Informatik 1 - Biomedical Engineering

Tutor Session 4 - Functional Programming

Overview

- What are functions?
- Defining functions in python
- Arguments and keword arguments
- Variable scope
- Return values
- Lambda functions

What are functions?

- Facilitate reusing code snippets
- Enable code structuring
- Quick changing of code throughout the program without copy/paste

Defining functions in python

```
# Use "def" to create new functions
def function_name(arg1, arg2, ..., argN):
    # do something
    return something # optional!
```

- input arguments (0 - n) and keyword arguments - optional
- return values (0 - n) - optional
- indentation (think of control structures)

In [ ]:
```
def hello_world():
    print("Hello World!")
```

In [ ]:
```
hello_world()
```

Arguments and keword arguments

In [ ]:
```
def subtract(x, y):
    return x - y
```

In [ ]:
```
subtract(5,7)
```

In [ ]:
```
subtract(5) # one argument missing -> TypeError
```

In [ ]:
```
subtract(y=5, x=7) # -> different order of arguments, by using the names explici
tly
```

Keyword argument:

- optional when calling the function
- have default values
- default values are overwritten when keyword argument is used in call
- have to be specified *after* non-keyword (=positional) arguments

In [ ]:
```
def root(number, degree=2):
    return number**(1/degree)
```

In [ ]:
```
root(2)
```

In [ ]:
```
root(10, degree=3)
```

In [ ]:
```
root(10,7)
```

In [ ]:
```
def all_the_args(*args, **kwargs):
    print(args)
    print(kwargs)

args = (1, 2, 3, 4)
kwargs = {"a": 3, "b": 4}
all_the_args(*args, **kwargs)
```

- "*" is used to expand tuples
- "**" to expand dictionaries
- this way, you can pass an arbitrary number of arguments and keyword arguments to a function
- the asterisks (stars) have to be there for the function call too!
- if you use positional arguments, a non-keyworded list and keyworded arguments together, the order has to be like this:

```
func(fargs, *args, **kwargs)
```

Variable scope

In [ ]:
```
x = 5
y = "hello"
```

In [ ]:
```
def set_x(n):
    print(y) # can be accessed
    x = n    # this creates a new "x" that only lives inside then function
    print(x) # this uses the local x

set_x(10)
print(x) # here, outside the function, x is still 5
```

In [ ]:
```
def set_global_x(n):
    global x   # now x inside the function is the same as outside
    x = n      # global var x is now set to n
    print(x)

set_global_x(10)
print(x)
```

- variables from "outside" can be read, but not changed
- assigning a value creates a new *local* variable
- to change the *global* value, the keyword "global" has to be used
- trying to access a global variable when there's a local one with the same name created afterwards raises an UnboundLocalError - confusing, so:

In [ ]:
```
def set_x_(n):
    print(y) # this works
    print(x) # this does not, because there will be a local x later
    x = n

set_x_(10)
```

Return values

- python functions are of type "void" by default, so the return value is optional
- multiple values can be returned (even of different types)
- return can be used on its own to break out of a function prematurely

In [ ]:
```
def sqrt(x):
    return x**0.5
```

In [ ]:
```
print(f"The square root of 7 is {sqrt(7)}")
```

In [ ]:
```
def swap(x, y):
    return y, x # multiple return values (implicitly creates a tuple)
    # return (y, x) -> this would be the same
```

In [ ]:
```
a = 10
b = 20
print(f"a equals {a}, b equals {b}")
a, b = swap(a, b)
print(f"a equals {a}, b equals {b}")
```

In [ ]:
```
def foo():
    return "hallo", 12   # different types

a, b = foo()
print(a)
print(b)
```

In [ ]:
```
def times_table(limit):
    for i in range(1, 11):
        for j in range(1, 11):
            print(f"{i} * {j} = {i*j}")
            if j == limit:
                # break   # only jumps out of inner loop
                return  # ends the function

times_table(5)
```

Unpacking

In [ ]:
```
a, b, c = (1, 2, 3)
print(a)
print(b)
print(c)
```

In [ ]:
```
a, *b, c = 1, 2, 3, 4, 5, 6   # no parentheses needed to create a tuple
print(a)
print(b)
print(c)
```

In [ ]:
```
a, *b, c, d= (1, 2, 3, 4, 5, 6)
print(a)
print(b)
print(c)
print(d)
```

In [ ]:
```
a, *b, c, *d= (1, 2, 3, 4, 5, 6) # does not work
print(a)
print(b)
print(c)
print(d)
```

In [ ]:
```
a = 1
b = 2
a, b = b, a     # Swap variables
print(f"a={a}")
print(f"b={b}")
```

Lambda functions

- functions for one-time use
- can be created anywhere using the *lambda*-keyword
- useful f.ex. for sorting or filtering

In [ ]:
```
grade_list = [('Alex', 3), ('Michi', 5), ('Sasha', 1)]
grade_list.sort(key=lambda person: person[0])
print("sorted by name: ", grade_list)
grade_list.sort(key=lambda person: person[1])
print("sorted by grade:", grade_list)
```

In [ ]:
```
list1 = [3, 4, 5, 6, 7]
```

In [ ]:
```
list(filter(lambda x: x > 5, list1))
```

In [ ]:
```
[i for i in list1 if i > 5]
```

In [ ]:
```
f = lambda x: x**x
[f(x) for x in list1]
```

Built-in Functions

https://docs.python.org/3/library/functions.html (https://docs.python.org/3/library/functions.html)

In [ ]:
```
abs(-5)   # absolute value
```

In [ ]:
```
all([True, False, False, True]) # like "and" for all elements of an iterable
```

In [ ]:
```
any([True, False, False, True]) # like "or" for all elements of an iterable
```

In [ ]:
```
list1 = ["a", "b", "c", "d"]
for index, value in enumerate(list1):
    print(f"Value at index {index}: {value}")
```

In [ ]:
```
a = "123.123"
a = float(a) # converts the string to a float
print(a+5)
```

In [ ]:
```
a = "123"
a = int(a) # converts the string to an float
print(a+5)
```

In [ ]:
```
list1 = [1,4,2,6,3]
len(list1)
```

In [ ]:
```
max(list1)
```

In [ ]:
```
min(list1)
```

In [ ]:
```
list1 = ["1","4","2","6","3"]
print(list1)

list2 = map(int, list1) # applies "int()" to every element of "list1"
print(list2)

list3 = list(list2) # to convert the map object back to a list
print(list3)
```

In [ ]:
```
# alternative:
[int(x) for x in list1]
```

and many more...

Student Task

Write a function to solve a quadratic equation

$$x^2 + px + q = 0$$

In [ ]:
```
### Example solution
def solve_quadratic_equation(p, q):
    main_term = -(p/2)
    root_term = ((p/2)**2-q)**0.5
    x_1 = main_term + root_term
    x_2 = main_term - root_term
    return x_1, x_2
```

In [ ]:
```
p = -6
q = 5
x_1, x_2 = solve_quadratic_equation(p, q)
print(f"The solutions for x^2+{p}x+{q}=0 are: {x_1} and {x_2}")
```

Student Task #2

Write a function to calculate the median of a list

In [ ]:
```
numbers = [2,7,3,9,27,8]
more_numbers = [2,7,3,9,27,8,10,0,1]

### Example solution
def median(elements):
    elements.sort()
    print(elements)
    length = len(elements)
    if length % 2 == 0:
        return (elements[length//2-1] + elements[length//2])/2
    else:
        return elements[length//2]
```

In [ ]:
```
median(numbers)
```

In [ ]:
```
median(more_numbers)
```