எங்கள் வாழ்வும் எங்கள் வளமும்
மங்காத தமிழ் என்று சங்கே முழங்கு … *புரட்சிக்கவி*

NOTICE

- We support open-source products to spread Technology to the mass.
- This is completely a FREE training course to provide introduction to Python language
- All materials / contents / images/ examples and logo used in this document are owned by the respective companies / websites. We use those contents for FREE teaching purposes only.
- We take utmost care to provide credits when ever we use materials from external source/s. If we missed to acknowledge

any content that we had used here, please feel free to inform us at info@DataScienceInTamil.com.

- All the programing examples in this document are for teaching purposes only.

Thanks to all the open source community and to the below websites: for further readings, please make use of the below websites

https://www.w3schools.com/python/python_datatypes.asp
Python Notes For Professionals.pdf – this is the book we follow
https://docs.python.org/3/tutorial/
https://docs.python.org/3.9/tutorial/index.html
https://www.geeksforgeeks.org/python-data-types/
https://www.programmersought.com/article/54325076363/
https://www.educative.io/edpresso/what-are-constants-in-python
https://phoenixnap.com/kb/python-data-types
https://www.tutorialspoint.com/python/comparison_operators_example.htm
https://www.educba.com/python-comparison-operators/
https://inderpsingh.blogspot.com/2019/09/PythonTutorial5.html
https://blog.simpliv.com/python-programming-operators-and-decision-making-statements/

What to cover today?

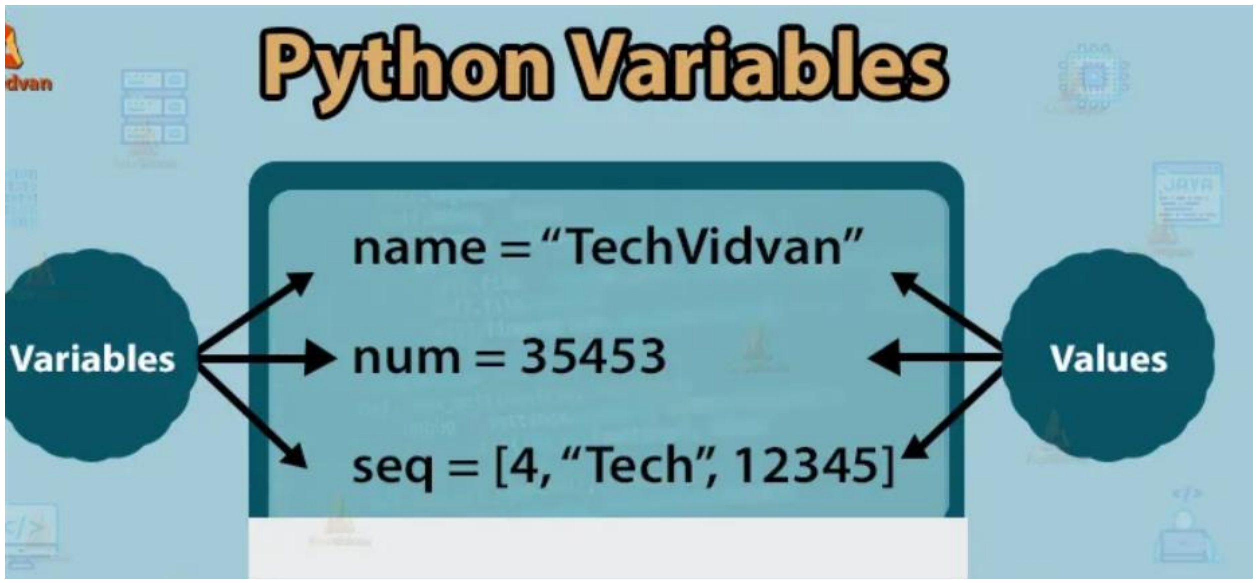
1. Python Fundamentals
2. What is variable
3. Rules for creating variables in Python
4. Re-declare the Variable
5. Assigning different values to multiple variables
6. Basic Data Types in Python
7. Example code for all data types
8. Python Keywords / Reserved Words
9. Keyword rules – when not to use

Python Operators

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators

- Membership operators
- Bitwise operators

WHAT IS VARIABLE?



```
name = "TechVidvan"
num = 35453
seq = [4, "Tech", 12345]
```

A Python variable is **a reserved memory location to store values**. In other words, a variable in a python program gives data to the computer for processing. Every value in Python has a datatype. Different data types in Python are int, float, bool, List, Tuple, Strings, Dictionary, set etc.

```
# An integer assignment
age = 45

# A floating point
salary = 1456.8

# A string
name = "John"

print(age)
print(salary)
print(name)
```

RULES FOR CREATING VARIABLES IN PYTHON

- ✓ A variable name must start with a letter or the underscore character.
- ✓ A variable name cannot start with a number.
- ✓ A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _).
- ✓ Variable names are case-sensitive (name, Name and NAME are three different variables).
- ✓ The reserved words(keywords) cannot be used naming the variable.
- ✓ It cannot have whitespace and signs like + and -, !, @, $, #, %.
- ✓ Variable names are case sensitive.

- ➢ Python variable names are case-sensitive. The variable ‘name’ is different than the variable ‘Name’.

- According to PEP8, you should name long variables names like this- long_variable_name with underscores.

