**Introduction of python:-**

Python is a high-level, interpreted programming language.

* It is known for its simplicity, readability, and ease of use.
* Used in various fields such as web development, data science, machine learning, and automation.
* Python is platform-independent and has a large community with extensive libraries.

**Data types in python**

**Numeric Types:-**

Int(integer)

Float(decimal numbers)

Complex(complex numbers

EXAMPLES:-

x=10:-int

x=3.2(float)

x=2+2j(complex)

**Text types :-`str`**string used for textual data

EXAMPLE:-

Name=”jeevan”

**Boolean type:-** it means `bool` just true or false

Is\_active=true

**Sequence types:-**

List

Tuple

Range

List []:- [1,2,3]

Tuple ():-(3,4,5)

Range:-range(10)

**Mapping type:-`dict` dictionary**

Map\_dict{“key”: “value” }

**Set types :-** `set`,`frozenset`.

My\_set={1,2,3}

**None type:-** represents no value

Value = none

**Operators:-**

**Types of operators:-**

**1.** Arithmetic Operators:- Perform mathematical operations.

EXAMPLES:-

**`+`, `-`, `\*`, `/`, `//`, `%`, `\*\*`.**

a=10

b=20

Print(a+b):- 30 #

Print(a-b):- 10 #

Print(a\*b):- 200#

Print(a/b):-0.5 #

Print (a\*\*b) :- print a to the power of b

1. Comparison operators :-compare values and return a boolean

**-`<`,`>`,`<=`,`>=`,`==`,`!=`**

EXAMPLES:- a=10 b=12

print a>b :- false

Print a<b:- true

Print a<=b=true

Print a>=b=true

Print a==b= false

1. logical opertors:- combine multiple conditions

Examples:- **`and`,`or`,`not`**

x=true

b=false

Print (x and y):- false Print(x or y):- true

1. assignment operators:-assign and modify values

**`=`, `+=`, `-=`, `\*=`, `/=`, `%=`, `\*\*=`.**

Example :-

total=10

total+=5

Print (total) #15

1. membership operators:-check membership in seqeence.

Examples:-`in`,`not`

a=[1,2]

b=a

Print (a is b) # true

**conditional handling and control statements**

**Conditional handling**

1. if statement:-excutes a block if condition is true

Example :-

x=10

x>5

True

1. if else stament :- excutes one block if the condition is true,otherwise another block

Example:-

if x % 2 == 0:

print("x is even")

else:

print("x is odd")

1. \*if-elif-else Statement\*\*: Multiple conditions.

if x < 5:

print("x is less than 5")

elif x == 5:

print("x is 5")

else:

print("x is greater than 5")

1. nested if :- you can nest one if statement inside another to check multiple conditions

x = 15

if x > 10:

print("x is greater than 10")

if x % 2 == 0:

print("x is even")

else:

print("x is odd")

**Control statements:-** control statements manage the flow of execution.

**break statement**: break statement is used to terminate the loop before the ending the sequence . it brings control out of the loop.

**Ex:**

for i in range(1,10):

if i==5:

break

print( i , end=’ ‘)

output: 1, 2, 3, 4

**continue statement:** if returns the control to the beginning of the loop.

For I in range(1,10):

If i%2 == 0 or i%3==0:

continue

print( i , end=’ ‘)

output: 5, 3, 7

**Loops:**

Looping statement: A loop statements is used to execute a statement or group of statements multiple times

**For:** for loop is used for sequential traversal example traversing a list or string or range etc.

Syntax:

For iterator-var in sequence:

statements

Program :

n=int(input())

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

print(I, end=””)

input: 10

output: 1,2,3,4,5,6,7,8,9,10

**break statement:** break statement is used to terminate the loop before the ending of the sequence it brings control out of the loop

syntax:

for I in range(1,10):

if I == 5:

break

print(I, end=””)

**continue statement:** it returns the control to the beginning of the loop

Program:

for I in range(1,10):

if i%2 == 0 or i%3 == 0:

continue

print(I, end=””)

output: 1, 5, 7

**pass statement:** pass statement is used to represent empty control statements , functions and classes.

Syntax:

For I in range (1,10):

If i%3==0 or i% 2 == 0:

Pass

else:

print(I, end=” ”)

output: 1, 5, 7

**for else:** for else is used to avoid flag variables else part will execute when the loop is terminated due to end of the sequences. else part will not execute if loop is terminated due to break statement.

