8) Introduction to Python and variables

Agenda

- Intro to programming
- Variables
- Input
- If/else

Front-end

So far we learned about **HTML and CSS** - this is called **front-end development**. Why front-end? Because you create a **visual** look of the website with it.

But website is often more than just its looks. For example, we usually would like to have a **contact form** on our website where visitors would leave a message which would be then sent to the website owner.

You can't do this with front-end only. Sure you create a visual appearance of the contact form with HTML and CSS. But HTML&CSS **cannot** process the message and send it to the owner's email address.

This is the job that **back-end** does.

Back-end

Back-end is a sort of a brain of the website. The relationship between **web front-end** and **back-end** is similar to relation between **car body** and **car engine**. Front-end is like a car body (the looks of the car) and back-end is like a car engine.

Back-end code is usually written in one of the **programming languages**: PHP, Java, C#, Python etc. There are hundreds of programming languages. But back-end code **cannot** be written in HTML and CSS, because they are **not** programming languages (HTML is a markup language and CSS is a styling language).

We said before that HTML and CSS run in your browser. On the other hand, **back-end code runs on the server**. Remember this image?

Back-end code is the one responsible for **taking** your browser request, **evaluating** it, **processing** it and then **returning** the appropriate HTML page that then shows up in your browser. If there is something **wrong** with your request, back-end might return an error, such as the famous **404**.

Server

What is a server?

Server is a **computer that runs 24/7** (this means all the time). The purpose of this computer is to host a website. It takes requests from different clients (user's browsers) and returns back to them a response (usually an HTML webpage).

Because server returns back HTML (and CSS) code, you can see it in your browser. But you can't see a back-end code that's running on the server.

What will I learn in this lesson?

In this (and the following 6 lessons) you'll learn how to **use Python to write programs that run on the computer**. This will not be a website back-end code yet. Before you start writing back-end code, you have to learn the **basic principles of programming**.

You will learn these basics by using the **Python programming language**. But you should know that **these basics are the same in any programming language** (like PHP, Java, C#, Ruby etc.), so once you know the basics in one language it's easy to switch to another one if necessary (but we won't do this in this course).

What is programming anyway?

Programming is basically **writing commands** (or set of instructions) for our computer to execute. We write these commands in one of the programming languages, which is then automatically translated to the machine code (binary code: zeros and ones) which computer processor (or computer's brain) can then execute.

A simple task to understand programming (without actually programming) would be to write down a **set of instructions to make a sandwich**. This could look something like this:

> - *Open a kitchen cabinet.*
> - *Take out a plate.*
> - *Put a plate on the table.*
> - *Close a kitchen cabinet.*
> - *Open a fridge.*
> - *Take out salami.*
> - *Put salami on the table.*
> - *Find cheese.*
> - *If there's cheddar cheese, take cheddar.*
> - *If not, take gouda cheese.*
> - *Put cheese on the table.*
> - *etc.*

As you can see, you **have to be very specific with commands.** The above example might even be **too little** specific. You have to know that **computers are not as smart as humans are** (yet), so you have to tell them **precisely** what you want from them.

Ok, enough mumbo jumbo - let's get to action!

> ***Fun fact***
>
> *Python is named after the Monty Python's Flying Circus. The Python's creator, Guido Van Rossum, is a big fan of this British comedy group.*

Create a Python project via PyCharm

- Open PyCharm and click on **Create New Project**
- Select the right location for your project folder and type in the name of your folder (the folder does not have to be created yet, PyCharm will create it for you)
- **Virtual environment location:** Use a separate folder (outside your project folder) to create a virtual environment in.
- Also make sure you have **Python 3** selected as your **Base interpreter** (should be by default, but check anyway).

When the project is created, also create a new Python file and name it `main.py`.

Let's dive into the first programming topic: variables.

Variables

Variables are a **basic element** of any programming language. Actually each of us has already met variables. Where? In the math class in school. The concept of variables basically means that you **save (or store) some value under some name**.

```
x = 5
y = 2

x + y = 7
```

Remember this from math? We stored a value 5 under a name `x`. And value 2 under name `y`. And then we calculated the sum of these two values (using their **names**).

