

0.1 Introduction to Python

Python is a programming language for **text-based coding**. This means that the actions we want to be executed must be coded as text. The file containing all the code is referred to as a **script**. The visible result of running the script is termed **output**¹. There are various ways to run one's script; for example, one can use an online compiler like programmiz.com.

0.1.1 Object, Type, Function, and Expression

Our first script consists of just one line of code:

```
1 print("Hello world!")
```

```
Utdata  
Hello world!
```

In the upcoming sections, the terms **object**, **type**, **function**, and **expression** will frequently be discussed.

- Almost everything in Python is objects. In the above script, both `print()` and `"Hello world"` are objects.
- Objects come in different types. `print()` is of the **function** type, while `"Hello world"` is of the **str** type². The operations that can be executed with various objects depend on their types.
- Functions can accept **arguments** and then perform operations. In the script above, the `print()` function takes the argument `"Hello world"` and displays the text as output.
- Expressions have strong similarities with functions, but they don't accept arguments.

¹**Output** in English.

²'str' is an abbreviation for the English word 'string'.

Assignment and Calculation

Text and numbers can be seen as some of the smallest building blocks (objects). Python has one type for text and two types for real numbers:

<code>str</code>	text
<code>int</code>	integer
<code>float</code>	decimal

It is usually useful to give our objects names. We do this by writing the name followed by = and the object. **Comments** are text that is not treated as code. We can write comments by starting the sentence with #.

```
1 hei = "hei" # hei er av typen str. Legg merke til "  
                ved start og slutt  
2 a = 3 # a er av typen int  
3 b = 2.8 # b er av typen float  
4 c = 2. # c=2.0, og er av typen float  
5 d = .7 # d=0.7, og er av typen float  
6 e = -5 # e er av typen int  
7 f = -0.01 # f er av typen float
```

With Python, we can of course perform classic arithmetic operations:

```

1 a = 5
2 b = 2
3
4 print("a+b = ", a+b);
5 print("a-b = ", a-b);
6 print ("a*b = " ,a*b);
7 print ("a/b = " ,a/b);
8 print("a**b = " ,a**b); # potens med grunntall a og
                           eksponent b
9 print ("a//b = ", a//b); # a/b rundet ned til nærmeste
                           heltall
10 print ("a%b = ", a%b); # resten til a//b

```

Utdata

```

a+b = 7
a-b = 3
a*b = 10
a/b = 2.5
a**b = 25
a//b = 2
a%b = 1

```

The functions `str()`, `int()` and `float()` can be used to convert objects to types `int` or `float`:

```
1 s = "2"
2 b = 3
3 c = 2.0
4
5 b_s = str(b) # b omgjort til str
6 c_s = str(c) # c omgjort til str
7 print(b_s+c_s)
8
9 s_i = int(s)
10 print(s_i*b)
11
12 s_f = float(s)
13 print(s_f*b)
```

Utdata

32.0

6

6.0

One important thing to be aware of is that `=` in Python *does not* mean the same as `=` in mathematics. While `=` can be translated to "equals", we can say that `=` can be translated to 'is assigned to'.

```
1 a = 5 # a ER nå 5
2 print(a)
3 a = a+1 # a ER nå det a VAR, + 1
4 print(a)
```

Utdata

5

6

For an object to add itself and another value is so common in programming that Python has its own operator for it:

```
1 a = 5 # a ER nå 5
2 a+= 1 # Samme som å skrive a = a+1
3 print(a)
```

Utdata

5
6

Although computers are extremely fast at performing calculations, they have a limitation that is important to be aware of: rounding errors. One reason for this is that computers can only use a certain number of decimals to represent numbers. Another reason is that computers use the [binary system](#). There are many values that we can write exactly in the decimal system that cannot be written exactly in the binary system. To address this, we can use the `round()` function:

```
1 a = 8.3/10
2 print(a) # avrundingsfeil, da vi skulle hatt a = 0.83
3
4 a = round(a, 2) # runder av a til tall med to
   desimaler
5 print(a)
```

Utdata

0.8300000000000001
0.83

0.1.2 Custom Functions

Using the method **def**, you can create your own functions. A function can perform actions, and it can **return** one or more objects. It can also accept arguments. The code we write inside a function is only executed if we **call** the function.

```
1 # a er en funksjon som ikke tar noen argumenter.
2 # Legg merke til 'def' først og ':' til slutt.
3 # Kodelinjene som hører til funksjonen må stå med
  innrykk
4 def a():
5     print("Hei, noen kalte visst på funksjon a?")
6
7
8 # b er en funksjon som tar argumentet 'test'
9 def b(tekst):
10     print("Hei. Noen kalte på funksjon b. Argumentet som
      ble gitt var: ", tekst)
11
12 # c er en funksjon som tar argumentene a og b
13 # c returnerer et objekt
14 def c(a, b):
15     return a+b
16
17 b("Hello!") # vi kaller på b med argumentet "hello"
18
19 d = c(2,3) # Vi kaller på c med argumentene 2 og 3
20
21 print(d)
22
23 # merk at teksten gitt i a ikke blir printet, fordi vi
      ikke har kalt på a.
24
25
26
27
28
29
```

