Outline

- control flow
- function
- module

Control flow

if-else statement ¶

```
 if condition1:
    statement1_1
    statement1_2
    ...
elif condition2:
    statement2_1
    statement2_2
    ...
...
else:
    statementn_1
    statementn_2
    ...
```

In [1]:

```
x=input("Give an integer: ")
x=int(x)
if x >= 0:
    a=x
else:
    a=-x
print("The absolute value of %i is %i"  % (x, a))
```

```
Give an integer: 3
The absolute value of 3 is 3
```

In [2]:

```
c=float(input("Give a number: "))
if c > 0:
    print("c is positive")
elif c<0:
    print("c is negative")
else:
    print("c is zero")
```

```
Give a number: 4
c is positive
```

loops

In Python we have two kinds of loops: `while` and `for` .

In [3]:

```
i=1
while i*i < 200:
    print("Square of", i, "is", i*i)
    i = i + 1
print("Finished printing all the squares below 200.")
```

```
Square of 1 is 1
Square of 2 is 4
Square of 3 is 9
Square of 4 is 16
Square of 5 is 25
Square of 6 is 36
Square of 7 is 49
Square of 8 is 64
Square of 9 is 81
Square of 10 is 100
Square of 11 is 121
Square of 12 is 144
Square of 13 is 169
Square of 14 is 196
Finished printing all the squares below 200.
```

In [4]:

```
s=0
for i in [0,1,2,3,4,5,6,7,8,9]:
    s = s + i
print("The sum is", s)
```

```
The sum is 45
```

In [5]:

```
s=0
for i in range(10):
    s = s + i
print("The sum is", s)
```

```
The sum is 45
```

In [6]:

```
for i in range(3,7):
    print(i)
```

```
3
4
5
6
```

In [7]:

```
for i in range(3,7,2):
    print(i)
```

```
3
5
```

Breaking and continuing loop

- Breaking the loop, with the `break` statement
- Stopping current iteration and continuing to the next one with the `continue` statement

In [8]:

```
l=[1,3,65,3,-1,56,-10]
for x in l:
    if x < 0:
        break
print("The first negative list element was", x)
```

```
The first negative list element was -1
```

In [9]:

```
from math import sqrt, log
l=[1,3,65,3,-1,56,-10]
for x in l:
    if x < 0:
        continue
    print(f"Square root of {x} is {sqrt(x):.3f}")
    print(f"Natural logarithm of {x} is {log(x):.4f}")
```

```
Square root of 1 is 1.000
Natural logarithm of 1 is 0.0000
Square root of 3 is 1.732
Natural logarithm of 3 is 1.0986
Square root of 65 is 8.062
Natural logarithm of 65 is 4.1744
Square root of 3 is 1.732
Natural logarithm of 3 is 1.0986
Square root of 56 is 7.483
Natural logarithm of 56 is 4.0254
```

Function

Functions

A function is defined with the `def` statement.

In [10]:

```
def double(x):
    "This function multiplies its argument by two."
    return x*2
print(double(4), double(1.2), double("abc")) # It even happens to work for strings!
```

```
8 2.4 abcabc
```

```
In [11]: help(double)
```

```
Help on function double in module __main__:

double(x)
    This function multiplies its argument by two.
```

Set defalut value to argument

```
In [12]: def double(x=2):
             "This function multiplies its argument by two."
             return x*2

         double()
```

```
Out[12]: 4
```

Return multiple outputs

```
In [13]: def dou_tri(x=2):
             "This function multiplies its argument by two."
             return x*2, x*3

         dou_tri()
```

```
Out[13]: (4, 6)
```

```
In [14]: def dou_tri(x=2):
             "This function multiplies its argument by two."
             return { 'double': x*2, 'triple': x*3}

         dou_tri()
```

```
Out[14]: {'double': 4, 'triple': 6}
```

Global and local evironments

```
In [15]: def myfun():
             eggs = 10
         myfun()
         print(eggs)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-15-8a1a5dd8b584> in <module>()
      2     eggs = 10
      3 myfun()
----> 4 print(eggs)

NameError: name 'eggs' is not defined
```

```
In [16]: def myfun2():
             print(eggs)
         eggs = 10
         myfun2()
```

```
10
```

Example

```
In [17]: def sum_of_squares(a, b):
             "Computes the sum of arguments squared"
             return a**2 + b**2
         print(sum_of_squares(3, 4))
```

```
25
```

- It would be nice that the number of arguments could be arbitrary, not just two. We could pass a list to the function as a parameter.

Example (Cont.)

```
In [18]: def sum_of_squares(lst):
             "Computes the sum of squares of elements in the list given as parameter"
             s=0
             for x in lst:
                 s += x**2
             return s
         print(sum_of_squares([-2]))
         print(sum_of_squares([-2,4,5]))
```

```
4
45
```

- There is however some extra typing with the brackets around the lists.
- *A better solution?*

Example (Cont.)

```
In [19]: def sum_of_squares(*t):
             "Computes the sum of squares of arbitrary number of arguments"
             s=0
             for x in t:
                 s += x**2
             return s
         print(sum_of_squares(-2))
         print(sum_of_squares(-2,4,5))
```

```
4
45
```

- The star(*) is called **argument packing**.
- It packs all the given positional arguments into a tuple `t`.

