(https://ratzraturi.blogspot.com/)

RATZ_CODE (HTTPS://RATZRATURI.BLOGSPOT.COM/)

August 28, 2021 (https://ratzraturi.blogspot.com/2021/08/usrbinenv-python-coding-utf-8.html)

PYTHON BASICS

Variables

In [160]:
```
a=8
b="abhi"
c=9.0
```

In [161]:
```
list=[a,b,c]
for i in list:
    print(type(i))
```

```
<class 'int'>
<class 'str'>
<class 'float'>
```

Boolean

In [162]:
```
a=10
b=5
c=a>b
c
```

Out[162]:
```
True
```

In [163]:
```
#Adding two numbers
# input() taken input as a string and has to be converted into integer (typecast) if required
a=int(input("Enter a number: "))
b=int(input("Enter a second number: "))
print("Sum of the two numbers is : ",a+b)
```

```
Enter a number: 12
Enter a second number: 23
Sum of the two numbers is :  35
```

In [164]:
```
# Find the remainder when dividing
a=10
b=6
print("remainder is",a%b)
```

```
remainder is 4
```

In [165]:
```
# Comparison Operator will result into boolean
a=90
b=101
a>b
```

Out[165]:
```
False
```

In [166]:
```
# Find average of two numbers
a=int(input("Enter a number: "))
b=int(input("Enter a second number: "))
c=(a+b)/2
print("average of two numbers is : ", c)
```

```
Enter a number: 23
Enter a second number: 12
average of two numbers is :  17.5
```

In [167]:
```
# Square of a number aftre user input
a=int(input("Enter a number: "))
c=a*a
print("Square of two numbers is : ", c)
```

```
Enter a number: 567
Square of two numbers is :  321489
```

Chapter 3 - Strings

In [168]:
```
a=34
b="harry"
```

In [169]:
```
print(a,b)
```

```
34 harry
```

In [170]:
```
print(type(a),type(b))
```

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

In [171]:
```
a='harry'
b="harry"
c='''harry'''
print(type(b))
print(type(b))
print(type(c))
```

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

In [172]:
```
#why we use single quotes, double quotes and triple quotes strings
# Difference between the three quotes
a='harry"s'
b="harry's"
c='''"harry"s and harry's"'''
d='''"harry"s and
potter's"'''
print(type(b))
print(type(b))
print(type(c))
print(a)
```

```
<class 'str'>
<class 'str'>
<class 'str'>
harry"s
```

In [173]:
```
print(b)
```

```
harry's
```

In [174]:
```
print(c)
```

```
"harry"s and harry's"
```

In [175]:
```
print(d)
```

```
"harry"s and
potter's"
```

String Slicing

In [176]:
```
name="Abhishek"
greeting="Good Morning "
c=greeting + name
print(c)
```

```
Good Morning Abhishek
```

In [177]:
```
#Behind the scene for strings it store in an array starting with index 0
print(c[0])
print(c[4])
print(c[5])
print(c[-1])
```

```
G

M
k
```

In [178]:
```
# 'str' object does not support item assignment
# c[0]="Z" will not work
```

Slicing

In [179]:
```
# last index is excluded and first index is included
name[0:3]
```

Out[179]:
```
'Abh'
```

In [180]:
```
name[1:4]
```

Out[180]:
```
'bhi'
```

In [181]:
```
# printing entire string
name[0:]
```

Out[181]:
```
'Abhishek'
```

In [182]:
```
# finding length of a string
print(len(name))
print(len(greeting))
```

```
8
13
```

In [183]:
```
# why we use negative index
# we use to count from last part of the string and if the string is really long nstead of
# counting from starting we can count from the end
name[-1]
```
Out[183]:
```
'k'
```

In [184]:
```
name[-8:]
```
Out[184]:
```
'Abhishek'
```

In [185]:
```
# Including -8 and excluding -1 index
name[-8:-1]
```
Out[185]:
```
'Abhishe'
```

Slicing with skip value

In [186]:
```
name= "ABHISHEK"
name[0:9]
```
Out[186]:
```
'ABHISHEK'
```

In [187]:
```
name[:9]
```
Out[187]:
```
'ABHISHEK'
```

In [188]:
```
name[0:]
```
Out[188]:
```
'ABHISHEK'
```

In [189]:
```
# 3 argument is the skip value i.e . skip every 2 and print
name[0:9:2]
```
Out[189]:
```
'AHSE'
```

```
In [190]: name[0::2]
```

```
Out[190]: 'AHSE'
```

String functions

```
In [191]: story="once upon a time there was a boy named abhishek"
          story
```

```
Out[191]: 'once upon a time there was a boy named abhishek'
```

```
In [192]: # String Functions
          print(len(story))
```

```
47
```

```
In [193]: print(story.endswith("abhishek"))
```

```
True
```

```
In [194]: print(story.endswith("raturi"))
```

```
False
```

```
In [195]: print(story.count("a"))
```

```
5
```

```
In [196]: print(story.count("abhishek"))
```

```
1
```

```
In [197]: print(story.count("abhi"))
```

```
1
```

In [198]:
```
# First letter will be capitalized
print(story.capitalize())
```
```
Once upon a time there was a boy named abhishek
```

In [199]:
```
print(story.find("abhi"))
```
```
39
```

In [200]:
```
print(story.find("Abhi"))
```
```
-1
```

In [201]:
```
# Finds only the first occurence and returns its ondex
print(story.find("once"))
```
```
0
```

In [202]:
```
# Replaces all the occurences
print(story.replace("abhishek","harry"))
```
```
once upon a time there was a boy named harry
```

Escape sequence characters

In [203]:
```
story="Harry styles. He is a singer"
print(story)
```
```
Harry styles. He is a singer
```

In [204]:
```
story="Harry styles. \n He \t is a singer"
print(story)
```
```
Harry styles.
 He      is a singer
```

In [205]:
```
story="Harry styles. He is a \\singer"
print(story)
```

```
Harry styles. He is a \singer
```

