```
#Find list of Keywords and count

import keyword
print(keyword.kwlist)
print("Total Keywords are:",len(keyword.kwlist))
```

```
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except',
Total Keywords are: 35
```

Variable:

Containers for storing data values. Variables do not need to be declared with any particular type, and can even change type after they have been set.

- Case Sensitive
- Can't start with the number
- No Special character (Except _)
- No Keywords can be used
- Identifier has no length limit

```
a=10
a
```

```
10
```

```
1a=10
print(1a)
```

```
  File "<ipython-input-5-4731af0d5bab>", line 1
    1a=10
    ^
SyntaxError: invalid decimal literal
```

```
@=1
print(@)
```

```
  File "<ipython-input-6-c1180040189a>", line 1
    @=1
    ^
SyntaxError: invalid syntax
```

```
# Keyords cant be used as variables

if=a
print(if)
```

```
  File "<ipython-input-7-ed13bb588fa1>", line 3
    if=a
      ^
SyntaxError: invalid syntax
```

```
#variables can be of any size

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa=10
print(aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa)
```

```
10
```

```
# True==1 & False==0
print(True+True)
print(True+False)
print(False+True)
print(False+False)
```

```
2
1
1
0
```

```
# Camel Case
myVariableName='Hello'

#Snake Case
my_Variable_Name='Hello World'

#Pascal Case
myVariableName='Hello World'


#Many Values to Multiple Variables

a,b,c='Apple','Banana','Custard'
print(a)
print(b)
print(c)
```

```
Apple
Banana
Custard
```

```
#One Value multi variables

a=b=c='Variable'

print(a)
print(b)
print(c)
```

```
Variable
Variable
Variable
```

```
#Unpack Collection

Variables=['Apple','Banana','Custard']
a,b,c=Variables

print(a)
print(b)
print(c)
```

```
Apple
Banana
Custard
```

```
Variables=['Apple','Banana','Custard','Dates','Elephant']
a,b,c=Variables
```

```
print(a)
print(b)
print(c)

# Error as variables doesnot match the count the length of list
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-16-ceec1a981c0e> in <cell line: 2>()
      1 Variables=['Apple','Banana','Custard','Dates','Elephant']
----> 2 a,b,c=Variables
      3
      4 print(a)
      5 print(b)

ValueError: too many values to unpack (expected 3)
```

```
Variables=['Apple','Banana','Custard','Dates','Elephant']
a,*b,c=Variables

print(a)
print(b)
print(c)

# Adding * to variable assigns the values to that variable fullfilling the count of variable
```

```
Apple
['Banana', 'Custard', 'Dates']
Elephant
```

```
a = "Python"
b = "is"
c = "awesome"

print(a,b,c) # , gives space
print(a+b+c) # + works as mathematic operator and gives no gaps
```

```
Python is awesome
Pythonisawesome
```

```
# Global Variables

x='Computer'

def myfunc():
  x='System'
  print('Inside Function',x)


print('Outside Function',x)
myfunc()
```

```
Outside Function Computer
Inside Function System
```

```
x='Beautiful'

def myfunc():
  x='Handsome'
  print('He is',x)

myfunc()
print('She is',x)
```

```
He is Handsome
She is Beautiful
```

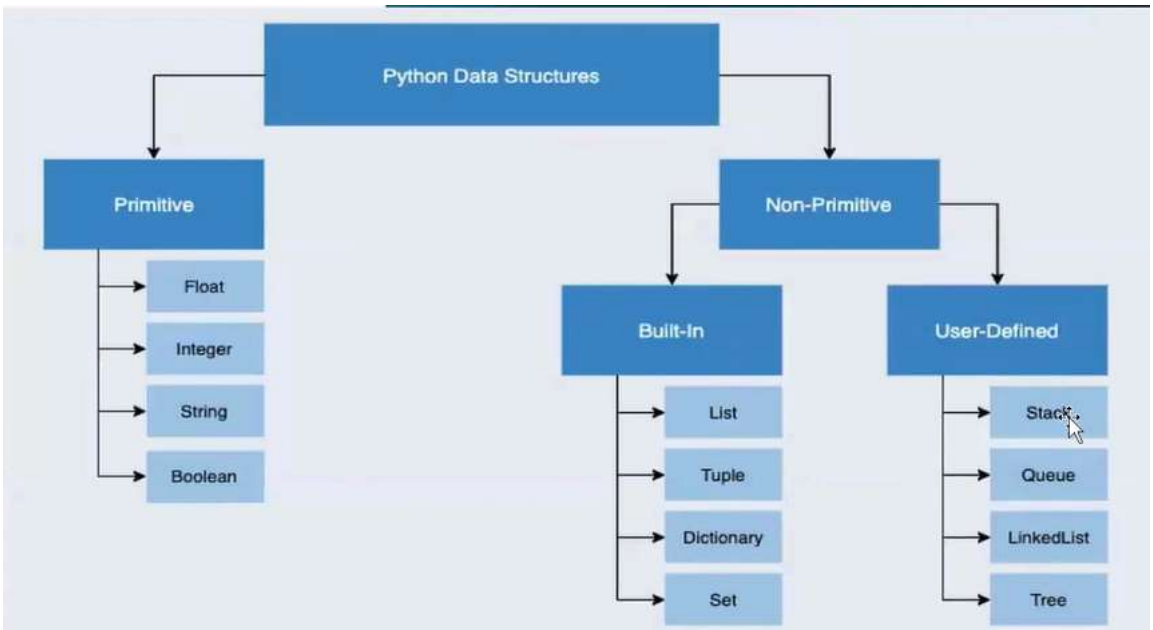
```
# Using global keyword belongs to global scope

def myfunc():
  global x
  x='Hello'
  print(x,'World')
```

```
myfunc()
print(x,'Handsome')
```

```
Hello World
Hello Handsome
```

