# Python, del 2

Daniel Bosk[1]

Avdelningen för informations- och kommunikationssytem (IKS),
Mittuniversitetet, Sundsvall.

python2.tex 1340 2013-09-25 16:35:44Z danbos

# Översikt

# Översikt



Mittuniversitetet
MID SWEDEN UNIVERSITY

# Datatypen bool

- `bool` är en datatyp.
- Till skillnad från t.ex. `int`, kan ta enbart två olika värden.
  - `True` och `False` (skrivs också vanligtvis som 1 respektive 0)[2].

[2] Detta genom typkonvertering.

# Logiska operatorer

- På denna datatyp `bool` finns några operatorer definierade, likt `+ - * /` för flyttal.
- Dessa operatorer är `and or not`.
- `not` är en operator som skiljer sig något från de tidigare; den är unär, d.v.s. den opererar enbart på en operand.
- (Till skillnad från binära operatorer som opererar på två operander.)

# Logiska operatorer

| Operation | Resultat |
|---|---|
| not True | False |
| not False | True |

Tabell : Sanningtabell för Not

| and | True | False |
|---|---|---|
| True | True | False |
| False | False | False |

| or | True | False |
|---|---|---|
| True | True | True |
| False | True | False |

Tabell : Sanningstabeller för And och Or

# Logiska operatorer I

Några exempel från Python-tolken:

```
1  Python 2.2.3 (#1, Jan  5 2005, 16:36:30)
2  [GCC 3.4.2] on sunos5
3  Type "help", "copyright", "credits" or
       "license" for more information.
4  >>> True or False
5  True
6  >>> True and False
7  False
8  >>> not True
9  False
10 >>> not False
11 True
12 >>> not True and False
13 False
14 >>> not (True and False)
```

# Logiska operatorer II

```
15 True
16 >>> not False and True
17 True
18 >>> not (False and True)
19 True
20 >>>
```

# Jämförelseoperationer

- Precis som de aritmetiska och logiska operatorerna finns det operatorer för jämförelse definierade för de vanliga datatyperna i Python.
- Dessa är `==` `<` `>` `<=` `>=` `!=`.
- De används på samma sätt som de aritmetiska operatorerna, men skiljer sig genom att de ger upphov till ett värde som är av boolesk typ (`bool`) istället för ett värde av samma typ som operanderna.

# Jämförelseoperationer

Några exempel:

```
1  >>> 5+5
2  10
3  >>> 5 < 5
4  False
5  >>> 3 < 5
6  True
7  >>> 5 == 5
8  True
9  >>> 4 <= 5
10 True
11 >>> 5 <= 5
12 True
13 >>>
```

# Översikt

# Villkorssatser

- Villkorssatser används för att kontrollera programflödet, d.v.s. i vissa fall vill vi att programmet ska göra en sak och i andra fall något annat.
- För detta behöver vi olika villkor.
- Ett villkor är ett uttryck som kan evalueras till antingen `True` eller `False`.
- Exempelvis `x < 5` eller `x < 5 and x != 0`.

# Villkorssatser

- En typ av villkorssats är `if-elif-else`-satsen.
- Den fungerar på följande sätt:

```
1 if <villkor>:
2     # kod
3 # en eller flera elif-satser
4 elif <annat_villkor>:
5     # kod
6 else:
7     # kod
```

- Notera att man inte nödvändigtvis behöver ha med varken `elif` eller `else`, och man kan ta med hur många `elif` man önskar.

# Villkorssatser

- När något villkor är sant ignoreras alla efterföljande `elif` och `else` även om deras villkor skulle vara sanna.
- Ett exempel:

```
1  >>> x=5
2  >>> if x < 5:
3  ...     print( "less than 5" )
4  ... elif x < 10:
5  ...     print( "less than 10" )
6  ... else:
7  ...     print( "too large" )
8  ...
9  less than 10
10 >>>
```

# Slingor

- Slingor används för att få dynamiska upprepningar i programmet, t.ex. när vi vill läsa in en fil eller gå igenom en lista.
- Det finns två olika iterationskonstruktioner som tas upp här, de är `for`- och `while`-satserna.

