PythonBasics

November 1, 2021

1 Python Basics

Python is an interpreted programming language. It means that the computer interpretes the code while it is running. In contrast, many other programming languages are compiled, and you need to compile the code before it can be run. As intepreted language, python is very interactive, which makes it easy to learn by trial and error method.

[2]:
```
print("Why python?")
```

```
Why python?
```

- It is *free and open source* and it *runs everywhere* (in your phone, Raspberry-Pi, Cray, …)
- It is *easy to learn* (interactive, simple commands, works as a desktop calculator)
- It has really *good libraries* for Machine Learning and AI
- It can be used in a *notebook*, like this, without installing anything in your own computer

1.1 What is programming?

Programming is telling instructions to the computer, to make it work for you. The computer is your slave, which you can command as you like, but it is only a little picky about the language. You need to learn it's language before you can start commanding it.

Lets start with some examples:

1.1.1 Tell the computer to say hello

The computer does not actually say anything, but it can print a message to the screen. The printing is handled using a command `print` and giving a message to the print-command. These commands are in computing slang often called as functions, and the message and other data is given to the command by enclosing it in parentheses after the function name. Just like using a functions in mathematics: $sin(0.5)$ or $y = f(x)$. Because we want computer to output a string of characters, we have to give this data in parentheses to the `print`-function. The computer wants all string literals to be quoted, so it is given then in single or double quotes, as follows:

[3]:
```
print("Hello!")
```

```
Hello!
```

The program code was in this notebook written in specific code cell, and the output from that cell is printed after the code cell, like shown above. This text is written in the specific text cell, using a markup language called (Markdown).

In similar way, the computer can be ordered to print integers, floating point numbers and even results of the calculations, etc:

[5]:
```
print(2)
print(3.14159)
print(2+2)
```

```
2
3.14159
4
```

The cell can be also used for making calculations, and the result of the last calculation is automatically printed

[47]:
```
print(2+5)
5*7
```

```
7
```

[47]:
```
35
```

1.1.2 Repeating boring things

Computer can be also made to repeat boring tasks as many time as you want. If you want to say hello to many different persons, you can repeat the printing in a loop.

- In the following code, there is first a keyword `for`. It is the beginning of the loop, and it's purpose is to repeat a same task *for* a list of values.
- `name` is a placeholder for a string, and each name in the following list will be assigned into this placeholder placeholder. This kind of placeholder is called as a *variable*, very similar than variables in mathematics.
- `in` is a keyword determining in a for-loop, the list whole values will be assigned one by one to the variable `name`.
- Next there is a list of quoted string values, the names, which are simply separated by commas. This list is exactly how you would write a list of names with pen and paper, just withoud quotes perhaps. The computer insists using quotes.
- The last character in the line is a semicolon `:`. It is also mandatory. It shows the beginning of the block which is going to be be repeated in a for loop.
- Next line (line 2) is intended with a few white spaces, which you can enter by just pressing the TAB-key once. Every line which is intended, belongs to the block, which is going to be repeated in the for-loop. The purpose of the line is to print a list of strings. The first string is the string literal "Hello", and the second string is the contents of the variable name.
- The last line (line 3) is not intended any longer, and it is not therefore repeated by the for loop, but only executed after the for-loop is finished. It will print the final string literal.
- After running the line 3, the program is finished, and it's execution is terminated, untill it is possibly re-run by you later.

[14]:
```
for name in "Aapeli", "Elmo", "Ruut", "Lea", "Harri":
    print("Hello", name)
print("Good bye.")
```

```
Hello Aapeli
Hello Elmo
Hello Ruut
Hello Lea
Hello Harri
Good bye.
```

Try it yourself: Try to add intendation in the beginning of line 3, so that it will be also repeated in the loop. Press Shft-Enter while still staying in the program cell, to run the code after the changes.

1.1.3 Looping over pre-defined number of times

For-loop can easily repeat a needed task for a certain number of times.

- In the following code, in line 1, a variable called `N` is created, and an integer-value 10 is stored in it.
- In line 2, a for loop is repeated over a range of integer values. The `range(N)` function creates a list of integers in the range of 0..N. In this case the result is the following list `0, 1, 2, 3, 4, 5, 6, 7, 8, 9`. Notice that the range starts from zero, and number 10 is not included. The produced list contains exactly 10 values.
- The print statement in line 3 is repeated within a for loop. It prints one string, which contains among other things, a placeholder for integer value `%d`. The string literal is followed by a percent sign, and the list of values which will be assigned into the placeholder in the string. In this case, the integer in the variable `i` will be expanded in place of placeholder.

