Grunderna i Python programmering, del 1

Mar10

by nimmis on March 10, 2013 at 6:46 pm

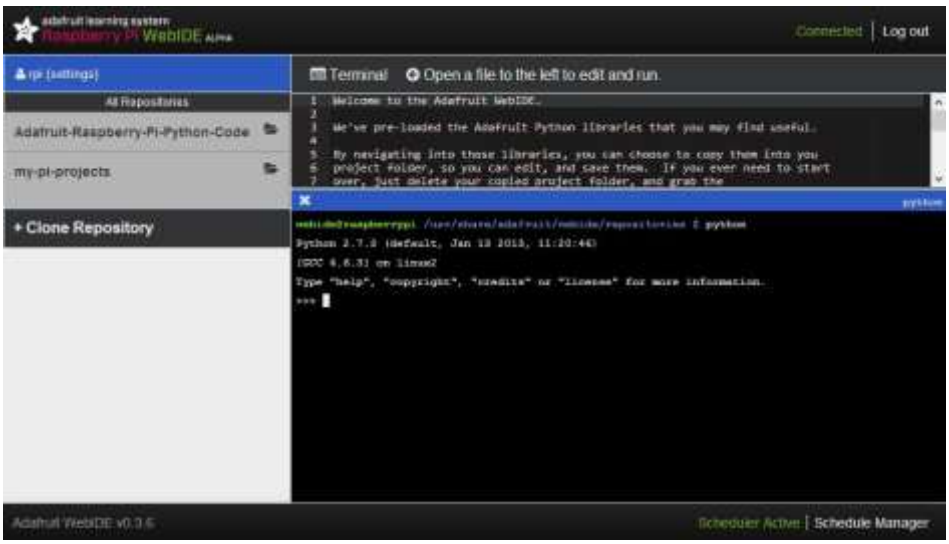
Posted In: Allmänt, komma igång, programmering, python

För att kunna utnyttja enkortsdatorn behöver man något typ av program som utför någon typ av uppgift. Som jag beskrev i tidigare artikel (se Nybörjarspråk, vad ska jag välja?) är python ett utmärkt språk att börja med. I denna artikel ska jag beskriva de grundläggande i pythonprogrammering. Eftersom python är ett interpreterade språk så kan man programmera interaktivt dvs man kan starta interpretatorn och skriva in programmet som utförs direkt utan att man först måste kompilera det.

För att visa grunderna är det enklast att köra den interaktivt för att visa hur det fungerar. För att göra exemplen i denna artikel måste du antingen logga in på Raspberry PI och starta kommandot python

```
pi@raspberrypi ~ $ python
Python 2.7.3 (default, Jan 13 2013, 11:20:46)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

eller klicka på **Terminal** i WebIDE (se tidigare artikel) så att du får upp ett terminalfönster och skriv in python



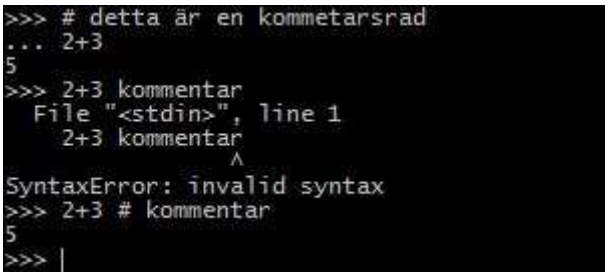
Raden med 3 större-än tecken >>> visar att python är klar och väntar på inmatning.

Kommentarer

Kommentarer är information som inte har någon funktion för programmeringsspråket utan är avsett för den som läser programkoden.

Många som börjar programmera tycker att man inte behöver skriva några kommentarer eftersom man vet vad programmet gör. Men om någon annan ska sätta sig in i koden eller man ska göra några ändringar i koden efter ett längre tag så uppskattar man kommentarerna då dessa fungerar som stöd för att förstå vad olika saker i programmet gör.

I python används brädgårdstecknet # för att indikera att allt efter detta tecken på raden ska tolkas som kommentarer och inte innehåller någon kod.



Som man kan se så fick vi en rad med 3 punkter …, detta betyder att python har inte fått nog med indata för att generera ett svar. I detta fall så skrev vi in en kommentar och detta är ingen kod som Python ska behandla så den frågar efter mer indata.

Alla tecken på en rad behandlas så länge som Python inte upptäcker ett #-tecken (om det inte finns i en sträng, förklaras längre fram) så på raden **2+3 kommentar** så får vi ett fel så Python inte känner igen **kommentar** och vet inte vad den ska tolka det som. Däremot på sista raden så avbryter Python att avkoda raden efter #-tecknet och utför bara det som stod före dvs **2+3**.

Numeriska värden

I datorsammanhang delar man upp tal i heltal och flyttal, skillnaden mellan dessa är att ett flyttal har decimaler.Viktigt att veta är att när man blandar heltal och flyttal i Python så blir resultatet alltid ett flyttal.

Man använder sig av +, -, * och / för de fyra räknesätten. Beräkningen sker med prioritering dvs / och * är högre prioriterade än + och – dvs 2+5*2 blir 12 och inte 14 pga att 5*2 beräknas innan man lägger till 2 till summan. För att förändra detta kan man använda parenteser så (2+5)*2 blir 14. Allt som man lär sig i matematiken.

För att utföra beräkningen skriver man helt enkelt den efter >>> och trycker **RETURN** så får man svaret direkt på nästa rad.

