**Python Keywords**

Keywords are the reserved words in Python.

We cannot use a keyword as a [variable](https://www.programiz.com/python-programming/variables-datatypes) name, [function](https://www.programiz.com/python-programming/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 over the course of time.

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| False | await | else | Import | pass |
| None | break | except | In | raise |
| True | class | finally | Is | return |
| And | continue | for | Lambda | try |
| As | def | from | Nonlocal | while |
| Assert | del | global | Not | with |
| Async | elif | if | Or | yield |

## 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

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 a valid name.
3. Keywords cannot be used as identifiers.

global = 1

**Output**

File "<interactive input>", line 1

global = 1

^

SyntaxError: invalid syntax

1. We cannot use special symbols like **!**, **@**, **#**, **$**, **%** etc. in our identifier.

a@ = 0

**Output**

File "<interactive input>", line 1

a@ = 0

^

SyntaxError: invalid syntax

1. An 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 give the identifiers a name that makes sense. While c = 10 is a valid name, writing count = 10 would make more sense, and it would be easier to figure out what it represents when you look at your code after a long gap.

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

# 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 that can be changed later in the program. For example,

number = 10

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

You can think of variables as a bag to store books in it and that book can be replaced at any time.

number = 10

number = 1.1

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

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

## Assigning values to Variables 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 value to a variable

website = "apple.com"

print(website)

**Output**

apple.com

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

**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 apple.com is a string and declares the website variable as a string.

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

website = "apple.com"

print(website)

# assigning a new value to website

website = "programiz.com"

print(website)

**Output**

apple.com

programiz.com

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

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

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

print (a)

print (b)

print (c)

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

x = y = z = "same"

print (x)

print (y)

print (z)

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.

You can think of constants as a bag to store some books which cannot be replaced once placed inside the bag.

## Assigning value to constant in Python

In Python, constants are usually declared and assigned in a module. Here, the module is a new file containing variables, functions, etc which is imported to the 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)

**Output**

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. Naming them in all capital letters is a convention to separate them from variables, however, it does not actually prevent reassignment.

## Rules and Naming Convention for Variables and constants

1. 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:
2. snake\_case
3. MACRO\_CASE
4. camelCase

CapWords

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

current\_salary

1. Use capital letters possible to declare a constant. For example:
2. PI
3. G
4. MASS
5. SPEED\_OF\_LIGHT

TEMP

1. Never use special symbols like !, @, #, $, %, etc.
2. Don't start a variable name with a digit.

## 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?

a = 0b1010 #Binary Literals

b = 100 #Decimal Literal

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)

**Output**

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 parts of complex numbers.

To learn more about Numeric Literals, refer to [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)

**Output**

This is Python

C

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

Ünicöde

raw \n string

In the above program, This is Python is a string literal and C is a character literal.

The value in triple-quotes """ assigned to the multiline\_str is a multi-line string literal.

The string u"\u00dcnic\u00f6de" is a Unicode literal which supports characters other than English. In this case, \u00dc represents Ü and \u00f6 represents ö.

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)

**Output**

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 a 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 that the field has not been 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)

**Output**

Available

None

In the above program, we define a menu function. Inside menu, when we set the argument as drink then, it displays Available. And, when the argument 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)

**Output**

['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, a tuple of numbers, a dictionary dict having values with keys designated to each value and a set of vowels.

## Python Statement

Instructions that a Python interpreter can execute are called statements. For example, a = 1 is 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, the 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 an 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 can 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++, and Java use braces { } to define a block of code. Python, however, 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 are preferred over tabs. Here is an example.

for i in range(1,11):

print(i)

if i == 5:

break

The enforcement of indentation in Python makes the code look neat and clean. This results in Python programs that look similar and consistent.

Indentation can be ignored in line continuation, but it's always a good idea to indent. It makes the code more readable. For example:

if True:

print('Hello')

a = 5

and

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

both are valid and do the same thing, but the former style is clearer.

Incorrect indentation will result in IndentationError.

## Python Comments

Comments are very important while writing a program. They describe what is 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 the time to explain these concepts in the 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 to better understand a program. Python Interpreter ignores comments.

#This is a comment

#print out Hello

print('Hello')

### Multi-line comments

We can have comments that extend up to multiple lines. One way is to use the hash(**#**) symbol at 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 a 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"""

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

### Docstrings in Python

A docstring is short for documentation string.

Python docstrings (documentation strings) are the [string](https://www.programiz.com/python-programming/string) literals that appear right after the definition of a function, method, class, or module.

Triple quotes are used while writing docstrings. For example:

def double(num):

"""Function to double the value"""

return 2\*num

Docstrings appear right after the definition of a function, class, or a module. This separates docstrings from multiline comments using triple quotes.

The docstrings are associated with the object as their \_\_doc\_\_ attribute.

So, we can access the docstrings of the above function with the following lines of code:

def double(num):

"""Function to double the value"""

return 2\*num

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

**Output**

Function to double the value