In [1]:
```
print("hello world !!!", "Myself Himanshu")
print(35-25)
```

```
hello world !!! Myself Himanshu
10
```

In [87]:
```
a=str(10)
b=10
print(int(a)+10)
print(int(a)+10.31)
print(int(a)+b)
print(a+str(10))
```

```
20
20.310000000000002
20
1010
```

In [110…
```
a=10
b="hello" # single/Double/tripple quotes all are allowed
c=input("Enter a number: ") # by default input is string only
check=True # False [first letter is capital]
print(type(a))
print(type(b))
print(type(c))
print(type(check))
```

```
Enter a number: 21
<class 'int'>
<class 'str'>
<class 'str'>
<class 'bool'>
```

In [4]:
```
'''NoneType in python
#uses
1. Default Return Value for Functions
2. Placeholder for Optional Arguments
def connect(hostname, port=None):
    if port is None:
        port = 8080
    print(f"Connecting to {hostname} on port {port}")

connect("localhost")  # Default port 8080 will be used

3. Initializing variables
4. Representing the end of a list


'''
a=None
print(type(a))
```

```
<class 'NoneType'>
```

Note: Python is case sensitive

In [5]:
```
"""
This is a multi line comment
using 3 double quotes
"""
'''
This is also a multi line comment
using 3 single quotes
'''
print("Hello")
```

```
Hello
```

In [6]:
```
a=10
b=2
c=a/2 # division operator always return a float value
print(type(a))
print(type(b))
print(type(c))

"""
% = modulo = remainder operator
** = power operator e.g. 5**2 = 5^2 =25
"""
print(a%b)
print(a**b)

print(a==b) # return a boolean value
print(a!=b)
print(a>b)
print(a<b)
```

```
<class 'int'>
<class 'int'>
<class 'float'>
0
100
False
True
True
False
```

Strings

In [111…
```
a=input("Enter a: ")
print("Type of a is",type(a))
b=int(input("Enter b: "))
print("Type of b is",type(b))
```

```
Enter a: 10
Type of a is <class 'str'>
Enter b: 21
Type of b is <class 'int'>
```

In [9]:
```
str1="This is a string"
str2='This is also a string'
str3='''Another string'''
str4="""same string"""
for i in range(1,5):
    temp="str"+str(i)
    print(temp,"is of type",type(globals()[temp]),globals()[temp])
    #globals()[var_name] dynamically creates a global variable with the name
    # you can also use locals()
```

```
str1 is of type <class 'str'> This is a string
str2 is of type <class 'str'> This is also a string
str3 is of type <class 'str'> Another string
str4 is of type <class 'str'> same string
```

In [10]:
```
#String indexing

str1="hello this is a string"
print(str1[0])

#note: String does not support item assignment..they are read only


#String Slicing

print(str1[0:10])
print(str1[:10]) # same as [0:10]
print(str1[5:])
print(str1[::-1]) # this will print the string in reverse

#negative indexing

print(str1[-1]) # last element of the original string
print(str1[-len(str1)]) # first element of the original string
print(str1[-len(str1):-15])
```

```
h
hello this
hello this
 this is a string
gnirts a si siht olleh
g
h
hello t
```

Traversing a string

In [26]:
```
#method 1 using index
s="hello123"
for i in range(len(s)): # len() is an inbuilt function to find the length of
    print(s[i],end=" ")
print("")

i=0
while i<len(s): # forward traversing
```

```
    print(s[i],end=" ")
    i+=1

print("")

i=-1
while i>=-len(s): # reverse traverssing
    print(s[i],end=" ")
    i-=1

print("")

#method 2 using slice operator


for num in s[:]:   # forward traversing
    print(num,end=" ")
print(" ")

for num in s[::-1]:
    print(num,end=" ")
```

```
h e l l o 1 2 3
h e l l o 1 2 3
3 2 1 o l l e h
h e l l o 1 2 3
3 2 1 o l l e h
```

Mathematical operators in string

In [31]:
```
s="hello"
print(s*2) # multiplying the array with duplicate the string
print(s+s)
```

```
hellohello
hellohello
```

Membership operator in string

In [27]:
```
s='hello'
print('o' in s)

print('a' in s)
```

```
True
False
```

String Comaprison operator

In [33]:
```
# Less than (<): Meaning: Checks if the first string comes before the second
print("apple" < "banana")  # True (because 'a' comes before 'b' in Unicode)
print("banana" < "apple")  # False
```

```
True
False
```

Remove space and other characters from the end/beginning of the string

1. rstrip() ===>To remove blank spaces present at end of the string (i.e.,right hand side)

