där vi ska skriva in vår kod.



De rader som redan står där (det som är skrivet med grå och grön text) låter vi vara kvar.
Vi använder funktionen `linspace` för att få 50 punkter jämnt fördelade mellan 0 och π, då blir graferna jämna och snygga.

```
x=linspace(0,pi)
f=sin(x)
g=sin(4*x)
```

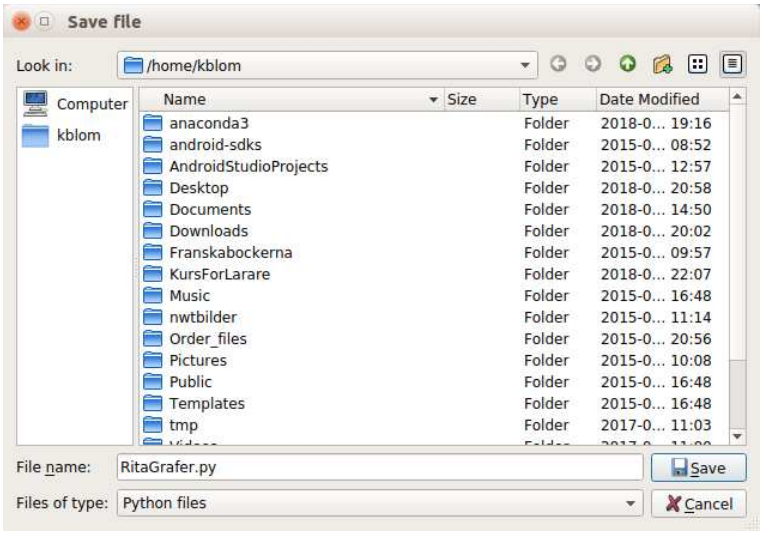
Vi ritar båda graferna samtidigt med `plot`, både paret `x`, `f` och paret `x`, `g`.

```
plot(x,f,'green',x,g,'red')
```

För att skilja graferna åt gjorde vi $\sin(x)$-grafen grön `'green'` och $\sin(4x)$-grafen röd `'red'`.
Vi sätter text på axlarna och rubrik samt lägger på ett rutnät med

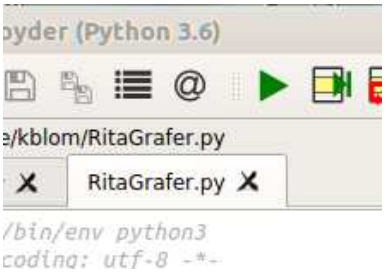
```
xlabel('x')
ylabel('y')
title('sin(x) och sin(4x)')
grid()
```

Texterna inom apostrofer (`' '`), t.ex. `'green'` och `'x'`, är s.k. textsträngar.
Spara kan vi göra genom att klicka på disketten ovanför Editorn.

