Discovering Python

Bertrand

October 10, 2024

1 Basics: python scripts and print function

A python script is a text file (with a `*.py` extension) containing lines of python code. When launching in the terminal the command[1] `python3 my_script.py`, the python interpreter will read the script line by line and execute the lines of code, in order. If the interpreter does not understand a line of code, it stops and returns an error in the terminal.

print function In `my_script.py`, there are 2 kinds of lines. Some start by '#': they are comments, and are ignored by the interpreter. Others are of the form `print("something")`. `print` is a **function**, that prints out in the terminal whatever it is given within its parenthesis (in this case, "something").

Try to execute the script `my_script.py`
→ Can you modify the script so that it prints `"Hello world"` only one time ?

Note: The quotes (`""`) around `"hello world"` are required, so that the interpreter understand it is given text that should be printed directly (and not be read as python code).

2 Variables

2.1 Definition

A variable is a piece of information (a number, a piece of text, ...) that we ask the program to **store** under a **name**.

For instance, if a line of code is `x = 10`, it means we are asking the program to store the value 10 under the name `x`. We may then use this variable as a number anywhere we want: to give to the print function (`print(x)`), to define another variable (par exemple `y=x+2`) ...

[1] the following command requires my_script.py to be in the current directory.

Take a look at the script `variables.py`, and execute it. The questions are in two parts:

- Part 1:
  - → What will be the value of d ? You can uncomment one of the lines to check.
  - → Likewise, what will be the value of e ? As above, print it to check.
- Part 2:
  - → Taking inspiration from the first note below, and un-commenting the last line, how would you update `d` so that is value is augmented by 2 ?

Note: You can use the value of a variable to redefine it, and therefore update it. For instance:

```
x = x + 3
```

increments the value of variable `x` by 3. If it was 10, it is now 13.

2.2 Variable type

Each variable has a **type**. For instance, the variables we have just played with were either integers (for instance `d`) or strings of letters (for instance `s`).

- → Try to put the line `z = s+d` in `variables.py`, and execute it. What happens ? Any idea why ?
- → The function `str()` allows to convert (pretty much) anything into a string. Use it to convert d into a string so that the line above works. Take inspiration from the notes below.

Note: Some functions give an **output** that you may store in a variable, or use in the definition of a variable. For instance, the function `str` outputs a string. `z = str(3)` is the same as `z = "3"`. If `a = 4` and `b = str(a)` then `b` is the string `"4"`.

Note: The addition (`+`) on strings is the **concatenation**. For instance, if `x="abc"`, `y="def"` and `z=x+y`, then `z` contains `"abcdef"`.

3 Loops and lists

3.1 Lists

So far, we have seen two kinds of variables: integers and strings. We are going to enrich our knowledge with **lists**. A list contains several elements, along a given **order**.

For example, the line of code

```
l = ["a","b","c"]
```

defines a list containing three strings, one equal to `"a"`, one equal to `"b"`, and one equal to `"c"`.

To access the elements of a list, one can use **indices**. For instance,

```
x = l[0]
```

followed by `print(x)` should print `"a"`. This is because the first element of a list is at position 0[2]. `l[1]` would have returned `"b"`.

- → How can you minimally modify the code of `lists.py` such that "rouge" is printed instead of "bleu" ?
- → Try out the following:

```
for c in l:
  print(c)
```

  Note that there must be exactly 4 spaces (or 1 tab) before print(c). Can you explain what is happening ?

4 Functions and methods

We have already encountered a two functions: `print` and `str`, which respectively print into the terminal, and convert variables to strings.

More generally, a **function** takes as **input** one or several variables and either **outputs** other variables or act on the variables it is given as inputs. Some functions are **built-in**, that is to say already included in the python language. It is the case of the `print` and `str` functions. A non-built-in function is one the programmer defines. We are going to do that yet.

Examples of built-in functions are given in the following table

[2] Why not 1 ? It is more practical for it to be 0 in more general contexts. #shh #trustme

name	what it does	example
`print`	print into the terminal	`print("bonjour")`
`str`	converts the input into a string	`s=str(3)`

Table 1: Some built-in functions in python. This table is referred to in Section 4.