Programmeringsteknik I

5 hp

Lärare: Johan Öfverstedt
E-post: johan.ofverstedt@it.uu.se
+ Ett stort antal labbassistenter

Föreläsning 1: Kursinformation och introduktion till programmering med Python

Kursen består av

- (Vanligtvis) 6 föreläsningar
- Ett stort antal laborationstillfällen (med 5 obligatoriska redovisningar) (3 hp)
- Hemtentamen (2 hp)

Laborationstillfällena med eget arbete vid dator är essentiellt för aktivt lärande och mängdträning.
Föreläsningarna ger en fördjupad bild (med teori) av de begrepp som ni stöter på under lösning av de praktiska uppgifterna.

Kurshemsidan

http://www.it.uu.se/edu/course/homepage/prog1/python/ht20/

Kurshemsidan innehåller 10 nätlektioner som är de praktiska momenten i kursen, och minilektioner som kan ses som en extra resurs.

Viktiga datum

OU1: Laborationstillfälle 8 - 10/9
OU2: Laborationstillfälle 12 - 17/9
OU3: Laborationstillfälle 15 - 22/9
OU4: Laborationstillfälle 20 - 30/9
OU5: Laborationstillfälle 28 - 16/10

Det är tillåtet att redovisa dessa före deadline.

Tentamen: 23/10 (Obligatoriskt att anmäla sig i förväg om man ska skriva tentan)

10 nätlektioner

Lektion 1: Grundläggande Python
Lektion 2: Programmeringsmiljön IDLE
Lektion 3: Sköldpaddsvärlden
Lektion 4: Att skriva funktioner
Lektion 5: Mer om funktioner

Lektion 6: Listor och tupler
Lektion 7: Arbeta med text
Lektion 8: Introduktion till klasser
Lektion 9: Datahantering
Lektion 10: Trafiksimulering

Obligatorisk redovisning för nätlektionerna i fetstil

Laborationer

Laborationerna (som kommer äga rum på distans via nätet) kommer gå till så att ni skriver upp ert namn i ett Google-dokument som vi kommer dela, med information om

Namn, Zoom-länk

Där ni själva skapar ett Zoom-möte och skriver länken sist i listan. När det är din tur ansluter en assistent till Zoom-samtalet och hjälper till eller tar emot redovisning av obligatorisk uppgift.

Programmering

Programmering är en process där en mängd instruktioner sätts samman för att formellt beskriva en uppgift/ett problem som skall lösas och/eller dess lösning.

Funktionell programmering: Beskriv problemet som en matematisk funktion och låt datorn evaluera denna.
Procedurell programmering: Beskriv en explicit lösning steg-för-steg som datorn kan utföras, med procedurer som beskriver hur en beräkning går till.

Programmering

Programmering är en process där en mängd instruktioner sätts samman för att formellt beskriva en uppgift/ett problem som skall lösas och/eller dess lösning.

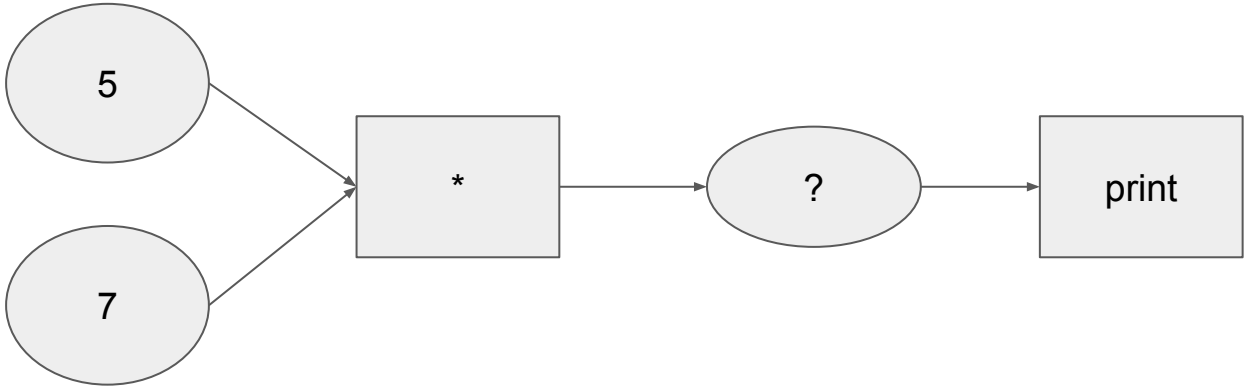
Deklarativ programmering: Beskriv problemet i form utav en modell, och låt datorn lösa problemet på egen hand.
Objekt-orienterad programmering: Kompatibelt med övriga programmeringsmodeller. Beskriv problemet med klasser som samlar data och logik / operationer i ett konceptuellt paket.

