Introduction:

- python was developed by Guido van Rossum in the late 1980's and early 1990's in Netherlands.
- The inspiration came from the BBC's Tv Show Monty Pythons Flying circus which is very famous tv show at that time.
- And also, he wanted a short, unique and slightly mysterious so it is named as python.
- Python is high level interpreted, interactive and object-oriented scripting language.

Applications:

- Desktop applications
- Web applications
- Database applications
- Machine learning
- IOT
- Artificial intelligence applications

Some features:

- In python we are not required to specify the type explicitly. Based on value provided, the type will be assigned automatically.
- In python 2 we have long integer whereas in python 3 there is no long integer instead of long integer we use int only.
- Being a programmer, we can specify literal values in decimal, binary, octal and hexadecimal form. But pvm will always provide values only in decimal form.

```
b=0o10
print(b)#
c=0x10
print(c)#16
d=0b10
print(d)# 2
```

In python identifier can have a maximum length of 79 characters in python.

There is no use of using curly braces or semicolons in python programming language.it is an English like language but python define a block of code. Indentation is nothing but adding the whitespace before the statements when it is needed.

Comments in python:

There are two types of comment line in python

1. Single Line Comment:

```
#print ("hello world")
```

2. Multiline comments:

```
#This is comment
#written in
#More than just one line
```

Variables:

- A variable is a name of memory location. It is used for store the data.
- variables are changeable we can change the value of the variable during the execution of the program.

Rules:

- Variables must start with letters or underscore.
- No special symbols are used for declaring variables except underscore.
- Variables names cannot contain special names or spaces.
- Variable name cannot be keyword because

Multiple values:

Python allows you to assign values to multiple variables in one line:

```
x,y,z="orange","Banana","cherry"
print(x)
print(y)
print(z) which is not possible in C
```

Keywords (Python)

- 35 keywords available in python.

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

and	def	exec	if	not	return	True
Assert	del	finally	import	or	try	False
Break	elif	for	in	pass	while	None
Class	else	from	is	print	with	
Continue	except	global	lambda	raise	yield	

There are 2 types pf keywords are there in

Soft keyword

Hard keyword

Soft keyword:

Here it only uses _, match, case as keywords.

```
keyword.softkwlist
['_', 'case', 'match']
```

```
match 10:
  case 15:
    print(15)
 case 10:
   print(10)
case _:
 print(default)
```

Here it behaves as name of the variables.

```
match =10
case=3
```

Hard keyword:

Without depend on syntax it will always behave as keyword

e.g: def f1:

    print("dff")

def r=10//error

One value to multiple variables:(which is not possible in c language, we can able to allocate same values to different variables)

```
x=y=z="orange"
print(x)
print(y)
print(z)
```

unpack a collection

If you have a collection of a values in a list, tuple, etc. Python allows you to extract the values into variables. This is called as unpacking.

```
Fruits=["apple", "banana", "cherry"]
x,y,z=fruits
print(x)
print(y)
print(z)
```

fruits=['apple','banana','cherry']

```
x,y,z=fruits
print(x)
```

apple

```
print(y)
```

banana

```
print(z)
```

cherry

fruits={'red','apple','june'}

x,y,z=fruits

print(x)

june

print(y)

apple

print(z)

red

fruits={'red':2024,'rrr':234} while in dictionary it only gives key values.

x,y=fruits

print(x)

red

print(y)

rrr

In the print() function, when you try to combine a string and a number with the + operator, python will give an error:

X=5

Y="john"

Print(x+y)

o/p: Error

The best way to output multiple variables in the print() function is to separate them with commas, which even support different data types:

X=5

Y="john"

Print(x,y)

o/p: 5, john

x="python"

y="is"

z="awesome"

print(x +y +z)

pythonisawesome//(if we want to give space in between python is awesome then we have to declare it only in while declaring only)

x=”python “

y=”is “

z=”awesome”

print(x+y+z)//python is awesome

Global variables:

variables that are created outside of a function is called as global variables.

global variables can be used by everyone, both inside and outside.

When we declare any variable inside function by giving global keyword we can use that variable anywhere.

Also, use the global keyword if you want to change a global variable inside a function.

```
X=” awesome”
def myfunc():
    global x
    x=”fantastic”
myfunc()
print(“python is”)//python is fantastic
```

If you create a variable with the same name inside a function, this variable will be local, and can only be used inside the function.

```
X=” awesome”
def myfunc():
   X=”fantastic”
myfunc()
```

Print("python is"+x)//python is fantastic

PEP 8 stands for Python Enhancement Proposal, it can be defined as a document that helps us to provide the guidelines on how to write the Python code. It is basically a set of rules that specify how to format Python code for maximum readability. It was written by Guido van Rossum, Barry Warsaw and Nick Coghlan in 2001. PEP 8 is a style guide for Python code. Coding conventions are about indentation, formatting, tabs, maximum line length, imports organization, line spacing etc.

Built-in Data Types

In programming, data type is an important concept.

Variables can store data of different types, and different types can do different things.

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 (The tuple takes place lesser space because tuples are immutable so, it doesn't require extra space to store new objects, whereas lists are allocated two blocks of memory one for fixed one and variable size block list than list, tuple is faster than list, tuples make the code safe from any accidental modification)
Mapping Type:	dict
Set Types:	set, frozenset

Boolean Type: bool

Binary Types: bytes, bytearray, memoryview

None Type: None Type

Range: range data type is not modifiable. sequence of integers.

Range(start, stop, step)//here mentioning stop value is necessary.

Stop value is always excluded

```
x = range(6)
print(x)//range(0,6)
print(type(x))
```

String: If we want to use + operator to string then the both must be strings.

```
x =str("Hello World")
print(x)
print(type(x))
```

int:

```
x = int(20)
print(x)
print(type(x))
```

Float:

```
x = float(20.5)
print(x)
print(type(x))
```

bytes: bytes data type represents a group of bytes numbers, just like an array.

Directly stored on the disk. Utf-8 or utf-32 encoding here it contains 11 lakh 12 thousand 64 characters it also contains Telugu not only Telugu but also so many other

Languages.

```
X=[10,20,25,30]
b=bytes(x)
Type(b)#bytes
Print(b[0])
Print(b[-1])
For i in b:
    Print(i)
```

The only values are allowed in bytes are 0 to 256.

Once we create bytes data type we cannot change values.

```
x = bytes(5)
print(x)//b'\x00\x00\x00\x00'
```

Each \x00 is a byte with the hexadecimal value \x00, which is equivalent to the decimal value 0.

```
print(type(x))
str="hello"
tuple=(9,5,28,90)
Range=range(20)
set={2,3,4,5}
frozenset=frozenset(set)
blist=bytes(list)
print(blist)
b'\x01\x02\xff\x04'
blist[0]
1
blist[1]
2
```

blist[2]

255

blist[3]

4

Bstr=bytes(str,'utf-8')

print(Bstr)

b'hello'

Bstr.decode() #if we want to display like normal variable.

'hello'

We cannot convert complex type to bytes.

And also, we cannot convert float type to bytes.

All string methods will work on bytes and bytearray

Wherever we use bytes there we can use bytearray also but only difference is that here we can bytearary is mutable whereas bytes are immutable.

Range=range(20)

brange=bytes(Range)

print(brange)

b'\x00\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12\x13'

Here some values print like \t etc. because if any value is having Unicode character, then it will print otherwise normal numbers will print i.e. if printable characters are there then it will print like that , otherwise it will print as hexadecimal values.

If we convert dictionary values to byte there here only keys will consider and display values are not considered.

bytearray: It is same as bytes data type, but here we can modify.

x = bytearray(5)

print(x)//(b'\x00\x00\x00\x00\x00')

print(type(x))

```
x=[10,20,30,40]
b=bytearray(x)
for i in b:
    print(i)
b[0]=100
for i in b:
    print(i)
str="rupa"
bastr=bytearray(str,encoding='utf-8')
print(bastr)
bytearray(b'rupa')
bastr[1]='a'
    bastr[1]='a'
Type Error: 'str' object cannot be interpreted as an integer
bastr[1]=ord('a')
print(bastr)
bytearray(b'rapa')
```

Memoryview: it works only on bytes and bytearray. Here in memory view one default function is there i.e. **readonly()** it gives true for immutable objects it gives false for mutable ones

```
x = memoryview(bytes(5))
print(x)// 0x000001E14EBDB1C0
print(type(x))
```

complex:

This are used in real time like: a.c current, resistance, voltage, polynomial equations, signal processing control theory, electro magnetism, dynamic equations.

```
x = complex(1j)
```

c=10+5j

c.real

10.0

c.imag

5.0

Complex(10,-2)

10-2j

Complex(True,False)

1+0j

Compound data type can store more than one value at a time

list: an ordered, mutable, heterogenous collection of elements where duplicates are allowed.

insertion order is preserved

heterogenous objects are allowed

duplicates are allowed

growable in nature

values should be enclosed within square brackets

x=["apple","banana","grapes"]

print(x)

type(x)

l=[1,2,3]

Memory will be allocated for 4 blocks if we add any element then we the memory space will be added 4 blocks now it will be 8 blocks like memory will be allocated like 0,4,8,16,24,32.

Taking list as input:

a=input().split()

a

['1', '2' ,'3', '4']# it gives output as string format if we don't want that manner then use below one;

List(map(int,input().split()))# if you want float then in place of int give float here it will accepts the values that which we are given like if we give int it only takes integer if it is float it takes only float.

123 345 5677

[123,345,5677]

lists are mutable

squares=[1,4,9,16,25]

squares[0]=24

print(squares)

[24, 4, 9, 16, 25]

Slist=[1,10,20,30]

Sum(slist)

81

S2list=[1,2,4.3,True]

Sum(s2list)# **here sum values in list it will be Boolean or integer or float and other values are not supported.**

List() constructor

thislist = list(("apple", "banana", "cherry"))

print(thislist)

For access:

thislist = ["apple", "banana", "cherry"]

print(thislist[1])

Negative index:

print(thislist[::-1])

['orange', 'lemon', 'cherry', 'banana', 'PPLE']

Check if item exists:

```
thislist = ["apple", "banana", "cherry"]
if "apple" in thislist:
  print("Yes, 'apple' is in the fruits list")
```

Change item value:

```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)
```

Change a range of item values:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]
thislist[1:3] = ["blackcurrant", "watermelon"]
print(thislist)
```

If you insert more than you replace, the new items will be inserted where you specified, and the remaining items will move accordingly.

```
thislist = ["apple", "banana", "cherry"]
thislist[1:2] = ["blackcurrant", "watermelon"]
print(thislist)//[“apple”,”blackcurent”,”watermelon”,”cherry”]
```

If you insert less items than you replace, the new items will be inserted where you specified, and remaining items will move accordingly:

```
thislist = ["apple", "banana", "cherry"]
thislist[1:3] = ["watermelon"]
print(thislist)//[“apple”,”watermelon”,”cherry”]
```

List methods:

append(): To add an item to the end of the list, use the append() method:

```
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)//[“apple”,”banana”,”cherry”,”orange”]
```

a.append([1,2,3])

print(a)

[1, 2, 3, [1, 2, 3]]

a.append("banana")

print(a)

[1, 2, 3, [1, 2, 3], 'banana']

a.append("apple","banana")

    a.append("apple","banana")

TypeError: list.append() takes exactly one argument (2 given)

a.append(("apple","banan"))

print(a)

[1, 2, 3, [1, 2, 3], 'banana', ('apple', 'Banan')]

a.append({1:"rupa",2:"devi"})

print(a)

[1, 2, 3, [1, 2, 3], 'banana', ('apple', 'banan'), {1: 'rupa', 2: 'devi'}]

a.append({1,2,3})

print(a)

[1, 2, 3, [1, 2, 3], 'banana', ('apple', 'banan'), {1: 'rupa', 2: 'devi'}, {1, 2, 3}]

clear(): The clear() method empties the list. The list still remains, but it has no content.

thislist = ["apple", "banana", "cherry"]

thislist.clear()

print(thislist)

a.clear("apple")

TypeError: list.clear() takes no arguments (1 given)

copy():

There are 2 types of copy are there

deep copy()

shallow copy()

In order to use these copies, we have to import copy module

thislist.copy(a)

TypeError: list.copy() takes no arguments (1 given)

```
thislist = ["apple", "banana", "cherry"]
mylist=thislist.copy()
print(mylist)
array=[12,19,13,[90,45]]
array1=array.copy()
print(array1)
[12, 19, 13, [90, 45]]
array.append(890)
print(array)
[12, 19, 13, [90, 45], 890]
print(array1)
[12, 19, 13, [90, 45]]
array[3][0]=23
print(array)
[12, 19, 13, [23, 45], 890]
print(array1)
[12, 19, 13, [23, 45]]
```

Deepcopy: here any changes made to a copy of the object do not reflect in the original object.

