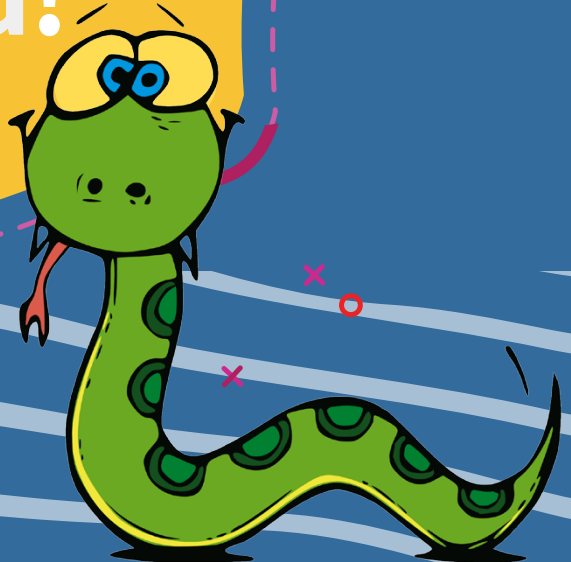


PROGRAMMING WITH python

'Hello,
world!'



Lesson # 1

Atom Installation. Data Input and Output

CONTENTS

Introduction	3
Python Programming Language	3
Preparing the PC and Installing the Main Components	4
Python 3.6.	5
Atom	8
Your First Program	11
Data Input and Output	14
The print() Function	14
The input() Function	16
Variables and Operations with Variables	17
Typical Mistakes.....	20

Introduction

Programming is the process of creating programs, which consists in writing a special code that the machine understands. **Computer code** is a kind of instruction that follows certain algorithms.

In turn, the **algorithm** is a set of rules/instructions that, as a result, solves the problem set or results in a certain result. Algorithms are used not only in programming but also in our daily lives. For example, solving a problem involves: read the statement, write what you know, calculate, write down the answer.

Python Programming Language

Python is a fairly young and promising programming language. In the framework of this course, we will study it on the examples of creating interesting games.

Why Python? We can speak at length why it is worth to learn this language, but there is a number of reasons that describe its advantages:

- **Simplicity.** Python is easy to learn and understand.
- **Conciseness.** It has a clear syntax and simple constructions. The written code will be understood by many on an intuitive level.
- **Quick start.** The beginning of programming on Python will be pretty quick and productive. Even a simple program can be written in only a few lines of code.

- **Variety.** You can use Python to develop any application (desktop, web, mobile, and others) (Figure 1).



Figure 1

Preparing the PC and Installing the Main Components

We will begin with installation of all the necessary components for further programming (Figure 2).

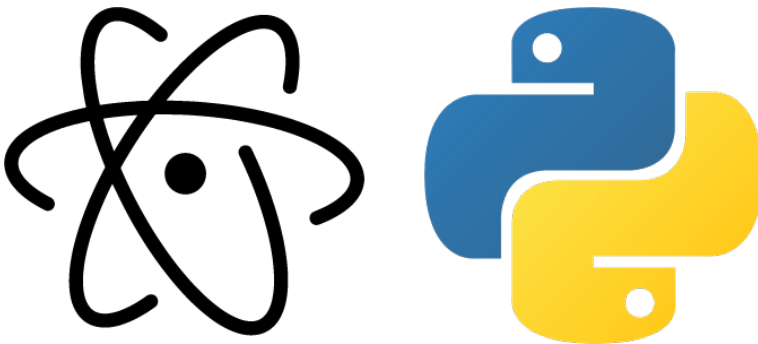


Figure 2

Python 3.6.

Of course we should do the most important thing—**install the current version of Python!**

As the title implies, we will use version 3.6, which can be downloaded from the official [Python](https://www.python.org/) website. Select the required distribution that corresponds to your OS and start the download process. Run the downloaded file and follow the instructions below.

Select **Customize installation** (Figure 3).

