“Basic Programming”

Homework Assignment 1

October 17, 2018

1 Tasks

1.1 Mini Tasks (5p - 0.25 per Task)

Download the assignment files[1] to your local system. The ZIP archive chontains two notebooks (German/English) and three files that you need for the mini tasks **Via jupyter notebook.** Open the notebook

with the tasks in your preferred language and fill in the missing cells. Please make sure you do not modify the cells that already contain code. (**Hint:** You will still need to execute those cells to use the code within them!).

Before you submit your work make sure the filename of your notebook is correct (`assigment_1.ipynb`)!

Via native Python. You can also submit the requested functions in a plain python file. Here copy the content of requested cells (that already contain code) into your program. You can find a printcopy of the assignment notebook at the very end of this document.

2 Restrictions

- Do not add any bloat to your submission
- Use built-in functions where possible
- Do NOT load extra packages (no imports)
- Do NOT use any command line arguments!

[1] https://github.com/pkasper/info1-bm/raw/master/2018/assignments/assignment_1_files.zip

3 File Headers

All Python source files have to contain a comment header with the following information:

- Author: – Your name
- MatNr: – Your matriculate number
- Description: – Purpose of the file
- Comments: – Any comments as to why if you deviate from the task or restrictions

Please copy and paste the example and change its contents to your data. (Depending on your PDF-viewer you may have to fix the whitespaces!)

Example Header:

```
######################################################################
# Author:      [FIRST NAME] [LAST NAME]
# MatNr:       [MAT NR]
# Description: [SHORTDESCRIPTION]
# Comments:    [ANY RELEVANT COMMENTS.
#              CAN BE MULTILINE]
######################################################################
```

4 Coding Standard

This lecture follows the official PEP 8 Standard[2]. it defines basic formalities as to how your code should look like. In particular, please look at the following aspects:

Language. Programming is done in English. This is to ensure someone else at the other end of the world can read your code. Please make sure all sources you submit are thus written in English. This covers both variable names and comments!

Spaces, not tabs. Indent your files with 4 spaces instead of tabs. PEP 8 forces this in Python 3 (whilst allowing more freedom in Python 2). Most editors have an option to indent with 4 spaces when you press the tab button!

Descriptive names. Use descriptive names for variables where possible! Whilst a simple i can be enough for a simple single loop code can become very messy really fast. If a variable has no purpose at all, use a single underscore (_) for its name. Should you be uncertain if a name is descriptive enough, simply as a comment describing the variable.

120 **characters line-limit** PEP 8 suggests a maximum line length of 79 characters. A different established limit proposes 120 characters. In this lecture, we thus use the wider limit to allow for more freedom. The maximum number of characters is 120 including indentations! Note that this is also applies to comments!

[2] https://www.python.org/dev/peps/pep-0008/

5 Automated Tests

Your code will be tested by an automated program.Please make sure your submission follows all the restrictions defined in this document. In particular concentrate on the predefined names for functions and variables!
If your submission fails any of the automated tests, we will look into your code to ensure the reason was not on our end and to evaluate which parts of your code are correct and awards points accordingly.

6 Submission

6.1 Deadline

13th of November 2018 at 23:59:59.

Any submission handed in too late will be ignored with the obvious exception of emergencies. In case the submission system is globally unreachable at the deadline it will be pushed back for 24 hours. If for whatever reason you are unable to submit your work, contact your tutor *PRIOR* to the deadline.

6.2 Uploading your submission

Assignment submissions in this course will always be archives. You are allowed to use .zip or .tar.gz formats.
In addition to your Python source files, please also pack a *readme.txt*. The file is mandatory but its contents are optional. In this file please write down how much time you spent on the assignment and where you ran into issues. Your feedback allows for immediate adjustments to the lecture and tutor sessions.
Upload your work to the Palme website. Before you do please double-check the following aspects:

- File names and folder structure (see below)
- Comment headers in every source code file
- Coding standard upheld

6.3 Your submission file

```
■─ assignment_1.zip (or assignment_1.tar.gz)
   ■─ assignment_1.ipynb (or assignment_1.py)
   ■─ readme.txt
```

In [ ]:

```
######################################################################
# Author:      [FIRST NAME] [LAST NAME]
# MatNr:       [MATRICULATE NUMBER]
# Description: [SHORT DESCRIPTION]
# Comments:    [COMMENRS.
#              CAN BE MULTIROW]
######################################################################
```

Assignment 1

Informatics 1 for Biomedical Engineering 2018/2019

Solve these individual tasks. Write the solution for each task as an individual function (into the empty code cells provided below the task descriptions). Pay attention to name the functions exactly as asked!