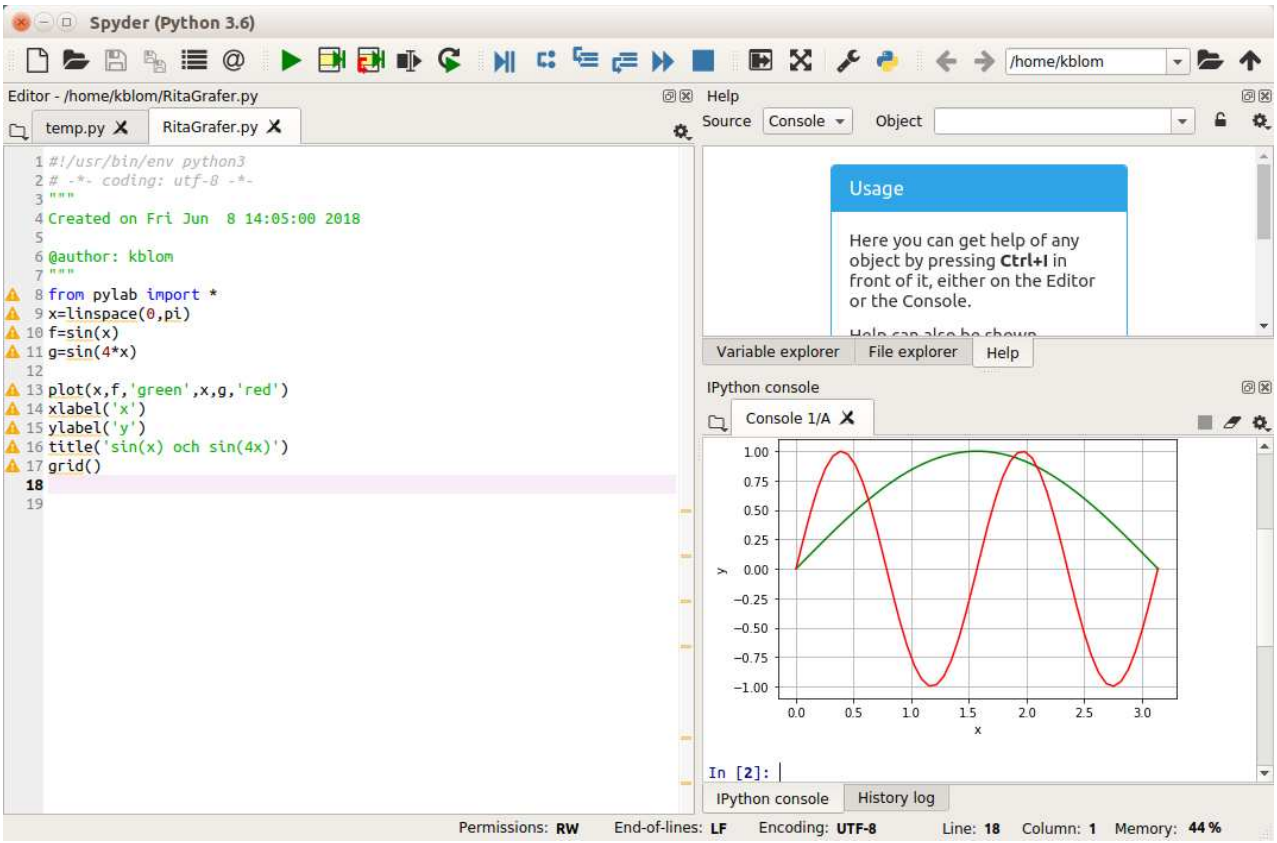


Vi låter filnamnet sluta på `.py` och väljer filtypen Python Files. I rutan högst upp står var någonstans vi valt att placera vår fil.

Vi kör programmet genom att klicka på den gröna pilen ovanför Editorn.



Det ser ut ungefär så här när vi är klara



När pythonscriptet körs kommer Spyder att utföra rad för rad (med start från första raden).

5 Lite programmering

I Python finns repetitions- och villkorssatser. Vi nöjer oss för tillfället med att se på en repetitionssats, en `for`-sats, som vi använder för att beräkna en summa i följande exempel.

Exempel 4. Beräkna summan $s = 3 + 4 + 5 + \cdots + 52$

Vi använder `arange` och skapar en lista med talen $3, 4, \ldots, 52$ som vi sedan löper igenom ett tal i taget i en `for`-sats. Anropet `arange(3,53,1)` ger värdena $1, 2, \ldots, 52$, dvs. värden med start i 3, steget 1 och slut ett heltal< 53 så nära 53 som möjligt. Om man anropar `arange` med bara två argument, t.ex. `arange(3,53)`, får man automatiskt steglängden 1.

Vi gör ett pythonscript med programkoden

```
s=0
for i in arange(3,53,1): s=s+i
```

Första satsen `s=0` är en tilldelningssats, variabeln `s` ges värdet 0. Andra satsen `for i in arange(3,53)` är en repetitionssats, den utför satsen som följer efter kolonet `:`, först för `i=3` sedan för `i=4` osv., ända tills satserna slutligen utförs för `i=52`. Första gången, dvs. då `i=3`, ges `s` ett nytt värde som är summan av gamla värdet på `s`, dvs. 0, och värdet på `i`, dvs. 3. Alltså kommer `s` ges värdet 0+3=3. Andra gången, dvs. då `i=4`, ges `s` återigen ett nytt värde som är summan av gamla värdet på `s`, dvs. 3, och värdet på `i`, dvs. 4. Alltså kommer `s` ges värdet 3+4=7. Så här fortsätter det ända tills `i=52` och `s` får sitt slutgiltiga värde. (Man måste ange 53 som sista värde och inte 52. Man får närmsta heltal < 53).