```
import copy
array2=copy.deepcopy(array)
print(array2)
[12, 19, 13, [23, 45], 890]
```

array[3][0]=45

SyntaxError: invalid syntax

array[3][1]=34

print(array)

[12, 19, 13, [23, 34], 890]

print(array2)

[12, 19, 13, [23, 45], 890]

Shallow copy()"here any changes made to a copy of an object do reflect in the original object

or

thislist = ["apple", "banana", "cherry"]

mylist = list(thislist)

print(mylist)

count(): It will count how many times that the element is there in the given list.

str=["app","appl"]

str.count("appl")

1

str.count([1,2,3])

0

str.count({1,2,3})

0

str.count({1:"edd",2:"ced"})

0

str.count()# error need an argument

s=[1,2,3,4,5,5,6]

s.count(3,2,4)

TypeError: list.count() takes exactly one argument (3 given)

index():it will give that where that particular element is located, we can give in ranges also.

```
array=[2,90,3,4,10,11,'As',7]
array.index('As')
6
array.index(90,1,4)#here  1 is start value and 4 is end value as we know end value is always excluded.
```

If we want to change value using index

```
A = ['a', 'r', 'k', 1, 2, 3, 4, 5, 6, 'r']
A[A.index(6)] = 100
print(A)
```

extend()

To append elements from *another list* to the current list, use the extend () method.

```
thislist = ["apple", "banana", "cherry"]
tropical = ["mango", "pineapple", "papaya"]
thislist.extend(tropical)
print(thislist)
```

1.

```
list1 = ["a", "b", "c"]
list2 = [1, 2, 3]
for x in list2:
  list1.append(x)
print(list1)
```

2.

```
a=["aplle","baanj","dsaf"]
b=("jhjhjh","jhjhkjh")
a.extend(b)
print(a)
['aplle', 'baanj', 'dsaf', 'jhjhjh', 'jhjhkjh']
```

c={1,2,3,4}

c.extend(a)

AttributeError: 'set' object has no attribute 'extend'

c={1,2,3}

a.extend(c)

print(a)

['apple', 'banana', 'deep', 1, 2, 3]

d={1:"fgg",2:"dfff"}

a.extend(d)

print(a)

['apple', 'banana', 'deep', 1, 2, 3, 1, 2] # here for dictionaries it takes only keys values are not considered

Add any iterable:

you can add any iterable objects like tuple, list etc.

thislist = ["apple", "banana", "cherry"]

thistuple = ("kiwi", "orange")

thislist.extend(thistuple)

print(thislist)

del:

The del keyword also removes the specified index: here this will take only index

thislist=["apple", "banana", "cherry"]

del thislist[0]

print(thislist)

The del keyword can also delete the list completely.

thislist = ["apple", "banana", "cherry"]

del thislist

insert():

To insert a new list item, without replacing any of the existing values, we can use the insert() method. The insert() method inserts an item at the specified index:

Insert only excepts two arguments.

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(2,"watermelon")
print(thislist)
str=["apple","bananan","ornage"]
str.insert(2,"rupa")
print(str)
```

['apple', 'bananan', 'rupa', 'ornage']

pop(): The pop() method removes the specified index.

```
thislist = ["apple", "banana", "cherry"]
thislist.pop(1)
print(thislist)
```

If you don't specify the index pop will remove the last item.

```
thislist.pop("apple")
```

TypeError: 'str' object cannot be interpreted as an integer

remove()

The remove() method removes the specified item.

Here definitely we have to specify the item not index.

```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
```

If more than one item with the specified value, the remove() method removes the first occurrence.

```
thislist.remove("cherry",3)
```

TypeError: list.remove() takes exactly one argument (2 given)

reverse()

The reverse() method reverses the current sorting order of the elements.

```
thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.reverse()
print(thislist)
```

sort()

List objects have a sort() method that will sort the list alphanumerically, ascending, by default:

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort()
print(thislist)
thislist = [100, 50, 65, 82, 23]
thislist.sort()
print(thislist)
```

sort descending:

To sort descending, use the keyword argument **reverse = True:**

```
thislist =["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort(reverse =True)
print(thislist)
```

case insensitive sort:

By default, the **sort () method is case sensitive, resulting in all capital letters being sorted before lower case letters**:

```
thislist =["banana", "Orange", "Kiwi", "cherry"]
thislist.sort()
print(thislist)
```

Luckily, we can use built-in functions as key functions when sorting a list. So if you want a case-insensitive sort function, use str.lower as a key function:

```
thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.sort(key = str.lower)
print(thislist)
```

Loop Lists:

```
thislist = ["apple", "banana", "cherry"]
for x in thislist:
  print(x)
thislist = ["apple", "banana", "cherry"]
for i in range(len(thislist)):
  print(thislist[i])
```

o/p:apple

banana

cherry

using while:

```
thislist = ["apple", "banana", "cherry"]
i = 0
while i < len(thislist):
  print(thislist[i])
  i = i + 1
```

List Comprehension: It provides us with the short and a concise way to construct a new sequence**.** List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = []
for x in fruits:
  if "a" in x:
```

```
    newlist.append(x)
print(newlist)
```

syntax:

newlist = [*expression* for *item* in *iterable* if *condition* == True]

The return value is a new list, leaving the old list unchanged.

Condition: The *condition* is like a filter that only accepts the items that valuate to True.

```
newlist = [x for x in fruits if x != "apple"]
string = 'Hello'
L = [1,14,5,9,12]
M = ['one', 'two', 'three', 'four', 'five', 'six']
```

List comprehension Resulting list

```
[0 for i in range(10)]
[0,0,0,0,0,0,0,0,0,0]
[i**2 for i in range(1,8)]
[1,4,9,16,25,36,49]
 [i*10 for i in L]
[10,140,50,90,120]
[c*2 for c in string]
['HH','ee','ll','llsss','oo']
[m[0] for m in M]
['o','t','t','f','f','s']
[i for i in L if i<10]
 [1,5,9]
[m[0] for m in M if len(m)==3]
 ['o','t',s']
L = [[i,j] for i in range(2) for j in range(2)]
```

[[0, 0], [0, 1], [1, 0], [1, 1]]

This is the equivalent of the following code:

[[i,j] for i in range(4) for j in range(i)]

[[1, 0], [2, 0], [2, 1], [s

3, 0], [3, 1], [3, 2]]

The *condition* is optional and can be omitted

newlist = [x for x in fruits]

The *iterable* can be any iterable object, like a list, tuple, set etc.

newlist = [x for x in range(10)]

Accept only numbers lower than 5:

newlist = [x for x in range(10) if x < 5]

ExpressionThe *expression* is the current item in the iteration, but it is also the outcome, which you can manipulate before it ends up like a list item in the new list:

newlist = [x.upper() for x in fruits]

Tuple: it is exactly same as list but it is immutable

x=("apple",banana","grape")

print(x)

type(x)

Methods that not work in tuple:

append()

clear()

extend()

pop()

remove()

reverse()

sort()

del()

Functions that work:

count()

index()

sorted:

sorted(["python","is","a","programming","langage"])

['a', 'is', 'langage', 'programming', 'python']

sorted(a,reverse=True)

[4, 3, 2, 1]

s=("dfghjk","fgyhjk","tyhjuy")

sorted(s,key=len)

['dfghjk', 'fgyhjk', 'tyhjuy']

sorted(s,key=str.lower)

['dfghjk', 'fgyhjk', 'tyhjuy']

Tuple length:

thistuple = ("apple", "banana", "cherry")

print(len(thistuple))

a=(2,4,5,1,4)

sorted(a)

[1, 2, 4, 4, 5]

Create tuple with one item:

thistuple = ("apple",)

print(type(thistuple))

#NOT a tuple

```
thistuple = ("apple")
print(type(thistuple))
```

Tuple constructor:

```
thistuple = tuple(("apple", "banana", "cherry")) # note the double round-brackets
print(thistuple)
```

tuple accessing is same as list access.

Change tuple values:

Once a tuple is created, you cannot change its values. Tuples are **unchangeable**, or **immutable**.

If we want to change tuple then convert that into list then add the element and then convert into tuple:

```
x = ("apple", "banana", "cherry")
y = list(x)#converting tuple to list
y[1] ="kiwi"
x = tuple(y)#converting list to tuple
print(x)
```

Add tuple to a tuple. You are allowed to add tuples to tuples, so if you want to add one item, (or many), create a new tuple with the item(s), and add it to the existing tuple:thistuple = ("apple", "banana", "cherry")

```
y = ("orange",)
thistuple += y
print(thistuple)
```

Tuples are **unchangeable**, so you cannot remove items from it,

```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.remove("apple")
thistuple = tuple(y)
```

If the number of variables is less than the number of values, you can add an * to the variable name and the values will be assigned to the variable as a list:

```
fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")
(green, yellow, *red) = fruits
print(green)#apple
print(yellow)#banana
print(red)#['cherry','rasberry','strawberry']
```

If the asterisk is added to another variable name than the last, Python will assign values to the variable until the number of values left matches the number of variables left

```
fruits = ("apple", "mango", "papaya","pineapple", "cherry")
(green, *tropic, red) = fruits
print(green)
print(tropic)
print(red)
```

Loop through

```
thistuple = ("apple", "banana", "cherry")
for x in thistuple:
  print(x)
```

loop through index numbers:

```
thistuple=("apple", "banana", "cherry")
for i in range(len(thistuple)):
print(thistuple[i])
```

using while loop:

Use the len() function to determine the length of the tuple, then start at 0 and loop your way through the tuple items by referring to their indexes.

```
thistuple = ("apple", "banana", "cherry")
i = 0
while i < len(thistuple):
  print(thistuple[i])
  i = i + 1
```

join tuples:

```
tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)
tuple3 = tuple1 + tuple2
print(tuple3)
```

multiply tuples: This operation is possible for list, tuple. But not possible for set and dictionary.

```
fruits = ("apple", "banana", "cherry")
mytuple = fruits * 2
print(mytuple)
```

count():returns the number of times a specified value occurs in a tuple.

Index():searches the tuple for a specified value and returns the position of where it was found.

Dictionary: Duplicate keys are not allowed**.** If we try are trying to insert an entry with duplicate key the old value will be replaced with new value.

```
x={"name":"john","age":36}

print(x) //{'name': 'john', 'age': 36}

type(x) //<class 'dict'>
```

A dictionary is a key value pair set arranged in any order. It stores a specific value for each key. Value is any python object, while the key can hold any primitive data type.

The comma and curly braces are used to separate the items in the dictionary.

```
d = {1:'Jimmy',2:'Alex',3:'john',4:'mike'}
print (d)
print("1st name is "+d[1])
print("2nd name is "+ d[4])
print (d.keys())
print (d.values())
dict={1:"hello",2:"devi",3:"santhoshi"}
dict(1)
TypeError: 'dict' object is not callable
dict[3]
'santhoshi'
dict[0]
KeyError: 0
```

dict constructor:

```
thisdict =dict(name = "John", age = 36, country = "Norway")
print(thisdict)
```

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = thisdict["model"]
```

There is also a method called get() that will give you the same result:

x = thisdict.get("model")

Get Keys

The keys() method will return a list of all the keys in the dictionary.

x = thisdict.keys()

Accessing items:

You can access the items of a dictionary by referring to its key name, inside square brackets:

Get Values

The values () method will return a list of all the values in the dictionary.

x = thisdict.values()

Get Items

The items() method will return each item in a dictionary, as tuples in a list.

x = thisdict.items()

check if key exists:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
if "model" in thisdict:
print("Yes, 'model' is one of the keys in the thisdict dictionary")
```

Update Dictionary

The update () method will update the dictionary with the items from the given argument. The argument must be a dictionary, or an iterable object with key: value pairs.thisdict = {

```
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.update({"year": 2020})
```

Adding Items

Adding an item to the dictionary is done by using a new index key and assigning a value to it:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["color"] = "red"
print(thisdict)
```

Removing Items: here removing is not possible.

There are several methods to remove items from a dictionary

Pop():here in braces we have to mention at least one value, if we specify more than one value then it will takes only first one.

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.pop("model")
print(thisdict)
```

max:

max({1:3,10:5,3:1} #10

min:

min({a:3,b:5,c:7**})**

popitem():here it will delete the last item here it doesn’t except any key value in braces if we give any value than it will get error.