```
>>> 2+5
7
>>> 9-3
6
>>> 7*8
56
>>> 50/9
5
>>> 2+5*2
12
>>> (2+5)*2
14
>>>
```

Men 50/9 = 5 stämmer ju inte??? Detta beror på det jag skrev tidigare om talen inte har någon decimaldel så tolkas de som heltal och om man delar 2 heltal så får man ett heltal till svar. 50/9 =5.555555….. och heltals delen av detta tal är 5, här gäller alltid att man avrundar nedåt.

Hur får man rätt svar? Ja minst ett av talen måste var ett flyttal, Python har en funktion som konverterar heltal till flyttal som heter float()

```
>>> float(50/9)
5.0
>>> 50.0/9
5.555555555555555
>>> float(50)/9
5.555555555555555
>>>
```

Varför blev inte första talet rätt? Jo vad vi gjorde var att konvertera **svaret** till flyttal dvs uträkningen skedde fortfarande med heltal. Det finns även en specialfunktion för att få modulo dvs resten av en heltals division och då använder man ett procenttecken %

```
>>> float(50/9)
5.0
>>> 50.0/9
5.555555555555555
>>> float(50)/9
5.555555555555555
>>> 50%9
5
```

Används ofta för att kontrollera om ett tal är delbart med ett annat, om modulo = 0 så är det delbart. I Python finns massor med inbyggda matematiska funktion som t.ex sinus logaritmen m.m se Python dokumentation för en utförlig lista. Viktigt att notera är att man använder sig av punkt . och inte som definierat i Svenskan komma , som separator mellan heltal och decimaldel.

Variablar

Som i alla andra programmeringsspråk har man tillgång till variabler som kan tilldelas ett värde. I Python använder man inte sig av någon typ av suffix för att definiera vad en variabel innehåller utan värdet som tilldelas sätter vilken typ av innehåll som en variabel har.

Numeriska variabler

```
>>> a=2
>>> b=3
>>> a/b
0
>>> float(a)/b
0.6666666666666666
>>> a=2.0
>>> a/b
0.6666666666666666
```

Här ser man att allt fungera på samma sätt som i tidigare exempel, **a** och **b** tilldelades ett heltalsvärde så resultatet blir ett heltal. Notera att när man tilldelar **a** ett flyttalsvärde så ändras **a** till att vara en flyttalsvariabel. För att ta bort en definierad variabel använder man sig av kommandot **del variable**, så **del a** skulle ta bort variabeln **a**.

Strängar

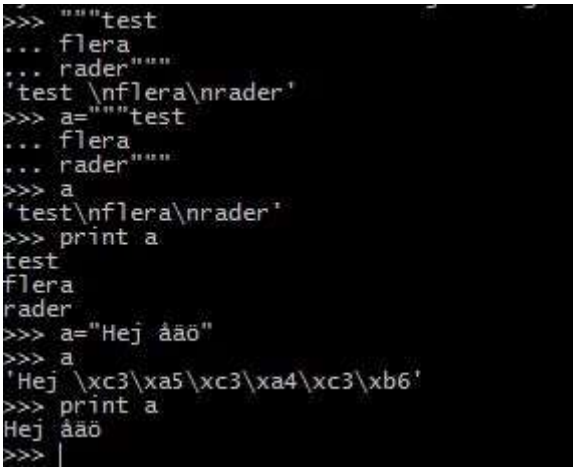
Strängar är en definition av en icke-numeriskt värde och behandlas som en sekvens av tecken, siffror eller bokstäver. En sträng börjar men antingen med ett enkel-citat tecken ‘ eller en dubbel-citat tecken ” och avslutas med samma tecken som man började med.

