Python-priming i Matematik, undervisning och bedömning 2020

Samuel Bengmark

Introduktion

Geogebra är ett underbart matematiskt verktyg som ni säkert kommer att använda en hel del i ert arbete som lärare. Den dynamiska aspekten av Geogebra är fantastisk. Geogebra är också lätt att komma in i eftersom man antingen klickar i menyer eller skriver enradiga kommandon. Det innebär dock också att Geogebra i vissa situation är ett olämpligt verktyg. Dessutom är Geogebra inte reguljärt programmeringsspråk. I allt fler länder lägger man till programmering bland de obligatoriska kurserna redan i grundskolan, så även i Sverige. I Sverige har man valt att lägga detta som del av matematikundervisningen i grundskolan och gymnasiet.

För att förbereda er för detta har vi på Lärande och ledarskap valt ni inkludera undervisning om detta i ett programmeringsspråk som heter Python. Detta finns inkluderat i tre kurser. Först möter du nu på det kort i denna kurs, Matematik, undervisning och bedömning. I kursen Problemlösning och lärande, sker mer regelrätt undervisning om detta. Slutligen får du chans att skaffa dig mer erfarenhet genom att du använder Python som verktyg för modellering i kursen Modeller för förståelse. Målet är att du under dina studier på Lärande och ledarskap skall få så pass mycket kunskap i detta så att du skall enkla program som använder selektion och repetition.

Vissa av er har redan goda kunskaper inom programmering och kommer uppfatta denna laboration som mycket trivial. Men för andra är det nytt. Kanske är det så för dig. Jag vill då förbereda dig på att det är troligt att du då upplever det som att du inte fattar nått den första tiden. Sen, plötsligt, släpper det och du kommer att fatta hur du skall tänka. Programmering är en tröskelkunskap. Så ge detta tid. Det är möjligt att du gör denna labb tillsammans med någon som redan kan, någon som redan klivit över tröskeln. Kanske sitter ni båda som frågetecken. Det kan upplevas svårt, men jag är mycket övertygad om att om du bara sitter lite själv också så kommer en ny värld öppna sig där du får möjlighet att göra Python till en assistent som kan hjälpa dig med repetitiva och arbetskrävande uppgifter.

En framgångsrik väg att lära sig programmera är att våga vrida och vända på redan befintliga program. Så efter att denna labb är genomförd kan det vara bra att återvända till programmen ni skrivit och fortsätta att ändra i dem och se vad

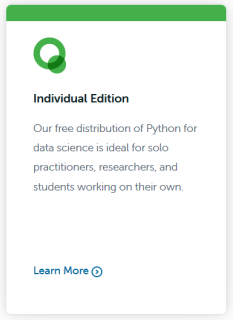
det ger för resultat. Du skall inte ha någon respekt för programmen. Inget kan gå sönder, men du kan lära dig mycket på att experimentera med kod och se vad det får för effekt.

Som redovisning på denna labb är det bara uppgift 8, 9 och 10 som skall lämnas in. Övriga uppgifter är övning på väg fram mot inlämningsuppgifterna.

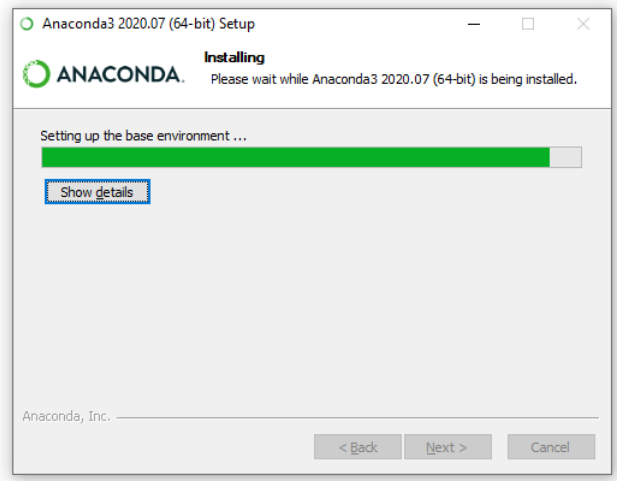
Starta Python i Jupyter

Installera på egen dator

Gå till anaconda.com. Finn och klicka på följande knapp.



