Functions and Modules

Berner Fachhochschule

What is a function

- A function is a procedure that converts the entered values (parameters/arguments) into a result using processing instructions.
- The function definition specifies how to calculate the result from the arguments.
- Example:

```
def positiveSum(a,b):
    result = a + b
    if result < 0:
        result = -result
    return result
```

Function signature

Processing
Instructions

Return value

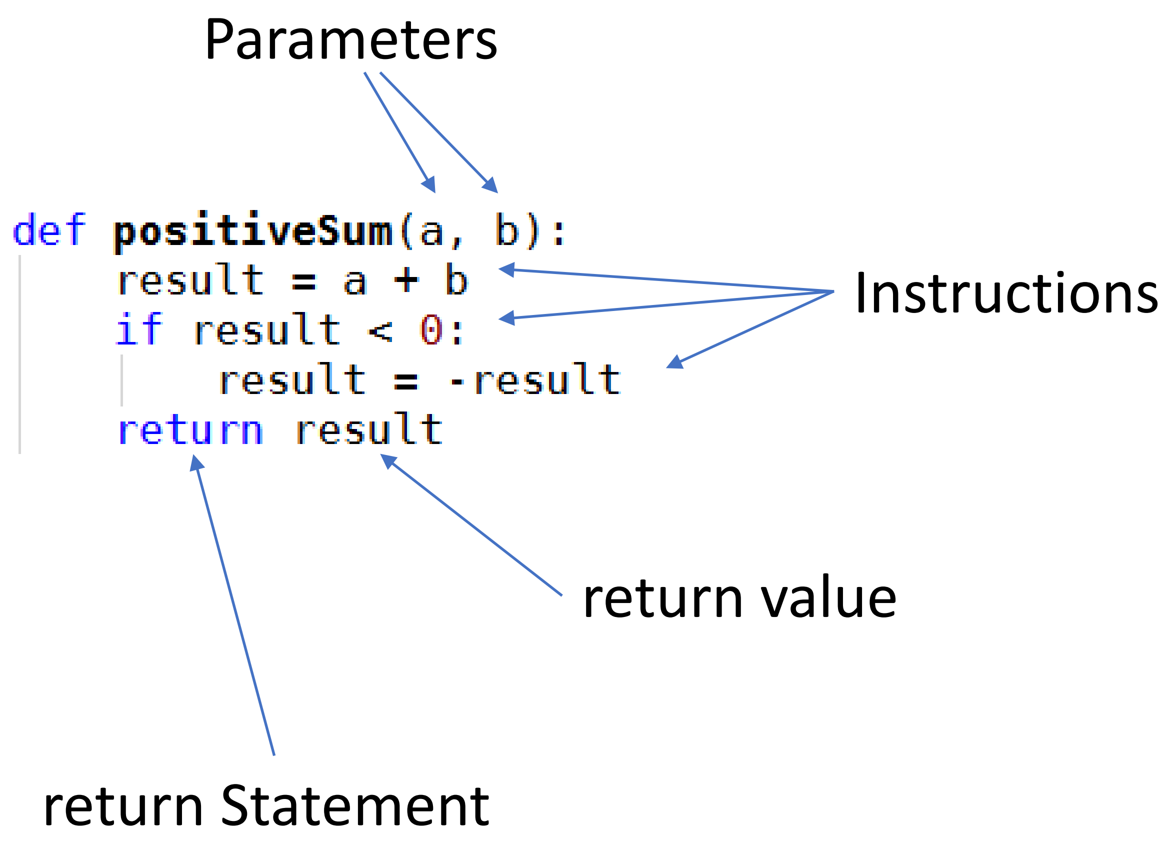
Writing functions

- A function is started in Python with the keyword def

```
def functionName(parameter list):
    instructions
    return result
```

- The parameter list consists of one or more identifiers separated by commas.
- The functional body consists of instructions
- A return statement ends the function call
- The value that returns the result of the function stands after the return keyword.

Terms



```python
def positiveSum(a, b):
    result = a + b
    if result < 0:
        result = -result
    return result
```

Using Functions

A function is called using the functions name. The parameters are assigned by matching arguments (values).

```
result = positiveSum(7, 3)
print(result)
```

Resultat 10

(7, 3)

```
def positiveSum(a, b):
    result = a + b
    if result < 0:
        result = -result
    return result
```

The parameter values (arguments) can also be assigned by previously defined variables.