Vi skriver vårt program i Editorn.

Vi låter de två första raderna stå kvar. Om vi tar bort raden

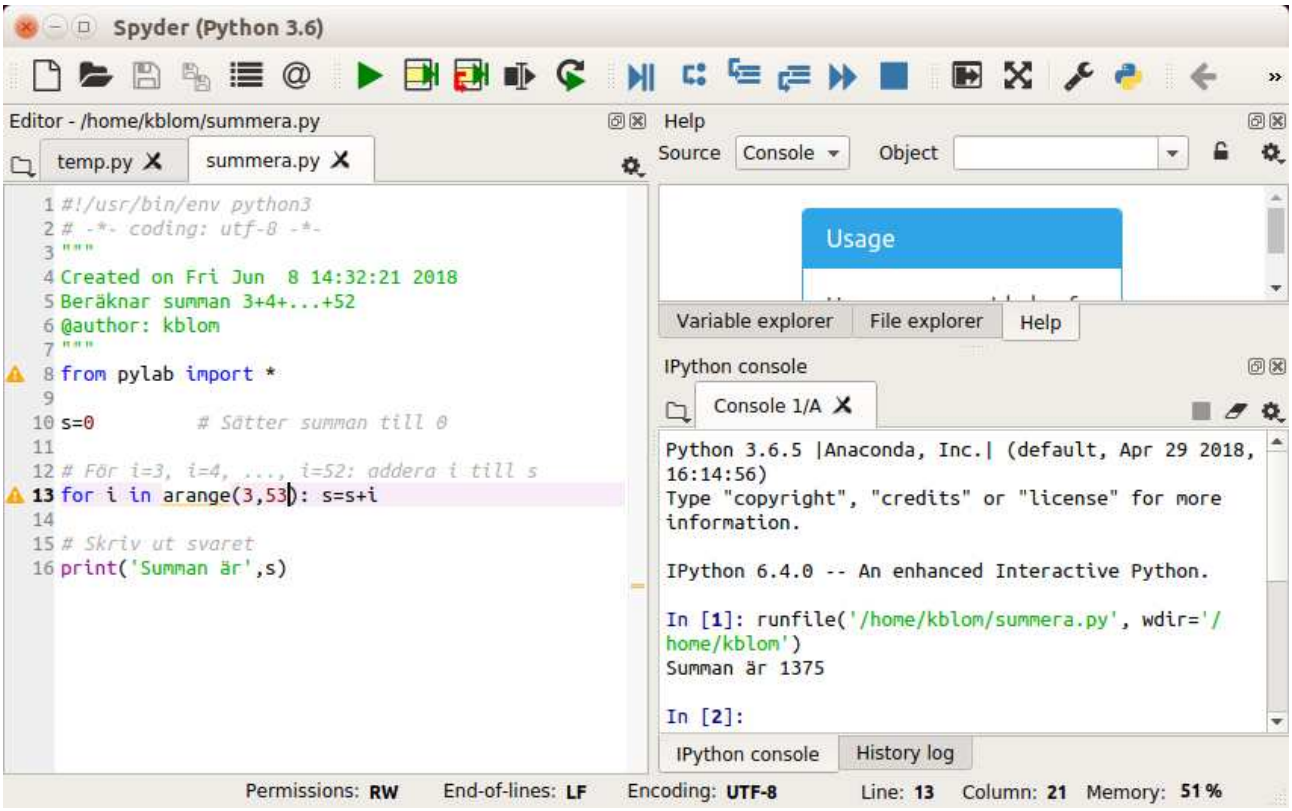
```
# -*- coding: utf-8 -*-
```

kan vi få problem med att PYTHON ger felutskrifter när vi använder de svenska bokstäverna, å, ä, ö, Å, Ä, Ö i utskrifter och kommentarer.

Vi kan ta bort den gröna texten som står där om vi vill – eller infoga en egen kommentar på raderna mellan de tre gröna citattecknen """.

