Website: https://www.programiz.com/python-programming/keywords-identifier

**Python Identifiers**

Identifiers are the name given to variables, classes, methods, etc. For example,

language = 'Python'

Here, language is a variable (an identifier) which holds the value 'Python'.

We cannot use keywords as variable names as they are reserved names that are built-in to Python. For example,

continue = 'Python'

The above code is wrong because we have used continue as a variable name. To learn more about variables, visit [Python Variables](https://www.programiz.com/python-programming/variables-constants-literals).

**Rules for Naming an Identifier**

* Identifiers cannot be a keyword.
* Identifiers are case-sensitive.
* It can have a sequence of letters and digits. However, it must begin with a letter or \_. The first letter of an identifier cannot be a digit.
* It's a convention to start an identifier with a letter rather \_.
* Whitespaces are not allowed.
* We cannot use special symbols like **!**, **@**, **#**, **$**, and so on.

**Some Valid and Invalid Identifiers in Python**

|  |  |
| --- | --- |
| Valid Identifiers | Invalid Identifiers |
| score | @core |
| return\_value | Return |
| highest\_score | highest score |
| name1 | 1name |
| convert\_to\_string | convert to\_string |

## 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 Comments

In this tutorial, we will learn about Python statements, why we use them, and how to use comments in the right way.

In computer programming, comments are hints that we use to make our code more understandable.

Comments are completely ignored by the interpreter. They are meant for fellow programmers. For example,

# declare and initialize two variables

num1 = 6

num2 = 9

# print the output

print('This is output')

Here, we have used the following comments,

* declare and initialize two variables
* print the output

**Types of Comments in Python**

In Python, there are two types of comments:

* single-line comment
* multi-line comment

**Single-line Comment in Python**

A single-line comment starts and ends in the same line. We use the # symbol to write a single-line comment. For example,

# create a variable

name = 'Eric Cartman'

# print the value

print(name)

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

**Output**

Eric Cartman

Here, we have created two single-line comments:

1. # create a variable
2. # print the value

We can also use the single-line comment along with the code.

name = 'Eric Cartman' # name is a string

Here, code before # are executed and code after # are ignored by the interpreter.

**Multi-line Comment in Python**

Python doesn't offer a separate way to write multiline comments. However, there are other ways to get around this issue.

We can use # at the beginning of each line of comment on multiple lines. For example,

# This is a long comment

# and it extends

# to multiple lines

Here, each line is treated as a single comment, and all of them are ignored.

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

These triple quotes are generally used for multi-line strings. But if we do not assign it to any variable or function, we can use it as a comment.

The interpreter ignores the string that is not assigned to any variable or function.

Let's see an example,

''' This is also a

perfect example of

multi-line comments '''

Here, the multiline string isn't assigned to any variable, so it is ignored by the interpreter. Even though it is not technically a multiline comment, it can be used as one.

**Use of Python Comment**

**1. Make Code Easier to Understand**

If we write comments in our code, it will be easier for future reference.

Also, it will be easier for other developers to understand the code.

**2. Using Comments for Debugging**

If we get an error while running the program, we can comment the line of code that causes the error instead of removing it. For example,

print('Python')

# print('Error Line )

print('Django')

Here, print('Error Line) was causing an error so we have changed it to a comment. Now, the program runs without any errors.

This is how comments can be a valuable debugging tool.

**Note**: Always use comments to explain why we did something rather than how we did something. Comments shouldn't be a substitute to explain poorly written code.

# Python Variables, Constants and Literals

In this tutorial, we will learn about Python variables, constants, literals with the help of examples.

**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

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

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)

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

**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

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

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

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

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, 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,

pass = true

Here, true is a boolean literal assigned to pass.

**String and Character Literals in Python**

Char9acter 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

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

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)

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

**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 of fruits, a tuple of numbers, a dictionary of alphabets having values with keys designated to each value and a set of vowels.

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

# Python Data Types

In this tutorial, you will learn about different data types we can use in Python with the help of examples.

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, frozeenset | hold collection of unique items |

Since everything is an object in Python programming, data types are actually classes and variables 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() 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))

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

**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

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

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

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

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)

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

**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))

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

**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)

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

**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

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

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).

