**CHAPTER 2**

**Python Basics**

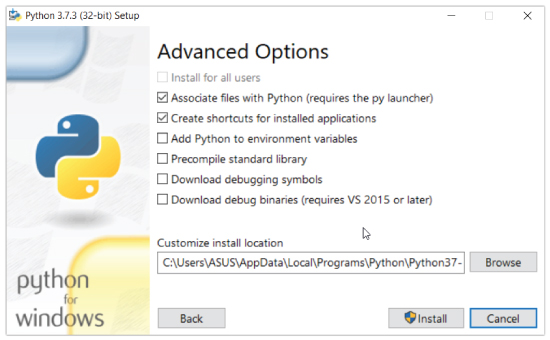
# **PYTHON INTRODUCTION**

Python is a cross-platform programming language, meaning, it runs on multiple platforms like Windows, MacOS, Linux and has even been ported to the Java and .NET virtual machines. It is free and open source.

Even though most of today’s Linux and Mac have Python preinstalled in it, the version might be out-of-date. So, it is always a good idea to install the most current version.

1. Download the [latest version of Python](https://www.python.org/downloads/).

Run the installer file and follow the steps to install Python  
during the install process, check Add Python to environment variables. This will add Python to environment variables and you are able to run Python from any part of the computer.  
Also, you can choose the path where Python is installed.

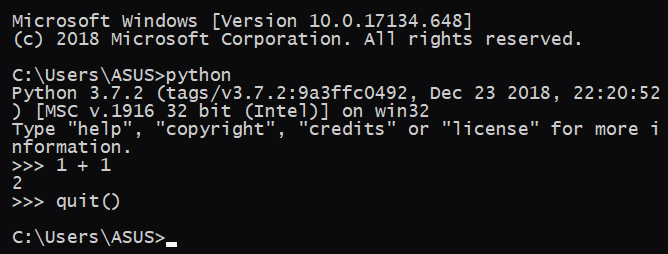


Once you finish the installation process, you can run Python.

1. **Run Python in Immediate mode**

Once Python is installed, typing python in the command line will invoke the interpreter in immediate mode. We can directly type in Python code and press enter to get the output.

Try typing in 1 + 1 and press enter. We get 2 as the output. This prompt can be used as a calculator. To exit this mode type quit() and press enter.

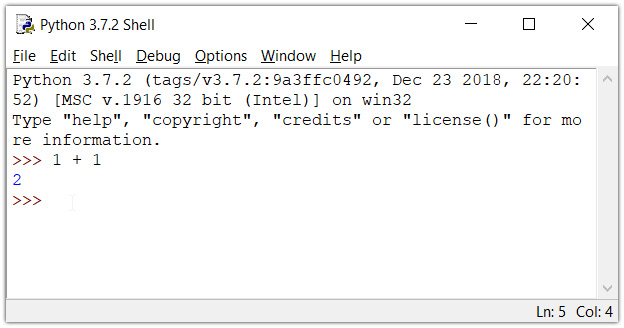


**2. Run Python in the Integrated Development Environment (IDE)**

We can use any text editing software to write a Python script file. We just need to save it with the .py extension. But using an IDE can make our life a lot easier. IDE is a piece of software that provides useful features like code hinting, syntax highlighting and checking, file explorers etc. to the programmer for application development.

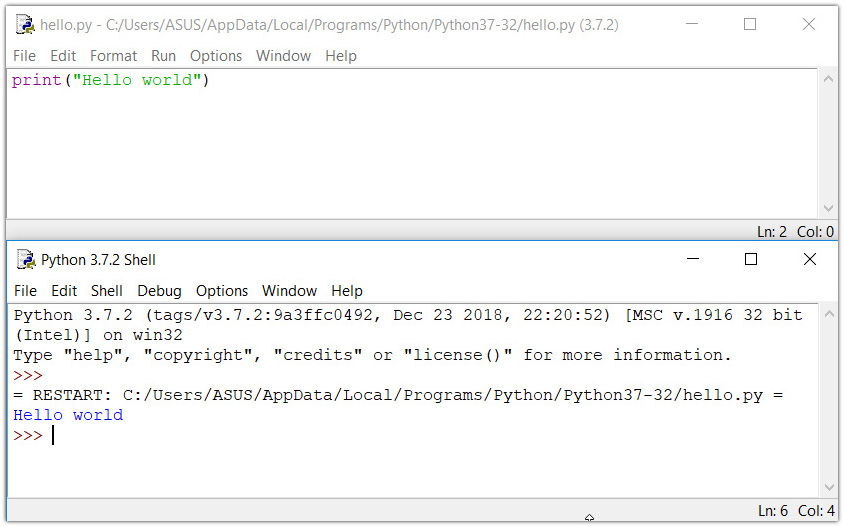
By the way, when you install Python, an IDE named **IDLE** is also installed. You can use it to run Python on your computer. It's a decent IDE for beginners.

When you open IDLE, an interactive Python Shell is opened.



Now you can create a new file and save it with **.py** extension. For example, **hello.py**

Write Python code in the file, save it. To run the file go to **Run** > **Run Module** or simply click **F5**.



**Your first Python Program**

Now that we have Python up and running, we can write our first Python program. Let's create a very simple program called "Hello World!".  A "Hello, World!" is a simple program that outputs Hello, World! on the screen. Since it's a very simple program, it's often used to introduce a new programming language to a newbie.

Type the following code in any text editor or an IDE and save it as helloWorld.py

print ("Hello world!")

Then, run the file. You will get the following output.

Hello world!

# **Python Keywords and Identifiers**

## Python Keywords

Keywords are the reserved words in Python. We cannot use a keyword as a variable name, function name or any other identifier. They are used to define the syntax and structure of the Python language. In Python, keywords are case sensitive. There are 33 keywords in Python 3.7. This number can vary slightly in the course of time.

All the keywords except True, False and none are in lowercase and they must be written as it is. The list of all the keywords is given below.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | False | class | finally | is | Return | | None | continue | for | lambda | Try | | True | def | from | nonlocal | while | | And | del | global | not | with | | As | elif | if | or | yield | | Assert | else | import | pass |  | | Break | except | in | raise |  | |  | | | | | |

## Python Identifiers

An identifier is a name given to entities like class, functions, variables, etc. It helps to differentiate one entity from another.

### Rules for writing identifiers

Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore \_. Names like myClass, var\_1 and print\_this\_to\_screen, all are valid example.

An identifier cannot start with a digit. 1variable is invalid, but variable1 is perfectly fine.

Keywords cannot be used as identifiers.