Utdata

Hi. Someone called function b. The argument given was:
Hello!
5

0.1.3 Boolean Values and Conditions

The values **True** and **False** are called **boolean values**. These will be the result when we check if objects are equal or different. To check this, we have the **comparative operators**:

operator	meaning
<code>==</code>	is equal to
<code>!=</code>	is <i>not</i> equal to
<code>></code>	is greater than
<code>>=</code>	is greater than, or equal to
<code><</code>	is less than
<code><=</code>	is less than, or equal to

```
1 a = 5
2 b = 4
3
4 print(a == b)
5 print(a != b)
6 print(a > b)
7 print(a < b)
```

Utdata
False
True
True
False

In addition to the comparative operators, we can use the **logical operators** **and**, **or**, and **not**.

```
1 a = 5
2 b = 4
3 c = 9
4
5 print(a == b and c > a)
6 print(a == b or c > a)
7 print(not a == b)
```

Utdata
False
True
True

Språkboksen

Checks that use both comparative and logical operators will henceforth be called **conditions**.

0.1.4 Expressions **if**, **else**, and **elif**

When we want to perform actions only *if* a condition is true (**True**), we use the expression **if** in front of the condition. The code we write indented under the **if** line will only be executed if the condition evaluates to **True**.

```
1 a = 5
2 b = 4
3 c = 9
4
5 if c > b: # legg merke til kolon (:) til slutt
6     print("Jepp, c er større enn b")
7
8 if a > c: # legg merke til kolon (:) til slutt
9     print("Denne teksten kommer ikke i output, siden
    vilkåret er False")
```

Utdata

Yep, c is greater than b

If you first want to check if a condition is true, and then perform actions if it's *not*, you can use the expression **else**:

```
1 a = 5
2 c = 9
3
4 if a > c: # legg merke til kolon (:) til slutt
5     print("Denne teksten kommer ikke i output, siden
    vilkåret er False")
6
7 else: # legg merke til kolon (:) til slutt
8     print("Men denne kommer, fordi vilkåret i if-linja
    over var False")
```

Utdata

But this comes because the condition in the if-line above was False

The expression **else** only considers (and doesn't make sense without) the **if** expression right above it. If we want actions to be performed

only if no previous `if` expressions produced any result, we must use¹ the expression `elif`. This is an `if` expression that takes effect if the `if` expression above did *not* take effect.

```
1 a = 2
2
3 if a > 3:
4     print("Denne linja printes ikke, vilkåret er False")
5
6 elif a < 1: #Siden if uttrykket over ikke ga utslag,
7             sjekkes vilkåret b < 1
8     print("Denne linja printes ikke, vilkåret er False")
9
10 else:
11     print("Nå er vi sikre på at 1 < b < 3")
```

Utdata

Now we are sure that $1 < b < 3$

Note

When working with numbers, some conditions you expect to be `True` might turn out to be `False`. This often deals with rounding errors, as mentioned on page 5.

¹`elif` is a shortcut for `else if`, which can also be used.

0.1.5 Lister

Lister kan vi bruke for å samle objekter. Objektene som er i listen kalles **elementene** til listen.

```
1 strings = ["98", "99", "100"]
2 floats = [1.7, 1.2]
3 ints = [96, 97, 98, 99, 100]
4 mixed = [1.7, 96, "100"]
5 empty = []
```

Elementene i lister er **indekserte**. Første objekt har indeks 0, andre objekt har indeks 1 og så videre:

```
1 strings = ["98", "99", "100"]
2 floats = [1.7, 1.2]
3 ints = [96, 97, 98, 99, 100]
4 mixed = [1.7, 96, "100"]
5 empty = []
```

Utdata

```
96
99
98
```

Med den innebygde funksjonen `append()` kan vi legge til et objekt i enden av listen. Dette er en **innebygd funksjon**¹, som vi skriver i enden av navnet på listen, med et punktum foran.

```
1 min_liste = []
2 print(min_liste)
3
4 min_liste.append(3)
5 print(min_liste)
6
7 min_liste.append(7)
8 print(min_liste)
```

Utdata

```
[]
[3]
[3, 7]
```

¹Kort fortalt betyr det at det bare er noen typer objekter som kan bruke denne funksjonen.

Med funksjonen `pop()` kan vi hente ut et objekt fra listen

```
1 min_liste = [6, 10, 15, 19]
2
3 a = min_liste.pop() # a = det siste elementet i listen
4 print("a =",a)
5 print("min_liste =",min_liste)
6
7 a = min_liste.pop(1) # a = elementet med indeks 1
8 print("a =",a)
9 print("min_liste =",min_liste)
```

Utdata

```
a = 19
min_liste = [6, 10, 15]
a = 10
min_liste = [6, 15]
```

Forklar for deg selv

Hva er forskjellen på å skrive `a = min_liste[1]` og å skrive `a = min_liste.pop(1)`?

