

# Identifiers & Variables

## Identifiers

Python Identifiers are user-defined names to represent a variable, function, class, module or any other object. If you assign some name to a programmable entity in Python, then it is nothing but technically called an identifier. Python language lays down a set of rules for programmers to create meaningful identifiers.

### Creating Identifiers In Python.

To form an identifier, use a sequence of letters either in lowercase (a to z) or uppercase (A to Z). However, you can also mix up digits (0 to 9) or an underscore (\_) while writing an identifier.

For example – Names like shapeClass, shape\_1, and upload\_shape\_to\_db are all valid identifiers.

You can't use digits to begin an identifier name. It'll lead to the syntax error. For example – The name, 0Shape is incorrect, but shape1 is a valid identifier. Also, the Keywords are reserved, so you should not use them as identifiers.

```
>>> for=1
SyntaxError: invalid syntax
>>> True=1
SyntaxError: can't assign to keyword
```

Python Identifiers can also not have special characters [‘.’, ‘!’, ‘@’, ‘#’, ‘\$’, ‘%’] in their formation. These symbols are forbidden.

```
>>> @index=0  
  
SyntaxError: invalid syntax  
  
>>> isPython?=True  
  
SyntaxError: invalid syntax
```

Python doc says that you can have an identifier with unlimited length. But it is just the half truth. Using a large name (more than 79 chars) would lead to the violation of a rule set by the PEP-8 standard. It says. Limit all lines to a maximum of 79 characters.

### **Best Practices for Identifier Naming.**

- ✓ Better have class names starting with a capital letter. All other identifiers should begin with a lowercase letter.
- ✓ Declare private identifiers by using the (‘\_’) underscore as their first letter.
- ✓ Don’t use ‘\_’ as the leading and trailing character in an identifier. As Python built-in types already use this notation.
- ✓ Avoid using names with only one character. Instead, make meaningful names.

For example – While `i = 1` is valid, but writing `iter = 1` or `index = 1` would make more sense.

- ✓ You can use underscore to combine multiple words to form a sensible name.

For example – `count_no_of_letters`.

## **Variables**

A variable in Python represents an entity whose value can change as and when required. Conceptually, it is a memory location which holds the actual value. And we can retrieve the value from our code by querying the entity. But it requires assigning a label to that memory location so that we can reference it. And we call it as a variable in the programming terms.

Following are some of the key facts about Python variables. These will help programmers to use them efficiently.

- ❖ Variables don't require declaration. However, you must initialize them before use.

For example –

```
test = 10
```

The above expression will lead to the following actions. Creation of an object to represent the value 10. If the variable (test) doesn't exist, then it'll get created. Associate the variable with the object, so that it can refer the value.

The variable 'test' is a reference to the value '10'. Please refer from the illustration shown below.

Whenever the expression changes, Python associates a new object (a chunk of memory) to the variable for referencing that value. And the old one goes to the garbage collector.

Example.

```
>>> test = 10
>>> id(test)
1716585200
>>> test = 11
>>> id(test)
1716585232
>>>
```

Also, for optimization, Python builds a cache and reuses some of the immutable objects, such as small integers and strings.

❖ An object is just a region of memory which can hold the following.


The actual object values.

A type designator to reflect the object type.

The reference counter which determines when it's OK to reclaim the object.

- ❖ It's the object which has a type, not the variable. However, a variable can hold objects of different types as and when required.

Example.



```
>>> test = 10
>>> type(test)
<class 'int'>
>>> test = 'techbeamers'
>>> type(test)
<class 'str'>
>>> test = {'Python', 'C', 'C++'}
>>> type(test)
<class 'set'>
```

## **Keywords / Reserved Words**

Keywords are the reserved words in Python. We cannot use a keyword as variable name, function name or any other identifier. They are used to define the syntax and structure of the Python language. In Python, keywords are case sensitive.

There are 33 keywords in Python 3.3. 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    |        |

### Operators in python

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







For example:

```
>>> 2+3
```

```
5
```

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

**Python language supports the following types of operators.**

-  Arithmetic Operators
-  Comparison (Relational) Operators
-  Assignment Operators
-  Bitwise Operators
-  Membership Operators
-  Identity Operators

## **1. Arithmetic operators**

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

Arithmetic operators in Python

| Operator | Meaning   | Example                           |
|----------|---|-----------------------------------|
| +        | Add two operands or unary plus                                      | $x + y$                           |
| -        | Subtract right operand from the left or unary minus                 | $x - y$                           |
| *        | Multiply two operands   | $x * y$                           |
| /        | Divide left operand by the right one<br>(always results into float) | $x / y$                           |
| %        | Modulus - remainder of the division of left operand by the right    | $x \% y$<br>(remainder of $x/y$ ) |

|    |  |                         |
|----|--|-------------------------|
| // | Floor division - division that results into whole number adjusted to the left in the number line | x // y                  |
| ** | Exponent - left operand raised to the power of right   | x**y (x to the power y) |

## 2. Comparison Operators

These operators compare the values on either sides of them and decide the relation among them. They are also called Relational operators.