* 1. >>> global = 1
  2. File "<interactive input>", line 1
  3. global = 1
  4. ^
  5. SyntaxError: invalid syntax

1. We cannot use special symbols like **!**, **@**, **#**, **$**, **%** etc. in our identifier.
   1. >>> a@ = 0
   2. File "<interactive input>", line 1
   3. a@ = 0
   4. ^
   5. SyntaxError: invalid syntax
2. Identifier can be of any length

### Things to Remember

Python is a case-sensitive language. This means, Variable and variable are not the same. Always name identifiers that make sense.

While, c = 10 is valid. Writing count = 10 would make more sense and it would be easier to figure out what it does even when you look at your code after a long gap.

Multiple words can be separated using an underscore, this\_is\_a\_long\_variable.

# **Python Statement, Indentation and Comments**

## Python Statement

Instructions that a Python interpreter can execute are called statements. For example, a = 1is an assignment statement. if statement, for statement, while statement etc. are other kinds of statements which will be discussed later.

### Multi-line statement

In Python, end of a statement is marked by a newline character. But we can make a statement extend over multiple lines with the line continuation character (\). For example:

1. a = 1 + 2 + 3 + \
2. 4 + 5 + 6 + \
3. 7 + 8 + 9

This is explicit line continuation. In Python, line continuation is implied inside parentheses ( ), brackets [ ] and braces { }. For instance, we can implement the above multi-line statement as

1. a = (1 + 2 + 3 +
2. 4 + 5 + 6 +
3. 7 + 8 + 9)

Here, the surrounding parentheses ( ) do the line continuation implicitly. Same is the case with [ ] and { }. For example:

1. colors = ['red',
2. 'blue',
3. 'green']

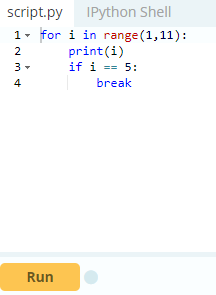
We could also put multiple statements in a single line using semicolons, as follows

1. a = 1; b = 2; c = 3

## Python Indentation

Most of the programming languages like C, C++, Java use braces { } to define a block of code. Python uses indentation.

A code block (body of a [function](https://www.programiz.com/python-programming/function), [loop](https://www.programiz.com/python-programming/for-loop) etc.) starts with indentation and ends with the first unintended line. The amount of indentation is up to you, but it must be consistent throughout that block. Generally four whitespaces are used for indentation and is preferred over tabs. Here is an example.



The enforcement of indentation in Python makes the code look neat and clean. This results into Python programs that look similar and consistent. Indentation can be ignored in line continuation. But it's a good idea to always indent. It makes the code more readable. For example:

1. if True:
2. print('Hello')
3. a = 5

and

1. if True: print('Hello'); a = 5

both are valid and do the same thing. But the former style is clearer.Incorrect indentation will result into Indentation Error.

## Python Comments

Comments are very important while writing a program. It describes what's going on inside a program so that a person looking at the source code does not have a hard time figuring it out. You might forget the key details of the program you just wrote in a month's time. So taking time to explain these concepts in form of comments is always fruitful.

In Python, we use the hash (#) symbol to start writing a comment. It extends up to the newline character. Comments are for programmers for better understanding of a program. Python Interpreter ignores comment.

1. #This is a comment
2. #print out Hello
3. print('Hello')

### Multi-line comments

If we have comments that extend multiple lines, one way of doing it is to use hash (#) in the beginning of each line. For example:

1. #This is a long comment
2. #and it extends
3. #to multiple lines

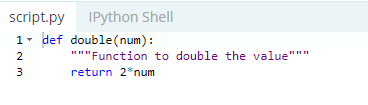
Another way of doing this is to use triple quotes, either ''' or """.

These triple quotes are generally used for multi-line strings. But they can be used as multi-line comment as well. Unless they are not docstrings, they do not generate any extra code.

1. """This is also a
2. perfect example of
3. multi-line comments"""

### Docstring in Python

Docstring is short for documentation string. It is a [string](https://www.programiz.com/python-programming/string) that occurs as the first statement in a module, function, class, or method definition. We must write what a function/class does in the docstring. Triple quotes are used while writing docstrings. For example:



Docstring is available to us as the attribute \_\_doc\_\_ of the function. Issue the following code in shell once you run the above program.

1. >>> print(double.\_\_doc\_\_)
2. Function to double the value

# **Python Variables, Constants and Literals**

## Python Variables

A variable is a named location used to store data in the memory. It is helpful to think of variables as a container that holds data which can be changed later throughout programming. For example,

1. number = 10

Here, we have created a named number. We have assigned value 10 to the variable.

You can think variable as a bag to store books in it and those books can be replaced at any time.

1. number = 10
2. number = 1.1

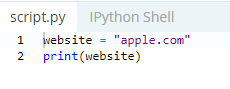
Initially, the value of number was 10. Later it's changed to 1.1.

Note: In Python, we don't assign values to the variables, whereas Python gives the reference of the object (value) to the variable.

### Assigning a value to a Variable in Python

As you can see from the above example, you can use the assignment operator = to assign a value to a variable.

**Example 1: Declaring and assigning a value to a variable**



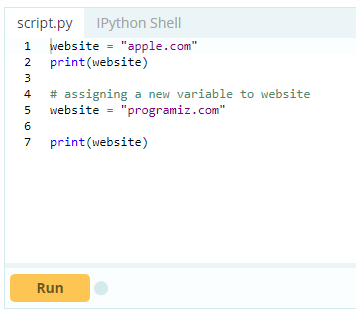
When you run the program, the output will be:

apple.com

In the above program, we assigned a value apple.com to the variable website. Then we print the value assigned to website i.e. apple.com

Note: Python is a [type inferred](https://en.wikipedia.org/wiki/Type_inference) language; it can automatically know apple.com is a string and declare website as a string.

**Example 2: Changing the value of a variable**



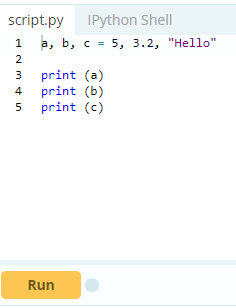
When you run the program, the output will be:

apple.com

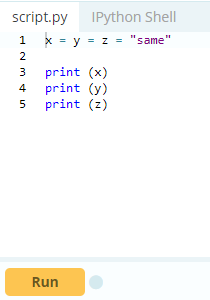
programiz.com

In the above program, we have assigned apple.com to the website variable initially. Then, it's value is changed to programiz.com.

**Example 3: Assigning multiple values to multiple variables**



If we want to assign the same value to multiple variables at once, we can do this as



The second program assigns the same string to all the three variables x, y and z.

