links

roadmap.sh_python
programiz_python

RoadMap Python

Learn the Basics

Getting Started with Python -> site

You can run Python on your computer using the following two methods:

- Run Python online: you can use our free online Python editor.
- Install Python on your computer:
  1. Install VS Code Official website
  2. Download Python Installer File [Python website](https://www.python.org/downloads/

Python Basic Input and Output ->YouTube

The following program displays `Hello, World!` on the screen
In Python, we can simply use the print() function to print output. For example,

```
print("Hello, World!")

# Output: Hello, World!
```

Python Comments

Comments are hints that we add to our code to make it easier to understand. Python comments start with `#`. For example,

```
# print a number
print(25)
```

Python Variables

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

```
number = 10
```

Here, number is a variable storing the value **10**.

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
```

Changing the Value of a Variable in Python

```
site_name = 'programiz.pro'
print(site_name) # Output: programiz.pro

# assigning a new value to site_name
site_name = 'apple.com'

print(site_name) # Output: apple.com
```

Other examples

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

print (a)  # Output:  5
print (b)  # Output: 3.2
print (c)  # Output: Hello

site1 = site2  = 'programiz.com'

print (x)  # Output:  programiz.com
print (y)  # Output:  programiz.com
```

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

Data Types	Classes	Description
Mapping	dict	holds data in key-value pair form
Boolean	bool	holds either `True` or `False`
Set	set, frozenset	hold collection of unique items

Python Operators

Operators are special symbols that perform operations on variables and values. For example,

```
print(5 + 6)   # 11
```

Types of Python Operators

1. Python Arithmetic Operators

Operator	Operation	Example
`+`	Addition	`5 + 2 = 7`
`-`	Subtraction	`4 - 2 = 2`
`*`	Multiplication	`2 * 3 = 6`
`/`	Division	`4 / 2 = 2`
`//`	Floor Division	`10 // 3 = 3`
`%`	Modulo	`5 % 2 = 1`
`**`	Power	`4 ** 2 = 16`

Example : Arithmetic Operators in Python

```
a = 7
b = 2

# addition
print ('Sum: ', a + b)

# subtraction
print ('Subtraction: ', a - b)

# multiplication
print ('Multiplication: ', a * b)

# division
print ('Division: ', a / b)
```

```
# floor division
print ('Floor Division: ', a // b)

# modulo
print ('Modulo: ', a % b)

# a to the power b
print ('Power: ', a ** b)
```

Output:

```
Sum: 9
Subtraction: 5
Multiplication: 14
Division: 3.5
Floor Division: 3
Modulo: 1
Power: 49
```

---

2. Python Assignment Operators

Here's a list of different assignment operators available in Python.

Operator	Name	Example
`=`	Assignment Operator	`a = 7`
`+=`	Addition Assignment	`a += 1 # a = a + 1`
`-=`	Subtraction Assignment	`a -= 3 # a = a - 3`
`*=`	Multiplication Assignment	`a *= 4 # a = a * 4`
`/=`	Division Assignment	`a /= 3 # a = a / 3`
`%=`	Remainder Assignment	`a %= 10 # a = a % 10`
`**=`	Exponent Assignment	`a **= 10 # a = a ** 10`

Example : Assignment Operators

```
# assign 10 to a
a = 10

# assign 5 to b
b = 5
```

```
# assign the sum of a and b to a
a += b      # a = a + b

print(a)

# Output: 15
```

---

3. Python Comparison Operators

Here, the a list of different comparison operator available in Python

Operator	Meaning	Example
`==`	Is Equal To	`3 == 5` gives us `False`
`!=`	Not Equal To	`3 != 5` gives us `True`
`>`	Greater Than	`3 > 5` gives us `False`
`<`	Less Than	`3 < 5` gives us `True`
`>=`	Greater Than or Equal To	`3 >= 5` give us `False`
`<=`	Less Than or Equal To	`3 <= 5` gives us `True`