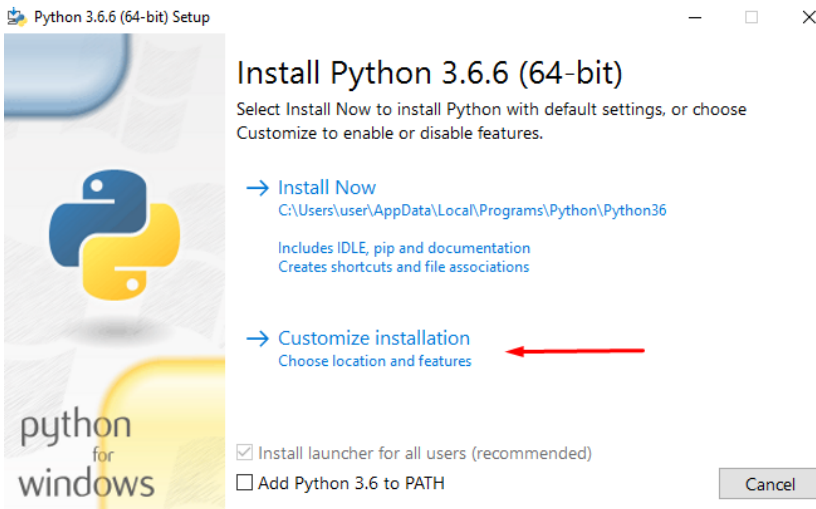


Figure 3

Then click **Next** and go to the next installation step (see Figure 4 on page 6).

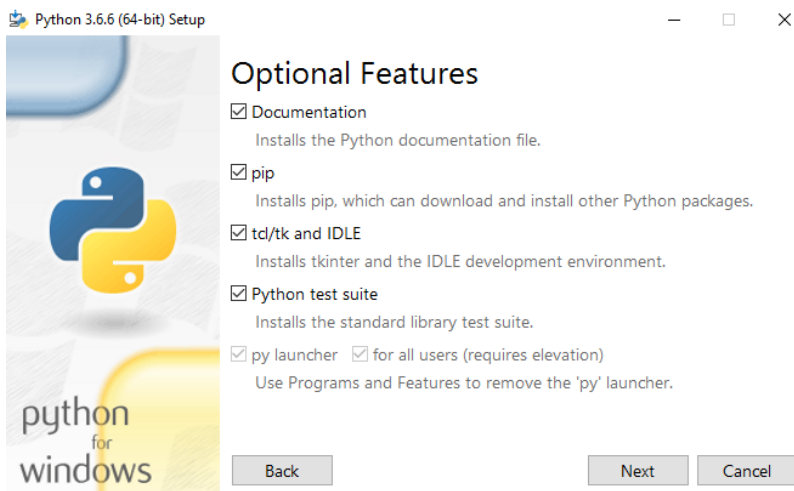


Figure 4

Be sure to check the box next to **Add Python to environment variables** and in the **Location** field, choose the path for installation (the one that is proposed or any other that meets your security policy). Then click **Install** and wait until the installation is complete (Figure 5).

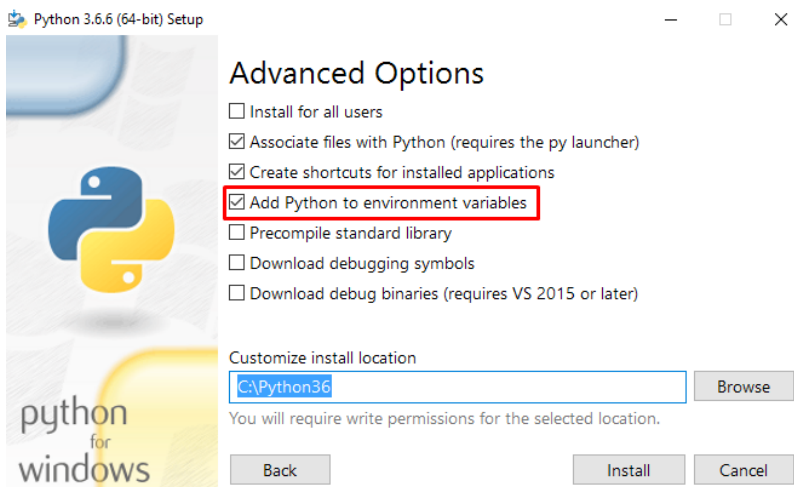
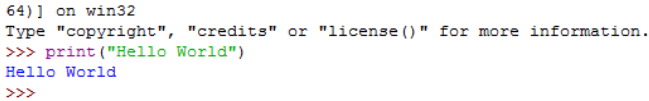


Figure 5

After that, the folder named **Python 3.6** appears in the start menu. Click on it and a list will open, find and run **IDLE (Python 3.6)**. Note that the name may vary slightly depending on the version installed. Open **Python Shell**, where you can write your code (Figure 6).



```
64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello World")
Hello World
>>>
```

Figure 6

Go to **File > New File** to create a new program file with the **.py** extension. Press **F5** to run the code.