Data Types:




```
# Primitive

x='Hello World'
print(x)
print(type(x))

x=10
print(x)
print(type(x))

x=10.5
print(x)
print(type(x))

x=2+3j
print(x)
print(type(x))

x=False
```

```
print(x)
print(type(x))
```

```
Hello World
<class 'str'>
10
<class 'int'>
10.5
<class 'float'>
(2+3j)
<class 'complex'>
False
<class 'bool'>
```

```
# Non Primitive- Built in

x=['A','B',1,2]
print(x)
print(type(x))

x=('A','B',1,2)
print(x)
print(type(x))

x={'A','B',1,2}
print(x)
print(type(x))

x={'A':1,'B':2}
print(x)
print(type(x))
```

```
['A', 'B', 1, 2]
<class 'list'>
('A', 'B', 1, 2)
<class 'tuple'>
{'A', 2, 'B', 1}
<class 'set'>
{'A': 1, 'B': 2}
<class 'dict'>
```

Int :

- Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.

```
int_1=1
int_2=-345
int_3=12345789

print(type(int_1))
print(type(int_2))
print(type(int_3))
```

```
<class 'int'>
<class 'int'>
<class 'int'>
```

Float

- Float, or "floating point number" is a number, positive or negative, containing one or more decimals.
- It can also be scientific numbers with an "e" to indicate the power of 10.

```
float_1=1.0
float_2=-6.9
float_3=9.710

print(type(float_1))
print(type(float_2))
print(type(float_3))
```

```
<class 'float'>
<class 'float'>
<class 'float'>
```

```
float_sci=2e2

print(float_sci)
print(type(float_sci))
```

```
200.0
<class 'float'>
```

```
import numpy as np
type(np.nan)
```

```
float
```

Complex

- numbers are written with a "j" as the imaginary part

```
a=20+30j
print(a.real)
print(a.imag)
print(a)
print(type(a))
```

```
20.0
30.0
(20+30j)
<class 'complex'>
```

```
x=1+2j
y=3+2j

print(x+y)
print(x+1) # 1 gets added to real value of x
print(x+2j) # 2j gets added to imaginary value of y
```

```
(4+4j)
(2+2j)
(1+4j)
```

Strings

- Immutable
- Can be declared by both single and double quotes
- Strings in Python are arrays.
- Square brackets can be used to access elements of the string.

```
S='Hello'
print(type(S))
```

```
<class 'str'>
```

```
# We can use quotes inside the string untill unless they match outside

str_1="It's a string"
print(str_1)
print(type(str_1))
```

```
It's a string
<class 'str'>
```

```
a='Hello'
b="Hello"
c=''' Hello
```

```
        This my World'''

print(a)
print(b)
print(c)
print(type(a))
print(type(b))
print(type(c))
```

```
Hello
Hello
 Hello
        This my World
<class 'str'>
<class 'str'>
<class 'str'>
```

```
# Strings are Arrays we can access the values by Slicing

s='Pavan'
print(s[0:2])
print(s[1:-1])
print(s[:])
print(s[0])
print(s[2])

#length of string
print(len(s))
```

```
Pa
ava
Pavan
P
v
5
```

```
# Looping through a String

str_1="Hello World"

for i in str_1:
  print(i)
```

```
H
e
l
l
o

W
o
r
l
d
```

```
# Modify Strings

String="Hello World"

print(String.upper())
print(String.lower())

String_1="     Remove Whitespaces     "
print(String_1.strip()) # Removes White spaces from begin or end

print(String.replace('Hello','Bye'))

a="Hello World"
print(a.split(" "))
```

```
HELLO WORLD
hello world
Remove Whitespaces
Bye World
['Hello', 'World']
```

```
# String Concatenation

a="Hello"
b="World"

print(a+b)
print(a+" "+b)
```

```
HelloWorld
Hello World
```

```
# Format string
rose="Rose"
lilly="Lilly"

print(f"{rose} is the beautiful flower")
print(f"{lilly} is the beautiful flower")
```

```
Rose is the beautiful flower
Lilly is the beautiful flower
```

```
# Check String
a=' The best city is Hyderabad'

print('Hyderabad' in a)
print('hyderabad' in a)
```

```
True
False
```

```
# Escape Character

'''
\'  Single Quote
\\  Backslash
\n  New Line
\r  Carriage Return
\t  Tab
\b  Backspace
\f  Form Feed
\ooo    Octal value
\xhh    Hex value
'''


S='That\'s Right'
print('Single Quote:',S)
print('\n')

S='That\\s Right'
print('Backslash:',S)
print('\n')

S='That\rs Right'
print('Carriage Return:',S)
print('\n')

S='That\ts Right'
print('Tab:',S)
print('\n')

S='That\bs Right'
print('Backspace:',S)
print('\n')

S='That\fs Right'
print('Form Feed:',S)
print('\n')
```

```
Single Quote: That's Right


Backslash: That\s Right
```

```
s Right

Tab: That	s Right

Backspace: Thas Right

Form Feed: Thats Right
```

String Methods

- capitalize(): Converts the first character to upper case
- casefold(): Converts string into lower case
- center(): Returns a centered string
- count(): Returns the number of times a specified value occurs in a string
- encode(): Returns an encoded version of the string
- endswith(): Returns true if the string ends with the specified value
- expandtabs(): Sets the tab size of the string
- find(): Searches the string for a specified value and returns the position of where it was found
- format(): Formats specified values in a string
- format_map(): Formats specified values in a string
- index(): Searches the string for a specified value and returns the position of where it was found
- isalnum(): Returns True if all characters in the string are alphanumeric
- isalpha(): Returns True if all characters in the string are in the alphabet
- isascii(): Returns True if all characters in the string are ascii characters
- isdecimal(): Returns True if all characters in the string are decimals
- isdigit(): Returns True if all characters in the string are digits
- isidentifier(): Returns True if the string is an identifier
- islower(): Returns True if all characters in the string are lower case
- isnumeric(): Returns True if all characters in the string are numeric
- isprintable(): Returns True if all characters in the string are printable
- isspace(): Returns True if all characters in the string are whitespaces
- istitle(): Returns True if the string follows the rules of a title
- isupper(): Returns True if all characters in the string are upper case
- join(): Joins the elements of an iterable to the end of the string
- ljust(): Returns a left justified version of the string
- lower(): Converts a string into lower case
- lstrip(): Returns a left trim version of the string
- maketrans(): Returns a translation table to be used in translations
- partition(): Returns a tuple where the string is parted into three parts
- replace(): Returns a string where a specified value is replaced with a specified value
- rfind(): Searches the string for a specified value and returns the last position of where it was found
- rindex(): Searches the string for a specified value and returns the last position of where it was found
- rjust(): Returns a right justified version of the string
- rpartition(): Returns a tuple where the string is parted into three parts
- rsplit(): Splits the string at the specified separator, and returns a list
- rstrip(): Returns a right trim version of the string
- split(): Splits the string at the specified separator, and returns a list
- splitlines(): Splits the string at line breaks and returns a list
- startswith(): Returns true if the string starts with the specified value
- strip(): Returns a trimmed version of the string
- swapcase(): Swaps cases, lower case becomes upper case and vice versa
- title(): Converts the first character of each word to upper case
- translate(): Returns a translated string
- upper(): Converts a string into upper case
- zfill(): Fills the string with a specified number of 0 values at the beginning