We do the same in programming when we store values into some variable.

Let's write it in our Python program:

```
x = 5
y = 2
```

And now we'll use a very useful Python function to show the result of adding these two numbers together:

```
print(x + y)
```

Print does not mean that the result will be printed on a piece of paper via your printer.

Instead, the result will be printed in the Python console. The console is a nice tool that will help us see what our Python programs are doing.

Running the Python program

Let's run the Python program, so we can see the result in the console.

Right-click on main.py and select "Run main":

As you'll see, PyCharm will automatically open the Python console in the bottom of the screen:

You can see the computer correctly calculated the result: `7`.

You have just created your first Python program. Congrats! :)

More than just numbers

Strings

But we can do so much more in programming than just dealing with numbers. We can also store letters and words into variables:

```
some_string = "Hey there!"

print(some_string)
```

We call this data type a **string**, which means a sequence of characters.

A string can hold either one character, multiple characters or none at all. See the following examples:

```
string_a = "a"
string_b = "I am a SmartNinja"
string_c = ""
```

Anything can be a string, as long as it's written in-between quote marks: `"` or `'`.

We can also join strings:

```
string_a = "Hello"
string_b = "World!"

print(string_a + string_b)
print(string_a + " " + string_b)
```

Boolean

Another interesting variable type is called a **boolean**. It's very simple, actually, it's either `True` or `False`. So only two options here:

```
bool_one = True
bool_two = False
```

You'll find out later on where this might come in handy. For now just know that this exists.

Nothing

Yes, we can even code nothing. Sounds funny, right? But when you think about it, it's pretty logical. We can say that some variable does not hold a value (yet):

```
some_var = None
```

We can still assign it a value later though.

Primitive data types recap

These are the main primitive data types in Python:

- **Integer:** a whole number, such as `3` or `25`
- **Float:** a decimal number, such as `2.5` or `23.764` (remember to use dot for a decimal point, not a comma!)
- **String:** a sequence of characters, always within quote marks. Example: `"this is a string"`
- **Boolean**: boolean has only two possible values, either `True` or `False`
- **nothing**: a variable can have no value. In Python we denote this as `None`

We can imagine a variable like a **cup** in which we can store (put) some value - the same way as we put coffee into a coffee cup :)

If we don't store any value in a variable (for example: `x = None`), it means the cup is empty.

Bonus: comments

Comments are a part of code that your computer will ignore. In Python we use `#` :

```
# this is a comment
```

User's input

Let's make our Python programs more **interactive**. We'll let anyone that uses the program to enter some data via Python console and the program will respond to this data.

We'll do this by using a nice little function called `input()` . Let's see how it works:

```
user_name = input("Please enter your name: ")

print("Hello " + user_name + "!")
```

First we'll ask the user to enter her/his name into the console. When the user will enter the name and hit Enter, our program will print out "Hello" and the user's name.

Pretty cool, right?

Now let's try to create a simple calculator that adds two numbers that the user writes in:

```
first_num = input("Enter the first number: ")
second_num = input("Enter the second number: ")

print(first_num + second_num)
```

Let's run this program:

Oops! Something is not right. The result should be `7` , not `52` ! What happened?

Well, let's get back to variable types. More precisely to strings and integers.

If we add two integers, we get the correct result:

```
var_a = 5
var_b = 2

print (var_a + var_b)  # result is 7
```

But if we write numbers as strings (quote marks), what do we get?

```
var_a = "5"
var_b = "2"

print (var_a + var_b)  # result is 52
```

We get a result "52". Because we just joined two strings together, "5" and "2". **Remember:** Anything that's written inside the quote marks (even if it's a number) is always a string.

The solution is to change a string into a number. We do this using the `int()` function:

```
var_a = int("5")
var_b = int("2")

print (var_a + var_b)  # result is 7
```

Now the result is correct.