At the moment of the code execution, **Python Shell** opens, where you will enter data and where the result will be output. The sign **>>>** means that your program is completed. To restart the code, you need to go back to the window, where you wrote the code and, as you might have guessed, press **F5**.

It is not very convenient to use the default preinstalled environment because brackets and quotation marks will not close automatically, there are no auto-transfers and prompts.

Earlier, many programmers wrote code in text editors (notepads, etc.). However, this is difficult and completely inconvenient because they had to seek for all the mistakes on their own. And what if the code is big? In this case, this process can take a very long time. **IDE** helps us improve the development process, it:

- indicates errors;
- completes commands;
- shows tips.

So now we'll move on to installing a more user-friendly development environment.

Atom

Of course, you need software for programming in Python—the **Atom development environment**. It has a lot of built-in tools for working with code and we will get acquainted with them as we learn.

Download **Atom** from [the official site](#) and run the installer (Figure 7).

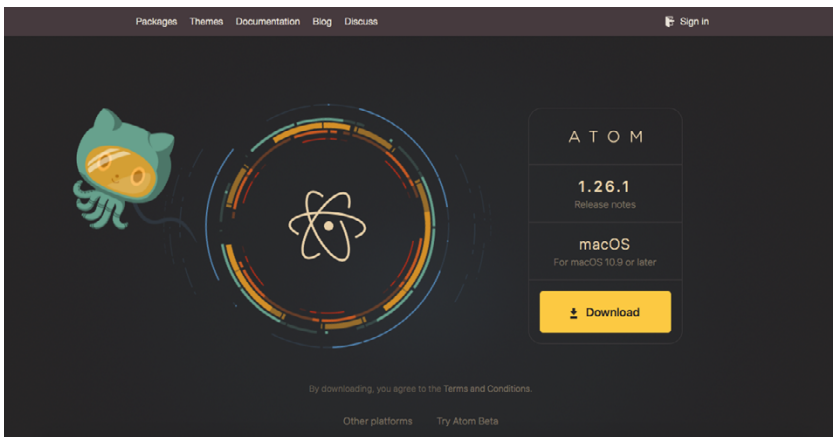


Figure 7

The **Atom** development environment window opens. Now you need to perform basic settings so that you can work and write code in the future.

Click **File** and open **Settings**, as shown in the Figure 8 (see page 9). In the **Settings** window, you can configure the environment, view installed packages, customize a

theme, configure the code editor (font, font size, and more).

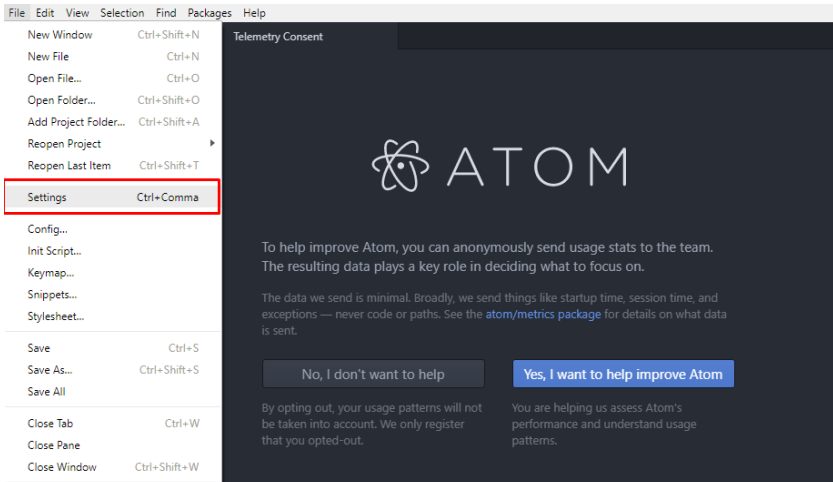


Figure 8

Click **Install** to open all packages available for installation (Figure 9).

