Laboratory Exercise 03

Topics

1. `while` and `for` loops
2. Elaboration of complex inputs
3. Simulations

Discussions

A. Discuss similarities and differences among the two following groups of instructions:
   a. `if`, `elif`, and `else`.
   b. `while` and `for`.
B. What is an infinite loop?
C. What is a simulation?

Exercises

Part 1 – Basic loops

***Delivery**: For each of the following exercises, write a program in Python that responds to the requests indicated. Complete <u>at least two</u> exercises during the lab session, and the rest at home.*

3.1.1 Integer numbers. Write a program reading a sequence of integer numbers (an empty string is the end of the sequence) and, **after each input**, executing and visualizing:

I. Partial sums of every number acquired; the program must visualize the result of the calculations after each input.
   As an example, if the input values are `1 7 2 9`, the program shall visualize the partial sum of the numbers acquired after each input:
   a. After the first input (`1`), the first acquired value: `1`.
   b. After the second input (`7`), the sum between the first and the second acquired values: `1 + 7 = 8`.
   c. After the third input (`2`), the sum between the first, the second, and the third acquired values: `1 + 7 + 2 = 10`.
   d. After the fourth input (`9`), the sum between the first, the second, the third, and the fourth values acquired: `1 + 7 + 2 + 9 = 19`.
II. The minimum and the maximum values among the acquired ones.
III. How many acquired values are even, how many acquired values are odd. [P4.2]

3.1.2 String analysis. Write a program that takes as an input a line of text as a string and outputs the following requests:

I. Only the capital letters in the string.
II. The letters in the string in even positions.
III. The same string with vowels (uppercase and lowercase) replaced by an underscore (_). IV. How many digits are contained in the string.
IV. The positions of all vowels in the string.

Bonus: Try doing this exercise using an f-string to store all the results and use print() only once. [P4.3]

3.1.3 Shapes. Write a program that takes as an input an integer number `n` and prints a square and a rhombus filled with asterisks (`*`), with each side long `n` asterisks. Example: using `n=4`, the program shows:

```
****
****
****
****

   *
  ***
 *****
*******
 *****
  ***
   *
```

[P4.22]

3.1.4 Words in reverse. Write a program that reads a word and outputs:

I. The reversed word. If the user writes the string `'Hello'`, the program shall output `'olleH'` [P4.9]
II. The uppercase letters starting from the end. If the user writes the string `'HeLlO'`, the program shall output `'OLH'`.

3.1.5 Prime numbers. Write a program that asks the user for an integer number and shows as an output a message showing whether the input number is prime.

3.1.6 Prime numbers. Write a program that asks the user for an integer number and shows all the prime numbers **lower or equal** to that number. Example: if the user inputs `20`, the program shall output:

```
2
3
5
7
11
13
17
19
```

[P4.17]

3.1.7 Words and spaces. Write a program reading a word and showing all its substring, sorted by increasing length. If the user inputs the string `'rum'`, the program shall output:

```
r
u
m
ru
um
rum
```

[P4.12]

3.1.8 Duplicate numbers. Write a program reading a sequence of integer numbers (the sequence ends with an empty line) and, after each acquisition, compute and shows only the adjacent duplicate numbers.

Example: if the input sequence at step IV are the values `1 3 3 4 5 5 6 6 6 3`, the program shall output the adjacent numbers repeated at least two times; at the end of the sequence of equal numbers (after acquiring the first different number), it shall print:

I. Nothing at the first input (`1`), as it is the first value.
II. Nothing at the second input (`3`), as the adjacent values `1` and `3` are not duplicates (equal).
III. Nothing after the third input (`3`), as the acquired value is the duplicate of the previous one, and the sequence of duplicate numbers may not be terminated yet.
IV. The value `3` after the fourth input (`4`), as the two latest adjacent values (`3` and `4`) are not duplicated anymore; thus, the sequence of duplicate adjacent numbers is terminated, and the program shall output the duplicate number.