## Constants

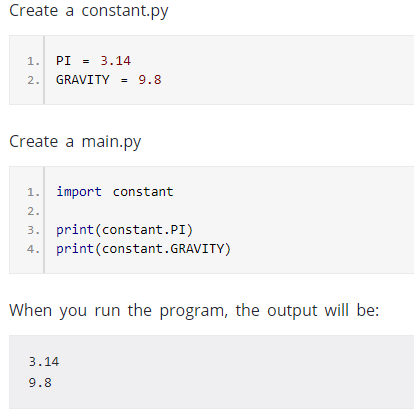
A constant is a type of variable whose value cannot be changed. It is helpful to think of constants as containers that hold information which cannot be changed later.

Non technically, you can think of constant as a bag to store some books and those books cannot be replaced once placed inside the bag.

### Assigning value to a constant in Python

In Python, constants are usually declared and assigned on a module. Here, the module means a new file containing variables, functions etc which is imported to main file. Inside the module, constants are written in all capital letters and underscores separating the words.

#### **Example 3: Declaring and assigning value to a constant**



In the above program, we create a constant.py module file. Then, we assign the constant value to PI and GRAVITY. After that, we create a main.py file and import the constant module. Finally, we print the constant value.

Note: In reality, we don't use constants in Python. The globals or constants module is used throughout the Python programs

## Rules and Naming convention for variables and constants

1. Create a name that makes sense. Suppose, vowel makes more sense than v.
2. Use camel Case notation to declare a variable. It starts with lowercase letter. For example:



1. Use capital letters where possible to declare a constant. For example:



1. Never use special symbols like !, @, #, $, %, etc.
2. Don't start name with a digit.
3. Constants are put into Python modules and meant not be changed.
4. Constant and variable names should have combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (\_). For example:



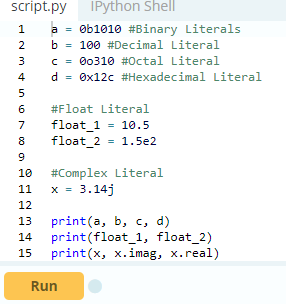
## Literals

Literal is a raw data given in a variable or constant. In Python, there are various types of literals they are as follows:

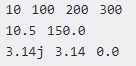
### Numeric Literals

Numeric Literals are immutable (unchangeable). Numeric literals can belong to 3 different numerical types Integer, Float and Complex.

#### **Example 4: How to use Numeric literals in Python?**



When you run the program, the output will be:



In the above program,

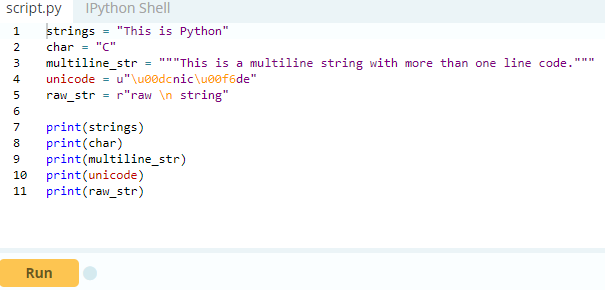
* We assigned integer literals into different variables. Here, a is binary literal, b is a decimal literal, c is an octal literal and d is a hexadecimal literal.
* When we print the variables, all the literals are converted into decimal values.
* 10.5 and 1.5e2 are floating point literals. 1.5e2 is expressed with exponential and is equivalent to 1.5 \* 102.
* We assigned a complex literal i.e 3.14j in variable x. Then we use imaginary literal (x.imag) and real literal (x.real) to create imaginary and real part of complex number.

To learn more about Numeric Literals, refer [Python Numbers](https://www.programiz.com/python-programming/numbers).

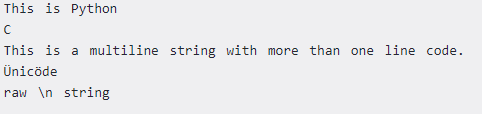
### String literals

A string literal is a sequence of characters surrounded by quotes. We can use both single, double or triple quotes for a string. And, a character literal is a single character surrounded by single or double quotes.

#### **Example 7: How to use string literals in Python?**



When you run the program, the output will be:

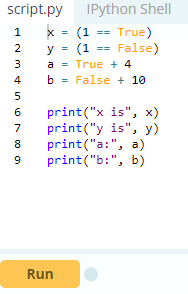


In the above program, This is Python is a string literal and C is a character literal. The value with triple-quote """ assigned in the multiline\_str is multi-line string literal. The u"\u00dcnic\u00f6de" is a unicode literal which supports characters other than English and r"raw \n string" is a raw string literal.

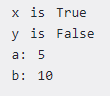
### Boolean literals

A Boolean literal can have any of the two values: True or False.

#### **Example 8: How to use boolean literals in Python?**



When you run the program, the output will be:



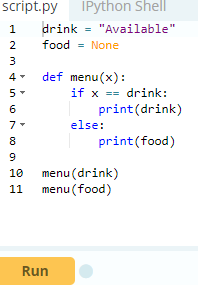
In the above program, we use boolean literal True and False. In Python, True represents the value as 1 and False as 0. The value of x is True because 1 is equal to True. And, the value of y is False because 1 is not equal to False.

Similarly, we can use the True and False in numeric expressions as the value. The value of a is 5 because we add True which has value of 1 with 4. Similarly, b is 10 because we add the False having value of 0 with 10.

### Special literals

Python contains one special literal i.e. none. We use it to specify to that field that is not created.

#### **Example 9: How to use special literals in Python?**



When you run the program, the output will be:

Available

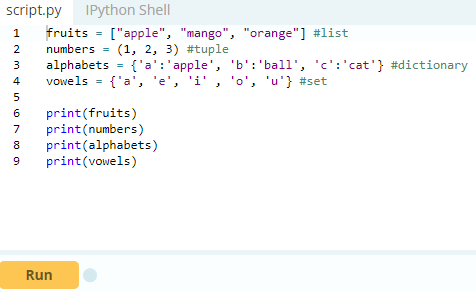
None

In the above program, we define a menu function. Inside menu, when we set parameter as drink then, it displays Available. And, when the parameter is food, it displays None.

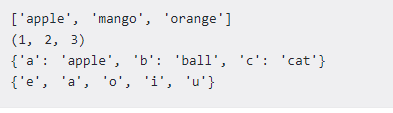
### Literal Collections

There are four different literal collections List literals, Tuple literals, Dict literals, and Set literals.