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}// pop item will except at least one value
thisdict.popitem()
print(thisdict)//it will remove the last item.
```

Del():here it will delete the specified index if we don’t specify index it will delete the whole.

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
```

```
  "year": 1964
}
del thisdict["model"]
print(thisdict)
```

clear():it doesn't take any argument

```
thisdict= {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.clear()
print(thisdict)
```

sorted():this will sort the key values only

sum():it will sum only key values

loop dictionaries:

Print all key names in the dictionary, one by one:

```
for x in thisdict:
  print(x)
for x in thisdict:
  print(thisdict[x])#it will give output values like ford,1984,Mustang
for x in thisdict:
   print(len(thisdict[x]))# it will print the length of the values.
for x in thisdict.values():
  print(x)
for x in thisdict.keys():
  print(x)
```

```
for x, y in thisdict.items():
  print(x, y)
```

copy():

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
mydict = thisdict.copy()
print(mydict)
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
mydict = dict(thisdict)
print(mydict)
```

nested dictionaries:

```
myfamily = {
  "child1" : {
    "name" : "Emil",
    "year" : 2004
  },
  "child2" : {
    "name" : "Tobias",
    "year" : 2007
  },
```

```
  "child3" : {
    "name" : "Linus",
    "year" : 2011
  }
}
```

access items in nested dictionaries:

print(myfamily["child2"]["name"])

set default: set default (key, value)

key we want to insert in dictionary

value the value is inserted in the key in dictionary

default value for value is parameter is None1

Fromkeys():method when you want to give same value for all keys in dictionary.

Dict.fromkeys(iterable,value)

Dict: dictionary class

Iterable: list/set/tuple/string/range/array etc.

Value:the value we want to store in keys.

Default value for value parameter is none

set: sets are very fast compared to list and tuples. because here in sets we don't use duplicate elements.

Here insertion order is not preserved

duplicates are not allowed

heterogenous objects are allowed

index concept is not applicable

It is growable in nature

It is mutable in collection

x={"apple","banana","cherry"}

print(x)// {'cherry', 'apple', 'banana'}

type(x)//<class 'set'>

Set items are unchangeable, but you can remove items and add new items.

Sets are written only in curly braces.

The values true and 1 are considered the same value in sets, and are treated as duplicates.

The values false and 0 are considered the same value in sets, and are treated aa duplicates.

Set constructor:

Thsiset=set((“apple”,”banana”,”cherry”)) **here we are using round brackets.**

Print(thiset)

```
set= {1,2,3}
set [0]
Traceback (most recent call last):
File "<pyshell#5>", line 1, in <module>
set[0]
```

TypeError: 'set' object is not sub scriptable

```
set1= {1,2,3,3,4,5,5,6,7,3}
Len(set1)
7
set1={1, 2, 3, 4, 5, 6, 7}
1 in set1
True
2 in set1
True
10 in set1
False
```

For creating empty set then we use a=set(),set doesn't support slicing and indexing because it is not a sequence data type

Frozenset is immutable we are not able to do addition all these

```
set1.add(5)
print(set1)
{1, 2, 3, 4, 5, 6, 7}
```

we can able to add only immutable concepts like tuple etc.

```
set1.add[1,2,3]
TypeError: 'builtin_function_or_method' object is not subscriptable
set1.add({1,2,3})
TypeError: unhashable type: 'set'
set1.add(1,2,3)
 set1.add(1,2,3)
```

TypeError: set. Add() takes exactly one argument (3 given)

```
set1.add(1)
print(set1)
{1, 2, 3, 4, 5, 6, 7}
set1.add[2]
   set1.add[2]
TypeError: 'builtin_function_or_method' object is not subscript able
set3=set()
```

sorted():sorted(s)

isdisjoint: **if set1 and set2 not contains any common element then it will return true**

```
>>> set1= {1,2,3}
>>> set2= {2,3,4}
>>> set1.isdisjoint(set2)
```

```
False
>>> set2.remove(3)
>>> print(set2)
{2, 4}
```

Issubset():

```
>>> set1={1,2,3,4,5,6}
>>> set2={2,3}
>>> set2.issubset(set1)
True
```

issuperset

```
>>> set1.issuperset(set2)
True
>>> set1>=set2
True
>>> set1==set2
False
>>> set1<=set2
False
>>> set1.union(set2)
{1, 2, 3, 4, 5, 6}
```

frozen set: frozen set is similar to set but it is immutable

so, all the update operations don't work, remaining all functions work

even in frozen set we cannot add mutable data types like list, set etc.

Access items:

you cannot access items in a set by referring to an index or a key.

But you can loop through the set items using a for loop, or ask if a specified value is present in asset, by using the in keyword.

```
thisset = {"apple","banana", "cherry"}
for x in thisset:
  print(x)
thisset = {"apple", "banana", "cherry"}
print("banana" in thisset)
thisset = {"apple", "banana", "cherry"}
print("banana" not in thisset)
```

once item is created you cannot change its items but you can add new items.

To add one item to a set, use the add () method.

Add():here add only adds one element.

```
thisset = {"apple", "banana", "cherry"}
thisset.add("orange")//add only takes one argument
print(thisset)
s={1,2,3,4}
s.add(((1,2,3),(2,3,4)))
print(s)
```

Update():here we can add so many elements.

```
thisset = {"apple", "banana", "cherry"}
tropical = {"pineapple", "mango", "papaya"}
thisset.update(tropical)
print(thisset)
```

Add Any Iterable

The object in the update () method does not have to be a set, it can be any iterable object (tuples, lists, dictionaries etc.).

```
thisset ={"apple", "banana", "cherry"}
mylist =["kiwi", "orange"]
```

```
thisset.update(mylist)
print(thisset)
```

Remove Item

To remove an item in a set, use the remove (), or the discard () here remove will delete the all elements not first one, here it takes only one argument.

```
method.thisset = {"apple", "banana", "cherry"}
thisset.remove("banana")
print(thisset)
```

If the item to remove does not exist, remove () will raise an error.

```
thisset = {"apple", "banana", "cherry"}
```

discard():

```
thisset.discard("banana")
print(thisset)
```

If the item to remove does not exist, discard () will **NOT** raise an error.

Pop()

You can also use the pop () method to remove an item, but this method will remove a random item, so you cannot be sure what item that gets removed. The return value of the pop() method is the removed item.Here in set it doesn't take any argument it will default removes the first element and return the popped element

```
thisset = {"apple", "banana", "cherry"}
x = thisset.pop()

print(x)
print(thisset)
```

clear():

```
thisset={"apple", "banana", "cherry"}
thisset.clear()
print(thisset)
```

del():

```
thisset ={"apple", "banana", "cherry"}
del thisset
print(thisset)
```

loop items:

```
thisset = {"apple", "banana", "cherry"}
for x in thisset:
  print(x)
```

Union

The union () method returns a new set with all items from both sets.

```
set1= {"a", "b", "c"}
set2= {1, 2, 3}
set3=set1.union(set2)
print(set3)
```

You can use the | operator instead of the union () method, and you will get the same result.

```
set1 = {"a", "b", "c"}
set2 = {1, 2, 3}
set3 = set1 | set2
print(set3)
```

All the joining methods and operators can be used to join multiple sets. When using a method, just add more sets in the parentheses, separated by commas:

```
set1 = {"a", "b", "c"}
set2 = {1, 2, 3}
set3 ={"John", "Elena"}
set4 = {"apple", "bananas", "cherry"}
myset = set1.union(set2, set3, set4)
print(myset)
set1 = {"a", "b", "c"}
set2 = {1, 2, 3}
set3 = {"John", "Elena"}
set4 = {"apple", "bananas", "cherry"}
myset = set1 | set2 | set3 |set4
print(myset)
x = {"a", "b", "c"}
y = (1, 2, 3)
z = x.union(y)
print(z)
```

The | operator only allows you to join sets with sets, and not with other data types like you can with the union () method

Update

The update () method inserts all items from one set into another. The update () changes the original set, and does not return a new set

```
set1 = {"a", "b", "c"}
set2 = {1, 2, 3}
set1.update(set2)
print(set1)
```

Both union () and update () will exclude any duplicate items.

Intersection

Keep ONLY the duplicates

The intersection () method will return a new set, that only contains the items that are present in both sets.

```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "Microsoft", "apple"}
set3 = set1.intersection(set2)
print(set3)
```

You can use the & operator instead of the intersection () method, and you will get the same result.

```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}
set3 = set1 & set2
print(set3)
```

The & operator only allows you to join sets with sets, and not with other data types like you can with the intersection() method.

The intersection_update() method will also keep ONLY the duplicates, but it will change the original set instead of returning a new set.

```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}
set1.intersection_update(set2)
print(set1)
```

Difference

The difference() method will return a new set that will contain only the items from the first set that are not present in the other set.

```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "Microsoft", "apple"}
set3 = set1.difference(set2)
print(set3)
```

You can use the - operator instead of the difference() method, and you will get the same result.

```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}
set3 = set1 - set2
print(set3)
```

The - operator only allows you to join sets with sets, and not with other data types like you can with the difference() method.

difference_update() method will also keep the items from the first set that are not in the other set, but it will change the original set instead of returning a new set

```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}
set1.difference_update(set2)
print(set1)
```

Symmetric Differences:The symmetric_difference() method will keep only the elements that are NOT present in both sets.

```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "Microsoft", "apple"}
set3 = set1.symmetric_difference(set2)
print(set3)
```

You can use the ^ operator instead of the symmetric_difference() method, and you will get the same result.

```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}
set3 = set1 ^ set2
print(set3)
```

The ^ operator only allows you to join sets with sets, and not with other data types like you can with the symmetric_difference() method.

The symmetric_difference_update() method will also keep all but the duplicates, but it will change the original set instead of returning a new set.