Syntax:

for iteration\_var in sequence:

if condition:

break

else:

Statements

**Nested for:** for written a for every iteration of the outer loop, inner loop will execute completely

Syntax:

for iteration\_var in sequence:

for iter\_var in sequence:

statements

**while:** if the number of iterations doesn’t know in advance or if the sequence needs to update inside of the loop then while is useful

syntax:

while condition:

statements

Program:

n=int(input())

i=1

while i<=n:

print(I, end=” “)

i+=1

**Nested loop:** A looping statement inside of another looping statement is called nested loop that may be for inside for , while inside while, while inside for , for inside while

Program:

n=int(input())

while n>9:

s=0

while n!=0

s=s+n%10

n=n//10

n=s

print(n)

input: 276

output: 6

**Keywords**

* Keywords are reserved words ; functionality is related to these words already defined in the translator
* With the same functionality programmer need to use these words
* In python a keyword is a reserved word that has a specific meaning and is used to define the syntax and structure of the language
* There are 36 keyword in python

Syntax:

Import keyword

Print(keyword.kwlist)

Keywords:

[

“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”

]

lists and tuples are two commonly used data structures that allow you to store collections of items.

**List:**

A list is a collection of ordered elements. Lists can be of different data types such as integers, floats ,strings etc. various operations such as adding or removing elements of searching and sorting for elements can be performed in a list in python

# create list  
  
my\_list=[1,5.5,"jeevan","nagendra",True,False]  
  
print(my\_list) # opt: [1,55,14,'jeevan','nagendra',True,False]  
  
# 1. Append – add the new items into the list  
  
my\_list.append("gopinath")  
  
print(my\_list) # opt: [1,5.5,'jeevan','nagendra',True,False,'gopinadh']  
  
# 2. Extend – extend the list to add new items  
  
my\_list.extend([5,3])  
  
print(my\_list)  
  
# opt: [1,5.5 , 'jeevan', 'nagendra' , True, False 'gopinadh' 5, 3]  
  
# 3. insert(index, item) – insert the data at particular (index) position.  
  
my\_list.insert(2,"jeevan")  
  
print(my\_list)  
  
# opt: [1,5.5, 'jeevan','jeevan','nagendra',True,'False','gopinadh',5,3]  
  
# 4. remove(item) – remove the item in the list  
  
my\_list.remove("nagendra")  
  
print(my\_list)  
  
# opt: [1,5.5, 'jeevan', 'jeevan', True, False, 'gopinadh',5,3]  
  
# 5.pop(item) – remove last item in the list  
  
my\_list.pop()  
  
print(my\_list)  
  
# opt: [1,5.5, 'jeevan' 'jeevan', True , False, 'gopinadh', 5]  
  
# 6. sort () – sort the list items  
  
my\_list1=[1,43,5,2,535,21,444,20]  
  
my\_list1.sort()  
  
print(my\_list1) # opt: [1, 2, 5, 20, 21, 43, 444, 535]

Applications of list:

* Lists can be used to store and access sequential data.
* We can modify elements in a list because lists are mutable.

**Tuple:**

a tuple is a sequence of elements separated by commas and enclosed in parentheses. tuples are similar to lists, but they cannot be modified once created. This means that you cannot add. Remove or modify elements of a tuple.

# create the list of numbers

list\_tuple = (1,2,3,4,5)

#print first three elements of tuple

print(list\_tuple[:3])

# opt: (1,2,3)

# deleting a tuple

del\_tuple = (1,2,3,4,5)

# delete tuple using del

tuple\_del=(1,2,3,4,5)

# delete tuple using del

del tuple\_del

print(tuple\_del)

Applications of Tuple:

* Unlike lists , tuples are immutable and can be used to store dictionary keys.
* Tuples can return multiple values from a function such as you can use a tuple to return minimum and maximum values from a list.

**Set:**

A set is a built-in data structure in Python that represents a collection of unique elements. We can perform different operations on sets, such as union, intersection, difference, and symmetric difference.