Programmeringspråk

Textbaserade programmeringsspråk:

```
x = 5 * 7
print(x)
y = x + 3
print(y)
```

Programmeringsspråk



Grafisk nodbaserad programmering

Python

Python är ett dynamiskt allmängiltigt programmeringsspråk (anses ofta som ett skriptspråk) som kan köra program utan att först kompilera dessa till maskinkod.

Terminologi
Maskinkod: En binärkodad datarepresentation som en processor kan köra.
Kompilering: En process som omvandlar kod med en hög abstraktionsnivå till t.ex. maskinkod.

Interaktiv Python i Terminal

Python kan programmeras interaktivt i terminalen, rad för rad, vilket kan vara behändigt antingen i början när man lär sig programmera, och även för att snabbt testa saker även när man är en van och erfaren programmerare.

```
C:\Users\johan>python
Python 3.7.4 (default, Aug  9 2019, 18:34:13) [MSC v.1915 64
bit (AMD64)] :: Anaconda, Inc. on win32

Type "help", "copyright", "credits" or "license" for more
information.
```

IDE (Integrated Development Environment)

Det finns ett stort antal IDE som kan underlätta skrivande, läsande och testande av programmeringskod.

- IDLE : Följer med Python
- Visual Studio Code : En gratis (open-source) IDE från Microsoft
- PyCharm : En kommersiell IDE som har många bra och kraftfulla hjälpmedel (man kan få en akademisk licens gratis)

Anaconda: En Python-distribution

Anaconda (https://www.anaconda.com/distribution/)
är en distribution av Python som innehåller en stor samling av populära paket.

Exempel

```
5*2**4
>> 80

5*(2**4)
>> 80


1+1*2+3
>> 6
```

Aritmetiken i Python följer vanliga prioriteringsregler, multiplikation före addition, etc.

Logik i Python

`a == b` : test för likhet (värde)

`not (a)` : negation av uttryck

`a is b` : test för likhet (identitet)

`a < b, a <= b, a > b, a >= b` : olikheter

Aritmetik i Python

`a + b` : addition
`a - b` : subtraktion
`a * b` : multiplikation
`a / b` : flyttalsdivision
`a // b` : heltalsdivision
`a ** b` : a upphöjt till b
`a % b` : modulo / rest vid division, `5 % 3 == 2`

Variabler: Associerar ett namn med ett värde

```
a = 2
b = 7
c = a**b
print(c)
>> 128

print(type(c))
>> <class 'int'>
```

c innehåller ett heltal (integer).

```
a = 2.0
b = 7
c = a**b
print(c)
>> 128.0

print(type(c))
>> <class 'float'>
```

c innehåller ett flyttal (float).

Kommentarer

```
# This is a comment. It will be ignored by Python.

'''
This is
a multiline
comment.
'''
```

Kommentarer används för att beskriva vad ett stycke kod gör, och varför, men ska inte användas för att beskriva självklarheter. Antag att läsaren av koden behärskar Python.

Strängar

```
name = 'Johan'
s = f'Hej jag heter {name}.'
print(s)
```

Utskrift:

```
>> Hej jag heter Johan.
```

Variablerna name och s är strängar. (Sekvenser av tecken som representerar text)

Utskrift på skärmen med “print”

Proceduren print används för att skriva ut på skärmen / till terminalen.

print('a', 'b', 'c') skriver ut
>> a b c

print('a', 'b', 'c', sep=', ') skriver ut
>> a, b, c

print('a', end=', '); print('b', end=', '); print('c')
>> a, b, c

Mer exempel med strängar

Man kan också skapa strängar med följande notation:
s = 'Hej jag heter %s' % 'Johan'

Problemet med den här notationen är att det är lättare att göra fel:
s = 'a = %d, b = %d, c = %d' % (a, b)

```
>>
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: not enough arguments for format string
```

Villkorssatser

```
a = 7
if a > 0:
  print('a is positive')
elif a == 0:
  print('a is zero')
else:
  print('a is negative')

>> a is positive
```