Practise set with Strings

In [206]:
```
# Problem 1
user=input("Enter your name : ")
print("Good Morning, ",user)
print("Good Morning, "+user)
```

```
Enter your name : abhishek
Good Morning,  abhishek
Good Morning, abhishek
```

In [207]:
```
# Problem 2 - Letter template
letter='''Dear <|Name|>,
You are selected!

Date: <|Date|>
'''
name=input("Enter your name \n: ")
date=input("Enter date :\n ")
letter.replace("<|Name|>",name)
letter.replace("<|Date|>",date)
```

```
Enter your name
: abhishek
Enter date :
 2 nov 2021
```

Out[207]:
```
'Dear <|Name|>,\nYou are selected!\n\nDate: 2 nov 2021\n'
```

In [208]:
```python
# it did not change because it does not replaces in the actual string
print(letter)
```

```
Dear <|Name|>,
You are selected!

Date: <|Date|>
```

In [209]:
```python
# you have to use letter=letter.replace
# Problem 2 - Letter template
letter='''Dear <|Name|>,
You are selected!

Date: <|Date|>
'''
name=input("Enter your name \n: ")
date=input("Enter date :\n ")
letter=letter.replace("<|Name|>",name)
letter=letter.replace("<|Date|>",date)
print("\n\n\n")
print(letter)
```

```
Enter your name
: abhishek
Enter date :
 2 Nov 2021




Dear abhishek,
You are selected!

Date: 2 Nov 2021
```

In [210]:
```
#Problem 3 - Detect double spaces in a string
str="This is a string with double  spaces"
c=str.find("  ")
if c==-1:
    print("Double spaces not present")
else:
    print("Double spaces are present at index : ",c)
```

```
Double spaces are present at index :  28
```

In [211]:
```
#Problem 4 - Replace double spaces in the above string with single spaces
str="This is a string with double  spaces"
print(str)
str=str.replace("  "," ")
print(str)
```

```
This is a string with double  spaces
This is a string with double spaces
```

In [212]:
```
#Problem 5 - To format the below string using escape sequence characters
letter1="Dear Harry, This python course is nice.Thanks!"
letter2="Dear Harry,\n\nThis python course is nice.\n\nThanks!"
print(letter1)
print("\n\n\n")
print(letter2)
```

```
Dear Harry, This python course is nice.Thanks!




Dear Harry,

This python course is nice.

Thanks!
```

Chapter 4 - List and Tuples

List

In [213]:
```
#Lists
# List stores list of values of any data types
a=[1,2,3,4,57,6]
print(a)
```

```
[1, 2, 3, 4, 57, 6]
```

In [214]:
```
a[4]
```

Out[214]:
```
57
```

In [215]:
```
a[0]=98
```

In [216]:
```
print(a)
```

```
[98, 2, 3, 4, 57, 6]
```

In [217]:
```
a=[1,"harry",9.8]
print(a)
```

```
[1, 'harry', 9.8]
```

List Slicing

In [218]:
```
print(a[0:])
```

```
[1, 'harry', 9.8]
```

In [219]:
```
print(a[0:2])
```

```
[1, 'harry']
```

In [220]:
```
print(a[-3:])
```

```
[1, 'harry', 9.8]
```

List Methods

In [221]:
```
l1= [1,5,23,7,8,3,5,0,6]
```

In [222]:
```
li_sorted=l1.sort()  # will not work
print(li_sorted)
```

```
None
```

In [223]:
```
l1.sort()
print(l1)
```

```
[0, 1, 3, 5, 5, 6, 7, 8, 23]
```

In [224]:
```
#append list - adds at the end of the list

l1.append(7)
print(l1)
```

```
[0, 1, 3, 5, 5, 6, 7, 8, 23, 7]
```

In [225]:
```
# List insert - insert at index
l1.insert(0,100) # insert 100 at 0 th index
print(l1)
```

```
[100, 0, 1, 3, 5, 5, 6, 7, 8, 23, 7]
```

In [226]:
```
# pop
l1.pop(2) # removes element from 2 nd index
print(l1)
```

```
[100, 0, 3, 5, 5, 6, 7, 8, 23, 7]
```

In [227]:
```
#remove
l1.remove(100) # remove the element
print(l1)
```

```
[0, 3, 5, 5, 6, 7, 8, 23, 7]
```

In [228]:
```
#reverse
l1.reverse() # reverse the list
print(l1)
```

```
[7, 23, 8, 7, 6, 5, 5, 3, 0]
```

Tuples

In [229]:
```
t=(1,7,3,8)
print(t[3])
```

```
8
```

In [230]:
```
# Cannot update tuples, is immutable
# mutation means to change
# Once tuple is defined, we cannot change it
# t[0]=34 # will not work
```

In [231]:
```
a=() # empty tiple
b=(1) # wrong way to create a tuple with single element
c=(1,) # tuple with one element needs a comma, not needed when using jupyter but will be needed when using idle
d=(5,3,9,9) # tuple with multiple element
print(a)
print(b)
print(c)
print(d)
```

```
()
1
(1,)
(5, 3, 9, 9)
```

Tuples Method

In [232]:
```
t1=(1,5,1,8,1,9,1,10,53,1)
t1.count(1) # count the occurence of element 1 in the tuple
```

Out[232]:
```
5
```

In [233]:
```
t1.index(1) # returns the index of the element in the tuple, the first occurence
```

Out[233]:
```
0
```

Lists and Tuples - Practise

In [234]:
```
# Q1 - store 3 fruits in a list entered by user
fruits=[]
for i in range(3):
    a=input("Enter fruit name : ")
    fruits.append(a)
print(fruits)
```

```
Enter fruit name : mango
Enter fruit name : banana
Enter fruit name : apple
['mango', 'banana', 'apple']
```

