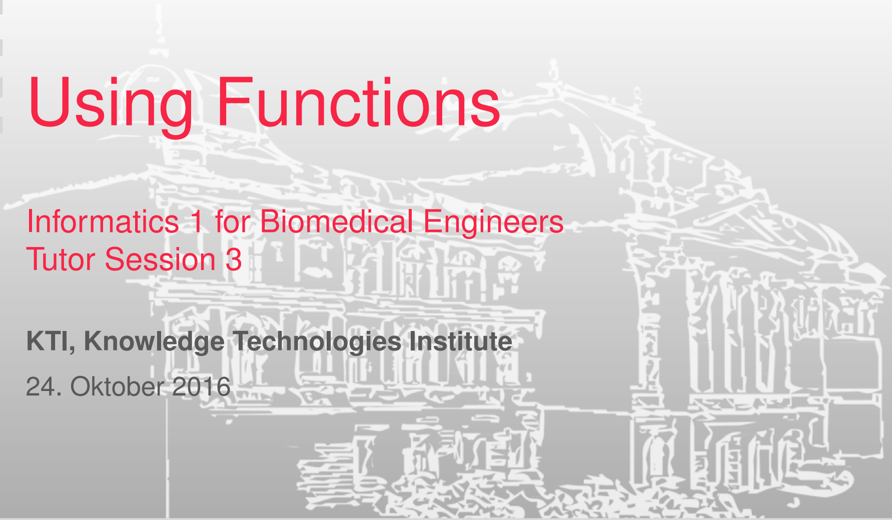

Using Functions

Informatics 1 for Biomedical Engineers
Tutor Session 3

KTI, Knowledge Technologies Institute

24. 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

```python
# Syntax of Python functions: always the same
def function_name(arg1, arg2, ..., argN):
    # do something
    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

```python
# Defining our first function
def a_simple_function():
    print("Good␣morning,␣starshine!")

# Defining another function
def simple_calculus(input_argument):
    result = input_argument + 365
    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

```python
# Implementing some mathematical function
def math_function(number, exponent = 2):
    result = number ** exponent
    return result

# Calling the function
math_function(5)
math_function(5, 6)
math_function(number = 5, exponent = 8)

# 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

```python
# Function returning a tuple
def statistic_measures(mean, variance):
    stand_dev = variance ** 0.5
    conf_upper = 1.96 * stand_dev
    return (stand_dev, conf_upper)

# Assigning the result to several variables at once
standard_dev, confidence_up = statistic_measures(3, 1.59)
standard_dev = statistic_measures(3, 1.59)[0]
confidence_up = statistic_measures(3, 1.59)[1]
# "throwing away" a variable:
_, 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

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

# Sorting the list by grades
grade_list.sort(key=lambda name: name[1])
```

Lambda expressions

```python
def my_filter_fct(obj):
    """
      This function takes an object (which should be a two elemented
      tuple) and returns the second element of the obj
    """
    return obj[1]

grade_list.sort(key=my_filter_fct)

# With lambda, we don't need to declare separate functions
```

Separating functionality

- How many tasks in one function?

```python
# Example: triangle information
def triangle_circumfer_area(param_a, param_b, param_c, height):
    circumference = param_a + param_b + param_c
    area = (param_a * height) / 2
    return circumference, area

# We can return both with one function, but it
# would be better to split this function
# for clarity and usability
```

Complex example

Task: function for solving a quadratic equation

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

Complex Task: Sample Solution

```python
# Sample solution
def solve_quadr_equation(param_p, param_q):
    #x1,2 = -(p/2) +- sqrt((p**2/4)-q)
    main_term = -(param_p / 2)
    root_pos = ((param_p ** 2)/4 - param_q) ** 0.5
    root_neg = root_pos * (-1)
    x_1 = (main_term + root_pos)
    x_2 = (main_term + root_neg)
    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

```
def calc_median(elem_list):
    elem_list.sort()
    length = len(elem_list)
    half_length = int(length / 2)

    if length % 2 == 0: # Even number of elements
        first = elem_list[half_length - 1]
        second = elem_list[half_length]
        median = (first + second) / 2

    else: # Odd number of elements
        median = elem_list[half_length - 1]

    return median
```