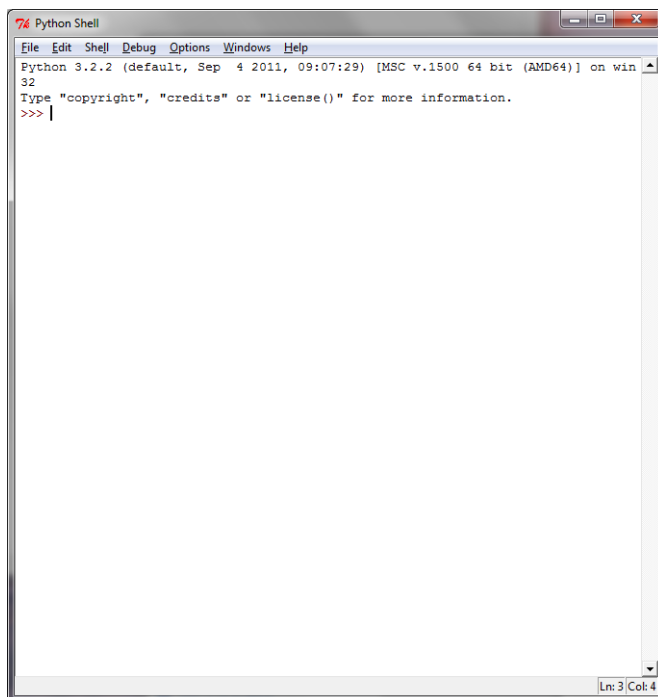


Laboration 1: Python-introduktion

För att bli bekant med Python, så bör du göra några av de föreslagna övningarna från "Läsanvisningar". Dessa uppgifter behöver ej redovisas, men är nyttiga för att klara de efterföljande laborationerna. Du väljer själv hur mycket du vill göra. Vill du börja direkt på laboration 2 så är det också OK.

Här är några små övningar som uppvärmning. Du bör i alla fall göra dessa.

1. **Starta Python.** Du får upp ett fönster som heter *Python Shell*.



2. **Använda moduler i Python:**

- a. Importera en modul med matematikfunktioner genom att skriva:
`>>> import math`
Det finns många olika moduler för Python, med praktiska funktioner i. Den stora fördelen med moduler är att man inte behöver skriva alla funktioner själv. Mycket finns redan färdigt.
- b. I matematik-modulen är vi särskilt intresserade av en konstant som innehåller värdet av pi. Skriv:

```
>>> math.pi
```

Se vad Python svara med.

3. **Matematiska uttryck i Python.**

- a. Beräkna ytan av en cirkel med radien 10. Det görs så här:

```
>>> math.pi * (10**2)
```

Operatörn `**` står för ”upphöjt till”.

- b. Skapa en variabel för radien:

```
>>> r = 10
```

- c. Beräkna ytan på nytt, men använd nu variabeln i stället:

```
>>> math.pi * (r**2)
```

- d. Beräkna den naturliga logaritmen av r .

```
>>> math.log(r)
```

- e. Beräkna längden på hypotenusan på en rätvinklig triangel med kateter med längderna 5 och 10. Använd upphöjt-till-operatörn `**` och funktionen `math.sqrt(x)` som ger kvadratroten av ett värde x .

4. **Skapa funktioner.** I stället för att behöva skriva om matematiska formler hela tiden så kan vi skriva dem en gång i en funktion och sedan använda funktionen.

- a. Börja med att öppna ett redigeringsfönster i Python (Menyn *File > New Window*).
b. Skriv

```
import math
```

på första raden.

- c. Spara fönstrets innehåll i en fil med namnet `mina_funktioner.py` (menyn *File > Save*).
d. Lägg till en tom rad, och sedan den här koden:

```
def circle_area(r):  
    return math.pi * (r**2)
```

Det här är en funktionsdefinition, vilket syns på nyckelordet `def` i början. Funktionens namn är `circle_area` och den har en parameter `r` inom parenteserna. Raden slutar med ett kolon (viktigt!). Den här raden kallas funktionens ”huvud”, och allt som kommer under kallas funktionens ”kropp”. Kroppen består av en enda rad, som genomför en beräkning och returnerar resultatet. Kroppen är indenterad (inskjuten) fyra kolumner.

- e. Spara filen igen (det går också att göra genom att trycka `Ctrl+S`). Sedan går du till menyn *Run* i redigeringsfönstret och väljer *Run Module* (eller trycker `F5`).

Nu kommer Python att köra koden i filen du sparade, och din funktion kommer att skapas.

- f. Hoppa tillbaka till fönstret som heter *Python Shell*.
- g. Skriv:

```
>>> circle_area(10)
```

Nu körs din funktion med **10** som inparameter.
Först skapas en variabel **r** som får värdet **10**.
Sedan körs funktionskroppen. Den beräknar arean och returnerar resultatet.
Det returnerade värdet skrivs ut nedanför anropet.

- h. Om du skriver:

```
>>> a = circle_area(10)
```

så skrivs inte resultatet ut, utan sparas i stället i variabeln **a**.

- i. Skriv en ny funktion under den förra (med en eller två tomma rader emellan). Den här funktionen ska beräkna längden på en hypotenus, och den ska ha längden på kateterna som parametrar. Du får själv bestämma vad funktionen och dess parametervariabler ska heta. Spara och kör (*Save, Run Module*) och testa att funktionen fungerar i *Python Shell*.