2. lstrip()===>To remove blank spaces present at the beginning of the string (i.e.,left hand side)

3. strip() ==>To remove spaces both sides

In [35]:
```
text = "  Hello World!   "
print(text.rstrip())  # Output: "  Hello World!"
print(text.lstrip())  # Output: "Hello World!   "
print(text.strip())  # Output: "Hello World!"

# to remove other characters from
text = "!!!Hello World!!!"
print(text.rstrip("!"))  # Output: "!!!Hello World"
print(text.lstrip("!"))  # Output: "Hello World!!!"
print(text.strip("!"))   # Output: "Hello World"
```

```
  Hello World!
Hello World!
Hello World!
!!!Hello World
Hello World!!!
Hello World
```

Finding substrings

```
 For forward direction:                                  For backward
direction:
    1. find()                                                1.
rfind()
    2. index()                                               2.
rindex()
```

Note: These functions returns the first index of the matching substring in the main string

By default find() method can search total string. We can also specify the boundaries to search.

```
s.find(substring,begin,end) ----[begin,end]
    It will always search from begin index to end-1 index.
```

Finding sub-strinig using index method

index() method:

index() method is exactly same as find() method except that if the specified substring is not available then we will get ValueError

In [51]:
```
s="Learning Python is very easy"
print(s.find("Python")) # 9
print(s.find("Java")) # -1
print(s.find("r")) # 3
print(s.rfind("r")) # 21
print(s.find("is",0,10)) # will return -1 since "is" is not between [0,10]

# using index method
print(s.index("Python",0,15)) #9

try:
    temp=s.index("java") # if not found then it will return an error
except ValueError:
    print("not found")
else:
    print(found)
```

```
9
-1
3
21
-1
9
not found
```

Counting substring in the given String:

1. s.count(substring) ==> It will search through out the string

2. s.count(substring, begin, end) ===> It will search from begin index to end-1 index

In [53]:
```
s="abcabcabcabcadda"
print(s.count('a')) #6
print(s.count('ab')) #4
print(s.count('a',3,7)) #2
```

```
6
4
2
```

In [61]:
```
# Ques) Write a Python Program to display all positions of substring in a gi
s="abcabcabcabcadda"
sub="ab"
i=0
match=[]
while i<len(s):
    loc=s.find(sub,i,len(s))
    if loc!=-1: # means we find the sub-string
        match.append(i)
        i=loc+len(sub)
    else:
        break
print(match)
print(f"we got \"{sub}\" {len(match)} times in \"{s}\"")
```

```
[0, 2, 5, 8]
we got "ab" 4 times in "abcabcabcabcadda"
```

Replacing a string with another string:

s.replace(oldstring,newstring)

We know strings are immutable therefore using replace function will create a new object

meaning it wont override on the old object

In [64]:
```
s="Learning Python is very difficult"
print(s,"is available at :",id(s))
s=s.replace("difficult","easy")
print(s,"is available at :",id(s))
```

```
Learning Python is very difficult is available at : 2129989932304
Learning Python is very easy is available at : 2129988265024
```

Splitting of Strings:

We can split the given string according to specified
seperator by using split() method.
We can split the given string according to specified
seperator in reverse direction by using rsplit() method

Note: There is no lsplit()

In [73]:
```
s="hello this is   python"
l=s.split()
print(f"l = {l} and is of type \"{type(l)}\"")
print(s.split('s')) # split using other separator [deafault = blank space]

# you can also define maximum splits that you want
print(s.split(' ',1))
```

```
l = ['hello', 'this', 'is', 'python'] and is of type "<class 'list'>"
['hello thi', ' i', '   python']
['hello', 'this is   python']
```

joining a string

We can join a group of strings(list or tuple) with respect to the given seperator

s=seperator.join(group of strings)

In [77]:
```
t=('sunny','bunny','chinny')
print('-'.join(t))
print(''.join(t))
print(':::'.join(t))
```

```
sunny-bunny-chinny
sunnybunnychinny
sunny:::bunny:::chinny
```

Changing case of a String:

1. upper()===>To convert all characters to upper case

2. lower() ===>To convert all characters to lower case

3. swapcase()===>converts all lower case characters to upper case and all upper case characters to lower case

4. title() ===>To convert all characters to title case. i.e first character in every word should be upper case and all remaining characters should be in lower case.

5. capitalize() ==>Only first character will be converted to upper case and all remaining characters can beconverted to lower case.