Methods of List

### [Python List append()](https://www.programiz.com/python-programming/methods/list/append)

Add a single element to the end of the list

### [Python List clear()](https://www.programiz.com/python-programming/methods/list/clear)

Removes all Items from the List

### [Python List copy()](https://www.programiz.com/python-programming/methods/list/copy)

returns a shallow copy of the list

### [Python List count()](https://www.programiz.com/python-programming/methods/list/count)

returns count of the element in the list

### [Python List extend()](https://www.programiz.com/python-programming/methods/list/extend)

adds iterable elements to the end of the list

### [Python List index()](https://www.programiz.com/python-programming/methods/list/index)

returns the index of the element in the list

### [Python List insert()](https://www.programiz.com/python-programming/methods/list/insert)

insert an element to the list

### [Python List pop()](https://www.programiz.com/python-programming/methods/list/pop)

Removes element at the given index

### [Python List remove()](https://www.programiz.com/python-programming/methods/list/remove)

Removes item from the list

### [Python List reverse()](https://www.programiz.com/python-programming/methods/list/reverse)

reverses the list

### [Python List sort()](https://www.programiz.com/python-programming/methods/list/sort)

sorts elements of a list

Methods of Dictionary

### [Python Dictionary clear()](https://www.programiz.com/python-programming/methods/dictionary/clear)

Removes all Items

### [Python Dictionary copy()](https://www.programiz.com/python-programming/methods/dictionary/copy)

Returns the Shallow Copy of a Dictionary

### [Python Dictionary fromkeys()](https://www.programiz.com/python-programming/methods/dictionary/fromkeys)

creates dictionary from given sequence

### [Python Dictionary get()](https://www.programiz.com/python-programming/methods/dictionary/get)

Returns Value of The Key

### [Python Dictionary items()](https://www.programiz.com/python-programming/methods/dictionary/items)

returns view of dictionary's (key, value) pair

### [Python Dictionary keys()](https://www.programiz.com/python-programming/methods/dictionary/keys)

Returns View Object of All Keys

### [Python Dictionary pop()](https://www.programiz.com/python-programming/methods/dictionary/pop)

removes and returns element having given key

### [Python Dictionary popitem()](https://www.programiz.com/python-programming/methods/dictionary/popitem)

Returns & Removes Latest Element From Dictionary

### [Python Dictionary setdefault()](https://www.programiz.com/python-programming/methods/dictionary/setdefault)

Inserts Key With a Value if Key is not Present

### [Python Dictionary update()](https://www.programiz.com/python-programming/methods/dictionary/update)

Updates the Dictionary

### [Python Dictionary values()](https://www.programiz.com/python-programming/methods/dictionary/values)

returns view of all values in dictionary

## List Methods

# Python List Methods

Python has a lot of list methods that allow us to work with lists. In this reference page, you will find all the list methods to work with Python lists. For example, if you want to add a single item to the end of the list, you can use the list.append() method.