#### **Example 10: How to use literals collections in Python?**



When you run the program, the output will be:



In the above program, we created a list of fruits, tuple of numbers, dictionary dict having values with keys designated to each value and set of vowels.

# **Python Data Types**

## Data types in Python

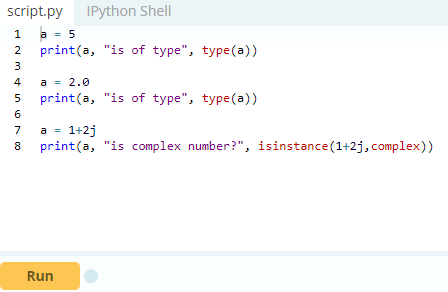
Every value in Python has a datatype. Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes.

There are various data types in Python. Some of the important types are listed below.

### Python Numbers

Integers, floating point numbers and complex numbers falls under [Python numbers](https://www.programiz.com/python-programming/numbers) category. They are defined as int, float and complex class in Python.

We can use the type() function to know which class a variable or a value belongs to and the isinstance() function to check if an object belongs to a particular class.



Integers can be of any length, it is only limited by the memory available. A floating point number is accurate up to 15 decimal places. Integer and floating points are separated by decimal points. 1 is integer, 1.0 is floating point number.

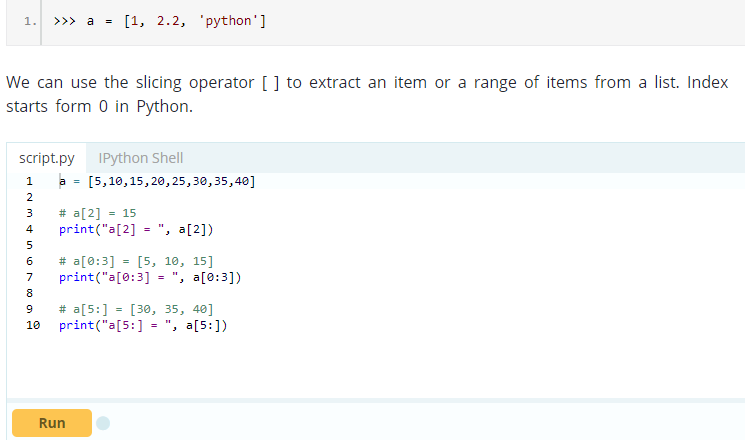
Complex numbers are written in the form, x + yj, where x is the real part and y is the imaginary part. Here are some examples.

### 

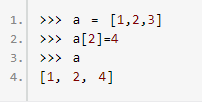
### Python List

[List](https://www.programiz.com/python-programming/list) is an ordered sequence of items. It is one of the most used datatype in Python and is very flexible. All the items in a list do not need to be of the same type.

Declaring a list is pretty straight forward. Items separated by commas are enclosed within brackets [ ].



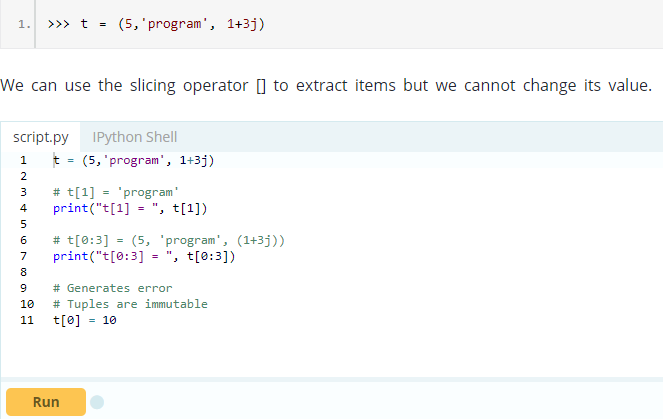
Lists are mutable, meaning, value of elements of a list can be altered.



### Python Tuple

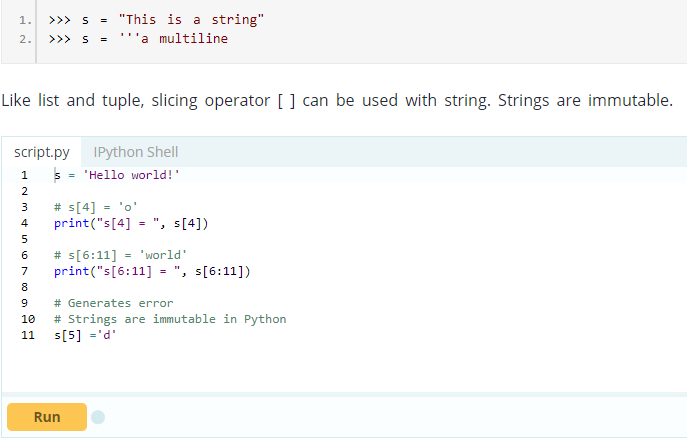
[Tuple](https://www.programiz.com/python-programming/tuple) is an ordered sequence of items same as list. The only difference is that tuples are immutable. Tuples once created cannot be modified. Tuples are used to write-protect data and are usually faster than list as it cannot change dynamically.

It is defined within parentheses () where items are separated by commas.



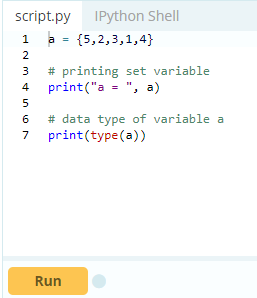
### Python Strings

[String](https://www.programiz.com/python-programming/string) is sequence of Unicode characters. We can use single quotes or double quotes to represent strings. Multi-line strings can be denoted using triple quotes, ''' or """.

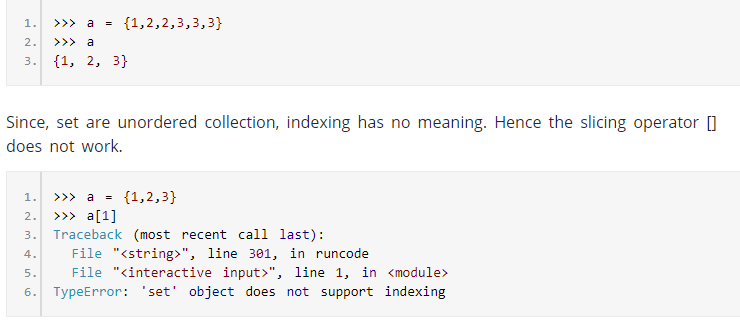


### Python Set

[Set](https://www.programiz.com/python-programming/set) is an unordered collection of unique items. Set is defined by values separated by comma inside braces { }. Items in a set are not ordered.