In [80]:
```
s='learning Python is very Easy'
print(s.upper())
print(s.lower())
print(s.swapcase())
print(s.title()) #first letter of each word will be capitalized
print(s.capitalize())
```

```
LEARNING PYTHON IS VERY EASY
learning python is very easy
LEARNING pYTHON IS VERY eASY
Learning Python Is Very Easy
Learning python is very easy
```

Checking starting and ending part of the string:

In [84]:
```
test="aabbccdd"
print(test.endswith("dd"))
print(test.endswith("e"))

s='learning Python is very easy'
print(s.startswith('learning'))
print(s.endswith('learning'))
print(s.endswith('easy'))
```

```
True
False
True
False
True
```

To check type of characters present in a string:

1) isalnum(): Returns True if all characters are alphanumeric(a to z , A to Z ,0 to9)

2) isalpha(): Returns True if all characters are only alphabet symbols(a to z,A to Z)

3) isdigit(): Returns True if all characters are digits only(0 to 9)

4) islower(): Returns True if all characters are lower case alphabet symbols

5) isupper(): Returns True if all characters are upper case aplhabet symbols

6) istitle(): Returns True if string is in title case

7) isspace(): Returns True if string contains only spaces

Note : We can't pass any arguments to these functions.

In [108…
```
# chr(unicode)===>The corresponding character
# ord(character)===>The corresponding unicode value

print(f"chr(97) = {chr(97)}")
print(f"chr(65) = {chr(65)}")

print(f"ord('a') = {ord('a')}")
print(f"ord('A') = {ord('A')}")
```

```
chr(97) = a
chr(65) = A
ord('a') = 97
ord('A') = 65
```

Conditional Statement

In [12]:
```
age=18
if(age >=18):
    print("Attained legal age")
else:
    print("Still a minor")
```

```
Attained legal age
```

In [97]:
```
def find_grade(x):
    if(x>=90):
        return "A"
    elif(x>=80 and x<90):
        return "B"
    elif(x>=70 and x<80):
        return "C"
    else:
        return "D"

x=int(input("Enter your makrs : "))
print("Your final grade is",find_grade(x))
```

```
Enter your makrs : 65
Your final grade is D
```

In [98]:
```
#check odd or even
a=int(input("Eneter a number to check: "))
if(a%2 == 0):
    print("even")
else:
    print("Odd")
```

```
Eneter a number to check: 21
Odd
```

In [99]:
```
#greatest of three number
a=int(input("Enter the 1st number "))
b=int(input("Enter the 2nd number "))
c=int(input("Enter the 3rd number "))
```

```
if(a>=b and a>=c):
    print("a is largest",a)
elif(b>=a and b>=c):
    print("b is largest",b)
else:
    print("c is largest",c)
```

```
Enter the 1st number 12
Enter the 2nd number 45
Enter the 3rd number 10
b is largest 45
```

List and Tuples

In [16]:
```
list1=["a",10,"b"] # list can store multiple data type in once
length=len(list1)
print("Length of list1 is ",length)
for i in range (0,length):
    print(list1[i],"is of type",type(list1[i]))

print(list1[-1]) # List also allow negative indexing
print(list1[-length])

#Stings are Mutubale but List is Mutable
print(list1[::-1]) # reverse the list
print(list1[-1:-length-1:-1])
```

```
Length of list1 is  3
a is of type <class 'str'>
10 is of type <class 'int'>
b is of type <class 'str'>
b
a
['b', 10, 'a']
['b', 10, 'a']
```

list Methods

list.append(4) #adds one element at the end

list.sort( ) #sorts in ascending order

list.sort(reverse=True) #sorts in descending order

list.reverse( ) #reverses list

list.insert(idx, el) #insert element at index

```
    Note: One important thing about inser if..if wr give an
higher index > length of array
```

```
                then the insert function will act like append
                arr=[3, 2, 6, 4, 3, 2, 1]
                arr.insert(10,-7)  -----> we can see index 10 is
    out of the bounds
                print(arr)
                [3, 2, 6, 4, 3, 2, 1, -7]
```

list.pop(idx) #removes element at idx

list.count(value) #Return the number of times the value appears in the list

In [17]:
```
list1=[2,1,3,1]
list1.append(-1)
print(list1[3])
list1.sort() # sorting and saving the list
print(list1)
list1.sort(reverse=True)
print(list1)
list1.insert(1,4) # <index,value>
print(list1)

print(list1.count(1)) #Number of times int<1> occurs in the list
print(list1.count(-10)) # returb zero if not found
```

```
1
[-1, 1, 1, 2, 3]
[3, 2, 1, 1, -1]
[3, 4, 2, 1, 1, -1]
2
0
```

In [18]:
```
#list of strings sorting
list1=["aa","ab","ba","ac","bc","bb"]
print(list1)
list1.sort()
print(list1)
```

```
['aa', 'ab', 'ba', 'ac', 'bc', 'bb']
['aa', 'ab', 'ac', 'ba', 'bb', 'bc']
```