Vi skriver lämpliga kommentarer i programkoden. Kommentarer börjar med #. Det som står efter # ignoreras av PYTHON när vi kör programmet.

Vi gör också lämplig utskrift med kommandot `print`, först textsträngen `Summan är` och sedan summans värde. Så här ser det när vi skrivit in och kört vårt program.



I matematik skriver man gärna summan $3+4+5+\cdots+52$ med beteckningen

$$\sum_{i=3}^{52} i$$

Uppgift 3. Skriv ett program som beräknar summan

$$s=\sum_{i=1}^{5} i^2 = 1^2+2^2+3^2+4^2+5^2$$

6 Egendefinerade funktioner

Man kan skapa egna funktioner i PYTHON.

Exempel 5. Vi vill hitta ett nollställe till funktionen $f(x) = x^3 - \cos(x)$.

Det finns en funktion i paketet `scipy.optimize` som heter `newton` som hittar nollställen. För att använda `newton` måste vi först beskriva vår funktion och det gör vi som en egendefinerad funktion enligt

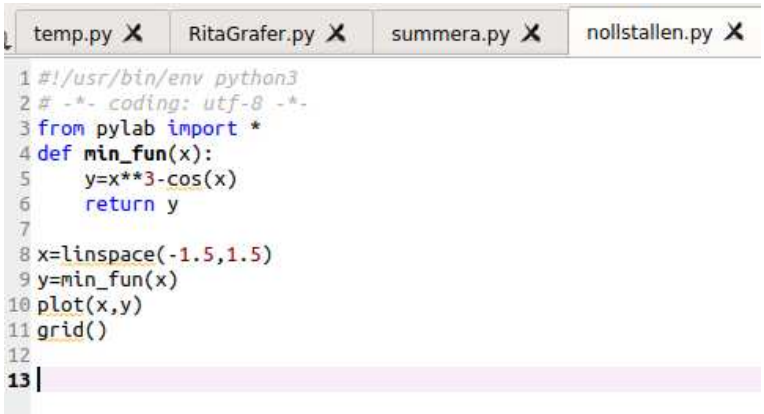
```
def min_fun(x):
    y=x**3-cos(x)
    return y
```

där `y` är funktionens värde (utdata), `x` är funktionens parameter (indata) och `min_fun` är funktionens namn (som vi själva valt). Först skrivs ordet `def` sedan namnet på funktionen (namnet hittar man på själv). Raden avslutas med kolon (`:`).

Observera att den andra och tredje raden börjar med fyra blanksteg. Det är viktigt att man skriver raderna i funktionen en bit in från vänstermarginalen, annars vet inte PYTHON var funktionen slutar någonstans. Man måste inte ha just fyra blanktecken, men minst ett och alla rader dras in lika mycket.

På sista raden står det `return y`. Det markerar att värdet på `y` ska skickas tillbaka till anroparen av funktionen.

Vi skriver in funktionen i editorn

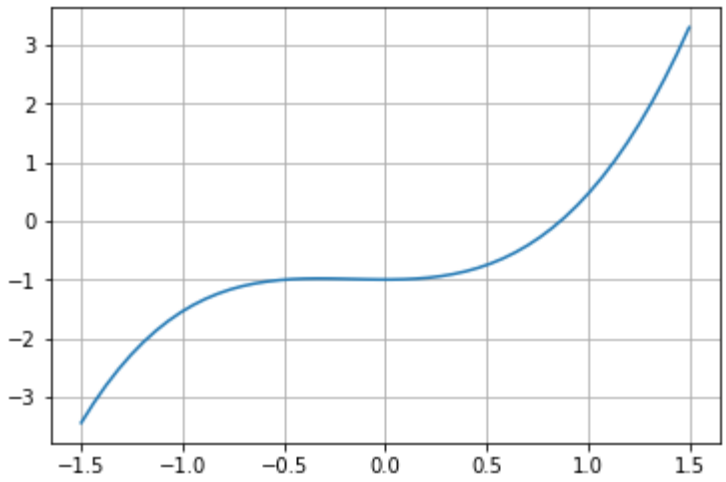


Vi ritar grafen genom att skriva till raderna

```
x=linspace(-1.5,1.5)
y=min_fun(x)
plot(x,y)
grid()
```

efter funktionen.