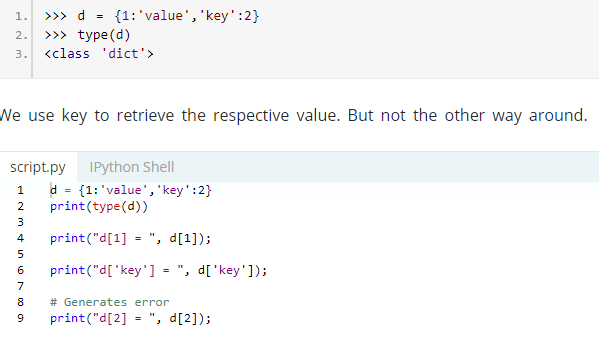


We can perform set operations like union, intersection on two sets. Set have unique values. They eliminate duplicates.



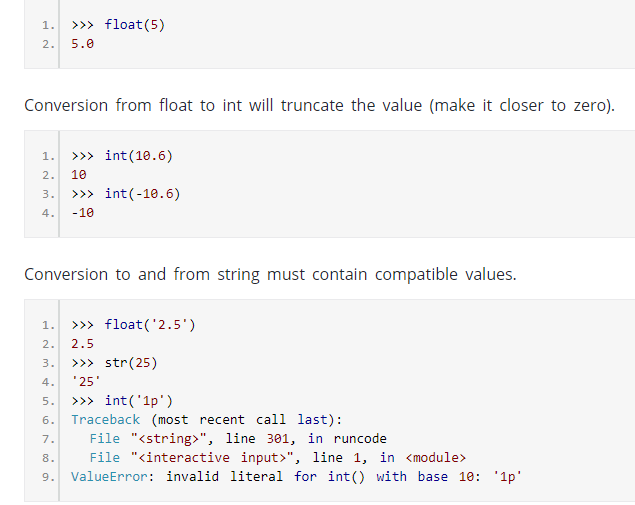
### Python Dictionary

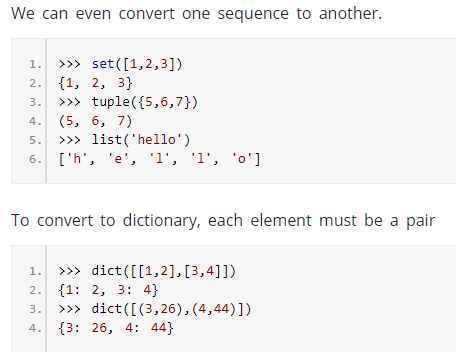
[Dictionary](https://www.programiz.com/python-programming/dictionary) is an unordered collection of key-value pairs. It is generally used when we have a huge amount of data. Dictionaries are optimized for retrieving data. We must know the key to retrieve the value. In Python, dictionaries are defined within braces {} with each item being a pair in the form key:value. Key and value can be of any type.



### Conversion between data types

We can convert between different data types by using different type conversion functions like int(), float(), str() etc.





# **Python Type Conversion and Type Casting**

Before learning Type Conversion in Python, you should have knowledge about [Python Data Types](https://www.programiz.com/python-programming/variables-datatypes).

## Type Conversion:

The process of converting the value of one data type (integer, string, float, etc.) to another data type is called type conversion. Python has two types of type conversion.

1. Implicit Type Conversion
2. Explicit Type Conversion

## Implicit Type Conversion:

In Implicit type conversion, Python automatically converts one data type to another data type. This process doesn't need any user involvement.

Let's see an example where Python promotes conversion of lower datatype (integer) to higher data type (float) to avoid data loss.

### Example 1: Converting integer to float

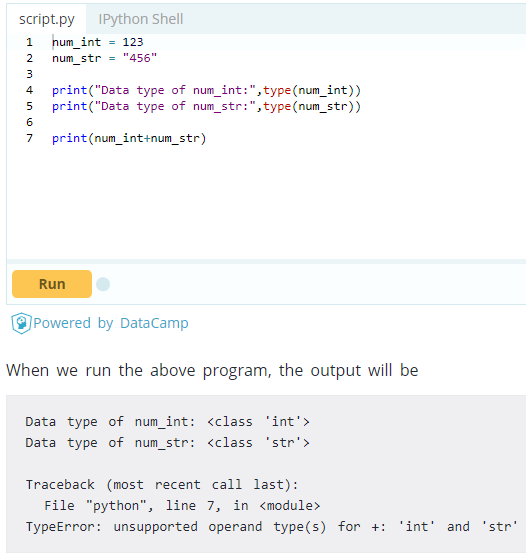
### 

In the above program,

* We add two variables num\_int and num\_flo, storing the value in num\_new.
* We will look at the data type of all three objects respectively.
* In the output we can see the datatype of num\_int is an integer, datatype of num\_flo is a float.
* Also, we can see the num\_new has float data type because Python always converts smaller data type to larger data type to avoid the loss of data.

Now, let's try adding a string and an integer, and see how Python treats it.

### Example 2: Addition of string(higher) data type and integer(lower) datatype



In the above program,

* We add two variable num\_int and num\_str.
* As we can see from the output, we got type error. Python is not able use Implicit Conversion in such condition.
* However Python has the solution for this type of situation which is know as Explicit Conversion.

## Explicit Type Conversion:

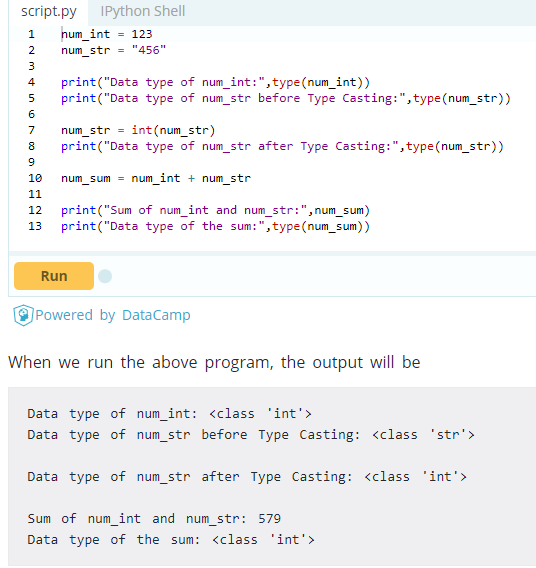
In Explicit Type Conversion, users convert the data type of an object to required data type. We use the predefined functions like int(), float(), str(), etc to perform explicit type conversion.

This type conversion is also called typecasting because the user casts (change) the data type of the objects.

