Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python’s elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

The Python interpreter and the extensive standard library are freely available in source or binary form for all major platforms from the Python Web site, <https://www.python.org/>, and may be freely distributed. The same site also contains distributions of and pointers to many free third party Python modules, programs and tools, and additional documentation.

The Python interpreter is easily extended with new functions and data types implemented in C or C++ (or other languages callable from C). Python is also suitable as an extension language for customizable applications.

Python is a powerful high-level, object-oriented programming language created by Guido van Rossum.

It has simple easy-to-use syntax, making it the perfect language for someone trying to learn computer programming for the first time.

**Python Keywords**

Keywords are the reserved words in Python.

In Python, keywords are case sensitive.

There are 33 keywords in Python 3.3. This number can vary slightly in 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 are given below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Keywords in Python programming language | | | | |
| 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 |  |

# List of Keywords in Python

>>> import keyword

>>> print(keyword.kwlist)

## Description of Keywords in Python with examples

### True, False

True and False are truth values in Python. They are the results of comparison operations or logical (Boolean) operations in Python. For example:

>>> 1 == 1

True

>>> 5 > 3

True

>>> True or False

True

>>> 10 <= 1

False

>>> 3 > 7

False

>>> True and False

False

Here we can see that the first three statements are true so the interpreter returns True and returns False for the remaining three statements. True and False in python is same as 1and 0. This can be justified with the following example:

>>> True == 1

True

>>> False == 0

True

>>> True + True

2

### None

None is a special constant in Python that represents the absence of a value or a null value.

It is an object of its own datatype, the NoneType. We cannot create multiple None objects but can assign it to variables. These variables will be equal to one another.

We must take special care that None does not imply False, 0 or any empty list, dictionary, string etc. For example:

>>> None == 0

False

>>> None == []

False

>>> None == False

False

>>> x = None

>>> y = None

>>> x == y

True

Void functions that do not return anything will return a None object automatically. None is also returned by functions in which the program flow does not encounter a return statement. For example:

def a\_void\_function():

a = 1

b = 2

c = a + b

x = a\_void\_function()

print(x)

**Output**

None

This program has a function that does not return a value, although it does some operations inside. So when we print x, we get None which is returned automatically (implicitly). Similarly, here is another example:

def improper\_return\_function(a):

if (a % 2) == 0:

return True

x = improper\_return\_function(3)

print(x)

**Output**

None

Although this function has a return statement, it is not reached in every case. The function will return True only when the input is even.

If we give the function an odd number, None is returned implicitly.

### and, or , not

and, or, not are the logical operators in Python. and will result into True only if both the operands are True. The truth table for and is given below:

|  |  |  |
| --- | --- | --- |
| Truth table for and | | |
| A | B | A and B |
| True | True | True |
| True | False | False |
| False | True | False |
| False | False | False |

or will result into True if any of the operands is True. The truth table for or is given below:

|  |  |  |
| --- | --- | --- |
| Truth table for or | | |
| A | B | A or B |
| True | True | True |
| True | False | True |
| False | True | True |
| False | False | False |

not operator is used to invert the truth value. The truth table for not is given below:

|  |  |
| --- | --- |
| Truth tabel for not | |
| A | not A |
| True | False |
| False | True |

some example of their usage are given below

>>> True and False

False

>>> True or False

True

>>> not False

True

### as

as is used to create an alias while importing a module. It means giving a different name (user-defined) to a module while importing it.

As for example, Python has a standard module called math. Suppose we want to calculate what cosine pi is using an alias. We can do it as follows using as:

>>> import math as myAlias

>>>myAlias.cos(myAlias.pi)

-1.0

Here we imported the math module by giving it the name myAlias. Now we can refer to the math module with this name. Using this name we calculated cos(pi) and got -1.0 as the answer.

### assert

assert is used for debugging purposes.

While programming, sometimes we wish to know the internal state or check if our assumptions are true. assert helps us do this and find bugs more conveniently. assert is followed by a condition.

If the condition is true, nothing happens. But if the condition is false, AssertionError is raised. For example:

>>> a = 4

>>> assert a < 5

>>> assert a > 5

Traceback (most recent call last):

File "<string>", line 301, in runcode

File "<interactive input>", line 1, in <module>

AssertionError

For our better understanding, we can also provide a message to be printed with the AssertionError.