När vi testkör programmet ser vi att vi har ett nollställe nära $x = 1$ och använder `newton` för att beräkna nollstället noggrant.



Först måste vi importera funktionen `newton`,

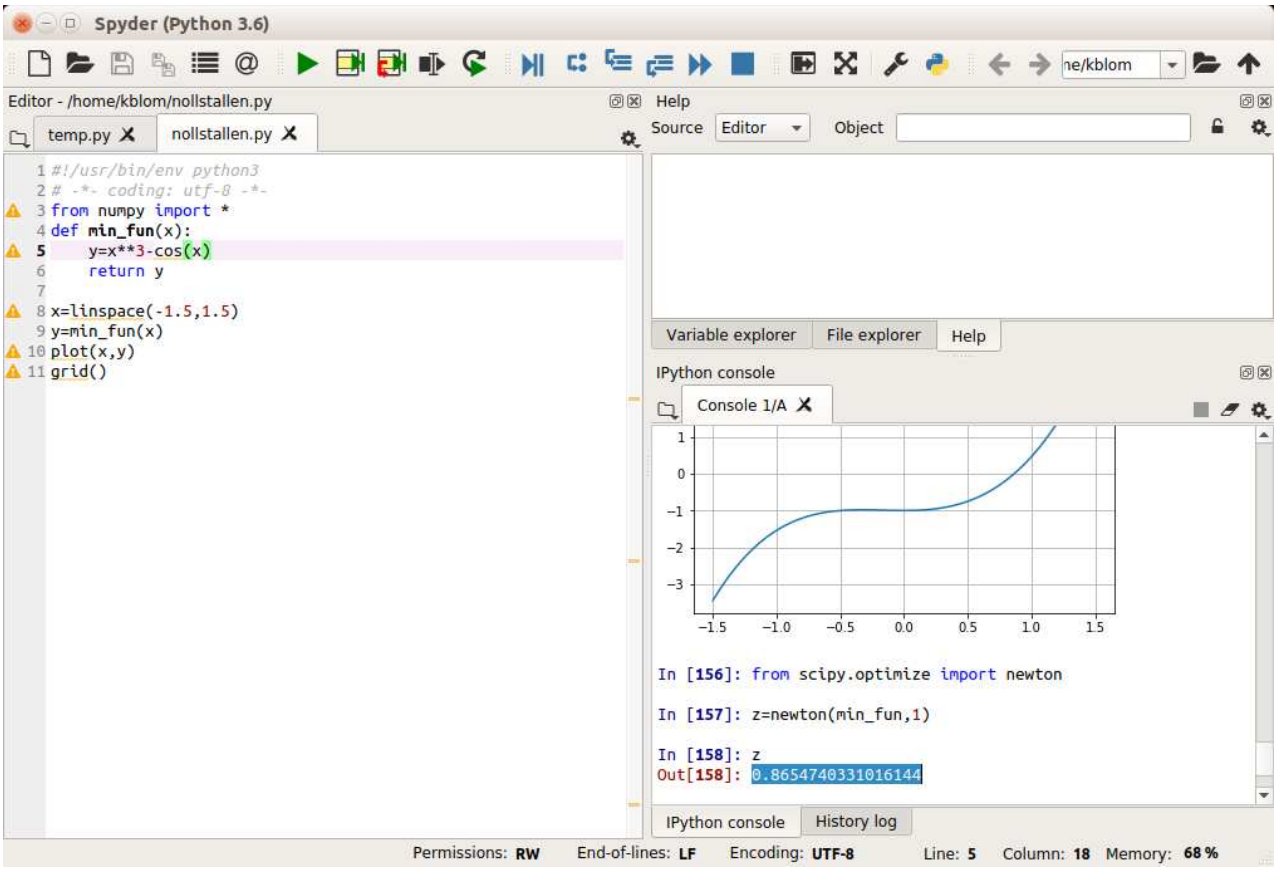
```
from scipy.optimize import newton
```

Vi hade också kunnat skriva `from scipy.optimize import *` och då importerat alla funktioner som finns i `scipy.optimize`.