Boolean

- represent one of two values: True or False.
- True==1
- False==0

```
print(10 > 9)
print(10 == 9)
print(10 < 9)
```

```
True
False
False
```

```
#Almost any value is evaluated to True if it has some sort of content.
#Any string is True, except empty strings.
#Any number is True, except 0.
#Any list, tuple, set, and dictionary are True, except empty ones.

print(bool("Hello"))
print(bool(15))
print(bool(0))
print(bool())
print(bool(False))
print(bool(None))
print(bool(0))
print(bool(""))
print(bool(()))
print(bool([]))
print(bool({}))
```

```
True
True
False
False
False
False
False
False
False
False
False
False
```

Casting in python is therefore done using constructor functions:

```
print(int(12.0))  # 12
print(int("12"))   # 12
print(int(True))   # 1
print(int(2+3j)) #Error
```

```
12
12
1
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-75-d81f0ec13200> in <cell line: 4>()
      2 print(int("12"))   # 12
      3 print(int(True))   # 1
----> 4 print(int(2+3j)) #Error

TypeError: int() argument must be a string, a bytes-like object or a real number, not 'complex'
```

```
print(float(1)) # 1.0
print(float('12')) # 12.0
print(float(True)) #1.0
print(float(3+3j)) #Error
```

Show hidden output

```
print(str(1))  # '1'
print(str(1.0))  #'1.0'
print(str(True)) #'True
print(str(2+3j)) #'(2+3j)'
```

```
1
1.0
True
(2+3j)
```

```
print(bool(1)) #True
print(bool(1.0)) #True
print(bool("Hello")) #True
print(bool()) #False
print(bool(2+1j)) #True
```

```
True
True
True
False
True
```

```
print(complex(1)) #(1+0j)
print(complex(1.0)) #(1+0j)
print(complex(True)) #(1+0j)
print(complex(False)) #0j
print(complex('10')) #10+0j
print(complex('H')) # Error
```

```
(1+0j)
(1+0j)
(1+0j)
0j
(10+0j)
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-80-8257b7794da9> in <cell line: 6>()
      4 print(complex(False)) #0j
      5 print(complex('10')) #10+0j
----> 6 print(complex('H')) # Error

ValueError: complex() arg is a malformed string
```

```
print("Hello")
```

```
Hello
```

Operators

```
# Arithmetic Operators

print('Addition: 5+10=',5+10)
print('Subtraction: 10-5=',10-5)
```

```
print('Multiplication: 5*10=',5*10)
print('Division: 10/5=',10/5)
print('Modulus: 10%5=',10%5)
print('Exponentiation: 2**3=',2**3)
print('Floor Division: 13//6=',13//6)
```

```
Addition: 5+10= 15
Subtraction: 10-5= 5
Multiplication: 5*10 50
Division: 10/5= 2.0
Modulus: 10%5= 0
Exponentiation: 2**3= 8
Floor Division: 13//6= 2
```

Python Assignment Operators

Assignment operators are used to assign values to variables:

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3

&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3
:=	print(x := 3)	x = 3 print(x)

```
x=5
print(x)
x+=1
print(x)
x-=1
print(x)
x*=2
print(x)
x/=2
print(x)
x%=2
print(x)
x**=2
print(x)
x//=2
print(x)
```

```
5
6
5
10
5.0
1.0
1.0
0.0
```

```
x = 5
x&=3
print(x)
x!=3
print(x)
x^=3
print(x)
x>>=3
print(x)
x<<=3
print(x)

print(x:=3) #x=3
```

```
1
1
2
0
0
3
```

Python Comparison Operators

Comparison operators are used to compare two values:

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

```
x = 7
y = 5

print(x == y)
print(x != y)
print(x > y)
print(x < y)
print(x >= y)
print(x <= y)
```

```
False
True
True
False
True
False
```

Python Logical Operators

Logical operators are used to combine conditional statements:

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

```
x = 5

print(x > 3 and x < 10) # T and T = T
print(x > 3 or x < 10) # T or T = T
print(not(x > 3 and x < 10)) # not(T and T)=not(T)=F
```

```
True
True
False
```

Python Identity Operators

Identity operators are used to compare the objects, not if they are equal, they are actually the same object, with the same memory location:

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

```
x = ["apple", "banana"]
y = ["apple", "banana"]
z = x

print(x is z)
# returns True because z is the same object as x

print(x is y)
# returns False because x is not the same object as y, even if they have the same content

print(x == y)
# to demonstrate the difference betweeen "is" and "==": this comparison returns True because x is equal to y

print(x is not z)
# returns False because z is the same object as x

print(x is not y)
# returns True because x is not the same object as y, even if they have the same content

print(x != y)
# to demonstrate the difference betweeen "is not" and "!=": this comparison returns False because x is equal to y
```

```
True
False
True
False
True
False
```

Python Membership Operators

Membership operators are used to test if a sequence is presented in an object:

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

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

print("banana" in x)
# returns True because a sequence with the value "banana" is in the list

print("banana" not in x)
# returns True because a sequence with the value "banana" is in the list
```