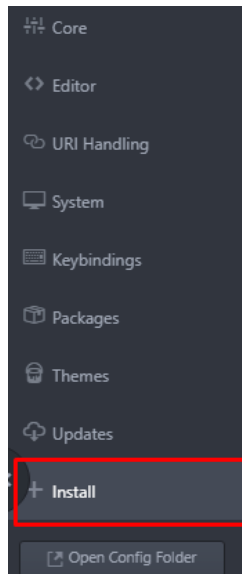


Figure 9

Type *python* in the line and click the **Packages** button (Figure 10).

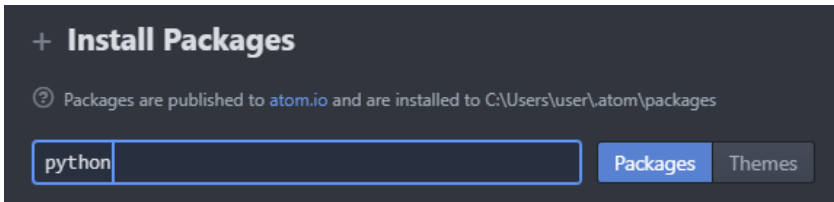


Figure 10

Go to the list of additional plug-ins, select and install **autocomplete-python** (Figure 11) and **atom-python-run** (Figure 12).

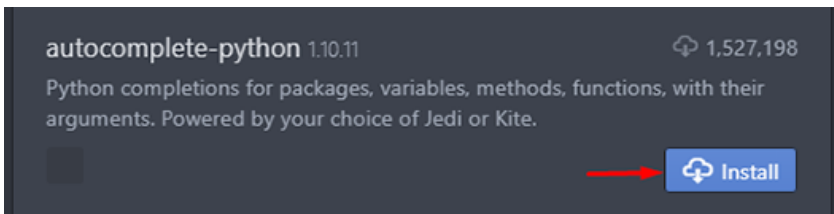


Figure 11



Figure 12

After a successful installation, the environment will enable autocompletion and running of script by pressing **F5**.

Your First Program

Let's write the simplest code and learn how to work in the **Atom**.

Click **File**, select **Add project folder** and specify the path where all project files will be saved. Right-click the folder name and select **New File** (Figure 13).

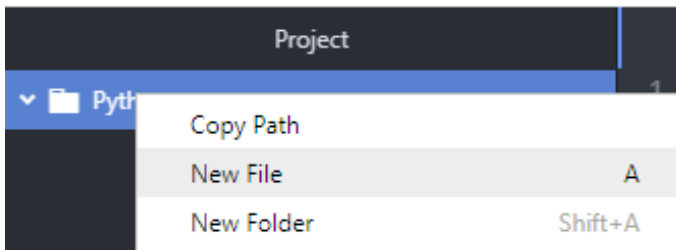


Figure 13

Name it somehow but do not forget to add the **.py** extension at the end (Figure 14).

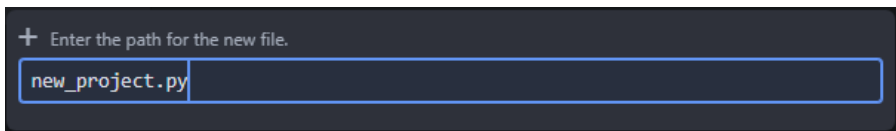


Figure 14

Great! Now we will write our first code. It will consist of only one line. This line is standard for all programs and consists of only two words: **Hello World**.

```
print("Hello World")
```

That's how it should look like in your program (Figure 15).

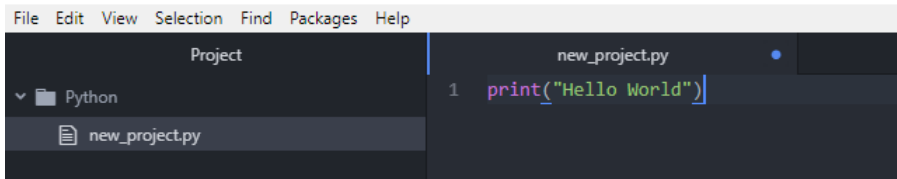


Figure 15

Run the program by pressing **F5**. As a result, a new window opens and you will see the result of the code execution in it (Figure 16).