In [235]:
```
# Q2 - Display marks of 5 students in a sorted manner
marks=[]
for i in range(5):
    a=int(input("Enter marks for student: "))
    marks.append(a)

marks.sort()
print("Marks sorted are : ",marks)
```

```
Enter marks for student: 12
Enter marks for student: 67
Enter marks for student: 45
Enter marks for student: 56
Enter marks for student: 65
Marks sorted are :  [12, 45, 56, 65, 67]
```

In [236]:
```
#Q3 - add a list of 4 numbers
a=[4,7,10,9]
sum=0
for i in range(4):
    sum=sum + a[i]
print("sum is : ",sum)
```

```
sum is :  30
```

In [237]:
```
# Q3 - find the number of zeros in the tuple
z=(7,0,8,0,0,9)
count=0
r=len(z)
for i in range(r):
    if z[i]==0:
        count=count+1
print("number of zeros in the tuple are : ",count)
```

```
number of zeros in the tuple are :  3
```

In [238]:
```
# Q3 - find the number of zeros in the tuple using function
z=(7,0,8,0,0,9)
print(z.count(0))
```

```
3
```

Chapter 5 - Dictionary and Sets

In [239]:
```
#Dictionary - collection of key value pairs

myDict={"Harry":"Styles",
        "abhi":"raturi",
        "ishan":"khan",
        "kate":"winslet",
        "marks":[34,45,76],
        "anotherDict":{"marks1":[45,87,90]},
        "a":(1,5,2)
       }
print(myDict)
```

```
{'Harry': 'Styles', 'abhi': 'raturi', 'ishan': 'khan', 'kate': 'winslet', 'marks': [34, 45, 76], 'ano
therDict': {'marks1': [45, 87, 90]}, 'a': (1, 5, 2)}
```

In [240]:
```
# printing corresponding value
print(myDict['abhi'])
print(myDict['marks'])
print(myDict['anotherDict'])
print(myDict['anotherDict']['marks1'])
print(myDict['a'])
```

```
raturi
[34, 45, 76]
{'marks1': [45, 87, 90]}
[45, 87, 90]
(1, 5, 2)
```

Properties of Dictionaries

In [241]:
```
# unordered
# mutable
# indexed
# cannot contain duplicate keys
```

Dictionary Methods

In [242]:
```
myDict={"Harry":"Styles",
        "abhi":"raturi",
        "ishan":"khan",
        "kate":"winslet",
        "marks":[34,45,76],
        "anotherDict":{"marks1":[45,87,90]},
        "a":(1,5,2)
       }
```

In [243]:
```
myDict.keys() # prints all the keys in the dicitinary
```

Out[243]:
```
dict_keys(['Harry', 'abhi', 'ishan', 'kate', 'marks', 'anotherDict', 'a'])
```

In [244]:
```
myDict.values() # prints all the values in the dicitinary
```

Out[244]:
```
dict_values(['Styles', 'raturi', 'khan', 'winslet', [34, 45, 76], {'marks1': [45, 87, 90]}, (1, 5,
2)])
```

In [245]:
```
myDict.items() # prints all the contents in the dicitinary
```

Out[245]:
```
dict_items([('Harry', 'Styles'), ('abhi', 'raturi'), ('ishan', 'khan'), ('kate', 'winslet'), ('mark
s', [34, 45, 76]), ('anotherDict', {'marks1': [45, 87, 90]}), ('a', (1, 5, 2))])
```

In [246]:
```
updateDict={"Lovish":"Friend"
}
myDict.update(updateDict)  # updates the dictionary with the key value pair at last
myDict
```

Out[246]:
```
{'Harry': 'Styles',
 'abhi': 'raturi',
 'ishan': 'khan',
 'kate': 'winslet',
 'marks': [34, 45, 76],
 'anotherDict': {'marks1': [45, 87, 90]},
 'a': (1, 5, 2),
 'Lovish': 'Friend'}
```

In [247]:
```
updateDict={"Harry":"Potter"
}
myDict.update(updateDict)  # updates the dictionary with the layest value of "Harry" as
                           #we can only have a unique key and therefore only one value corresponng to
that key
myDict
```

Out[247]:
```
{'Harry': 'Potter',
 'abhi': 'raturi',
 'ishan': 'khan',
 'kate': 'winslet',
 'marks': [34, 45, 76],
 'anotherDict': {'marks1': [45, 87, 90]},
 'a': (1, 5, 2),
 'Lovish': 'Friend'}
```

In [248]:
```
# Why we use get method instead of myDict["Harry"]

#print(myDict["Harry2"])  # will throw error because key is not present
myDict.get("Harry2")  # will not throw error even when key is not present and therefore it is recomme
nded to use get method
```

Sets

In [249]:
```
# Sets does not contain repetitive items
# Set is a collection of unique elements (no repetition of items)
a={1,6,3,6}
print(a)
print(type(a))
```

```
{1, 3, 6}
<class 'set'>
```

In [250]:
```
b={}  # this will create empty dictionary and not an empty set
print(type(b))
```

```
<class 'dict'>
```

In [251]:
```
# creating empty set
c=set()
print(type(c))
```

```
<class 'set'>
```

methods in sets

In [252]:
```
c=set()
```

In [253]:
```
c.add(4) # adding elements to set
c.add(5)
c.add(8)
print(c)
```

```
{8, 4, 5}
```

In [254]:
```
# creating a set
s={1,5,5,9}
print(s)
```

```
{1, 5, 9}
```

In [255]:
```
#cannot add list inside a set
#c.add([5,6]) # will not work,cannot add list inside a set
c.add((10,12)) # you can add a tuple inside a set
print(c)
#c.add({"h":"wer"}) # will not work,cannot add dictionary inside a set
```
```
{8, 4, 5, (10, 12)}
```

In [256]:
```
print(len(c)) # returns length of set
```
```
4
```

In [257]:
```
c.remove(5) # removes the element 5 from the set
print(c)
```
```
{8, 4, (10, 12)}
```

In [258]:
```
c.pop() # removes a random value from the set
print(c)
```
```
{4, (10, 12)}
```