Starta den nedladdade installationsfilen. Windows kommer att fråga om du verkligen vill köra denna fil. Det vill du, så klicka på Run för att bekräfta det. Du behöver klicka Next ett antal gånger till installationen sätter igång.



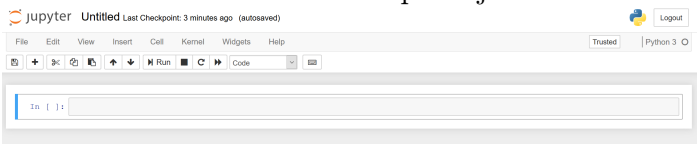
Efter en kort stund är installation av Anaconda klar.

Starta Jupyter Notebook

Starta Jupyter Notebook. I windows kan du tex söka efter Jupyter ner till vänster på skrivbordet och skriv Jupyter i rutan. Bland alternativen finner du Jupyter Notebook. Starta det!

Jupyter Notebook startas i den webbläsare som du satt som standard på din dator. I denna instruktion använder jag Firefox. Huvuddelen av din den öppnade vyn är en miljö där du kan skriva pythonkommanden, och som vi kommer att se, mycket mer. För att skriva Python kommandon behöver du starta en Notebook. Det gör du genom klicka på knappen New, uppe till höger, och välja Python 3.

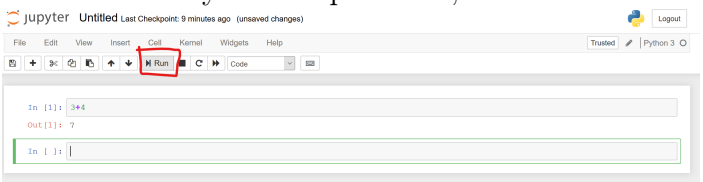
Ett ny flik öppnas nu i webbläsare där du får upp en notebook där du kan skriva Python kommandon. Det ser ut på följande sätt.



Ja, visst ser det lite fattigt ut. Inte lika attraktivt som i Geogebra, kanske, men ändå funktionellt skall du se.

Aritmetik i Python

Låt oss börja med enkel aritmetik. De fyra operationerna addition, subtraktion, multiplikation och division skrivs med hjälp av $+, -, *$ och $/$. Skriv in ett enkelt matematiskt uttryck i rutan, tex 3+4. För att få Pyton att räkna värdet trycker du på knappen Run. Alternativt kan du trycka CTRL-Enter. Skriver man bara Enter får man bara en ny rad i inputrutan, vilket vi kommer att ha nytta av inom kort.



Något mer oväntat är att potensfunktionen skrivs med hjälp av $**$, dvs 2^3 skrivs $2 * *3$. Man skriver $a\%m$ när man vill beräkna resten av a vid division med m.

Uppgift 1. Prova att undersöka behovet av parenteser i Python. Vad ger tex uttrycket $3 * *4 + 5$? Vad betyder $2 * *3 * *5$, $(2 * *3) * *5$ och $2 * *(3 * *5)$. Prova att

använda % för att beräkna 311 mod 4.

Andra elementära matematiska objekt, som kvadratrot, e och ln, eller trigonometriska funktioner finns inte direkt tillgängliga utan måste importeras. Kvadratroten heter sqrt(x) och sinus heter sin(x). Prova att skriva in följande tre steg. (Du skall bara skriva in det som står efter kolonet i följande rader.)

In [2]: from math import *
In [3]: sqrt(2)
In [4]: sin(2)

Uppgift 2. Undersök om sin() tar radianer eller grader som argument. Undersök vad log(x) har för bas. Undersök hur man skriver logaritmer med andra baser https://docs.python.org/3/library/math.html . Kolla också hur man skriver e och π.