>>> a = 4

>>> assert a > 5, "The value of a is too small"

Traceback (most recent call last):

File "<string>", line 301, in runcode

File "<interactive input>", line 1, in <module>

AssertionError: The value of a is too small

At this point we can note that,

assert condition, message

is equivalent to,

if not condition:

raise AssertionError(message)

### break, continue

break and continue are used inside for and while loops to alter their normal behavior.

break will end the smallest loop it is in and control flows to the statement immediately below the loop. continue causes to end the current iteration of the loop, but not the whole loop.

This can be illustrated with the following two examples:

for i in range(1,11):

if i == 5:

break

print(i)

**Output**

1

2

3

4

Here, the for loop intends to print numbers from 1 to 10. But the if condition is met when i is equal to 5 and we break from the loop. Thus, only the range 1 to 4 is printed.

for i in range(1,11):

if i == 5:

continue

print(i)

**Output**

1

2

3

4

6

7

8

9

10

Here we use continue for the same program. So, when the condition is met, that iteration is skipped. But we do not exit the loop. Hence, all the values except 5 is printed out.

Learn more about [Python break and continue statement](https://www.programiz.com/python-programming/break-continue).

### class

class is used to define a new user-defined class in Python.

Class is a collection of related attributes and methods that try to represent a real world situation. This idea of putting data and functions together in a class is central to the concept of object-oriented programming (OOP).

Classes can be defined anywhere in a program. But it is a good practice to define a single class in a module. Following is a sample usage:

class ExampleClass:

def function1(parameters):

…

def function2(parameters):

…

Learn more about [Python Objects and Class](https://www.programiz.com/python-programming/class).

### def

def is used to define a user-defined function.

Function is a block of related statements, which together does some specific task. It helps us organize code into manageable chunks and also to do some repetitive task.

The usage of def is shown below:

def function\_name(parameters):

…

Learn more about [Python functions](https://www.programiz.com/python-programming/function).

### del

del is used to delete the reference to an object. Everything is object in Python. We can delete a variable reference using del

>>> a = b = 5

>>> del a

>>> a

Traceback (most recent call last):

File "<string>", line 301, in runcode

File "<interactive input>", line 1, in <module>

NameError: name 'a' is not defined

>>> b

5

Here we can see that the reference of the variable a was deleted. So, it is no longer defined. But b still exists.

del is also used to delete items from a list or a dictionary:

>>> a = ['-x','y','z']

>>> del a[1]

>>> a

['x', 'z']

### if, else, elif

if, else, elif are used for conditional branching or decision making.

When we want to test some condition and execute a block only if the condition is true, then we use if and elif. elif is short for else if. else is the block which is executed if the condition is false. This will be clear with the following example:

def if\_example(a):

if a == 1:

print('One')

elif a == 2:

print('Two')

else:

print('Something else')

if\_example(2)

if\_example(4)

if\_example(1)

**Output**

Two

Something else

One

Here, the function checks the input number and prints the result if it is 1 or 2. Any input other than this will cause the else part of the code to execute.

Learn more about [Python if and if...else Statement](https://www.programiz.com/python-programming/if-elif-else).

### except, raise, try

except, raise, try are used with exceptions in Python.

Exceptions are basically errors that suggests something went wrong while executing our program. IOError, ValueError, ZeroDivisionError, ImportError, NameError, TypeError etc. are few examples of exception in Python. try...except blocks are used to catch exceptions in Python.

We can raise an exception explicitly with the raise keyword. Following is an example:

def reciprocal(num):

try:

r = 1/num

except:

print('Exception caught')

return

return r

print(reciprocal(10))

print(reciprocal(0))

**Output**

0.1

Exception caught

None

Here, the function reciprocal() returns the reciprocal of the input number.

When we enter 10, we get the normal output of 0.1. But when we input 0, a ZeroDivisionError is raised automatically.

This is caught by our try…except block and we return None. We could have also raised the ZeroDivisionError explicitly by checking the input and handled it elsewhere as follows:

if num == 0:

raise ZeroDivisionError('cannot divide')

### finally

finally is used with try…except block to close up resources or file streams.