Top of Form

Search List Methods



Bottom of Form

### [Python List append()](https://www.programiz.com/python-programming/methods/list/append)

Add a single element to the end of the list

### [Python List clear()](https://www.programiz.com/python-programming/methods/list/clear)

Removes all Items from the List

### [Python List copy()](https://www.programiz.com/python-programming/methods/list/copy)

returns a shallow copy of the list

### [Python List count()](https://www.programiz.com/python-programming/methods/list/count)

returns count of the element in the list

### [Python List extend()](https://www.programiz.com/python-programming/methods/list/extend)

adds iterable elements to the end of the list

### [Python List index()](https://www.programiz.com/python-programming/methods/list/index)

returns the index of the element in the list

### [Python List insert()](https://www.programiz.com/python-programming/methods/list/insert)

insert an element to the list

### [Python List pop(i)](https://www.programiz.com/python-programming/methods/list/pop)

Removes element at the given index

### [Python List remove()](https://www.programiz.com/python-programming/methods/list/remove)

Removes item from the list

### [Python List reverse()](https://www.programiz.com/python-programming/methods/list/reverse)

reverses the list

### [Python List sort()](https://www.programiz.com/python-programming/methods/list/sort)

sorts elements of a list

## String methods

### [Python String capitalize()](https://www.programiz.com/python-programming/methods/string/capitalize)

Converts first character to Capital Letter

### [Python String casefold()](https://www.programiz.com/python-programming/methods/string/casefold)

converts to case folded strings

### [Python String center()](https://www.programiz.com/python-programming/methods/string/center)

Pads string with specified character

### [Python String count()](https://www.programiz.com/python-programming/methods/string/count)

returns occurrences of substring in string

### [Python String encode()](https://www.programiz.com/python-programming/methods/string/encode)

returns encoded string of given string

### [Python String endswith()](https://www.programiz.com/python-programming/methods/string/endswith)

Checks if String Ends with the Specified Suffix

### [Python String expandtabs()](https://www.programiz.com/python-programming/methods/string/expandtabs)

Replaces Tab character With Spaces

### [Python String find()](https://www.programiz.com/python-programming/methods/string/find)

Returns the index of first occurrence of substring

### [Python String format()](https://www.programiz.com/python-programming/methods/string/format)

formats string into nicer output

### [Python String format\_map()](https://www.programiz.com/python-programming/methods/string/format_map)

Formats the String Using Dictionary

### [Python String index()](https://www.programiz.com/python-programming/methods/string/index)

Returns Index of Substring

### [Python String isalnum()](https://www.programiz.com/python-programming/methods/string/isalnum)

Checks Alphanumeric Character

### [Python String isalpha()](https://www.programiz.com/python-programming/methods/string/isalpha)

Checks if All Characters are Alphabets

### [Python String isdecimal()](https://www.programiz.com/python-programming/methods/string/isdecimal)

Checks Decimal Characters

### [Python String isdigit()](https://www.programiz.com/python-programming/methods/string/isdigit)

Checks Digit Characters

### [Python String isidentifier()](https://www.programiz.com/python-programming/methods/string/isidentifier)

Checks for Valid Identifier

### [Python String islower()](https://www.programiz.com/python-programming/methods/string/islower)

Checks if all Alphabets in a String are Lowercase

### [Python String isnumeric()](https://www.programiz.com/python-programming/methods/string/isnumeric)

Checks Numeric Characters

### [Python String isprintable()](https://www.programiz.com/python-programming/methods/string/isprintable)

Checks Printable Character

### [Python String isspace()](https://www.programiz.com/python-programming/methods/string/isspace)

Checks Whitespace Characters

### [Python String istitle()](https://www.programiz.com/python-programming/methods/string/istitle)

Checks for Titlecased String

### [Python String isupper()](https://www.programiz.com/python-programming/methods/string/isupper)

returns if all characters are uppercase characters

### [Python String join()](https://www.programiz.com/python-programming/methods/string/join)

Returns a Concatenated String

### [Python String ljust()](https://www.programiz.com/python-programming/methods/string/ljust)

returns left-justified string of given width

### [Python String lower()](https://www.programiz.com/python-programming/methods/string/lower)

returns lowercased string

### [Python String lstrip()](https://www.programiz.com/python-programming/methods/string/lstrip)

Removes Leading Characters

### [Python String maketrans()](https://www.programiz.com/python-programming/methods/string/maketrans)

returns a translation table

### [Python String partition()](https://www.programiz.com/python-programming/methods/string/partition)

Returns a Tuple

### [Python String replace()](https://www.programiz.com/python-programming/methods/string/replace)

Replaces Substring Inside

### [Python String rfind()](https://www.programiz.com/python-programming/methods/string/rfind)

Returns the Highest Index of Substring

