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` type[2]. 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:

`str`	text
`int`	integer
`float`	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 `#`.

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

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

```
a = 5
b = 2

print("a+b = ", a+b);
print("a-b = ", a-b);
print ("a*b = " ,a*b);
print ("a/b = " ,a/b);
print("a**b = " ,a**b); # potens med grunntall a og
                          eksponent b
print ("a//b = ", a//b); # a/b rundet ned til nærmeste
                          heltall
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`:

```
s = "2"
b = 3
c = 2.0

b_s = str(b) # b omgjort til str
c_s = str(c) # c omgjort til str
print(b_s+c_s)

s_i = int(s)
print(s_i*b)

s_f = float(s)
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'.

```
a = 5 # a ER nå 5
print(a)
a = a+1 # a ER nå det a VAR, + 1
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:

```
a = 5 # a ER nå 5
a+= 1 # Samme som å skrive a = a+1
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:

```
a = 8.3/10
print(a) # avrundingsfeil, da vi skulle hatt a = 0.83

a = round(a, 2) # runder av a til tall med to
    desimaler
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.

```
# a er en funksjon som ikke tar noen argumenter.
# Legg merke til 'def' først og ':' til slutt.
# Kodelinjene som hører til funksjonen må stå med innrykk
def a():
  print("Hei, noen kalte visst på funksjon a?")


# b er en funksjon som tar argumentet 'test'
def b(tekst):
  print("Hei. Noen kalte på funksjon b. Argumentet som ble gitt var: ", tekst)

# c er er funksjon som tar argmunentene a og b
# c returnerer et objekt
def c(a, b):
  return a+b

b("Hello!") # vi kaller på b med argumentet "hello"

d = c(2,3) # Vi kaller på a med argumentene 2 og 3

print(d)

# merk at teksten gitt i a ikke blir printet, fordi vi ikke har kalt på a.
```

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
==	is equal to
!=	is *not* equal to
>	is greater than
>=	is greater than, or equal to
<	is less than
<=	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`.

```
a = 5
b = 4
c = 9

if c > b: # legg merke til kolon (:) til slutt
  print("Jepp, c er større enn b")

if a > c: # legg merke til kolon (:) til slutt
  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`:

```
a = 5
c = 9

if a > c: # legg merke til kolon (:) til slutt
  print("Denne teksten kommer ikke i output, siden
    vilkåret er False")

else: # legg merke til kolon (:) til slutt
  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 use[1] the expression `elif`. This is an `if` expression that takes effect if the `if` expression above did *not* take effect.

```
a = 2

if a > 3:
  print("Denne linja printes ikke, vilkåret er False")

elif a < 1: #Siden if uttrykket over ikke ga utslag,
              sjekkes vilkåret b < 1
  print("Denne linja printes ikke, vilkåret er False")

else:
  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 Lists

Lists can be used to collect objects. The objects in the list are called the **elements** of the list.

```
strings = ["98", "99", "100"]
floats = [1.7, 1.2]
ints = [96, 97, 98, 99, 100]
mixed = [1.7, 96, "100"]
empty = []
```

The elements in lists are **indexed**. The first object has index 0, the second object has index 1, and so on:

```
strings = ["98", "99", "100"]
floats = [1.7, 1.2]
ints = [96, 97, 98, 99, 100]
mixed = [1.7, 96, "100"]
empty = []
```

Utdata

```
96
99
98
```

Using the built-in function `append()` we can add an object to the end of the list. This is an **in-built function**[1], which we write at the end of the list name, preceded by a dot.

```
min_liste = []
print(min_liste)

min_liste.append(3)
print(min_liste)

min_liste.append(7)
print(min_liste)
```

Utdata

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

[1] In short, it means that only certain types of objects can use this function.

With the `pop()` function, we can retrieve an object from the list.

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

a = min_liste.pop() # a = det siste elementet i listen
print("a =",a)
print("min_liste =",min_liste)

a = min_liste.pop(1) # a = elementet med indeks 1
print("a =",a)
print("min_liste =",min_liste)
```

Utdata

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

Explain to yourself

What's the difference between writing `a = min_liste[1]` and `a = min_liste.pop(1)`?

With the `sort()` function, we can sort the elements in the list.

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

heltall.sort()
bokstaver.sort()

print(heltall)
print(bokstaver)
```

Utdata

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

With the `count()` function, we can count repeated elements in the list.

```
heltall =  [2, 7, 2, 2, 2]
frukt = ['banan', 'eple', 'banan']

antall_toere = heltall.count(2)
antall_sjuere = heltall.count(7)
antall_bananer = frukt.count('banan')
antall_appelsiner = frukt.count('appelsin')

print(antall_toere)
print(antall_sjuere)
print(antall_bananer)
print(antall_appelsiner)
```

Utdata

```
4
1
2
0
```

With the `len()` function, we can find the number of elements in a list, and with the `sum()` function, we can find the sum of lists with numbers as elements.

```
heltall =  [2, 7, 2, 2, 2]
frukt = ['banan', 'eple', 'banan']

print(len(heltall))
print(len(frukt))
print(sum(heltall))
```

Utdata

```
5
3
15
```

With the `in` expression, we can check if an element is in a list.

```
heltall =  [1, 2, 3]

print(1 in heltall)
print(0 in heltall)
```

Utdata

```
True
False
```

0.1.6 Loops; `for` and `while`

`for` loop

For objects containing multiple elements, we can use `for` loops to perform actions for each element. The actions must be written with an indentation after the `for` statement:

```
min_liste = [5, 10, 15]

for number in min_liste:
  print(number)
  print(number*10)
  print("\n") # lager et blankt mellomrom

```

Utdata

```
5
50

10
100

15
150
```

Språkboksen

Going through each element in (for example) a list is called "iterating over the list".

Often, it's desired to iterate over the integers $0, 1, 2$ and so forth. For this, we can use `range()`:

```
ints = range(3)

for i in ints:
  print(i)

```

Utdata

```
0
1
2
```

`while` loop

If we want actions to be performed until a condition is met, we can use a `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()`

We can use the `input()` function to enter text while the script is running:

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

The text written inside `input()` in the script above is the text we want displayed before the text to be entered. Line 2 of this code will not execute until text is entered.

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

Utdata

```
Enter text here: OK
OK
```

The object provided by an `input()` function will always be of type `str`. One must always ensure to convert objects to the correct type:

```
print("La oss regne ut a*b")
a_str = input("a = ")
b_str = input("b = ")
a = float(a_str)
b = float(b_str)
print("a*b = ", a*b)
```

Utdata

```
Let's calculate a*b
a = 3.7
b = 4
a*b = 14.8
```

0.1.8 Error Messages

Claim: All programmers will experience that the script does not run because we haven't written the code correctly. This is called a **syntax error**. With a syntax error, you will be informed about which line the error is on and what the error is. The most common errors are:

- **Forgetting indentation when using methods like `def`, `for`, `while`, and `if`**

```
a = 472
b = 98

if a*b > 48000:
print("a*b er større enn 48000")
```

Utdata

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

- **Performing operations on types where it doesn't make sense**

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

Utdata

```
line 2, in <module>
b_raised_to_second = b**2
TypeError: unsupported operand type(s) for ** or
pow(): 'str' and 'int'
```