Re-declare the Variable:

```
# declaring the var
Number = 100

# display
print("Before declare: ", Number)

# re-declare the var
Number = 120.3

print("After re-declare:", Number)
```

Assigning different values to multiple variables:

Python allows adding different values in a single line with “,”operators.

```
a, b, c = (1, 20.2, "Learn Python in Tamil")

print(a)
print(b)
print(c)
------------
```

Basic Data Types in Python

Integers – This value is represented by int class. It contains positive or negative whole numbers (without fraction or decimal). In Python there is no limit to how long an integer value can be.

Float – This value is represented by float class. It is a real number with floating point representation. It is specified by a decimal point

Complex Numbers – Complex number is represented by complex class. It is specified as *(real part) + (imaginary part)j*. For example – 2+3j

In plane geometry, complex numbers can be **used to represent points, and thus other geometric objects as well such as lines, circles, and polygons**

Python converts the real numbers x and y into complex using the function complex(x,y)

```
print(complex(2,4))
print(complex(10.1, 3.2))
```

output

```
(2+4j)
```

The Python complex() is a **built-in function that returns a complex number with real and imaginary values** (eg. real + imag*j).

String

In Python, Strings are arrays of bytes representing Unicode characters. A string is a collection of one or more characters put in a

- single quote,
- double-quote or
- triple quote.

In python there is no character data type, a character is a string of length one. It is represented by str class.

Multi-line strings can be denoted using triple quotes

```
name = "Data Science"
Name = 'Data Science'
university = """ Anna University"""
University = '''Anna University'''

location = '''
Anna University
is located
in Chennai'''

print(name)
print(Name)
print(university)
print(University)
print(location)
```