### [Python String rindex()](https://www.programiz.com/python-programming/methods/string/rindex)

Returns Highest Index of Substring

### [Python String rjust()](https://www.programiz.com/python-programming/methods/string/rjust)

returns right-justified string of given width

### [Python String rpartition()](https://www.programiz.com/python-programming/methods/string/rpartition)

Returns a Tuple

### [Python String rsplit()](https://www.programiz.com/python-programming/methods/string/rsplit)

Splits String From Right

### [Python String rstrip()](https://www.programiz.com/python-programming/methods/string/rstrip)

Removes Trailing Characters

### [Python String split()](https://www.programiz.com/python-programming/methods/string/split)

Splits String from Left

### [Python String splitlines()](https://www.programiz.com/python-programming/methods/string/splitlines)

Splits String at Line Boundaries

### [Python String startswith()](https://www.programiz.com/python-programming/methods/string/startswith)

Checks if String Starts with the Specified String

### [Python String strip()](https://www.programiz.com/python-programming/methods/string/strip)

Removes Both Leading and Trailing Characters

### [Python String swapcase()](https://www.programiz.com/python-programming/methods/string/swapcase)

swap uppercase characters to lowercase; vice versa

### [Python String title()](https://www.programiz.com/python-programming/methods/string/title)

Returns a Title Cased String

### [Python String translate()](https://www.programiz.com/python-programming/methods/string/translate)

returns mapped charactered string

### [Python String upper()](https://www.programiz.com/python-programming/methods/string/upper)

returns uppercased string

### [Python String zfill()](https://www.programiz.com/python-programming/methods/string/zfill)

Returns a Copy of The String Padded With Zeros

Website: https://flexiple.com/python/looping-statements-in-python/

# Looping Statements in Python

The flow of the programs written in any programming language is sequential by default. The first statement in a function is executed first, followed by the second, and so on. There may be a situation when the programmer needs to execute a block of code several times. For this purpose, The programming languages provide various kinds of loops that are able to repeat some particular code numerous numbers of times. Here, we are going to talk about looping statements in Python.

In a programming language, a looping statement contains instructions that continually repeat until a certain condition is reached. Read on to find out more about them.

### Table of Content

* Looping statements in Python
* For Loop
* While Loop
* Nested Loop

## Looping statements in Python

Looping simplifies complicated problems into smooth ones. It allows programmers to modify the flow of the program so that rather than writing the same code, again and again, programmers are able to repeat the code a finite number of times.

In Python, there are three different types of loops: for loop, while loop, and nested loop.  
Here, we will read about these different types of loops and how to use them.

## For Loop

The for loop is used in the case where a programmer needs to execute a part of the code until the given condition is satisfied. The for loop is also called a pre-tested loop. It is best to use for loop if the number of iterations is known in advance.

In Python, there is no C style for loop, i.e., for (i=0; i<n; i++).

#### Syntax:

**for** variable **in** sequence:

    statements(s)

#### Input:

a = 5

**for** i **in** range(0, a):

    print(i)

#### Output:

0

1

2

3

4

The for loop runs till the value of i is less than a. As the value of i is 5, the loop ends.

## While Loop

The while loop is to be used in situations where the number of iterations is unknown at first. The block of statements is executed in the while loop until the condition specified in the while loop is satisfied. It is also called a pre-tested loop.

In Python, the while loop executes the statement or group of statements repeatedly while the given condition is True. And when the condition becomes false, the loop ends and moves to the next statement after the loop.

#### Syntax:

While condition:

statement(s)

#### Input:

count = 0

**while** (count < 5):

    count = count + 1

    print("Flexiple")

#### Output:

Flexiple

Flexiple

Flexiple

Flexiple

Flexiple

The loop prints ‘Flexiple’ till the value of count becomes 5 and the condition is False.

## Nested Looping statements in Python

The Python programming language allows programmers to use one looping statement inside [another looping](https://pynative.com/python-nested-loops/) statement.

#### Syntax:

#for loop statement

**for** variable **in** sequence:

**for** variable **in** sequence:

     statement(s)

statement(s)

#while loop statement

**while** condition:

**while** condition:

statement(s)

statement(s)

#### Input:

**for** i **in** range(1, 7):

**for** j **in** range(i):

         print(i, end=' ')

    print()

#### Output:

1

2 2

3 3 3

4 4 4 4

5 5 5 5 5

6 6 6 6 6 6

## Closing Thoughts

In this tutorial, we read about different looping statements in Python and their uses. The looping statements are used to repeat a specific block of code various number of times. One can read about other Python concepts [here](https://flexiple.com/python/control-statements-in-python/).

# Python for Loop

In this tutorial, we'll learn how to use a for loop in Python with the help of examples.