# Slingor

- `for`-satsen fungerar på följande sätt:

```
1 for <variabel> in <lista>:
2     # kod
```

- Koden i `for`-satsen kommer att köras en gång för varje element i listan, varje gång kommer variabeln ha värdet av ett element i listan.

# Slingor

Ett exempel i python-tolken:

```
>>> for x in [1, 2, 3]:
...     print( "hej"*x )
...
hej
hejhej
hejhejhej
>>> for i in range(10):
...     print( i )
...
0 1 2 3 4 5 6 7 8 9
>>>
```

# Slingor

- Den andra iterationskonstruktionen är `while`-satsen.
- Den kör sin kod så länge ett givet villkor är sant.
- Den börjar med att kolla villkoret, om det är sant körs koden, annars fortsätter exekveringen direkt efter while-loopen.

```
1 while <villkor>:
2     # kod
```

- Notera att den enbart kollar villkoret en gång per varv, så hela koden körs trots att villkorets värde ändras under körningens gång.

# Slingor

Exempel:

```
1 >>> i=0
2 >>> while i<10:
3 ...     print( i )
4 ...     i += 1
5 ...
6 0 1 2 3 4 5 6 7 8 9
7 >>>
```

# Programexempel I

## Beräkna porto

```
1  #encoding: utf8
2
3  # weight är vikten, i gram, hos ett brev.
4  # rätt porto returneras.
5  def check_postage( weight ):
6    ## POSTENS 1:a KLASS INRIKES BREV
7    if ( weight <= 20 ):
8      return 6
9    elif ( weight <= 100 ):
10     return 12
11   elif ( weight <= 250 ):
12     return 24
13   elif ( weight <= 500 ):
14     return 36
15   elif ( weight <= 1000 ):
16     return 48
```

# Programexempel II

## Beräkna porto

```
17    elif ( weight <= 2000 ):
18      return 72
19    ## POSTENS POSTPAKET
20    elif ( weight <= 3000 ):
21      return 150
22    elif ( weight <= 5000 ):
23      return 175
24    elif ( weight <= 10000 ):
25      return 225
26    elif ( weight <= 15000 ):
27      return 275
28    elif ( weight <= 20000 ):
29      return 320
30    return -1
31
32
33  ########################
```

# Programexempel III

## Beräkna porto

```
34 # Huvudprogrammet
35 ######################
36
37 # Loopa så länge som möjligt
38 while ( True ):
39   print( "Välkommen␣till␣Brevvågen" )
40   nletters = input( "Hur␣många␣brev␣vill␣du␣
         beräkna␣porto␣för␣(0␣" \
41     "för␣att␣avsluta):␣" )
42   nletters = int( nletters )
43   # 0 för att avsluta (negativa också)
44   if ( nletters <= 0 ):
45     break
46   sum = 0
47   for i in range( nletters ):
48     w = input( "Hur␣mycket␣väger␣brev␣" + str(
           i+1 ) + ":␣" )
```

# Programexempel IV

## Beräkna porto

```
49      postage = check_postage( float( w ) )
50      if ( postage < 0 ):
51        print( "Den vikten finns inte med i
              portotabellen." )
52      else:
53        print( "Du måste betala", postage, "SEK i
              porto." )
54        sum += postage
55    if ( nletters > 1 ):
56      print( "Det blir", sum, "SEK för alla brev,
            tack!" )
57    elif ( sum > 100 and nletters == 1 ):
58      print( "Mycket pengar för ett postpaket!" )
```

# Programexempel

```
(0):danbos@ID20809793:python\$ python porto.py
Välkommen till Brevvågen
Hur många brev vill du beräkna porto för (0 för
    att avsluta): 4
Hur mycket väger brev 1: 123
Du måste betala 24 SEK i porto.
Hur mycket väger brev 2: 23
Du måste betala 12 SEK i porto.
Hur mycket väger brev 3: 1
Du måste betala 6 SEK i porto.
Hur mycket väger brev 4: 10345
Du måste betala 275 SEK i porto.
Det blir 317 SEK för alla brev, tack!
Välkommen till Brevvågen
Hur många brev vill du beräkna porto för (0 för
    att avsluta): 0
(0):danbos@ID20809793:python\$
```

# Översikt

# Datatypen list

- List är ytterligare en datatyp i Python, den kan användas för att lagra ett godtyckligt antal värden eller objekt i.
- T.ex. när man samlar data och har en datamängd som är större än ett (1) element.
- En lista har ett antal metoder, en slags funktion definierad på objektet självt.

