

Using Functions

Informatics 1 for Biomedical Engineers
Tutor Session 3

KTI, Knowledge Technologies Institute

20. Oktober 2016

Today's Topics

1. What are functions?
2. Defining Python functions
 - Syntax
 - Function arguments - input parameters - keyword arguments
 - Return values
 - Lambda functions
3. Using your own functions

Student Goals

- Be able to define your own functions in Python
- Get a feeling about how much functionality to put into one function
- Use (self defined) Python functions in a small program

What are functions?

- Reuse code snippets
- Quick changing of code throughout the program without copy/paste
- Less typing - less mistakes (potentially)
- Prettier (better readable) and reusable code

Defining Python functions

```
1  # Syntax of Python functions: always the same
2  def function_name(arg1, arg2, ..., argN):
3      # do something
4      return something # remember: this is optional!
```



■ Syntax

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

Defining Python functions

- Note: Python functions are per definition void()-type functions - return value or function type definition not necessary
- Common: return a tuple and assign return values to several variables simultaneously

Defining Python functions

```
1  # Defining our first function
2  def a_simple_function():
3      print("Good_morning,_starshine!")
4
5  # Defining another function
6  def simple_calculus(input_argument):
7      result = input_argument + 365
8      return result
```



Keyword arguments

- Declared when defining a function
- Optional when calling the function - default value is used
- When specified, the new value is passed into the function for the keyword

Keyword arguments

```
1  # Implementing some mathematical function
2  def math_function(number, exponent = 2):
3      result = number ** exponent
4      return result
5
6  # Calling the function
7  math_function(5)
8  math_function(5, 6)
9  math_function(number = 5, exponent = 8)
10
11 # Non-default arguments before keyword arguments!
```



How to work with functions

- Using the return value
- Assigning the result of a function to some variable(s)
- Multiple assignment using Tuples

Using the return value

```
1  # Function returning a tuple
2  def statistic_measures(mean, variance):
3      stand_dev = variance ** 0.5
4      conf_upper = 1.96 * stand_dev
5      return (stand_dev, conf_upper)
6
7  # Assigning the result to several variables at once
8  standard_dev, confidence_up = statistic_measures(3, 1.59)
9  standard_dev = statistic_measures(3, 1.59)[0]
10 confidence_up = statistic_measures(3, 1.59)[1]
11 # "throwing away" a variable:
12 _, confidence_up = statistic_measures(3, 1.59)
```



Anonymous functions

- Lambda expressions
- Created at point of use with lambda keyword
- e.g. as an argument of a function

Lambda expressions

```
1  # Lambda expression as input for the .sort() method
2  # Sorting a list of tuples by name
3  grade_list = [(Alex, 3), (Michi, 5), ('Sasha', 3)]
4  grade_list.sort(key=lambda name: name[0])
5
6  # Sorting the list by grades
7  grade_list.sort(key=lambda name: name[1])
```



Lambda expressions

```
1  # The use of lambda functions
2  # Sorting the list with a separate function:
3  def sort_by_name(some_list):
4      return some_list[0][0]
5
6  grade_list.sort(key=sort_by_name)
7
8  # With lambda, we don't need to declare separate functions
```



Separating functionality

- How many tasks in one function?

```
1  # Example: triangle information
2  def triangle_circumfer_area(param_a, param_b, param_c, height):
3      circumference = param_a + param_b + param_c
4      area = (param_a * height) / 2
5      return circumference, area
6
7  # We can return both with one function, but it
8  # would be better to split this function
9  # for clarity and usability
```



Complex example

Task: function for solving a quadratic equation

- $x^2 + px + q = 0$
- $x_{1,2} = -(p/2) \pm (p^2/4 - q)^{0.5}$
- Input: parameter p and q from the equation
- Return: x_1 and x_2

Complex Task: Sample Solution

```
1  # Sample solution
2  def solve_quadr_equation(param_p, param_q):
3      #x1,2 = -(p/2) +- sqrt((p**2/4)-q)
4      main_term = -(param_p / 2)
5      root_pos = ((param_p ** 2)/4 - param_q) ** 0.5
6      root_neg = root_pos * (-1)
7      x_1 = (main_term + root_pos)
8      x_2 = (main_term + root_neg)
9      return (x_1, x_2)
```



Student Task

Task: Write a function to calculate the median

- Get the list of input data from pickle file
- Write a function to calculate the median of the given data list
 - Bring elements in correct order
 - Get the element(s) in the middle of the list
 - Calculate and return the median

Student Task: Sample Solution

```
1  def calc_median(elem_list):
2      elem_list.sort()
3      length = len(elem_list)
4      half_length = int(length / 2)
5
6      if length % 2 == 0: # Even number of elements
7          first = elem_list[half_length - 1]
8          second = elem_list[half_length]
9          median = (first + second) / 2
10
11     else: # Odd number of elements
12         median = elem_list[half_length - 1]
13
14     return median
```