# **Python**

1.Python is powerful, high level programming language which is platform independent.

2.It’s used in various applications including Data Analysis, AI, Web development, etc.

**SYNTAX**

**a=’hello world’ ##**here a variable is assigned to a string

**print(a) ##**printing statment

**Output:**

**hello world**

**Example :**

***# define main function to print out something***

def main():

i = 1

max = 10

while (i < max):

print(i)

i = i + 1

***# call function main***

main()

* Python Statement - statement is an instruction that a Python interpreter can execute
* Python Comments - Comments describe what is happening inside a program
* Python Key words -Python keywords are reserved words that have a special meaning.

Python keywords are **case-sensitive**.

'False', 'None', 'True', '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'

**Code to find the key words:**

import keyword

print(keyword.kwlist)

**Python Variables**

* A variable is a reserved memory area (memory address) to store value.
* A variable can be either mutable or immutable
* Local variable and global variable

Code :

a=5

Here **a** is a variable

**Python Operators**

1. Arithmetic operator
2. Relational operators
3. Assignment operators
4. Logical operators
5. Membership operators
6. Identity operators
7. Bitwise operators

**Arithmetic operator**

* [Addition operator +](https://pynative.com/python-operators/#h-addition-operator)
* [Subtraction –](https://pynative.com/python-operators/#h-subtraction)
* [Multiplication \*](https://pynative.com/python-operators/#h-multiplication)
* [Division /](https://pynative.com/python-operators/#h-division)
* [Floor division //](https://pynative.com/python-operators/#h-floor-division) - gives the quotient without decimals
* [Modulus ℅](https://pynative.com/python-operators/#h-modulus) -gives the remainder as the result
* [Exponent \*\*](https://pynative.com/python-operators/#h-exponent) - raised to the power values

**Relational operators**

Relational operators are also called comparison operators. It performs a comparison between two values. It returns a boolean True or False depending upon the result of the comparison.

* + (Greater than) - It returns **True** if the left operand is greater than the right

|  |  |
| --- | --- |
| * + < (Less than) - It returns True if the left operand is less than the right   + == (Equal to) -It returns True if both operands are equal   + ! = (Not equal to) - It returns True if both operands are equal   + >= (Greater than or equal to) - It returns True if the left operand is greater than or equal to the right   + <= (Less than or equal to) - It returns True if the left operand is less than or equal to the right |  |

Example :

x = 10

y = 5

z = 2

**# > Greater than**

print(x > y) *# True*

print(x > y > z) *# True*

**# < Less than**

print(x < y) *# False*

print(y < x) *# True*

**# Equal to**

print(x == y) *# False*

print(x == 10) *# True*

**# != Not Equal to**

print(x != y) *# True*

print(10 != x) *# False*

**# >= Greater than equal to**

print(x >= y) *# True*

print(10 >= x) *# True*

**# <= Less than equal to**

print(x <= y) *# False*

print(10 <= x) *# True*

**Assignment operators**

Assignment operators are used to assigning value to the variable

a = a+2

**Logical operators**

Logical operators are useful when checking a condition is true or not

|  |  |  |
| --- | --- | --- |
| and (Logical and) | True if both the operands are True | a and b |
| or (Logical or) | True if either of the operands is True | a or b |
| not (Logical not) | True if the operand is False | not a |

**Membership operators**

These operators are used to check for membership of objects in sequence, such as string, list, tuple

**In operator**

l = [11, 15, 21, 29, 50, 70]

n = 15

if number **in** l:

print("number is present")

else:

print("number is not present")

**Not in Operator**

a = (11, 15, 21, 29, 50, 70)

n= 35

if number **not** **in** a:

print("number is not present")

else:

print("number is present")

**Identity operators**

Compares the memory address.

**is operator**

x = 10

y = 11

z = 10

print(x is y) *# it compare memory address of x and y*

print(x is z) *# it compare memory address of x and z*

**is not operator**

x = 10

y = 11

z = 10

print(x is not y) *# it campare memory address of x and y*

print(x is not z) *# it campare memory address of x and z*

**Data Types**

There are mainly four types of basic/primitive data types available in Python

* Numeric: int, float, and complex
* Sequence: String, list, and tuple
* Set
* Dictionary (dict)

int : Stores Integer values **n=20**

float : Stores Decimal values **n=20.5**

complex : Stores complex values **n=2+3j**

str : Stores text or string data **name =’reena’**

bool :Stores Boolean values **flag = True**

list :Store a sequence of mutable data **l = [3, 'a', 2.5]**

tuple : Stores sequence immutable data **t =(2, 'b', 6.4)**

set : Stores unorder and unindexed values **s = {1, 3, 5}**

dict : Stores key: value pair **d = {1:'J', 2:'E'}**

range: Generate a sequence of number **numbers = range(10)**

**Python Casting: Type Conversion**

 We can convert one type of variable to another type. This conversion is called type casting or type conversion.

**Implicit casting**: (Indirect)

**Explicit casting**:(direct)

p = 3.14 # float number

print(type(p))

# Output class 'float'

# converting float integer

n = int(p)

print("Integer number:", n)

# Output 3

print(type(n))

# Output class 'int'

Python Data structures:

* Tuple
* List
* Set
* Dictionary

**Tuple**: A tuple is a list that cannot change. immutable

a = ('sita', 'gita', 'Rani')

It can access an individual element by its index.

Print(a[0])

Print(a[2])

tup = (10, 20, 30)

print(tup)

print(type(tup))

Methods in tuple :

count() ,index() ,len() ,max(), min(), sorted() ,sum()

**List** :

* It is a versatile and widely used data structure that allows you to store an ordered collection of items
* List\_a= [item1, item2, item3, ...]
* Ordered , Mutuable, Heterogeneous, Dynamic

Methods :

1. append(item) , extend(iterable) , insert(index, item)
2. remove(item) , pop(index), clear()
3. index(item, start, end)

**SET**:

a **set** is an unordered collection of unique elements. It is mutable, meaning you can add or remove items,

Creating a set

s = {1, 2, 3, 4}

Adds 5 to the set

s.add(5)

s.remove(3)

s.discard(10)

set\_a = {1, 2, 3}

set\_b = {3, 4, 5}

Union

print(set\_a | set\_b)

{1, 2, 3, 4, 5}

Intersection

print(set\_a & set\_b)

{3}

Difference

print(set\_a - set\_b)

{1, 2}

**dictionary :**

stores data in key-value pairs .contains key and values

d = {"name": "Alice", "age": 25, "city": "Bengaluru"}

|  |  |
| --- | --- |
| keys() | Returns a view object of all keys. |
| values() | Returns a view object of all values. |
| items() | Returns a view object of key-value pairs. |
| update() | Updates the dictionary with another dictionary or key-value pairs. |
| clear() | Removes all items from the dictionary. |
| copy() | Returns a shallow copy of the dictionary. |