```
True
False
```

Python Bitwise Operators

Bitwise operators are used to compare (binary) numbers:

Operator	Name	Description	Example
&	AND	Sets each bit to 1 if both bits are 1	x & y
|	OR	Sets each bit to 1 if one of two bits is 1	x | y
^	XOR	Sets each bit to 1 if only one of two bits is 1	x ^ y
~	NOT	Inverts all the bits	~x
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off	x << 2
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off	x >> 2

```
print(5 & 6)
print(5 | 6)
```

Summary:

Feature	List	Tuple	Set	Dictionary
Mutable	Yes	No	Yes	Yes
Ordered	Yes	Yes	No	Yes
Duplicates	Allowed	Allowed	Not Allowed	Not in Keys
Syntax	`[1, 2, 3]`	`(1, 2, 3)`	`{1, 2, 3}`	`{'a': 1}`

List

- like the arrays.
- Ordered
- Mutable
- Allows Duplicates

1. Slicing
2. Access
3. Insert
4. Update
5. append
6. extend
7. remove
8. pop
9. del
10. clear
11. List Comprehension
12. Short Hand
13. reverse
14. sort
15. enumerate
16. All/Any
17. Copy

```
List_1=[1,2,3,4,5]
List_2=['A','B','C','D']
List_3=[1.0,2.0,3.0]
List_4=[1+1j,2+2j,3+3j]
```

```
List_5=[1,'B',3.0,4+4j]

print(type(List_1))
print(type(List_2))
print(type(List_3))
print(type(List_4))
print(type(List_5))
```

```
<class 'list'>
<class 'list'>
<class 'list'>
<class 'list'>
<class 'list'>
```

```
# define list by list constructor and find length
#Note: double round-brackets

A=list((1,2,3,4,5,6,7,8))
print(type(A))
print(A)
print(len(A))
```

```
<class 'list'>
[1, 2, 3, 4, 5, 6, 7, 8]
8
```

```
# Access values by Indexing
A=list((1,2,3,4,5,6,7,8))

print(A)

print(A[0])
print(A[-1])
print(A[:])
print(A[:5])
print(A[5:])
print(A[2:5])
print(A[-5:-1])
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
1
8
[1, 2, 3, 4, 5, 6, 7, 8]
[1, 2, 3, 4, 5]
[6, 7, 8]
[3, 4, 5]
[4, 5, 6, 7]
```

```
# Change Item Values by indexing
A=list((1,2,3,4,5,6,7,8))
print(A)

A[0]='A' #Change Item Value
print(A)

A[1:8]=['B','C','D','E','F','G','H'] # Change a range
print(A)
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
['A', 2, 3, 4, 5, 6, 7, 8]
['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']
```

```
#Insert items at desired locations
print(A)

A.insert(1,'BB') # Here 1 is the position where 'BB' needed to be addes
print(A)
```

```
['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']
['A', 'BB', 'B', 'C', 'D', 'E', 'F', 'G', 'H']
```

```
# Append
#We can append only one value at a time and it gets added in the end
A=['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']
print(A)
```

```
A.append('I')
print(A)
```

```
['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']
['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
```

```
#Extend
#To append elements from another list to the current list, use the extend() method.
# Using extend we can add tuple/sets/Dict
List_1=[1,2,3,4]
List_2=['A','B','C']
print(List_1)
print(List_2)

List_1.extend(List_2)
print(List_1)
```

```
[1, 2, 3, 4]
['A', 'B', 'C']
[1, 2, 3, 4, 'A', 'B', 'C']
```

```
# Using extend() we can add any data structure(tuple, sets, dict)
List_1=[1,2,3,4]
List_2=('A','B','C')

List_1.extend(List_2)
print(List_1)
```

```
[1, 2, 3, 4, 'A', 'B', 'C']
```

```
#Remove
#Removes mentioned value
List_1=[1, 2, 3, 4, 'A', 'B', 'C']
print(List_1)
List_1.remove('A')
print(List_1)
```

```
[1, 2, 3, 4, 'A', 'B', 'C']
[1, 2, 3, 4, 'B', 'C']
```

```
#Pop
#removes specified index value
List_2=['A','B','C']
print(List_2)

List_2.pop()  # removes last element
print(List_2)

List_2.pop(0) # removes index 0 element
print(List_2)
```

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

```
#del
#removes the specified index
List_2=['A','B','C']
print(List_2)

del List_2[1]
print(List_2)

del List_2  # deleted entire list
```

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

```
#Clear - empties the list
List_3=[1,2,3]
print(List_3)
```

```
List_3.clear()

print(List_3)
```

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

```
# For Loop Lists

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

print('for loop')
for i in thislist:
  print(i)
print('\n')

# While Loop lists
print('while loop')
i=0
while(i<len(thislist)):
  print(thislist[i])
  i+=1
print('\n')

print('Short hand for loop')

[print(i) for i in thislist]
```

```
for loop
apple
banana
cherry


while loop
apple
banana
cherry


Short hand for loop
apple
banana
cherry
[None, None, None]
```

```
# List Comprehension

# List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

A=[1,2,3,4,5,6,7,8]
B=[]

for i in A:
  B.append(i)

B
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
```

```
# Syntax
# newlist = [expression for item in iterable if condition == True]

A=[1,2,3,4,5,6,7,8]
B=[i for i in A if i==1]
print(B)
C=[i for i in A if i%2==0]
print(C)
```

```
[1]
[2, 4, 6, 8]
```



```
D=[i for i in A if i>4]
D
```

```
[5, 6, 7, 8]
```

```
#reverse
thislist=[1,2,3,4,5]
print(thislist)
thislist.reverse()
print(thislist)
```

```
[1, 2, 3, 4, 5]
[5, 4, 3, 2, 1]
```

```
thislist=[1,2,3,4,5]
print(thislist)
print(thislist[::-1])
```

```
[1, 2, 3, 4, 5]
[5, 4, 3, 2, 1]
```

```
#sort

List_sort=[1,6,4,8,3,5,8,0,4]
List_sort.sort()
print(List_sort)
List_sort.sort(reverse=True)
print(List_sort)
```