In [259]:
```
d=set()
d.add(4)
d.add(10)
d.add(5)
d.add(9)
print(d)
d.union() # union of set
```
```
{9, 10, 4, 5}
```

Out[259]:
```
{4, 5, 9, 10}
```

In [260]:
```
d.clear() # removes all the  elements of the set
print(d)
```
```
set()
```

Properties of Sets

In [261]:
```
# Sets are unoredered i.e. cannot say return first or third selement and so on
# sets are indexed i.e. cannot access the elements using index
# cannot change items in a set
# cannot contain duplicate values
```

Sets - Practice Problems ¶

In [262]:
```
# Q1 - crate a dictionary and when user search for a word he/she will get the meaning of th word
dict={
      "dabba":"box",
      "pankha":"fan",
      "vastu":"item"
     }
print("Your options for entering a word are :",dict.keys())
word=input("Enter a hindi word : ")
print("Meaning of hindi word entered is : ",dict.get(word))
```

```
Your options for entering a word are : dict_keys(['dabba', 'pankha', 'vastu'])
Enter a hindi word : pankha
Meaning of hindi word entered is :  fan
```

In [263]:
```
# Q2 - input 3 numbers from user and display unique numbers
a=set()
for i in range(3):
    numbers=int(input("Enter number"))
    a.add(numbers)
print("Unique numbers are : ", a)
```

```
Enter number23
Enter number34
Enter number54
Unique numbers are :  {34, 54, 23}
```

In [264]:
```
# Q3 - find the length of the set
s=set()
s.add(20)
s.add(20.0)
s.add("20")
print(len(s))
```

```
2
```

In [265]:
```
# Q4 - What is the type of set?
s={}  # its a dcitionary
print(type(s))
```

```
<class 'dict'>
```

In [266]:
```
# Q5 - create a dictionary for favourite languages
favLang={}
a=input("enter your favourite language Shubham : ")
b=input("enter your favourite language Abhi : ")
c=input("enter your favourite language sonali : ")
d=input("enter your favourite language anil : ")
favLang['Shubham']=a
favLang['Abhi']=b
favLang['sonali']=c
favLang['anil']=d
print(favLang)
```

```
enter your favourite language Shubham : java
enter your favourite language Abhi : c++
enter your favourite language sonali : python
enter your favourite language anil : python
{'Shubham': 'java', 'Abhi': 'c++', 'sonali': 'python', 'anil': 'python'}
```

Chapter 6 - Conditional Expression

In [267]:
```python
a=3
if a>15:
    print("a is greater than 15")
elif a>17:
    print("a is greater than 17")
else:
    print("a is less than or equal to 3")
```

```
a is less than or equal to 3
```

In [268]:
```python
# Important points

# can have any number of elif
# last else executes when all the elif conditions fail
# else is optional
```

If else elif - practice Problems

In [270]:
```
# Q1 - enter 4 numbers from a user and find the greatest

num1=int(input("Enter number 1: "))
num2=int(input("Enter number 2: "))
num3=int(input("Enter number 3: "))
num4=int(input("Enter number 3: "))
if num1>num4:
    f1=num1
else:
    f1=num4
if num2>num3:
    f2=num2
else:
    f2=num3
if f1>f2:
    print("Greatest number is : ",f1)
else:
    print("Greatest number is : ",f2)
```

```
Enter number 1: 34
Enter number 2: 567
Enter number 3: 54
Enter number 3: 67
Greatest number is :  567
```

In [271]:
```
#Q2 - Enter three subject marks and print pass if total 40 % and 33 % in each subject
sub1=int(input("Enter marks for subject 1 : "))
sub2=int(input("Enter marks for subject 2 : "))
sub3=int(input("Enter marks for subject 3 : "))
if (sub1<33 or sub2<33 or sub3<33):
    print("Student failed")
if (sub1+sub2+sub3)/3 < 40:
    print("You failed because your total percentage is less than 40")
else:
    print("Congrats, you passed")
```

```
Enter marks for subject 1 : 34
Enter marks for subject 2 : 43
Enter marks for subject 3 : 45
Congrats, you passed
```

In [272]:
```
# Q3 - Write a program to detect a spam conatining the message
text=input("Enter a text : ")
spam=False
if ("make a lot of money" in text):
    spam=True
elif ("buy now" in text):
    spam=True
elif ("watch this" in text):
    spam=True
else:
    spam=False
if (spam):
    print("This text is spam")
else:
    print("This text is not spam")
```

```
Enter a text : watch this
This text is spam
```

In [273]:
```
# Q4 - Find whether a given username contains less than 10 characters or not

name=input("Enter your name : \n")
if len(name)<10:
    print("Your name has less than 10 characters")
elif len(name)==10:
    print("Your name has 10 characters")
else:
    print("Your name has more than 10 characters")
```

```
Enter your name :
abhi
Your name has less than 10 characters
```

In [274]:
```
# Q5 - If the name entered is present in a list
names=["abhishek","rahul","sarthak","shipra","shilpi"]
name=input("Enter your name : \n")
if name in names:
    print("Your name is present in the list")
else:
    print("Your name is not present in the list")
```

```
Enter your name :
abhishek
Your name is present in the list
```

In [275]:
```
# Q6 - Calculate grades accoring to the marks
marks=int(input("Enter marks : \n"))
if marks>90:
    grade="Excellent"
elif marks>=70:
    grade="A"
elif marks>=60:
    grade="B"
elif marks>=50:
    grade="C"
elif marks>=40:
    grade="C"
else:
    grade="F"
print("Your grade is : ", grade)
```

```
Enter marks :
44
Your grade is :  C
```

Chapter 7 - Loops

while loop

In [3]:
```
l=[2,7,45,68,698]
i=0
while i<len(l):
    print(l[i])
    i=i+1
```

```
2
7
45
68
698
```

for loop

In [2]:
```
l=[2,7,45,68,698]
for item in l:
    print(item)
```

```
2
7
45
68
698
```