output

```
Data Science

Data Science
```

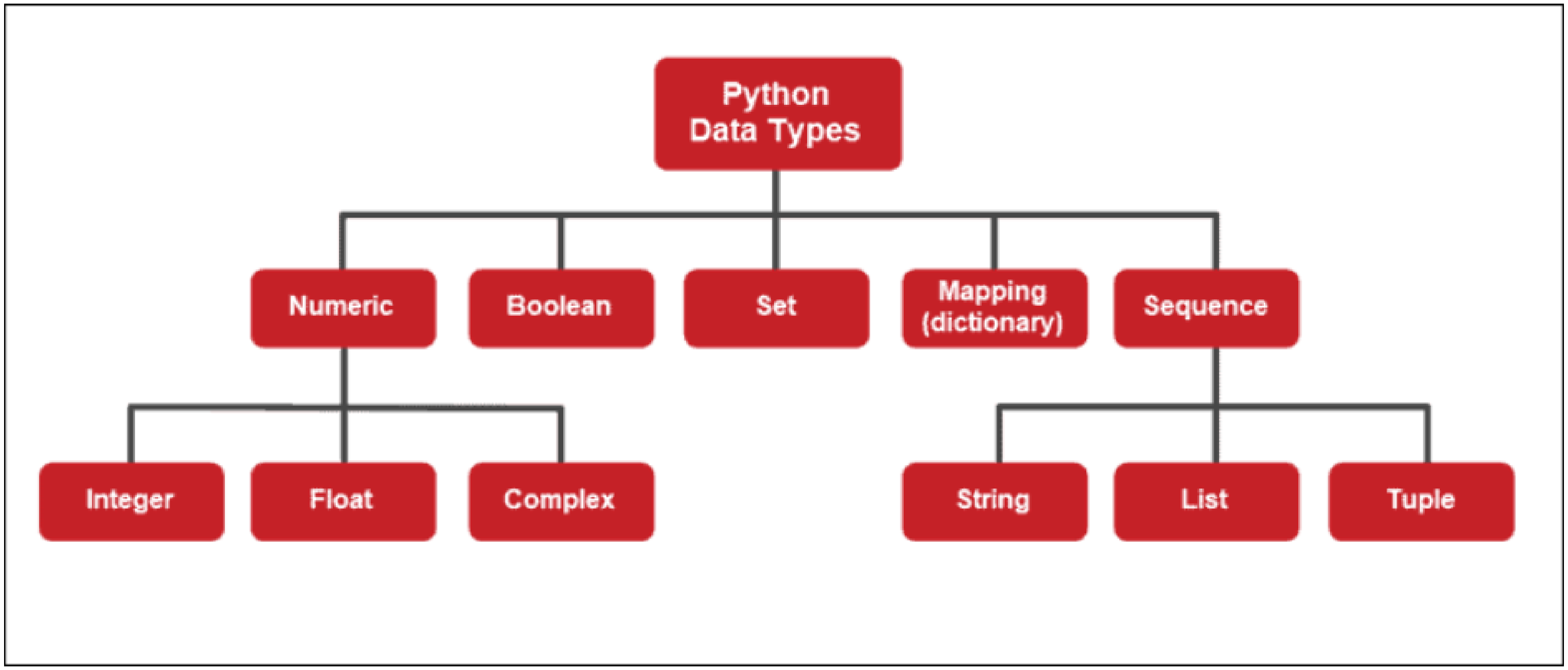
Anna University

Anna University

Anna University

is located

in Chennai



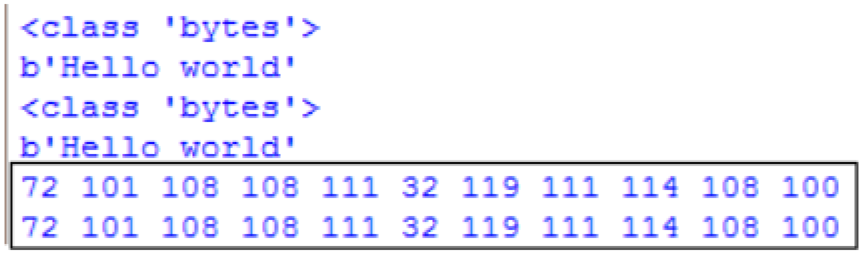
EXAMPLE CODE FOR ALL THE DATA TYPES

```
x = "Hello Python"  # str
print(x)
```

```
x = 21      #int
print(x)

x = 202.5  #float
print(x)

x = 3j     #complex
print(x)

x = True   #bool
print(x)

x = ["apple", "banana", "cherry"]  #list
print(x)

x = ("apple", "banana", "cherry") #    tuple
print(x)
```

```
x = range(6)  # range
print(x)

x = {"name" : "John", "age" : 36}   #dict
print(x)

x = {"apple", "banana", "cherry"}  #     set
print(x)

x = frozenset({"apple", "banana", "cherry"})    #frozenset
print(x)


x = b"Hello"    #bytes immutable
print(x)
```

One byte is **a memory location with a size of 8 bits**. A bytes object is an immutable sequence of bytes, conceptually similar to a string.
a string is a sequence of characters, byte is sequence of bits



```
<class 'bytes'>
b'Hello world'
<class 'bytes'>
b'Hello world'
72 101 108 108 111 32 119 111 114 108 100
72 101 108 108 111 32 119 111 114 108 100
```

----

bytearray() method returns a bytearray object which is **an array of given bytes**. It gives a mutable sequence of integers in the range 0 <= x < 256.

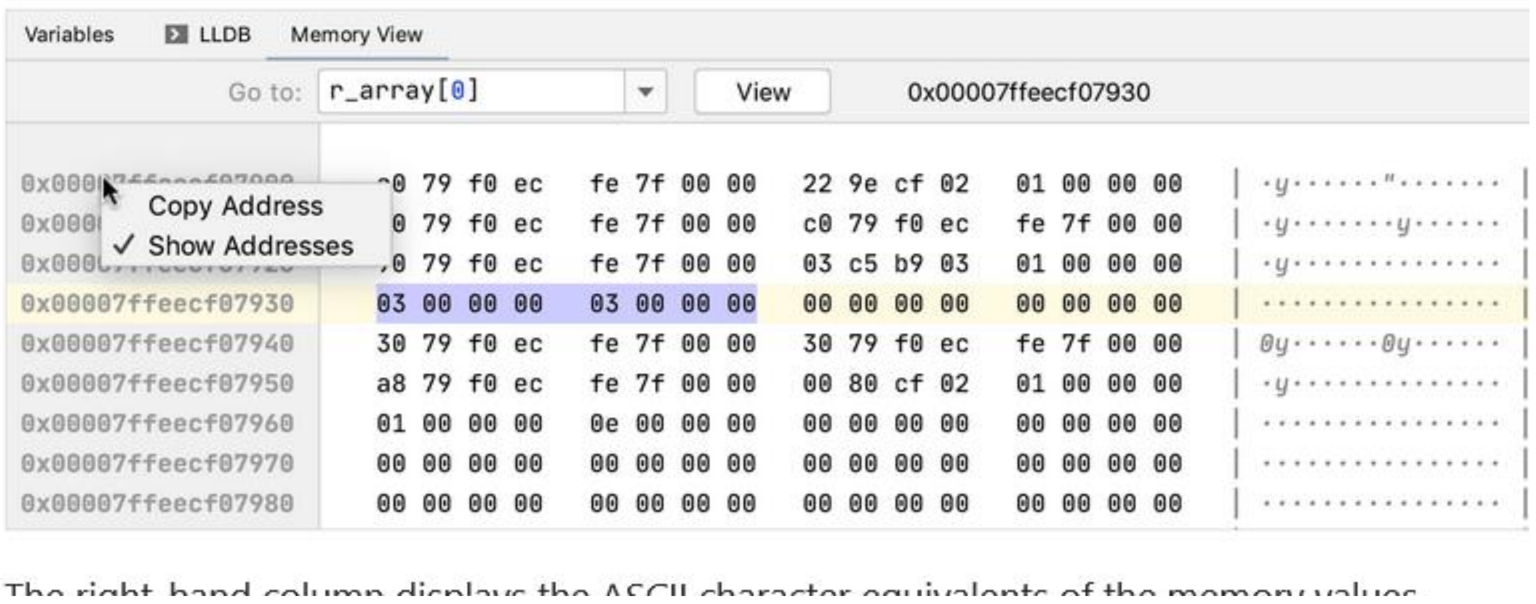
```
x = bytearray(5)    #bytearray mutable
print(x)

numbers = [10,20,3,5]
x = bytearray(numbers)    #bytearray
print(x)
output
bytearray(b'\n\x14\x03\x05')

numbers = [10,20,3,5,1000]
x = bytearray(numbers)    #bytearray
print(x)
output is error
    x = bytearray(numbers)    #bytearray
```

```
ValueError: byte must be in range(0, 256)
```

Additional Note:

This is useful because some applications use byte sequences in ways that perform poorly with immutable strings. When you are making lots of little changes in the middle of large chunks of memory, as in a database engine, or image library, strings perform quite poorly; since you have to make a copy of the whole (possibly large) string. `bytearray`s have the advantage of making it possible to make that kind of change without making a copy of the memory first.

```
x = memoryview(bytes(5))    #memoryview
print(x)
```