According to this logic, the sequence `1 3 3 4 5 5 6 6 6 3` produces, in the end, the following values: `3 5 6`. [P4.2]

Part 2 – Application of loops

***Delivery**: For each of the following exercises, write a program in Python that responds to the requests indicated. Complete <u>at least one</u> exercise during the lab session, and the rest at home.*

3.2.1 The game of Nim. In this game, two players alternate extracting marbles from a pile. At each round, the number of marbles taken is up to the player, providing it is at least one, and at most half of the number of marbles currently available. The player that takes the last marble loses.

Write a program allowing a user to play against the computer. The program shall:

I. Generate a random number between `10` and `100` to use as the quantity of marbles in the pile in the beginning of the game.
II. Generate a random number, either `0` or `1`, to decide whether the first move is on the player or on the computer.
III. Generate a random number, either `0` or `1`, to decide if the computer must play in a smart or dumb way:

a. By playing dumb, on every move the computer takes a random number of marbles from the sack (between 1 and n/2, given n as the current number of marbles left in the pile).
b. By playing smart, it takes a number of marbles so that the number of the remaining ones is a power of 2 diminished by 1 unit, meaning: 3, 7, 15, 31 o 63. This move is always valid, except when the number of marbles is equal to a power of 2 diminished by 1. If such is the case, it randomly plays a valid move.

You can experimentally verify that the computer cannot be beat in smart mode, unless the initial pile contains 15, 31 or 63 marbles. The same goes for human players: if the player draws first and knows this strategy, the computer cannot win. [P4.23]

3.2.2 Image diagnostics. The radioactive decay of radioactive material can be modelled through the following equation: $A = A_0 e^{-\lambda t}$, where A is the quantity of the material at time t, A_0 is the quantity of the material at time 0, and λ is the decay rate. More precisely $\lambda = \ln(2)/T_{1/2}$, where $T_{1/2}$ is the half-life of the substance (expressed in the same unit of measure of t).

Technetium-99 is a radioisotope used for diagnostics of brain images. It has a half-life of 6 hours. Write a program showing the relative quantity A/A_0 in a patient body for each hour during the 24 hours next to the administration of the dose. [P4.39]

3.2.3 Trajectories. A cannonball is launched through the air with starting velocity v_0. Physics books say that the position of the cannonball after t seconds is $s(t) = -1/2gt^2 + v_0 t$, where $g = 9.81\ m/s^2$ is the gravitational acceleration of Earth. However, no book mentions that this is a dangerous experiment, so we may not want to perform it; however, we're going to do that anyways, thanks to our computers, assessing theory through a simulation. In our simulation, we are going to consider very small time intervals Δt. In a small time interval, velocity v is almost constant, and we can consider the distance travelled as $\Delta s = v\Delta t$. Write a program that, using the constant `DELTA_T = 0.01` (seconds), updates the position of the cannonball using the following formula: `s = s + v * DELTA_T`.

Velocity is continuously changing, as the Earth gravitational force decreases it. In a small time interval, $\Delta v = -g\Delta t$. Thus, we can update velocity using the following formula:

`v = v - g * DELTA_T`.

In each iteration, we must use the new value for the velocity to update the computation of the travelled distance. The program shall continue the execution of the simulation up until the cannonball falls to the ground. The program takes the initial velocity as an input (example: 100 m/s). It updates the velocity 100 times per each simulated second, but shows the current velocity only once per simulated second, together with the value computed through the exact formula $s(t) = -1/2gt^2 + v_0 t$, for comparison.

Note: What is the advantage of simulating a phenomenon when we already have the exact formula? The point is that the formula is not actually exact. Gravity acceleration is different at different

altitudes or distances from Earth. This is not represented in the formula, while our simulation might be extended to include this observation. Thus, simulation is necessary for trajectories of objects flying higher than usual, such as missiles. [P4.37]