In [9]:
```
#range function in for loop
for i in range(4):
    print(i)
```

```
0
1
2
3
```

In [10]:
```
for i in range(1,4):
    print(i)
```

```
1
2
3
```

In [13]:
```
# step size in range function
for i in range(1,6,2):
    print(i)
```

```
1
3
5
```

In [21]:
```
# for loop with else
for i in range(5):
    print(i)
else:
    print("This is inside else of for")
```

```
0
1
2
3
4
This is inside else of for
```

In [25]:
```
# break statement
for i in range(10):
    print(i)
    if i==3:
        break
```

```
0
1
2
3
```

In [26]:
```
# else will print only when for loop executes fully,in this case it will
# not as the break statement exits the for and else because else is also part of for loop
for i in range(5):
    print(i)
    if i==3:
        break
else:
    print("This is inside else of for")
```

```
0
1
2
3
```

In [27]:
```
# continue statement, skips the iteration
for i in range(5):
    if i==3:
        continue
    print(i)
```

```
0
1
2
4
```

In [28]:
```
# pass statement , it is a null staement and means do nothing
# when you do not know what to do with the condition and want to comeback later to do it you can ment
ion pass
# we cannot leave the statement empty as it will throw error  so we put a pass statement to avoid the
error
if i==3:
    pass
while i<2:
    pass
print("Comeback to chack these above condition and work on it later")
```

```
Comeback to chack these above condition and work on it later
```

In [29]:
```
# pass statement with def
def a():
    pass
```

Loops - Practice Problems

In [32]:
```
#Q1 - mutliplication table of a given number
num=int(input("Enter a number : "))
print("Multiplication table of the entered number is : \n")
for i in range(1,11):
    print(str(num)+ "X" + str(i) + "=" + str(i*num))
```

```
Enter a number : 3
Multiplication table of the entered number is :

3X1=3
3X2=6
3X3=9
3X4=12
3X5=15
3X6=18
3X7=21
3X8=24
3X9=27
3X10=30
```

In [33]:
```
# using fstring
num=int(input("Enter a number : "))
print("Multiplication table of the entered number is : \n")
for i in range(1,11):
    print(f"{num} X {i}={num*i}")
```

```
Enter a number : 5
Multiplication table of the entered number is :

5 X 1=5
5 X 2=10
5 X 3=15
5 X 4=20
5 X 5=25
5 X 6=30
5 X 7=35
5 X 8=40
5 X 9=45
5 X 10=50
```

In [38]:
```
# Q2 - From the list of names, print a hello message to only the ones whose name starts with S
names=["Shivam","abhishek","shubham","zayn","cat"]
for i in names:
    if i.startswith("S") or i.startswith("s"):
        print("Hello : ",i)
```

```
Hello :  Shivam
Hello :  shubham
```

In [1]:
```
# Q3 - Number prime or not
num= int(input("Enter a number : "))
prime=True
for i in range(2,num):
    if num%i==0:
        prime=False
    break
if prime:
    print("Number is prime ")
else:
    print("Number is not prime ")
```

```
Enter a number : 4
Number is not prime
```

In [2]:
```
# Q4 - sum of first n natural numbers
num=int(input("Enter a number : "))
sum=0
num=num+1
for i in range(1,num):
    sum=sum+i
print(sum)
```

```
Enter a number : 4
10
```

In [2]:
```
# Q4 - sum of n natural numbers using while loop
num=int(input("Enter a number : "))
sum=0
i=1
while i<=num:
    sum=sum+i
    i=i+1
print(sum)
```

```
Enter a number : 4
10
```

In [7]:
```
# Q5 - Factorial of a number
num=int(input("Enter a number : "))
fact=1
for i in range(1,num+1):
    fact=fact*i
print(fact)
```

```
Enter a number : 0
1
```

In [8]:
```
# Q6 - Print pattern
for i in range(4):
    print("*" * (i))
```

```

*
**
***
```

In [11]:
```
# without using end
print("abhishek")
print("raturi")

# using end
print("abhishek",end="")
print("raturi",end="")
```

```
abhishek
raturi
abhishekraturi
```

In [3]:
```
#Q7 - multiplication table in reverse
num=int(input("Enter a number : "))
print("Multiplication table of the entered number is : \n")
for i in reversed(range(11)):
    print(f"{num} X {i}={num*i}")
```

```
Enter a number : 3
Multiplication table of the entered number is :

3 X 10=30
3 X 9=27
3 X 8=24
3 X 7=21
3 X 6=18
3 X 5=15
3 X 4=12
3 X 3=9
3 X 2=6
3 X 1=3
3 X 0=0
```

Chapter 8 - Functions and Recursions

In [4]:
```
# Functions - Group of statements to perform a specific task
```

In [5]:
```
# defining a funct:
def funct():
    print("This is a function")
```

In [6]:
```
# function call
funct()
```

```
This is a function
```

Types of functions

In [7]:
```
#1 Built in functions: sum(), len(), range()
#2 User defined functions
```

functions with arguments

In [10]:
```
def sum1(num1,num2):
    return(num1+num2)
a=sum1(4,3)
print(f"Sum of two numbers is {a}")
```

```
Sum of two numbers is 7
```

default parameter value

In [14]:
```
# without using default parameter value, this will not work as argument is not passed
def name(name1):
    return(f"Greetings, {name1}")
a=name()
print(a)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-14-cb49cfff965d> in <module>
      2 def name(name1):
      3     return(f"Greetings, {name1}")
----> 4 a=name()
      5 print(a)

TypeError: name() missing 1 required positional argument: 'name1'
```

In [13]:
```
# using default parameter value, this will  work as default argument is also passed
def name(name1="Stranger"):
    return(f"Greetings, {name1}")
a=name()
print(a)
```

```
Greetings, Stranger
```

Recursion

In [16]:
```
# recusrion is a function that calls itself
num=int(input("Enter a number : "))
def fact(num):
    if num==1 or num==0:
        return 1
    return(num*fact(num-1))
fact(num)
```

```
Enter a number : 5
```

