Programming Lab II
Handout 1

Dr. Dilyana Dimova, Dimitar Yonchev, Dr. Martin Vogt
martin.vogt@bit.uni-bonn.de

April 10, 2018

Deadline: April 19, 2018

Next handout: April 17, 2018

Course notes & requirements

- You will receive handouts containing assignments on a weekly or biweekly basis that are to be completed within one or two weeks.
- Assignments have to be completed in the allotted time. For each handout, there will be a deadline given by which you will have to hand in your code by email.
- Assignments have to be handed in **individually** to `martin.vogt@bit.uni-bonn.de`
- Use the following naming conventions for handing in your assignments: `'plab-{xx}-{surname}.pynb'` if you hand in a single notebook file or `plab-{xx}-{surname}.zip` if you zip multiple files as an archive. Here `{xx}` is the handout number and `{surname}` is your surname.
- Code needs to be documented or be accompanied by explanations.
- The assignments will be evaluated and points will be awarded by individually discussing your code with a tutor in a following class.(You will need to be present for this.)
- Although you need to hand in your assignments individually and they will be evaluated individually; you can still work together in small groups of at most 3 students. However, you will need to indicate this in your submission either as comment in the code or as markup text in your notebook file.
- In order to receive credits for this course you
  - will need to get 50% of the points
  - must not fail to hand in an assignment more than once.
- Materials will be made available using the sciebo cloud service. The link `https://uni-bonn.sciebo.de/s/meq780szilhzvH6` will give you access to the folder `programming-lab-ss18` where all materials will be deposited.
- A self-contained Python installation `pywin32.zip` can be found there as well as the basic introductory open-source book *Think Python*. (Available from `http://www.greenteapress.com/wp/think-python-2e/`)
- You can use any other Python 3 installation you are familiar with. For the desktops in the computer pool, I suggest you use the provided zip-package or the Anaconda distribution (`https://www.anaconda.com/download/`), which you can install on your home directory and allows you to add Python packages if required.

Getting started

Ex. -1 *Installing Python from the zip-file*
In `programming-lab-ss18` you find the zipped folder `pywin32` containing a full Python distributions for Windows. Copy and unzip it in your local home. The distribution folder contains two important links/scripts:

(a) `_python_session_` If you run this shortcut a new command line terminal is opened from which you can start the Python interpreter from the *command line*. Using a terminal like the command line is the traditional way of working interactively with Python. This is not a very comfortable way — especially on Windows. An interactive session usually consists of starting the Python interpreter by executing the `python` program. A special prompt appears (`>>>`) and you type stuff for the python interpreter to execute, hit return, and get a result on the screen on the next line, rinse, repeat... This is known as a Read-Eval-Print Loop (REPL) and is a common way for interacting with interpreted languages.

(b) `_python_notebook_` Because the standard REPL of Python is so useful but not very comfortable, the *IPython/Jupyter* project has made it its goal to provide a more comfortable way for interacting with Python. IPython provides several frontends for the user, notably a terminal-based, a GUI-based, and a browser based one. The browser-based frontend has developed into an independent web application known as Jupyter supporting many other programming languages. The supplied distribution contains a shortcut to start Python in this mode. It is suggested that you use Python notebooks for your interactive sessions. It will keep track of your session in notebook files that can be commented, edited, and extended.

Starting up the notebook will take a few seconds. It is set up to save notebooks in a folder called `_my_notebooks_`.

The (free) Python distribution is called *anaconda* and provided by a third party `https://www.anaconda.com/`. The advantage of using this version over the locally installed versions of Python on each of the computers in the PC-pools is that you can easily install additional packages for the Python programming language like `biopython` or `nltk` using the `conda` package manager from the command line.

On your own PC/notebook you can also get a full version of the Anaconda Python environment from `http://anaconda.com/download/` already containing most of the available packages.

Ex. 0 If you do not know already, get familiar with the usage of the jupyter notebook. E.g.

- Consult some online resources like `http://ipython.readthedocs.io/en/stable/interactive/index.html` or `http://bebi103.caltech.edu/2015/tutorials/t0b_intro_to_jupyter_notebooks.html`
- Try to execute some python code or run and edit some of your code from last semester.