# create the set  
my\_set = {1,44.4,15,7,51}  
  
print(my\_set) # opt: {1, 51, 7, 44.4, 15}  
  
# add – add items into the set  
  
my\_set.add(5)  
  
print(my\_set) # opt: {1, 51, 5, 7, 44.4, 15}  
  
# remove – remove the item in a set  
  
my\_set.remove(44.4)  
  
print(my\_set) # opt: {1, 51, 5, 7, 15}  
  
# union set – Returns a new set with all unique items from both sets.   
my\_set1={43,22,67,643}  
  
union\_set=my\_set.union(my\_set1) # opt: {1, 67, 643, 5, 7, 43, 15, 51, 22}  
  
print(union\_set)  
  
# intersection\_set - Returns a set of items common to both sets.   
  
intersection\_set=my\_set | my\_set1  
  
print(intersection\_set) # opt: {1,67, 643, 5, 7, 43, 15, 51, 22}

**Applications**

* Sets can be used to perform various operations such as union, intersection and difference.
* Sets can be used to remove duplicates from a list.

**Dictionary :**

A dictionary in Python is a collection of key-value pairs, where each key is unique and associated with a value. We can change the values of a dictionary. They are useful for storing and accessing data.

# create the dictionary  
  
my\_dict={"name":"jeevan","last\_name":"nagendra","address":"vizag"}  
  
print(my\_dict)  
  
# opt: {'name':'jeevan','last\_name':'nagendra','address':'vizag'}  
  
# get a key – get only specified key name in the dictionary  
  
get\_key=my\_dict.get("name")  
  
print(get\_key) # opt: jeevan  
  
# get keys – get all the keys in dictionary   
  
only\_keys=my\_dict.keys()  
  
print(only\_keys) # opt: dict\_keys( ['name', 'last\_name','address'] )  
  
# get values – get all the values in dictionary  
  
only\_values=my\_dict.values()  
  
print(only\_values) # opt: dict\_values(['jeevan', 'nagendra', 'vizag'])  
  
# items – git all items in a dictionary like a list  
  
only\_values=my\_dict.items()  
  
print(only\_values)  
  
# opt: ([('name', 'jeevan'),('last\_name','nagendra'),('address','vizag')])  
  
# pop(key) – remove the item in a dictionary with key name  
  
only\_pop=my\_dict.pop("name")  
  
print(only\_pop) # opt: jeevan

**Applications of Dictionary**

* Dictionaries can be used to store key-value pairs.
* Dictionaries are also useful for counting the frequency of elements in a list.

**Difference Between List, Tuple, Set and Dictionary**

Now we will draw a comparison table between a list, tuple, set and dictionary in Python and discuss about the characteristics of each of them

|  |  |  |  |
| --- | --- | --- | --- |
| **List** | **Tuples** | **Set** | **Dictionary** |
| A list is a collection of ordered elements. | A tuple is a sequence of elements separated by commas and enclosed in parentheses. | A set is a built-in data structure in Python that represents a collection of unique elements. | A Dictionary is a collection of key-value pairs, where each key is unique and associated with a value. |
| Lists maintain the order of the elements they contain. | Tuples maintain the order of the elements they contain | Sets do not maintain the order of the elements they contain | Dictionaries do not  maintain the order of the elements they contain. |
| Lists can be accessed by index | Tuples can be accessed by index | Sets cannot be accessed by index | Dictionaries cannot be accessed by index |
| Lists can be modified by adding or removing elements | Tuples cannot be modified by adding or removing elements | Sets can be modified by adding or removing elements | Dictionaries cannot be modified by adding or removing elements |
| Lists can contain duplicate elements | Tuples can contain duplicate elements | Sets cannot contain duplicate elements | Dictionaries cannot contain duplicate elements |
| Lists can be accessed by index | Tuples can be accessed by index | Sets cannot be accessed by index | Dictionaries cannot be accessed by index |
|  |  |  |  |

**FUCTIONS:-**

A function is a block of code which only runs when it is called. You can pass data , known as parameters , into a function. A function can return data as a result.

Creating a function:

In python a function is defined using the def keyword

Syntax:

def my\_function():

Print(“Hello jeevan nagendra”)

my\_function()

Calling a function:

To call a function, use the function name followed by parenthesis:

Syntax:

def my\_function():

print(“Hello from a function”)

my\_function()

Arguments:

Information can be passed into functions are arguments. Arguments are specified after the function name, inside the parentheses, you can add as many arguments as you want, just separate from with a comma.

Syntax:

def my\_function(first\_name):

print(first\_name+ “ Noted ”)

my\_function(“jeevan”)

my\_function(“gopi”)

my\_function(“mani”)

**Modules:**

Module is nothing but a file that contains python code in that in form of functions classes and variables which we can use in other programs.

Modules are used to reuse code.

Any file with .py extension is called as a module.