```
>>> "hej"
'hej'
>>> 'hej"
  File "<stdin>", line 1
    'hej"
        ^
SyntaxError: EOL while scanning string literal
>>> 'hej'
'hej'
```

Man kan slå ihop strängar genom att använda plustecknet + mellan strängarna

```
>>> a="hello"
>>> b=" world"
>>> a
'hello'
>>> b
' world'
>>> a+b
'hello world'
>>> c=a+b
>>> c
'hello world'
```

Det finns mycket mer special för att hantera strängar men det ingår inte i denna grundläggande beskrivning av Python. Det finns ett specialfall när man använder sig av 3 stycken ”’ eller “””" detta används när strängen ska sträcka sig över flera rader. Normalt får man ett felmeddelande om man trycker **RETURN** utan att avslutat en sträng.



I exemplet ovan se vi hur Python lagra strängar intern och hur den visar det som utdata. När man matade in den först så kunde man förväntat sig att se svaret som 3 rader, istället får man alla rader på samma rad med **\n** mellan varje rad. Om man tittar i Python dokumentationen för strängar så betyder **\n** samma sak som en ny rad men kodat. Endast när man använder funktionen **print** tolkar Python dessa special tecken

Samma kan man se med Svenska tecken som kodas men skrivs ut rätt när man använder **print** kommandot.

Subset av strängar

Ibland vill man hantera delar av strängar t.ex de 3 först tecknen i en sträng eller vilken är det första tecknet. I Python använder man sig av index i strängen med hjälp av hakparenterser [] efter variabelnamnet. Som i många andra språk är positionen för första tecknet i strängen 0.

Att definiera ett enstaka tecken i en sträng görs men **variable[position]**, så variable[2] betyder tecknet vid position 2 (3 tecknet).

Man kan även definiera en delmängd av strängen , syntaxen är **variabelnamn[startpostion:slutposition]** där **startposition** är index räknat från 0 och **slutposition** räknat från 1. Så variable[3:5] är 4:e och 5:e tecknet i strängen. Om man utelämnar **startpostion**, t.ex **variable[:4]** så betyder det de 4 första tecknen och om man utelämnar **slutpostion** t.ex **variable[3:]** så betyder det alla tecken från och med 4 tecknet. Värdet av ett subset kan inte ändras dvs variable[2]=”b” fungerar inte

Listor

En lista är ett antal element (av valfri typ t.ex strängar) separerade med kommatecken och omgivna av hakparenteser, t.ex **a=["adam","bertil","cecar"]** är en lista med 3 element.

Åtkomsten av innehållet i listan påminner om hanteringen av substrängar där varje element i listan är samma sak som ett tecken i strängen, så **a[0]** år lika med **“adam”**

```
>>> a=["adam","bertil","cecar"]
>>> a[0]
'adam'
>>> a[0:2]
['adam', 'bertil']
>>> |
```

När man sen har en lista finns ett antal funktioner som man kan använda på listan genom att ta **variable.funktion()**. En fullständig lista över funktioner för listor kan finns i Pythons Dokumentation om listor.

count() – räkna träffar i listan

count(element) räknar hur många gånger elementet finns i listan

```
>>> a=["adam","bertil","cecar","adam","nisse"]
>>> a.count("adam")
2
>>> a.count("nisse")
1
>>> a.count("eva")
0
>>> |
```

I exemplet så finns adam 2 ggr, nisse 1 ggr och eva finns inte i listan

extend() – utöka lista med lista

extend(lista) lägger till lista till listan.