Map function

The `map` function gets a list and a function as parameters, and it returns a new list whose elements are elements of the original list transformed by the parameter function.

```
In [20]: s="12 43 64 6"
         L=s.split()        # The split method of the string class, breaks the string at whitespaces
                            # to a list of strings.
         print(L)
         int(L)
```

```
['12', '43', '64', '6']

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-20-52acf90b84ec> in <module>()
      3                    # to a list of strings.
      4 print(L)
----> 5 int(L)

TypeError: int() argument must be a string, a bytes-like object or a number, not 'list'
```

```
In [21]: print(list(map(int, L)))  # The int function converts a string to an integer
```

```
[12, 43, 64, 6]
```

lambda function

- Sometimes it feels unnecessary to write a function.
- Use an expression called *lambda* to define a function with no name
- `lambda param1,param2, ... : expression`

```
In [22]: L=[2,3,5]
         list(map(lambda x : 2*x+x**2, L))
```

```
Out[22]: [8, 15, 35]
```

Filter function

- The `filter` function takes a function and a list as parameters.
- The parameter function must take exactly one parameter and return a truth value (True or False)
- The `filter` function creates a new list with only those elements from the original list for which the parameter function returns True.

```
In [23]: def is_odd(x):
             """Returns True if x is odd and False if x is even"""
             return x % 2 == 1         # The % operator returns the remainder of integer division
         L=[1, 4, 5, 9, 10]
         list(filter(is_odd, L))

Out[23]: [1, 5, 9]
```

Reduce function

- The `sum` function that returns the sum of a numeric list, can be though to reduce a list to a single element.
- It does this reduction by repeatedly applying the `+` operator until all the list elements are consumed.
- For instance, the list `[1,2,3,4]` is reduced by the expression `(((0+1)+2)+3)+4` of repeated applications of the `+` operator.

```
In [24]: L=[1,2,3,4]
         from functools import reduce   # import the reduce function from the functools module
         reduce(lambda x,y:x+y, L, 0)    # 0 is the starting value

Out[24]: 10
```

```
In [25]: reduce(lambda x,y:x*y, L, 1)

Out[25]: 24
```

This corresponds to the sequence `(((1*1)*2)*3)*4` of application of operator `*`

Modules

Modules

- To ease management of large programs, software is divided into smaller pieces. In Python these pieces are called **modules**.
- A module should be a unit that is as independent from other modules as possible.
- Each file in Python (with extension `.py`) corresponds to a module.
- Modules can contain classes, objects, functions, ...
- The standard library of Python consists of hundreds of modules. Some of the most common standard modules include
  - `re`
  - `math`
  - `random`
  - `os`
  - `sys`
- Import modules
  - `import module`
  - `import module as m`
  - `from module import fun1, fun2`

Example

```
In [26]: math.cos(1)

         ---------------------------------------------------------------------------
         NameError                                 Traceback (most recent call last)
         <ipython-input-26-2297999f6c16> in <module>()
         ----> 1 math.cos(1)

         NameError: name 'math' is not defined
```

```
In [27]: import math
         math.cos(1)

Out[27]: 0.5403023058681398
```

```
In  [28]: from math import cos
          cos(1)

Out[28]:  0.5403023058681398
```

Example

```
In  [29]: import math
          math.sqrt(3)

Out[29]:  1.7320508075688772
```

```
In  [30]: from math import sqrt
          sqrt(3)

Out[30]:  1.7320508075688772
```

```
In  [31]: import math as shuxue
          shuxue.sqrt(3)

Out[31]:  1.7320508075688772
```

Important libraries of Python

- numpy: Numerical Python. Basic library for numerical analysis
- pandas: panel data. Provides data.frame
- matplotlib: data visualization
- scikit-learn: machine learning including classification, regression(Lasso), clustering, PCA, model selection
- statsmodels: statistical modeling including regression, ANOVA, time series, density estimation
- SciPy: Scientific Python, including integrate, (sparse) matrix decomposition, optimization.

Numpy provides a basic data structure for numerical analysis, and enables storing and handling data in an efficient way.

```
In  [32]: import numpy as np
          my_arr = np.arange(10000000)
          my_list = range(10000000)

          %time a = my_arr*2

          Wall time: 16 ms
```

```
In  [33]: b = my_list*2

          ---------------------------------------------------------------------------
          TypeError                                 Traceback (most recent call last)
          <ipython-input-33-5c01f15ebca9> in <module>()
          ----> 1 b = my_list*2

          TypeError: unsupported operand type(s) for *: 'range' and 'int'
```

```
In  [34]: b = list(my_list) * 2
          len(b)

Out[34]:  20000000
```

```
In  [35]: %time b = [x*2 for x in my_list]

          Wall time: 1.22 s
```