Practical Work

For every exercises you can use the slides and the cheat sheet. Don't hesitate to ask any question about the exercises or computer science in general!

Exercise 1:

The goal of this exercise is to play with the fundamentals.

1) Write a program which ask the user a name and print "Hello <name>", where name is the name from the user input.
2) Now ask the age of the person and store both name and age in two variables. Then print "Hello <name>! You are <age> years old."
3) Add the following test :
    - if the age of the person is over 18 years old, print "You are of age!"
    - otherwise print "You are a minor".
4) Now we want the user to be able to add multiple names and ages :
    - We start by asking the user to enter a name and an age;
    - Then we ask the user if he want to enter another name;
    - If the answer is yes, then we go back to the first step;
    - If it's no, we print all the names and ages and we terminate the program.
    - *hint : You should use a while loop for the part dealing with the user's input and a for loop to go through the names and ages.*

Exercise 2:

In this exercise we will improve our mental calculation game.

1) You will need three variables, two for the random values, and a third random value (between 0 and 3) to choose the operator (among +,-,* and /).
2) Use three 'if' and one 'else' to choose between the operators, using the third value you generated as the condition.
3) In the case of the division, don't divide by 0!
4) After this, print the calculation, and ask the answer to the user. Then compare the answer with the good result (don't forget to cast the value you get from input() to an int!). If the answer is the right answer, congratulate the user, otherwise give her the good answer
5) Once all of this is done, you can use a while loop and ask after each calculation if the user wants to continue or to stop.

Exercise 3:

In this exercise we will write a mystery word game. We will randomly choose a word from a given file and ask the user to guess the word, giving him only some letters. Here are the steps of the game:

- We choose a word from the dictionary
- We print the word only with the first letter, the other letters are replaced by asterisks(*).
- Then the user can try to guess the word.
- We compare the input and the real word. For each well placed letter in the input, we replace the asterisks of the mystery word by the real letter.
- We keep displaying the mystery and asking for a try until the user finds the good word or doesn't have more tries (You can choose a maximum number of tries, according to the length of the word for instance).

The code used to open and load a file is in "exo3.py". It works as follows:

- 'file' contains the path of the file. as "exo3.py" and the dictionary file are in the same folder, we don't need the absolute path.
- then I use an operation called "list comprehension". It will fill a list with all the words from the file (opened by open(file)) and will remove the line break at the end of each word ('\n' is the code for the linebreak).

Here are some hints:

- the list "wordlist" contains all the words from our dictionary
- Start by choosing one random word in this list
- Copy the chosen word into a list of characters, modify it to replace all the letters except the first by an asterisk, then use join() to convert the list to a string
- now, while the user doesn't find the mystery word, keep asking him to try, and store the input
- if the input is the right answer, the game is finished, the user won!
- if it's not the case, compare each letter of the input with the good word, if two letters match, remove the asterisk from the mystery word and get the good one from the word to guess.

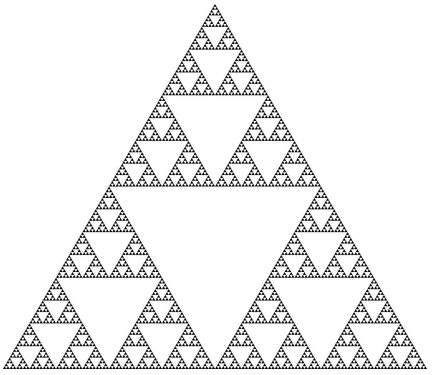
Exercise 4:
Let's draw now! We will use a module called turtle to do so.

Turtle is able to control a cursor, color pixel, draw shapes, here is the list of all commands available: https://docs.python.org/3/library/turtle.html
But here are some basics that will be sufficient:

- move forward x pixels : turtle.forward(x) or turtle.fd(x)
- move backward x pixels : turtle.backward(x) or turtle.bk(x)
- turn x degrees: turtle.right(x) or turtle.left(x)
- set the pen up or down : turtle.up() et turtle.down()
- go to position x,y : turtle.goto(x,y)
- draw a circle of radius r : turtle.circle(r)
- draw point of radius x (must be greater than 1) : turtle.dot(x)

Start by making some function to draw rectangles, squares, triangles...

You think you understood the basics? So let's get to something more interesting, a fractal! The Sierpinski triangle, here is what it looks like :



This fractal is made of triangles themselves including three triangles, endlessly.

First of all we will need a function that calculates the middle of a segment, from x1,y1 and x2,y2! If you don't remember how to do this, you just have to sum the two x and divide it by 2, and same thing for the two y.

If it's not already done, write a function that draw a triangle between points a,b and c.

Now, create a function named Sierpinski(a,b,c,n). a,b et c are the coordinates of the largest triangle and n a number we set at the beginning (let's say 10) and which will decrease after each function call.
In this function, the first thing we have to do is to check if n > 0, if not we exit the function. If n is greater than 0, we draw the triangle a,b,c, and we subtract 1 to n.
Then we will call the function Sierpinski (yes, within the function itself) for each of the 3 triangles included in the large one. To do it we call Sierpinski with a, middle[a,b] and middle[a,c], then with b, middle[b,a], middle[b,c] and finally with c, middle[a,c], middle[c,b] (and with n as the fourth argument of course). And that's it!
This kind of functions are called recursive function, because they call themselves, until some special cases where the function return a value and stop the recursive calls.

If you understood everything, congratulations! If you want to go further, ask us, we will give you some new challenges!