**Features of Python:**

1. **Simple and easy to learn:**

Python is a simple programming language. When we read Python program, we can feel like reading English statements. The syntaxes are very simple and only 30+ keywords are available. When compared with other languages, we can write programs with very less number of lines. Hence more readability and simplicity. We can reduce development and cost of the project.

1. **Freeware and Open Source:**

We can use Python software without any licence and it is freeware. Its source code is open, so that we can we can customize based on our requirement

. Eg: Jython is customized version of Python to work with Java Applications.

1. **High Level Programming language:**

Python is high level programming language and hence it is programmer friendly language. Being a programmer we are not required to concentrate low level activities like memory management and security etc..

1. **Platform Independent**: Once we write a Python program, it can run on any platform without rewriting once again. Internally PVM is responsible to convert into machine understandable form.
2. **Portability**: Python programs are portable. ie we can migrate from one platform to another platform very easily. Python programs will provide same results on any platform.
3. **Dynamically Typed:** In Python we are not required to declare type for variables. Whenever we are assigning the value, based on value, type will be allocated automatically. Hence Python is considered as dynamically typed language. But Java, C etc are Statically Typed Languages because we have to provide type at the beginning only. This dynamic typing nature will provide more flexibility to the programmer.
4. **Both Procedure Oriented and Object Oriented:** Python language supports both Procedure oriented (like C, pascal etc) and object oriented (like C++,Java) features. Hence we can get benefits of both like security and reusability etc
5. **Interpreted:** We are not required to compile Python programs explicitly. Internally Python interpreter will take care that compilation. If compilation fails interpreter raised syntax errors. Once compilation success then PVM (Python Virtual Machine) is responsible to execute.
6. **Extensible:** We can use other language programs in Python. The main advantages of this approach are:
7. We can use already existing legacy non-Python code
8. 2. We can improve performance of the application
9. **Embedded:** We can use Python programs in any other language programs. i.e we can embed Python programs anywhere.
10. **Extensive Library:** Python has a rich inbuilt library. Being a programmer we can use this library directly and we are not responsible to implement the functionality.

**Limitations of Python:**

1. Performance wise not up to the mark because it is interpreted language.

2. Not using for mobile Applications

3. Python is slower than C/C++.

4. It is difficult to pack a big Python application into a single executable file.

**HISTORY OF PYTHON**

Python was developed by Guido van Rossum in the late 80’s and early 90’s at the National Research Institute for Mathematics and Computer Science in the Netherlands. It has been derived from many languages such as ABC, Modula-3, C, C++, Alogl-68, SmallTalk, Unix shell, and other scripting languages. Since early 90’s Python has been improved tremendously. Its version 1.0 was released in 1991. Version 2.0 released in 2000.

Different versions are available like Python 2.1 to Python 2.7. and Python 3.0 to Python 3.8.

Here, we are providing some specific application areas where python can be applied.

* **Web Applications**
* **Desktop GUI Applications**
* **Software Development**
* **Scientific and Numeric**
* **Business Applications**
* **Console Based Application**
* **Audio or Video based Applications**

**WRITING AND EXECUTING FIRST PYTHON PROGRAM**

Download Python from [www.Python.org](http://www.Python.org).

After the installation, Python console can be accessed in several ways. We will discuss only two of them here. First, using the command line and running the Python interpreter directly. Second, using a GUI software that comes installed with Python called Python’s Integrated Development and Learning Environment (IDLE).

When you run the IDLE, you get a prompt of three right arrows. Type in your instructions at the prompt and press Enter.

Python IDLE works on different platforms.

**Writing Python Programs**

1. Open an editor.
2. Write the instructions.
3. Save it as a file with the filename having the extension .py.
4. Run the interpreter with the command python program\_name.py or use IDLE to run the programs.

**COMMENTS**

Comments are the simple text, which are non-executable statements in a program. Comments make the program easily readable and understandable by the programmer as well as other users who are seeing the code. The interpreter simple ignores the comments.

**Example:**

# this is a comment

print(‘hello’)

#program end here

**RESERVED WORDS**

Reserved words are keywords which have pre-defined meaning. They cannot be used as identifiers. List of reserved words

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| True | False | None | and | as | assert | 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 |  |  |

**INDENTATION**

Whitespaces at the beginning of the line is called indentation. These whitespaces or the indentation are very important in python. In python program, the leading white space including spaces and tabs at the beginning of the logical line determines the indentation level of that logical line.