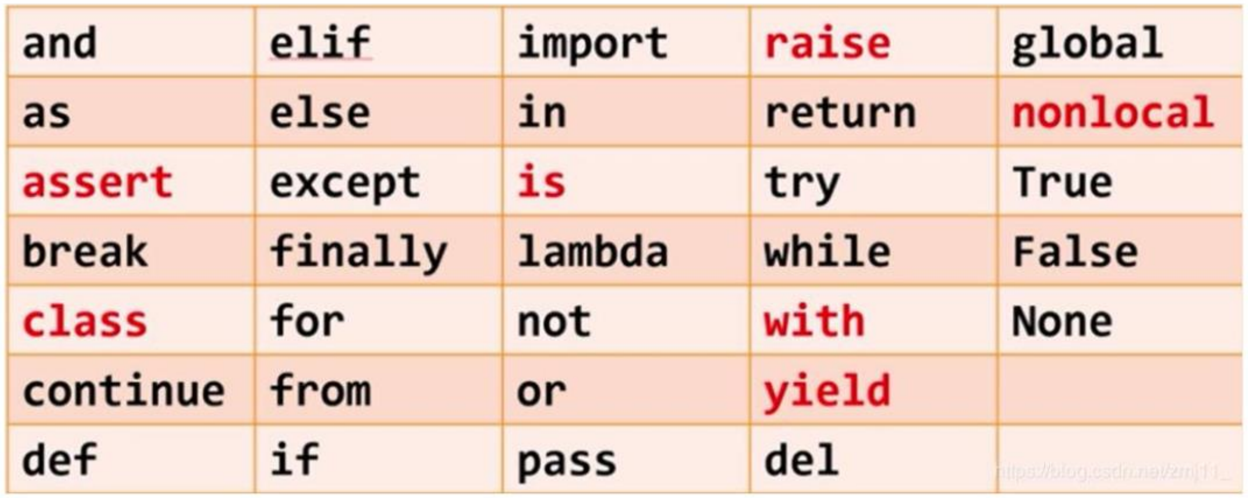
```
r_array[0]                                0x00007ffeecf07930

0x0000                 0 79 f0 ec   fe 7f 00 00   22 9e cf 02   01 00 00 00  | ·y······"······· |
0x0000                 0 79 f0 ec   fe 7f 00 00   c0 79 f0 ec   fe 7f 00 00  | ·y······y······· |
0x0000                 0 79 f0 ec   fe 7f 00 00   03 c5 b9 03   01 00 00 00  | ·y·············· |
0x00007ffeecf07930    03 00 00 00   03 00 00 00   00 00 00 00   00 00 00 00  | ················ |
0x00007ffeecf07940    30 79 f0 ec   fe 7f 00 00   30 79 f0 ec   fe 7f 00 00  | 0y······0y······ |
0x00007ffeecf07950    a8 79 f0 ec   fe 7f 00 00   00 80 cf 02   01 00 00 00  | ·y·············· |
0x00007ffeecf07960    01 00 00 00   0e 00 00 00   00 00 00 00   00 00 00 00  | ················ |
0x00007ffeecf07970    00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00  | ················ |
0x00007ffeecf07980    00 00 00 00   00 00 00 00   00 00 00 00   00 00 00 00  | ················ |
```

The right hand column displays the ASCII character equivalents of the memory values

https://www.jetbrains.com/help/clion/memory-view.html#invoke

Python Keywords / Reserved Words

The keywords are **some predefined and reserved words in python that have special meanings**. Keywords are used to define the syntax of the coding. The keyword cannot be used as an identifier, function, and variable name. All the keywords in python are written in lower case except True , False and None



and	elif	import	raise	global
as	else	in	return	nonlocal
assert	except	is	try	True
break	finally	lambda	while	False
class	for	not	with	None
continue	from	or	yield	
def	if	pass	del	

These are **35 reserved words**. They define the syntax and structure of Python. You cannot use a word from this list as a name for your variable or function.

In this list, all words except True, False, and None are in lowercase.

```
import keyword
print(keyword.kwlist)
print(len(keyword.kwlist))
```

output

```
['False', 'None', 'True', '__peg_parser__', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']

36
```

Python operators

Python divides the operators in the following groups:

1. Arithmetic operators
2. Assignment operators
3. Comparison operators
4. Logical operators
5. Identity operators

6. Membership operators
7. Bitwise operators

Arithmetic operators :Python Arithmetic Operators are used to perform basic math operations, which include addition, subtraction, and so on. The various operators are Subtraction, Division, Addition, Multiplication, Floor Division, Exponent, and Modulus.

Assignment operators : Python Assignment Operators are used to assign values to the variables. Various operators are +=, – = , *=, /= , etc.

Comparison operators : Python Comparison Operators are used to compare the values on both sides. Various operators are ==, != , <>, >,<=, etc.

Logical operators : Python Logical Operators are used for conditional statements. Various operators are Logical AND, Logical OR and Logical NOT.

Identity operators : Python Identity Operators are used for comparing the memory location of the two objects. The two identified operators used in Python are 'is' and 'is not.

Membership operators : Python Membership Operators are used to test the value, whether it is a member of a sequence or not. This sequence can be a list, tuple, or a string. The two identify operators used in Python are *‘in* and *not in’*.

Bitwise operators : Python Bitwise Operators works on bits and operates on operands bit by bit instead of whole. Various operators are –Python Bitwise AND, OR, XOR, Left-shift, Right-shift, and 1’s complement Bitwise Operator.