Att skapa egna funktioner i Python

Det är väldigt användbart att skapa funktioner när man programmerar, funktioner som man kan återanvända och använda som delar i ett lösning av ett problem. Låt oss börja med ett exempel där du skriver in följande instruktioner.

```
def minFunktion(x):
    print(x+2)
```

Här trycker du retur för att skapa en ny rad att skriva på i samma input-ruta.

Funktionsdefinitionen i Python har ett antal delar som har sina särskilda platser. Funktionens namn ges av texten mellan def och parentesen. Namnet väljer du själv. Ovan har vi valt namnet minFunktion. Det som står i parentesen är indata/argument till funktionen. (En tom parentes betyder att funktionen inte har någon indata.) Kolon visar var funktionens innehåll börjar. Kommande rader, som skall ingå i beskrivningen av funktionen, måste alla vara indenterade. Rader som inte är indenterade ingår inte i funktionen.

När du är klar med funktionens alla rader trycker du shift-return så att Python läser in det du skrivit.

Prova nu att skriva följande i en annan inputruta.

In [10]: minFunktion(2)

Prova nu att skriva

In [10]: minFunktion(2)+ 3

Varför blir det fel? Jo, för att så som funktionen är definierad skriver den bara ut sitt värde. Den ger inte ett värde som kan behandlas vidare. Mer användbart är att funktion kan returnera ett värde som man kan fortsätta att använda. Det gör man med kommandot *return*. Fyll på din fil från ovan med följande innehåll.

```
def minFunktion(x):
    return x+2
```

Prova nu igen att skriva
In [11]: In [10]: minFunktion(2)+ 3

Uppgift 3. Vad ger minFunktion(minFunktion(5))? Förutsäg med papper och penna vad det borde ge och kontrollera sedan vad Python ger.

Uppgift 4. Skapa en funktion som tar värden på p och q och skriver ut de reella lösningarna till ekvationen $x^2 + px + q = 0$.

Som du märker skulle det vara bra att kunna hantera olika fall beroende på om det finns reella lösningar eller ej. För detta behöver du nästa vanliga progr

Selektion

Kommandot *if* kan användas för att göra val och för att delas upp i olika fall. Skriv in följande program i en tom fil.

```
x=6
if x>10:
    print(”Ditt tal är större än 10.”)
elif x==10:
    print(”Ditt tal är 10.”)
else:
    print(”Ditt tal är mindre än 10.”)
```

Observera att man använder indentering för att markera vilka satser som grupperas. Att vi efter raden "if x>10"har låtit kommandot *print* vara indraget innebär att detta utförs bara om det är sant att svar>10". Hade vi inte indenterat print-raden hade den inte påverkats av if-satsen utan alltid utförts.

Uppgift 5. Modifiera if-satsen ovan så att den delar in reella talaxeln i fyra delar: $(-\infty, 7], (7, 11), \{11\}$ och $(11, \infty)$ och anger om vilket av dessa intervall som det givna x värdet tillhör.

Uppgift 6. Modifiera if-satsen ovan så att den delar in reella talaxeln i fyra delar: $(-\infty, 7], (7, 11), \{11\}$ och $(11, \infty)$ och anger om vilket av dessa intervall som det givna x värdet tillhör,

Uppgift 7. Modifiera den funktion som du tog fram ovan för att ge de reella lösningarna till en andragradsekvation så att det skriver ut att lösning saknas när så är fallet, att det finne en dubbelrot och anger det när så är fallet och skriver ut de två lösingarna när så är fallet.

1 Input

Du kan få programmet att stanna upp och invänta input när programmet körs. Det kan göras med kommandot *input*. Prova tex följande rader

```
s=input(”Ange ett heltal:”)
print(s)
```

Uppgift 8. Skriv ett program som efterfrågar och läser in ett personnamn och sedan skriver ut en fördefinierad mening där det inlästa namnet nu ingår.