Here's a table that summarizes the key differences between **lists** and **tuples** in Python: | Feature | List | Tuple | |----------|----------|----------| | **Mutability** | Mutable (can be changed) | Immutable (cannot be changed) | | **Syntax** | `[1, 2, 3]` | `(1, 2, 3)` | | **Performance** | Slower due to mutability | Faster due to immutability | | **Methods Available** | Many (e.g., `append()`, `remove()`, `sort()`) | Limited (`count()`, `index()`) | | **Memory Usage** | Larger | Smaller | | **Use Case** | Use when data changes | Use for fixed, constant data | | **Iteration Speed** | Slower | Faster | | **Hashable** | No (cannot be used as dictionary keys) | Yes (if all elements are hashable) | | **Modification** | Elements can be added, removed, or changed | Cannot modify elements | | **Nested Structures** | Can contain other lists | Can contain other tuples | | **Element Addition** | Possible (`append()`, `insert()`) | Not possible (must create a new tuple) | | **Indexing and Slicing** | Supported | Supported | | **Preferred For** | When you need to modify data frequently | When you need to protect data from modification | This table offers a quick overview of the similarities and differences between lists and tuples in Python.

In [19]:
```
#tuple [immutable = Cannot be changed]
tup1=(1,1,2,2,3,3,4,4)
print(type(tup1))
```

```
print(tup1[2])

#To Create a single value Tuple
tup2=(1)
print(type(tup2))
tup2=(1,) # add a comma at the end
print(type(tup2))
```

```
<class 'tuple'>
2
<class 'int'>
<class 'tuple'>
```

Tuple Methods

tup.index(el) # returns index of 1st occurence

tup.count(el) # Counts total occurence

l=[] l.append(int(input("enter 1st number: "))) l.append(int(input("enter 2nd number: "))) l.append(int(input("enter 3rd number: "))) print("original list = ",l) l.sort() print("sorted list = ",l)

In [88]:
```
def check_pal(a):
    length=len(a)
    flag=0
    for i in range(0,int(length/2)+1):
        if(a[i]!=a[-1-i]):
            print("Not a palindrome")
            return
    print("Palindrome")


def check_pal2(a):
    b=a.copy()
    b.reverse()
    if(a==b):
        print("palindrome")
    else:
        print("Not a palindrome")

def check_pal3(a):
    if(a==a[::-1]):
        print("palindrome")
    else:
        print("Not a palindrome")


# programm to check if the list contains a palindrom of elements
l=[1,2,3,2,1]
# use this if you take an input for user " list(map(int, temp.split(','))) "
print(l)
```

```
print(l[0])
check_pal(l)
check_pal2(l)
check_pal3(l)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-88-deaddae7621b> in <module>
     26 # programm to check if the list contains a palindrom of elements
     27 temp=[1,2,3,2,1]
---> 28 l = list(map(int, temp.split(',')))
     29 print(l)
     30 print(l[0])

AttributeError: 'list' object has no attribute 'split'
```

In [22]:
```
a=[1,2,3]
print(a)
print(a[::-1]) # same as a.reverse()
```

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

Dictionary and sets

In [23]:
```
# Dictionary are unordered, Mutable(Changable) & it dont allow duplicate key

dict = {
    "name" : "HB",
    "age" : 27,
    "hobbies" : ["eating","sleeping"]
}
print(dict)
print(dict["name"],"is of type",type(dict["name"]))
print(dict["age"],type(dict["age"]))
print(dict["hobbies"],"is of type",type(dict["hobbies"]))

# Changing the values of dictionaries
dict["hobbies"] = ["coding", "running"]
print(dict)
```

```
{'name': 'HB', 'age': 27, 'hobbies': ['eating', 'sleeping']}
HB is of type <class 'str'>
27 <class 'int'>
['eating', 'sleeping'] is of type <class 'list'>
{'name': 'HB', 'age': 27, 'hobbies': ['coding', 'running']}
```

In [24]:
```
#Nested Dictionaries

student = {
    "name" : "HB",
    "score" : {
        "math" : 95,
        "C/C++" : 70,
        "python" : 100
```

```
    }
}
print(student)
print(student["score"])
print(student["score"]["python"]) # way to access nested dictionary
```

```
{'name': 'HB', 'score': {'math': 95, 'C/C++': 70, 'python': 100}}
{'math': 95, 'C/C++': 70, 'python': 100}
100
```