PYTHON ARITHMETIC OPERATORS

Arithmetic operators are used with numeric values to perform common mathematical operations:

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

PYTHON CODE EXAMPLE FOR ARITHMETIC OPERATORS

```
internal_Marks = 15
external_Marks = 82
number_of_subject = 5

print(internal_Marks + external_Marks)  # Addition

print(internal_Marks - external_Marks)  # Subtraction

print(internal_Marks *   external_Marks)  # Multiplication

print(internal_Marks / external_Marks)  # Division

print(internal_Marks  % external_Marks)  # Modulus

print(internal_Marks ** external_Marks)  # Exponentiation
```

```python
print(internal_Marks // external_Marks) # Floor division
```

PYTHON ASSIGNMENT OPERATORS

Assignment operators are used to assign values to variables

Operators are used to perform operations on values and variables. These are the special symbols that carry out arithmetic, logical, bitwise computations. The value the operator operates on is known as **Operand**.

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

PYTHON CODE EXAMPLES FOR ASSIGNMENT OPERATORS

Assign: This operator is used to assign the value of the right side of the expression to the left side operand.

```
internal_marks =15
print(internal_marks)

x = 5
```

Add and Assign: This operator is used to add the right side operand with the left side operand and then assigning the result to the left operand.

```
internal_marks =+5
print(internal_marks)

a = 3
b = 5

# a = a + b
a += b

# Output / Result will be
print(a)
```

Subtract and Assign: This operator is used to subtract the right operand from the left operand and then assigning the result to the left operand.

```
internal_marks=-3
print(internal_marks)
```

```
a = 3
b = 5

# a = a - b
a -= b

# Output / Result will be
print(a)
```

Multiply and Assign:

This operator is used to multiply the right operand with the left operand and then assigning the result to the left operand.

```
internal_marks *=5
print(internal_marks)

a = 3
b = 5

# a = a * b
```

```
a *= b

# Output / Result will be
print(a)
```

Divide and Assign: This operator is used to divide the left operand with the right operand and then assigning the result to the left operand.

```
a = 3
b = 5

# a = a / b
a /= b

# Output / Result will be
print(a)
```

Modulus and Assign: This operator is used to take the modulus using the left and the right operands and then assigning the result to the left operand.

```
a = 3
b = 5

# a = a % b
a %= b

# Output / Result will be
print(a)
```

Divide (floor) and Assign: This operator is used to divide the left operand with the right operand and then assigning the result(floor) to the left operand.

```
internal_marks //= 5
print(internal_marks)
```

```
a = 3
b = 5

# a = a // b
a //= b

# Output / Result will be
print(a)
```

Exponent and Assign: This operator is used to calculate the exponent(raise power) value using operands and then assigning the result to the left operand

```
internal_marks **= 5
print(internal_marks)

a = 3
b = 5

# a = a ** b
```

```
a **= b

# Output / Result will be
print(a)
```

Bitwise AND and Assign: This operator is used to perform Bitwise AND on both operands and then assigning the result to the left operand.

```
internal_marks &= 5
print(internal_marks)

a = 3
b = 5

# a = a & b
a &= b

# Output / Result will be
print(a)
```

Bitwise OR and Assign: This operator is used to perform Bitwise OR on the operands and then assigning result to the left operand.

```
internal_marks |= 5
print(internal_marks)

a = 3
b = 5

# a = a | b
a |= b

# Output / Result will be
print(a)
```

Bitwise XOR and Assign: This operator is used to perform Bitwise XOR on the operands and then assigning result to the left operand.

```
internal_marks ^= 5
print(internal_marks)
```

```
a = 3
b = 5

# a = a ^ b
a ^= b

# Output / Result will be
print(a)
```

Bitwise Right Shift and Assign: This operator is used to perform Bitwise right shift on the operands and then assigning result to the left operand.

```
internal_marks >>= 5
print(internal_marks)

a = 3
b = 5

# a = a >> b
```

```
a >>= b

# Output / Result will be
print(a)
```

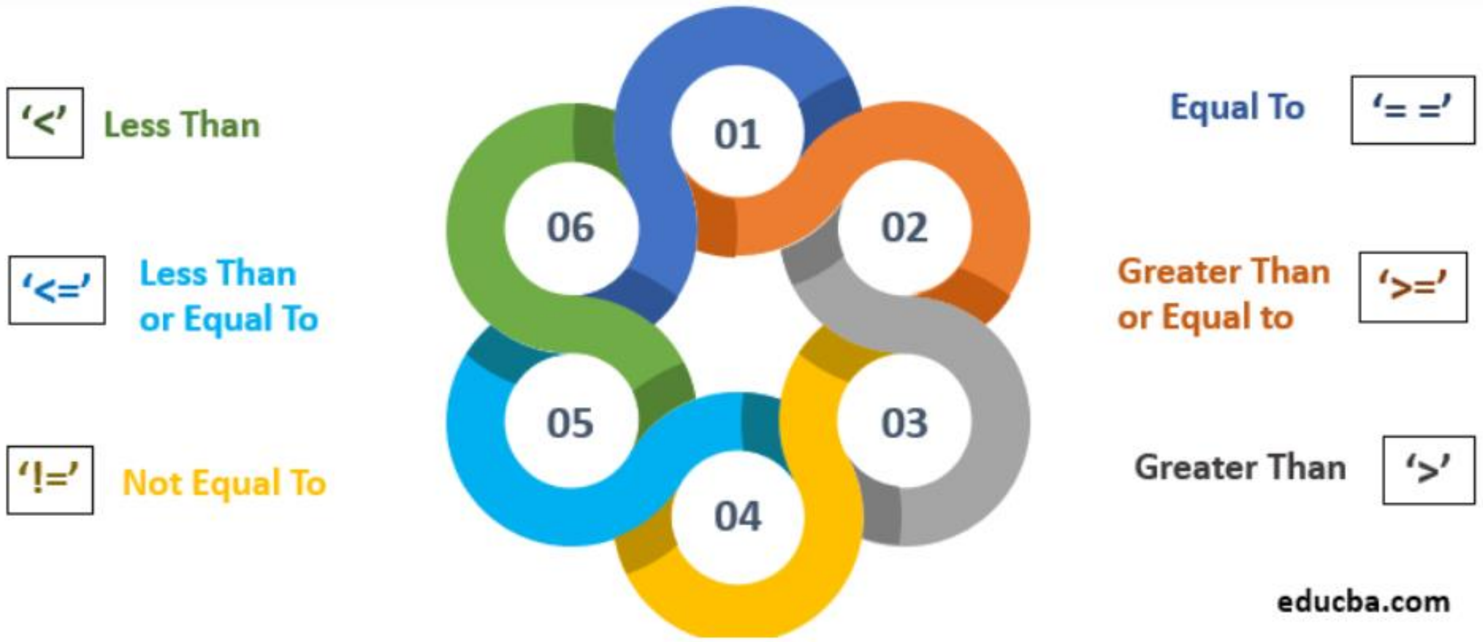
Bitwise Left Shift and Assign: This operator is used to perform Bitwise left shift on the operands and then assigning result to the left operand.