```
[0, 1, 3, 4, 4, 5, 6, 8, 8]
[8, 8, 6, 5, 4, 4, 3, 1, 0]
```

```
#Family Membership
List_sort=[1,6,4,8,3,5,8,0,4]
print(List_sort)

print(5 in List_sort)
print(2 in List_sort)
```

```
[1, 6, 4, 8, 3, 5, 8, 0, 4]
True
False
```

```
#enumerate
#prints with index number

for i in enumerate(List_sort):
  print(i)
```

```
(0, 1)
(1, 6)
(2, 4)
(3, 8)
(4, 3)
(5, 5)
(6, 8)
(7, 0)
(8, 4)
```

```
# All/Any

# The all() method returns,
# True: If all elements in a list are true
# False: If any element in a list is false

# The any() method returns,
# True: If any element in the list is True
# False: If all elements in a list is false'''

L1=[1,2,3,4,0]
print(all(L1))
print(any(L1))

print('\n')
L2=[0,0,0,0,0,1]
print(all(L2))
print(any(L2))
```

```
False
True


False
True
```

```
# Copy

A=[1,2,3,4,5,6]
B=A.copy()

print(B)
```

```
[1, 2, 3, 4, 5, 6]
```

Tuple

- Ordered
- Immutable
- Duplicates allowed

1. count
2. index
3. remove
4. del
5. unpack and pack
6. join(+)
7. multiply(*)

```
T=(1,2,3,3,4,5,)
type(T)
```

```
tuple
```

```
# find count of 3 in tuple T
T.count(3)
```

```
2
```

```
# Gives the index, incase multiple values it gives the initial value index
T.index(3)
```

```
2
```

```
# Slicing
T[2:5]
```

```
(3, 3, 4)
```

```
T1=([1,2,3],4,5,6,[7,8,9])
print(type(T1))

print(T1[0])
print(T1[4][1])
```

```
<class 'tuple'>
[1, 2, 3]
8
```

```
# remove and del works same as lists
```

```
# Unpacking and Packing Tuples

fruits = ("apple", "banana", "cherry")
(green, yellow, red) = fruits
```

```
print(green)
print(yellow)
print(red)
```

```
apple
banana
cherry
```

```
fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")

(green, yellow, red) = fruits

print(green)
print(yellow)
print(red)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-48-97af59ec49dc> in <cell line: 3>()
      1 fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")
      2
----> 3 (green, yellow, red) = fruits
      4
      5 print(green)

ValueError: too many values to unpack (expected 3)
```

Next steps: Explain error

```
# The above error is due to unpacking the tuple with insufficient variables
# add * to any variable to assign values to the variable until the number of values left matches the number of variables left.

(green, *yellow, red) = fruits

print(green)
print(yellow)
print(red)
```

```
apple
['banana', 'cherry', 'strawberry']
raspberry
```

```
# Join Tuples
T1 = ("a", "b" , "c")
T2 = (1, 2, 3)

print(T1+T2)
```

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

```
#Multiply tuples
fruits = ("apple", "banana", "cherry")
mytuple = fruits * 2

print(mytuple)
```

```
('apple', 'banana', 'cherry', 'apple', 'banana', 'cherry')
```

Sets

- Only unique values
- Indexing & Slicing not allowed
- Duplicate value not allowed
- Add in any order it ends up in asc order
- Unchangeble

1. add
2. update
3. pop

4. remove
5. discard
6. union
7. intersection
8. difference
9. symmetric difference
10. subset/disjoint/superset

```
S={}
print(type(S)) # Above way of defining is a dict

S_1=set()
print(type(S_1)) #This is the way to define empty set


S_2={2,1,8,4,9,6}
print(type(S_2))
print(S_2)
```

```
<class 'dict'>
<class 'set'>
<class 'set'>
{1, 2, 4, 6, 8, 9}
```

```
#add
#adds the value in the last

S_2.add(7)
print(S_2)
```

```
{1, 2, 4, 6, 7, 8, 9}
```

```
# Slicing not possible
print(S_2[0])
print(S_2[:])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-54-d15f8285e050> in <cell line: 2>()
      1 # Slicing not possible
----> 2 print(S_2[0])
      3 print(S_2[:])

TypeError: 'set' object is not subscriptable
```

Next steps: Explain error

```
S_1=set()

S_1.add(4)
S_1.add(1)
S_1.add(8)
S_1.add(3)
S_1.add(10)

print(S_1)
print(len(S_1))
```

```
{1, 3, 4, 8, 10}
5
```

```
#update

t_set = {"apple", "banana", "cherry"}
c_set= {"pineapple", "mango", "papaya"}

t_set.update(c_set)

print(t_set)
```

```
{'cherry', 'papaya', 'apple', 'pineapple', 'mango', 'banana'}
```

```
# using update() we can join any iterables
t_set = {"apple", "banana", "cherry"}
mylist = ["kiwi", "orange"]

t_set.update(mylist)

print(t_set)
```

```
{'orange', 'cherry', 'apple', 'banana', 'kiwi'}
```

```
# pop() methods removes random item from the set. So prefer remove/discard

print(t_set)
t_set.pop()
print(t_set)
```

```
{'orange', 'cherry', 'apple', 'banana', 'kiwi'}
{'cherry', 'apple', 'banana', 'kiwi'}
```

```
# remove
# it removes the given value else throws error
t_set={'cherry', 'kiwi', 'banana', 'apple'}
t_set.remove('kiwi')
print(t_set)

t_set.remove(1) # this throws error as 1 doesnot exist in t_set
print(t_set)
```

```
{'cherry', 'banana', 'apple'}
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-59-a90a233a44a6> in <cell line: 7>()
      5 print(t_set)
      6
----> 7 t_set.remove(1) # this throws error as 1 doesnot exist in t_set
      8 print(t_set)

KeyError: 1
```

Next steps: Explain error

```
#discard- it removes the given value else dont throw any error
t_set={'cherry', 'apple', 'banana'}
t_set.discard('apple')
print(t_set)

t_set.discard('Hello')
print(t_set)
```

```
{'cherry', 'banana'}
{'cherry', 'banana'}
```

```
S_2 = {"apple", "banana", "cherry"}
for i in S_2:
  print(i)

