

Module-1

Data Types & Structures, Operators in Python

## Python Introduction

* Python is a general-purpose programming language that is often applied in scripting roles.
* Invented in the Netherlands, early 90s by Guido van Rossum.
* Open sourced from the beginning.

## What Python Can Do?

* Python can be used on a server to create web applications.
* Python can be used alongside software to create workflows.
* Python can connect to database systems. It can also read and modify files.
* Python can be used to handle big data and perform complex mathematics.
* Python can be used for rapid prototyping, or for production-ready software development.

## Python Is Used For-

* Web application
* Data Science
* Software Development
* Gaming
* Analytics
* Artificial Intelligence
* Gaming

## Why Python?

* Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
* Python has a simple syntax similar to the English language.
* Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
* Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
* Python can be treated in a procedural way, an object-orientated way or a functional way.
* Python is object-oriented
* Structure supports such concepts as polymorphism, operation overloading, and multiple inheritance.

## It's Powerful

* Dynamic Typing
* Built in type and tools
* Library Utilities
* Automatic Memory Management

## Python packages for Data Analysis:

Being a general-purpose language Python is often used beyond data analysis and data science. Abundant availability of libraries makes Python remarkably useful for working with data functionalities.

The significant Python libraries that are used for working with data:

* **Numpy** – this library provides fundamental scientific computing.
* **Matplotlib** – used for plotting and visualization.
* **Pandas** – applied for data manipulation and analysis.
* **Scikit-learn** – library designed for machine learning and data mining.
* **StatsModels** – packed with statistical modeling, testing, and analysis
* **Scipy**-SciPy is a bunch of mathematical algorithms and convenience functions built on the Numpy extension of Python.
* **Seaborn**-Seaborn is mostly used for the visualization of statistical models.
* **Plotly**-a web-based toolbox for constructing visualizations.

## Comments

* Comments can be used to explain Python code.
* Comments can be used to make the code more readable.
* Comments can be used to prevent execution when testing code.
* Comments are not part of program execution.
* **Single line comments:**

A single-line comment begins with a hash (#) symbol.

#This is single line comment

* **Multiple line comment:**

A multiple-line comment can be done using triple quotes (‘’’ ‘’’) symbol

"""This is also

multiple line

comment"""

## Keywords

Keywords in Python are reserved words that can not be used as a variable name, function name, or any other identifier.

There are around 36 keywords in Python:

['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']

## Identifiers

Python Identifier is the name we give to identify a variable, function, class, module or other object. That means whenever we want to give an entity a name, that's called identifier.

**Rules for defining identifiers:**

* We cannot use a keyword as identifier.
* Identifier has to begin with a letter or underscore (\_).
* It should not contain white space.
* Special characters are not allowed.
* Identifiers can consist of only letters, digits, or underscore.
* **Legal variable names:**
  + var = 'python'
  + var\_1 = 'python'
  + var1 = 'programming'
* **Illegal variable names:**
  + else = 10
  + 2value = 90
  + value 1 = 12
  + value@ = 30
  + value12#$ = 45

## Statements

A statement is an instruction that the Python interpreter can execute.

**Single line statement**

Example of Single line statement:

value1=10+20

**Multiple line statements**

Example of Multiple line statement:

value2=10+20 \

+30+40

## Indentation

* Indentation refers to the spaces at the beginning of a code line.
* It is very important as Python uses indentation to indicate a block of code.
* If the indentation is not correct we will end up with IndentationError.

## Variables

A Python variable is a reserved memory location to store values. A variable is created the moment you first assign a value to it.

**Variable Creation:**

x = 10

y = “Fingertips"

**Variable Overwriting:**

value = 100

value = value + 200

**Single value assignment:**

value1 = 100 # Integer variable

value2 = 89.78 # Float Variable

value3 = "Python" # String variable(" ") or (' ')

**Multiple value assignment:**

value1, value2, value3 = 220, 45.78, 'Machine Learning'

print(value1, value2, value3)

**Assigning one value to all the variables:**

val1 = val2 = val3 = 'coding'

print(val1, val2, val3)

**type(), input() and help() functions:**

* **type()**

The type() function is used to get the type of an object.

print(f'{type(45)}, {type(67.90)}, {type("python")}'))

**Output:**

<class 'int'>, <class 'float'>, <class 'str'>

* **input()**

Python input() function is used to take user input. By default, it returns the user input in form of a string.

name=input("Enter your name!") #input a string

age=input("Enter your age!") #input a number

print(f'Type of Name - {type(name)}\nType of Age - {type(age)}')

**Output:**

Enter your name!