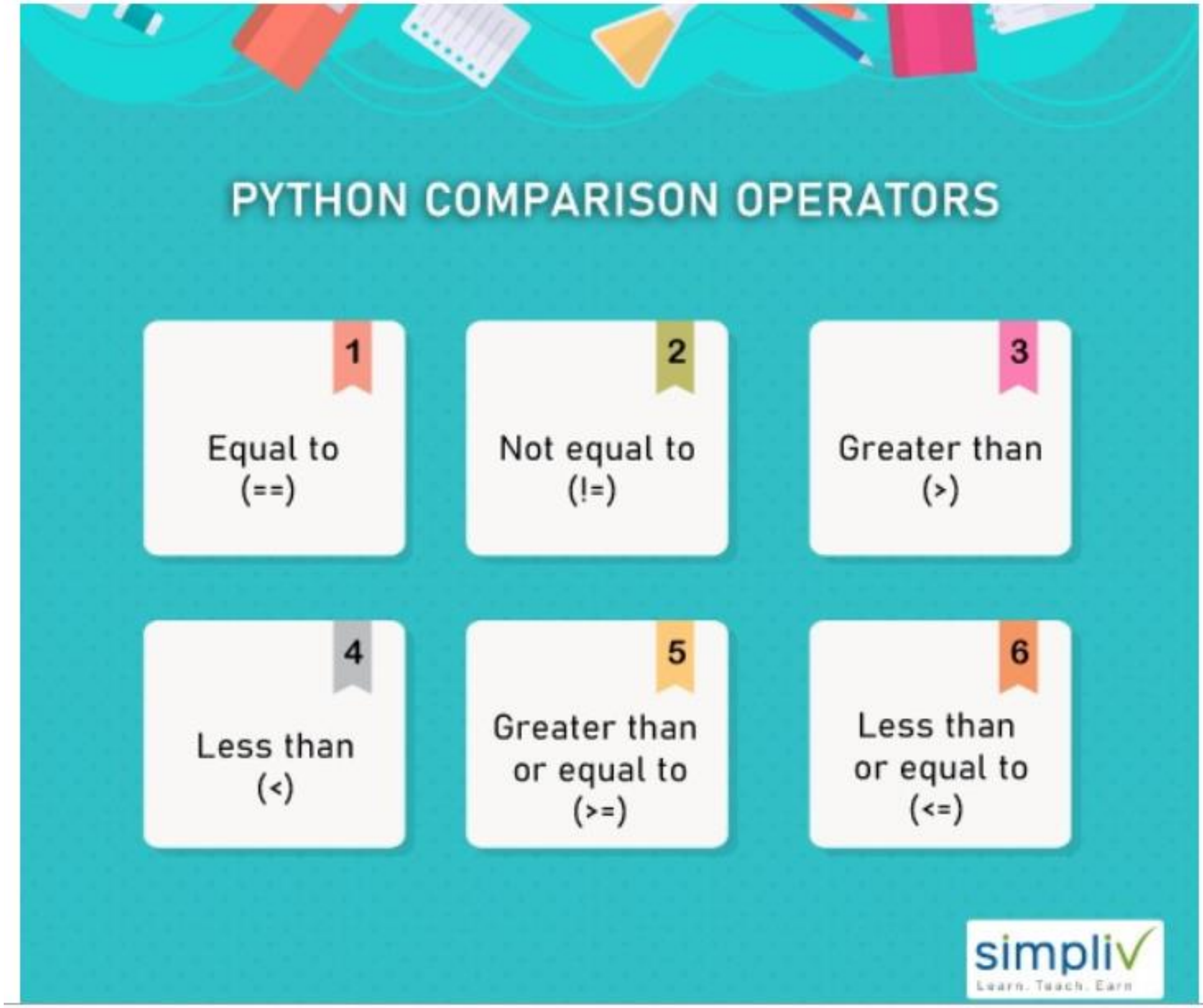
```
internal_marks <<= 5
print(internal_marks)


a = 3
b = 5

# a = a << b
a <<= b

# Output / Result will be
print(a)
```

PYTHON COMPARISON OPERATORS

Comparison operators are used to compare two values:

The comparison operators are also called as relational operators. These operators are used to compare the values and returns ‘True’ or ‘False’ based on the condition.