# Slicing is not possible but using loops we can print contents of sets
```

```
cherry
banana
apple
```

```
A={1,2,3,4,5}
B={4,5,6,7,8}
C={8,9,10}

# Union
print(A.union(B))
print(A|B)
```

```
#Intersection
print(B.intersection(C))
print(B&C)

#Difference
#Prints only difference element of left side set
print(A.difference(B))
print(B.difference(C))

#Symmetric Difference
#Prints difference of both set
print(A.symmetric_difference(B))
print(B.symmetric_difference(C))
```

```
{1, 2, 3, 4, 5, 6, 7, 8}
{1, 2, 3, 4, 5, 6, 7, 8}
{8}
{8}
{1, 2, 3}
{4, 5, 6, 7}
{1, 2, 3, 6, 7, 8}
{4, 5, 6, 7, 9, 10}
```

```
A1={1,2,3,4,5,6,7,8,9}
B1={3,4,5,6,7,8}
C1={10,20,30,40}

#issubset
print(B1.issubset(A1))
print(C1.issubset(B1))

#isdisjoint
print(B1.isdisjoint(A1))
print(C1.isdisjoint(B1))

#issuperset
print(A1.issuperset(B1))
print(B1.issuperset(A1))
```

```
True
False
False
True
True
False
```

```
print(list(enumerate(A1)))

for i in enumerate(A1):
  print(i)
```

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

Dictionary

- Ordered
- changeble
- No duplicates
- key:value pairs

1. keys
2. values
3. items

4. pop
5. update
6. add
7. del
8. clear

```
D={1:'A',2:'B',3:'C',4:'D',5:'E'}
print(D.keys())
print(D.values())
print(D.items())
print(len(D))
```

```
dict_keys([1, 2, 3, 4, 5])
dict_values(['A', 'B', 'C', 'D', 'E'])
dict_items([(1, 'A'), (2, 'B'), (3, 'C'), (4, 'D'), (5, 'E')])
5
```

```
print(D[1])
print(D[5])
```

```
A
E
```

```
#Accessing Items

print(D[1]) #access using key

print(D.get(2)) #access using get method
```

```
A
B
```

```
#pop() can remove any item from a dictionary as long as you specify the key.
#popitem() can only remove and return the value of the last element in the dictionary.

print(D)

D.pop(2)

print(D)

D.popitem()
```

```
{1: 'A', 2: 'B', 3: 'C', 4: 'D', 5: 'E'}
{1: 'A', 3: 'C', 4: 'D', 5: 'E'}
(5, 'E')
```

```
# Update
print(D)
D.update({2:'B'})
D.update({5:'E'})

print(D)
```

```
{1: 'A', 3: 'C', 4: 'D'}
{1: 'A', 3: 'C', 4: 'D', 2: 'B', 5: 'E'}
```

```
#Add

D[6]='F'
print(D)
```

```
{1: 'A', 3: 'C', 4: 'D', 2: 'B', 5: 'E', 6: 'F'}
```

```
#del
# The del keyword removes the item with the specified key name or completely deletes

D_1= {'A':'Apple','B':'Banana','C':'Cat'}
print(D_1)

del D_1['C']
print(D_1)
```

```
del D_1
print(D_1) #Throws error because D_1 got deleted
```

```
{'A': 'Apple', 'B': 'Banana', 'C': 'Cat'}
{'A': 'Apple', 'B': 'Banana'}
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-72-4dc58eedb0d2> in <cell line: 11>()
      9
     10 del D_1
---> 11 print(D_1) #Throws error because D_1 got deleted

NameError: name 'D_1' is not defined
```

Next steps: Explain error

```
#Clear
D_1={'A':'Apple','B':'Banana','C':'Cat'}

print(D_1)

D_1.clear()

print(D_1)
```

```
{'A': 'Apple', 'B': 'Banana', 'C': 'Cat'}
{}
```

```
print(D)

for i in D:
  print(D[i])
print('\n')

for i in D.keys():
  print(i)
print('\n')
for i in D.values():
  print(i)
print('\n')

for i,j in D.items():
  print(i,j)
```

```
{1: 'A', 3: 'C', 4: 'D', 2: 'B', 5: 'E', 6: 'F'}
A
C
D
B
E
F


1
3
4
2
5
6


A
C
D
B
E
F


1 A
3 C
4 D
2 B
5 E
6 F
```

Python Loops & Conditions

```
#If

a=10
b=5

if a>b:
  print('a is greater than b')
```

```
a is greater than b
```

```
#Elif

a=10
b=5

if a<b:
  print('a is less than b')
elif a>b:
  print('a is greater than b')
```

```
a is greater than b
```

```
# Else

a=10
b=5

if a<b:
  print('a is less than b')
elif a==b:
  print('a is equal to b')
else:
  print('a is greater than b')
```

```
a is greater than b
```

```
# Short hand If

a=10
b=5

if a>b: print('a is greater than b')
```

```
a is greater than b
```

```
# Short hand If Else

a=10
b=5

print('a is greater than b') if a>b else print('a is less than b')
#
```

```
a is greater than b
```

```
# Nested If

a=45

if a>10:
  print('a is greater than 10')
  if a>20:
    print('a is also greater than 20')
  else:
    print('a is not greater than 20')
```

```
a is greater than 10
 a is also greater than 20
```

```
for i in range(10):
  print('Hello World:',i)
```

```
Hello World: 0
 Hello World: 1
 Hello World: 2
 Hello World: 3
 Hello World: 4
 Hello World: 5
 Hello World: 6
 Hello World: 7
 Hello World: 8
 Hello World: 9
```

```
i=0

while i<10:
  print('Hello World:',i)
  i+=1
```

```
Hello World: 0
 Hello World: 1
 Hello World: 2
 Hello World: 3
 Hello World: 4
 Hello World: 5
 Hello World: 6
 Hello World: 7
 Hello World: 8
 Hello World: 9
```

```
# Nested For loop

for i in range(3):
  print('1st Loop')
  for j in range(2):
    print('2nd Loop')
```

```
1st Loop
 2nd Loop
 2nd Loop
 1st Loop
 2nd Loop
 2nd Loop
 1st Loop
 2nd Loop
 2nd Loop
```

