BERGISCHE UNIVERSITÄT WUPPERTAL

University of Wuppertal
School of Mathematics and Natural Sciences
Dr. Holger Arndt

Tools

Summer Term 2022

Exercise Sheet 1

Exercise 1 (Python 3, Collatz Conjecture, 4 p.)

Implement a Python function `collatz(n)` that does the following. It takes a given integer n and if n is even, it divides n by 2. Otherwise, it multiplies n by 3 and adds one to it. Then, this number is added to a list l. The new number is called n and this procedure is repeated until n is equal to 1. At the end the function returns the list l.

(The question whether this sequence will reach 1 for every starting value is an unsolved problem in mathematics. Test e. g. with 2223.)

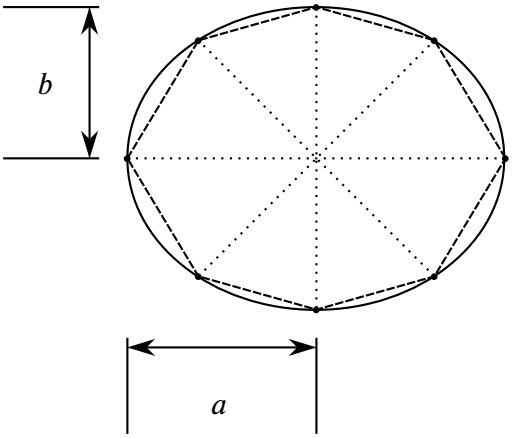
Exercise 2 (Python 3, Circumference of an Ellipse, 5 p.)

There is no closed formula for the circumference of an ellipse. Write a Python script that approximates the circumference. The ellipse is to be approximated by an increasing number of straight lines.

An ellipse is given by the parametric curve

$$f : [0; 2\pi] \to \mathbb{R}^2, t \mapsto (a \cos t, b \sin t)$$

where a and b are the half axes.



- Start with a function that determines the distance between two points (`x1`, `y1`) and (`x2`, `y2`):

  ```
  length(x1, y1, x2, y2)
  ```

- Add a function that computes points on the boundary of an ellipse. One way of returning two coordinates is returning a tuple.

  ```
  ellipse(a, b, t)
  ```

- Determine the circumference for an ellipse that is approximated by `n` straight line edges—split the interval $[0; 2\pi]$ into `n` parts of equal size.

  ```
  circumference(a, b, n)
  ```

- In the main program, read in the half axes a and b and the desired relative accuracy ε. Compute and print the circumference for $n = 2, 4, 8, 16, \ldots$ until the relative accuracy is reached (i. e. $\left|\frac{c_n - c_{\frac{n}{2}}}{c_n}\right| \leq \varepsilon$) or $n = 2^{24}$.

Exercise 3 (Python 3, Game of Life, 5 p.)

Implement Conway's Game of Life in Python.

The following rules are cited from https://en.wikipedia.org/wiki/Conway's_Game_of_Life where you will find more background information:

The universe of the Game of Life is an infinite, two-dimensional orthogonal grid of square cells, each of which is in one of two possible states, alive or dead, (or populated and unpopulated, respectively). Every cell interacts with its eight neighbours, which are the cells that are horizontally, vertically, or diagonally adjacent. At each step in time, the following transitions occur:

1. *Any live cell with fewer than two live neighbours dies, as if by underpopulation.*
2. *Any live cell with two or three live neighbours lives on to the next generation.*
3. *Any live cell with more than three live neighbours dies, as if by overpopulation.*
4. *Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.*

The initial pattern constitutes the seed of the system. The first generation is created by applying the above rules simultaneously to every cell in the seed; births and deaths occur simultaneously, and the discrete moment at which this happens is sometimes called a tick. Each generation is a pure function of the preceding one. The rules continue to be applied repeatedly to create further generations.

As there are no infinite arrays, we use a toroidal array: an array of a fixed size $x \times y$ where the right and left edge are stitched together and also the top and bottom edge. To get the coordinates of neighbouring cells in your program , you can simply use "index modulo size" (e. g. `(i + 1) % x`).

The size of the array should be passed in two command line parameters. Use `'X'` for live cells and `'O'` for dead cells. Start with a random pattern where the probability for a live cell is 20 % (`random.random()` returns random numbers between 0 and 1). (Feel free to provide additional fixed first generations.)

Perform 50 iterations as follows:

a) Print the current status. (Instead of `'X'` and `'O'` you could also print ANSI escape sequences like `'\033[0;41m  \033[0m'` and `'\033[0;47m  \033[0m'`.

b) Wait one second (`time.sleep(1)`).

c) Use a function `nextgen` to compute the next generation.

(Hint: If you need a copy of nested lists, use e. g. `new = copy.deepcopy(old)`.)

Exercise 4 (Python 3, Dictionary, 4 p.)

Implement a dictionary program in Python. The program should allow the user to look up words, list the whole dictionary, add new words, and delete words from it.

On the IT cluster in the directory `/home/tools/Exercises/04` (also available in Moodle) you can find additional material for this exercise. It contains a python script that is able to read and store a word list from and to a file. You may use it as a basis for your program. Furthermore, a word list is provided.

Hint: Use the Python `dict` type to manage the word list.

Note: Make sure that the program does not crash when the user looks up a word that is not in the word list.

A typical interaction of the user with the program could look like this:

```
Look up (1), List(2), New Entry (3), Delete Entry (4), Exit (0)?
--> 1
What is the word?
--> banana
Word not found!
Look up (1), List(2), New Entry (3), Delete Entry (4), Exit (0)?
--> 3
What is the word?
--> banana
What is its translation?
--> Banane
Look up (1), List(2), New Entry (3), Delete Entry (4), Exit (0)?
--> 1
What is the word?
--> banana
banana -> Banane
Look up (1), List(2), New Entry (3), Delete Entry (4), Exit (0)?
--> 0
```

Hand in by: Mon., 25.04.2022 until 14:00 by email to `tools@studs.math.uni-wuppertal.de`.