Let's update our calculator example:

```
first_num = int(input("Enter the first number: "))
second_num = int(input("Enter the second number: "))

print(first_num + second_num)
```

The result is now `7`. Yaaay! :)

> *As you can see, we can **wrap** one function (`input()`) with another (`int()`).*

If/else

The last thing we'll learn today are if/else statements. I'm sure you already used them somewhere, perhaps when using an Excel spreadsheet.

If we use an if/else statement, we basically tell our program to either execute one part of the code **or** the other.

Let's see an example:

```
mood = "happy"

if mood == "happy":
    print("It is great to see you happy!")
else:
    print("Cheer up, mate!")
```

So if the mood is `"happy"`, the program will print out `"It is great to see you happy!"`.

If it's anything **else** than `"happy"`, it will print out `"Cheer up, mate!"`.

Important: Note one more thing. When we check if the mood equals `"happy"`, we use **double equal sign**: `==`. This is not a mistake.

In programming, we use single equal sign (=) to **assign** a value. And we use double equals (==) to **check for equality**. This is a very important distinction!

Elif

`elif` is short for `else if`. This keyword helps you add **more options** into your if-else statement:

```
mood = "happy"

if mood == "happy":
    print("It is great to see you happy!")
elif mood == "nervous":
    print("Take a deep breath 3 times.")
else:
    print("Cheer up, mate!")
```

> ***Code indentation*** *is very important in Python because this way computer can know what part of the code belongs to* `if` *, and what to* `else` *. Create indentation with either 4 spaces or with a Tab button.*

Q&A

Any question?

If there's enough time after Q&A, students can start working on their homework.

Homework 7.1: The mood checker

For the first exercise let's continue the mood checking program from before. Ask user to tell you what mood s/he is in:

- if the mood is "happy", the program should print out `"It is great to see you happy!"`
- if the mood is "nervous", respond with `"Take a deep breath 3 times."` . Use elif to enter more if statements: `elif mood == "nervous":` .
- Make up responses also for "sad", "excited" and "relaxed".
- The last option should be the normal `else` , which responds with `"I don't recognize this mood"` .

When you finish, **upload the code to GitHub** and share the link to it on SmartNinja **forum**.

> *Solution (https://github.com/smartninja/wd1-py3-exercises/tree/master/lesson-08/mood-checker)*

Homework 7.2: Guess the secret number

We have a first customer - a local Casino! They want to expand their business to computer gambling so they want us to build a gambling game for them. For the beginning something small and simple - a game called **Guess the secret number**.

Your task is to create this game:

- First "hard-code" some number in the program (write the number into a variable). You can call this variable `secret` .

- Then the user has to find out what this number is (using `input()`). Store user's guess in a variable called `guess`.
- Check if your `secret` is equal to user's `guess`.
- If the user's guess is wrong, let him/her know that (use `print()` and if/else).
- If the user's guess is correct, congratulate him/her.

When you finish, **upload the code to GitHub** and share the link to it on SmartNinja **forum**.

> *Solution (https://github.com/smartninja/wd1-py3-exercises/tree/master/lesson-08/guess-secret-number)*

Homework 7.3: Calculator

Write a program that does the basic arithmetic operations:

- addition (+),
- subtraction (-),
- multiplication (*),
- and division (/).

Ask the user to enter two numbers and the arithemtic operation ("+", "-", "*" or "/").

Then use if/elif/else statements to do the right operation. A hint:

```
if operation == "+":
```

When you finish, **upload the code to GitHub** and share the link to it on SmartNinja **forum**.

> *Solution (https://github.com/smartninja/wd1-py3-exercises/tree/master/lesson-08/calculator)*

Bonus links

- Python variables - examples (https://www.w3schools.com/python/python_variables.asp)
- Python numbers - examples (https://www.w3schools.com/python/python_numbers.asp) (ignore the complex numbers)
- Working with Python strings (https://www.w3schools.com/python/python_strings.asp)
- Casting Python types (https://www.w3schools.com/python/python_casting.asp)
- Guido Van Rossum's blog post about how he created Python (http://neopythonic.blogspot.com/2016/04/kings-day-speech.html)