Enter your age!

Type of Name - <class 'str'>

Type of Age - <class 'str'>

* **help():**

The Python help function is used to display the documentation of modules, functions, classes, keywords, etc.

help(print)

## Data Types

A data type is a classification of a value or variable that determines what kind of operations can be performed on it, and what type of value it can hold.

1. **Numeric**

A numeric value is any representation of data which has a numeric value.

Python identifies three types of numbers:

* **Integer**

int\_val = 10

type(int\_val)

**Output:**

<class 'int'>

* **Float**

float\_val = 89.67

type(float\_val)

**Output:**

<class 'float'>

* **Complex**

complex\_val = 55+44j

type(complex\_val)

**Output:**

<class 'complex'>

1. **Boolean**

Boolean data type can have only two possible values true or false.

bool1, bool2 = True, False

print(bool1, bool2)

print(type(bool1), type(bool2))

**Output:**

True False

<class 'bool'> <class 'bool'>

1. **Strings**

String literals in python are surrounded by either single quotation marks, or double quotation marks.

‘hello’ is the same as “hello”.

* **String Creation**

str1 = 'Hello World' # Define string using single quotes

str2 = "Hello! Python is easiest language to learn" # Define string using double quotes

str3 = '''Hello

World ''' # Define string using triple quotes

print(f'{str1}\n{str2}\n{str3}')

**Output:**

Hello World

Hello! Python is easiest language to learn

Hello

World

* **Length of a string**

str1 = 'Hello World'

len(str1)

**Output:**

11

* **String Indexing**

str1 = 'Hello World'

* **Forward Indexing:**

str1[6]

**Output:**

'W'

* **Backward Indexing:**

str1[-2]

**Output:**

'l'

* **String Slicing**

str2 = "Hello! Python is easiest language to learn"

print(str2[3:15:2])

**Output:**

'l!Pto '

print(str2[15:3:-1])

**Output:**

si nohtyP !o

* **String Concatenation**

str1 = "Hello "

str2 = "World!"

str3 = str1 + str2

print(str3)

**Output:**

Hello World!

* **Update and Delete String**

**Update:**

str1[0:3] = 'HEY'

**Output:**

TypeError: 'str' object does not support item assignment

**Delete:**

del str1

* **String Membership or Substring**

str4 = "Hey all, python Lovers!"

print ('Hey' in str4)

print ('Everyone' in str4)

print ('lovers' in str4)

**Output:**

True

False

False

* **String Functions**

s='Hey all Python lovers here!'

s.lower()

**Output:**

'hey all python lovers here!'

s.title()

**Output:**

'Hey All Python Lovers Here!'

s.capitalize()

**Output:**

'Hey all python lovers here!'

s.split()

**Output:**

['Hey', 'all', 'Python', 'lovers', 'here!']

s.swapcase()

**Output:**

'hEY ALL pYTHON LOVERS HERE!'

s=' hello coders '

s.strip()

**Output:**

'hello coders'

* **check inbuilt methods-->returns True/False**

m='String check!'

m.isupper()

**Output:**

False

m.isalpha()

**Output:**

False

## Data Structures

1. **List**

A list is a collection of items that are ordered and changeable. Lists are written with square brackets []. They allows us to store multiple items in a single place, and you can access, add, remove, and modify elements of a list using various methods and operations.

You can also use different operations and methods with lists like:

* Accessing an item using its index
* Modifying an item using its index
* Adding an item to a list
* Removing an item from a list
* Counting the number of items in a list
* Sorting items in a list
* Reversing items in a list
* **List Creation**

**List can hold homogeneous data type**

list1=['python', 'SQL', 'ML', 'DL', 'NLP']

list1

**Output:**

['python', 'SQL', 'ML', 'DL', 'NLP']

**List can hold heterogeneous data type**

list2=['python', 23, 45.6, 56+2j, ['SQL', 657, 78.5], 'ML', 45]

list2

**Output:**

['python', 23, 45.6, (56+2j), ['SQL', 657, 78.5], 'ML', 45]

* **List Indexing**

list2=['python', 23, 45.6, 56+2j, ['SQL', 657, 78.5], 'ML', 45

**Forward indexing**

print(list2[4])

print(list2[4][0])

**Output:**

['SQL', 657, 78.5]