**IDENTIFIERS**

A name in Python program is called identifier. It can be class name or function name or module name or variable name.

A=10

Rules to define identifiers in Python:

1. They have alphabets, numbers and underscore(\_)
2. They starts with alphabet either lower or upper case.
3. They can start with underscore.
4. They cannot start with digit.
5. Reserved words cannot be used as identifiers.

**VARIABLE**

A variable is an identifier that denotes a name of a storage location to store a data value. The value in the location may vary during the execution of a program.

Rules to declare a variable:

1. They can be alphabet, digits and underscore.
2. They can be of any length.
3. They can start with letter either lower or upper case.
4. They can start with underscore.
5. They cannot start with digit.
6. They cannot be reserved word.
7. They are case sensitive.

**Declare and initializing variables:**

In Python, programmers need not explicitly declare variables to reserve memory space. The declaration is done automatically when a value is assigned to the variable using the equal sign(=).

**Syntax:** var=value

**Ex:** a=10

We can change the value of a variable as many times as we can. We may store value of different data type in a variable. This happens because Python variables do not have specific types.

**Ex:**

var='sbvr'

print(var)

var=123

print(var)

Single value can be assigned to multiple variables.

P=q=r=100

**DATA TYPES**

Data type represents the type of data present inside a variable. In Python we are not required to specify the type explicitly. Based on value provided, the type will be assigned automatically. Hence Python is Dynamically Typed Language.

Python has the following data types built-in by default, in these categories:

|  |  |
| --- | --- |
| Text Type: | str |
| Numeric Types: | int, float, complex |
| Sequence Types: | list, tuple, range |
| Mapping Type: | dict |
| Set Types: | set, frozenset |
| Boolean Type: | bool |
| Binary Types: | bytes, bytearray, memoryview |
| None Type: | NoneType |

**int data type:**

We use int data type to represent whole numbers (integer values).

Ex:

a=10

We can represent int values in the following ways

1. Decimal form: It allows digits : 0-9
2. Binary form: It allows digits: 0 &1. It prefix with 0b or 0B.
3. Octal form: It allows digits: 0-7. It prefix with 0
4. Hexa Decimal form: It allows digits:0-9 and A-F. It prefix with 0x

**float data type:**

We use float data type to represent floating point values (decimal values)

Ex: f=7.3

We can also represent floating values by using exponential form

Ex: f=2.3e2

**Complex data type:**

A complex number is of the form

a + bj

a and b contain integers or floating point values

Eg:

3+5j

10+5.5j

0.5+0.1j

In the real part if we use int value then we can specify that either by decimal, octal, binary or hexa decimal form. But imaginary part should be specified only by using decimal form.

**bool data type:**

We can use this data type to represent boolean values. The only allowed values for this data type are: True and False.

Internally Python represents True as 1 and False as 0

Eg:

a=10

b=20

c=a<b

**print(c ) 🡺True**

**str type:**

str represents String data type.

A String is a sequence of characters enclosed within single quotes or double quotes. s1='durga'

s1="durga"

By using single quotes or double quotes we cannot represent multi line string literals. s1="durga

soft"

For this requirement we should go for triple single quotes(''') or triple double quotes(""")

**bytes data type:**

Bytes data type represents a group of byte numbers just like an array. The only allowed values for byte data type are 0 to 256. By mistake if we are trying to provide any other values then we will get value error.

Once we create bytes data type value, we cannot change its values, otherwise we will get TypeError.

**Ex:**

x=[10,20,30,40]

b=bytes(x)

print(b[2])

print(b[-1])

**bytearray Data type:**

bytearray is exactly same as bytes data type except that its elements can be modified.

Ex:

x=[10,20,30,40]

b=bytearray(x)

b[0]=200

print(b[0])

print(b[-1])

**list data type:**

If we want to represent a group of values as a single entity where insertion order required to preserve and duplicates are allowed then we should go for list data type.

1. insertion order is preserved

2. heterogeneous objects are allowed

3. duplicates are allowed

4. Growable in nature

5. values should be enclosed within square brackets.