```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}
set1.symmetric_difference_update(set2)
print(set1)
```

None Data Type:

None means nothing or No value associated.

If the value is not available, then to handle such type sof cases None introduced.

It is something like null value in Java.

Constants concept is not allowed in python.

separator:

```
print(a,b,c,sep="}")
10}20}30
print(a,b,c sep="}")#here mentioning comma is necessary
SyntaxError: invalid syntax
```

End:

It prints new line.

<u>String format</u>:

Byte code instruction that has passed to format string it is at least 40 times more i.e. no. of instructions for format string is far higher than normal constant value. and also, time taken is also very less than others

```
import time
a=2
b=3
c=a+b
start_time = time.time()
print("The result of adding",a,"and",b,"is",c)
print("--- %s micro seconds ---" %((time.time()- start_time)*10**6))
start_time=time.time()
print(f"the result of adding {a} and {b} is {c}")
print("--- %s micro seconds ---" %((timess.time()-start_time)*10**6))
```

```
The result of adding 2 and 3 is 5
--- 5329.370498657227 micro seconds ---
the result of adding 2 and 3 is 5
--- 700.7122039794922 micro seconds ---
```

```
The result of adding 2 and 3 is 5
--- 5329.370498657227 micro seconds ---
the result of adding 2 and 3 is 5
--- 700.7122039794922 micro seconds ---
```

1. using the comma operator

```
s="durga"
a=48
s1="java"
s2="python"
print("hello",s,"your age is",a)
```

hello durga your age is 48

print("you are teaching" ,s1 ,"and" ,s2)//**here commas are very important if we miss commas then we will get error.**

you are teaching java and python

2.formatted string:

print("formatted string" %(variable list))

a=10

b=20

c=30

print("a value is %d"%a)

a value is 10

print("b value is %d and c value is %d"%(b,c))

b value is 20 and c value is 30

a=10

b=20

print("a value is %d and c value is %d"%(a,b))

a value is 10 and c value is

s="durga"

list=[10,20,30,40]

print("hello %s the list items are %s"%(s,list))

hello durga the list items are [10, 20, 30, 40]

3.print() with replacement operator {}

name="durga"

salary=10000

gf="sunny"

print("hello {0} your salary is{1} and your friend {2} is waiting".format(name,salary,gf))

hello durga your salary is10000 and your friend sunny is waiting

4.format string: the efficiency for these is higher than others even though it is having so many byte code instructions.

(f"addition of {x} and {y} is {z}")

Type conversion:

If we want to convert from any type to int except complex type.

If we want to convert str type to int compulsory str should contain only integer value and should be in decimal form.

We can convert any type to float except complex

You can convert from one type to another with the int(), float(), and complex() methods:

```
x=1
y=2.8
z=1j
a=float(x)
print(a)
1.0
b=int(y)
print(b)
2
c=complex(x)
print(c)
(1+0j)
type(C)
```

you cannot convert complex numbers into another number type.

Random number:

```
from random import randint
randint(a,b)
```

will return a random integer between a and b including both a and b. (Note that randint includes the right endpoint b unlike the range function).

Here is a short example:

```
from random import randint
 x = randint(1,10)
print('A random number between 1 and 10: ', x)
```

Import the random module and display a random number between 1 and 9:

```
Import random
Print(random.randrange(1,10))//3 here it only shows one value between the range not all values between them.
from math import sin, pi
print('Pi is roughly', pi)
print('sin(0) =', sin(0))
```

Built-in math functions

There are two built in math functions, a

abs (absolute value) and

round that are available without importing the math module. Here are some examples:

```
print(abs(-4.3))
print(round(3.336, 2))
print(round(345.2, -1))
```

The round function takes two arguments: the first is the number to be rounded and the second is the number of decimal places to round to. The second argument can be negative

max:

syntax: max(iterable,key=function)

here iterable may be list, tuple, set, string, dictionary, iterators, &generators

key=built in function

or user defined function

or lambda function

```
max(["fggg","rupa","devi"])
'rupa'
max(["shhhhh","ggg","ddd"])
'shhhhh'
<built-in function max>
max(["abcd","welcome","devi"])
'welcome'
max(["apple","deee","hjj"],key=len)
'apple'
max([[1,3,1,9],[10,20,0,3],[0,0,100,1]])
[10, 20, 0, 3]
max([[1,3,9,1],[2,3,4,5],[4,5,6,7]])
[4, 5, 6, 7]
max([[1,3,1,9],[2,3,4,5],[3,4,5,6]],key=lambda i :[2])
[1, 3, 1, 9]
max([[1,3,1,9],[2,3,4,5],[3,4,5,6]],key=lambda i :i][1])
[1, 3, 1, 9]
max([[1,3,1,9],[2,3,4,5],[3,4,5,6]],key=lambda i :i[2])
[3, 4, 5, 6]
max([[1,3,1,9],[2,3,4,5],[3,4,5,6]],key=lambda i :i[3])
[1, 3, 1, 9]
count1 = 0
count2 = 0
for i in range(10):
   num = eval(input('Enter a number: '))
```

```
if num>10:
  count1=count1+1
if num==0:
  count2=count2+1
print ('There are', count1, 'numbers greater than 10.')
print ('There are', count2, 'zeroes.')
```

Next, we have a slightly trickier example. This program counts how many of the squares from 1 to 101 end with 4.

1.
```
count = 0
  for i in range(1,101):
     if (i**2)%10==4:
        count = count + 1
 print(count)
```

2.
```
s = 0
  for i in range(1,101):
     s = s + i
  print('The sum is', s)
```

3.
```
s = 0
 for i in range(10):
    num = eval(input('Enter a number: '))
    s = s + num
print('The average is', s/10)
```

4.
```
num = eval(input('Enter number: '))
  flag = 0
 for i in range(2,num):
    if num%i==0:
       flag = 1
```

```
  if flag==1:
   print('Not prime')
else:
   print('Prime')
```

5.

```
from random import randint
  for i in range(randint(5,25)):
   print('Hello')
```

6.

```
from random import randint
   count = 0
   for i in range(10000):
     num = randint(1, 100)
       if num%12==0:
  count=count+1
print('Number of multiples of 12:', count)
```

7.

```
s = input('Enter a string')
if s[0].isalpha():
    print('Your string starts with a letter')
if not s.isalpha():
     sprint('Your string contains a non-letter.')
print(s)
```

Slicing:

S[beginindex:endindex:step]

If we are not specifying the begin index it will consider as from the beginning of the string.

If we are not specifying the end index it will consider up to the end of string.

The default value for step is 1.

If it is positive then it should be forward direction (left to right).

If it is backward then right to left.

s="abcdefgjk"

s[::]#'abcdefgjk'

s[:]#'abcdefgjk'

s[::1]#'abcdefgjk'

s[::-1]#'kjgfedcba'

s[::-2]#'kgeca'

slice operator never raises index error.

Immutable:

All Fundamental Data types are immutable. i.e. once we create an object, we cannot perform any changes in that object. If we are trying to change then with those changes a new object will be created. This non-changeable behavior is called immutability.

In Python if a new object is required, then PVM won't create object immediately. First it will check is any object available with the required content or not. If available then existing object will be reused. If it is not available then only a new object will be created. The advantage of this approach is memory utilization and performance will be improved.

a=10

b=10

id(a)

140709528069192

id(b)

140709528069192

a is b

True

a=10+5j

b=10+5j

id(a)

2282842976176

id(b)

2282842973808

a is b

False

a=True

b=True

id(a)

140709526543208

id(b)

140709526543208

a is b

True

a="durga"

b="durga"

id(a)

2282849096880

id(b)

2282849096880

a is b

True

Strings:

In Python, we can represent char values also by using str type and explicitly char type is not available. Strings in python are surrounded by either single or double quotations.

'hello' or "hello"

By using single quotes or double quotes we cannot represent multi line string literals.

String for concatenation we use '+' operator and * we use for repetition.

1.s = input('Enter some text: ')

```
for i in range(len(s)):
   if s[i]=='a':
        print(i)
```

```
word="python"
word[0]='j'
word[0]='j'
```

TypeError:'str' object does not support item assignment

If we want to change then

'j'+word[1:]

'jython'

```
S=" hello"
Print(s)
del s #it will show s is not defined
del s[0]# string doesn't support item declaration
```

Strings are arrays:

```
A=" hello world"
Print(a[1])
```

Python does not have a character data type a single character is simply a string with a length of 1.

<u>Looping through string:</u>

```
 For x in "banana":
    Print(x)
```

String length:

```
a="hello world"
print(len(a))
```

check string:

txt=" the best things in life are free"

print ("free" in txt) //true

txt=" the best things in life are feel"

if "free" in txt:

    print ("yes, 'free 'in present")//yes free in text

txt="the best things in life are free"

if "expensive" not in txt:

    print("No, expensive is not present")

txt="the best things in life are free"

print("expensive" not in txt)//True

Lower to upper and vice versa ()

a="hello world"

print(**a.upper**())

HELLO WORLD

b="HELLO WORLD"

print(**a.lower**())

hello world

strip()

Here strip removes all the space between start and end

It can remove any leading and trailing characters not only spaces it will remove characters. Not only single character it will remove group of characters.

"####Hello world###.strip('#')

hello world

"   ###hello World###".strip('#)

o/p:###Hello world

Here before hello world no # removed because it always starts from here first we have a white space character

strip() method always starts from the first character and it will never see the rest of the strings.

"hello world".strip('ldoh')

Ello wors

It starts from beginning of the string if in strip any one match then it will delete

i.e it is first compare h with ldoh so here h is matched then h will be removed

and e is not matching with any character then strip () method will stop comparing

now it starts from the last character

print(a. strip())

hello world

a="hello world"

print(a.strip())

hello world

a=" hello world "

print(a.strip())

hello world

rstrip():

fff="###gghh##"

print(fff.rstrip())

###ghhh

lstrip():by default removes the leading whitespaces only.

print(str.lstrip())

replace(): replace not only used for single characters it used for group of characters.

a="hello world"

print(a.replace("h","j"))

jello world

a="rupa"

print(a.replace('r','j'))

jupa

a=”chennamsetty santhoshi”

print(a.replace(“santhoshi”,”rupadevi”))

chennamsetty rupadevi

string.replace(old,new,count)

a=d3v3lopm3nt

a.replace(‘3’,’e’,2)

developm3nt

print(str.replace("things","text"))

#######STRANGER THIMGS

str="this is demo.this is another demo"

print(str.repalce("demo

SyntaxError: incomplete input

print(str.replace("demo","text",2))

this is text.this is another text

str="this is demo.this is another demo demo"

print(str.replace("demo","text",3))

this is text.this is another text text

print(str.replace("is","hjjj"))

thhjjj hjjj demo.thhjjj hjjj another demo demo

split()

a = "Hello, World!"

b = a.split(",")

print(b)

['Hello', ' World!']

b=a.split("&&")

print(b)

['hello', 'world']

b=a.split("llo")

print(b)

['he', '&&world']

c="santhoshi rupadevi"

c.split("thoshi")

['san', ' rupadevi']

Capitalize:

print(a.capitalize())//here it capitalize the first character in the given string

Hello everyone my name is santhoshi i am from Emani

Only capitalize the first word.

Casefold(): here it converts all letter into small case here only convert the capital letters into small letters only capital letters

a="ABHINAYA"

print(a.casefold())

abhinaya

Title: title converts whole name to each word to capital

a=”dggg gyhyhhh gghyh”

print(s.title())

Dgghh Gghh Ggguj

print(str.title())

One

Two

Egg

center():returns a centers string. Here we can able to add only one character we are not able to add more characters.

returns a centers string.

str="demo”

str.center(10,'#')

'###demo###'

str.center(20,'&')

'&&&&&&&&demo&&&&&&&&'

str.center(35,'%%%%')

TypeError: The fill character must be exactly one character long

count():returns the number of times a specified value occurs in a string

returns the number of times a specified value occurs in a string

str="this is demo"

str="this is demo this is another demo"

print(str.count("demo"))

2

print(str.count("demo",7,25))

1

s="san thos sgh"

s.count("san","thos")

TypeError: slice indices must be integers or None or have an __index__ method

encode():returns an encoded version of the string.

endswith():returns true if the string ends with the specified value.

returns true if the string ends with the specified value.

str="this is demo text"

str="this is demo text"

print(str.endswith("text"))

True

print(str.endswith("text",12,17))

True

Print(str.endswith('t'))

True

s.endswith("rupa","devi")

TypeError: slice indices must be integers or None or have an __index__ method

expandtabs()sets the tab size of the string

str="d\te\tm\to\t"

print(str)

d e m o

print(str.expandtabs())

d e m o

str="demo"

print(str.expandtabs())

demo

str="d\te\tm\to\t"

print(str.expandtabs(2))

d e m o

find():searches the string for the specified value and returns the position of where it was found.

```
str.find("text")
-1
str.find("this")
0
str="this is first text"
print(str.find("text"))
14
print(str.find("text",20,40))
-1
print(str.find("text',12,17))
SyntaxError: incomplete input
s.find(("fdff ,ggg"))
-1
str.find("text",12,17))

SyntaxError: unmatched ')'
str.find("text",12,17)
-1
s.find([1,2,3])
TypeError: must be str, not list
s.find("[1,2,3]")
-1
print(str.find("text",12,17))
13
```

format():formats specified values in a string.

format_map():formats specified values in a string.

index():searches the string for the specified value and returns the position of where it was found

```
str="hello my name is rupa"
print(str.index('n'))
```

9

```
print(str.index('n',2,10))
```

9

ValueError: substring not found

isalnum():returns true if all characters in the string are the alphanumeric.

```
str="jamesbond007"

print(str.isalnum())
```

True

isalpha():returns true if all characters in the string are in the are alpha.

```
str="hghjk"

print(str.isalpha())
```

True

isascii():returns true if all characters in the string are ASCII characters.

Ascii values encodes only English letters, decimals, symbols.

isdecimal():returns true if all the characters in the string are decimals.