Dictionary Methods

myDict.keys( ) #returns all keys

myDict.values( ) #returns all values

myDict.items( ) #returns all (key, val) pairs as tuples

myDict.get(“key““) #returns the valeu according to the key

myDict.update(newDict) #inserts the specified items to the dictionary

In [25]:
```
dict = {
    "name" : "HB",
    "age" : 27,
    "hobbies" : ["eating","sleeping"]
}

print(dict.keys())
print(dict.values())
print(dict.items())
print(dict.get("agesss","Key not found")) # get function prevent error when

#how to find length of dictionary
length = len(dict.keys())
print("Length of dictionary is",length)

#how to use items fuction
temp = list(dict.items())
print(temp,"is of type",type(temp))
print(temp[0]) # it will print a tuple with 2 values ('key','value')
print(temp[1])

print(temp[0][0],temp[0][1])

#using update function [Adding new key:value pair]
print(dict)
dict.update({"city":"Ghaziabad"})
print(dict)
```

```
dict_keys(['name', 'age', 'hobbies'])
dict_values(['HB', 27, ['eating', 'sleeping']])
dict_items([('name', 'HB'), ('age', 27), ('hobbies', ['eating', 'sleepin
g'])])
Key not found
Length of dictionary is 3
[('name', 'HB'), ('age', 27), ('hobbies', ['eating', 'sleeping'])] is of typ
e <class 'list'>
('name', 'HB')
('age', 27)
name HB
{'name': 'HB', 'age': 27, 'hobbies': ['eating', 'sleeping']}
{'name': 'HB', 'age': 27, 'hobbies': ['eating', 'sleeping'], 'city': 'Ghazia
bad'}
```

In [26]:
```
#multiple values to a single key
dict = {
    "table" : ["a piece of furniture","list of facts & figure"],
    "cat" : "a small animal"
}
dict.items()
print(dict["table"])
print(len(dict["table"]))
print(dict["table"][0])
print(dict["table"][1])
```

```
['a piece of furniture', 'list of facts & figure']
2
a piece of furniture
list of facts & figure
```

In [112…
```
#taking input for dictionaries in python
dict={}
dict = {
    "chem" : int(input("enter chem marks: ")),
    "math" : int(input("enter math marks: ")),
    "bio" : int(input("enter bio marks: "))
}
print(dict,"is of type",type(dict))
```

```
enter chem marks: 54
enter math marks: 87
enter bio marks: 90
{'chem': 54, 'math': 87, 'bio': 90} is of type <class 'dict'>
```

Sets

Set is the collection of unordered items

Each elements in the set must be unique and immutable [but set is mutable]

In [28]:
```
collection = {1,2,3,4}
print(collection,"is of type",type(collection))
```

```
temp = {1,2,2,4}
print("temp =",temp) # any dupicate value will be automatically deleted
print(temp,"is of type",type(temp))

temp2=set() # to create an empty set
print(type(temp2))
```

```
{1, 2, 3, 4} is of type <class 'set'>
temp = {1, 2, 4}
{1, 2, 4} is of type <class 'set'>
<class 'set'>
```

Set Methods

set.add(el) #adds an element

set.remove(el) #removes the element

set.clear( ) #empties the set

set.pop( ) #removes a random value

set.union(set2) #combines both set values & returns new

set.intersection(set2) #combines common values & returns new

Note: We cannot add array data type to a set

In [29]:
```
a={1,2,3}
b={2,3,4}
a.union(b)
print(a)
print(a.union(b))
print(a.intersection(b))
```

```
{1, 2, 3}
{1, 2, 3, 4}
{2, 3}
```

In [30]:
```
# Define a list
my_list = [1, 2, 3, 4, 5, 5, 6, 6]
print(my_list)

# Convert the list to a set
my_set = set(my_list)
```

```
# Print the set
print(my_set)
```

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

In [31]:
```
print(type(float(9.0)))
temp ={9,float(9.0)}
print(temp)

'''
In Python, when comparing 9 (integer) and 9.0 (float),
they are treated as equal since both represent the same value.
The set only keeps one of the two since they are considered duplicates.
Example:

print(type(float(9.0)))  # Output: <class 'float'>

temp = {9, float(9.0)}   # Set tries to store both 9 and 9.0
print(temp)              # Output: {9}
'''
```

```
<class 'float'>
{9}
```

Out[31]:
```
"\nIn Python, when comparing 9 (integer) and 9.0 (float), \nthey are treate
d as equal since both represent the same value.\nThe set only keeps one of
the two since they are considered duplicates.\nExample:\n\nprint(type(float
(9.0)))  # Output: <class 'float'>\n\ntemp = {9, float(9.0)}   # Set tries
to store both 9 and 9.0\nprint(temp)              # Output: {9}\n"
```