In computer programming, loops are used to repeat a block of code.

For example, if we want to show a message **100** times, then we can use a loop. It's just a simple example; you can achieve much more with loops.

There are 2 types of loops in Python:

* [for loop](https://www.programiz.com/python-programming/for-loop)
* [while loop](https://www.programiz.com/python-programming/while-loop)

**Python for Loop**

In Python, the for loop is used to run a block of code for a certain number of times. It is used to iterate over any sequences such as [list](https://www.programiz.com/python-programming/list), [tuple](https://www.programiz.com/python-programming/tuple), [string](https://www.programiz.com/python-programming/string), etc.

The syntax of the for loop is:

for val in sequence:

# statement(s)

Here, val accesses each item of sequence on each iteration. Loop continues until we reach the last item in the sequence.

**Flowchart of Python for Loop**

Working of Python for loop

**Example: Loop Over Python List**

languages = ['Swift', 'Python', 'Go', 'JavaScript']

# access items of a list using for loop

for language in languages:

print(language)

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

**Output**

Swift

Python

Go

JavaScript

In the above example, we have created a list called languages.

Initially, the value of language is set to the first element of the array,i.e. Swift, so the print statement inside the loop is executed.

language is updated with the next element of the array and the print statement is executed again. This way the loop runs until the last element of an array is accessed.

**Python for Loop with Python range()**

A [range](https://www.programiz.com/python-programming/methods/built-in/range) is a series of values between two numeric intervals.

We use Python's built-in function range() to define a range of values. For example,

values = range(4)

Here, **4** inside range() defines a range containing values **0, 1, 2, 3.**

In Python, we can use for loop to iterate over a range. For example,

# use of range() to define a range of values

values = range(4)

# iterate from i = 0 to i = 3

for i in values:

print(i)

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

**Output**

0

1

2

3

In the above example, we have used the for loop to iterate over a range from **0** to **3**.

The value of i is set to **0** and it is updated to the next number of the range on each iteration. This process continues until **3** is reached.

|  |  |  |
| --- | --- | --- |
| Iteration | Condition | Action |
| 1st | True | 0 is printed. i is increased to **1**. |
| 2nd | True | 1 is printed. i is increased to **2**. |
| 3rd | True | 2 is printed. i is increased to **3**. |
| 4th | True | 3 is printed. i is increased to **4**. |
| 5th | False | The loop is terminated |

**Note**: To learn more about the use of for loop with range, visit [Python range()](https://www.programiz.com/python-programming/methods/built-in/range).

**Python for loop with else**

A for loop can have an optional else block as well. The else part is executed when the loop is finished. For example,

digits = [0, 1, 5]

for i in digits:

print(i)

else:

print("No items left.")

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

**Output**

0

1

5

No items left.

Here, the for loop prints all the items of the digits list. When the loop finishes, it executes the else block and prints No items left.

**Note**: The else block will not execute if the for loop is stopped by a [break](https://www.programiz.com/python-programming/break-continue) statement.

# Python while Loop

In this tutorial, we will learn about the while loop in Python programming with the help of examples.

In programming, loops are used to repeat a block of code. For example, if we want to show a message **100** times, then we can use a loop. It's just a simple example, we can achieve much more with loops.

In the previous tutorial, we learned about [Python for loop](https://www.programiz.com/python-programming/for-loop). Here, we are going to learn about while loops.

**Python while Loop**

Python while loop is used to run a specific code until a certain condition is met.

The syntax of while loop is:

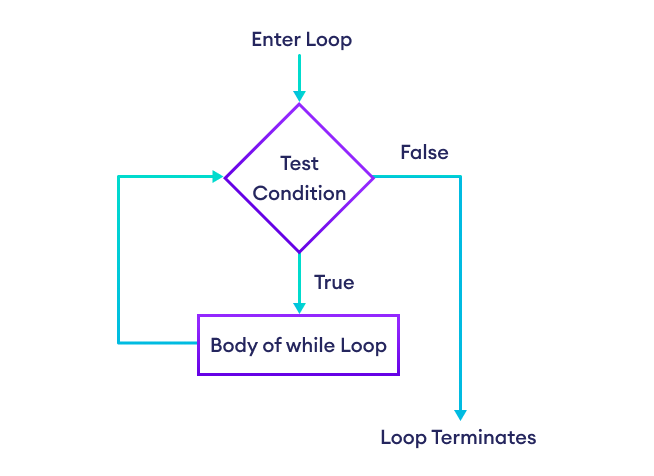
while condition:

# body of while loop

Here,

1. A while loop evaluates the condition
2. If the condition evaluates to True, the code inside the while loop is executed.
3. condition is evaluated again.
4. This process continues until the condition is False.
5. When condition evaluates to False, the loop stops.

**Flowchart for Python While Loop**

Flowchart for while Loop in Python

**Example: Python while Loop**

# program to display numbers from 1 to 5

# initialize the variable

i = 1

n = 5

# while loop from i = 1 to 5

while i <= n:

print(i)

i = i + 1

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

**Output**

1

2

3

4

5

Here's how the program works:

|  |  |  |
| --- | --- | --- |
| Variable | Condition: i <= n | Action |
| i = 1 n = 5 | True | 1 is printed. i is increased to **2**. |
| i = 2 n = 5 | True | 2 is printed. i is increased to **3**. |
| i = 3 n = 5 | True | 3 is printed. i is increased to **4**. |
| i = 4 n = 5 | True | 4 is printed. i is increased to **5**. |
| i = 5 n = 5 | True | 5 is printed. i is increased to **6**. |
| i = 6 n = 5 | False | The loop is terminated. |

**Example 2: Python while Loop to Display Game Level**

current\_level = 0

final\_level = 5

game\_completed = True

while current\_level <= final\_level:

if game\_completed:

print('You have passed level', current\_level)

current\_level += 1

print('Level Ends')

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

**Output**

You have passed level 0

You have passed level 1

You have passed level 2

You have passed level 3

You have passed level 4

You have passed level 5

Level Ends

In the above example, we have used the while loop to check the current level and display it on the console.

**Infinite while Loop in Python**

If the condition of a loop is always True, the loop runs for infinite times (until the memory is full). For example,

# infinite while loop

while True:

# body of the loop

In the above example, the condition is always True. Hence, the loop body will run for infinite times.

**Python While loop with else**

A while loop can have an optional else block as well.

The else part is executed after the condition in the while loop evaluates to False. For example,

counter = 0

while counter < 3:

print('Inside loop')

counter = counter + 1

else:

print('Inside else')

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

**Output**

Inside loop

Inside loop

Inside loop

Inside else

Here, we have used the counter variable to print the 'Inside Loop' string three times.

On the fourth iteration, the condition in while becomes False. Hence, the else part is executed.

**Note**: The else block will not execute if the while loop is stopped by a [break](https://www.programiz.com/python-programming/break-continue) statement.

**Python for vs while loops**

The for loop is usually used when the number of iterations is known. For example,

# this loop is iterated 4 times (0 to 3)

for i in range(4):

print(i)

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

And while loop is usually used when the number of iterations are unknown. For example,

while condition:

# body of loop

# Python break and continue

In this tutorial, we will learn to use break and continue statements to alter the flow of a loop.