```
str="123"
```

print(str.isdecimal())

True

isdigit():retuens true if all the characters in the string are digits the decimal are in base superscript,subscript,base.

2^2_3

print(str.isdigit())

isidentifier():returns true if the string is an identifier

str="fghj"

print(str.isidentifier())

islower():retuns true if all the characters in the string are lower case

str="fghjk"

print(str.islower())

True

isnumeric():returns true if all characters in the string are numeric

returns true if all characters in the string are numeric the numbers are in superscript, subscript, roman numerals, fractional numbers.

str="123455"

print(str.isnumeric())True

3.15.isnumeric()

Traceback (most recent call last):

  File "<pyshell#11>", line 1, in <module>

    3.15.isnumeric()

AttributeError: 'float' object has no attribute 'isnumeric'

"3.14".isnumeric

<built-in method isnumeric of str object at 0x000002827251D030>

"3.14".isnumeric()

False

#here point is not considered as full stop is numeric value returns true if it

#uses all characters to in decimal form

SyntaxError: incomplete input

"2/4".isnumeric()

False

isprintable():returns true if all the characters in the string are printable

print(str. isprintable())

True

str="678hjkkk"

print(str.isprintable())

True

str=" "

isspace():retuns true if all charcters in the string are whitespaces.

str=" "

print(str.issapce())

istitle():retuns true if the string follows the rules of a title.

str="Amit"

print(str.istitle())

True

isupper():returns true if all the characters in the string are uppercase.

print(s.isupper())

join(): Joins the elements of an inerrable(list, dictionary ,tuple, set and string)or

compound data types to the end of the string

syntax:separator.join(iterable)

l1=['h','e','l','l','o']

''.join(l1)

Hello

L2=['I','am','of','a','varibale']

''.join(l2)

'I am a rocky'

str={"frank","sahun","mporgam"}

sep="$$"

print(sep.join(str))

mporgam$$frank$$sahun

l3=['name','of','a','variable']

'-'.join(l3)

Name_of_the_variable

D={'name':'adam','country':'us'}

o/p:it only print key values name,country.

'hello'.join('123456')

1hello2hello3hello4hello5hello6

A='some'

a.join('meme')

msomeesomemsomeesome

difference between join and add:

'abcd'.join('1234')

1abcd2abcd3abcd4

Abcd+1234

Abcd1234

ljust():retuns a left justified version of the strin

maketras():returns a translation table to be used in translations

partition():returns a tuple where the string is parted into three parts

rfind():searches the string in the specified value and returns last position where it was found

print(str.rfind())

Traceback (most recent call last):

File "<pyshell#29>", line 1, in <module>

print(str.rfind())

TypeError: rfind() takes at least 1 argument (0 given)

print(str.rfind("demo"))

34

#rfind last index it will return

rindex similar to rfind

rindex():searches the string for a specified value and returns the last position of where it was found

print(str.rindex("demo"))

rjust():returns a right justified version of the string

print(str.rjust(25,"a"))

str="american vanda"

print(str.rjust(25"""total length of string""","a"))

print(str.rjust(25,'A'))

AAAAAAAAAAAamerican vanda

rpartition():returns a tuple where the string is parted into three parts

rsplit():spits the string at the specified separator, and returns list

used to split the string from the right.

"helli!$i$am$jaspreet".rspilt('$',maxsplit=2)

['hello!$I','am','jaspreet']

rstirp():retuns right trim version of the string

retuns right trim version of the string by default only removes from the trialing whitespaces only.

print(str2.rstrip())

" solving any problem is an art. ".strip('.')

o/p:solving any problem is an art

splitliness():splits the string at the specified list and returns a list

str="one\nTwpo\nfgg\n"

print(str.splitlines(True))

['one\n', 'Twpo\n', 'fgg\n']

print(str.splitlines(False))

['one', 'Twpo', 'fgg']

startswith()returns true if the string lines starts with the specified value

print(str.startswith("one"))

True

print(str.startswith("one",2,45))

swapcase():lowercase becomes uppercase and uppercase becomes lowrecase

print(str.swapcase())

ONE

tWPO

FGG

zfill():fills the string with a specified number of 0 values at the beginning

str="wow"

print(str.zfill(7))

0000wow

print(str.zfill(10))

0000000wow

#strip method is used to remove spaces from the beginning

bool:

most values are true:

Any string is true except the empty string.

Any number is true except the 0

Any list, tuple, set, and dictionary are True, except empty ones.

The following results false:

Bool(false)

Bool(none)

Bool(0)

Bool(“”)

Bool([])

Bool({})

Operators:

Assignment operator(:=)

Print(x:=3)//3s

Arithmetic operator(+,-,*,/,%,x**y i.e. x=5,y=3,print(x**y)=5*5*5=125

**** exponent**

// floor division

a=5

b=2

a//b

2// here floor division doesn’t show decimal values.

Comparison operator

Logical operator(and,or,not)

or used in an expression returns the value of the first operand that is not a false value (False , 0 , '' , [] ..). Otherwise, it returns the last operand.

print(0 or 1) ## 1

print(False or 'hey') ## 'hey'

print('hi' or 'hey') ## 'hi'

print([] or False) ## 'False'

print(False or []) ## '[]'

Identity operator(is and is not)

and only evaluates the second argument if the first one is true. So if the first argument is falsy (False , 0 , '' , [] ..), it returns that argument. Otherwise it evaluates the second argument:

print(0 and 1) ## 0

print(1 and 0) ## 0

print(False and 'hey') ## False

print('hi' and 'hey') ## 'hey'

print([] and False) ## []

print(False and []) ## False

The math package provides general math functions and constants

The cmath package provides utilities to work with complex numbers.

The decimal package provides utilities to work with decimals and floating-point numbers.

The fractions package provides utilities to work with rational numbers

Is: Returns true if both variables are the same object

Is not: returns true if both variables are not the same object

Membership operator

in: returns true if a sequence with the specified value is present in the object

not in: returns true if a sequence with the specified value is not present in the object.

Bitwise operator

Python conditions:

Here in python, we definitely follow the indentation rules.

- If
- Elif
- Else

Ternary operator:

```
a = 2
b = 330
print("A") if a > b else print("B")
```

pass statement: The difference between pass and comment is that the interpreter completely ignores the comment whereas pass is not ignored.

if statement cannot be empty, but if you for some reason have an I statement with no content, put in the pass statement to avoid getting an error.

```
A=33
B=44
If b>a:
  pass
```

python loops:

while loop

for loop

while: In loop repeat code as long as condition is true.

```
i = 1
while i < 6:
  print(i)
  i += 1
```

while loop using break: By using break statement we can exit loop

```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

while loop using continue statement: By using continue statement skip some iterations inside loop.

```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```

while loop along with else statement: with the else statement we can run a block of code once when the condition is no longer is true:

```
i = 1
while i < 6:
  print(i)
  i += 1
else:
  print("i is no longer less than 6")
```

For:

for loop repeat code for every item in sequence.

```
for variable_name in range( number of times to repeat ):
    statements to be repeated
```

The syntax is important here. The word for must be in lowercase, the first line must end with a colon, and the statements to be repeated must be indented. Indentation is used to tell Python which statements will be repeated.

To print hello in 10 times

```
For x in range(10)
    Print("hello")
```

To display numbers from 0 to 11:

```
For x in range(11)
    Print(x)
```

To display odd numbers from 0 to 20:

```
For x in range(21):
    If(x%2!=0):
        Print(x)
```

To display numbers from 10 to 1

```
for x in range(10,0,-1)
    print(x)
```

Displaying list:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

loop through a letter in the string:

```
for x in "banana":
  print(x)
for i in range(3):
   num = eval(input('Enter a number: '))
   print ('The square of your number is', num*num)
print('The loop is now done.')
```

Since the second and third lines are indented, Python knows that these are the statements to be repeated.

The fourth line is not indented, so it is not part of the loop and only gets executed once, after the loop has completed.

```
Enter a number: 3
The square of your number is 9
Enter a number: 5
```

The square of your number is 25

Enter a number: 23

The square of your number is 529

The loop is now done.

```
print('A')
print('B')
for i in range(5):
    print('C')
    print('D')
print('E')
print('A')
print('B')
for i in range(5):
  print('C')
for i in range(5):
  print('D')
print('E')
for i in range(100): print(i)//for printing #print numbers upto 100
for i in range(5,0,-1):
print(i, end=' ')
print('Blast off!!')
```

break in for:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
  if x == "banana":
    break
```

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    break
  print(x)
```

continue:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    continue
  print(x)
```

range:

```
for x in range(6):
  print(x)
for x in range(2, 6):
  print(x)//here 2 is included but 6 is not included
```

increment the sequence(here default it increments by 1)

```
for x in range(2, 30, 3):#(satrtvalue,endvalue,inceremnt)
  print(x)
```

else in for loop:

It specifies that a block pf code of be executed when the loop is finished.

```
for x in range(6):
  print(x)
else:
  print("Finally finished!")
```

else statement will not be executed if the loop statement stopped by a break statement.

Nested loops:

The inner loop will be executed one time for each iteration of the outer loop

```
adj =["red", "big", "tasty"]
fruits =["apple", "banana", "cherry"]
for x in adj:
  for y in fruits:
    print(x, y)
```

pass:

for loops cannot be empty, but if you for some reason have a for loop with no content, put in the pass statement to avoid getting an error.

```
for x in [0, 1, 2]:
  pass
```

python arrays:

In python arrays are not supported instead of these we are using list as python arrays.

Python user input:

```
Username=input("enter username")
Print("username is:"+username)
S=int(input("enter an integer"))#here by default it takes input in string format so here we have to specify the what type of data we were using otherwise it will take that as string.
```

Address:

We can find the address of the variable by using the **id**

```
a=50
b=a
print(id(a))//140734982691168
print(id(b))//140734982691168
```

Module: A file containing a set of functions you want to include in your application.

Create a Module

To create a module just save the code you want in a file with the file extension .py:

Save this code in a file named mymodule.py

Use a Module

Now we can use the module we just created, by using the import statement:

import mymodule

mymodule.greeting("Jonathan")

module is a collection of functions, variables, classes etc.

math is a module that contains several mathematical operations.

If we want to use any module in python first, we need to import module

Import math

Print(math.sqrt(15))

Once we import the module then we can call that function of that module.

We can also create alias name for that module

Import math as m

Print(m.sqrt(24))

Print(m.pi)

We can also import explicitly if we import explicitly then no need to call

from math import sqrt, pi

Print(sqrt(15))

Print(sqrt(pi))

Important functions:

Ceil(x)

Floor(x)

Pow(x)

factorial(x)

Trunc(x)

gcd(x)

lcm(x)

Sin(x)

Cos(x)

Tan(x)

Base conversions:

bin(12)

'0b1100'

oct(10)

'0o12'

hex(100)

'0x64'

Python and java use Unicode division: Unicode is a universal international standard character encoding that is capable of representing all languages.

0-31: non printable characters

127: left array

32-47: special symbols

48-57: 0-9

58-64: special symbols

65-90: 'A'-'z'

91-96: special symbols

97-122: 'a'-'z'

123-126: special symbols

If we want to find unicode of a number: ord can give Unicode for single value

print(ord('5'))

53

print(ord('b'))

98

print(ord(96))

TypeError: ord() expected string of length 1, but int found

chr is used to know Unicode value for a double characters:

print(chr(96))

`

print(chr(126))

~

import string

print(string.ascii_letters)

abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ

print(string.ascii_lowercase)

abcdefghijklmnopqrstuvwxyz

print(string.ascii_uppercase)

ABCDEFGHIJKLMNOPQRSTUVWXYZ

print(string.digits)

0123456789

print(string.octdigits)

01234567

print(string.hexdigits)

0123456789abcdefABCDEF

Write a program to enter the given charter is letter or not

a=input("enter a letter")

p=ord(a)

```
if(p>=65 and p<=90) or (p>=97 and p<=122):
    print('letters')
else:
    print('digits')'''
```

2.

```
import string
c=input("enter a letter")
if c in string.ascii_letters:
    print('letters')
else:
    print('digits')
```

conditional statements:

1) If

2) if – else

3) if-elif-else

4) if-elif

There is no switch statement in Python here we are using match in place of string the difference between match and switch is in switch we use break keyword for every case statement but here we don't use any switch statement

If:

```
a= 33
b= 200
if b>a:
  print("b is greater than a")
```

elif:

```
a= 33
b= 33
if b>a:
```

```
  print("bisgreaterthana")
elif a==b:
  print("a and b are equal")
```

else:

```
a= 200
b= 33
if b>a:
  print("bisgreaterthana")
elif a==b:
  print("aandbareequal")
else:s
  print("a is greater than b")
```

Functions:

A function is 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 function can be created by using **def** keyword:

```
def my_fucntion():
   Print("hello from a function")
```

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

```
Def my_fucntion():
   Print("hello from a function")
My_function()
```

Arguments:

Information can be passed into functions as arguments:

Arguments are specified has a function name, inside the parentheses you can add as many arguments as you want, just separate them with a comma.

```
Def my_function(fnasme):
    Print(fname+"refsnes")
My_fucntion("emil")
My_fucntion("tobias")
my_function("linus")
```

Parameters or arguments:

The term parameter and argument can be used for the same thing:

Information that are passed into a function:

A parameter is the variable listed inside the parameters in the function definition.