While Loop

In [37]:
```
n=0
while True:
    print("n =",n,"hello")
    if(n==4):
        break
    n+=1
```

```
n = 0 hello
n = 1 hello
n = 2 hello
n = 3 hello
n = 4 hello
```

Let‘s Practice

1. Print numbers from 1 to 100.
2. Print numbers from 100 to 1.
3. Print the multiplication table of a number n.

4. Print the elements of the following list using a loop: [1, 4, 9, 16, 25, 36, 49, 64, 81,100]

5. Search for a number x in this tuple using loop: [1, 4, 9, 16, 25, 36, 49, 64, 81,100]

In [57]:
```
a=[1, 4, 9, 16, 25, 36, 49, 64, 81,100]
n=0

while n<len(a):
    print(a[n],end=" ")
    n+=1
print("")

#print table
n=1
x=3
while n<=10:
    print(x*n,end=" ")
    n+=1

#find x in the tuple
temp=(1, 4, 9, 16, 25, 36, 49, 64, 81,100)
n=0
found=False
x=25
while n<len(temp):
    if(x==temp[n]):
        found=True
        break
    n+=1
if found:
    print(f"\n{x} is found in the tuple {temp}.")
else:
    print(f"\n{x} is not found in the tuple {temp}.")

#Note: f in print will allow you to embed expressions inside string literals
```

```
1 4 9 16 25 36 49 64 81 100
3 6 9 12 15 18 21 24 27 30
25 is found in the tuple(1, 4, 9, 16, 25, 36, 49, 64, 81, 100).
```

In [62]:
```
#continue in python will skip the remaining lines below
# and jump to the next interation

n=0
while(n<20):
    n+=1
    if(n%2==0):
        continue
    print(n,end=" ")
```

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

For loop

In [69]:
```
l=[1,2,3,4,]
for i in l:
    print(i,end=" ")
else:
    print("End") # this else is not compulsory

for i in "hello its a string":
    print(i,end=" ")
```

```
1 2 3 4 End
h e l l o   i t s   a   s t r i n g
```

In [113…
```
# search a number x in the tuple using for loop
a=(1, 4, 9, 16, 25, 36, 49, 64, 81,100)
x=int(input("Enter the number to find: "))
for i in a:
    if(i==x):
        print(f"found {x} in the tuple {a}")
        break
else: # this else will only run when the complete "for loop" will run withou
    print(f"{x} not found")
```

```
Enter the number to find: 4
found 4 in the tuple (1, 4, 9, 16, 25, 36, 49, 64, 81, 100)
```

In [87]:
```
for i in range(1,15,2): #<start,end,step_size>
    print(i,end=" ")

print("")
for i in range(-1,-10,-1): # For loop always exculde stop value but do inclu
    print(i,end=" ")

print("")
for i in range(10):
    print(i,end=" ")
```

```
1 3 5 7 9 11 13
-1 -2 -3 -4 -5 -6 -7 -8 -9
0 1 2 3 4 5 6 7 8 9
```

In [114…
```
#factorial
a=int(input("Enter a number"))
for i in range(1,a):
    a=a*i
print(a)
```

```
Enter a number5
120
```

In [115…
```
#factorial
a=int(input("Enter a number"))
n=1
fact=1
```

```
while(n<=a):
    fact*=n
    n+=1
print(fact)
```

```
Enter a number10
3628800
```

Functions

In [102…
```
def sum(a,b):
    return (a+b)

print(sum(10,5))
```

```
15
```

NoneType in python #uses 1. Default Return Value for Functions 2. Placeholder for Optional Arguments def connect(hostname, port=None): if port is None: port = 8080 print(f"Connecting to {hostname} on port {port}") connect("localhost") # Default port 8080 will be used 3. Initializing variables 4. Representing the end of a list

In [108…
```
#default value in function
def cal_prod(a=1,b=1):
    print(a*b)
    return (a*b)

cal_prod(10) # by default b is one if not provided
cal_prod(2,7)
cal_prod(b=5) # directly give value to arguments
```

```
10
14
5
```

Out[108… 5

In [111…
```
def show(n):
    if(n==0):
        return
    print(n)
    show(n-1)

show(5)
```

```
5
4
3
2
1
```

In [127…
```
#factorial using recurssion
def factorial(n):
    if(n==1 or n==0):
        return 1
    else:
        return n*factorial(n-1)
```

```
print(factorial(4))
```

```
24
```

sum of (1,n) using recurssion

def sum(n): if(n==1): return 1 else: return n+sum(n-1)

print(sum(3))