# Datatypen list

Om vi skapar en tom lista `l`, genom att skriva `l = list()` alternativt `l = []`, kan vi använda följande metoder:

| | |
|---|---|
| l.append(e) | Lägg till `e` som ett element sist i listan. |
| l.extend(l2) | Lägg till listan `l2` sist i listan. |
| l.insert(p, e) | Sätt in `e` som ett element framför elementet på positionen `p`. |
| l.index(e) | Returnera det lägsta möjliga index för elementet som matchar `e`. |
| l.remove(e) | Ta bort första förekomsten av ett element som matchar `e`. |

Man kan också skapa en lista som redan innehåller några element genom koden `l = [1, 2, 3, "hej"]`.

# Datatypen list I

```
1  Python 2.2.3 (#1, Jan  5 2005, 16:36:30)
2  [GCC 3.4.2] on sunos5
3  Type "help", "copyright", "credits" or
       "license" for more information.
4  >>> l = [1, 2, 3]
5  >>> print( l )
6  [1, 2, 3]
7  >>> print( l[0] )
8  1
9  >>> print( l[2] )
10 3
11 >>> l.append(5)
12 >>> print( l )
13 [1, 2, 3, 5]
14 >>> l.extend([2,3])
15 >>> print( l )
16 [1, 2, 3, 5, 2, 3]
```

# Datatypen list II

```
17 >>> l.insert(0, 10)
18 >>> print( l )
19 [10, 1, 2, 3, 5, 2, 3]
20 >>> l.index(3)
21 3
22 >>> l.remove(3)
23 >>> l.index(3)
24 5
25 >>>
```

# Datatypen dictionary

- Ännu en datatyp hos Python, denna kan användas till att para ihop saker.
- T.ex. namn och telefonnummer, man kan söka på ett namn och få ut ett telefonnummer.

# Datatypen dictionary

Om vi skapar ett dictionary `d`, genom koden `d = {}`, kan vi sedan använda följande metoder:

| | |
|---|---|
| d.keys() | Returnerar en lista med alla lagrade nycklar. |
| d.values() | Returnerar en lista med alla lagrade värden. |
| k in d | Evalueras till sant (`True`) om nyckeln `k` finns i `d`. |

För att skapa en dictionary innehållandes några par skriver man följande: `d = {"nyckel": "värde", "programmering": "python", 1 : -1}`.

# Datatypen dictionary

```
1  >>> d = {"hej" : "svejs", "kth" : "kungl tekn
       högskol"}
2  >>> d.keys()
3  ['kth', 'hej']
4  >>> d.values()
5  ['kungl tekn högskol', 'svejs']
6  >>> "kth" in d
7  1
8  >>> "su" in d
9  0
10 >>>
```

# Lite mer om strängar

Strängar har, likt listorna och dictionaries, också metoder. Några av dessa metoder beskrivs här:

| | |
|---|---|
| s.split(c) | Delar upp strängen vid varje c, returnerar en lista med alla delarna. |
| s.strip(c) | Returnerar en sträng där alla c tagits bort från början och slut av strängen. |
| s.rstrip(c) | Samma som strip(), men behandlar enbart slutet av strängen – början lämnas orörd. |

# Lite mer om strängar I

```
>>> "hej svejs i lingonskogen".split()
['hej', 'svejs', 'i', 'lingonskogen']
>>> s = "hej svejs i lingonskogen"
>>> print( s )
hej svejs i lingonskogen
>>> s.split()
['hej', 'svejs', 'i', 'lingonskogen']
>>> s.split('e')
['h', 'j sv', 'js i lingonskog', 'n']
>>> s = "   hej svejs i lingonskogen    "
>>> print( s.strip() )
hej svejs i lingonskogen
>>> print( s.rstrip() )
   hej svejs i lingonskogen
>>> s = "...hej svejs i lingonskogen...."
>>> print( s.strip('.') )
hej svejs i lingonskogen
```