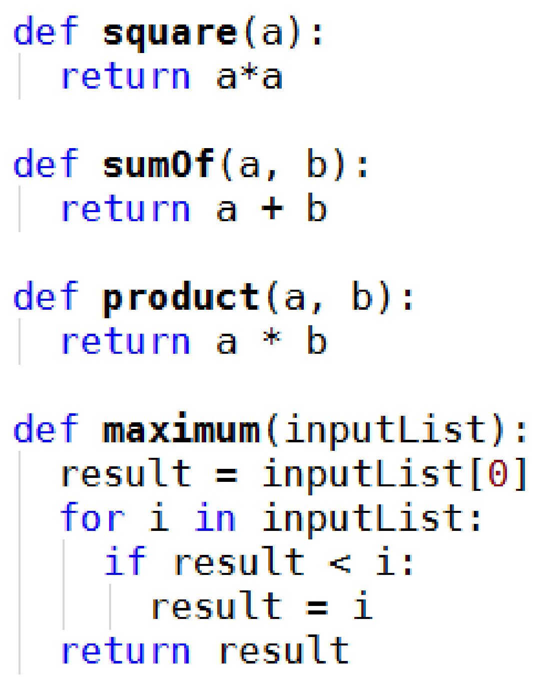
```
p1 = -9
p2 = 2
result = positiveSum(p1, p2)
print(result)
```

Resultat 7

(p1, p2)

```
def positiveSum(a, b):
    result = a + b
    if result < 0:
        result = -result
    return result
```

Examples of functions



```
def square(a):
    return a*a

def sumOf(a, b):
    return a + b

def product(a, b):
    return a * b

def maximum(inputList):
    result = inputList[0]
    for i in inputList:
        if result < i:
            result = i
    return result
```

Definition of the square function

Definition of a function for the sum of two numbers

Definition of a function to calculate the product of two numbers

Definition of a function to determine the maximum of all values in a list

Using functions

```
x = 4
y = 7
print("x =", x)
print("y =", y)

resultat = square(x)
print("square(x)=", resultat)

resultat = sumOf(x, y)
print("sumOf(x, y)=", resultat)

resultat = product(x, y)
print("product(x, y)=", resultat)

inputList = [3,6,8,2,1]
print(inputList)
print("Maximum=", maximum(inputList))
```

Using the square function

Using the sum function

Using the product function

Using the max function

Result

```
x = 4
y = 7
quadrat(x)= 16
summe(x, y)= 11
produkt(x, y)= 28
[3, 6, 8, 2, 1]
Maximum= 8
```

Examples of functions

```
def nearThirty(n):
    return abs(30-n) <= 2

def evenNumbers(inputList):
    count = 0
    for x in inputList:
        if x % 2 == 0:
            count = count + 1
    return count

def average(*numbers):
    sum=0
    for x in numbers:
        sum = sum + x
    result = round(sum/len(numbers),2)
    return result
```

Returns True if n is near 30.

Counts the number of even numbers in the list.

Average value of the given numbers (any number of)

Using the functions

```
d1 = average(2,11,1,19,4)
print("The average of 2,11,1,19,4 =", d1)

d2 = average(3,6,2,7,1,9,2)
print("The average of 3,6,2,7,1,9,2 =", d2)
```

Using the average function

```
a = evenNumbers(list1)
print("Even numbers in list =", a)

print("31 near 30?", nearThirty(31))
print("27 near 30?", nearThirty(27))
```

Using the evenNumbers function

Using the almostThirty function

Result

```
The average of 2,11,1,19,4 = 7.4
The average of 3,6,2,7,1,9,2 = 4.29
Even numbers in list = 2
31 near 30? True
27 near 30? False
```

Arguments and parameters

```python
def boxVolume(h, b, l):
    volumen = h * b * l
    return volumen
```

Parameters: h, b, l
for the height, width and length of the box

```python
v = boxVolume(10,12,8)
print("Box volume =", v)
```

Arguments: Height = 10, Width = 12, Length = 8

The values 10, 12 and 8 are each used as values in the variables h, b and l

Result

```
Volumen = 960
```

Arguments and Parameter

```
def commonElements(l1, l2):
    set1 = set(l1)
    set2 = set(l2)
    return list(set1 & set2)
```

Parameters l1 and l2 (two lists)

Arguments list1 and list2 -> the two input lists

```
list1=[2,11,1,19,4]
list2=[3,6,2,7,1,9,2]

print("list1: ", list1)
print("list2: ", list2)
common = commonElements(list1, list2)
print("In both lists: ", common)
```

Result

```
list1:  [2, 11, 1, 19, 4]
list2:  [3, 6, 2, 7, 1, 9, 2]
In both lists:  [1, 2]
```

Berner Fachhochschule

Any number of arguments

```
def average(*zahlen):
    sum = 0
    for x in zahlen:
        sum = sum + x
    result = round(sum / len(zahlen), 2)
    return result
```

The star means that any number of arguments can be passed.