Arrays

In [3]:
```
import array
arr=array.array('i',[1,2,3,4]) # here 'i' specifies integer
print(arr,type(arr))
```

```
array('i', [1, 2, 3, 4]) <class 'array.array'>
```

In [4]:
```
import array as ar # give alias to array library
arr=ar.array('i',[1,2,3,4]) # here 'i' specifies integer
print(arr,type(arr))
```

```
array('i', [1, 2, 3, 4]) <class 'array.array'>
```

Supported types in the array module: 'b': signed char 'B': unsigned char 'u': Unicode character 'h': signed short 'H': unsigned short 'i': signed int 'I': unsigned int 'f': float 'd': double

In [27]:
```
#Operations in Array
import array
arr = array.array('i', [1, 2, 3])
print(arr[1])

#slicing array
print(arr[::-1])
print(arr[1:3])

#looping through array
for elem in arr:
    print(elem,end=" ")

#adding elements to an array
arr.append(10)
print("\n",arr)

#deleting elemnts of an array

arr.pop()     # Removes last element
arr.remove(3) # Removes the first occurrence of 3

print(arr)
```

```
2
array('i', [3, 2, 1])
array('i', [2, 3])
1 2 3
 array('i', [1, 2, 3, 10])
array('i', [1, 2])
```

Feature	Array	List
Data Type	Homogeneous (same type)	Heterogeneous (mixed types)
Performance	Faster for numerical tasks	Slower for large datasets
Memory Usage	More memory-efficient	Less memory-efficient
Built-in Support	Requires `array` or `numpy` module	Native to Python
Methods	Limited methods for array manipulation	Extensive list methods (e.g., append, remove)
Flexibility	Fixed data type	Can hold different data types (e.g., int, str)

In [29]:
```python
# Merge two arrays
import array as arr

array1 = arr.array('i', [1, 2, 3])
array2 = arr.array('i', [4, 5, 6])
merged_array = array1 + array2
print(f"Merged array: {merged_array}")
```

```
Merged array: array('i', [1, 2, 3, 4, 5, 6])
```

In [30]:
```python
#count the occurence of an element in an array
import array as arr

my_array = arr.array('i', [1, 2, 3, 4, 2, 2, 5])
print("Occurrence of 2: ", my_array.count(2))
```

```
Occurrence of 2:  3
```

In [33]:
```python
#sort an array
import array as arr

# Create an array
my_array = arr.array('i', [4, 2, 5, 1, 3])

# Convert the array to a list, sort the list, then convert it back to an arr
sorted_array = arr.array('i', sorted(my_array))

print("Original array: ", my_array)
print("Sorted array: ", sorted_array)
```

```
Original array:  array('i', [4, 2, 5, 1, 3])
Sorted array:  array('i', [1, 2, 3, 4, 5])
```

In [39]:
```python
import array as arr

# Create an array
my_array = arr.array('i', [4, 2, 5, 1, 3])
print(sorted(my_array,reverse=True))
```

```
#sorted() will return a list only no matter what the input type was
```

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

Feature	`sort()`	`sorted()`
Usage	A method for lists only.	A function that can be applied to any iterable (lists, tuples, strings, etc.).
In-place vs Copy	Sorts the list in place, modifying the original list.	Returns a new sorted list without modifying the original iterable.
Return value	Returns `None` (modifies list in-place).	Returns a new sorted list.
Applicable to	Only works for lists.	Can be applied to lists, tuples, dictionaries, strings, and any iterable.
Syntax	`list.sort(key=None, reverse=False)`	`sorted(iterable, key=None, reverse=False)`
Reversing order	Use `reverse=True` argument for descending order.	Use `reverse=True` argument for descending order.
Example	`my_list.sort()`	`sorted(my_list)`

In [41]:
```
import array as arr

# Create an array
my_array = arr.array('i', [4, 2, 5, 1, 3])
temp=sorted(my_array)
print(temp)
```

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

In [94]:
```
arr=[0,2,1,2,0]
temp=[0]*3
for i in arr:
    temp[i]+=1
print(arr)
arr[0:temp[0]]=[0]*(temp[0])
arr[temp[0]:temp[0]+temp[1]]=[1]*temp[1]
arr[temp[0]+temp[1]:]=[2]*temp[2]
print(arr)
```

```
[0, 2, 1, 2, 0]
[0, 0, 1, 2, 2]
```