An argument is the value that is sent to the function when it is called.

Number of arguments:

By default, a function must be called with the correct number of arguments. Means that if your function expects 2 arguments, you have to call the function with 2 arguments, not more, and not less.

```
Def my_function(fname, lname):
    Print(fname+" "+lname)
My_function("emil", refsenss")
```

If you try to call the function with 1 or 3 arguments, you will get ana error:

1.arbitary arguments *args:

If you don't know how many arguments that will be passed into your function, add a * before the parameter name in the function definition.

```
def my_function(*kids):
  print("The youngest child is " + kids[2])
my_function("Emil", "Tobias", "Linus")
```

o/P: The youngest child is Linus

2.Keywords arguments:

You can also send arguments with the key=value

This way the order of the arguments does not matter.

```
def my_function(child3, child2, child1):
  print("The youngest child is " + child3)
my_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus")
```

o/P: the youngest child is Linus

3.arbitary keyword arguments:

If you do not know how many keyword arguments that will be passed into your function, add two asterisk ** before the parameter name in the function definition.

```
def my_function(**kid):
  print("His last name is " + kid["lname"])
my_function(fname = "Tobias", lname = "Refsnes")
```

his last name is refsness

4.default parameter or optional parameter:

If we call the function without argument, it uses the default value:

If we don’t pass any value then default value will be assigned.

```
Def my_fuction(country=”Norway”
    Print(“I am from”+country)
My_function(“swedan”)
My_fucntion(“india”)
my_function()
my_funtion(“Brazil”)
```

passing a list as a argument:

```
def my_function(food):
  for x in food:
    print(x)
```

```
fruits = ["apple", "banana", "cherry"]
my_function(fruits)
```

o/p:apple

banana

cherry

return values:

to let a function return a value, use the return statement:

```
def my_fucntion(x):
  return 5*x
print(my_function(3))
print(my_funtion(5))
print(my_function(9))
```

15

25

45

4.positional only arguments:

You can specify that a function can have only positional arguments, or only keyword argument first value will go to first parameter

Second value will go to second parameter

To specify that a function can have only positional arguments, add, / after the arguments:

```
Def my_fucntion(x./):
  Print(X)
My_fucntion(3)#3
```

Without the , / you are allowed to use keyword arguments even if the function expects positional arguments:

```
Def my_fucntion(x):
```

```
    Print(x)
My_function(x=3)
```

o/p:3

But when adding the , / you will get an error if you try to send a keyword argument:

```
def my_function(x,                                                    /):
  print(x)
my_function(x = 3)
```

keyword only argumnets:

To specify that a function can have only keyword arguments, add *, *before* the arguments:

```
def my_function(*, x):
  print(x)
my_function(x = 3)
```

But when adding the *, / you will get an error if you try to send a positional argument:

```
def my_function(*, x):
  print(x)
my_function(3)
```

Combine Positional-Only and Keyword-Only

You can combine the two argument types in the same function.

Any argument *before* the / , are positional-only, and any argument *after* the *, are keyword-only.

```
def my_function(a, b, /, *, c, d):
  print(a + b + c + d)


my_function(5, 6, c = 7, d = 8)
```

try and exception:

The try block lets you test a block of code for errors.

The except block lets you handle the error.

The else block lets you execute code when there is no error.

The finally block lets you execute code, regardless of the result of the try- and except blocks.

Exception Handling

When an error occurs, or exception as we call it, Python will normally stop and generate an error message.

These exceptions can be handled using the try statement:

```
try:
  print(x)
except:
  print("An exception occurred")
```

Since the try block raises an error, the except block will be executed. Without the try block, the program will crash and raise an error:

Many Exceptions: You can define as many exception blocks as you want, e.g. if you want to execute a special block of code for a special kind of error:

```
try:
  print(x)
```

```
except NameError:
  print("Variable x is not defined")
except:
  print("Something else went wrong")
```

Else: You can use the else keyword to define a block of code to be executed if no errors were raised:

```
try:
  print("Hello")
except:
  print("Something went wrong")
else:
  print("Nothing went wrong")
```

Finally: The finally block, if specified, will be executed regardless if the try block raises an error or not.

```
try:
  print(x)
except:
  print("Something went wrong")
finally:
  print("The 'try except' is finished")
```

```
try:
  f = open("demofile.txt")
  try:
    f.write("Lorum Ipsum")
  except:
```

```
    print("Something went wrong when writing to the file")
  finally:
    f.close()
except:
  print("Something went wrong when opening the file")
```

Raise an exception

As a Python developer you can choose to throw an exception if a condition occurs.

To throw (or raise) an exception, use the raise keyword.

```
x = -1


if x < 0:
  raise Exception("Sorry, no numbers below zero")
```

The raise keyword is used to raise an exception.

You can define what kind of error to raise, and the text to print to the user.

```
x = "hello"
if not type(x) is int:
  raise TypeError("Only integers are allowed")
```

scope:

Local Scope

A variable created inside a function belongs to the *local scope* of that function, and can only be used inside that function.

Global Scope

A variable created in the main body of the Python code is a global variable and belongs to the global scope.

Global variables are available from within any scope, global and local.

Naming Variables

If you operate with the same variable name inside and outside of a function, Python will treat them as two separate variables, one available in the global scope (outside the function) and one available in the local scope (inside the function).

Nonlocal Keyword

The nonlocal keyword is used to work with variables inside nested functions.

The nonlocal keyword makes the variable belong to the outer function.

```
def myfunc1():
  x = "Jane"
def myfunc2():
    nonlocal x
    x = "hello"
  myfunc2()
  return x
print(myfunc1())
```

Dates:

```
import datetime
x = datetime.datetime.now()
print(x)
```

Output:

```
2024-06-12 17:47:56.805186
```

The date contains year, month, day, hour, minute, second, and microsecond.

The datetime module has many methods to return information about the date object.

Here are a few examples, you will learn more about them later in this chapter:

```
import datetime
x=datetime.datetime.now()
print(x.year)
print(x.strftime("%A"))
```

Creating Date Objects

To create a date, we can use the datetime() class (constructor) of the datetime module.

The datetime() class requires three parameters to create a date: year, month, day.

```
import datetime

x = datetime.datetime(2020, 5, 17)

print(x)
```

The datetime() class also takes parameters for time and timezone (hour, minute, second, microsecond, tzone), but they are optional, and has a default value of 0, (None for timezone).

The strftime() Method

The datetime object has a method for formatting date objects into readable strings.

The method is called strftime(), and takes one parameter, format, to specify the format of the returned string:

```
import datetime

x = datetime.datetime(2018, 6, 1)

print(x.strftime("%B"))
```

There are certain format specifiers for date:

1.%a

```
import datetime

x = datetime.datetime.now()

print(x.strftime("%a"))//wed
```

string format time, it converts date time object into a string based on a specified format representing the date and time.

2.%A

```
import datetime
x = datetime.datetime.now()
print(x.strftime("%A"))//Wednesday
```

3.%w: weekday as decimal number

```
import datetime
x = datetime.datetime.now()
print(x.strftime("%w"))#3
```

4.%d : day of the month as zero added decimal

```
import datetime
x = datetime.datetime.now()
print(x.strftime("%d"))
```

5.%b : abbreviated month name

```
import datetime
x = datetime.datetime.now()
print(x.strftime("%b"))
```

6.%B:full month name

```
import datetime
x = datetime.datetime.now()
print(x.strftime("%B"))
```

7.%m: moth as zero added decimal

```
import datetime
x = datetime.datetime.now()
print(x.strftime("%m"))
```

8.%y: year without century as a zero added decimal number

```
import datetime
x = datetime.datetime.now()
print(x.strftime("%y"))
```

9.%Y : year with century as a decimal number

```
import datetime
x = datetime.datetime.now()
```

```
print(x.strftime("%Y"))
```

10.%H:hour as a zero added decimal number(24 hour)

```
import datetime
x = datetime.datetime.now()
print(x.strftime("%H"))
```

11. %I: hour as a zero added decimal number

```
import datetime
x = datetime.datetime.now()
print(x.strftime("%I"))
```

12.%p: Am or Pm

```
import datetime
x = datetime.datetime.now()
print(x.strftime("%p"))
```

13.%M: minute as a zero added decimal

```
import datetime
x = datetime.datetime.now()
print(x.strftime("%M"))
```

14.%s: second as a zero added decimal number

```
import datetime
x = datetime.datetime.now()
print(x.strftime("%S"))
```

15.%f: microsecond as a decimal number, zero added on the left

```
import datetime
x = datetime.datetime.now()
print(x.strftime("%f"))
```

16.%Z:time zone name

17.%j day of the year as a zero added decimal number

```
import datetime
x = datetime.datetime.now()
print(x.strftime("%j"))
```

18.%U: week number of the year Sunday as the first day of all the week. All the days in ayear preceding the first Sunday are considered to be week 0

```
import datetime
x = datetime.datetime.now()
print(x.strftime("%U")
```

18.%W: week number of the year (monady as the first day if the week)all days in a new year preceding the first Monday are considered to be in a week

```
import datetime
x = datetime.datetime(2018, 5, 31)
print(x.strftime("%W"))
```

19.

```
import datetime

x = datetime.datetime.now()

print(x.strftime("%c"))
```

20.

```
import datetime

x = datetime.datetime(2018, 5, 31)

print(x.strftime("%W"))
```

21.

```
import datetime
```

```
x = datetime.datetime.now()

print(x.strftime("%c"))
```

22.

```
import datetime

x = datetime.datetime.now()

print(x.strftime("%C"))
```

23.

```
import datetime

x = datetime.datetime.now()

print(x.strftime("%x"))
```

24.

```
import datetime
x = datetime.datetime.now()
print(x.strftime("%X"))
```

25.

```
import datetime
x = datetime.datetime.now()
print(x.strftime("%%"))
```

26.

```
import datetime
x = datetime.datetime.now()
```

```
print(x.strftime("%G"))
```

27.

```
import datetime
x = datetime.datetime.now()
print(x.strftime("%u"))
```

28.

```
import datetime
x = datetime.datetime.now()
print(x.strftime("%V"))
```

Decorator:

A Python Decorator is a mechanism to wrap a Python function and modify its behavior by adding more functionality to it. We can use @ symbol to call a Python Decorator function.

File Handling:

open()

The open() function takes two parameters; *filename*, and *mode*.

There are four different methods (modes) for opening a file:

"r": read default value opens a file for reading, error if the file does not exist

"a": append: opens a file for appending, creates the file if it does not exist

"w": write opens a file for writing creates the file if it doesn't not exist.

"x": creates the specified file returns an error if the file exists

"t":text: defult value .text mode

"b": binary mode (images)

To open a file for reading it is enough to specify the name of the file:

f = open("demofile.txt")

The code above is the same as:

f = open("demofile.txt", "rt")

Because "r" for read, and "t" for text are the default values, you do not need to specify them.

Make sure the file exists otherwise you will get an error.

Python file open:

To open the file, use the built-in open() function.