**Ex:**

list=[2,3,'ffgf',23.4]

print(list[0])

print(list[2])

print(list[1:3])

list[3]='jack'

print(list)

list is growable in nature. i.e based on our requirement we can increase or decrease the size.

**Ex:**

list=[23,34,24,43]

list.append("reddy")

print(list)

list.remove(24)

print(list)

list2=list\*2

print(list2)

**tuple data type:**

Tuple data type is exactly same as list data type except that it is immutable. i.e we cannot change values. Tuple elements can be represented within parenthesis.

Ex:

t=(2,3,4,5)

type(t)

**range Data Type:**

range Data Type represents a sequence of numbers. The elements present in range Data type are not modifiable. i.e range Data type is immutable.

**Form-1**: range(10)

generate numbers from 0 to 9

**Ex**: r=range(10)

for i in r : print(i) 0 to 9

**Form-2**: range(10,20) generate numbers from 10 to 19

**Ex**: r = range(10,20)

for i in r : print(i) 10 to 19

**Form-3:** range(10,20,2)

2 means increment value

**Ex:** r = range(10,20,2)

for i in r : print(i) **10,12,14,16,18**

We can access elements present in the range Data Type by using index.

r=range(10,20)

r[0]==>10

r[15]==>IndexError: range object index out of range

We cannot modify the values of range data type

**set Data Type:**

If we want to represent a group of values without duplicates where order is not important then we should go for set Data Type.

1. insertion order is not preserved

2. duplicates are not allowed

3. heterogeneous objects are allowed

4. index concept is not applicable

5. It is mutable collection

6. Growable in nature

**Ex:**

s={2,3,5,'ashwika','nitiksha'}

print(s)

s.add('reddy')

print(s)

s.remove(5)

print(s)

**frozenset Data Type:**

It is exactly same as set except that it is immutable. Hence we cannot use add or remove functions.

**Ex:**

fs=frozenset({3,5,6,4,'kik'})

for i in fs:

print(i)

**dict Data Type:** If we want to represent a group of values as key-value pairs then we should go for dict data type.

**Eg:** d={101:'durga',102:'ravi',103:'shiva'}

Duplicate keys are not allowed but values can be duplicated. If we are trying to insert an entry with duplicate key then old value will be replaced with new value.

**Ex:**

d={1:"sreenu",2:"ashwika",3:"nitiksha"}

print(d)

d1={}

d1['a']='apple'

d1['b']='banana'

print(d1)

**INPUT OPERATION/OUTPUT**

To take input from the users, Python makes use of the input() function. The input function takes user’s input as string. So whether you input a number or a string, it is treated as string only.

**Syntax:** variable name=input() **or** variable name=input(‘string’)

**Example:** a=input(‘Enter your name’)

Print function is used to display the contents on the screen.

**Syntax:** print(argument)

**Example:** print(‘sbvr degree college’)

print(1000)

**OPERATORS**

Operator is a symbol that performs certain operations.

Python provides the following set of operators

1. Arithmetic Operators
2. Comparison (Relational) Operators
3. Assignment Operator
4. Logical operator
5. Bitwise Operator
6. Membership Operator
7. Identity operator

**Arithmetic Operators:** These operators are used to perform arithmetic operations

a=100, b=200

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Example** | **Output** |
| + | Addition | >>>print(a+b) | 300 |
| - | Subtraction | >>>print(a-b) | -100 |
| \* | Multiplication | >>>print(a\*b) | 20000 |
| / | Division | >>>print(b/a) | 2.0 |
| % | Modulus | >>>print(b%a) | 0 |
| // | Floor division | >>>print(12//5)  >>>print(12.0//5.0)  >>>print(-20.0//3) | 2  2.0  -7.0 |
| \*\* | Exponentiation | >>>print(a\*\*b) | 100200 |

**Comparison operators:** Comparison operators also known as **relational operators** are used to compare the values on its either sides and determine the relation between them. Consider a=100 and b=200.

|  |  |  |  |
| --- | --- | --- | --- |
| Operator | Name | Example | Output |
| == | Equal | >>>print(a==b) | False |
| != | Not equal | >>>print(a!=b) | True |
| > | Greater than | >>>print(a>b) | False |
| < | Less than | >>>print(a<b) | True |
| >= | Greater than or equal | >>>print(a>=b) | False |
| <= | Less than or equal | >>>print(a<=b) | true |