Python allt som läses in via *input* som en textsträng, dvs ser inte direkt om det skulle vara ett tal. Prova tex nu ändra i ovanstående kod så att det istället står:

```
s=input(”Ange ett heltal:”)
print(s+3)
```

Då får du ett felmeddelande. För att kunna använda det inlästa som ett tal behöver Python tolka det som ett tal. Det görs genom att man använder något av talkonverteringsinstruktionerna *int, long, float* eller *complex*. I ovanstående fall skulle vi kunna skriva

```
s=input(”Ange ett heltal:”)
x=int(s)
print(x+3)
```

Skulle användaren skriva in något som inte är ett heltal kommer programmet krascha. Prova gärna.

Uppgift 9. Modifiera nu din kod för lösningar till en andragradsekvation så att den efterfrågar och läser in koefficienterna p och q innan den skriver ut lösningarna till ekvationen som har dessa koefficienter.

Repetition

Datorer har oändligt tålamod. Dessutom är de snabba. Därför är det smart att använda dem när man skall göra något välbestämt många gånger. I Python finns ett par olika former av loopar, dvs repetitionssatser. Vi skall nu titta på den så kallade *for*-loopen.

Skapa ett program med följande innehåll.

```
for i in range(0,7):
    print(i)
```

Resultatet är att programmet upprepat skriver ut värdet i variabeln i när det vandrar genom värdena som finns i listan range(0,7). Det visar sig då att listan innehåller alla heltal från och med 0 tom heltalet innan 7, dvs 6. Detta är kanske något oväntat och något man bör lägga på minnet.

Uppgift 10. Skapa ett program med en *for*-loop som tar fram värdena i 3:ans multiplikationstabell, till och med $3 \cdot 10$.

Uppgift 11. Skapa ett program med en *for*-loop som tar fram värdena i 3:ans multiplikationstabell, till och med $3 \cdot 10$.

Uppgift 12. Funktionen $y = x^3 - 2x^2 + x + 5$ har ett nollställe mellan -1 och -2. Skriv ett program med en loop som halverar intervall och väljer den halva där nollstället finns. Ange intervallet du får efter 10 halveringar? Med hur många korrekta decimaler bör du då kunna ge ett närmevärde till nollstället?

Man kan lägga en loop i en annan, så kallade nästlade loopar, för att få två variabler som vandrar genom två mängder.

```
for i in range(0,7):
    for k in range(0,7):
        print([i,k])
```

Vi har då två loopar, en yttre (här med variabeln i, som för varje steg låter den inre loopen, här med variabeln k, genomföra alla sina 7 steg innan den yttre loopen övergår till sitt nästa steg.

Listor

Det är mycket användbart att kunna samla många värden i en lista. Man kan tex skapa en tom lista genom att skriva r=[ ]. Sedan kan man lägga till ett nytt p element i sist i listan genom att skriva r.append(p).

Uppgift 13. Skapa ett program med två for-loopar som ger multiplikationstabellen för alla tal 1 till och med 10. Börja med att bara få programmet att beräkna dessa värden och skriva ut dem ett och ett.

Övergå sedan till att lägga värdena för varje i en lista som du sedan skriver ut. Kan du lägga in detta i programmet så att listan skrivs ut och töms igen när du börjar räkna för ett nytt värde i den yttre loopen, dvs så att resultatet liknar en rektangulär tabell. (Var inte noga med estetiken, dvs bry dig inte om att försöka få räta kolumner i tabellen.)

Skapa uppgift

Uppgift 14. Skapa en uppgift som använder ovanstående Python-kommanden som du anser skulle vara lämplig för dina klasskompisar.

Slump

Paketet *math* som vi introducerade ovan innehåller många vanliga matematiska funktioner. Dock innehåller den ingen funktion för att ta fram slumptal. För detta kan man tex använda paketet *random*. Vill man tex med lika stor sannolikhet slumpa vilket rationellt tal som helst i ett intervall $[a, b]$ kan man använda funktionen *random.uniform(a,b)*. Här är ett exempel.

```
import random
def slumptal(a,b):
    for i in range(0,3):
        x=random.uniform(a,b)
        print(x)
```