Out[16]:
```
120
```

functions and recusrions - Practice Problems

In [21]:
```
#Q1 - Greatest number using function
def greatest(num1,num2,num3):
    if num1>num2:
        a=num1
    else:
        a=num2
    if a>num3:
        return(a)
    else:
        return(num3)
num1=int(input("Enter number 1 : \n"))
num2=int(input("Enter number 2 : \n"))
num3=int(input("Enter number 3 : \n"))
greater_num=greatest(num1,num2,num3)
print(f"Greatest of the three numbers is : {greater_num}")
```

```
Enter number 1 :
34
Enter number 2 :
64
Enter number 3 :
35
Greatest of the three numbers is : 64
```

In [30]:
```
# Q3 - To remove a given word and strip it at the same time


def remove_and_strip(string,word):
    new_str=string.replace(word," ")
    return(new_str.strip()) # removes the spaces from the starting
this="    Abhi is smarty.   "
str1=remove_and_strip(this,"Abhi")
print(str1)
```

```
is smarty.
```

Chapter 9 - File I/O

Types of files:

1. Text files (.txt, .c)

2. Binary files (.jpg, .dat)

Functions performed on a file:

Opening a file

In [3]:
```
f=open("sample.txt","r") # by deafult mode is read i.e. r
data=f.read() # reads entire file
#data=f.read(5) # reads first 5 characters
print(data)
f.close()
```

```
This is a text file. Will be performing different file functions/operations on this file for learning
file I/O.
```

other methods to read a file

In [4]:
```
f=open("sample.txt")
text=f.readline()
print(text)
f.close()
```

```
This is a text file.
```

modes of opening a file

In [8]:
```
#  r - open for reading
#  w - open for writing
#  a - open for appending
#  + - open for updating
#  rb - will open the file in binary mode
#  rt = will open file ion text mode
```

writing to a file

In [ ]:
```
# this will override the file

#f=open("sample.txt","w")
#f.write("Adding new lines to the file")
#f.close()
```

In [10]:
```
# appening the file
f=open("sample.txt","a")
f.write("\nAdding new lines to the file")
f.close()
```

with statement

with statement automatically closes the file and does not require closing it manually

In [11]:
```
with open("sample.txt","r") as f:
    a=f.read()
print(a)
```

```
This is a text file.
We Will be performing different file functions/operations on this file for learning file I/O.
Hope this will help everybody.Adding new lines to the file
Adding new lines to the file
```

In [12]:
```
with open("sample.txt","a") as f:
    a=f.write("\n This line was added using with statement")
```

```
42
```

File I/O : Practice Problems

In [13]:
```
# Q1 - Read the text from the given file "poems.txt" and check if it contains the word "twinkle"

with open("twinkle.txt") as f:
    t=f.read()
    if "twinkle" in t:
        print("Twinkle word is present in the file")
    else:
        print("Twinkle word is not present in the file")
```

```
Twinkle word is present in the file
```

In [25]:
```
# Q2 - Generate multiplication table from 2 to 20 and and write it to different files


#for i in range(2,21):
#    with open(f"file_{i}.txt","w") as f:
#        for j in range(1,11):
#            f.write(f"{i}X{j}={i*j}\n")
#
```

In [28]:
```
# Q3 - Replace the word donkey with ######
#with open("Donkey.txt","r") as f:
#    content=f.read()
#content=content.replace("Donkey","######")
#with open("Donkey.txt","w") as f:
#    f.write(content)
```

In [33]:
```
# Q4 - Mining the log file and checking if python word exists and also give the line number
lineNumber=1
content=True
with open("Donkey.txt","r") as f:
    while content:
        content=f.readline()
        if 'python' in content.lower():
            print(f"Yes python is present in line number :  {lineNumber}")
        lineNumber+=1
```

```
Yes python is present in line number :  3
Yes python is present in line number :  12
```

In [37]:
```
# Q5 - Rename the file
#import os
#os.rename("Donkey.txt","renamed_by_python.txt")
```

Chapter 10 - Object Oriented Programming

In [6]:
```
# Solving a problem by creating objects is one of the most popular approaches in programming.
# Class is a blueprint for creating objects
# classes do not occupy any memory unless object is instantiated
# when the code becomes long and complex then classes become necessary to use
```

In [3]:
```
class RailwayForm:
    forType="Railway"
    def printData(self):
        print(f"Name is {self.name}\n")
        print(f"Train name is {self.train}\n")
rail=RailwayForm()    # object initializing
rail.name="Abhishek Raturi"
rail.train="Rajdhani Express"
rail.printData()
```

```
Name is Abhishek Raturi

Train name is Rajdhani Express
```

In [4]:
```
# Encapsulation : inside a game progrmmming,functions of player are inside the class
#"players" and functions of a remote are inside the class "remote"

#Abstraction: need not to understand how "isleftpressed" is working and how it is implemented
#player1=Player()
#remote1=Remote()
#if remote1.isleftpressed():
#    pass
```

In [5]:
```
# class : Employee
# Attribute : Name, Age, Salary
# methods: getSalary(),getAge()
```

Attributes and its types

1. Class attributes

In [10]:
```
class Employee:
    company="Google"
abhi=Employee()
rajni=Employee()
print(abhi.company)
print(rajni.company)
Employee.company="You Tube"    # Changing the class attribute changes the attribute in the class
print(Employee.company)
print(abhi.company)
print(rajni.company)
```

```
Google
Google
You Tube
You Tube
You Tube
```

2. Instance attribute

In [11]:
```
class Employee:
    company="Google"
abhi=Employee()
rajni=Employee()
print(abhi.company)
print(rajni.company)

abhi.company="Abhi company" # Changing the instance attribute changes the attribute only for that ins
tance and not the class
rajni.company="Rajni company"

print(Employee.company)
print(abhi.company)
print(rajni.company)
```

```
Google
Google
Google
Abhi company
Rajni company
```