Note that you can organize notebooks using headings and markdown text for documentation purposes.

Code you hand in needs to be documented in on form or the other. You can hand in documented Python programs or notebooks containing your code and documentation.

Programming exercises

Ex. 1 *(5 pts) Srinivasa Ramanujan calculates π*
The mathematician Srinivasa Ramanujan found an infinite series that can be used to generate a numerical approximation of π:

$$\frac{1}{\pi} = \frac{2\sqrt{2}}{9801} \sum_{k=0}^{\infty} \frac{(4k)!(1103 + 26390k)}{(k!)^4 396^{4k}}$$

Write a function called `estimate_pi` that uses this formula to compute and return an estimate of π. It should use a `while` loop to compute terms of the summation until the last term is smaller than `1e-15` (which is Python notation for 10^{-15}). You can check the result by comparing it to `math.pi`.

Ex. 2 *(5+3 pts) Happy numbers*
Happy numbers are defined by the following process: Start with a positive number. Replace the number with the sum of the squares of its digits and repeat until you reach the number 1 or the process enters a loop not involving the number 1. A number that reaches 1 is called a happy number all other numbers are unhappy.

(a) *(3 pts)* Write a function `isHappy(n)` that checks whether a number is happy or unhappy. It should return `true` if the the number is happy and `false` otherwise.
Hint: It is known that, if a number is unhappy, it will enter a loop involving the number 4. Thus you can repeat the process until the number reaches either 1 (happy) or 4 (unhappy).
Hint: You will somehow need to extract the individual digits of a number. Section 8.7 (pg. 75) of Think Python explains how to iterate over the individual characters of a string. A number can be represented as a string using the `str` function and vice versa a string consisting of digits can be converted back to an integer using the `int` function.

(b) *(1 pt)* Find all happy numbers from 1 to 100.

(c) *(1 pts)* Solve the problem in two different ways using a) `while`-loops and b) recursion.

(d) *Bonus: (3 pts)* Modify the problem by taking the sum of cubes instead of the sum of squares. Find all the loops that can occur and all numbers that are equal to the sum of the cubes of their digits. A number is called cube-happy if its iteration ends in a number that is identical to the sum of the cubes of its digits. Find all cube-happy numbers from 1 to 1000.

Ex. 3 *(5 pts) The Birthday Paradox*

(a) *(1 pt)* Write a function called `has_duplicates` that takes a list and returns `True` if there is any element that appears more than once. It should not modify the original list.

(b) *(4 pts)* If there are $n = 27$ students in your class, what are the chances that two of you have the same birthday (day/month)? You can estimate this probability by generating random samples of n birthdays and checking for matches. (For simplicity, assume that every year has 365 days and the probability to be born on any day is the same.) *Hint:* You can generate random birthdays with the `randint` function in the `random` module.

i. *(2 pts)* Estimate the probability on the basis generating 10000 trials of $n = 27$ birthdays and determine the fraction of trials, where at least two persons share a birthday.

ii. *(2 pts)* How do your estimates compare to the approximated probability $p_m(n) \approx 1 - e^{-\frac{n^2}{2m}}$ for $m = 365$,$n = 27$ and the exact probability

$$1 - p_m(n) = 1 \cdot \frac{m-1}{m} \cdot \frac{m-2}{m} \cdot \ldots \cdot \frac{m-n+1}{m}$$

.

iii. *Bonus: (2 pts)* Modify your approach to estimate the probability that at least three people share a birthday with another one.

You can read about this problem at `http://en.wikipedia.org/wiki/Birthday_paradox`.

Ex. 4 *(5 pts) Anagrams*

(a) *(2 pts)* Write a program that reads a word list from the file `words.txt` and prints all the sets of words that are anagrams. Limit your output to words having at least 6 anagrams (including itself).

Here is an example of what the output might look like:

```
['deltas', 'desalt', 'lasted', 'salted', 'slated', 'staled']
['retainers', 'ternaries']
['generating', 'greatening']
['resmelts', 'smelters', 'termless']
```

(b) *(2 pts)* Modify the previous program so that it prints the largest set of anagrams first, followed by the second largest set, and so on.

(c) *(1 pt)* Which set of 8 letters contains the most anagrams and what are they? Hint: there are seven.