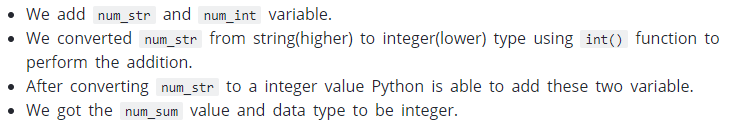
Syntax : (required\_datatype)(expression)

Typecasting can be done by assigning the required data type function to the expression

### Example 3: Addition of string and integer using explicit conversion



In above program,



## Key Points to Remember:

1. Type Conversion is the conversion of object from one data type to another data type.
2. Implicit Type Conversion is automatically performed by the Python interpreter.
3. Python avoids the loss of data in Implicit Type Conversion.
4. Explicit Type Conversion is also called Type Casting, the data types of object are converted using predefined function by user.
5. In Type Casting loss of data may occur as we enforce the object to specific data type.

# **Python Input, Output and Import**

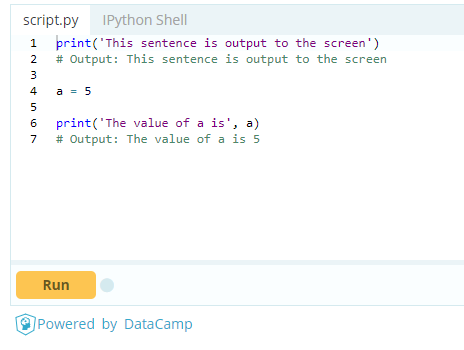
Python provides numerous [built-in functions](https://www.programiz.com/python-programming/built-in-function) that are readily available to us at the Python prompt.

Some of the functions like input() and print() are widely used for standard input and output operations respectively. Let us see the output section first.

## Python Output Using print() function

We use the print() function to output data to the standard output device (screen).

We can also [output data to a file](https://www.programiz.com/python-programming/file-operation), but this will be discussed later. An example use is given below.

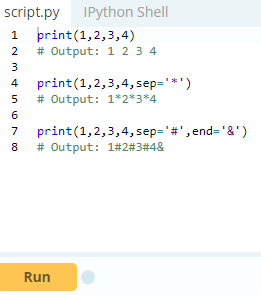


In the second print() statement, we can notice that a space was added between the [string](https://www.programiz.com/python-programming/string)and the value of variable a. This is by default, but we can change it.

The actual syntax of the print () function is

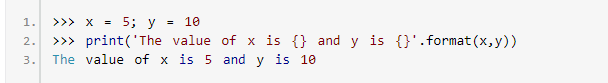


Here, objects is the value(s) to be printed. The sep separator is used between the values. It defaults into a space character. After all values are printed, end is printed. It defaults into a new line. The file is the object where the values are printed and its default value is sys.stdout(screen). Here are an example to illustrate this.



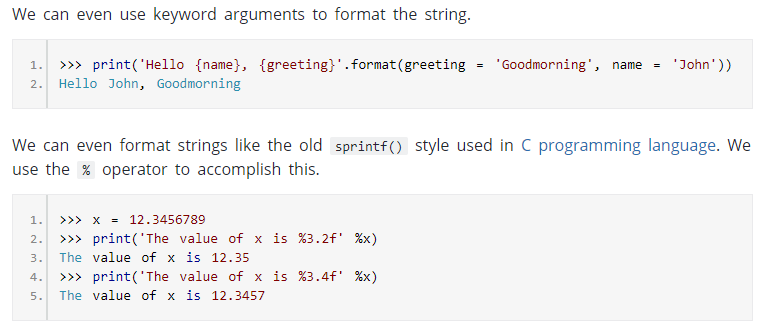
### Output formatting

Sometimes we would like to format our output to make it look attractive. This can be done by using the str.format() method. This method is visible to any string object.



Here the curly braces {} are used as placeholders. We can specify the order in which it is printed by using numbers (tuple index).



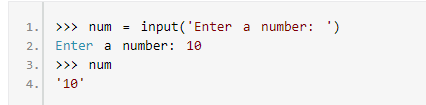


## Python Input

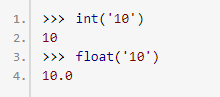
Up till now, our programs were static. The value of variables were defined or hard coded into the source code. To allow flexibility we might want to take the input from the user. In Python, we have the input() function to allow this. The syntax for input() is

input([prompt])

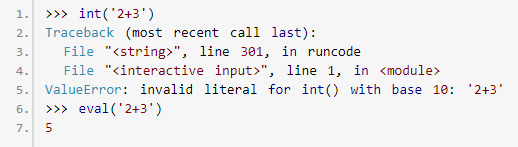
where prompt is the string we wish to display on the screen. It is optional.



Here, we can see that the entered value 10 is a string, not a number. To convert this into a number we can use int() or float() functions.



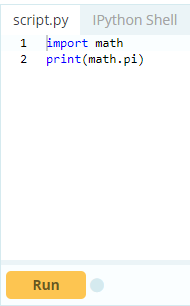
This same operation can be performed using the eval() function. But it takes it further. It can evaluate even expressions, provided the input is a string.



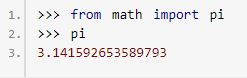
## Python Import

When our program grows bigger, it is a good idea to break it into different modules. A module is a file containing Python definitions and statements. [Python modules](https://www.programiz.com/python-programming/modules) have a filename and end with the extension .py.

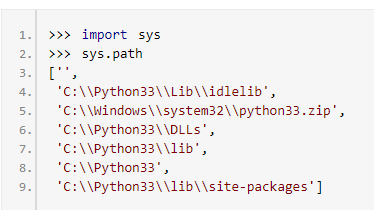
Definitions inside a module can be imported to another module or the interactive interpreter in Python. We use the import keyword to do this. For example, we can import the math module by typing in import math.



Now all the definitions inside math module are available in our scope. We can also import some specific attributes and functions only, using the from keyword. For example:



While importing a module, Python looks at several places defined in sys.path. It is a list of directory locations.



We can add our own location to this list as well.

# **Python Operators**

**What are operators in python?**

Operators are special symbols in Python that carry out arithmetic or logical computation. The value that the operator operates on is called the operand.

For example:

1. >>> 2+3
2. 5

Here, + is the operator that performs addition. 2 and 3 are the operands and 5 is the output of the operation.