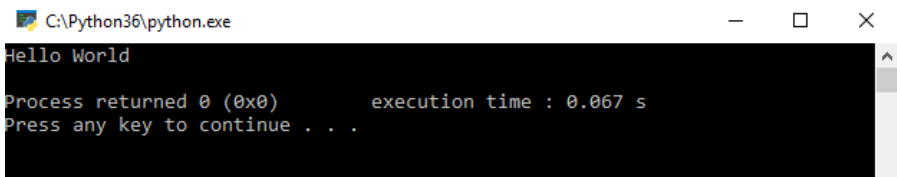


Figure 16

Well, you did it, and now let's figure out what we wrote. The **print** command prints the contents to the console (parentheses are required for any command). The text can be in any language, but it should be inside double or single quotes (Figure 17).

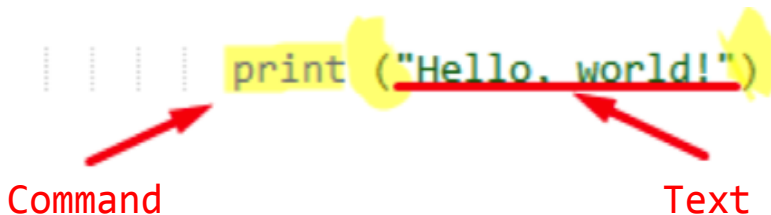


Figure 17

We will sometimes need to use additional **modules (libraries)** in Python. In this lesson we'll get to know what they are, and at the next lesson we'll consider them in more detail and even use them in our code.

You probably know what libraries are from real life. In the programming language, they are **open source code available for use by other programmers**. Often, this is very convenient because there is no need to re-write a module that will perform some standard task.

In order to import to Python, you need to write the word `import` at the very beginning of the code and specify the name of the plug-in. There may be several of them, depending on what you need. What are modules and what are they intended for? You can learn this from the official [Python Documentation contents](#).

Now let's try to import a library that will help you to display the current date and time:

```
import datetime
print(datetime.datetime.now())
```

Notice that we imported the library, then the module from this library. All this is taken from the official [Python documentation](#), therefore, once you have a question, it is strongly recommended that you refer to it.

Data Input and Output

Now let's learn how to write your program. For this we will use the already known function `print()` and the new for us `input()`.

The `print()` Function

Let's work with the `print()` function and see all the features that you probably do not know about yet.

Let's consider the case where the text should be placed in two different lines:

```
print("Hello")  
print("World")
```

Please note that this entry looks rather cumbersome. And what if you need to place more than two such lines, for example? This is easy to fix if we enter `Hello World` in one line and separate them with `\n`. This is a **string literal** (`\n`) used for line break. Thus, this text will be displayed in two lines, similar to the two `print()` functions:

```
print("Hello\nWorld")
```

Thus, both variants result in two separate lines (Figure 18).

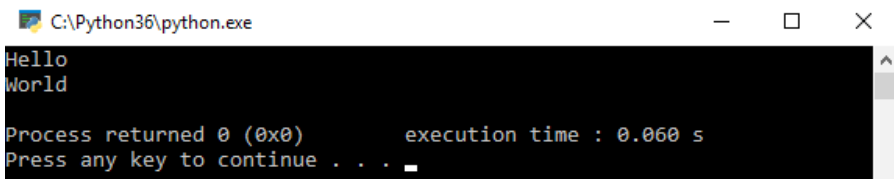


Figure 18

Now let's consider ways to output text in a single line. There are several ways to do it actually. Use comma for this, and data will be output separated by space.

As you can see, there is nothing complicated here:

```
print("Hello", "World")
```

And now we'll use the `+` sign that adds two strings:

```
print("Hello" + "World")
```

The `end= ' '` parameter allows replacing line break with space:

```
print("Hello", end=' ')  
print("World")
```

The result will be the same (Figure 19).

