Programming Lab I
Handout 1

Dr. Martin Vogt
martin.vogt@bit.uni-bonn.de

April 21, 2020

Deadline: May 3, 2020 **Next handout: April 28, 2020**

Course notes & requirements

- Online classes will be **Tuesdays from 9-10:30** and **Thursdays from 13:30-15:00** via Zoom meetings.
- During the online session you can work on the code, ask questions and discuss code. It will not take the form of a lecture, although sometimes solutions of general interest might be discussed.
- You will receive handouts containing assignments on a weekly or biweekly (i.e., once per fortnight) basis that are to be completed within one or two weeks.
- Assignments have to be completed and **sent in by email by the deadline given**.
- Assignments have to be handed in **individually** to `martin.vogt@bit.uni-bonn.de`
- For the assignments use the following naming conventions: `{surname}-plab-{x}.ipynb` if you hand in a single notebook file or `{surname}-plab-{x}.zip` if you zip multiple files as an archive. Here `{x}` is the handout number and `{surname}` is your surname (without the curly braces).
- **Code should be documented or be accompanied by explanations. This can be done really nicely using Python notebooks and the integrated markup language.**
- The assignments will be evaluated and points will be awarded based on the code and after individual discussions of the code to assess your understanding.
- These discussions will be carried out online in separate Skype or Zoom meetings either during online class or by appointment once you handed in the code.
- Although you need to hand in your assignments individually and they will be evaluated separately, you can still work together in small groups of at most 3 students. However, you will need to indicate who you worked with in your submission, for instance in a comment of the code or as markup text in your notebook file.
- In order to receive credits for this course you
  - must not fail to hand in an assignment more than once
  - obtain 50% of the points for all handouts combined.
- Planned are 9 handouts in total with 20-30 points each and 200 points overall.

- The course will graded. Upon successful completion you will be awarded 8 CP. The following grading scheme applies:

Points	Grade
$100 - 108.5$	4.0
$109 - 117.5$	3.7
$118 - 126.5$	3.3
$127 - 135.5$	3.0
$136 - 144.5$	2.7
$145 - 153.5$	2.3
$154 - 162.5$	2.0
$163 - 171.5$	1.7
$172 - 180.5$	1.3
$181 - 200$	1.0

- Materials will be made available using the sciebo cloud service. The link `https://uni-bonn.sciebo.de/s/wukB3cTMMeCDYG3` will give you access to the folder `programming-lab-ss20` where all materials will be deposited.

Programming exercises

Ex. 1 *(3 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}). ("Term"refers to the expression after the summation sign, not the summation itself.) You can check the result by comparing it to `math.pi`. The error should be less than 10^{-15}.

Ex. 2 *(5 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) *(1 pt)* 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. E.g.:

$$97 \;\rightarrow\; 9^2 + 7^2 = 130 \;\rightarrow\; 1^2 + 3^2 + 0^2 = 10 \;\rightarrow\; 1^2 + 0^2 = 1$$

Hint: It is known that, if a number is unhappy, it will enter a specific loop

$$4 \;\rightarrow\; 16 \;\rightarrow\; 37 \;\rightarrow\; 58 \;\rightarrow\; 89 \;\rightarrow\; 145 \;\rightarrow\; 42 \;\rightarrow\; 20 \;\rightarrow\; 4$$

Thus you can repeat the process until the number reaches either 1 (happy) or one of the values of the loop, for instance 4 (unhappy).

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

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

(d) Let us modify the iteration by taking the sum of cubes of the digits instead. In this case three possible outcomes can be distinguished:

- Happy numbers: The iteration end in 1.
- Almost happy but still unhappy numbers: The iteration ends in a number > 1 that is equal the sum of the cubes of its digits (e.g., $153 = 1^3 + 5^3 + 3^3$)
- Really unhappy numbers: The iteration ends in a loop involving two or more numbers.

Tasks:

i. (*1 pt)* Write a function `determineHappiness(n)` that determines whether a number is happy, almost happy, or unhappy. It could return the strings `"happy"`, `"almost happy"`, `"unhappy"` depending on the number.

ii. (*1 pt)* Use the function to determine all happy numbers from 1 to 1000. How many numbers in the range 1 to 1000 are almost happy?

.

Ex. 3 *(4 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) 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. *(1 pts)* Estimate the probability on the basis generating 10 000 trials of $n = 27$ birthdays and determine the fraction of trials, where at least two persons share a birthday.

ii. *(1 pt)* 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

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

.

iii. *(1 pt)* Modify your approach to estimate the probability that at least three people have a non-unique birthday in their class. *Hint:* Read carefully! In a class with two pairs of people where each pair shares a single birthday there are also "at least three" people having a non-unique birthday.

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

Ex. 4 *(2 pts) Making triangles*

Imagine you are given a stick of length 1 (meter) and you break the stick randomly at two points breaking leaving you with 3 smaller sticks of random length. How likely is it that the three sticks can be combined to form 3 sides of a triangle? (See exercise 5.3 of Think Python.)

Write Python code to simulate 1 000 000 trials splitting the stick randomly into 3 pieces and estimate the probability that a triangle can be formed from the three pieces.

Ex. 5 *(2 pts) Estimating π*

Imagine a square of side length 2 centered around the origin, i.e. with x-coordinates in the range -1 to 1 and y-coordinates in the range -1 to 1:

$$S = \{(x, y) | -1 \leq x \leq 1 \wedge -1 \leq y \leq 1\}$$

Now consider a circle with radius 1 around the origin:

$$C = \{(x, y)|x^2 + y^2 \leq 1\}$$

The area of S is 4 and the area of the circle is π.

If we randomly choose points inside the square by repeatedly choosing random values x and y in the interval -1 to 1 some of these points will also fall inside the circle and some will fall outside the circle. The probability of falling inside the circle is determined by the ratio of the areas of C and S, i.e. $\pi/4$. Thus, if we repeatedly choose random points, the fraction of points inside the circle should approximate $\pi/4$.

Choose 10, 100, 1000, 10 000, 100 000, 1 000 000 random points and determine an approximation of π by determining $4 \cdot f$ where f is the fraction of points falling inside the circle. How does the approximation of π improve.

Note: This technique is known as *Monte-Carlo-integration*. In general Monte-Carlo techniques and can be used in many scenarios to estimate quantities that are hard to determine otherwise.

Ex. 6 *(4 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) *(1 pt)* 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:* the solution has seven anagrams.