In [ ]:
```
#First it searches for the instance attribute and if not found then only it takes the class attribute
# Instance attributes takes preference over class attributes
# Creating instance attribute does not change the class attributes.
#
```

what is self

In [19]:
```
class Employee:
    company="Google"
    def getSalary(self):
        print("Salary is not there")
abhi=Employee()
abhi.getSalary() # this is equivalent to Employee.getSalary(abhi),it passes a argument automatially a
nd so self is needed
```

```
Salary is not there
```

self is used to get both class and instance attribute

In [26]:
```
class Employee:
    company="Google"
    def getSalary(self):
        print(F"Salary is {self.salary}")    # self taking instance attribute
        print(F"Company is {self.company}")  # self taking class attribute as well
abhi=Employee()
abhi.salary=100000
abhi.getSalary() # this is equivalent to Employee.getSalary(abhi),it passes a argument automatially a
nd so self is needed
```

```
Salary is 100000
Company is Google
```

static method

In [ ]:

In [4]:
```
class Employee:
    company="Google"
    def getSalary(self):
        print(F"Salary is {self.salary}")    # self taking instance attribute
        print(F"Company is {self.company}")  # self taking class attribute as well
    @staticmethod
    def greet():
        print("This is a static method and does not require a self to be mentioned")
abhi=Employee()
abhi.salary=100000
abhi.getSalary() # this is equivalent to Employee.getSalary(abhi),it passes a argument automatially a
nd so self is needed
abhi.greet() # this is equivalent to Employee.greet(),it passes a argument automatially and so self i
s needed
```

```
Salary is 100000
Company is Google
This is a static method and does not require a self to be mentioned
```

constructor

init() is a special function which is first run as soon as an object is created. **init**() is also called constructor It takes self argument and can also take further arguments

In [7]:
```
class Employee:
    company="Google"
    def __init__(self): # Constructor
        print("Employee is created")
    def getSalary(self):
        print(F"Salary is {self.salary}")    # self taking instance attribute
        print(F"Company is {self.company}")  # self taking class attribute as well
    @staticmethod
    def greet():
        print("This is a static method and does not require a self to be mentioned")
abhi=Employee() # object was created and only the constructor function was run therefore prints only the statement under the constructor
```

```
Employee is created
```

In [10]:
```
class Employee:
    company="Google"
    def __init__(self,name,salary,company): # Constructor
        self.name=name
        self.salary=salary
        self.company=company
        print("Employee is created")
    def getDetails(self):
        print(F"Name is {self.name}")
        print(F"Salary is {self.salary}")    # self taking instance attribute
        print(F"Company is {self.company}")  # self taking class attribute as well
    @staticmethod
    def greet():
        print("This is a static method and does not require a self to be mentioned")
abhi=Employee("Abhishek Raturi",1000000,"NTS") # object was created,only the constructor function was run therefore prints only the constructor function runs
abhi.getDetails()
```

```
Employee is created
Name is Abhishek Raturi
Salary is 1000000
Company is NTS
```

Practice Problems - OOPs

Q1 - create class programmer for storing info of few programmers working in Microsoft

In [16]:
```
class Programmers:
    company="Microsoft" #Creating class attribute beacuse evry orogrammer will be in Microsoft as per
question
    def __init__(self,name,product):
        self.name=name
        self.product=product
    def getInfo(self):
        print(f"The name of the {self.company} programmer is {self.name} and the product is {self.pro
duct}")

abhi=Programmers("Abhishek Raturi","Python")
alka=Programmers("Alka Lamba","Git Hub")
abhi.getInfo()
alka.getInfo()
```

```
The name of the Microsoft programmer is Abhishek Raturi and the product is Python
The name of the Microsoft programmer is Alka Lamba and the product is Git Hub
```

Q2 - Create a class Calculator to find square, cube and square root of a number

In [22]:
```
class calculator:
    def __init__(self,number):
        self.number=number
    def square(self):
        print(f"Square of a number is {self.number*self.number}")
    def cube(self):
        print(f"Cube of a number is {self.number*self.number*self.number}")
    def squareroot(self):
        print(f"Square root of a number is {self.number**0.5}")
a=calculator(9)
a.square()
a.cube()
a.squareroot()
```

```
Square of a number is 81
Cube of a number is 729
Square root of a number is 3.0
```

Q3 - Create a class with class attribute a, create an object from it and set a directly using a=o.Does this change the class attribute?

In [30]:
```
class Name:
    company="Microsoft"
   # def __init__(self):
        #self.name=name
obj=Name()
obj.company="Vicky" # instance attribute set same as class attribute
print(Name.company)
print(obj.company)
print(Name.company) # class attribute remains same
```

```
Microsoft
Vicky
Microsoft
```

Q4 - Write a class Train which has method to book a ticket, set status (no of seats) and get fare information of trains running under indian railways

In [44]:
```
# class Train:
def __init__(self,seats,fare):
    self.seats=seats
    self.fare=fare

def bookTicket(self):
    if self.seats>0:
        print(f"Seat has been booked. Seat number is {self.seats}")
        self.seats=self.seats-1
    else:
        print("No seats are available, try in tatkaal")

def setStatus(self):
    print(f"Number of seats available are : {self.seats}")

def getfareInfo(self):
        print(f"Fare for the seats is {self.fare}")

abhi=Train(2,90)
abhi.bookTicket()
abhi.setStatus()
abhi.getfareInfo()
```

```
Seat has been booked. Seat number is 2
Number of seats available are : 1
Fare for the seats is 90
```

Q5 - Can you change the self parameter inside the class? Yes, canuse any name but not a best practice

In [42]:
```
class Name:
    def __init__(slf,name):
        slf.name=name
obj=Name("Harry")
print(obj.name)
```

```
Harry
```

Chapter 11 - Inheritance

Inheritance is a way of creating a class from an existing class, we can use the methods and attributes of a base class in our derived class and will override the base class function if the same function is there in derived class and so derived class function will come into the picture

