



IIT Madras
BSc Degree

[Home](#)[Lesson-1.2](#)

Lesson-1.1

Lesson-1.1

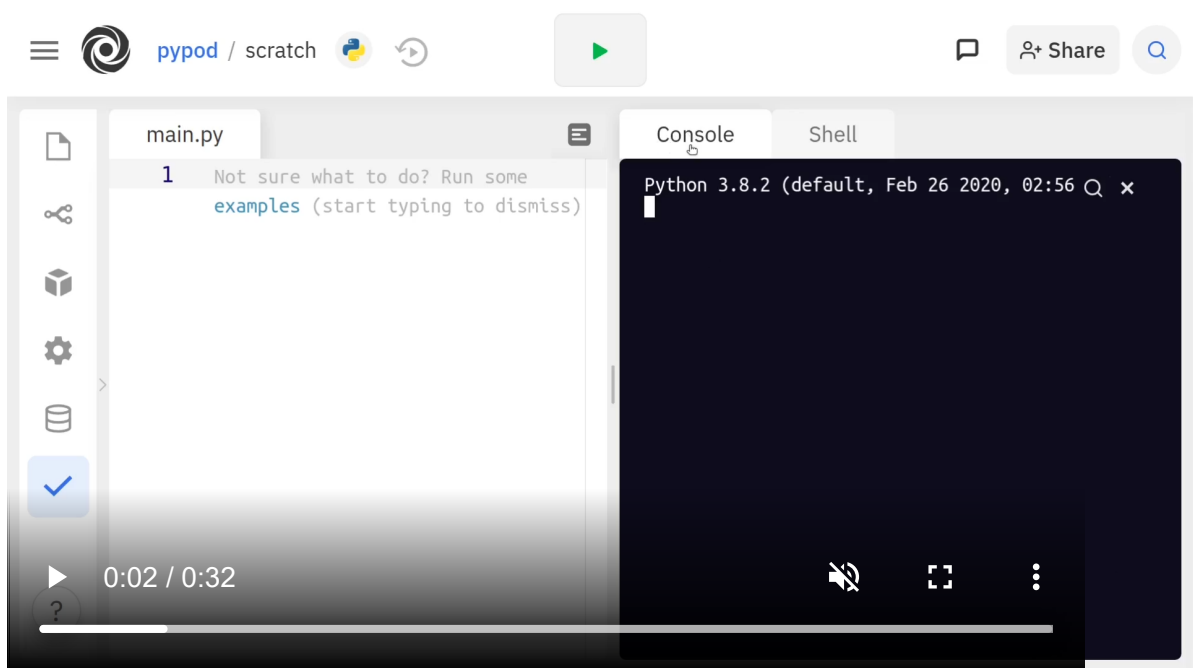
[Python shell | Replit Console](#)[Prompts](#)[Output](#)[Emojis](#)[Literals and Variables](#)[Basic Data Types | type\(\)](#)[Integer](#)[Float](#)[String](#)[Boolean](#)[Comments](#)

Python shell | Replit Console

In this lesson, we will be working with the Python interpreter in the interactive mode. This is often called a Python shell. It is a tool that lets us execute individual lines of code and see the output right away. We will drop the phrase "interactive mode" and just refer to it as the interpreter. Have a look at the official Python documentation for more details about the [Python Interpreter](#). If you have Python installed on your system, then the Python shell will look like this:

```
>>>
>>>
>>>
>>> print('Hello World!')
Hello World!
>>>
>>>
>>> 
```

In Replit, this corresponds to the console screen on the right of the repl. This will be our playground for quite sometime:



Prompts



The orange symbol that is displayed above is called a prompt. Its role is similar to that of the blinking cursor while editing documents. It is an invitation to type code. Code that is typed at the prompt is executed by the interpreter. In these lessons, we will use the following symbol to refer to the prompt: `>>>`.

We are all set to write our first line of code:

```
1 >>> print('Hello world!')
2 Hello world!
```

Fire up a repl and type the code in the console. You should be getting the output on the next line.