Med funksjonen `sort()` kan vi sortere elementene i listen.

```
1 heltall = [9, 0, 8, 3, 1, 7, 4]
2 bokstaver = ['c', 'a', 'b', 'e', 'd']
3
4 heltall.sort()
5 bokstaver.sort()
6
7 print(heltall)
8 print(bokstaver)
```

Utdata

```
[0, 1, 3, 4, 7, 8, 9]
['a', 'b', 'c', 'd', 'e']
```

Med funksjonene `count()` kan vi telle gjentatte elementer i listen.

```
1 heltall = [2, 7, 2, 2, 2]
2 frukt = ['banan', 'eple', 'banan']
3
4 antall_toere = heltall.count(2)
5 antall_sjuere = heltall.count(7)
6 antall_bananer = frukt.count('banan')
7 antall_appelsiner = frukt.count('appelsin')
8
9 print(antall_toere)
10 print(antall_sjuere)
11 print(antall_bananer)
12 print(antall_appelsiner)
```

Utdata

4
1
2
0

Med funksjonen `len()` kan vi finne antall elementer i en liste, og med funksjonen `sum()` kan vi finne summen av lister med tall som elementer.

```
1 heltall = [2, 7, 2, 2, 2]
2 frukt = ['banan', 'eple', 'banan']
3
4 print(len(heltall))
5 print(len(frukt))
6 print(sum(heltall))
```

Utdata

5
3
15

Med uttrykket `in` kan vi sjekke om et element er i en liste.

```
1 heltall = [1, 2, 3]
2
3 print(1 in heltall)
4 print(0 in heltall)
```

Utdata

True

False

0.1.6 Looper; **for** og **while**

for loop

For objekter som inneholder flere elementer, kan vi bruke **for**-looper til å utføre handlinger for hvert element. Handlingene må vi skrive med et innrykk etter **for**-uttrykket:

```
1 min_liste = [5, 10, 15]
2
3 for number in min_liste:
4     print(number)
5     print(number*10)
6     print("\n") # lager et blankt mellomrom
7
```

Utdata

```
5
50

10
100

15
150
```

Språkboksen

Å gå gjennom hvert element i (for eksempel) en liste kalles "å iterere over listen".

Ofte er det ønskelig å iterere over heltallene 0, 1, 2 og så videre. Til dette kan vi bruke **range()**:

```
1 ints = range(3)
2
3 for i in ints:
4     print(i)
5
```

Utdata

```
0
1
2
```

while loop

Hvis vi ønsker at handlinger skal utføres fram til et vilkår er sant, kan vi bruke en **while**-loop:

```
1 a = 1
2
3 while a < 5:
4     print(a)
5     a += 1
```

Utdata

```
1
2
3
4
```

0.1.7 input()

Vi kan bruke funksjonen **input()** til å skrive inn tekst mens skriptet kjører:

```
1 innskrevet_tekst = input("Skriv inn her: ")
2 print(innskrevet_tekst)
```

Teksten vi har skrevet inni **input()** i skriptet over er teksten vi ønsker vist foran teksten som ønskes innskrevet. Linje 2 i denne koden vil ikke kjøres før en tekst er innskrevet.

```
1 innskrevet_tekst = input("Skriv inn her: ")
2 print(innskrevet_tekst)
```

Utdata

```
Skriv inn tekst her: OK
OK
```

Objektet gitt av en `input()`-funksjon vil alltid være av typen `str`. Man må alltid passe på å gjøre om objekter til rett type:

```
1 print("La oss regne ut a*b")
2 a_str = input("a = ")
3 b_str = input("b = ")
4 a = float(a_str)
5 b = float(b_str)
6 print("a*b = ", a*b)
```

Utdata

```
La oss regne ut a*b
a = 3.7
b = 4
a*b = 14.8
```

0.1.8 Feilmeldinger

Påstand: Alle programmerere vil erfare at skriptet ikke kjører fordi vi ikke har skrevet koden på rett måte. Dette kalles en **syntax error**. Ved syntax error vil man få beskjed om på hvilken linje feilen befinner seg, og hva som er feil. De vanligste feilene er

- Å glemme innrykk når man bruker metoder som `def`, `for`, `while`, og `if`

```
1 a = 472
2 b = 98
3
4 if a*b > 48000:
5 print("a*b er større enn 48000")
```

Utdata

```
line 5, in <module>
print("a*b er større enn 48000")
^
IndentationError: expected an indented block after
'if' statement on line 4
```

- Å utføre operasjoner på typer det ikke gir mening for


```
1 b = "98"  
2 b_opphøyd_i_andre = b**2
```

Utdata

```
line 2, in <module>
```

```
b_opphøyd_i_andre = b**2
```

```
TypeError: unsupported operand type(s) for ** or  
pow(): 'str' and 'int'
```