Programming for Data Science
DS2002 course
03.11.2025

Lab 0

Summary: This lab is designed to help you practice (and thus understand more in-depth) the basic programming concepts such as literals and variables, operations, sequences, functions, conditionals and loops.

This lab is based on "concise presentation of Python" by Veronica Gaspes. It is strongly recommended that you read it beforehand. If you are lost at any point, just look up the key concepts: https://veronica-gaspes.github.io/Concise-Python/intro.html

In this lab, you do not need to solve all the tasks; however, you should do as much as possible. If you struggle with the first three, you lack the fundamentals and need to work hard to catch up. Most students should be able to manage Tasks 1-5, while Task 6 is more ambitious.

Task 1: Factorial

Goal: write a function that calculates the factorial of a number provided as an argument https://en.wikipedia.org/wiki/Factorial

Overall, this is very similar to "Our first function" from the "Defining functions" chapter. Start by creating a function, called "factorial," which takes one argument.

In fact, you can solve this task in an almost identical manner. However, while `sum` is one of the functions built into Python, and thus always available, the equivalent for multiplication is a bit more tricky. You first need to import the math module using `import math` statement. Then, you'll be able to use `math.prod` function, together with list comprehension – almost identical to the `first_odds_sum` function.

In addition, however, you should also solve this task in a more elementary, "step-by-step" way – using a look instead of the `math.prod` function. To do so, create another function, this time called "factorial_with_loop".

First, create a local variable (inside this function) to keep track of the intermediate result, as we are counting towards the final number. It should start with a value of `1`. Then, use a loop (either a `for` or a `while`, either will work) to iterate over all the numbers – up to the one given as an argument – and multiply your intermediate result with the current number. Once done, return the final result.

Make sure (by writing some tests) that both functions give the same result.

Task 2: Prime Numbers

Goal: Write a function to check whether a number provided as an argument is a prime number https://en.wikipedia.org/wiki/Prime_number

You should, again, create a function that takes a number as an argument, called `nr`. Inside that function, write a loop, iterating over all the numbers that are smaller than `nr`. In each iteration, check whether your current number is a divisor of `nr`.

Hint: use the modulo operator or `is_integer()` method on a result of division: https://www.freecodecamp.org/news/the-python-modulo-operator-what-does-the-symbol-mean-in-python-solved/
https://docs.python.org/2/library/stdtypes.html#float.is_integer

Task 3: Approximating the Square Root

Goal: Implement Newton's method for approximating the square root of a natural number `n`. Check the following link to get a brief introduction to the algorithm: https://python.pages.doc.ic.ac.uk/2022/lessons/core04/04-applied/06-newton.html

As discussed in the link above, the starting point is to make an initial guess `x`. While the quality of this guess affects how quickly the method converges, the algorithm works for any initial guess (keep in mind that zero is not valid).

Following that, you should execute the main update rule of the algorithm until you reach the required precision. The update rule is:

$$x_{i+1} = \frac{1}{2} * \left(\frac{n}{x_i} + x_i\right)$$

The desired precision is 10^{-10} (in other words, your solution `x` must be such that `x*x` is no further than 10^{-10} from `n`).

Task 4: Khayyam-Pascal Triangle

Goal: The Khayyam-Pascal triangle is a "pretty" pattern of numbers; your task is to generate such a triangle of a given size (number of rows). Check out this link to get a better understanding of the triangle: https://en.wikipedia.org/wiki/Pascal%27s_triangle

The easiest implementations of the Khayyam-Pascal triangle use the implementation of the factorial, so feel free to use your code from Task 1. Build the Khayyam-Pascal triangle using your factorial to calculate the *n*th row and the *k*th column using the "Formula" from the link above, using the binomial coefficient:

$$Combination(n, k) = \frac{n!}{k!\,(n-k)!}, \qquad n > k$$

You do not need to worry about aligning the numbers nicely in print, just about the values.

Task 5: Find Common Divisors

Goal: Implement a function listing the shared divisors of a given pair of natural numbers.

The task is somewhat similar to the prime numbers example. You should iterate over all the relevant numbers (think about what the boundary is!) and check which of these numbers are divisors of both your arguments.

Your function should return a list with all the common divisors.

Task 6: Babbage’s Difference Engine

Goal: Calculate the next value of a polynomial, imitating Babbage’s Difference Engine.
https://en.wikipedia.org/wiki/Difference_engine

Charles Babbage came up with the design of a mechanical computing machine, known as Babbage’s Difference Engine, capable of calculating the following values of polynomials. The evaluation of the next value of a polynomial is done according to the constant differences. Check out the link below to get a better understanding of Babbage’s Difference Engine.
https://twobithistory.org/2018/08/18/ada-lovelace-note-g.html

You get started with the calculation of differences between the elements of a given list. Keep in mind that the list of the differences must have the length of the original list minus one. The process of the calculation of the difference lists must continue until the point when the differences are constant. Once the differences are constant, using the Babbage algorithm, you are able to compute the next value of the original list.