SQL

**Backward indexing**

list2[-3]

**Output:**

['SQL', 657, 78.5]

* **List Slicing**

list3 = ['zero', 'one' , 'two' , 'three' , 'four' , 'five' , 'six' , 'seven' , 'eight']

**List all items from 2nd to 5th index**

list3[2:6]

**Output:**

['two', 'three', 'four', 'five']

**Return last two items**

list3[-2:]

**Output:**

['seven', 'eight']

* **Add, Remove Items in List**

**Add Items:**

list3.append('nine') # Add an item to the end of the list

list3

**Output:**

['zero',

'one',

'two',

'three',

'four',

'five',

'six',

'seven',

'eight',

'nine']

**Remove Items:**

list3.remove('ONE') # Remove item "ONE"

list3

**Output:**

['zero',

'one',

'two',

'three',

'four',

'five',

'six',

'seven',

'eight',

'nine']

* **Copy List**

list4 = ['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine']

list5 = list4

list6 = list4.copy()

list4[0] = 1

print(list4)

print(list5)

print(list6)

**Output:**

[1, 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine']

[1, 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine']

['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine']

* **Join List**

list1 = ['one', 'two', 'three', 'four']

list2 = ['five', 'six', 'seven', 'eight']

list3 = list1 + list2 # Join two lists by '+' operator

list3

**Output:**

['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight']

* **List Membership**

list1= ['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight']

print('one' in list1)

print('ten' in list1)

**Output:**

True

False

* **Reverse and Sort List**

**Reverse:**

list1= ['one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight']

list1 = list1[::-1] # Reverse the list

list1

OR

list1.reverse() # Reverse the list

list1

**Output:**

['eight', 'seven', 'six', 'five', 'four', 'three', 'two', 'one']

**Sorting:**

list3 = [34,56,32,67,56,90,78,32]

list3.sort(reverse=True) # Sort list in descending order

list3

**Output:**

[90, 78, 67, 56, 56, 34, 32, 32]

* **Count**

list5 =['one', 'two', 'three', 'four', 'one', 'one', 'two', 'three']

list5.count('one') # Number of times item "one" occurred in the list.

**Output:**

3

* **List Comprehension**

**Multiple whole list by 20**

list1 = [2,3,4,5,6,7,8]

list1 = [i\*20 for i in list1]

list1

**Output:**

[40, 60, 80, 100, 120, 140, 160]

1. **Tuple**

Tuples are similar to lists in that they allow you to store multiple items in a single place, however, unlike lists, the items in a tuple cannot be modified once they are created. This means that you cannot add, remove, or change the values of items in a tuple.

You can also use different operations and methods with tuples like:

* Accessing an item using its index
* Counting the number of items in a tuple
* Slicing a tuple
* **Tuple Creation**

tuple0 = () # Empty tuple

tuple1 = (1,)

tuple2 = (45, 67, 90) # tuple of integers numbers

tuple3 = (45.89,56.78, 23.99) # tuple of float numbers

tuple4 = ('one','two' , "three") # tuple of strings

tuple5 = ('Python', 11 ,(45, 'sql'),(555, 78.987)) # Nested tuples

tuple6 = (112, 'SQL', 17.765) # Tuple of mixed data types

tuple7 = ('Python', 85 ,[890, 189],[19.750, 790] , {'ML', 'NLP'} , (99, 'coders'))

print(f'{type(tuple0)},{type(tuple1)},{type(tuple2)},{type(tuple3)},{type(tuple4)},{type(tuple5)},{type(tuple6)},{type(tuple7)}')

print(f'{len(tuple0)},{len(tuple1)},{len(tuple2)},{len(tuple3)},{len(tuple4)},{len(tuple5)},{len(tuple6)},{len(tuple7)}')

**Output:**

<class 'tuple'>,<class 'tuple'>,<class 'tuple'>,<class 'tuple'>,<class 'tuple'>,<class 'tuple'>,<class 'tuple'>,<class 'tuple'>

0,1,3,3,3,4,3,6

* **Tuple Indexing**

print(tuple2[0]) # Retreive first element of the tuple

print(tuple7[2][0]) # Nested indexing - Access the first character of the first tuple elements

print(tuple4[-1]) # Last item of the tuple

**Output:**

45

890

three

* **Tuple Slicing**

tuple1 = ('one' , 'two' , 'three' , 'four' , 'five' , 'six' , 'seven' , 'eight')

print(tuple1[2:5])

print(tuple1[:3])

print(tuple1[-3:])

print(tuple1[-1])

print(tuple1[:])