| Operator | Meaning   | Example   |
|----------|---|-----------|
| ==       | If the values of two operands are equal, then the condition becomes true. | (a == b)  |
| !=       | If values of two operands are not equal, then condition becomes true.     | (a != b). |
| <>       | If values of two operands are not equal, then condition becomes true.     | (a <> b)  |



|    |   |          |
|----|---|----------|
| >  | If the value of left operand is greater than the value of right operand, then condition becomes true.             | (a > b)  |
| <  | If the value of left operand is less than the value of right operand, then condition becomes true.                | (a < b)  |
| >= | If the value of left operand is greater than or equal to the value of right operand, then condition becomes true. | (a >= b) |
| <= | If the value of left operand is less than or equal to the value of right operand, then condition becomes true.    | (a <= b) |

### 3. Assignment Operators

| Op<br>era<br>tor | meaning   | Example   |
|------------------|---|---|
| =                | Assigns values from right side operands to left side operand                            | <code>c = a + b</code> assigns value of <code>a + b</code> into <code>c</code>  |
| +=               | It adds right operand to the left operand and assign the result to left operand         | <code>c += a</code> is equivalent to <code>c = c + a</code>   |
| -=               | It subtracts right operand from the left operand and assign the result to left operand  | <code>c -= a</code> is equivalent to <code>c = c - a</code>   |
| *=               | It multiplies right operand with the left operand and assign the result to left operand | <code>c *= a</code> is equivalent to <code>c = c * a</code>   |
| /=               | It divides left operand with the right operand and assign the result to left operand    | <code>c /= a</code> is equivalent to <code>c = c / a</code><br><code>ac /= a</code> is equivalent to <code>c = c / a</code> |

|                  |  |   |
|------------------|--|---|
| <code>%=</code>  | It takes modulus using two operands and assign the result to left operand                  | <code>c %= a</code> is equivalent to <code>c = c % a</code>   |
| <code>**=</code> | Performs exponential (power) calculation on operators and assign value to the left operand | <code>c **= a</code> is equivalent to <code>c = c ** a</code> |
| <code>//=</code> | It performs floor division on operators and assign value to the left operand               | <code>c //= a</code> is equivalent to <code>c = c // a</code> |

#### 4. Bitwise Operators

Bitwise operator works on bits and performs bit by bit operation. Assume if `a = 60`; and `b = 13`; Now in binary format they will be as follows –

`a = 0011 1100`

`b = 0000 1101`

-----

`a&b = 0000 1100`

`a|b = 0011 1101`

`a^b = 0011 0001`

`~a = 1100 0011`

There are following Bitwise operators supported by Python language

| Operator                       | Meaning   | Example   |
|--------------------------------|---|---|
| & Binary<br>AND                | Operator copies a bit to the result if it exists in both operands | (a & b)<br>(means 0000<br>1100)   |
| Binary OR                      | It copies a bit if it exists in either operand.                   | (a   b) = 61<br>(means 0011<br>1101)  |
| ^ Binary<br>XOR                | It copies the bit if it is set in one operand but not both.       | (a ^ b) = 49<br>(means 0011<br>0001)  |
| ~ Binary<br>Ones<br>Complement | It is unary and has the effect of 'flipping' bits.                | (~a) = -61<br>(means 1100<br>0011 in 2's<br>complement<br>form due to a<br>signed binary<br>number. |

|                             |  |                                      |
|-----------------------------|--|--------------------------------------|
| <<    Binary<br>Left Shift  | The left operands value is moved left by the number of bits specified by the right operand.  | $a \ll 2 = 240$<br>(means 1111 0000) |
| >>    Binary<br>Right Shift | The left operands value is moved right by the number of bits specified by the right operand. | $a \gg 2 = 15$<br>(means 0000 1111)  |

## 5. Membership Operators

Python's membership operators test for membership in a sequence, such as strings, lists, or tuples. There are two membership operators as explained below –

| Operator | Meaning   | Example  |
|----------|---|--|
| in       | Evaluates to true if it finds a variable in the specified sequence and false otherwise. | $x \text{ in } y$ , here in results in a 1 if x is a member of sequence y. |

|        |  |  |
|--------|--|--|
| not in | Evaluates to true if it does not finds a variable in the specified sequence and false otherwise. | x not in y, here not in results in a 1 if x is not a member of sequence y. |
|--------|--|--|

## 6. Identity Operators

Identity operators compare the memory locations of two objects. There are two Identity operators explained below

| Operator | Meaning   | Example   |
|----------|---|---|
| is       | Evaluates to true if the variables on either side of the operator point to the same object and false otherwise. | x is y, here <b>is</b> results in 1 if id(x) equals id(y).                  |
| is not   | Evaluates to false if the variables on either side of the operator point to the same object and true otherwise. | x is not y, here <b>is not</b> results in 1 if id(x) is not equal to id(y). |

## Operator precedence

The operator precedence in Python are listed in the following table. It is in descending order, upper group has higher precedence than the lower ones.

| Operators                                    | Meaning   |
|--|---|
| ()   | Parentheses                                       |
| **   | Exponent  |
| +X, -X, ~X                                   | Unary plus, Unary minus, Bitwise NOT              |
| *, /, //, %                                  | Multiplication, Division, Floor division, Modulus |
| +, -   | Addition, Subtraction                             |
| <<, >>                                       | Bitwise shift operators                           |
| &  | Bitwise AND                                       |
| ^  | Bitwise XOR                                       |
|  | Bitwise OR  |
| ==, !=, >, >=, <, <=, is, is not, in, not in | Comparisons, Identity, Membership operators       |
| Not  | Logical NOT                                       |
| And  | Logical AND                                       |
| Or   | Logical OR  |