# **Python Variables, Constants, Literals, Comments, and Data types**

## **Python Variables**

In programming, a variable is a container (storage area) to hold data. For example,

number = 10

Here, number is the variable storing the value 10.

## **Assigning values to Variables in Python**

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

# assign value to site\_name variable

site\_name = 'programiz.pro'

print(site\_name)

# Output: programiz.pro

In the above example, we assigned the value 'programiz.pro' to the site\_name variable. Then, we printed out the value assigned to site\_name.

Note: Python is a [type-inferred](https://en.wikipedia.org/wiki/Type_inference) language, so you don't have to explicitly define the variable type. It automatically knows that programiz.pro is a string and declares the site\_name variable as a string.

## **Changing the Value of a Variable in Python**

site\_name = 'programiz.pro'

print(site\_name)

# assigning a new value to site\_name

site\_name = 'apple.com'

print(site\_name)

Output

programiz.pro

apple.com

Here, the value of site\_name is changed from 'programiz.pro' to 'apple.com'.

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

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

print(a) # prints 5

print(b) # prints 3.2

print(c) # prints Hello

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

site1 = site2 = 'programiz.com'

print(site1) # prints programiz.com

print(site2) # prints programiz.com

Here, we have assigned the same string value 'programiz.com' to both the variables site1 and site2.

## **Rules for Naming Python Variables**

* Constant and variable names should have a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (\_). For example:

snake\_case

MACRO\_CASE

camelCase

CapWords

* Create a name that makes sense. For example, vowel makes more sense than v.
* If you want to create a variable name having two words, use underscore to separate them. For example:

my\_name

current\_salary

* Python is case-sensitive. So num and Num are different variables. For example,

var num = 5

var Num = 55

print(num) # 5

print(Num) # 55

* Avoid using [keywords](https://www.programiz.com/python-programming/keywords-identifier) like if, True, class, etc. as variable names.

## **Python Constants**

A constant is a special type of variable whose value cannot be changed.

In Python, constants are usually declared and assigned in a [module](https://www.programiz.com/python-programming/modules) (a new file containing variables, [functions](https://www.programiz.com/python-programming/function), etc which is imported to the main file).

Let's see how we declare constants in separate file and use it in the main file,

Create a constant.py:

# declare constants

PI = 3.14

GRAVITY = 9.8

Create a main.py:

# import constant file we created above

import constant

print(constant.PI) # prints 3.14

print(constant.GRAVITY) # prints 9.8

In the above example, we created the constant.py module file. Then, we assigned the constant value to PI and GRAVITY.

After that, we create the main.py file and import the constant module. Finally, we printed the constant value.

Note: In reality, we don't use constants in Python. Naming them in all capital letters is a convention to separate them from variables, however, it does not actually prevent reassignment.

## **Python Literals**

Literals are representations of fixed values in a program. They can be numbers, characters, or strings, etc. For example, 'Hello, World!', 12, 23.0, 'C', etc.

Literals are often used to assign values to variables or constants. For example,

site\_name = 'programiz.com'

In the above expression, site\_name is a variable, and 'programiz.com' is a literal.

## **Python Numeric Literals**

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

| Type | Example | Remarks |
| --- | --- | --- |
| Decimal | 5, 10, -68 | Regular numbers. |
| Binary | 0b101, 0b11 | Start with 0b. |
| Octal | 0o13 | Start with 0o. |
| Hexadecimal | 0x13 | Start with 0x. |
| Floating-point Literal | 10.5, 3.14 | Containing floating decimal points. |
| Complex Literal | 6 + 9j | Numerals in the form a + bj, where a is real and b is imaginary part |

## **Python Boolean Literals**

There are two boolean literals: True and False.

For example,

result1 = True

Here, True is a boolean literal assigned to result1.

## **String and Character Literals in Python**

Character literals are unicode characters enclosed in a quote. For example,

some\_character = 'S'

Here, S is a character literal assigned to some\_character.

Similarly, String literals are sequences of Characters enclosed in quotation marks.

For example,

some\_string = 'Python is fun'

Here, 'Python is fun' is a string literal assigned to some\_string.

## **Special Literal in Python**

Python contains one special literal None. We use it to specify a null variable. For example,

value = None

print(value)

# Output: None

Here, we get None as an output as the value variable has no value assigned to it.