**Assignment operator:** Assignment operators are used to assign values to variables.

|  |  |  |
| --- | --- | --- |
| Operator | Examples | Meaning |
| += | a+=1 | a=a+1 |
| -= | a-=b | a=a-b |
| \*= | a\*=b | a=a\*b |
| /= | a/=b | a=a+b |
| %= | a%=b | a=a%b |
| //= | a//=b | a=a//b |
| \*\*= | a\*\*=b | a=a\*\*b |

**Logical operators:** Logical operators are used to combine conditional statements:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Meaning** | **example** |
| and | Logical and | x < 5 and x>4 |
| or | Logical or | x < 5 or x < 4 |
| not | Logical not | not(x < 5 and x < 10) |

**Bitwise operators:** Bitwise operators perform operations at the bit level. . Bitwise operators may not be applied to float or double data type.

|  |  |
| --- | --- |
| Operator | Meaning |
| & | Bitwise AND |
| | | Bitwise OR |
| ^ | Bitwise XOR (Exclusive OR) |
| ~ | 1’s complement |
| >> | Right shift |
| << | Left shift |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P | Q | P&Q |  | P | Q | P|Q |  | P | Q | P^Q |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 |  | 1 | 0 | 1 |  | 1 | 0 | 1 |
| 0 | 1 | 0 |  | 0 | 1 | 1 |  | 0 | 1 | 1 |
| 0 | 0 | 0 |  | 0 | 0 | 0 |  | 0 | 0 | 0 |

Left shift Right shift

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| P | ~P |  | A | 0100101 |  | A | 0100101 |
| 1 | 0 |  | A<<3 | 0101000 | A>>3 | 0000100 |
| 0 | 1 |  | A<<5 | 0100000 | A>>5 | 0000001 |

**Membership operator:** We can use Membership operators to check whether the given object present in the given collection. (It may be String,List,Set,Tuple or Dict)

**in ->** Returns True if the given object present in the specified Collection

**not in->** Retruns True if the given object not present in the specified Collection

**Example:**

x="i sreenuvasa reddy"

print('h' not in x)

print('s' in x)

**Identity operator:**

We can use identity operators for address comparison. Identity operators are

1. is
2. is not

r is r1 returns True if both r and r1 are pointing to the same object

r is not r1 returns True if both r and r1 are not pointing to the same object

**Example:**

|  |  |
| --- | --- |
| list2=['one','two','three']  list1=['one','two','three']  print(id(list1))  print(id(list2))  print(list2 is list1)  print(list2 is not list1)  print(list2==list1) | a='sreenu'  b='sreenu'  print(id(a))  print(id(b))  print(a is b) |

**CONDITIONAL STATEMENTS**

**if:**

if condition: statement

or

if condition:

statement-1

statement-2

statement-3

If condition is true then statements will be executed.

**Ex:** name=input(" Enter the name ")

if name=='srinu':

print("hello srinu Good morning")

print(" how are you !")

**if-else:**

n1=int(input("Enter first number"))

if n1%2==0:

print("even number")

else:

print("odd number")

**syntax:**

if condition:

action1

else:

action2

if condition is true then action1 will be executed otherwise action-2 will be executed.

**Ex:**

name=input(" Enter the name ")

year=int(input("Enter first number"))

if year%4==0 and year%100!=0 or year%400==0:

print(year," is leap year")

else:

print(year," is not leap year")

if name=='srinu':

print("hello srinu Good morning")

print(" how are you !")

else:

print(" hello guest")

print(" how are you ")

**if-elif-else:**

**syntax:**

if condition-1

action-1

elif condition-2

action-2

elif condition-3

action-3

else

default action

Based on the condition the corresponding action will be executed.

**Ex:**

n1=int(input("Enter first number"))

n2=int(input("Enter second number "))

n3=int(input("Enter third number "))

if n1>n2 and n1>n3:

print("%d is greter",n1)

elif n2>n3:

print("%d is greater",n2)

else:

print("%d is greater",n3)

mark=int(input(" Enter the mark value "))

if mark>=75:

print("distinction")

elif mark>=60 and mark<75:

print("first class")

elif mark>=50 and mark<60:

print("second class")

elif mark>=35 and mark<50:

print("third class")

else:

print("fail")

**ITERATIVE STATEMENTS**

If we want to execute a group of statements multiple times then we should go for Iterative statements.