Using finally ensures that the block of code inside it gets executed even if there is an unhandled exception. For example:

try:

Try-block

except exception1:

Exception1-block

except exception2:

Exception2-block

else:

Else-block

finally:

Finally-block

Here if there is an exception in the Try-block, it is handled in the except or else block. But no matter in what order the execution flows, we can rest assured that the Finally-blockis executed even if there is an error. This is useful in cleaning up the resources.

Learn more about [exception handling in Python programming](https://www.programiz.com/python-programming/exception-handling).

### for

for is used for looping. Generally we use for when we know the number of times we want to loop.

In Python we can use it with any type of sequence like a list or a string. Here is an example in which for is used to traverse through a list of names:

names = ['John','Monica','Steven','Robin']

for i in names:

print('Hello '+i)

**Output**

Hello John

Hello Monica

Hello Steven

Hello Robin

Learn more about [Python for loop](https://www.programiz.com/python-programming/for-loop).

### from, import

import keyword is used to import modules into the current namespace. from…import is used to import specific attributes or functions into the current namespace. For example:

import math

will import the math module. Now we can use the cos() function inside it as math.cos(). But if we wanted to import just the cos() function, this can done using from as

from math import cos

now we can use the function simply as cos(), no need to write math.cos().

Learn more on [Python modules and import statement](https://www.programiz.com/python-programming/modules).

### global

global is used to declare that a variable inside the function is global (outside the function).

If we need to read the value of a global variable, it is not necessary to define it as global. This is understood.

If we need to modify the value of a global variable inside a function, then we must declare it with global. Otherwise a local variable with that name is created.

Following example will help us clarify this.

globvar = 10

def read1():

print(globvar)

def write1():

global globvar

globvar = 5

def write2():

globvar = 15

read1()

write1()

read1()

write2()

read1()

**Output**

10

5

5

Here, the read1() function is just reading the value of globvar. So, we do not need to declare it as global. But the write1() function is modifying the value, so we need to declare the variable as global.

We can see in our output that the modification did take place (10 is changed to 5). The write2() also tries to modify this value. But we have not declared it as global.

Hence, a new local variable globvar is created which is not visible outside this function. Although we modify this local variable to 15, the global variable remains unchanged. This is clearly visible in our output.

### in

in is used to test if a sequence (list, tuple, string etc.) contains a value. It returns True if the value is present, else it returns False. For example:

>>> a = [1, 2, 3, 4, 5]

>>> 5 in a

True

>>> 10 in a

False

The secondary use of in is to traverse through a sequence in a for loop.

for i in 'hello':

print(i)

**Output**

h

e

l

l

o

### is

is is used in Python for testing object identity. While the == operator is used to test if two variables are equal or not, is is used to test if the two variables refer to the same object.

It returns True if the objects are identical and False if not.

>>> True is True

True

>>> False is False

True

>>> None is None

True

We know that there is only one instance of True, False and None in Python, so they are identical.

>>> [] == []

True

>>> [] is []

False

>>> {} == {}

True

>>> {} is {}

False

An empty list or dictionary is equal to another empty one. But they are not identical objects as they are located separately in memory. This is because list and dictionary are mutable (value can be changed).

>>> '' == ''

True

>>> '' is ''

True

>>> () == ()

True

>>> () is ()

True

Unlike list and dictionary, string and tuple are immutable (value cannot be altered once defined). Hence, two equal string or tuple are identical as well. They refer to the same memory location.

### lambda

lambda is used to create an anonymous function (function with no name). It is an inline function that does not contain a return statement. It consists of an expression that is evaluated and returned. For example:

a = lambda x: x\*2

for i in range(1,6):

print(a(i))

**Output**

2

4

6

8

10

Here, we have created an inline function that doubles the value, using the lambda statement. We used this to double the values in a list containing 1 to 5.

Learn more about [Python lamda function](https://www.programiz.com/python-programming/anonymous-function).

### nonlocal

The use of nonlocal keyword is very much similar to the global keyword. nonlocal is used to declare that a variable inside a nested function (function inside a function) is not local to it, meaning it lies in the outer inclosing function. If we need to modify the value of a non-local variable inside a nested function, then we must declare it with nonlocal. Otherwise a local variable with that name is created inside the nested function. Following example will help us clarify this.

def outer\_function():

a = 5

def inner\_function():

nonlocal a

a = 10

print("Inner function: ",a)

inner\_function()

print("Outer function: ",a)

outer\_function()

**Output**

Inner function: 10

Outer function: 10

Here, the inner\_function() is nested within the outer\_function.