[23]:
```
list(range(10))
```

[23]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

[48]:
```
N=10
for i in range(N):
    print("Current value is %d." % (i))
```

```
Current value is 0.
Current value is 1.
Current value is 2.
Current value is 3.
Current value is 4.
Current value is 5.
Current value is 6.
Current value is 7.
Current value is 8.
Current value is 9.
```

1.1.4 While

If you do not know beforehand how many times to repeat, use the `while` loop.

- The while loop in line 3, repeats the block of code until the condition holds. In this particular case, the loop is repeated until the i squared, i^2, is smaller or equal to 90.

- The repeated block contains only one line (line 4) and its only purpose is to add one to the previous value of `i`, and assign the result into `i`. This is called as incrementing the value of `i` by one in each repeat.
- When the value of `i` is incremented to 10, its squareis 100, and it is no longer smaller than 90, the conditions is false, and the loop terminates. The current value of `i` is kept and will be printed after the loop.

[26]:
```python
i=0

while (i**2)<=90:
    i = i + 1

print("The smallest number whose square is bigger than 90 is", i)
```

```
The smallest number whose square is bigger than 90 is 10
```

1.1.5 What if ...

[49]:
```python
value=5
if value<10:
    print("The value is small")
else:
    print("It is not that small any longer")
```

```
The value is small
```

Try it yourself: Try to assign different integers in `value`, and test how the program works. Try also assigning a decimal value, like 9.5 in `value`. What happens if you assing a string literal in `value`?

1.1.6 Nesting

The structures can be easily nested. In this case, the if statement is inside a block repeated by for loop.

[40]:
```python
for i in range(15):
    if i < 5:
        print("Current value = %d is small" % (i))
    elif i<10:
        print("Current value = %d is medium" % (i))
    elif i<12:
        print("Current value = %d is largish" % (i))
    else:
        print("Current value = %d is large" % (i))
```

```
Current value = 0 is small
Current value = 1 is small
Current value = 2 is small
Current value = 3 is small
Current value = 4 is small
Current value = 5 is medium
Current value = 6 is medium
Current value = 7 is medium
Current value = 8 is medium
Current value = 9 is medium
Current value = 10 is largish
Current value = 11 is largish
Current value = 12 is large
Current value = 13 is large
Current value = 14 is large
```

1.1.7 Own functions

When you have sometimes created a very convenient piece of code, wouldn't it be nice, if you could just give a name of that code and use it any time you want by just calling it by name?

It is, and it is exactly what your own functions are. They are in very center of all kinds of programming.

[46]:
```python
def compareNumbers(maxNumber):
    for i in range(maxNumber):
        if i < 5:
            print("Current value = %d is small" % (i))
        elif i<10:
            print("Current value = %d is medium" % (i))
        elif i<12:
            print("Current value = %d is largish" % (i))
        else:
            print("Current value = %d is large" % (i))

compareNumbers(10)
compareNumbers(5)
```

```
Current value = 0 is small
Current value = 1 is small
Current value = 2 is small
Current value = 3 is small
Current value = 4 is small
Current value = 5 is medium
Current value = 6 is medium
Current value = 7 is medium
Current value = 8 is medium
Current value = 9 is medium
Current value = 0 is small
Current value = 1 is small
Current value = 2 is small
Current value = 3 is small
Current value = 4 is small
```

```
[ ]:
```

1.2 Some additional links for those interested

Good tools for data analysis with Python

- Pandas for statistics, time series data processing and very convenient access to data
- Scipy for engineering functions, optimisation, intrpolation, …
- NumPy for efficient numerial calculations, linear algebra, …
- Scikit-Learn for Machine learning
- Matplotlib for plotting graphics
- Seaborn another plotting library
- Keras for deep learning, using GPU
- Python is also very versatile and can be used for many other purposes as well, and not only for ML

Typical tools for programming - Spyder MATLAB-like programming and analysis interface for Python - Jupyter notebooks and jupyterlab are web based programming environments, which support also dozens of other languages - Geany, Atom, Notepad++ VS-Code, Sublimetex, Emacs, VIM, or any other editor can be used of course - Sometimes it is convenient to work just by using interactive IPython console

```
[ ]:
```