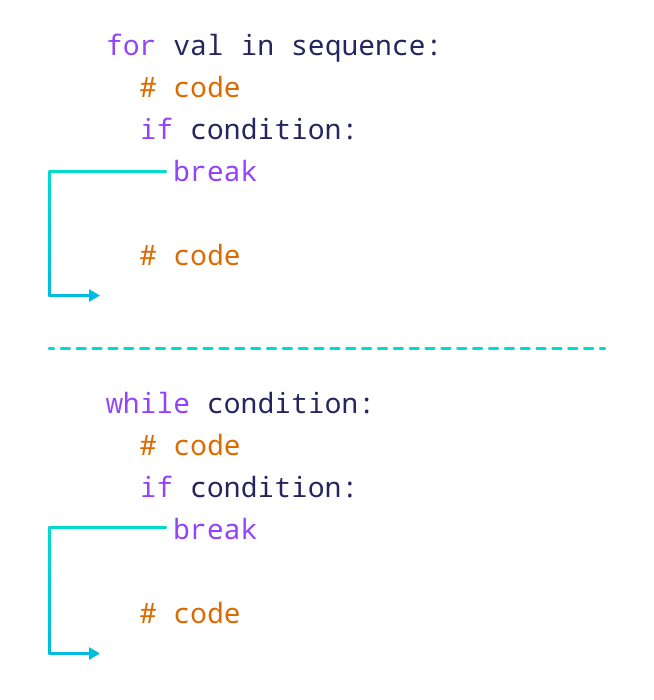
**Python break Statement**

The break statement is used to terminate the loop immediately when it is encountered.

The syntax of the break statement is:

break

**Working of Python break Statement**

Working of the break statement

The working of break statement in [for loop](https://www.programiz.com/python-programming/for-loop) and [while loop](https://www.programiz.com/python-programming/while-loop) is shown above.

**Python break Statement with for Loop**

We can use the break statement with the for loop to terminate the loop when a certain condition is met. For example,

for i in range(5):

if i == 3:

break

print(i)

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

**Output**

0

1

2

In the above example, we have used the for loop to print the value of i. Notice the use of the break statement,

if i == 3:

break

Here, when i is equal to **3**, the break statement terminates the loop. Hence, the output doesn't include values after **2**.

**Note**: The break statement is almost always used with decision-making statements.

**Python break Statement with while Loop**

We can also terminate the while loop using the break statement. For example,

# program to find first 5 multiples of 6

i = 1

while (i<=10):

print('6 \* ',(i), '=',6 \* i)

if i >= 5:

break

i = i + 1

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

**Output**

6 \* 1 = 6

6 \* 2 = 12

6 \* 3 = 18

6 \* 4 = 24

6 \* 5 = 30

In the above example, we have used the while loop to find the first **5** multiples of **6**. Here notice the line,

if i >= 5:

break

This means when i is greater than or equal to **5**, the while loop is terminated.

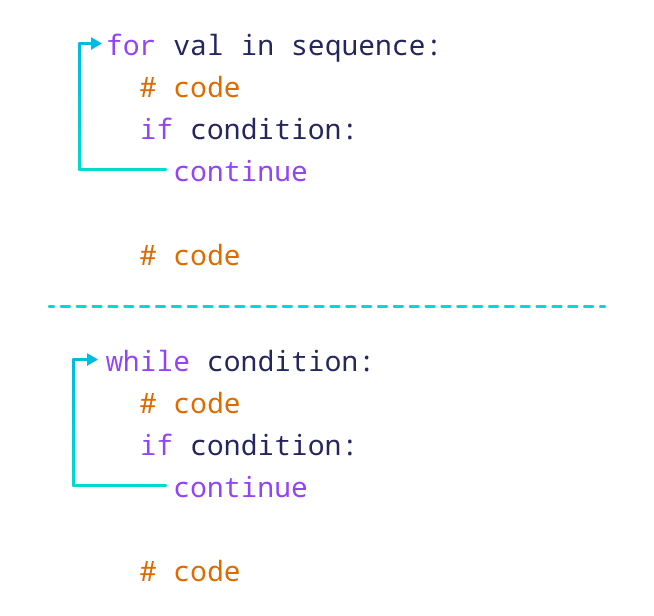
**Python continue Statement**

The continue statement is used to skip the current iteration of the loop and the control flow of the program goes to the next iteration.

The syntax of the continue statement is:

continue

**Working of Python continue Statement**

How continue statement works in python

The working of the continue statement in for and while loop is shown above.

**Python continue Statement with for Loop**

We can use the continue statement with the for loop to skip the current iteration of the loop. Then the control of the program jumps to the next iteration. For example,

for i in range(5):

if i == 3:

continue

print(i)

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

**Output**

0

1

2

4

In the above example, we have used the for loop to print the value of i. Notice the use of the continue statement,

if i == 3:

continue

Here, when i is equal to **3**, the continue statement is executed. Hence, the value **3** is not printed to the output.