```
>>> b=["sune","ivar","kalle"]
>>> a.extend(b)
>>> a
['adam', 'bertil', 'cecar', 'adam', 'nisse', 'sune', 'ivar', 'kalle']
>>> |
```

I exemple skapades en ny lista med sune, ivar och kalle genom att använda funktionen **extend** så lades allt i lista **b** till de som redan fanns i lista **a**. Ett annat sätt är att använda plustecknet + som innebär summering av listorna

```
>>> a+b
['adam', 'bertil', 'cecar', 'adam', 'nisse', 'sune', 'ivar', 'kalle']
>>> a=a+b
>>> a
['adam', 'bertil', 'cecar', 'adam', 'nisse', 'sune', 'ivar', 'kalle']
>>> |
```

insert() – lägga till element till lista

insert(postion, element) lägger till ett element efter en viss postion i listan.

```
>>> a.insert(2,"eva")
>>> a
['adam', 'bertil', 'eva', 'cecar', 'adam', 'nisse', 'sune', 'ivar', 'kalle']
>>> |
```

pop() – hämta element från lista

pop(position) returnerar värdet av elementet som har position **position** och samtidigt raderar det från listan.

```
>>> a
['adam', 'bertil', 'cecar', 'adam', 'nisse', 'sune', 'ivar', 'kalle']
>>> a.pop(1)
'bertil'
>>> a.pop() # sista postionen i listan
'kalle'
>>> a
['adam', 'cecar', 'adam', 'nisse', 'sune', 'ivar']
>>>
```

append() – lägg till element till lista

append(element) lägger till elementet sist i listan

```
>>> a.append(2)
>>> a
['adam', 'cecar', 'adam', 'nisse', 'sune', 'ivar', 2]
>>>
```

sort() – sortera elementen i stigande ordning i listan

sort() sorterar elementen i stigande ordning i listan

```
>>> a=['adam', 'cecar', 'adam', 'nisse', 'sune', 'ivar', 2]
>>> a.append('1')
>>> a.sort()
>>> a
[2, '1', 'adam', 'adam', 'cecar', 'ivar', 'nisse', 'sune']
>>>
```

Som ni ser så kommer 2 före ’1′ och detta beror på att numeriska värden räknas lägre än strängar så det är viktigt att hålla koll på vilka typer av värden som man har i listan.

reverse() – sortera elementen i sjunkande ordning i listan

reverse() sorterar elementen i sjunkande ordning i listan, sorteringen gäller som i sort att numeriska värdlen är lägre än sträng värden.

```
>>> a.reverse()
>>> a
['sune', 'nisse', 'ivar', 'cecar', 'adam', 'adam', '1', 2]
>>>
```

del – ta bort element från listan

del är ett kommando och inte en funktion eftersom den inte returnerar ett svar. **del a[2]** tar bort det 3:e elementet i listan och **del a** tar bort hela listan

```
>>> a
['sune', 'nisse', 'ivar', 'cecar', 'adam', 'adam', '1', 2]
>>> del a[2]
>>> a
['sune', 'nisse', 'cecar', 'adam', 'adam', '1', 2]
>>> del a
>>> a
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'a' is not defined
>>>
```

Tuples – enkla listor

tuples (närmaste på Svenska är mängd) är en enkel lista som inte kan manipuleras dvs endast läsbar.

Man kan använda plustecknet för att slå ihop 2 tuples men inget mer, används främst för statiska listor

```
>>> a="adam","bertil","cecar"
>>> a
('adam', 'bertil', 'cecar')
>>> b="sune","kalle"
>>> a=a+b
>>> a
('adam', 'bertil', 'cecar', 'sune', 'kalle')
```

Sets – Unika listor

Sets är tuples där alla element är unika, man kan skapa ett set från listor och tuples med funktionen set(), resultatet oavsett om det är en lista eller tuples är en tuples.
Alla dubbletter tar bort från resultatet.

```
>>> a=['adam', 'cecar', 'adam', 'nisse', 'sune', 'ivar']
>>> b=set(a)
>>> b
set(['sune', 'cecar', 'ivar', 'adam', 'nisse'])
>>> b.sort()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'set' object has no attribute 'sort'
```

Dictionary - Koppla nycklar mor värden

Dictionary (närmast på Svenska är lexikon) används för att koppla nyckelord mot ett värden, anges inom krullparanteser {} och anges som nyckelord:värde med kommatecken mellan värdena.

```
>>> a={ 'kalle':39, 'bertil':27, 'cecar':87 }
>>> a['kalle']
39
>>> a.keys()
['kalle', 'bertil', 'cecar']
>>>
```

keys() funktionen returnerar en lista av alla nyckelord i lexikonet.