Detta program kommer skriva ut tre x tal i intervallet $a \leq x \leq b$. Prova några gånger och se hur du får olika resultat varje gång. Att importera paket kan ske på några olika sätt. Använder man *import random* måste man anropa funktioner i paketet, tex uniform, genom att skriva paketets namn följt av punkt och funktionens namn. Importerar man istället med *from random import uniform*, eller *from random import ** slipper man (och kan inte) skriva paketets namn utan skall bara skriva random(2,3). Ovanstående funktion skulle då skrivas.

```
from random import *
def slumptal(a,b):
    for i in range(0,3):
        x=uniform(a,b)
        print(x)
```

Uppgift 15. Använd randint(1,6) för att slumpa heltal som simulerar tärningskast. Du slumpar två tärningskast och räknar ut deras summa. Hur stor andel av kasten får summan 6 om du gör 10 kast? Hur stor blir andelen om du gör 100 kast? Hur stor andel får summan 2? Vad är den matematiska sannolikheten att man får summan 6 respektive 2?

Elaborate-uppgifter

Dessa uppgifter får ni gärna göra tillsammans.

Uppgift 16. I denna uppgift skall du skapa två funktioner.

a) Tänk dig en enhetskvadrat med en inskriven cirkel. Dela nu in sidan och höjden i kvadraten i n lika stora delar. Då blir enhetskvadraten uppdelad i n^2 små rutor. Tänk dig att du sätter ut en punkt i varje sådan liten ruta. Beräkna nu ett approximativt värde på π genom att räkna hur många av dessa punkter som ligger inuti enhetscirkeln.

   Du skall samla denna kod i en funktion PiA(n) som returnerar den beräknade approximationen. Du kommer troligen att använda två nästlade loopar och en if-sats.

b) Gör nu om ovanstående uppgift men med skillnaden att du slumpar ut n^2 punkter i enhetskvadraten och räknar hur många av dem som ligger innanför cirkeln. Du skall samla denna kod i en funktion som du ger namnet PiMC(n). MC står här för Monte Carlo, som är en benämning på sådana här slumpmässiga simuleringar.

c) Bonusuppgift: Jämför PiA(n) och PiMC(n) för ökande värden på n. Är det någon av dem som verkar mer effektiv?

Uppgift 17. Även i denna uppgift skall du skapa två funktioner.

a) Tänk dig en enhetskvadrat i vilken man slumpar ut två punkter. Vad är sannolikheten att de två punkterna ligger på ett avstånd som är mer än 1 från varandra? Skriv en Python funktion BigDist(n) som gör en Monte Carlo simulering med n par av punkter och som returnerar en approximation av den efterfrågade sannolikheten.

b) Skapa en funktion FrekvensBigDist(n) som skapar en frekvenstabell tabell för avstånd mellan n slumpade par av punkter i enhetskvadraten. Frekvenserna skall ange antalet punkter som har avstånd mindre än 0.1, mellan 0.1 och 0.2, osv.

c) Bonusuppgift som inte måste lämnas in: Gör en hypotes om vilket medelvärdet, standardavvikelse och vilken typ av fördelning vi har. Kan du utifrån detta beräkna det exakta värdet i a)?

Uppgift 18. Slutligen skall du skapa följande två funktioner.

a) Skriv en funktion TriangelArea(a, b, c) som beräknar arean av en triangel med hörn i de tre punkterna a, b och c.

b) Tänk dig en enhetskvadrat i vilken man slumpar ut tre punkter. Skriv en funktion FrekvensTriangelArea(n) som ger en frekvenstabell av areastorlekar uppdelat i lämpliga intervall.

c) Bonusuppgift som inte måste lämnas in: Gör en hypotes om vilket medelvärdet, standardavvikelse och typ av fördelning vi har.

Uppgift 19. Detta är en frivillig uppgift tänkt för dig som redan är van att använda programmering i problemlösning.

a) Formulera en liknande fråga för någon annat geometriskt objekt än punkter, linjer och trianglar.

b) Skapa program med vilkas hjälp du kan undersöka din fråga samt finna approximativa svar och hypoteser.