The variable a is in the outer\_function(). So, if we want to modify it in the inner\_function(), we must declare it as nonlocal. Notice that a is not a global variable.

Hence, we see from the output that the variable was successfully modified inside the nested inner\_function(). The result of not using the nonlocal keyword is as follows:

def outer\_function():

a = 5

def inner\_function():

a = 10

print("Inner function: ",a)

inner\_function()

print("Outer function: ",a)

outer\_function()

**Output**

Inner function: 10

Outer function: 5

Here, we do not declare that the variable a inside the nested function is nonlocal. Hence, a new local variable with the same name is created, but the non-local a is not modified as seen in our output.

### pass

pass is a null statement in Python. Nothing happens when it is executed. It is used as a placeholder.

Suppose we have a function that is not implemented yet, but we want to implement it in the future. Simply writing,

def function(args):

in the middle of a program will give us IndentationError. Instead of this, we construct a blank body with the pass statement.

def function(args):

pass

We can do the same thing in an empty class as well.

class example:

pass

### return

return statement is used inside a function to exit it and return a value.

If we do not return a value explicitly, None is returned automatically. This is verified with the following example.

def func\_return():

a = 10

return a

def no\_return():

a = 10

print(func\_return())

print(no\_return())

**Output**

10

None

### while

while is used for looping in Python.

The statements inside a while loop continue to execute until the condition for the whileloop evaluates to False or a break statement is encountered. Following program illustrates this.

i = 5

while(i):

print(i)

i = i – 1

**Output**

5

4

3

2

1

Note that 0 is equal to False.

Learn more about [Python while loop](https://www.programiz.com/python-programming/while-loop).

### with

with statement is used to wrap the execution of a block of code within methods defined by the context manager.

Context manager is a class that implements \_\_enter\_\_ and \_\_exit\_\_ methods. Use of withstatement ensures that the \_\_exit\_\_ method is called at the end of the nested block. This concept is similar to the use of try…finally block. Here, is an example.

with open('example.txt', 'w') as my\_file:

my\_file.write('Hello world!')

This example writes the text Hello world! to the file example.txt. File objects have \_\_enter\_\_ and \_\_exit\_\_ method defined within them, so they act as their own context manager.

First the \_\_enter\_\_ method is called, then the code within with statement is executed and finally the \_\_exit\_\_ method is called. \_\_exit\_\_ method is called even if there is an error. It basically closes the file stream.

### yield

yield is used inside a function like a return statement. But yield returns a generator.

Generator is an iterator that generates one item at a time. A large list of value will take up a lot of memory. Generators are useful in this situation as it generates only one value at a time instead of storing all the values in memory. For example,

>>> g = (2\*\*x for x in range(100))

will create a generator g which generates powers of 2 up to the number two raised to the power 99. We can generate the numbers using the next() function as shown below.

>>> next(g)

1

>>> next(g)

2

>>> next(g)

4

>>> next(g)

8

>>> next(g)

16

And so on… This type of generator is returned by the yield statement from a function. Here is an example.

def generator():

for i in range(6):

yield i\*i

g = generator()

for i in g:

print(i)

**Output**

0

1

4

9

16

25

Here, the function generator() returns a generator that generates square of numbers from 0 to 5. This is printed in the for loop.

## Python Identifiers

Identifier is the name given to entities like class, functions, variables etc. in Python. It helps differentiating one entity from another.

### Rules for writing identifiers

1. 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.
2. An identifier cannot start with a digit. 1variable is invalid, but variable1 is perfectly fine.
3. Keywords cannot be used as identifiers.
4. >>> global = 1
5. File "<interactive input>", line 1
6. global = 1
7. ^

SyntaxError: invalid syntax

1. We cannot use special symbols like !, @, #, $, % etc. in our identifier.
2. >>> a@ = 0
3. File "<interactive input>", line 1
4. a@ = 0
5. ^

SyntaxError: invalid syntax

1. Identifier can be of any length.

### Things to care about

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.