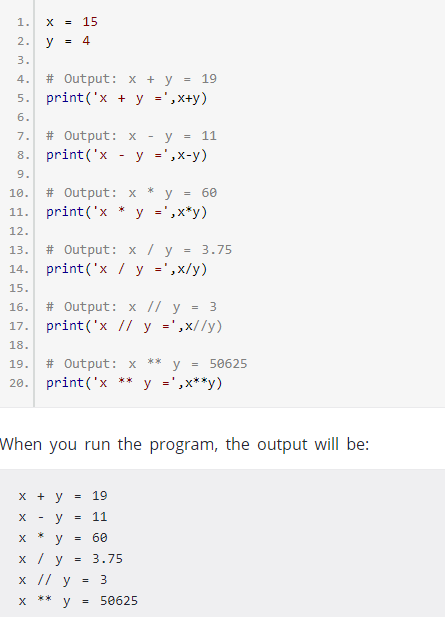
**Arithmetic operators**

Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication etc.

**Arithmetic operators in Python**

|  |  |  |
| --- | --- | --- |
| Operator | Meaning | Example |
| + | Add two operands or unary plus | x + y +2 |
| - | Subtract right operand from the left or unary minus | x - y -2 |
| \* | Multiply two operands | x \* y |
| / | Divide left operand by the right one (always results into float) | x / y |
| % | Modulus - remainder of the division of left operand by the right | x % y (remainder of x/y) |
| // | Floor division - division that results into whole number adjusted to the left in the number line | x // y |
| \*\* | Exponent - left operand raised to the power of right | x\*\*y (x to the power y) |
|  | | |

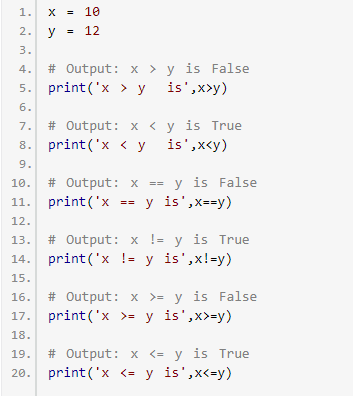
**Example #1: Arithmetic operators in Python**

****

**Comparison operators**

Comparison operators are used to compare values. It either returns True or False according to the condition.

|  |  |  |
| --- | --- | --- |
| Operator | Meaning | Example |
| > | Greater that - True if left operand is greater than the right | x > y |
| < | Less that - True if left operand is less than the right | x < y |
| == | Equal to - True if both operands are equal | x == y |
| != | Not equal to - True if operands are not equal | x != y |
| >= | Greater than or equal to - True if left operand is greater than or equal to the right | x >= y |
| <= | Less than or equal to - True if left operand is less than or equal to the right | x <= y |
|  | | |



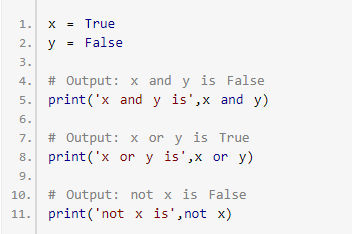
## 

## Logical operators

Logical operators are the and, or, not operators.

|  |  |  |
| --- | --- | --- |
| Operator | Meaning | Example |
| and | True if both the operands are true | x and y |
| or | True if either of the operands is true | x or y |
| not | True if operand is false (complements the operand) | not x |
| Logical operators in Python | | |

### Example #3: Logical Operators in Python



Here is the [truth table](https://www.programiz.com/python-programming/keyword-list#and_or_not) for these operators.

## Bitwise operators

Bitwise operators act on operands as if they were string of binary digits. It operates bit by bit, hence the name.

For example, 2 is 10 in binary and 7 is 111.

**In the table below:** Let x = 10 (0000 1010 in binary) and y = 4 (0000 0100 in binary)

**Bitwise operators in Python**

|  |  |  |
| --- | --- | --- |
| Operator | Meaning | Example |
| & | Bitwise AND | x& y = 0 (0000 0000) |
| | | Bitwise OR | x | y = 14 (0000 1110) |
| ~ | Bitwise NOT | ~x = -11 (1111 0101) |
| ^ | Bitwise XOR | x ^ y = 14 (0000 1110) |
| >> | Bitwise right shift | x>> 2 = 2 (0000 0010) |
| << | Bitwise left shift | x<< 2 = 40 (0010 1000) |
|  | | |

## Assignment operators

Assignment operators are used in Python to assign values to variables.

a = 5 is a simple assignment operator that assigns the value 5 on the right to the variable a on the left.

There are various compound operators in Python like a += 5 that adds to the variable and later assigns the same. It is equivalent to a = a + 5.

**Assignment operators in Python**

|  |  |  |
| --- | --- | --- |
| Operator | Example | Equivatent to |
| = | x = 5 | x = 5 |
| += | x += 5 | x = x + 5 |
| -= | x -= 5 | x = x – 5 |
| \*= | x \*= 5 | x = x \* 5 |
| /= | x /= 5 | x = x / 5 |
| %= | x %= 5 | x = x % 5 |
| //= | x //= 5 | x = x // 5 |
| \*\*= | x \*\*= 5 | x = x \*\* 5 |
| &= | x &= 5 | x = x & 5 |
| |= | x |= 5 | x = x | 5 |
| ^= | x ^= 5 | x = x ^ 5 |
| >>= | x >>= 5 | x = x >> 5 |
| <<= | x <<= 5 | x = x << 5 |
|  | | |

**Special operators**

Python language offers some special type of operators like the identity operator or the membership operator. They are described below with examples.

### Identity operators

is and is not are the identity operators in Python. They are used to check if two values (or variables) are located on the same part of the memory. Two variables that are equal does not imply that they are identical.

|  |  |  |
| --- | --- | --- |
| Operator | Meaning | Example |
| Is | True if the operands are identical (refer to the same object) | x is True |
| is not | True if the operands are not identical (do not refer to the same object) | x is not True |
| Identity operators in Python | | |

### Example #4: Identity operators in Python

### 

Here, we see that x1 and y1 are integers of same values, so they are equal as well as identical. Same is the case with x2 and y2 (strings).

But x3 and y3 are list. They are equal but not identical. It is because interpreter locates them separately in memory although they are equal.

### Membership operators

in and not in are the membership operators in Python. They are used to test whether a value or variable is found in a sequence ([string](https://www.programiz.com/python-programming/string), [list](https://www.programiz.com/python-programming/list), [tuple](https://www.programiz.com/python-programming/tuple), [set](https://www.programiz.com/python-programming/set) and [dictionary](https://www.programiz.com/python-programming/dictionary)). In a dictionary we can only test for presence of key, not the value.

|  |  |  |
| --- | --- | --- |
| Operator | Meaning | Example |
| In | True if value/variable is found in the sequence | 5 in x |
| not in | True if value/variable is not found in the sequence | 5 not in x |

### Example #5: Membership operators in Python

### 

Here, 'H' is in x but 'hello' is not present in x (remember, Python is case sensitive). Similary, 1 is key and 'a' is the value in dictionary y. Hence, 'a' in y returns False.