**Output:**

('three', 'four', 'five')

('one', 'two', 'three')

('six', 'seven', 'eight')

eight

('one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight')

* **Remove and Change items**

**Remove Items:**

tuple1= ('one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight')

del tuple1[0]

**Output:**

TypeError: 'tuple' object doesn't support item deletion

**Change Items:**

tuple1[0] = 1

**Output:**

TypeError: 'tuple' object does not support item assignment

* **Join Tuples**

tuple1 = ("a", "b" , "c")

tuple2 = (1, 2, 3)

tuple3 = tuple1 + tuple2

print(tuple3)

**Output:**

('a', 'b', 'c', 1, 2, 3)

* **Tuple Membership**

tuple1=('one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight')

print('one' in tuple1) # Check if 'one' exist in the list

print('ten' in tuple1) # Check if 'ten' exist in the list

**Output:**

True

False

* **Index Position**

tuple1=('one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight')

tuple1.index('five') # Index of first element equal to 'five'

**Output:**

4

* **Sorting**

tuple3 = (56,322,567,89,900,56)

print(sorted(tuple3)) # Returns a new sorted list and doesn't change original tuple

print(sorted(tuple3, reverse=True)) # Sort in descending order

**Output:**

[*56, 56, 89, 322, 567, 900]*

*[900, 567, 322, 89, 56, 56]*

* **Count**

tuple1 =('one', 'two', 'three', 'four', 'one', 'one', 'two', 'three')

tuple1.count('one') # Number of times item "one" occurred in the tuple.

**Output:**

3

1. **Sets**

Sets are similar to lists and tuples in that they allow you to store multiple items in a single place, however, unlike lists and tuples, sets do not have duplicate items and do not maintain any order of the items. This makes sets useful for performing operations such as mathematical set operations like union, intersection, difference, etc. Sets are also useful for removing duplicates from a list or checking membership of an item in a list.

You can also use different operations and methods with sets like:

* Adding an item to a set
* Removing an item from a set
* Checking if an item exists in a set
* Finding the length of a set
* Finding the union, intersection, difference, etc. of two sets
* **Set Creation**

set1 = {1,2,3,4,5}

set1

**Output:**

{1, 2, 3, 4, 5}

set2 = set(('one' , 'two' , 'three' , 'four'))

set2

**Output:**

{'four', 'one', 'three', 'two'}

* **Set Membership**

set3 = {34.567, 45, "python lovers", (23, 67.98, 'code')}

print(45 in set3 ) # Check if 45 exist in the set

print('code' in set3) # Check if 'code' exist in the set

**Output:**

True

False

* **Add and Remove Items**

**Add Items:**

set4 = {'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}

set4.add('NINE') # Add item to a set using add() method

set4

**Output:**

{'NINE', 'eight', 'five', 'four', 'one', 'seven', 'six', 'three', 'two'}

**Remove Items:**

set4.remove('NINE') # remove item in a set using remove() method

set4

**Output:**

{'ELEVEN',

'TEN',

'TWELVE',

'eight',

'five',

'four',

'one',

'seven',

'six',

'three',

'two'}

* **Copy Set**

set1 = {'one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight'}

set2 = set1

print(id(set1) , id(set2))

set3 = set1.copy()

print(id(set3))

set1.add('nine')

print(set1)

print(set2)

print(set3

**Output:**

2693440767136 2693440767136

2693440769152

{'one', 'four', 'nine', 'two', 'eight', 'seven', 'six', 'three', 'five'}

{'one', 'four', 'nine', 'two', 'eight', 'seven', 'six', 'three', 'five'}

{'one', 'four', 'two', 'eight', 'seven', 'six', 'three', 'five'}

* **Set Operation**

A = {1,2,3,4,5}

B = {4,5,6,7,8}

**Union:**

A.union(B)

**Output:**

{1, 2, 3, 4, 5, 6, 7, 8}

**Intersection:**

A.intersection(B)

**Output:**

{4, 5}

**Difference:**

B.difference(A)

**Output:**

{6, 7, 8}

**Symmetric Difference:**

A.symmetric\_difference(B)