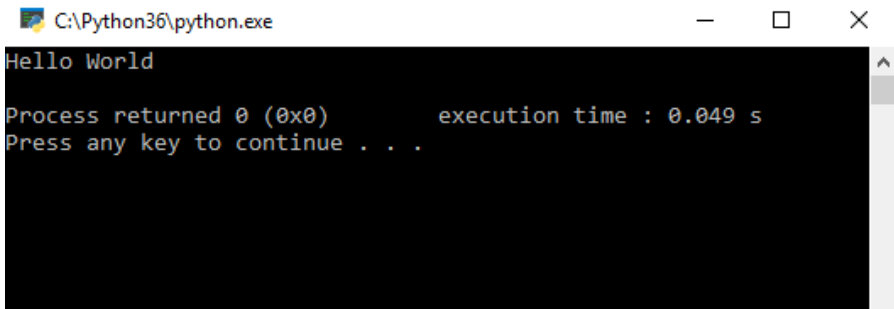


Figure 19

Let's figure out what to do when we need to substitute a value. In this case it is much more convenient to use formatting of lines.

Let's look at the simplest method:

```
a = "World"
print("Hello {}".format(a))
```

How it works: first comes the text with the formatted template "Hello {}", in this entry {} means that it will get the value specified in the `format()` method. In the code, `a` is a variable used to write a value to it.

Use the following method if you want to use several arguments and output them in different places:

```
a = 1
b = 0
print(f"Pupils = {a} Students={b}")
```

Code will result in this string:

`Pupils = 1 Students=0.`

We'll also consider the `sep` parameter, which is used as a separator. When we use `sep=", "`, each of the parameters will be separated by comma, excluding the last one:

```
print("small", "medium", "large")
print("small", "medium", "large", sep="")
print("small", "medium", "large", sep=", ")
```

The `input()` Function

The `input()` function is responsible for data input. It terminates the program execution and waits for a user to enter data.

Let's look at a simple example where we offer the user to enter their name and then say hello to them:

```
name = input("Your name: ")
print("Hello, " + name)
```

Text entered inside `input()` will be print to the console, after this the user will have to write their name and press **Enter**. In our code, `name` is a variable that stores values.

Variables and Operations with Variables

Python programming language has a number of arithmetic operations that you will need. In our program, we can multiply, divide, add, subtract, raise to a power. The operators of addition, multiplication, etc. are used to perform the above operations.

We have already figured out how to work with the `print()` function, and now let's use it to print results of performing the simplest arithmetic operations in the console.

Operation	Name	Example	Result
+	Addition	<code>print(8+2)</code>	10
-	Subtraction	<code>print(8-2)</code>	6
*	Multiplication	<code>print(8*2)</code>	16
**	Raising to power	<code>print(8**2)</code>	64

Operation	Name	Example	Result
/	Division	<code>print(8/2)</code>	4.0
//	Floor division	<code>print(8//2)</code>	4
%	Remainder of the division	<code>print(8%2)</code>	0

Note that we use the same numbers every time (8 and 2). Imagine that you have a large program and suddenly you had to change 8 to 10. With this form of writing you will have to search for this value every time and change it. Imagine how much extra work you will do!

To simplify the task, we can simply use variables. We will need them, one way or another, almost in each program. **Variables are named cells that store data.** Put simply, it's like a box where we send a value (number, word), and then we get it from the same box. In order not to get confused with what we put in what box, we give it a name (Figure 20).

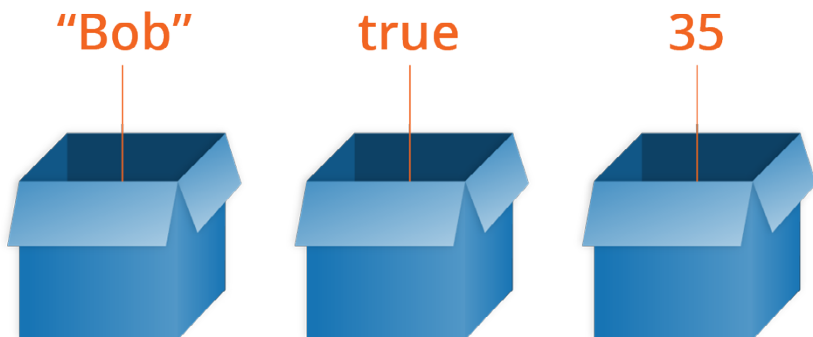


Figure 20

The conclusion from the above is that to work with a variable, you must give it a name and assign a value to it. It is important to remember that a variable name cannot begin with a number. It is acceptable to use upper and lower case, lower underscores, numbers after letters or words.

```
a = 8
b = 2
print(a+b)
```