Inmatning

`input` - Läser en rad från terminalen och returnerar den:

```
name = input('Namn? ')
print(f'Hej {name}')
>> Namn? Johan
Hej Johan
```

Loopar/Slingor och förändring av variablers värden

```
i = 10
while i > 0:
  if i % 2 == 0:
    print(i)
  i = i - 1
```

Indentering (avstånd till vänstermarginalen)

I Python har kodens indentering betydelse

```
if a > 0:
  print('a', end='')
  print('b', end='')
```

Skriver ut 'ab', om a > 0, ingenting annars

```
If a > 0:
  print('a', end='')
print('b', end='')
```

Skriver ut 'ab', om a > 0, ‘b’ annars

Loopar inuti loopar

```
i = 0
while i < 3:
  j = 0
  while j < 4:
    print(f'({i}, {j})')
    j = j + 1
  i = i + 1
```

Loopar inuti loopar

```
i = 0
while i < 3:
  j = 0
  while j < 4:
    print(f'({i}, {j})')
    j = j + 1
  i = i + 1
```

Svar:
(0, 0)
(0, 1)
(0, 2)
(0, 3)
(1, 0)
(1, 1)
(1, 2)
(1, 3)
(2, 0)
(2, 1)
(2, 2)
(2, 3)

Exempel: Fakultet (flyttal)

```
fac = 1.0
n = 7
while n > 0:
  fac = fac * n
  n = n - 1

print(fac)

>>5040.0
```

Exempel: Fakultet (flyttal)

```
fac = 1.0
n = 1000
while n > 0:
  fac = fac * n
  n = n - 1

print(fac)

>>inf
```

Exempel: Fakultet (heltal)

```
fac = 1
n = 1000
while n > 0:
  fac = fac * n
  n = n - 1
print(fac)
```

```
4023872600770937735437024339230039857193748642107146325437999104299385123986290205920442084869694048004799886101971960586316668729948085589013238296699445909974245040870737599188236277271887325197795059509952761208749754624970436014182780946464962910563938874378864873371191810458257836478499770124766328898359557354325131853239584630755574091142624174743493475534286465766116677973966688202912073791438537195882498081268678383745597317461360853795345242215865932019280908782973084313928444032812315586110369768013573042161687476096758713483120254785893207671691324484262361314125087802080002616831510273418279777047846358681701643650241536913982812648102130927612448963599287051149649754199093422215668325720808213331861168115536158365469840467089756029009505376164758477284218896796462449451607653534081989013854424879849599533191017233555566021394503997362807501378376153071277619268490343526252000158885351473316117021039681759215109077880193931781141945452572238655414610628921879602238389714760885062768629671466746975629112340824392081601537808898939645182632436716167621791689097799119037540312746222899880051954444142820121873617459926429565817466283029555702990243241531816172104658320367869061172601587835207515162842255402651704833042261439742869330616908979684825901254583271682264580665267699586526822728070757813918581788896522081643483448259932660433676601769996128318607883861502794659551311565520360939881806121385586003014356945272242063446317974605946825731037900840244324384656572450144028218852524709351906209290231364932734975655139587205596542287497740114133469627154228458623773875382304838656889764619273838149001407673104466402598994902222217659043399018860185665264850617997023561938970178600408118897299183110211712298459016419210688843871218556461249607987229085192968193723886426148396573822911231250241866493531439701374285319266498753372189406942814341185201580141233448280150513996942901534830776445690990731524332782882698646027898643211390835062170950025973898635542771967428222487575867657523442202075736305694988250879689281627538488633969099598262809561214509948717012445164612603790293091208890869420285106401821543994571568059418727489980942547421735824010636774045957417851608292301353580818400969963725242305608559037006242712434169090041536901059339838357779394109700277534720000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

Listor

Ofta vill man inte bara använda sig av ett konstant antal variabler, alla med olika namn. Ofta har man en stor /varierande) mängd objekt. Då används en lista:

```
a = [1, 1, 2, 3, 5, 8, 13, 21]
print(a)
print(type(a))

>> [1, 1, 2, 3, 5, 8, 13, 21]
<class = 'list'>
```

Listor

```
a = [1, 1, 2, 3, 5, 8, 13, 21]
i = 0
while i < len(a):
  x = a[i]
  print(x**2)
  i = i + 1
```

Resultat

Fundera 1 min, vad skrivs ut?

Listor