The open() function returns a file object, which has a read() method for reading the content of the file:

```
f= open("demofile.txt", "r")
print(f.read())
```

If the file is located in a different location, you will have to specify the file path, like this:

Example

```
f= open("D:\\myfiles\welcome.txt", "r")
print(f.read())
```

Read Only Parts of the File

By default the read() method returns the whole text, but you can also specify how many characters you want to return:

```
f= open("demofile.txt", "r")
print(f.read(5))//returns first five characters of the memory.
```

Read lines:

You can return one line by using the readline() method:

```
f= open("demofile.txt", "r")
print(f.readline())
```

By calling readline() two times, you can read the two first lines:

```
f= open("demofile.txt", "r")
print(f.readline())
print(f.readline())
```

By looping through the lines of the file, you can read the whole file, line by line:

```
f= open("demofile.txt", "r")
for x in f:
  print(x)
```

Close Files

It is a good practice to always close the file when you are done with it.

```
f= open("demofile.txt", "r")
print(f.readline())
f.close()
```

Write to an Existing File

To write to an existing file, you must add a parameter to the open() function:

"a" - Append - will append to the end of the file

"w" - Write - will overwrite any existing content

```
f= open("demofile2.txt", "a")
f.write("Nowthefilehasmorecontent!")
f.close()
#openandreadthefileaftertheappending:
f= open("demofile2.txt", "r")
print(f.read())
```

overwrite the content:

```
f= open("demofile3.txt", "w")
f.write("Woops!Ihavedeletedthecontent!")
f.close()
#openandreadthefileaftertheoverwriting:
f= open("demofile3.txt", "r")
print(f.read())
```

the "w" method will overwrite the entire file

Create a New File

To create a new file in Python, use the open() method, with one of the following parameters:

"x" - Create - will create a file, returns an error if the file exist

"a" - Append - will create a file if the specified file does not exist

"w" - Write - will create a file if the specified file does not exist

```
f = open("myfile.txt", "x")
```

Result: a new empty file is created!

```
f = open("myfile.txt", "w")
```

Delete a File

To delete a file, you must import the OS module, and run its os.remove() function:

```
import os
os.remove("demofile.txt")
```

Check if File exist:

To avoid getting an error, you might want to check if the file exists before you try to delete it:

```
import os
if os.path.exists("demofile.txt"):
 os.remove("demofile.txt")
else:
  print("The file does not exist")
```

Delete Folder

To delete an entire folder, use the os.rmdir() method:

```
import os
os.rmdir("myfolder")
```

you can only remove empty folders

Some Important Points:

Multiple Statements on a Single Line:

The semicolon (;) allows multiple statements on the single line given that neither statement starts a new code block.

```
import sys; a = 'abc'; sys.stdout.write(a + '\n')
```

```
i = 12
print i
```

is effectively

same as

```
i = 12;
print i;
```

which is also same as i = 12; print i;

and same as i = 12; print i

If we divide an integer by integer then result will also be an integer. Any one of that must be float then only the answer in float form i.e. points form.

Eval:

```
num = eval(input('Enter a number: '))
print('Your number squared:', num*num)
```

The eval function converts the text entered by the user into a number. One nice feature of this is you can enter expressions, like 3*12+5, and eval will compute them for you.

End:

In python print statement automatically enters into new line if we don't want new line then use end=' '

```
print("on the first line",end='')
print("second line")#on the first line second line
```

Write a program that generates a list L of 50 random numbers between 1 and 100 Count how many numbers are greater than 50

```
count = 0
 for item in L:
if item>50:
count=count+1
```

Given a list L that contains numbers between 1 and 100, create a new list whose first element is how many ones are in L, whose second element is how many twos are in L, etc.

```
frequencies = []
for i in range(1,101):
frequences.append(L.count(i))
```

Write a program that prints out the two largest and two smallest elements of a list called scores.

```
scores.sort() s
print('Two smallest: ', scores[0], scores[1])
print('Two largest: ', scores[-1], scores[-2])
num_right = 0
# Question 1
print('What is the capital of France?', end=' ')
guess = input()
if guess.lower()=='paris':
    print('Correct!')
    num_right+=1
else:
print('Wrong. The answer is Paris.')
print('You have', num_right, 'out of 1 right')
 #Question 2
print('Which state has only one neighbor?', end=' ')
guess = input()
if guess.lower()=='maine':
print('Correct!') num_right+=1
else:
print('Wrong. The answer is Maine.')
print('You have', num_right, 'out of 2 right,')
or
questions = ['What is the capital of France?', 'Which state has only one neighbor?']
answers = ['Paris','Maine']
num_right = 0
 for i in range(len(questions)):
        guess = input(questions[i])
```

```
        if guess.lower()==answers[i].lower():
            print('Correct')
            num_right=num_right+1
        else:
            print('Wrong. The answer is', answers[i])
        print('You have', num_right, 'out of', i+1, 'right.')
```

list and random module:

choice(L) picks a random item from L

sample(L,n) picks a group of n random items from L

shuffle(L) Shuffles the items of L

choice:from random import choice

```
names = ['Joe', 'Bob', 'Sue', 'Sally']
current_player = choice(names)
from random import sample
 names = ['Joe', 'Bob', 'Sue', 'Sally']
team = sample(names, 2)
from random import choice
s='abcdefghijklmnopqrstuvwxyz1234567890!@#$%^&*()'
for i in range(10000):
   print(choice(s), end='')
from random import shuffle
players = ['Joe', 'Bob', 'Sue', 'Sally']
shuffle(players)
for p in players:
   print(p, 'it is your turn.') # code to play the game goes here...
shuffle(names)
teams = [] f
```

```
or i in range(0,len(names),2):
teams.append([names[i], names[i+1]])
split()
s = 'Hi! This is a test.' print(s.split())
from string import punctuation
for c in punctuation:
    s = s.replace(c, '')
```

how many times certain word occurs

```
from string import punctuation
s = input('Enter a string: ')
for c in punctuation:
      s = s.replace(c, '')
s = s.lower()
L = s.split()
word = input('Enter a word: ')
print(word, 'appears', L.count(word), 'times.')
s = '1-800-271-8281'
print(s.split('-'))
```

join

The join method is in some sense the opposite of split. It is a string method that takes a list of strings and joins them together into a single string. Here are some examples, using the list

```
L = ['A','B','C']
```

Operation	Result
' '.join(L)	A B C
''.join(L)	ABC
', '.join(L)	A, B, C

'***'.join(L) A***B***C

Write a program that creates an anagram of a given word. An anagram of a word uses the same letters as the word but in a different order. For instance, two anagrams of the word there are three and ether. Don't worry about whether the anagram is a real word or not. MORE WITH LISTS This sounds like something we could use shuffle for, but shuffle only works with lists. What we need to do is convert our string into a list, use shuffle on it, and then convert the list back into a string. To turn a string s into a list, we can use list(s). (See Section 10.1.) To turn the list back into a string, we will use join.

```
from random import shuffle
word = input('Enter a word: ')
letter_list = list(word)
shuffle(letter_list)
anagram = ''.join(letter_list)
print(anagram)
```

Here is an example that finds all the palindromic numbers between 1 and 10000. A palindromic number is one that is the same backwards as forwards, like 1221 or 64546.

```
for i in range(1,10001):
    s = str(i)
   if s==s[::-1]:
      print(s)
```

Here is an example that tells a person born on January 1, 1991 how old they are in 2010.

```
birthday = 'January 1, 1991'
year = int(birthday[-4:])
print('You are', 2010-year, 'years old.')
```

The year is in the last four characters of birthday. We use int to convert those characters into an integer so we can do math with the year.

Write a program that takes a number num and adds its digits. For instance, given the number 47, the program should return 11 (which is 4 + 7). Let us start with a 2-digit example.

```
digit = str(num)
answer = int(digit[0]) + int(digit[1])
```

The idea here is that we convert num to a string so that we can use indexing to get the two digits separately. We then convert each back to an integer using the int function.

Here is a version that handles numbers with arbitrarily many digits:

```
digit = str(num)
answer = 0
for i in range(len(digit)):
    answer = answer + int(digit[i])
```

To break a decimal number, num, up into its integer and fractional parts, we can do the following:

```
ipart = int(num)
dpart = num - int(num)
```

String formatting

Suppose we are writing a program that calculates a 25% tip on a bill of $23.60. When we multiply, we get 5.9, but we would like to display the result as $5.90, not $5.9. Here is how to do it:

```
 a = 23.60 * .25
print('The tip is {:.2f}'.format(a))
```

This uses the format method of strings. Here is another example:

```
bill = 23.60
tip = 23.60*.25
```

print('Tip: ${:.2f}, Total: ${:.2f}'.format(tip, bill+tip))

The way the format method works is we put a pair of curly braces {} anywhere that we want a formatted value. The arguments to the format function are the values we want formatted, with the first argument matching up with the first set of braces, the second argument with the second set of braces, etc. Inside each set of curly braces you can specify a formatting code to determine how the corresponding argument will be formatted.

Formatting integers To format integers, the formatting code is {:d}. Putting a number in front of the d allows us to right-justify integers. Here is an example:

```
print('{:3d}'.format(2))
print('{:3d}'.format(25))
print('{:3d}'.format(138))
```

```
  2
 25
138
```

The number 3 in these examples says that the value is allotted three spots. The value is placed as far right in those three spots as possible and the rest of the slots will be filled by spaces. This sort of thing is useful for nicely formatting tables.

To center integers instead of right-justifying, use the ^ character, and to left-justify, use the < character. print('{:^5d}'.format(2))

```
print('{:^5d}'.format(222))
 print('{:^5d}'.format(13834))
```

```
  2
 122
13834
```

Each of these allots five spaces for the integer and centers it within those five spaces.

Putting a comma into the formatting code will format the integer with commas. The example below prints 1,000,000:

print('{:,d}'.format(1000000))

NESTED LOOPS

Formatting floats To format a floating point number, the formatting code is {:f}. To only display the number to two decimal places, use {:.2f}. The 2 can be changed to change the number of decimal places. You can right-justify floats. For example, {:8.2f} will allot eight spots for its value—one of those is for the decimal point and two are for the part of the value after the decimal point. If the value is 6.42, then only four spots are needed and the remaining spots are filled by spaces, causing the value to be right-justified. The ^ and < characters center and left-justify floats.

Formatting strings To format strings, the formatting code is {:s}. Here is an example that centers some text:

print('{:^10s}'.format('Hi'))

print('{:^10s}'.format('there!'))

Hi

there!

To right-justify a string, use the > character: print('{:>6s}'.format('Hi'))

print('{:>6s}'.format('There')) Hi there! There is a whole lot more that can be done with formatting. See the Python documentation [1].

How are arguments passed in python by call by reference or call by value?

Every argument in a python method is an object. All the variables in python have reference to an object. Therefore, arguments in python method are passed by reference. Since some objects passed as reference are mutable, we can change those objects in a method. But for an immutable object like string any changes made that are not reflected outside.

Generator:

Lambda expression:

A lambda expression in python is used for creating an anonymous function wherever we need a function, we can also use lambda expression

That can be defined in a single block

Variable_name=lambda arguments:expression

Any no of arguments passed to the lambda function

2

We have to use lambda keyword for creating a lambda expression

Syntax:

Lambda argumentList:expression

Lambda a,b:a+b

Iterators:

Zip() function:

Python zip() function returns a zip object, which maps a similar index of multiple containers. It takes an iterable, converts it into an iterator and aggregates the elements based on iterasbles passed. It returns an iterator of tuples.

Pip()

PIP is an acronym for Python Installer Package which provides a seamless interface to install various Python modules. It is a command-line tool that can search for packages over the internet and install them without any user interaction.

Self()

Enum:

Enums are readable names that are bound to a constant value. To use enums, import Enum from the enum standard library module:

Annotations:

operator overloading

multithreading: in any programming language every code runs on a main thread

Bootcamp:

Logical Operator used in expression and, or, and not Bitwise operators for bit operations &, |,...Special Operators for memory comparison like is & is not

Help if a Python Built-In function to essentially get more helpful information of python built-in function. The execution of help(print) results in print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Convert Inch to CM using 1 inch = 2.54 centimeters

Type conversion is converting from one data type to another. Explicit type conversion is achieved through Python's built-in functions such as int(), float(), and str()

Constants are types of variables whose values cannot be altered or changed after initialization. These values are universally proven to be true and they cannot be changed over time. e.g. PI = 22/7

- The magic of f-strings is that you can embed Python expressions directly inside them by using curly braces {}.
- The expression is evaluated and converted to string representation, and the result is interpolated into the original string in that location

- Naming Convention to define Constant is all caps
- **Global or Local Variables -** Should be in lower case and use an underscore to join words.
- **Constants in Python -** Should be only in upper case and use an underscore to join words.
- **Function Names -** Should be in lower case and use an underscore to join words.
- **Module Names -** Should be in lower case and use an underscore to join words.
- **Package Names -** Should be in lower case and use an underscore to join words.