'<'
Less Than
'<='
Less Than or Equal To
'!='
Not Equal To
01
02
03
04
05
06
Equal To
'= ='
Greater Than or Equal to
'>='
Greater Than
'>'
educba.com



PYTHON COMPARISON OPERATORS

1 Equal to (==)	2 Not equal to (!=)	3 Greater than (>)
4 Less than (<)	5 Greater than or equal to (>=)	6 Less than or equal to (<=)

simpliv
Learn. Teach. Earn

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true.
!=	If values of two operands are not equal, then condition becomes true.	(a != b) is true.
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value	(a >= b) is not true.

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

PYTHON CODE EXAMPLES COMPARISON OPERATORS

```
salary = 7500
bonus = 7400
print(salary == bonus)
output
```

```
False
```

```
salary = 7500
bonus = 7400
print(salary != bonus)
output

True

-------------

salary = 7500
bonus = 7400
print(salary > bonus)
#True
----------------------
salary = 7500
bonus = 7400
print(salary < bonus)
#False
-------------------
```

```
salary = 7500
bonus = 7400
print(salary >= bonus)
#False
--------------
salary = 7500
bonus = 7400
print(salary <= bonus)
#True
--------------
```

PYTHON LOGICAL OPERATORS

Logical operators are used to combine conditional statements:

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Python - Logical Operators

- not

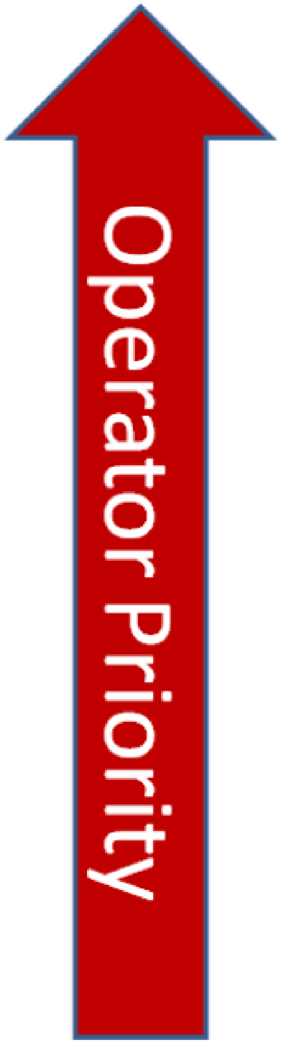
x	not x
False	True
True	False

- and

x	y	x and y
False	False	False
False	True	False
True	False	False
True	True	True

- or

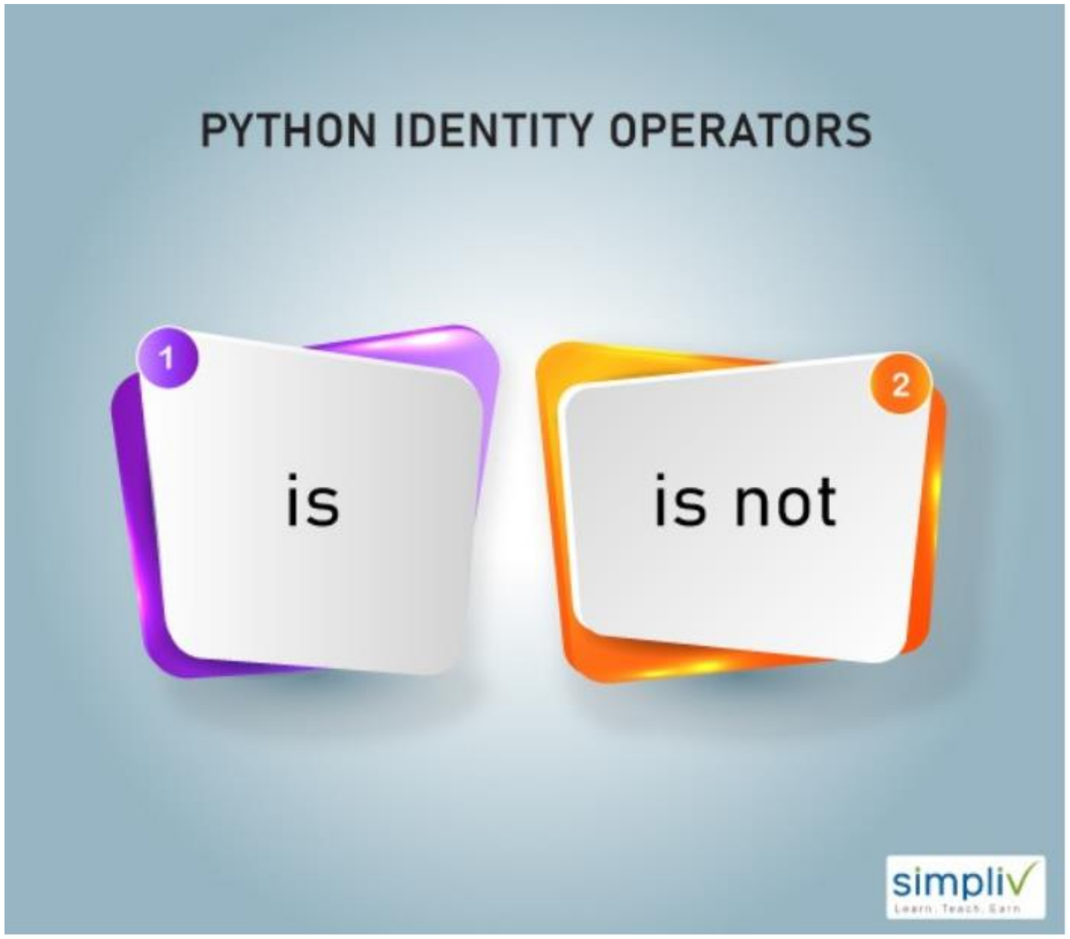
x	y	x or y
False	False	False
False	True	True
True	False	True
True	True	True