Examples

With return value and arguments:
Write a function *add(a,b)*, that calculates and returns the sum of two given numbers a and b.

```python
def add(a, b):
    result = a + b
    return result

#for testing:
x = 2
print(add(x, 5)) # -> 7
```

Without return value and arguments:
Write a function *print_hello*, that prints the string "Hello!".

```python
def print_hello():
    print("Hello!")

#for testing:
print_hello() # -> Hello!
```

1 Input/Output

1.1 I/O combined

Write a function *io_1()*. It should use *input* to ask for a user's name and age. Then, it should print out the following sentence (with the entered data):
Hi, my name is *name* and I'm *age* years old.

In [ ]:

```

```

1.2 File I/O

Write a function *io_2()*. The function should read the content of the file "query.txt". Use it as a query for user input and write the entered answer to a file named "result.txt".

In [ ]:

```

```

2 Functions

Create a function *solve_quadratic* to calculate the quadratic formula. The function should accept three arguments which are the coefficients a, b and c and return the results as a tuple. You don't have to worry about complex numbers, python handles them just fine! Test out the function using the following equations:
$x^2 + 2x - 15 = 0$
$2x^2 - 6x - 36 = 0$
(Results should be $(3, -5)$ and $(24, -12)$)

In [ ]:

```

```

3 Datatypes

3.1 Lists

3.1.1

Write a function *lists_1()*. Create a list containing all multiples of 3, starting at 0 and ending at 21. Calculate the square root of each element and save it to a new list. Print the sum of the new list's first and last element.

In [ ]:

```

```

3.1.2

Write a function *task__31_2(given_numbers)*. Create a list containing the squares of the numbers 1 to 10. Merge it with a tuple passed to the function (you can use *given_example* for testing) and remove every number that is divisible by 5. Calculate and print the sum over the remaining numbers.

In [ ]:

```
# do not edit this cell
given_example = (4, 1, 5, 23, 5)
```

In [ ]:

```

```

3.1.3

Write a function *lists_3()*.
Create a list of lists to represent the following tic-tac-toe game (use strings for X and O):
XOO
OXO
XXO
Now, rotate the grid clockwise 4 times by 90° and print the result and an empty line after every step, like this:
XOX
XXO
OOO

OXX
OXO
OOX ...

In [ ]:

```

```

3.2 Strings

3.2.1

Why do the Germans always put commas instead of decimal points? Write a function *strings_1()*. Convert the numbers given in the string below into a list of floats. Now, sum up all numbers and print out the calculation and its result in the following style (limit the number of decimal points for the result to two!):
$1.22 + 3.56 + 3.21 = 7.9$

In [ ]:

```
# do not edit this cell
numbers = "8,78; 2,89; 9,5; 0,71; 4,14; 1,27; 9,7; 1,65; 7,31; 6,21; "
numbers += "6,2; 9,02; 0,49; 1,73; 0,54; 0,9; 7,63; 9,69; 1,81; 8,07"
```

In [ ]:

```

```

3.2.2

Write a function *strings_2()*. For each word in the given string, take only every second letter and reverse the word. Print the resulting sentence!

In [ ]:

```
# do not edit this cell
encrypted = "MeAhGT bkDcxizuDq vnxwNoPrsb ixfoCf ysHpMmsuOj orbeJvgo bejhBt yywzZ
azl A.ugaond"
```

In [ ]:

```

```

3.2.3

A text has been encoded in the string *mystery* and needs your help to be decoded. Write a function *strings_3()* that follows the steps below to modify the text and receive the correct text. Print the result.

1. Every capital letter has been lowercased and surrounded by two words "big"
2. Every letter a has been surrounded by two x's
3. Each comma has been turned into a hashtag
4. Each word ends with 1b and begins with .8
5. Every r has been extended to say rat and t has been turned into a turtle
6. Each space has been replaced by an underscore

In [ ]:

```
# do not edit this cell
mystery = ".8biglbigxaxnd1b_.8derat1b_.8bigbbigeratge1b#.8biglbigxaxnd1b_.8xaxm1
b.8bigsbigturtleratome1b#"
mystery += ".8biglbigxaxnd1b.8derat1b_.8xbigabigxckerat1b#.8biglbigxaxnd1b.8derat
1b_.8bigdbigome1b#""
mystery += ".8biglbigxaxnd1b.8derat1b_.8bighbigxäxmmerat1b#_.8zukunfturtlesrateic
h1b"
```

In [ ]:

```

```

3.2.4

Write a function *strings_4()*. Convert the given string *num* into a list of double digit numbers. Remove the integer 23 from the list and then insert the integer 96. Then sort the list into integers of ascending order and print out the mean, range, min and max (e.g. "The mean is 28").