```
a = [1, 1, 2, 3, 5, 8, 13, 21]
i = 0
while i < len(a):
  x = a[i]
  print(x**2)
  i = i + 1
```

Resultat

```
>> 1
1
4
9
25
64
169
441
```

Listor

```
a = [1, 1, 2, 3, 5, 8, 13, 21]
b = []
i = 0
while i < len(a):
  x = a[i]
  b.append(x**2)
  i = i + 1

print(b)
```

Resultat

```
>> [1, 1, 4, 9, 25, 64, 169, 441]
```

Iteration över listor, baklänges

```
a = [1, 1, 2, 3, 5, 8, 13, 21]
i = len(a)-1
while i >= 0:
    print(a[i])
    i = i - 1
```

Listor

len(lista) - ger listans längd, uttryckt i antalet objekt som lagras i listan
lista[0] - ger första elementet i listan
lista[len(lista)-1] - ger sista elementet i listan

Python använder 0-indexering, till skillnad från t.ex. Matlab som använder 1-indexering.

for-loop

Ett stort problem med while-satsen är att den lätt inbjuder till logiska fel.

Se följande kod:

```
i = 0
while i < 10:
  print(i)
```

Vad händer när detta kodstycke körs?

for-loop

Ett stort problem med while-satsen är att den lätt inbjuder till logiska fel, t.ex.:

```
i = 0
while i < 10:
  print(i, end='')
```

```
>> 0000000000000000000000000000000000000000000000000…
```

for-loop

Ett stort problem med while-satsen är att den lätt inbjuder till logiska fel, t.ex.:

```
i = 0
while i < 10:
  print(i, end='')
```

Problemet är att iteration med while-satsen ofta kräver att tre olika bitar kod är skrivna korrekt:
Initialiseringen av loop-variabeln (i)
Loop-villkoret
Uppdateringen av loop-variabeln (i)

for-loop (en lite bättre loop… för det mesta)

```
for i in range(10):
    print(i, end='')
```

Resulterar i:

```
>> 0123456789
```

for-loop

Man kan även iterera direkt över listor:

```
a = [1, 3, 7, 11, 13]

for i in a:
    print(i)
```

```
Resulterar i:
>>
1
3
7
11
13
```

Här behöver man inte ens komma ihåg att första elementet i listan har index `0` och det sista `len(a) - 1`.

for-loop med range (baklänges)

Man kan även iterera baklänges över en range:

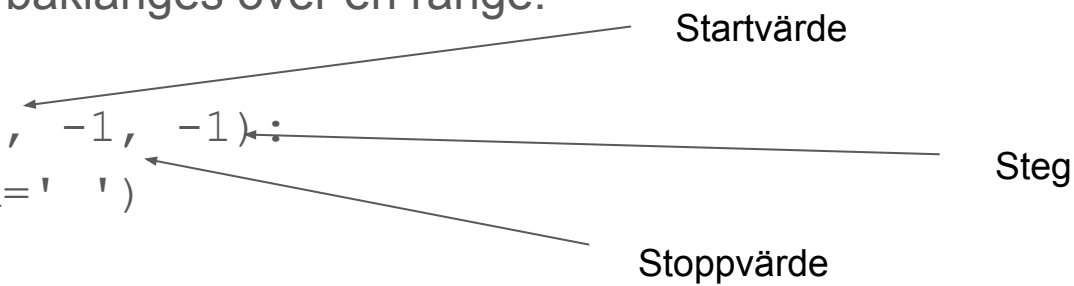
```
for i in range(9, -1, -1):
    print(i, end=' ')

>> 9 8 7 6 5 4 3 2 1 0
```

for-loop med range (baklänges)

Man kan även iterera baklänges över en range:

```
for i in range(9, -1, -1):
    print(i, end=' ')
```




```
>> 9 8 7 6 5 4 3 2 1 0
```

Terminologi

Tilldelning - Förändring av en variabels värde
Sekvens - Satser utförs, i den ordning som de står textuellt i koden
Iteration - Repetition av en process (ex. `while` / `for`)
Selektion - Val av satser som körs beroende av ett eller flera villkor (ex. `if`)

Datatyper

Idag har vi sett ett antal typer av data:

- Heltal (int)
- Flyttal/decimaltal (float)
- Textsträngar (string)
- Listor
- Logiska, boolska, sanningsvärden (bool) : False / True