5 arguments

7 arguments

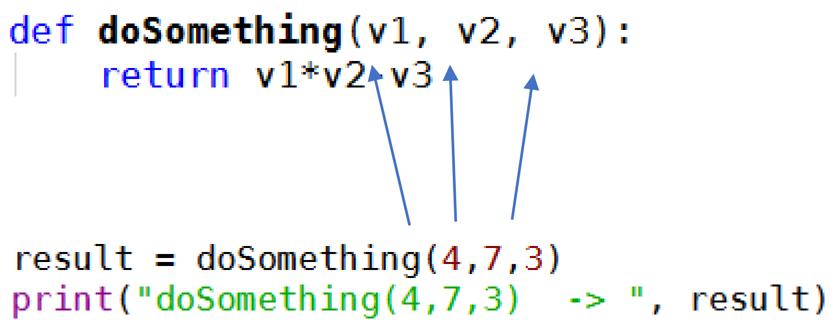
```
d1 = average(2,11,1,19,4)
print("The average of 2,11,1,19,4 =", d1)

d2 = average(3,6,2,7,1,9,2)
print("The average of 3,6,2,7,1,9,2 =", d2)
```

Result

```
The average of 2,11,1,19,4 = 7.4
The average of 3,6,2,7,1,9,2 = 4.29
```

Assign arguments

The arguments of a function can be assigned explicitly in any order.

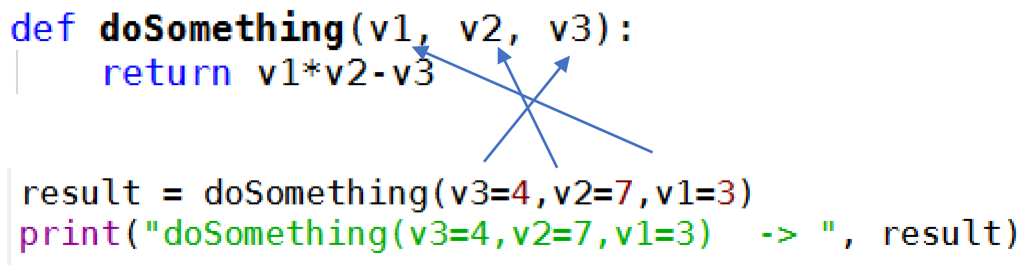


```
def doSomething(v1, v2, v3):
    return v1*v2-v3

result = doSomething(4,7,3)
print("doSomething(4,7,3)  -> ", result)
```

The variable v1 is given a value of 4, the variable v2 is 7, the variable v3 is 3

Result

```
something(4,7,3)  ->  25
```



```
def doSomething(v1, v2, v3):
    return v1*v2-v3

result = doSomething(v3=4,v2=7,v1=3)
print("doSomething(v3=4,v2=7,v1=3)  -> ", result)
```

The variable v1 is given a value of 3, the variable v2 is 7, the variable v4 is 3

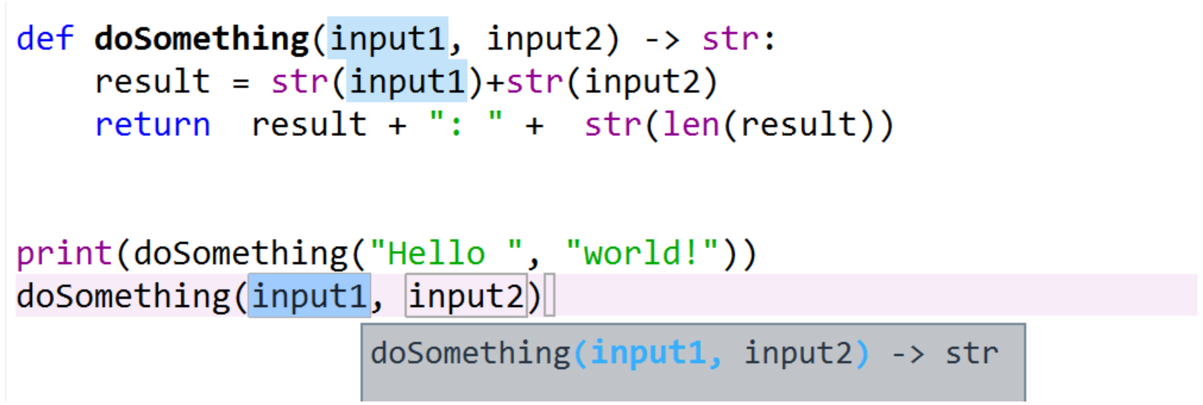
Result

```
something(v3=4,v2=7,v1=3)  ->  17
```

Berner Fachhochschule

Support return type

Since Python 3.5 you can give the user of a function a type hint for the return type of a function:



```
def doSomething(input1, input2) -> str:
    result = str(input1)+str(input2)
    return  result + ": " +  str(len(result))


print(doSomething("Hello ", "world!"))
doSomething(input1, input2)
                doSomething(input1, input2) -> str
```

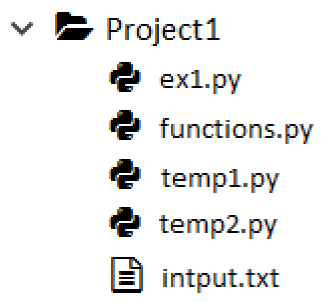
The user gets information about the expected return type of this function call.