Variables may be of different **types: numbers, strings, lists, etc.**

Variable Type	Name	Example
int	Integer	-150 0 13
float	Floating-point number	-12.0 1.1 150.5
str	String	"Hello" "My name is.."
bool	Logical data type	True False

Typical Mistakes

Python has a simple syntax, but even it has a set of rules that cannot be violated in any way.

1. **A comment is part of the code that does not affect the operation of the program** and is used as programmer's notes. Perhaps you and your friends will develop a big project, and one day you will need to edit one or another section of the code.

Comments are written in order to not remember what each code snippet is responsible for. They can be single-line and multiline. A single-line comment begins with `#`:

```
# comments
```

Multiline comments are written with `'''...'''`:

```
'''  
comments  
comments  
comments  
'''
```

Writing a multiline comment with `#` is an error. In this case the program won't work correctly because it will try to process it.

2. **Variable names must be created strictly according to rules.** Thus, don't write a name with a capital letter, separate

words with a lower underscore, do NOT put a space or other symbols between words.

Correct	Incorrect
variable variable1 my_variable	lvariable my-variable my variable

3. Semicolon at the end of the string is not required.

```
a = 8
b = 0.8
c = "string"
d = True
```

4. Instructions are combined into blocks by the amount of indentations. Use **tabulation** to create blocks of code (**four spaces**). In the IDE, they are automatically set when the line breaks (**Enter**) or when **Tab** is pressed. This is especially important. Do not try to set tabulation with spaces, otherwise an error will occur.

Correct	Incorrect
<code>print(a)</code> <code>print(b)</code>	<code>print(a)</code> <code>print(b)</code>

5. Another common mistake may be the absence of a quote inside `print()`. Do not forget that you should

not only open quotes but also close them. The same goes for brackets.

Correct	Incorrect
<code>print("Hello World")</code>	<code>print("Hello World)</code>

6. Another common mistake is the mismatch of data types.

We already understood that there are different types of data. For example, if we add an integer (`int`) and a string (`str`), an error will be output to the console.

Correct	Incorrect
<pre>a = "Happy" b = " New Year" print(a + b)</pre>	<pre>a = "Happy New Year" b = 2020 print(a + b)</pre>

7. Remember that before you refer to a variable, you should declare it first. Otherwise an error will be thrown.

Correct	Incorrect
<pre>a = 8 b = 22 print(a+b)</pre>	<pre>a = 8 print(a+b)</pre>

8. It's important not to confuse `=` and `==` since they are two different signs. The first one is an assignment and indicates that the value on the right will be stored in the variable that is on the left (`a = 5`). The second one compares whether

the value on the left is equal to the value on the right. As a result of this code, an error message will appear.

Correct	Incorrect
<pre>a = 8 b = 22 print(a+b)</pre>	<pre>a == 8 b == 22 print(a+b)</pre>



Lesson # 1

Atom Installation. Data Input and Output

© STEP IT Academy

www.itstep.org

All rights to protected pictures, audio, and video belong to their authors or legal owners. Fragments of works are used exclusively in illustration purposes to the extent justified by the purpose as part of an educational process and for educational purposes in accordance with Article 1273 Sec. 4 of the Civil Code of the Russian Federation and Articles 21 and 23 of the Law of Ukraine "On Copyright and Related Rights". The extent and method of cited works are in conformity with the standards, do not conflict with a normal exploitation of the work, and do not prejudice the legitimate interests of the authors and rightholders. Cited fragments of works can be replaced with alternative, non-protected analogs, and as such correspond the criteria of fair use.

All rights reserved. Any reproduction, in whole or in part, is prohibited. Agreement of the use of works and their fragments is carried out with the authors and other right owners. Materials from this document can be used only with resource link.

Liability for unauthorized copying and commercial use of materials is defined according to the current legislation of Ukraine.