Sedan anropar vi funktionen och beräknar nollstället:

```
z=newton(min_fun,1)
z
```

Med det första argumentet `min_fun` talar vi om för `newton` vilken funktion som skall användas, dvs. vilken funktion det skall sökas nollställe till. Det andra argumentet är ett tal nära nollstället och anger var `newton` ska börja leta efter nollstället. När vi kör raderna får vi utskriften
`0.8654740331016144`



Uppgift 4. Hitta alla nollställen till funktionen $f(x) = x^2 - \cos(x)$. Gör först en funktion som beskriver vår funktion och rita en graf. Använd sedan `newton` för att beräkna varje nollställe, ett i taget.

7 Information om Python

I de här laborationerna hinner vi förstås inte gå igenom hela språket, utan de fungerar mest som en sorts vägledning till Python.

Programmet vi använde för att köra Python heter Spyder. Vi hämtade Spyder från Anaconda-navigator där man har samlat olika program som gör det möjligt att köra Python. Man kallar dessa program för miljöer. Fördelen med att använda miljöer från Anaconda-navigator är att de paket vi är intresserade av att använda (`NumPy` och `Matplotlib`) redan finns installerade så vi behöver inte installera dem själva.

Om vi vill veta mer om hur man använder Spyder väljer vi alternativet Help ovanför Console och klickar på tutorial i meningen Read our tutorial. Här finns information om hur man använder Python i Spyder

Paketen vi importerar med raden `from pylab import *` heter `NumPy` och `Matplotlib`. Dessa har egna tillverkare. Vill vi veta vad de innehåller kan vi gå till deras webbsidor

`http://www.numpy.org/`
`https://matplotlib.org/`

Funktionen, `newton`, vi använde för att beräkna nollstället med finns beskriven här

https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.newton.html

Det finns mycket skrivet om PYTHON. Den mest aktuella beskrivningen hittar man på PYTHONs egna hemsidor:

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

Där finns en genomgång av språket, hur det är uppbyggt och tänkt att fungera.

8 Hjälp i SPYDER

En beskrivning av alla funktioner som finns i PYTHON och de paket man laddat in hittar man i det inbyggda hjälptextsystemet som man når med `help`.

Skriv

```
help(newton)
```

i CONSOLE. Programmet SPYDER svarar med en hjälptext för kommandot. Nedan ser vi början av hjäptexten. (Vi måste importera funktionen innan vi kan titta på hjälptexten).

```
Help on function newton in module scipy.optimize.zeros:

newton(func, x0, fprime=None, args=(), tol=1.48e-08, maxiter=50, fprime2=None)
    Find a zero using the Newton-Raphson or secant method.

    Find a zero of the function ‘func‘ given a nearby starting point ‘x0‘.
    The Newton-Raphson method is used if the derivative ‘fprime‘ of ‘func‘
    is provided, otherwise the secant method is used.  If the second order
    derivative ‘fprime2‘ of ‘func‘ is provided, then Halley’s method is used.

    Parameters
    ----------
    func : function
        The function whose zero is wanted. It must be a function of a
        single variable of the form f(x,a,b,c...), where a,b,c... are extra
        arguments that can be passed in the ‘args‘ parameter.
    x0 : float
        An initial estimate of the zero that should be somewhere near the
        actual zero.
    fprime : function, optional
        The derivative of the function when available and convenient. If it
```

Det är viktigt att lära sig att läsa hjälptexterna. De är inte skrivna för att lära ut till nybörjare hur man löser ett problem med PYTHON, utan för att visa exakt hur en funktion eller ett kommando används. Hjälptexterna är inte speciellt lättlästa. Man måste lära sig att plocka fram den informationen som är av intresse för tillfället, dvs. man måste lära sig att ”skumma” texterna.

Uppgift 5(a). Läs hjälptexten för `linspace` som vi använde i samband med grafritning. Hur anger man antal punkter man vill ha? Hur många punkter får man som standard om man inte anger något antal? (`linspace` finns i paketet `NumPy`).

(b). Läs hjälptexten för `newton`. Vilken metod användes när vi bestämde nollstället i exempel 5? Hur kan vi få funktionen att istället använda Newtons metod?