## **Literal Collections**

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

# list literal

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

print(fruits)

# tuple literal

numbers = (1, 2, 3)

print(numbers)

# dictionary literal

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

print(alphabets)

# set literal

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

print(vowels)

Output

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

(1, 2, 3)

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

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

In the above example, we created a:

* [list](https://www.programiz.com/python-programming/list) of fruits
* [tuple](https://www.programiz.com/python-programming/tuple) of numbers
* [dictionary](https://www.programiz.com/python-programming/dictionary) of alphabets having values with keys designated to each value
* [set](https://www.programiz.com/python-programming/set) of vowels

To learn more about literal collections, refer to [Python Data Types](https://www.programiz.com/python-programming/variables-datatypes).

# **Python Comments**

Comments are hints that we add to our code to make it easier to understand.

When executing code, Python's interpreter ignores comments.

For example, we have a program to print a text entered by the user.

name = input("Enter your name:")

print(name)

To make this program more readable, we can add comments like:

# Program to take the user's name

name = input('Enter your name')

print(name)

Here, the line starting with # is a comment. The Python compiler ignores everything after the # symbol.

Now, let's understand the different types of comments in Python.

## **Single-line Comment**

We use the hash(#) symbol to write a single-line comment. For example,

# declare a variable

name = 'John'

# print name

print(name) # John

[Run Code](https://www.programiz.com/python-programming/online-compiler)

In the above example, we have used three single-line comments:

* # declare a variable
* # print name
* # John

We can also use single-line comments alongside the code:

print(name) # John

Note: Remember the keyboard shortcut to apply comments. In most text editors, it's Ctrl + / if you are on Windows & Cmd + / if you are on a Mac.

## **Multiline Comments**

Python doesn't have dedicated multi-line comment syntax like some other programming languages like C++ and Java.

However, we can achieve the same effect by using the hash (#) symbol at the beginning of each line.

Let's look at an example.

# print(1)

# print(2)

# print(3)

We can also use multiline strings as comments like:

'''This program takes an input from the user

and prints it'''

name = input('Enter your name: ')

print(name)

Output

Enter your name: John

John

We can see that these unassigned multiline strings are ignored.

## **Prevent Executing Code Using Comments**

Comments are valuable when debugging code.

If we encounter an error while running a program, instead of removing code segments, we can comment them out to prevent execution.

For example,

number1 = 10

number2 = 15

sum = number1 + number2

print('The sum is:', sum)

print('The product is:', product)

Here, the code throws an error because we have not defined a product variable.

We can comment out the code that's causing the error.

For example,

number1 = 10

number2 = 15

sum = number1 + number2

print('The sum is:', sum)

# print('The product is:', product)

Output

The sum is 25

Now, the code runs without any errors.

Here, we have resolved the error by commenting out the code related to the product.

If we need to calculate the product in the near future, we can uncomment it.

## **Why Use Comments?**

We should use comments for the following reasons:

* Comments make our code readable for future reference.
* Comments are used for debugging purposes.
* We can use comments for code collaboration as it helps peer developers to understand our code.

# **Python Data Types**

In computer programming, data types specify the type of data that can be stored inside a variable. For example,

num = 24

Here, 24 (an integer) is assigned to the num variable. So the data type of num is of the int class.

## **Python Data Types**

| Data Types | Classes | Description |
| --- | --- | --- |
| Numeric | int, float, complex | holds numeric values |
| String | str | holds sequence of characters |
| Sequence | list, tuple, range | holds collection of items |
| Mapping | dict | holds data in key-value pair form |
| Boolean | bool | holds either True or False |
| Set | set, frozenset | hold collection of unique items |

Since everything is an object in Python programming, data types are actually [classes](https://www.programiz.com/python-programming/class) and [variables](https://www.programiz.com/python-programming/variables-constants-literals) are instances(object) of these classes.

## **Python Numeric Data type**

In Python, numeric data type is used to hold numeric values.

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

* int - holds signed integers of non-limited length.
* float - holds floating decimal points and it's accurate up to 15 decimal places.
* complex - holds complex numbers.

We can use the [type()](https://www.programiz.com/python-programming/methods/built-in/type) function to know which class a variable or a value belongs to.