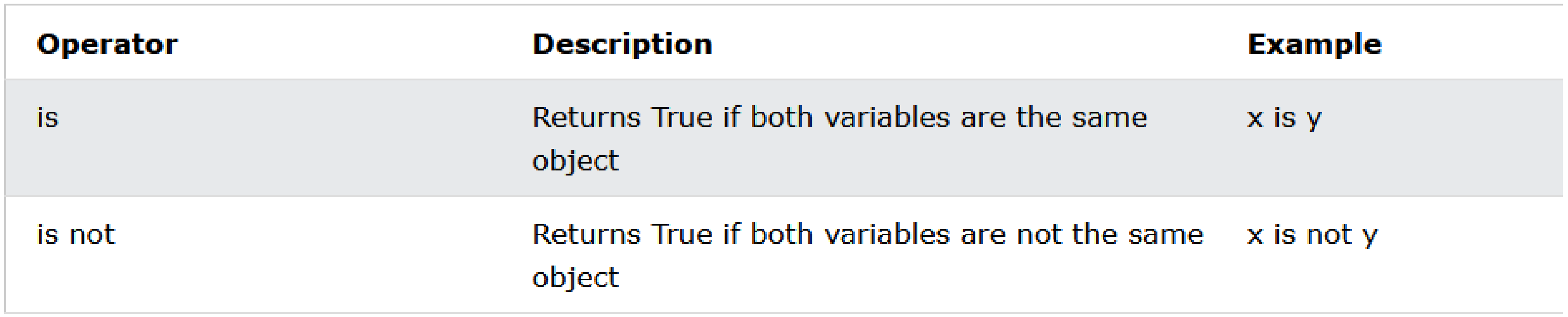
PYTHON CODE EXAMPLES FOR LOGICAL OPERATORS

```
x = 10
print(x < 5 and  x < 10)

print (x < 5 or x < 4)

print(not(x < 5 and x < 10))
output
False
False
True

------------------
```

PYTHON IDENTITY OPERATORS



Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:



Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

PYTHON CODE EXAMPLES FOR IDENTIFY OPERATORS

```
salary= 85202
bonus = 8

x2 = "Data Science"
y2 = "Data Science"

x3 = [4,5,690]
y3 = [4,5,690]
```

```
print(salary is not bonus)
print(x2 is y2)
print(x3 is y3)
output
True
True
False
```

PYTHON MEMBERSHIP OPERATORS

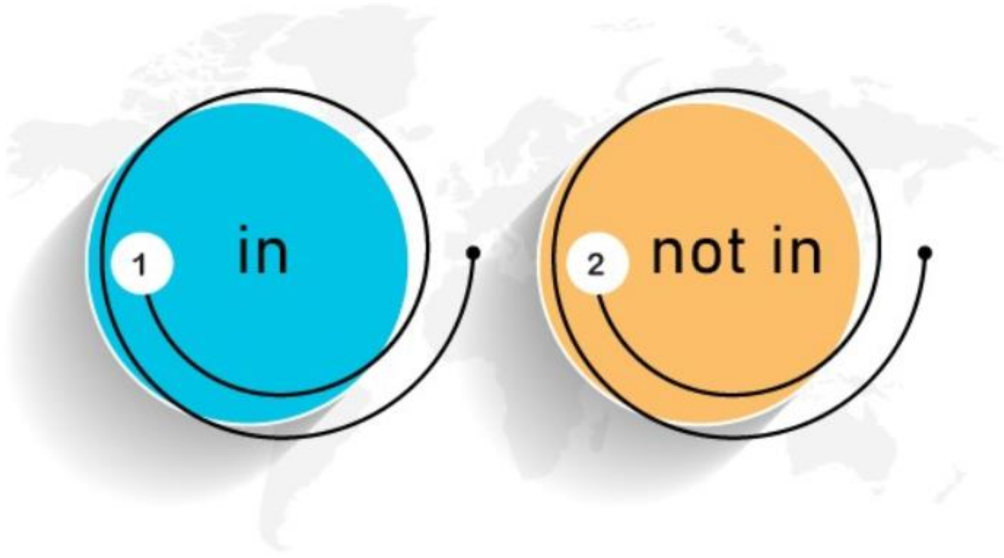
Membership operators are used to test if a sequence is presented in an object:

Python membership operators are used to check whether a value is a member of a sequence. Here the sequence may be a list, a string or a tuple.

There are two membership operators – **‘in’** and **‘not in’.**

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

PYTHON MEMBERSHIP OPERATORS



PYTHON CODE EXAMPLES FOR MEMBERSHIP OPERATORS

```
list_of_leaders = ["Jeeva", "Kakkan", "Kamaraj", "Nehru",
"Gandhi"]
print ("Jeeva" in list_of_leaders)
print ("AAA" in list_of_leaders)
print ("AAA" not in list_of_leaders)
```

What we covered today??