In [ ]:

```
# do not edit this cell
num = "28561823817394"
```

In [ ]:

```

```

3.3 Dictionaries

3.3.1

Write a function *dicts_1()*. Ask the user to enter name, amount and price per item for 3 different items (use *input*!). Save the entered information into a dictionary (product names as keys, the other two values as a tuple (price, amount); f.ex. {"milk": (1.00, 2), ...}). Use appropriate data types! Add the passed dictionary containing additional products (*old_products*) to the same dictionary. Change the price of cheese to 3.65 (but keep the amount). Print all products together with their prices in the following style:

cheese: 4.65€
flour: 0.49€
...

Calculate what all products in their given amounts would cost and print the result like this:
"The total price is 123.45€"

In [ ]:

```
# do not edit this cell
old_products = {"cheese": (4.65, 1), "flour": (0.49, 2)}
```

In [ ]:

```

```

3.3.2

Write a function *dicts_2()*. Read the file "data.csv" to create a nested dictionary of health profiles of four patients. Each patient has personal as well as medical data. In the first level dictionary, use the names as keys, the values should be dictionaries containing key-value pairs for birth state, gender, heart rate and red blood cell count. Each patient should have the same formatted dictionary. Make sure to use correct data types, ignore units!
The finished dictionary for one person should look like this:
health_profile = {"Max": {"Birth state": "Bayern", "Gender": "Male", "Resting Heart Rate": 74, "Red blood cell count": 4.33}, "Omar": {...}, ...}

In [ ]:

```

```

3.4 Tuples

3.4.1

Write a function *tuples_1()*. Create two tuples, containing the coordinates of two points:

- $P_1 = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$
- $P_2 = \begin{pmatrix} 7 \\ -4 \end{pmatrix}$

Calculate and print the distance between the two points $d = \sqrt{(P_{1_x} - P_{2_x})^2 + (P_{1_y} - P_{2_y})^2}$. Change the x-coordinate of P_1 to 4. Calculate and print the distance again!

In [ ]:

```

```

3.4.2

Write a function *tuples_2()*. Create a tuple containing the numbers from 1 to 10 (inclusive). Use multiple assignment/tuple unpacking to assign the values 1 to 3 to the variables a to c (integers), the remaining numbers to d (a list). Return all 4 variables

In [ ]:

```

```

3.5 Sets

Write a function *sets_1()*. Create three sets:

- even_numbers: Containing all even numbers starting at 2, up to 100 (inclusive)
- multiples_of_3: Containing all multiples of 3 up to 100
- multiples_of_7: Containing all multiples of 7 up to 100

Now, print all numbers (in this range),

- which are divisible by both 2 and 7
- which are divisible by 7, but odd

Add up the smallest and largest numbers which are either divisible by 7 and 3 (but not both!) and print the result!

```
In [ ]:
```

4 Control Flow Statements

4.1 If/Else

4.1.1

Write a function *ifelse_1()*. Iterate over the numbers from 1 to 100 (both inclusive). For numbers divisible by 5, print out "Fizz". For numbers whose digit sum is 7, print out "Buzz". If both conditions apply, print "FizzBuzz". When both do not apply, just print the number.

```
In [ ]:
```

4.1.2

Write a function *ifelse_2()* that tells the user their grade depending on points entered (using *input*). It should work like this:

Enter your points: 77
You got a C!

Calculate the grades according to the following table:

Grade	Range
A	90 <= P
B	80 <= P < 90
C	70 <= P < 80
D	60 <= P < 70
E	50 <= P < 60
F	P < 50

```
In [ ]:
```

4.2 For Loops

4.2.1

Write a function *for_loops_1()*. Create an empty list. In a for loop generate a list of 1000 random integers from 0 - 100 and with each iteration append the new integer to this list. Return a list containing only those numbers that are even.

In [ ]:

```

```

4.2.2

Write a function *for_loops_2()*. Open the file "for_loops.txt". Read line for line and display each line's content in the following style

Line No. 1: "Whatever line 1 says"
Line No. 2: "Whatever line 2 says"
...

Stop printing after the first line containing the word "end".

In [ ]:

```

```

4.3 While Loops

Write a function *while_loops_1()*. Create an empty list. Now, using *input*, ask the user to enter a number. Check if what was entered is actually an integer, otherwise print a warning message. If it is, append it to the list (use the appropriate data type!). Repeat this process until the string "stop" is entered. Then, print out all numbers in the list in reversed order.

In [ ]:

```

```