Let's see an example,

num1 = 5

print(num1, 'is of type', type(num1))

num2 = 2.0

print(num2, 'is of type', type(num2))

num3 = 1+2j

print(num3, 'is of type', type(num3))

Output

5 is of type <class 'int'>

2.0 is of type <class 'float'>

(1+2j) is of type <class 'complex'>

In the above example, we have created three variables named num1, num2 and num3 with values 5, 5.0, and 1+2j respectively.

We have also used the type() function to know which class a certain variable belongs to.

Since,

* 5 is an integer value, type() returns int as the class of num1 i.e <class 'int'>
* 2.0 is a floating value, type() returns float as the class of num2 i.e <class 'float'>
* 1 + 2j is a complex number, type() returns complex as the class of num3 i.e <class 'complex'>

## **Python List Data Type**

List is an ordered collection of similar or different types of items separated by commas and enclosed within brackets [ ]. For example,

languages = ["Swift", "Java", "Python"]

Here, we have created a list named languages with 3 string values inside it.

### **Access List Items**

To access items from a list, we use the index number (0, 1, 2 ...). For example,

languages = ["Swift", "Java", "Python"]

# access element at index 0

print(languages[0]) # Swift

# access element at index 2

print(languages[2]) # Python

In the above example, we have used the index values to access items from the languages list.

* languages[0] - access first item from languages i.e. Swift
* languages[2] - access third item from languages i.e. Python

To learn more about lists, visit [Python List](https://www.programiz.com/python-programming/list).

## **Python Tuple Data Type**

Tuple is an ordered sequence of items same as a list. The only difference is that tuples are immutable. Tuples once created cannot be modified.

In Python, we use the parentheses () to store items of a tuple. For example,

product = ('Xbox', 499.99)

Here, product is a tuple with a string value Xbox and integer value 499.99.

### **Access Tuple Items**

Similar to lists, we use the index number to access tuple items in Python . For example,

# create a tuple

product = ('Microsoft', 'Xbox', 499.99)

# access element at index 0

print(product[0]) # Microsoft

# access element at index 1

print(product[1]) # Xbox

To learn more about tuples, visit [Python Tuples](https://www.programiz.com/python-programming/tuple).

## **Python String Data Type**

String is a sequence of characters represented by either single or double quotes. For example,

name = 'Python'

print(name)

message = 'Python for beginners'

print(message)

Output

Python

Python for beginners

In the above example, we have created string-type variables: name and message with values 'Python' and 'Python for beginners' respectively.

To learn more about strings, visit [Python Strings](https://www.programiz.com/python-programming/string).

## **Python Set Data Type**

Set is an unordered collection of unique items. Set is defined by values separated by commas inside braces { }. For example,

# create a set named student\_id

student\_id = {112, 114, 116, 118, 115}

# display student\_id elements

print(student\_id)

# display type of student\_id

print(type(student\_id))

Output

{112, 114, 115, 116, 118}

<class 'set'>

Here, we have created a set named student\_info with 5 integer values.

Since sets are unordered collections, indexing has no meaning. Hence, the slicing operator [] does not work.

To learn more about sets, visit [Python Sets](https://www.programiz.com/python-programming/set).

## **Python Dictionary Data Type**

Python dictionary is an ordered collection of items. It stores elements in key/value pairs.

Here, keys are unique identifiers that are associated with each value.

Let's see an example,

# create a dictionary named capital\_city

capital\_city = {'Nepal': 'Kathmandu', 'Italy': 'Rome', 'England': 'London'}

print(capital\_city)

Output

{'Nepal': 'Kathmandu', 'Italy': 'Rome', 'England': 'London'}

In the above example, we have created a dictionary named capital\_city. Here,

1. Keys are 'Nepal', 'Italy', 'England'
2. Values are 'Kathmandu', 'Rome', 'London'

### **Access Dictionary Values Using Keys**

We use keys to retrieve the respective value. But not the other way around. For example,

# create a dictionary named capital\_city

capital\_city = {'Nepal': 'Kathmandu', 'Italy': 'Rome', 'England': 'London'}

print(capital\_city['Nepal']) # prints Kathmandu

print(capital\_city['Kathmandu']) # throws error message

Here, we have accessed values using keys from the capital\_city dictionary.

Since 'Nepal' is key, capital\_city['Nepal'] accesses its respective value i.e. Kathmandu

However, 'Kathmandu' is the value for the 'Nepal' key, so capital\_city['Kathmandu'] throws an error message.

To learn more about dictionaries, visit [Python Dictionary](https://www.programiz.com/python-programming/dictionary).