# Lite mer om strängar II

```
18 >>> print( s.rstrip('.') )
19 ...hej svejs i lingonskogen
20 >>>
```

# Översikt



Mittuniversitetet
MID SWEDEN UNIVERSITY

# Översikt

- Förutom den grundläggande felhantering man kan skriva själv i ett program finns det vissa fel som man inte lika enkelt kan skydda sig emot, där erbjuder Python något som kallas för särfall (eng. exceptions).
- Ett *särfall* är en slags signal som ”sänds ut” när ett fel uppstår, detta särfall kan man sedan fånga upp på ett speciellt ställe i koden och där behandla.

# Översikt

Man hanterar särfall enligt följande:

```
1 try:
2     # kod som kan ge upphov till särfall
3 except <typ0>:
4     # kod som ska köras när ett exception av typ
          <typ0> fångas
5 except <typ1>:
6     # körs när man fångar ett exception av typ
          <typ1>
7 except:
8     # kod som körs för alla andra särfall
```

Det är helt frivilligt hur många `except` man använder sig av, men man måste ha minst ett (1).

# Exempelkod I

```
1  Python 2.6.2 (r262:71600, Aug 12 2009, 11:11:06)
2  [GCC 3.3.5 (propolice)] on openbsd4
3  Type "help", "copyright", "credits" or
       "license" for more information.
4  >>> try:
5  ...     int("hej")
6  ... except:
7  ...     print( "exception" )
8  ...
9  exception
10 >>> int("hej")
11 Traceback (most recent call last):
12   File "<stdin>", line 1, in <module>
13 ValueError: invalid literal for int() with base
       10: 'hej'
14 >>> try:
15 ...     int("hej")
```

# Exempelkod II

```
16 ... except ValueError:
17 ...     print( "valueerror" )
18 ... except:
19 ...     print( "exception" )
20 ...
21 valueerror
22 >>> def f(a,b):
23 ...     return float(a)/float(b)
24 ...
25 >>> f("5","3")
26 1.6666666666666667
27 >>> f("hej", "du")
28 Traceback (most recent call last):
29   File "<stdin>", line 1, in <module>
30   File "<stdin>", line 2, in f
31 ValueError: invalid literal for float(): hej
32 >>> f("5","0")
```

# Exempelkod III

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in f
ZeroDivisionError: float division
>>> def f(a,b):
...     try:
...             return float(a)/float(b)
...     except ValueError:
...             print( "valueerror" )
...     except ZeroDivisionError:
...             print( "zerodivisionerror" )
...
>>> f("5","4")
1.25
>>> f("a","b")
valueerror
>>> f("5","0")
```

# Exempelkod IV

```
50 zerodivisionerror
51 >>>
```

# Översikt



Mittuniversitetet
MID SWEDEN UNIVERSITY

# Översikt

- Ibland kan man vilja ”komma ihåg” saker mellan körningarna av programmet, när alla variabler försvinner när programmet avslutas, eller läsa in stora mängder data som är otympligt att mata in från tangentbordet.
- Där kommer filhantering lämpligt in i bilden.
- Vi kan skriva det vi vill komma ihåg till en fil, och sedan läsa in den nästa gång vi startar programmet.

# Översikt

- För att öppna en fil i Python använder man sig av funktionen `open()`, denna funktion tar som första argument en sträng, som är sökvägen till filen som ska öppnas.
- Sökvägen kan vara absolut eller relativ. Om den är relativ utgår man från den mapp som programfilen finns i.
- Det andra argumentet är sättet vi vill öppna filen på, för läsning eller för skrivning.

# Översikt

- Open returnerar ett objekt som representerar filen. Objektet har några metoder för att kunna läsa och skriva data från respektive till filen.
- Vi öppnar filen `testfil.txt` för läsning genom