So here assume we are developing a big project ABCD so we divide this into module A,B,C,D so we will be using this modules in another project if it is needed so code re-usability is a very important thing in programming.

In modules we have 2 types of modules:

* User defined modules
* Built-in modules

**Built-in modules :**

The modules which are already predefined in python

So lets have a look on some built-in modules:

1.Math

2. Random

3. Datetime

These are few predefined modules in python let’s see with one simple examples

Import math

Print(math.pi) #output: 3.141592653589793

\*\*\*\*

import random

print(random.randint(1, 10)) # Output: Random integer between 1 and 10

\*\*\*\*

import datetime

now = datetime.datetime.now()

print(now) # Output: Current date and time

**Userdefined modules :**

The modules which are created by programmers is called userdefined modules

Here I have defined some functions in the module file

def add(a,b):  
 return a+b  
def sub(a,b):  
 return a-b  
def mul(a,b):  
 return a\*b  
def div(a,b):  
 return a/b

So I need to do some operations on add.sub,mul and div to do these operations no need to do the coding from the scratch so just I will import those modules.

from module import \*

now just I will do the operations which ever I needed.

a = 9  
b = 7  
  
c = add(a,b)  
print(c) # 16  
  
c = sub(a,b)  
print(c) # 2

c= mul(a,b)  
print(c) # 63

c = div(a,b)  
print(c) # 1.2857142857142858

**Package**

Package is the collection of modules and subpackages

Each package contains a special file called as \_\_init\_\_.py

When a directory contains a \_\_init\_\_.py file python treats the directory as a package allowing u to import modules and sub packages.

In python we have larger developer community so code which is written by the developer he will add that package to the Pypl

So knowing about the pypl(python package index) we call this pypl as a repository more than 2 lakh packages are present in this repository so we will install those packages from this repository with the help of pip we can say this as package manager we can install, uninstall, search , upgrade, list operations in with this pip.

Strings:

Strings in python are surrounded by either single quotation marks, or double quotation marks

‘hello’ is the same as “hello”.

Syntax:

print(“Hello”)

print(‘hello’)

Quotes inside Quotes:

You can use quotes inside a sting, as long as they don’t match the quotes surrounding the string:

print("It's alright")  
print("hi iam 'jeevan'")  
print('hi my name is "jeevan"')

Assign String to a variable

Assign a string to a variable is done with the variable name followed by an equal sign and the string:

a=”Hello”

print(a)

Multiline Strings:

You can use three double quotes:

a = """Lorem ipsum dolor sit amet,  
consectetur adipiscing elit,  
sed do eiusmod tempor incididunt  
ut labore et dolore magna aliqua."""  
print(a)

or three single quotes:

a = '''Lorem ipsum dolor sit amet,  
consectetur adipiscing elit,  
sed do eiusmod tempor incididunt  
ut labore et dolore magna aliqua.'''  
print(a)

Strings are Arrays:

Strings in python are arrays of bytes representing Unicode characters. Python does not have a character data type, a single character is simply a string with a length of 1

A=”Hello, world”

print(a[1])

looping through a string:

loop through the letters in the word “banana”:

for x in “banana”:

print(x)

String length:

to get the length of a string , use the len() function

a=”Hello, world”

print(len(a))

check string:

to check if a certain phrase or character is present in a string, we can use the keyword in.

str=”The best things in life are free!”

print(“free” in str)

Check if NOT:

to check if certain phrase or character is NOT present in a string we can use the keyword no in

str=”the best things in life are free”

if “expensive” not in str:

print(“No, ‘expensive’ is NOT present. “)

File Handling Operations:

1. Open to a file
2. Reading from a file
3. Writing to a file
4. Appending to a file
5. Closing to file
6. Checking if file exists

open file(): open(“filename.txt”,mode)

ready file(): open(“example.txt”,’r’)

content=file.read()

print(content)

file.close()

writing a file:

ex: hello world this is python file operations are easy writing a file

syntax:

file=open(“example.txt”,’w’)

file.write(“hello world \n”)

file.close()

appending a file:

ex:

file=open(“example.txt”,’a’)

file.write(“this is an append mode”)

file.close()

closing a file : file.close()

checking if file exists:

import os

if os.path.exists(“example.txt”)

print(“file exists”)

else:

print(“file does’ not exists”)

Advantages of file handling in python

* Versatility
* Flexibility
* User-friendly
* Cross-platform

Dis advantages of file handling in python:

* Error prone
* Security-risks
* Complexity
* performance