```
#Nested While loop

i=0
while i<3:
  print('1st Loop')
  i+=1
  j=0
  while j<2:
    print('2nd Loop')
    j+=1
```

```
1st Loop
2nd Loop
2nd Loop
1st Loop
2nd Loop
2nd Loop
1st Loop
2nd Loop
2nd Loop
```

When we have a condition then While Loop is used instead of For

More than 1 condition Nested is used

```
for i in range(10):
  print('Hello')

print('\n')

for j in range(10):
  print('World',end=' ') # End helps to define how the output should display

print('\n')

for k in range(10):
  print('Program',end='\\')
```

```
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello


World World World World World World World World World World

Program\Program\Program\Program\Program\Program\Program\Program\Program\Program\
```

For loop can be used to iterate any sequence of (list, string, int)

```
a="Hello"
for i in a:
  print(i)
print('\n')
b=['A',2,'C',4,1+2j,'Hello']
for j in b:
  print(j)
```

```
H
e
l
l
o


A
2
C
4
(1+2j)
Hello
```

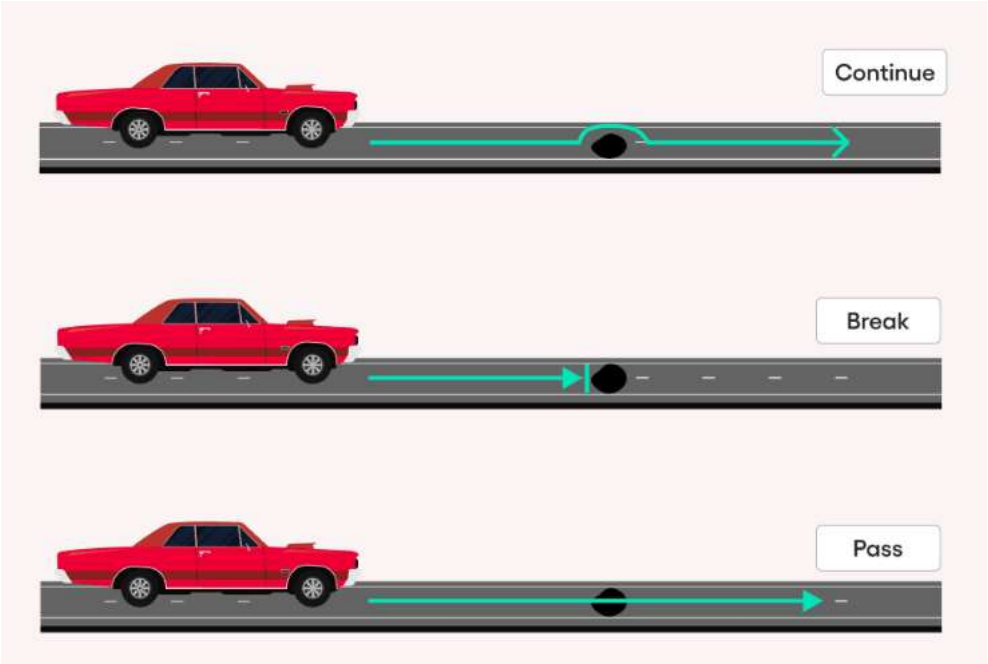
```
for i in range(0,21,5):
  print(i)
```

```
0
5
10
15
20
```

Q. Print the number which are divisible by 5

```
for i in range(1,41):
  if(i%5==0):
    print(i)
```

```
5
10
15
20
25
30
35
40
```




```
# Break
# terminates the loop and continuous

for i in range(5):
  if(i==3):
    break
  print("Hello")
```

```
Hello
Hello
Hello
```

```
# Continue
# Skips the iteration of loop when condition matches

for i in range(8):
  if(i==5):
    continue
  print(i)
```

```
0
1
2
3
4
6
7
```

```
# Pass - used as a placeholder for future code (nothing happens when executed)
# avoid getting error when empty code is not allowed
```

```
for i in range(3):

  pass
```

Q. Print the number from 1 to 50, but dont print the value which is divisible by 3 or 5

```
#using nested if condition
for i in range(1,51):
  if(i%3!=0):
    if(i%5!=0):
      print(i)
```

```
1
2
4
7
8
11
13
14
16
17
19
22
23
26
28
29
31
32
34
37
38
41
43
44
46
47
49
```

```
# Using continue
for i in range(1,51):
  if(i%3==0 or i%5==0):
    continue
  print(i)
print('end')
```

```
1
2
4
7
8
11
13
14
16
17
19
22
23
26
28
29
31
32
34
37
38
41
43
44
46
47
49
end
```

Q. Print the value which are not even number

```
# Using Pass
for i in range(20):
  if(i%2==0):
    pass
  else:
    print(i)
```

```
1
3
5
7
9
11
13
15
17
19
```

```
for i in range(20):
  if(i%2!=0):
    print(i)
```

```
1
3
5
7
9
11
13
15
17
19
```

Patterns in python

```
for i in range(4):
  print("#")
```

```
#
#
#
#
```

```
for i in range(4):
  print("#",end=" ")
```

```
# # # #
```

```
for i in range(4):
  for j in range(4):
    print("#",end=" ")
  print()
```

```
# # # #
# # # #
# # # #
# # # #
```

```
for i in range(4):
  for j in range(i+1):
    print('#',end=" ")
  print()
```

```
#
# #
# # #
# # # #
```

```
for i in range(4):
  for j in range(4-i):
    print('#',end=" ")
  print()
```

```
# # # #
# # #
# #
#
```

Q. Prime number between range of 1 to 20

```
for i in range(1,21):
  for j in range(2,i):
    if(i%j==0):
      break
  else:
    print(i)
```

```
1
2
3
5
7
11
13
17
19
```

Q. Find Given number is Prime or not

```
num=int(input("Enter the number:"))

for i in range(2,num//2):
  if(num%i)==0:
    print("Not Prime")
    break
else:
  print("Prime")
```

```
Enter the number:6
Not Prime
```

```
# 1Q. Even or Odd Number Write a program that checks if a number entered by the user is even or odd.

a=eval(input("Enter the number"))
if(a%2==0):
  print('Even')
else:
  print('Odd')
```