Output

Let us take a closer look at the first line of code that we wrote. `print` is called a built-in function in Python. A function is an object that accepts inputs and returns outputs. The term built-in refers to the fact that this function is something that is readily provided by Python for our use.

```
1 >>> print('Hello world!')
2 Hello world!
3 >>> print("Hello world!")
4 Hello world!
```

The object inside the parenthesis of the `print` function is called a string. A string is a sequence of characters enclosed in quotes. Strings can either be in single quotes or double quotes. However, a single quote can't be matched against a double quote to enclose a string. We have used single quotes in line 1 and double quotes in line 3. Both lines give identical outputs. The ability to use both single quotes and double quotes comes in handy in situations like this:

Print a string that has an apostrophe in it:

```
1 >>> print("India's capital is New Delhi.")
```

Run the code given above and observe the output. `print` can also be used to print numbers:

```
1 >>> print(1)
2 1
3 >>> print(2.0)
4 2.0
```

Multiple items can be printed on the same line in the following way:

```
1 >>> print(1, 2)
2 1 2
3 >>> print('online', 'degree', 'program')
4 online degree program
```

Note the presence of a space between successive elements. If the `print` command is called without passing any input to it, then it prints a blank line:

```
1 >>> print()
2
3 >>>
```

What happens if we just use type `print` without having the parenthesis?

```
1 >>> print
2 <built-in function print>
```

We don't get an error. Instead, the message is that `print` is a built-in function. But the following code throws an error:


```
1 >>> print 'Hello world!'
2 File "<stdin>", line 1
3     print 'Hello world!'
4         ^
5 SyntaxError: Missing parentheses in call to 'print'. Did you mean
  print('Hello world!')?
```

The interpreter hits back with a `SyntaxError`. Think about the syntax like the grammar of human languages. In the code given above, we have missed the parentheses. The fourth lesson will take up this issue in greater detail.


Emojis

Before we jump into the serious stuff, let us try and print some emojis!


```
❏ print('\N{smiling face with smiling eyes}')
```




```
❏ print('\N{grinning face}')
```



```
❏ print('\N{smiling face with halo}')
```



```
❏ print('\N{thinking face}')
```



Try this out in your repl! A full list of emojis can be found [here](#).

Literals and Variables

Strings like `'Hello world!'` and numbers like `1`, `2.0` are called literals in Python. Formally, a literal is something that describes a constant value. Variables are containers that are used to store values. Variables in Python are defined in the following way:

```
1 >>> x = 1
2 >>> print(x)
3 1
4 >>> y = 'a string'
5 >>> print(y)
6 a string
7 >>> foo_bar = 123.456
8 >>> print(foo_bar)
9 123.456
```

`=` is called the assignment operator. Whenever the assignment operator is present in a statement, it is used for one of the following purposes:

- define a new variable
- update an existing variable

```
1 >>> x = 1      # define a new variable
2 >>> x = x + 1   # update an existing variable
3 >>> print(x)
4 2
```

The assignment operator is evaluated from right to left. That is, the expression to the right of the assignment operator is evaluated first. This result is then assigned to the variable on the left. Variables will be taken up in greater detail in the lessons of the second chapter.

Basic Data Types | type()

We will be looking at the following basic data types:

- Integer
- Float
- String
- Boolean

Integer

The `int` type represents integers. Python provides a command called `type` to determine the type of an object:

```
1 >>> print(1)
2 1
3 >>> type(1)
4 <class 'int'>
```

Float

The `float` type represents real numbers:

```
1 >>> print(1.0)
2 1.0
3 >>> type(1.0)
4 <class 'float'>
```

The following is also a valid float literal:

```
1 >>> print(1.)
2 1.0
```

`1.` and `1.0` are one and the same literal.

String

The `str` type represents strings:

```
1 >>> print('one')
2 one
3 >>> type("one")
4 <class 'str'>
```

Boolean

The `bool` type represents boolean values:

```
1 >>> print(True)
2 True
3 >>> type(False)
4 <class 'bool'>
```

Please note that `bool` values are case sensitive. That is, `true` and `false` are not `bool` values.

Comments

A comment is a line of text that is not executed by the interpreter. Comments begin with the `#` symbol. The following are comments:

```
1 >>> # This is a comment
2 >>> # print(1)
3 >>>
```

As line-2 is a comment, `1` is not printed in the next line. Comments can also come at the end of a line of code:

```
1 >>> print(1) # This line is printing the value 1
2 1
```

Adding comments is one of the ways to make code more readable. Its use will become clear in subsequent chapters.