```
1 fil = open("testfil.txt", "r")
```

- Vi öppnar filen för skrivning genom

```
1 fil = open("testfil.txt", "w")
```

# Översikt
## Läsning

Följande metoder finns för att utföra olika handlingar på filen:

text = fil.read() Läs in hela filens innehåll till variabeln `text`.

rader = fil.readlines() Läs in hela filen, spara alla rader som separata element i listan `rader`.

rad = fil.readline() Läs in nästa rad till variabeln `rad`.

När vi inte längre behöver använda filen stänger vi den med `fil.close()`.

# Översikt

Följande metoder finns för att utföra olika handlingar på filen:

fil.write("hello file\n") Skriv *hello file* till filen, följt av en radbrytning.

fil.writelines(rader) Skriv alla strängar i listan rader till filen, notera att de inte skrivs på varsin rad om de inte själva innehåller radbrytningstecken.

När vi inte längre behöver använda filen stänger vi den med `fil.close()`.

# Översikt I

```
1  dbosk@my:/tmp\$ ls testfil.txt
2  testfil.txt: No such file or directory
3  dbosk@my:/tmp\$ python
4  Python 2.2.3 (#1, Jan  5 2005, 16:36:30)
5  [GCC 3.4.2] on sunos5
6  Type "help", "copyright", "credits" or
       "license" for more information.
7  >>> fil = open("testfil.txt", "w")
8  >>> fil.write("hello file\nen rad till")
9  >>> fil.close()
10 >>> \^D
11 dbosk@my:/tmp\$ ls testfil.txt
12 testfil.txt
13 dbosk@my:/tmp\$ python
14 Python 2.2.3 (#1, Jan  5 2005, 16:36:30)
15 [GCC 3.4.2] on sunos5
```

# Översikt II

```
16 Type "help", "copyright", "credits" or
       "license" for more information.
17 >>> fil = open("testfil.txt", "r")
18 >>> print( fil.read() )
19 hello file
20 en rad till
21 >>> fil.close()
22 >>> fil = open("testfil.txt", "r")
23 >>> print( fil.readline() )
24 hello file
25
26 >>> print( fil.readline() )
27 en rad till
28 >>> print( fil.readline() )
29
30 >>> \^D
31 dbosk@my:/tmp\$
```

# Översikt III

# Programexempel I

```
1  #encoding: utf8
2
3  # en lista att spara kommandohistoriken i.
4  history = []
5
6  # vår dictionary som håller koll på var
       programmet
7  # finns för att köra ett visst kommando.
8  cmds = {
9    "ls" : "/usr/bin/ls",
10   "cp" : "/usr/bin/cp",
11   "mv" : "/usr/bin/mv",
12   "res" : "/usr/local/bin/res",
13   "course" : "/usr/local/bin/course",
14   "python" : "/opt/sfw/bin/python",
15   "emacs" : "/usr/local/bin/emacs",
16   "vim" : "/opt/sfw/bin/vim",
```

# Programexempel II

```
17    "g++" : "/usr/sfw/bin/g++",
18    "cc" : "/usr/local/bin/cc",
19    "gdb" : "/opt/sfw/bin/gdb"
20    }
21
22  cmd = input( "Enter␣command:␣" )
23  while cmd != "exit":
24    try:
25      print( cmds[cmd] )
26      # koden nedan kommer aldrig inträffa om det
            blir
27      # en exception i cmds[cmd], då hoppar den
            direkt
28      # till except nedan.
29      history.append( cmd )
30
31    except KeyError:
```

# Programexempel III

```
32      print( "Sorry, that command is not in the
            dictionary." )
33    except:
34      print( "There is something seriously wrong
            here." )
35
36    cmd = input("Enter command: ")
37
38  for cmd in history:
39    print( cmd, end = "\n" )
40
41  try:
42    histfile = open( "history", "w" )
43  except:
44    print( "Couldn't open file for writing." )
45  else:
46    for cmd in history:
```

# Programexempel IV

```
47      histfile.write( cmd + "\n" )
```

# Referenser