```
Enter the number7
Odd
```

```
# 2Q. Sum of First N Natural Numbers Write a program to find the sum of the first n natural numbers using a loop.

n=eval(input('Enter number'))
sum=0
for i in range(n+1):
  sum=sum+i

print(sum)
```

```
Enter number5
15
```

```
# 3Q. Print Multiplication Table Write a program to print the multiplication table of a given number using a loop.

n=eval(input('Enter number'))

for i in range(11):
  print(n,'*',i,'=',n*i)
```

```
Enter number5
5 * 0 = 0
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
```

```
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
```

```
# 4Q. Factorial of a Number Write a program to calculate the factorial of a number using a loop.

n=eval(input('Enter number'))
fact=1

if n<0:
  print('Enter Valid Input')
else:
  for i in range(1,n+1):
    fact*=i

print(fact)
```

```
Enter number8
40320
```

```
# 5Q. Fibonacci Sequence Write a program that prints the first n numbers of the Fibonacci sequence using loops.

def Fibo(n):
  if n < 0:
    print('Enter Valid Input')

  elif n==0:
    return 0

  elif n==1:
    return 1

  else:
    return Fibo(n-1)+Fibo(n-2)


n=eval(input('Enter number'))
print(Fibo(n))
```

```
Enter number6
8
```

```
# 6Q. Prime Number Check Write a program to check whether a number is prime using a loop and conditional statements.

n=eval(input('Enter number'))

if n>1:
  for i in range(2,(n//2)+1):
    if (n%i)==0:
      print(n,'is not Prime')
      break
  else:
    print(n,'is a Prime')

else:
  print(n,'is not Prime')
```

```
Enter number7
7 is a Prime
```

```
# 7Q. Sum of Digits Write a program to calculate the sum of digits of a number using a loop.

n=eval(input('Enter number'))
sum=0
while(n>0):
  digit=n%10
  sum+=digit
  n=n//10
```

```
print(sum)
```

```
Enter number5774
23
```

```
# 8Q. Reverse a String Write a program to reverse a string input by the user using loops.

n=input('Enter number')
rev=''
for i in n:
  rev=i+rev

print(rev)
```

```
Enter number123
321
```

```
# 9Q. Find Maximum in a List Write a program to find the maximum element in a list using a loop.

List_len=eval(input("Enter Length of List:"))
A=[]
for i in range(List_len):
  n=eval(input("Enter Length of List:"))
  A.append(n)

big=0
for i in A:
  if i>big:
    big=i

print(big)
```

```
Enter Length of List:4
Enter Length of List:1
Enter Length of List:8
Enter Length of List:3
Enter Length of List:6
8
```

```
# 10Q. Count Vowels in a String Write a program to count the number of vowels in a string using loops and conditions.

a=input("Enter the String")
count=0
vowel=['a','e','i','o','u','A','E','I','O','U']
for i in a:
  if(i in vowel):
    count+=1

print(count)
```

```
Enter the StringHELLO
2
```

```
# 11Q. Palindrome Check Write a program to check whether a given number or string is a palindrome.

def is_pal(s):
  inp_val_str=str(inp_val)
  length=len(inp_val_str)

  for i in range(length//2):
    if inp_val_str[i]!=inp_val_str[length-i-1]:
      return False
  return True


inp_val=input('Enter the number or string')

if is_pal(inp_val):
  print(inp_val,'is Palindrome')
else:
  print(inp_val,'is not Palindrome')
```

```
Enter the number or stringbasab
basab is Palindrome
```

```
# 12Q. Find Prime Numbers in a Range Write a program that prints all prime numbers in a given range using loops and conditions.

a=int(input('Enter the number till what you want the prime number'))

for i in range(2,a+1):
  for j in range(2,i):
    if(i%j==0):
      break
  else:
    print(i)
```

```
Enter the number till what you want the prime number5
2
3
5
```

```
# 13Q. Pattern Printing Write a program to print the following pattern:
for i in range(4):
  for j in range(4):
    print('*', end=' ')
  print()
```

```
* * * *
* * * *
* * * *
* * * *
```

```
# 14Q. Sum of Even Numbers in a List Write a program to find the sum of all even numbers in a list using a loop and condition.
sum=0
L=[1,2,3,4,5,6,7,8,9,10]
for i in L:
  if(i%2==0):
    sum+=i

print(sum)
```

```
30
```

```
# 15Q. GCD of Two Numbers Write a program to find the greatest common divisor (GCD) of two numbers using loops.
a=int(input('Enter 1st number'))
b=int(input('Enter 2nd number'))

while b!=0:
  rem=a%b
  a=b
  b=rem
gcd=a

print(gcd)
```

```
Enter 1st number45
Enter 2nd number32
1
```

```
# 16Q. Count Occurrences of a Character Write a program to count how many times a specific character appears in a string using loops and con

S=input('Enter the String')
E=input('Enter the word to get the count')
count=0
for i in S:
  if i==E:
    count+=1

print(f'The word {E} appeared {count} times')
```

```
Enter the StringHello
Enter the word to get the countl
The word l appeared 2 times
```

```
# 17Q. Find Common Elements in Two Lists Write a program to find the common elements between two lists using loops and conditions.
A=[1,2,3,4,5]
B=[3,4,5,6,7]
C=[]

for i in A:
  if i in B and i not in C:
    C.append(i)

print(C)
```

```
[3, 4, 5]
```

```
# 18Q. Armstrong Number Write a program to check if a number is an Armstrong number.
A=int(input('Enter the number to find Armstring'))
n=len(str(A))
temp=A
sum=0
while temp > 0:
  digit=temp%10
  sum+=digit**n
  temp=temp//10

if A==sum:
  print('Armstrong')
else:
  print('Not Armstrong')
```

```
Enter the number to find Armstring153
Armstrong
```

```
# 19Q. Find the Second Largest Element in a List Write a program to find the second largest element in a list using a loop.
A=[1,2,3,4,5,6,7,8,9]
largest=0
second_l=0
for i in A:
  if i > largest:
    largest=i

for i in A:
  if i>second_l and i<largest:
    second_l=i

print(second_l)
```

```
8
```