- **Class Names -** Should follow Cap Words convention.
- **Method Names -** Should be in lower case and use an underscore to join words
- Python constants are declared and initialized in the beginning of the program as global or preferably in different modules/files.

Python Operator Precedence - In Python, the order of operations follows the same principles as PEMDAS.

- PEMDAS is an acronym that stands for "Parentheses, Exponents, Multiplication and Division, and Addition and Subtraction"
- That is, expressions inside parentheses are evaluated first, followed by exponents, then multiplication and division (from left to right), and finally addition and subtraction (from left to right).
- : separates the number from format specifier 02d.
- 02 specifies to put leading zeros for 2-digit number if the number is a single digit number
- d indicates that the value should be formatted as a integer
- Tags – {'Item Code':^10} - Center align 'Item Code' with length 10\
- ^ means center align the text.
- {'-' * 74} - repeat '-' 74 times
- {' ' * 54} - repeat space ' ' 54 times
- round(10.7) - rounds float to nearest integer 11
- round(2.665, 2) - rounds float 2.665 to 2 decimal places, 2.67

Python Functions are block of code that is run only when called. Guidelines for Python functions

- 1: Proper Comments for the function
- 2: Use of Global Variable should be avoided

- 3: Use of function type annotation, by Default is any

```python
# Step 1: Setting Constants for Default Character, Vowel String,
# Vowel as List and Vowels as Dictionary
DEFAUTL_CHAR = 'A'
VOWEL = 'AEIOU'

# Creating VOWELS Object from Sequence of Vowel Characters
# VOWELS = ['A', 'E', 'I', 'O', 'U']
VOWELS = list(VOWEL)

# Creating VOWEL_DICT and setting all the keys Vowel characters and value as vowel
# VOWEL_DICT = {'A': 'Vowel', 'E': 'Vowel', 'I': 'Vowel', 'O': 'Vowel', 'U': 'Vowel'
VOWEL_DICT = dict.fromkeys(VOWELS,"Vowel")

# Printing Vowel list and Dict
print(f"VOWEL List Type: {type(VOWELS)}",
      f"VOWEL Dict Type: {type(VOWEL_DICT)}", sep = '\n')
print(f"VOWEL List: {VOWELS}", f"VOWEL Dict: {VOWEL_DICT}", sep = '\n')

# Step 2: Prompt user for input
user_unput = input("Enter a single character ( A-Z ): ")

# Step 3: Checking the Length of user input, either accessing the DEFAUTL_CHAR
# or the first character of the user input using string indexing and
# converting the same to upper case
char = len(user_unput) == 0 and DEFAUTL_CHAR or user_unput[0].upper()

# Step 4: Checking if the character is alpha, if not then assigning DEFAUTL_CHAR
char = char.isalpha() and char or DEFAUTL_CHAR

# Step 5: Determining Vowel or Consonent using multiple optional ways

# Option 5: Using Membership Operator to check if character is in VOWELS List
# defined as global
print(f"\nOPTION 5: The character '{char}' is a",
      f"{char in VOWELS and 'Vowel' or 'Consonant'}.")

# Option 6: Using Vowel Dict get() method to get Character if Vowel, else
# default to Consonent
print(f"\nOPTION 6: The character '{char}' is a",
      f"{VOWEL_DICT.get(char, 'Consonent')}.")
```

```
# Step 1: Setting constants for default character and vowel
DEFAUTL_CHAR = 'A'
VOWEL = 'AEIOU'

# Step 2: Prompt user for input
user_unput = input("Enter a single character ( A-Z ): ")

# Step 3: Checking the Length of user input, either accessing the DEFAUTL_CHAR
# or the first character of the user input using string indexing and
# converting the same to upper case
char = len(user_unput) == 0 and DEFAUTL_CHAR or user_unput[0].upper()

# Step 4: Checking if the character is alpha, if not then assigning DEFAUTL_CHAR
char = char.isalpha() and char or DEFAUTL_CHAR

# Step 5: Determining Vowel or Consonent using multiple optional ways

# Option 3: Using String class built-in find() method to check if character
# is present in the global constant VOWEL
print(f"\nOPTION 3: The character '{char}' is a",
      f"{VOWEL.find(char) == 0 and 'Vowel' or 'Consonant'}.")

# Option 4: Using Membership Operator to check if character is in VOWEL String
# defined as global
print(f"\nOPTION 4: The character '{char}' is a",
      f"{char in VOWEL and 'Vowel' or 'Consonant'}.")
```

```
Enter a single character ( A-Z ): TEST

OPTION 3: The character 'T' is a Consonant.

OPTION 4: The character 'T' is a Consonant.
```

```python
# PROG 2.2: Temperature Conversion using match block

# Step 1: Prompt user for input temperature
temperature_entered = input("Enter the temperature: ")

# Step 2: Check if Temperature is in digit, then take up conversion
if temperature_entered.isdigit() :
  temperature = float(temperature_entered);
  # Step 3: Take Input for Unit and Check to perform conversion accordingly
  unit = input("Enter the unit of temperature (C for Celsius, F for Fahrenheit): ")
  match unit:
    case 'C' | 'c':
      celcius = temperature
      fahrenheit = (celcius * 9/5) + 32
      print(f"{celcius}°C is equal to {fahrenheit:.1f}°F")
    case 'F' | 'f':
      fahrenheit = temperature
      celcius = (fahrenheit - 32) * 5/9
      print(f"{fahrenheit}°F is equal to {celcius:.1f}°C")
    case _:
      print(f"Invalid unit {unit}. Please enter 'C' for Celsius or 'F' for Fahrenheit.")
else :
  print(f"Invalid Temperature {temperature_entered}. Please enter digits.")
```

```
Enter the temperature: 0
Enter the unit of temperature (C for Celsius, F for Fahrenheit): C
0.0°C is equal to 32.0°F
```

PROG 2.1: Temperature Conversion using if block

```python
# PROG 2.1: Temperature Conversion using if block

# Step 1: Prompt user for input temperature
temperature_entered = input("Enter the temperature: ")

# Step 2: Check if Temperature is in digit, then take up conversion
if temperature_entered.isdigit() :
  temperature = float(temperature_entered);
  # Step 3: Take Input for Unit and Check to perform conversion accordingly
  unit = input("Enter the unit of temperature (C for Celsius, F for Fahrenheit): ")
  if unit == 'C' or unit == 'c':
      converted_temp = (temperature * 9/5) + 32
      print(f"{temperature}°C is equal to {converted_temp:.1f}°F")
  elif unit == 'F' or unit == 'f':
      converted_temp = (temperature - 32) * 5/9
      print(f"{temperature}°F is equal to {converted_temp:.1f}°C")
  else:
      print(f"Invalid unit {unit}. Please enter 'C' for Celsius or 'F' for Fahrenheit.")
else :
  print(f"Invalid Temperature {temperature_entered}. Please enter digits.")
```

```
Enter the temperature: 0
Enter the unit of temperature (C for Celsius, F for Fahrenheit): Test
Invalid unit Test. Please enter 'C' for Celsius or 'F' for Fahrenheit.
```

```python
# Step 1: Prompt user to enter a year and assign to global variable year_input
year_input = input("Enter a year: ")

# Step 2: Check if input is a digit
if year_input.isdigit() :
    # Step 3: Convert input to integer and assign it to the local variable yea
    year = int(year_input)

    # Step 4: Check if the year is at least 1582 which is the start of
    # the Gregorian calendar
    if year >= 1582 :

      # Step 5: Divided by 100 means century year (ending with 00)
      # and century year divided by 400 is leap year
      if (year % 100 == 0) and (year % 400 == 0) :
          print(f"The year {year} is a leap year")

      # Step 6: Check if the year is divisible by 4 and not by 100
      elif (year % 4 == 0) and (year % 100 != 0) :
          print(f"The year {year} is a leap year")
      else:
          print(f"The year {year} is not a leap year")
    else:
        print(f"The entered year {year} must be 1582 or later.")
else:
    print(f"Invalid year {year_input} enetered. Please enter a valid year.")
```

```
Enter a year: 1200
The entered year 1200 must be 1582 or later.
```

```python
# Step 1: Prompt user to enter a year and assign to global variable year_input
year_input = input("Enter a year: ")

# Step 2: Check if input is a digit
if year_input.isdigit() :
    # Step 3: Convert input to integer and assign it to the local variable yea
    year = int(year_input)

    # Step 4: Check if the year is at least 1582 which is the start of
    # the Gregorian calendar
    if year >= 1582 :

      # Step 5: Divided by 100 means century year (ending with 00)
      # and century year divided by 400 is leap year
      if (year % 100 == 0) and (year % 400 == 0) :
          print(f"The year {year} is a leap year")

      # Step 6: Check if the year is divisible by 4 and not by 100
      elif (year % 4 == 0) and (year % 100 != 0) :
          print(f"The year {year} is a leap year")
      else:
          print(f"The year {year} is not a leap year")
    else:
        print(f"The entered year {year} must be 1582 or later.")
else:
    print(f"Invalid year {year_input} enetered. Please enter a valid year.")
```

```
Enter a year: 1200
The entered year 1200 must be 1582 or later.
```

Open with Google Docs

```
# PROG 1.2: Leap Year Function

def check_leap_year(year_input) :
  # Step 1: Check if input is a digit
  if year_input.isdigit() == False :
    return f"Invalid year {year_input} enetered. Please enter a valid year."

  # Step 2: Convert input to integer and assign it to local variable year
  year = int(year_input)

  # Step 3: Check if the year is at least 1582 which is the start of
  # the Gregorian calendar
  if year < 1582 :
    return f"The entered year {year} must be 1582 or later."

  # Step 4: Divided by 100 means century year (ending with 00)
  # and century year divided by 400 is leap year
  if (year % 100 == 0) and (year % 400 == 0) :
      return f"The year {year} is a leap year"

  # Step 5: Check if the year is divisible by 4 and not by 100
  if (year % 4 == 0) and (year % 100 != 0) :
      return f"The year {year} is a leap year"

  # Step 6: Not a Leap Year
  return f"The year {year} is not a leap year"

print(check_leap_year(input("Enter a year: ")))
```

```
Enter a year: 2024
The year 2024 is a leap year
```

```
birth_year = int(input('Enter Birth Year: '))

# Step 2: Print the Generation using Python Shorthand Condition
print(f'\nBirth Year is {birth_year}',
      f'Is Baby Boomer: {birth_year > 1946 or birth_year == 1946 and birth_year < 1964}',
      f'Is Gen X: {birth_year >= 1965 and birth_year < 1980}',
      f'Is Millennial: {birth_year >= 1981 and birth_year < 1996}',
      f'Is Gen Z: {birth_year >= 1997 and birth_year < 2012}',
      f'Is Gen Alpha: {birth_year >= 2013 and birth_year < 2025}', sep = '\n')
```

```
Enter Birth Year: 1946

Birth Year is 1946
Is Baby Boomer: True
Is Gen X: False
Is Millennial: False
Is Gen Z: False
Is Gen Alpha: False
```

```python
# PROG 5: Time Conversion

# Step 1: Take input for time duration in total_seconds
total_seconds = int(input("Enter the time duration in seconds: "))

# Step 2: Convert seconds to hours, minutes, and seconds using
# floor division and modulo operator
hours = total_seconds // 3600
minutes = (total_seconds % 3600) // 60
seconds = total_seconds % 60

# Print the converted time duration
print(f"\nTime duration of {total_seconds} seconds in HH:MM:SS format is",
      f"{hours:02d}:{minutes:02d}:{seconds:02d}")
```

```
Enter the time duration in seconds: 3800

Time duration of 3800 seconds in HH:MM:SS format is 01:03:20
```

PYTHON