**Python continue Statement with while Loop**

In Python, we can also skip the current iteration of the while loop using the continue statement. For example,

# program to print odd numbers from 1 to 10

num = 0

while num < 10:

num += 1

if (num % 2) == 0:

continue

print(num)

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

**Output**

1

3

5

7

9

In the above example, we have used the while loop to print the odd numbers between **1** to **10**. Notice the line,

if (num % 2) == 0:

continue

Here, when the number is even, the continue statement skips the current iteration and starts the next iteration.

# Python pass Statement

In this tutorial, we'll learn about the pass statement in Python programming with the help of examples.

In Python programming, the pass statement is a null statement which can be used as a placeholder for future code.

Suppose we have a loop or a function that is not implemented yet, but we want to implement it in the future. In such cases, we can use the pass statement.

The syntax of the pass statement is:

pass

**Using pass With Conditional Statement**

n = 10

# use pass inside if statement

if n > 10:

pass

print('Hello')

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

Here, notice that we have used the pass statement inside the if statement.

However, nothing happens when the pass is executed. It results in no operation (NOP).

Suppose we didn't use pass or just put a comment as:

n = 10

if n > 10:

# write code later

print('Hello')

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

Here, we will get an error message: IndentationError: expected an indented block

**Note**: The difference between a [comment](https://www.programiz.com/python-programming/statement-indentation-comments) and a pass statement in Python is that while the interpreter ignores a comment entirely, pass is not ignored.

**Use of pass Statement inside Function or Class**

We can do the same thing in an empty [function](https://www.programiz.com/python-programming/function) or [class](https://www.programiz.com/python-programming/class) as well. For example,

def function(args):

pass

class Example:

pass

# Python Functions

In this tutorial, we will learn about the Python function and function expressions with the help of examples.

# Python Global Keyword

In this tutorial, we'll learn about the global keyword with the help of examples.

In Python, the global keyword allows us to modify the variable outside of the current scope.

It is used to create a global variable and make changes to the variable in a local context.

Before we learn about the global keyword, make sure you have got some basics of [Python Variable Scope](https://www.programiz.com/python-programming/global-local-nonlocal-variables).

## Access and Modify Python Global Variable

First let's try to access a global variable from the inside of a function,

c = 1 # global variable

def add():

print(c)

add()

# Output: 1

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

Here, we can see that we have accessed a global variable from the inside of a function.

However, if we try to modify the global variable from inside a function as:

# global variable

c = 1

def add():

# increment c by 2

c = c + 2

print(c)

add()

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

**Output**

UnboundLocalError: local variable 'c' referenced before assignment

This is because we can only access the global variable but cannot modify it from inside the function.

The solution for this is to use the global keyword.

### Example: Changing Global Variable From Inside a Function using global

# global variable

c = 1

def add():

# use of global keyword

global c

# increment c by 2

c = c + 2

print(c)

add()

# Output: 3

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

In the above example, we have defined c as the global keyword inside add().

Then, we have incremented the variable c by **2**, i.e c = c + 2.

As we can see while calling add(), the value of global variable c is modified from **1** to **3**.

## Global in Nested Functions

In Python, we can also use the global keyword in a nested function. For example,

def outer\_function():

num = 20

def inner\_function():

global num

num = 25

print("Before calling inner\_function(): ", num)

inner\_function()

print("After calling inner\_function(): ", num)

outer\_function()

print("Outside both function: ", num)

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

**Output**

Before calling inner\_function(): 20

After calling inner\_function(): 20

Outside both function: 25

In the above example, we declared a global variable inside the nested function inner\_function().

Inside outer\_function(), num has no effect of the global keyword.

Before and after calling inner\_function(), num takes the value of the local variable i.e num = 20.

Outside of the outer\_function() function, num will take the value defined in the inner\_function() function i.e x = 25.

This is because we have used the global keyword in num to create a global variable inside the inner\_function() function (local scope).

So, if we make any changes inside the inner\_function() function, the changes appear outside the local scope, i.e. outer\_function().

## Rules of global Keyword

The basic rules for global keyword in Python are:

* When we create a variable inside a function, it is local by default.
* When we define a variable outside of a function, it is global by default. You don't have to use the global keyword.
* We use the global keyword to read and write a global variable inside a function.
* Use of the global keyword outside a function has no effect.