Use of Python functions

```
      # We used some simple builtin functions before, like:
F1001 >>> min(9, 4, 6, 8)  # return the smallest value from a list
      ==> 4
F1002 >>> drinks = ["whiskey", "beer", "wine", "juice", "water"]  # define a list
F1003 >>> sorted(drinks)  # returns a new list
      ==> ['beer', 'juice', 'water', 'whiskey', 'wine']
F1004 >>> print("hello world")
      p() hello world
F1005 >>> round(5.53703, 2)
      ==> 5.54
F1006 >>> bool(2+4==6)
      ==> True
```

Create our own function

```
        # let's create a function with the name 'foo'. Does it already exist?
F1007 >>> foo
      err!NameError("name 'foo' is not defined",)
        # to create our own functions, we use the 'def' statement (=define)
F1008 >>> def foo():
      ...     print('this is the foo-function')
        # the code inside the function is indented one level
        # the function is now defined, python knows its name:
F1009 >>> foo
      ==> <function foo at 0x00000000033F8730>
        # now we call the function - using the name with parenthesis
F1010 >>> foo()
      p() this is the foo-function
        # we see: calling a function means to execute the statements inside
```

Functions can take arguments

```
F1011 >>> def foo(text):
      ...     print("foo was called with '{}'".format(text))
        # we just used the same name for new function. Now call it:
F1012 >>> foo('stars')
      p() foo was called with 'stars'

        # several arguments can be used:
F1013 >>> def foo(arg1, arg2):
      ...     print("called with arguments: {}, {}".format(arg1, arg2))
F1014 >>> foo(987, 38)
      p() called with arguments: 987, 38
```

Functions can return results

```
        # calculate and return a value
F1015 >>> def square(arg1):
      ...     prod = arg1 * arg1
      ...     return prod
F1016 >>> square(16)
      ==> 256

F1017 >>> def multiply(fact1, fact2):
      ...     return fact1 * fact2
F1018 >>> multiply(7, 13)
      ==> 91

        # try a special function:
F1019 >>> def python_is_cool():
      ...     return True
F1020 >>> python_is_cool()  # no further argument is needed :-)
      ==> True
```

Return multiple values

```
F1021 >>> drinks = ["whiskey", "beer", "wine", "juice", "water"]  # define a list

F1022 >>> def first_and_last(sequ):  # take a sequence (like a list)
      ...     first = sequ[0]    # get the first element
      ...     last = sequ[-1]    # and the last
      ...     return first, last  # return more than one value
F1023 >>> first_and_last(drinks)  # this returns a tuple
      ==> ('whiskey', 'water')

F1024 >>> df, dl = first_and_last(drinks)  # 'unpack' the tuple
F1025 >>> dl
      ==> 'water'

F1026 >>> both = first_and_last("LISBOA")  # yes, a string can be used like a list
F1027 >>> both   # the tuple was assigned to a variable
      ==> ('L', 'A')
```

Return from a function

```
        # A function does not need a return statement
F1028 >>> def noreturn(text):
...           print("called with:", text)
F1029 >>> result = noreturn('some text')
p()   called with: some text
F1030 >>> str(result)  # Show the 'None' result
==>   'None'

        # A function can have more than  one return statement
F1031 >>> def decide(p1, p2):
...           if p2 == True:
...               return p1
...           print('no', p1)
...           return False
F1032 >>> decide("success", True)
==>   'success'
F1033 >>> decide("success", False)
p()   no success
==>   False
```

Function arguments with defaults (1)

```
        # An argument can have a default (predefined) value
F1034 >>> def decide(p1, p2=False):
      ...     if p2 == True:
      ...         return p1
      ...     print('no', p1)
      ...     return False
F1035 >>> decide("success")  # argument p2 is not specified on call
      p() no success
      ==> False
F1036 >>> decide("success", True)
      ==> 'success'
```

Function arguments with defaults (2)

```
        # Arguments without a default value must be specified at the function call
F1037 >>> def manyargs(name, number, third="3rd", fourth=None, fifth="5th"):
      ...     print("name:{}, number:{}, third:{}, fourth:{}, fifth:{}"
      ...           .format(name, number, third, fourth, fifth))
F1038 >>> manyargs('Tom', 17)
      p() name:Tom, number:17, third:3rd, fourth:None, fifth:5th
F1039 >>> manyargs('Tom', 17, 'drei')  # specify arguments by position
      p() name:Tom, number:17, third:drei, fourth:None, fifth:5th
F1040 >>> manyargs('Tom', 17, fourth='vier')  # specifiy an argument by name
      p() name:Tom, number:17, third:3rd, fourth:vier, fifth:5th
F1041 >>> manyargs('Tom', number=17, fifth='fünf')  # specify by name also for args without default
      p() name:Tom, number:17, third:3rd, fourth:None, fifth:fünf

        # On a function definition:
        #       First arguments without defaults, then arguments with defaults
        # On a function call:
        #       First arguments without names (positional) then args with names (keyword)
```

Functions used for a general program structure

```
        #  Most of the code of a program should be enclosed in functions,
        #  starting with a 'main()' function, followed by other function definitions
        #  The last statement in the program is then the call of the main function.

F1042 >>> # python3  #
      ... """ This is a sample for the general form
      ...     of a python program
      ... """
      ... import random   # import statements
      ...
      ... def main():
      ...     print("start of main function")
      ...     result = sample('test')
      ...     print("result:", result)
      ...
      ... def sample(arg1):
      ...     print("start of sample function")
      ...     return arg1 +' '+ str(random.random() * 999)
      ...
      ... print("call of the main function")
      ... main()
      ... #
    p() call of the main function
    p() start of main function
    p() start of sample function
    p() result: test 902.8477380982321
```