In [69]:
```
arr= [9, 4, -2, -1, 5, 0, -5, -3, 2]
pos=[]
neg=[]
for i in arr:
    if(i<0):
        neg.append(i)
    else:
        pos.append(i)

print(neg)
print(pos)
for i in range(min(len(pos),len(neg))):
```

```
    arr[i*2]=pos[i]
    arr[i*2+1]=neg[i]
    print(f"{i} iteration arr = {arr}")

if(len(pos)>len(neg)):
    arr[len(neg)*2:] = pos[len(neg):]
else:
    arr[len(pos)*2:] = neg[len(pos):]

print(arr)
```

```
[-2, -1, -5, -3]
[9, 4, 5, 0, 2]
0 iteration arr = [9, -2, -2, -1, 5, 0, -5, -3, 2]
1 iteration arr = [9, -2, 4, -1, 5, 0, -5, -3, 2]
2 iteration arr = [9, -2, 4, -1, 5, -5, -5, -3, 2]
3 iteration arr = [9, -2, 4, -1, 5, -5, 0, -3, 2]
[9, -2, 4, -1, 5, -5, 0, -3, 2]
```

In [70]:
```
def rearrange_alternate(arr):
    # Separate positive and negative numbers
    pos = [x for x in arr if x >= 0]  # 0 is considered positive
    neg = [x for x in arr if x < 0]

    result = []
    i, j = 0, 0

    # Add elements alternately from positive and negative lists
    while i < len(pos) and j < len(neg):
        result.append(pos[i])
        result.append(neg[j])
        i += 1
        j += 1

    # Append remaining positive or negative elements
    result.extend(pos[i:])  # Append remaining positives if any
    result.extend(neg[j:])  # Append remaining negatives if any

    return result

# Example usage:
arr = [9, 4, -2, -1, 5, 0, -5, -3, 2]
output = rearrange_alternate(arr)
print(output)
```

```
[9, -2, 4, -1, 5, -5, 0, -3, 2]
```

In [72]:
```
arr = [9, 4, -2, -1, 5, 0, -5, -3, 2]
pos = [x for x in arr if x >= 0]  #Fastest method to filter elemts from an l
print(pos)
```

```
[9, 4, 5, 0, 2]
```

`extend()`

- **Usage**: `list1.extend(iterable)`

- **Modifies the Original List**: `extend()` modifies the list in place by adding elements from the iterable to the end of the list.
- **Does Not Return a New List**: It returns `None`.
- **Efficiency**: `extend()` is generally more efficient when you need to add elements to an existing list especially when working with large lists or iterables.

Example:

```
list1 = [1, 2, 3]
list2 = [4, 5, 6]
list1.extend(list2)
print(list1)  # Output: [1, 2, 3, 4, 5, 6]
```

`+` Operator

- **Usage**: `list1 + list2`
- **Creates a New List**: The `+` operator creates a new list that is the concatenation of `list1` and `list2`.
- **Returns a New List**: The result is a new list with elements from both lists.
- **Efficiency**: Using `+` results in the creation of a new list, which may be less efficient compared to `extend()` when working with large lists.

Example:

```
list1 = [1, 2, 3]
list2 = [4, 5, 6]
result = list1 + list2
print(result)  # Output: [1, 2, 3, 4, 5, 6]
```

Key Differences:

1. **Modification vs. New List**:
   - `extend()` modifies the original list.
   - `+` creates a new list without modifying the original lists.
2. **Return Value**:
   - `extend()` returns `None`.
   - `+` returns a new list that is the result of the concatenation.
3. **Use Cases**:
   - Use `extend()` when you want to add elements to an existing list and modify it in place.
   - Use `+` when you need to create a new list that combines elements from multiple lists.

Summary

- `extend()` is used when you want to modify the original list by appending elements from another iterable.
- `+` is used to concatenate lists and create a new list, leaving the original lists unchanged.

Choosing between them depends on whether you need to modify an existing list or create a new one.

In [75]:
```
# Convert List to array using the arrya module

import array

# Example list
my_list = [1, 2, 3, 4, 5]
print(my_list,type(my_list))

# Convert list to array
# 'i' is the type code for integers. You can use other type codes for differ
my_array = array.array('i', my_list)

print(my_array)  # Output: array('i', [1, 2, 3, 4, 5])
```

```
[1, 2, 3, 4, 5] <class 'list'>
array('i', [1, 2, 3, 4, 5])
```

In [81]:
```
arr = [9, 8, 7, 6, 4, 2, 1, 3]
temp[0]=arr[-1]
temp[1:]=arr[0:len(arr)-1]
print(temp)
arr = array.array('i', temp)
print(arr)
```

```
[3, 9, 8, 7, 6, 4, 2, 1]
array('i', [3, 9, 8, 7, 6, 4, 2, 1])
```

In [ ]:
```
class Solution:
    import array
    def rotate(self, arr):
        if len(arr)==0:
            return arr
        last=arr.pop()
        arr.insert(0,last)
```