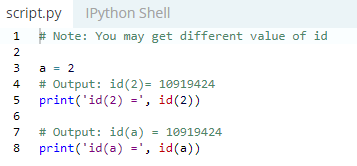
# **Python Namespace and Scope**

## What is Name in Python?

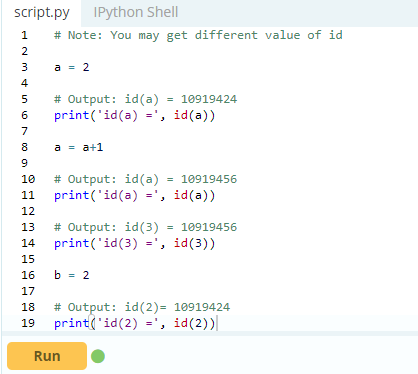
If you have ever read 'The Zen of Python' (type "import this" in Python interpreter), the last line states, **Namespaces are one honking great idea -- let's do more of those!** So what are these mysterious namespaces? Let us first look at what name is.

Name (also called identifier) is simply a name given to objects. Everything in Python is an [object](https://www.programiz.com/python-programming/class). Name is a way to access the underlying object.

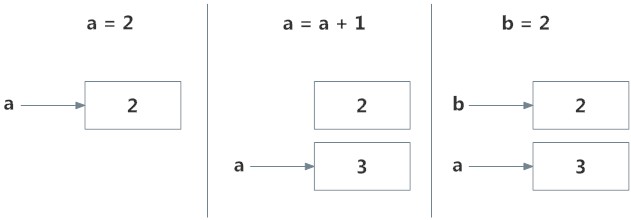
For example, when we do the assignment a = 2, here 2 is an object stored in memory and a is the name we associate it with. We can get the address (in RAM) of some object through the [built-in function](https://www.programiz.com/python-programming/built-in-function), id(). Let's check it.



Here, both refer to the same object. Let's make things a little more interesting.



What is happening in the above sequence of steps? A diagram will help us explain this.

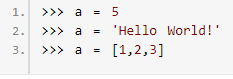


Initially, an object 2 is created and the name a is associated with it, when we do a = a+1, a new object 3 is created and now a associates with this object.

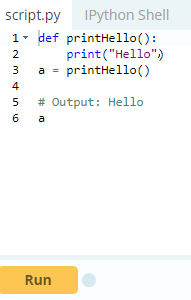
Note that id(a) and id(3) have same values.

Furthermore, when we do b = 2, the new name b gets associated with the previous object 2.

This is efficient as Python doesn't have to create a new duplicate object. This dynamic nature of name binding makes Python powerful; a name could refer to any type of object.



All these are valid and a will refer to three different types of object at different instances. [Functions](https://www.programiz.com/python-programming/function) are objects too, so a name can refer to them as well.



Our same name a can refer to a function and we can call the function through it, pretty neat.

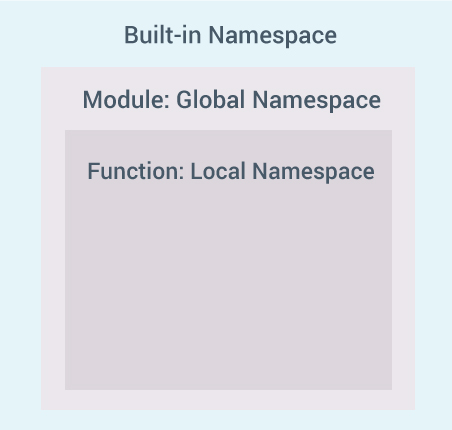
## What is a Namespace in Python?

So now that we understand what names are, we can move on to the concept of namespaces. To simply put it, namespace is a collection of names. In Python, you can imagine a namespace as a mapping of every name, you have defined, to corresponding objects.

Different namespaces can co-exist at a given time but are completely isolated. A namespace containing all the built-in names is created when we start the Python interpreter and exists as long we don't exit.

This is the reason that built-in functions like id(), print() etc. are always available to us from any part of the program. Each [module](https://www.programiz.com/python-programming/modules) creates its own global namespace. These different namespaces are isolated. Hence, the same name that may exist in different modules do not collide.

Modules can have various functions and classes. A local namespace is created when a function is called, which has all the names defined in it. Similar, is the case with class. Following diagram may help to clarify this concept.



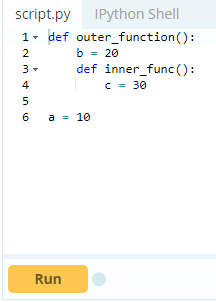
## Python Variable Scope

Although there are various unique namespaces defined, we may not be able to access all of them from every part of the program. The concept of scope comes into play. Scope is the portion of the program from where a namespace can be accessed directly without any prefix. At any given moment, there are at least three nested scopes.

1. Scope of the current function which has local names
2. Scope of the module which has global names
3. Outermost scope which has built-in names

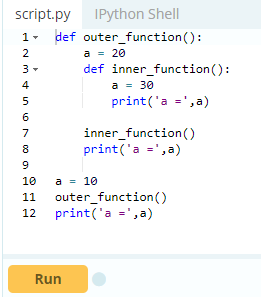
When a reference is made inside a function, the name is searched in the local namespace, then in the global namespace and finally in the built-in namespace. If there is a function inside another function, a new scope is nested inside the local scope.

## Example of Scope and Namespace in Python



Here, the variable a is in the global namespace. Variable b is in the local namespace of outer\_function() and c is in the nested local namespace of inner\_function(). When we are in inner\_function(), c is local to us, b is nonlocal and a is global. We can read as well as assign new values to c but can only read b and a from inner\_function().

If we try to assign as a value to b, a new variable b is created in the local namespace which is different than the nonlocal b. Same thing happens when we assign a value to a. However, if we declare a as global, all the reference and assignment go to the global a. Similarly, if we want to rebind the variable b, it must be declared as nonlocal. The following example will further clarify this.



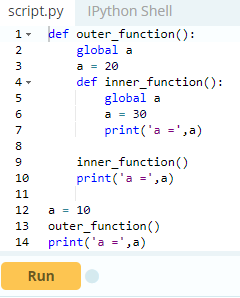
As you can see, the output of this program is

a = 30

a = 20

a = 10

In this program, three different variables a are defined in separate namespaces and accessed accordingly. While in the following program,



The output of the program is.

a = 30

a = 30

a = 30

Here, all reference and assignment are to the global a due to the use of keyword global.