**Output:**

{1, 2, 3, 6, 7, 8}

* **Subset, Superset, Disjoint**

A = {1,2,3,4,5,6,7,8,9}

B = {3,4,5,6,7,8}

C = {10,20,30,40}

print(B.issubset(A)) # Set B is said to be the subset of set A if all elements of B are

print(A.issuperset(B)) # Set A is said to be the superset of set B if all elements of B

print(C.isdisjoint(A)) # Two sets are said to be disjoint sets if they have no common elements

print(B.isdisjoint(A)) # Two sets are said to be disjoint sets if they have no common elements

**Output:**

True

True

True

False

* **Built-in Functions**

A= {1, 2, 3, 4, 5, 6, 7, 8, 9}

print(sum(A))

print(max(A))

print(min(A))

print(len(A))

print(list(enumerate(A)))

**Output:**

45

9

1

9

[(0, 1), (1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 7), (7, 8), (8, 9)]

1. **Dictionary**

Dictionaries are similar to lists and tuples in that they allow you to store multiple items in a single place, but unlike lists and tuples, you can't access the items in a dictionary by their position, instead, you use the keys to access the values.

You can also use different operations and methods with dictionaries like:

* Accessing an item using its key
* Modifying an item using its key
* Adding an item to a dictionary
* Removing an item from a dictionary
* **Create Dictionary**

**Create dictionary with integer keys**

dict1 = {1:'one' , 2:'two' , 3:'three'}

dict1

**Output:**

{1: 'one', 2: 'two', 3: 'three'}

**Create dictionary using dict()**

dict1 = dict({1:'one' , 2:'two' , 3:'three'})

dict1

**Output:**

{1: 'one', 2: 'two', 3: 'three'}

* **Nested Dictionary**

dict2 = {1:'one' , 2:'two' , 'A':['python', 'sql', 'ml'], 'B':{'course1':'dl', 'course2':'nlp'}}

dict2

**Output:**

{1: 'one',

2: 'two',

'A': ['python', 'sql', 'ml'],

'B': {'course1': 'dl', 'course2': 'nlp'}}

* **Accessing Items**

dict1 = {1:'one' , 2:'two' , 3:'three' , 4:'four'}

dict1[1]

**Output:**

'one'

* **Change and Remove Items**

**Change Item:**

dict1 = {'course1':'dl', 'course2':'nlp', 'number':345}

dict1['course1'] = 'python core'

dict1['number'] = 1232

dict1

**Output:**

{'course1': 'python core', 'course2': 'nlp', 'number': 1232}

**Remove Item:**

dict1= {'course1': 'python advanced', 'course2': 'MySQL', 'job role': 'Data Analyst'}

dict1.pop('job role') # Removing items in the dictionary using Pop method

dict1

**Output:**

{'course1': 'python advanced', 'course2': 'MySQL'}

**Delete Dictionary:**

del dict1

* **Copy Dictionary**

dict1 = {'course1': 'python advanced', 'course2': 'MySQL', 'job role': 'Data Analyst'}

dict2 = dict1

print(id(dict1) , id(dict2))

dict3 = dict1.copy()

print(id(dict3))

dict1['course1'] = 'ML'

print(dict1)

print(dict2)

print(dict3)

**Output:**

2693422856448 2693422856448

2693422953792

{'course1': 'ML', 'course2': 'MySQL', 'job role': 'Data Analyst'}

{'course1': 'ML', 'course2': 'MySQL', 'job role': 'Data Analyst'}

{'course1': 'python advanced', 'course2': 'MySQL', 'job role': 'Data Analyst'}

* **Dictionary Membership**

dict1 = {'course1': 'python advanced', 'course2': 'MySQL', 'job role': 'Data Analyst'}

print('job role' in dict1)

print('MySQL' in dict1)

**Output:**

True

False

* **Dictionary Comprehension**

**Squares of numbers from 1 to 10**

square = {i:i\*\*2 for i in range(1, 11)}

square

**Output:**

{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}

## Type Conversion and Type Casting

1. **Type Conversion**

This is also known as automatic type casting. It occurs when the interpreter automatically converts a value to a different data type based on the context in which it is used.

**Upcasing** (converts overall result into higher memory allocation variable type)

1+2.0

**Output:**

3.0

**Concatenation**

'python'+'python'+'python'

**Output:**

'pythonpythonpython'

**Or,**

'python'\*3

**Output:**

'pythonpythonpython'

1. **Type Castings**

Explicit type conversion: This is also known as manual type casting. It is done by using built-in functions such as int(), float(), str(), list(), tuple(), set(), dict(), etc.