We can also use camel-case style of writing, i.e., capitalize every first letter of the word except the initial word without any spaces. For example: camelCaseExample

## 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:

a = 1 + 2 + 3 + \

4 + 5 + 6 + \

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

a = (1 + 2 + 3 +

4 + 5 + 6 +

7 + 8 + 9)

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

colors = ['red',

'blue',

'green']

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

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 unindented 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.

for i in range(1,11):

print(i)

if i == 5:

break

## 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.

#This is a comment

#print out Hello

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:

#This is a long comment

#and it extends

#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.

"""This is also a

perfect example of

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:

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

>>> print(double.\_\_doc\_\_)

Function to double the value

## Variable

In most of the programming languages a variable is a named location used to store data in the memory. Each variable must have a unique name called identifier. It is helpful to think of variables as container that hold data which can be changed later throughout programming.

Non technically, you can suppose variable as a bag to store books in it and those books can be replaced at anytime.

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

### Declaring Variables in Python

In Python, variables do not need declaration to reserve memory space. The "variable declaration" or "variable initialization" happens automatically when we assign a value to a variable.

### Assigning value to a Variable in Python

You can use the assignment operator = to assign the value to a variable.

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

**website = "Apple.com"**

**print(website)**

**\**

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 infer (know) Apple.com is a String and declare website as a String.

#### Example 2 : Changing value of a variable

**website = "Apple.com"**

**# assigning a new variable to website**

**website = "Programiz.com"**

**print(website)**

When you run the program, the output will be:

Programiz.com

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

a, b, c = 5, 3.2, "Hello"

print (a)

print (b)

print (c)

## 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

Create a constant.py

PI = 3.14

GRAVITY = 9.8

Create a main.py

import constant

print(constant.PI)

print(constant.GRAVITY)

When you run the program, the output will be:

3.14

9.8

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 camelCase notation to declare a variable. It starts with lowercase letter. For example:
3. myName
4. myAge

myAddress

1. Use capital letters where possible to declare a constant. For example:
2. PI
3. G
4. MASS

TEMP

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:
5. snake\_case
6. MACRO\_CASE
7. camelCase

CapWords

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?

a = 0b1010 #Binary Literals

b = 100 #Decimal LLiteral 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: Hoiteral

c = 0o310 #Octal Literal

d = 0x12c #Hexadecimal Literal

#Float Literal

float\_1 = 10.5

float\_2 = 1.5e2

#Complex Literal

x = 3.14j

print(a, b, c, d)

print(float\_1, float\_2)

print(x, x.imag, x.real)

[Powered by DataCamp](https://www.datacamp.com/)

When you run the program, the output will be:

This is Python

C

This is a multiline string with more than one line code.

Ünicöde

raw \n string

When you run the program, the output will be:

10 100 200 300

10.5 150.0

3.14j 3.14 0.0

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?

strings = "This is Python"

char = "C"

multiline\_str = """This is a multiline string with more than one line code."""

unicode = u"\u00dcnic\u00f6de"

raw\_str = r"raw \n string"

print(strings)

print(char)

print(multiline\_str)

print(unicode)

print(raw\_str)

When you run the program, the output will be:

This is Python

C

This is a multiline string with more than one line code.

Ünicöde

raw \n string

n 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?

x = (1 == True)

y = (1 == False)

a = True + 4

b = False + 10

print("x is", x)

print("y is", y)

print("a:", a)

print("b:", b)

When you run the program, the output will be:

x is True

y is False

a: 5

b: 10

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?

drink = "Available"

food = None

def menu(x):

if x == drink:

print(drink)

else:

print(food)

menu(drink)

menu(food)

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?

fruits = ["apple", "mango", "orange"] #list

numbers = (1, 2, 3) #tuple

alphabets = {'a':'apple', 'b':'ball', 'c':'cat'} #dictionary

vowels = {'a', 'e', 'i' , 'o', 'u'} #set

print(fruits)

print(numbers)

print(alphabets)

print(vowels)

When you run the program, the output will be:

['apple', 'mango', 'orange']

(1, 2, 3)

{'a': 'apple', 'b': 'ball', 'c': 'cat'}

{'e', 'a', 'o', 'i', 'u'}

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

.