5. Strängar och utskrifter.

- a. Lägg till den här funktionen till din fil:

```
def print_circle_area(r):  
    a = circle_area(r)  
    print("Areal för en cirkel med radien", r, "är", a)
```

Spara och kör, och testa funktionen med olika inparametrar.

Lägg märket till att strängar alltid måste omges av citattecken (" eller ').
Däremot har man aldrig citattecken kring variabler, som **r** och **a**.

- b. Indenteringen är väldigt viktig här. Den talar om vilka rader som tillhör funktionens kropp. Prova vad som händer om du inte indenterar den sista raden. Varför då?
- c. Gör en liknande funktion som skriver ut längden på hypotenusan för en rätvinklig triangel.
- d. Två eller flera strängar kan slås ihop med operatoren **+** (konkatenering), t ex:

```
>>> "Hi"+" "+"there"
```

Skriv en funktion **hej1(namn1,namn2)** som tar två namn som inparametrar och returnerar en sträng enligt mönstret:

Hej *namn1* och *namn2*, trevligt att råkas!

Din funktion ska använda + för att slå ihop delsträngarna.
Tänk på sätta citattecken runt strängarna när du anropar funktionen.

- e. Gör en funktion `hej2(namn1, namn2)` som skapar samma sorts sträng, men skriver ut den i stället för att returnera den.
- f. Testa med:

```
>>> v1 = hej1("Bob", "Ann")
```

```
>>> v2 = hej2("Bob", "Ann")
```

Vad händer, och vilka värden får `v1` och `v2`?

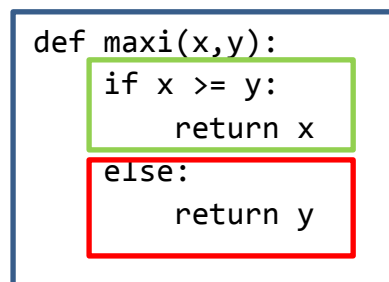
Ett `return`-värde skrivs inte ut, om det t ex används i en variabeltilldelning.
En utskrift, å andra sidan, returnerar ingenting. I Python används symbolen `None` för att representera inget värde.

6. Villkorssatser.

- a. Den här funktionen returnerar det största av två tal:

```
def maxi(x,y):  
    if x >= y:  
        return x  
    else:  
        return y
```

Lägg särskilt märke till indenteringen: raderna med `return` är indenterade längre in än de med `if` och `else`. Det är för att visa att de hör till `if`- och `else`-satserna (se bilden nedan). Glöm inte kolon (:) efter `if` och `else`!



```
def maxi(x,y):  
    if x >= y:  
        return x  
    else:  
        return y
```

- b. Skriv en funktion `abso(x)` som returnerar absolutvärdet av `x`. Om `x` är negativt så ska du alltså returnera `-x`. I annat fall returnerar du `x` rakt av.

7. Loopar.

- a. Den här funktionen tar en sträng och skriver ut tecknen var för sig:

```
def print_chars(str):  
    for ch in str:  
        print(ch)
```

Pröva funktionen med olika strängar som indata.

- b. Skriv en funktion som tar en sträng och för varje tecken *c* i strängen skriver ut:

Ge mig ett *c*!

Efter loopen är klar ska den skriva:

Vad blir det? *strängen*!

- c. Den här funktionen letar igenom en sträng efter ett visst tecken, och returnerar tecknets (första) position i strängen, eller -1 om det inte finns där.

```
def pos_in_string(ch,str):
    l = len(str)      # l är strängens längd
    for p in range(l): # loop med p från 0 till l-1.
        if str[p] == ch: # Finns tecknet ch på position p?
            return p     # Avbryt och returnera position p.
    return -1         # Vi hittade inte ch, så vi returnerar -1
```

Läs noga igenom funktionen, och prova den också med olika indata, t ex

```
>>> pos_in_string("a", "Hello pal! ")
```

Några viktiga saker för att förstå funktionen:

- `len(str)` ger längden på strängen
- `range(l)` ger alla tal från 0 till l-1.
- Variabeln *p* kommer alltså få värdet 0 första varvet i loopen, 1 i andra varvet, osv tills sista varvet där *p* blir l-1.
- `str[p]` ger tecknet på position *p* i strängen. Det jämförs med tecknet *ch*.
- Så fort *ch* hittas, så returneras positionen *p* och funktionen avbryts. Loopen avbryts naturligtvis också då.
- Om *ch* inte hittas och loopen tar slut, så returneras -1 (sista raden). Eftersom sista raden är indenterad lika mycket som raden med `for`, så tolkas det som att den ligger efter hela loopen.

- d. Skriv en funktion som räknar antalet gånger som ett visst tecken förekommer i en sträng. Ex:

```
>>> count_in_string("e", "Hello")
1
>>> count_in_string("a", "Hahaha")
3
```

Tips! Funktionen behöver en loop över tecknen i strängen, och en variabel

`count` som håller reda på antalet funna tecken. Sätt `count=0` innan loopen börjar. För att räkna upp `count` med 1 när du hittar tecknet skriver du:

```
count = count + 1
```

Returnera `count` när loopen är slut.

Du kan skriva funktionen antingen så att den loopar direkt över tecknen i strängen (som i 7a här ovan) eller så att den loopar över positioner i strängen (som i 7c här ovan).