**String concatenation**

'426'+'coding'

**Output:**

'426coding'

**float to int**

int(20.7)

**Output:**

20

**int to float**

float(7)

**Output:**

7.0

## Operators

1. **Arithmetic operators**

Arithmetic operators are used to perform mathematical operations such as addition, subtraction, multiplication, division, and modulus. These operators are used to manipulate numerical values in a program.

**Example:**

a = 13

b = 12

x = 'Hello '

y = 'Coders!'

**Addition:**

c = a + b

print('Addition of {} and {} will give : {}\n'.format(a,b,c))

#Concatenate string using plus operator

z = x+y

print ('Concatenate string \'x\' and \'y\' using \'+\' operaotr : {}\n'.format(z))

**Subtraction:**

c = a - b

print('Subtracting {} from {} will give : {}\n'.format(b,a,c))

**Multiplication:**

c = a \* b

print('Multiplying {} and {} will give : {}\n'.format(a,b,c))

**Division:**

c = a / b

print('Dividing {} by {} will give : {}\n'.format(a,b,c))

**Modulus:**

c = a % b

print('Modulo of {} , {} will give : {}\n'.format(a,b,c))

**Power:**

c = a \*\* b

print('{} raised to the power {} will give : {}\n'.format(a,b,c))

# Division(floor)

c = a // b

print('Floor division of {} by {} will give : {}\n'.format(a,b,c))

**Output:**

Addition of 13 and 12 will give : 25

Concatenate string 'x' and 'y' using '+' operaotr : Hello Coders!

Subtracting 12 from 13 will give : 1

Multiplying 13 and 12 will give : 156

Dividing 13 by 12 will give : 1.0833333333333333

Modulo of 13 , 12 will give : 1

13 raised to the power 12 will give : 23298085122481

Floor division of 13 by 12 will give : 1

1. **Comparison operators**

Comparison operators are used to compare two or more values and determine their relationship. They return a Boolean value of either True or False based on the comparison made.

**Example:**

x = 13

y = 15

print('Is x greater than y : ',x>y)

print('\nIs x less than y : ',x<y)

print('\nIs x equal to y : ',x==y)

print('\nIs x not equal to y : ',x!=y)

print('\nIs x greater than or equal to y : ',x>=y)

print('\nIs x less than or equal to y : ',x<=y)

**Output:**

Is x greater than y : False

Is x less than y : True

Is x equal to y : False

Is x not equal to y : True

Is x greater than or equal to y : False

Is x less than or equal to y : True

1. **Logical operators**

Logical operators are used to combine or manipulate Boolean values (True or False). They are used to create more complex statements by combining simple statements using logical connectives such as AND, OR, NOT.

**Example:**

x = True

y = False

print('Logical AND operation : ',x and y) # True if both values are true

print('Logical OR operation : ',x or y) # True if either of the values is true

print('NOT operation : ',not x ) # True if operand is false

**Output:**

Logical AND operation : False

Logical OR operation : True

NOT operation : False

1. **Bitwise operators**

Bitwise operators are used to perform operations on binary representations of numbers. These operators work on the individual bits of a number, rather than the number as a whole.

**Example:**

x = 65

y = 34

print('Bitwise AND operation - {}'.format(x&y))

print('Bitwise OR operation - {}'.format(x|y))

print('Bitwise XOR operation - {}'.format(x^y))

print('Bitwise NOT operation - {}'.format(~x))

print('Bitwise right shift operation - {}'.format(x>>2))

print('Bitwise left shift operation - {}'.format(x<<2))

**Output:**

Bitwise AND operation - 0

Bitwise OR operation - 99

Bitwise XOR operation - 99

Bitwise NOT operation - -66

Bitwise right shift operation - 16

Bitwise left shift operation - 260

1. **Assignment operators**

Assignment operators are used to assign a value to a variable. They are used to set the value of a variable, or to update its value.

**Example:**

x = 67

print('Initialize x with value : ',x)

**Output:**

Initialize x with value : 67

1. **Membership operators**

The membership operator is used to test whether a value is present in a sequence, such as a string, list, tuple, or set. The membership operator returns a Boolean value of either True or False depending on whether the value is found in the sequence or not.

The membership operators in Python are:

* In
* Not in

**Example:**

string = 'Hey all, python lovers!'

'python' **in** string, 'all' **not in** string

**Output:**

(True, False)