1. For loop
2. While loop

**For loop:**

If we want to execute some action for every element present in some sequence (it may be string or collection) then we should go for **for loop**.

**Syntax:**

for i in sequence:

body

Where sequence can be any string or any collection.

Body will be executed for every element present in the sequence.

Ex:

s=input(" Enter the string ")

i=0;

for x in s:

print(" the character at the ",i," index is: ",x)

i=i+1

s="sreenu reddy"

for x in s:

print(x,end=" ")

**#EVEN OR ODD**

for x in range(1,20):

if(x%2==0):

print(x,"is even")

else:

print(x,"is odd")

for x in range(10):

print(x,end=" ")

**#Multiplication table:**

n=int(input(" enter the number "))

for x in range(1,11):

print(n,"\*",x,"=",n\*x)

#**sum of numbers**

sum=0

for i in range(0,10):

sum=sum+i

print(" the sum is ",sum)

#**PRIME NUMBER**

n=int(input(" enter the number "))

count1=0

for x in range(1,n+1):

if(n%x==0):

count1=count1 + 1

if(count1==2):

print("prime")

else:

print("not prime")

#**REVERSE NUMBER**

a=eval(input(" Enter the number: "))

rev=0

while(a>0):

i=a%10;

rev=(rev\*10)+i

a=a//10

print(" The reverse number is ",rev)

**While loop:**

If we want to execute a group of statements iteratively until some condition false, then we should go for while loop.

**Syntax:**

while condition :

body

**Ex:**

x=1

while x<=10:

print(x)

x=x+1

**2:**

n=int(input(" Enter number : "))

sum=0

i=1

while i<=n

sum=sum+i

i=i+1

print(" The sum of first",n."number is :",sum)

**Nested Loops:**

Sometimes we can place a loop inside another loop, which are also known as nested loops.

n=int(input(" Enter number of rows "))

for i in range(1,n+1):

for j in range(1,i+1):

print("\*",end=" ")

print()

for i in range(4):

for j n range(4):

print("i=",i,"j=",j)

**Break:**

We can use break statement inside loops to break loop execution based on some condition.

Ex:

cart=[10,20,600,60,70]

for item in cart:

if item>500:

print(" to place this order insurance must be required ")

break

print(item)

for i in range(10):

if i==7:

print(" processing is enough, plz break ")

break

print(i)

**Continue statement:**

We can use continue statement to skip current iteration and continue next iteration.

cart=[10,20,500,700,50,60]

for item in cart:

if item>=500:

print("We cannot process the item", item)

continue

print(item)

for i in range(10)

if i%2==0

continue

print(i)

**Pass statement:**

The pass statement is used when a statement is required syntactically but no command or code has to be executed. It specifies a null operation or simply No Operation (NOP) statement. Nothing happens when the pass statement is executed.

**Syntax:** pass

**Ex:**

for i in range(1,10):

if i==3:

pass

print(" Reached pass statement ")

print(" The current number is ",i)

|  |  |
| --- | --- |
| #**ARMSTRONG NUMBER**  a=int(input(" Enter the number: "))  rev=0  temp=a  while(a>0):  i=a%10;  rev=rev+(i\*i\*i);  a=a//10  if(temp==rev):  print(" armstrong number ")  else:  print(" not armstrong ") | **#PALINDROME NUMBER**  a=int(input(" Enter the number: "))  rev=0  temp=a  while(a>0):  i=a%10;  rev=(rev\*10)+i  a=a//10  if(temp==rev):  print(" Palindrome number ")  else:  print(" Not Palindrome number ") |
| **#FIBONACCI SERIES**  n=int(input(" How many terms : "))  a=0  b=1  print(a)  print(b)  i=2  while(i<n):  temp=a+b  a=b  b=temp  print(temp)  i=i+1 | **#FACTORIAL PROGRAM**  n=int(input(" Enter the number : "))  fact=1  while(n>0):  fact=fact\*n  n=n-1  print(" factorial = ",fact) |