Example : Comparison Operators

```
a = 5

b = 2

# equal to operator
print('a == b =', a == b)

# not equal to operator
print('a != b =', a != b)

# greater than operator
print('a > b =', a > b)

# less than operator
print('a < b =', a < b)

# greater than or equal to operator
print('a >= b =', a >= b)

# less than or equal to operator
print('a <= b =', a <= b)
```

Output

```
a == b = False
a != b = True
a > b = True
a < b = False
a >= b = True
a <= b = False
```

---

4. Python Logical Operators

Logical operators are used to check whether an expression is `True` or `False`. They are used in decision-making. For example,

```
a = 5
b = 6

print((a > 2) and (b >= 6))    # True
```

Here, `and` is the logical operator **AND**. Since both `a > 2` and `b >= 6` are `True`, the result is True.

Operator	Example	Meaning
`and`	a **and** b	**Logical AND**: `True` only if both the operands are `True`
`or`	a **or** b	**Logical OR**: `True` if at least one of the operands is `True`
`not`	**not** a	**Logical NOT**: `True` if the operand is `False` and vice-versa.

Example: Logical Operators

```
# logical AND
print(True and True)     # True
print(True and False)    # False

# logical OR
print(True or False)     # True

# logical NOT
print(not True)          # False
```

Note: Here is the truth table for these logical operators.

---

5. Python Bitwise operators

Bitwise operators act on operands as if they were strings of binary digits. They operate bit by bit, hence the name.

For example, **2** is `10` in binary, and **7** is `111`.

In the table below: Let x = **10** (`0000 1010` in binary) and y = **4** (`0000 0100` in binary)

Operator	Meaning	Example
`&`	Bitwise AND	x & y = 0 (`0000 0000`)
`|`	Bitwise OR	x | y = 14 (`0000 1110`)
`~`	Bitwise NOT	~x = -11 (`1111 0101`)
`^`	Bitwise XOR	x ^ y = 14 (`0000 1110`)
`>>`	Bitwise right shift	x >> 2 = 2 (`0000 0010`)
`<<`	Bitwise left shift	x 0010 1000)

---

6. Python Special operators

Python language offers some special types of operators like the **identity** operator and the **membership** operator. They are described below with examples.

Identity operators

In Python, `is` and `is not` are used to check if two values are located at the same memory location.

It's important to note that having two variables with equal values doesn't necessarily mean they are identical.

Operator	Meaning	Example
`is`	`True` if the operands are identical (refer to the same object)	`x is True`
`is not`	`True` if the operands are not identical (do not refer to the same object)	`x is not True`

Example Identity operators in Python

```
x1 = 5
y1 = 5
x2 = 'Hello'
y2 = 'Hello'
x3 = [1,2,3]
y3 = [1,2,3]

print(x1 is not y1)  # prints False

print(x2 is y2)  # prints True

print(x3 is y3)  # prints False
```

Here, we see that x1 and y1 are integers of the same values, so they are equal as well as identical. The same is the case with x2 and y2 (strings).

But x3 and y3 are lists. They are equal but not identical. It is because the interpreter locates them separately in memory, although they are equal.

Membership operators

In Python, `in` and `not in` are the membership operators. They are used to test whether a value or variable is found in a sequence (string, list, tuple, set and dictionary).

In a dictionary, we can only test for the presence of a key, not the value.

Operator	Meaning	Example
`in`	`True` if value/variable is **found** in the sequence	`5 in x`
`not in`	`True` if value/variable is **not found** in the sequence	`5 not in x`

Example: Membership operators in Python

```
message = 'Hello world'
dict1 = {1:'a', 2:'b'}

# check if 'H' is present in message string
print('H' in message)  # prints True

# check if 'hello' is present in message string
print('hello' not in message)  # prints True

# check if '1' key is present in dict1
print(1 in dict1)  # prints True
```

```
# check if 'a' key is present in dict1
print('a' in dict1)  # prints False
```

Output

```
True
True
True
False
```

Here, `'H'` is in message, but `'hello'` is not present in message (remember, Python is case-sensitive).

Similarly, `1` is key, and `'a'` is the value in dictionary dict1. Hence, `'a' in y` returns `False`.