But: no check is performed, whether the result element has the correct type.

Outsourcing functions to modules

A module is a file that contains python definitions and instructions.

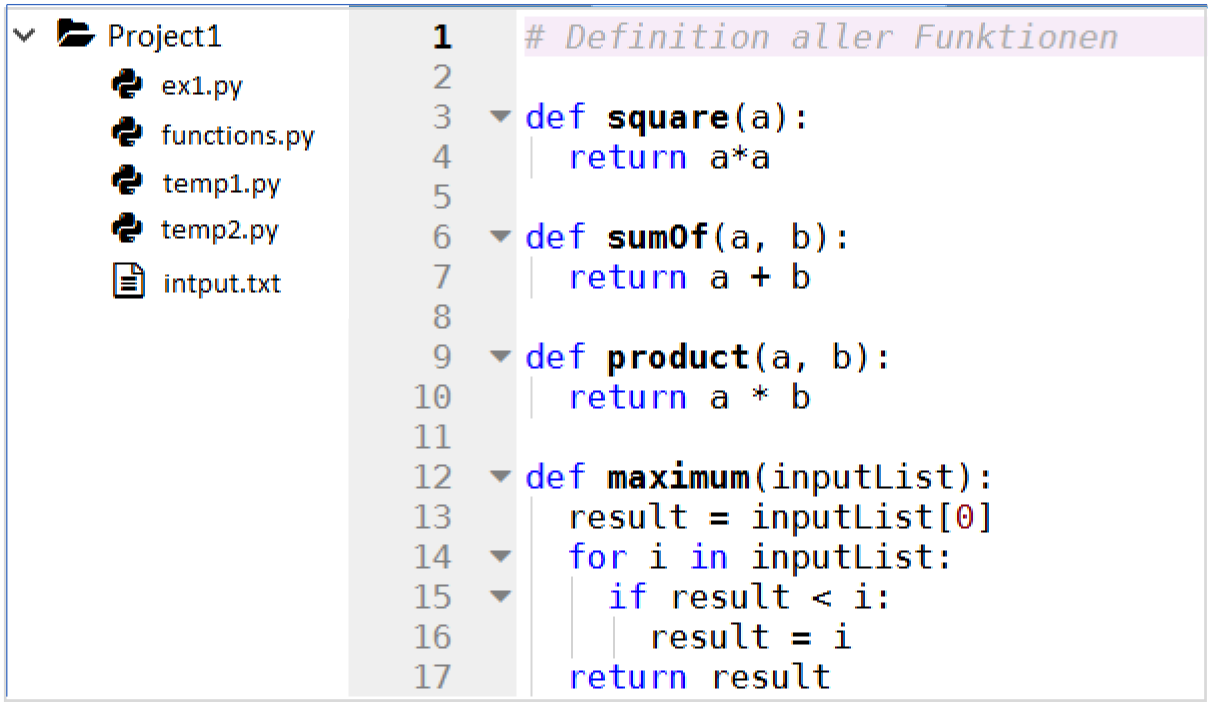
The file name is the module name with the appended suffix .py.



```
Project1
    ex1.py
    functions.py
    temp1.py
    temp2.py
    intput.txt
```

All functions are collected in one file.
The file name is arbitrary.

Collection of function definitions



```
# Definition aller Funktionen

def square(a):
    return a*a

def sumOf(a, b):
    return a + b

def product(a, b):
    return a * b

def maximum(inputList):
    result = inputList[0]
    for i in inputList:
        if result < i:
            result = i
    return result
```

Reading modules

A module can be read into another file with the "import" instruction. This means that all defined functions are known in the new file.

```
import functions
d = functions.average(3, 6, 2, 7, 1, 9, 2)
print("The avarage value of 3, 6, 2, 7, 1, 9, 2=", round(d, 1))
```

For better usability, an abbreviation is often defined for the name of the file.

```
import functions as fc

d = fc.average(3, 6, 2, 7, 1, 9, 2)
print("The avarage value of 3, 6, 2, 7, 1, 9, 2=", round(d, 1))
```

Berner Fachhochschule

Importing modules

All functions defined in the module can be used after the import.

```
import functions as fc
d = fc.average(3, 6, 2, 7, 1, 9, 2)
print("The avarage value of 3, 6, 2, 7, 1, 9, 2=", round(d, 1))


print("Sqare value of 4 =", fc.square(4))

print("Sum of 143.5 and 23.42 =", fc.sumOf(143.4, 23.42))
```

Funktionen aus dem importierten Modul