In [59]:
```
#Syntax

class Employee:                                         # Base class
    company="Google"
    def showDetails(self):
        print("This is an employee")

class Programmer(Employee):                              # derived class
    language="python"
    def getLanguage(self):
        print("Language is {self.language}")
    def showDetails(self):
        print("This is an Programmer")
e=Employee()
e.showDetails()
p=Programmer()
p.showDetails()
p.company         # p object does not have its company variable and wuill take it from the base clas
s by inhereting from it
```

```
This is an employee
This is an Programmer
```

Out[59]:
```
'Google'
```

Types of Inheritance

1. Single Inheritance When child class inherits only a single parent class

2. Multiple Inheritance Child class inherits from multiple base classes

3. Multilevel Inheritance When a child class becomes a Parent for another child class

In [9]:
```
# Multiple inheritance

class Employee:
    company="Visa"
    ecode=120
class Freelancer:
    company="Fiver"
    level=2
    def upgrade(self):
        self.level=self.level+1
class Programmer(Employee,Freelancer):
    name="Rohit"
p=Programmer()
print(p.level)
p.upgrade()
print(p.level)
print(p.company)# it takes the value of the first class that is mentioned while inheriting i.e. class Programmer(Employee,Freelancer)
```

```
2
3
Visa
```

In [18]:
```
# Multilevel Inheritance

class Person:
    country="India"
    def takeBreath(self):
        print("I am breathing")

class Employee(Person):
    company="Honda"
    def getSalary(self):
        print("Salary is 1000")
    def takeBreath(self):
        print("I am an employee, I am luckily breathing")

class Programmer(Employee):
    company="Fiverrr"
    def getSalary(self):
        print("No salary to programmers")

p=Person()
print(p.takeBreath())

e=Employee()
print(e.takeBreath())
print(e.company)

pr=Programmer()
print(pr.takeBreath())
print(pr.getSalary())
```

```
I am breathing
None
I am an employee, I am luckily breathing
None
Honda
I am an employee, I am luckily breathing
None
No salary to programmers
None
```

super () method

super(): super method is used to access the methods of super class in the derived class

super().init()

In [25]:
```
# super method

class Abhi:
    country="India"
    def takeBreath(self):
        print("I am breathing")

class raturi(Person):
    company="Honda"
    def getSalary(self):
        print("Salary is 1000")
    def takeBreath(self):
        print("I am an employee, I am luckily breathing")

class Programmers(Employee):
    company="Fiverrr"

    def takeBreath(self):
        super().takeBreath()  # this will run super class method as well
        print("I am a programmer, I am breathing ++")

p=Abhi()
print(p.takeBreath())

e=raturi()
print(e.takeBreath())

pr=Programmers()
print(pr.takeBreath())
```

```
I am breathing
None
I am an employee, I am luckily breathing
None
I am an employee, I am luckily breathing
I am a programmer, I am breathing ++
None
```

class methods

class method is a method bound to the class and not the object of the class

@classmethod decorator is used to create a class method

In [28]:
```
# without using class methods

#class Emp:
#    company="Camel"
#    salary=100
#    locatio="Delhi"
#e=Emp()
#e.salary
```

Out[28]:
```
100
```

In [43]:
```
# without using class methods
class Emp:
    company="Camel"
    salary=100
    locatio="Delhi"
    def changeSal(self,sal):
        self.salary=sal     # this will only change the instance attribute and not the class attribut
e salary
e=Emp()
print(e.salary)
print(e.changeSal(1400))
print(e.salary)
print(Emp.salary)
```

```
100
None
1400
100
```

In [44]:
```
class Emp:
    company="Camel"
    salary=100
    locatio="Delhi"
#     def changeSal(self,sal):
#         self.__class__.salary=sal # this will change the class attribute salary

    @classmethod
    def changeSal(cls,sal):
        cls.salary=sal # this will change the class attribute salary

e=Emp()
print(e.salary)
print(e.changeSal(1400))
print(e.salary)
print(Emp.salary)
```

```
100
None
1400
1400
```

Decorator (getter and setter)

In [55]:
```
# decorator

class Ratz:
    company="Bharat Gas"
    salary=1000
    bonus=10
    @property  # property decorator makes the function as a property rather than a function.Also call
ed getter.
    def totalSalary(self):  # I want totalSalary to be varyong and cannot be fixed in the class
        return self.salary + self.bonus

    @totalSalary.setter
    def totalSalary(self,val):
        self.bonus = val - self.salary


e=Ratz()
print(e.totalSalary)
e.totalSalary=10000
print(e.salary)
print(e.bonus)
```

```
1010
1000
9000
```

Operator overloading

In [70]:
```
#Operator overloading with dunder methods**

# Overload + with __add__
# Overload - with __sub__
# Overload * with __mul__
```

In [71]:
```
class Number:
    def __init__(self,num):
        self.num=num

    def __add__(self,num2):
        print("lets add")
        return self.num + num2.num

    def __mul__(self,num2):
        print("lets multiply")
        return self.num * num2.num

n1=Number(4)
n2=Number(6)
print(n1+n2)   # this calls function __add__(self,num2)
print(n1*n2)   # this calls function __MUL__(self,num2)
```

```
lets add
10
lets multiply
24
```

Chapter 11 - Practice Problems

Q1 - Create a class to create a 2d vector and using it create another class to create a 3d vector

In [3]:
```
class c2dvec:
    def __init__(self,i,j):
        self.icap=i
        self.jcap=j
    def __str__(self):
        return f"{self.icap}i+{self.jcap}j"

class c3dvec(c2dvec):
    def __init__(self,i,j,k):
        super().__init__(i,j)
        self.kcap=k
    def __str__(self):
        return f"{self.icap}i+{self.jcap}j+{self.kcap}k"


v2d=c2dvec(1,3)
v3d=c3dvec(1,9,7)
print(v2d)
print(v3d)
```

```
1i+3j
1i+9j+7k
```

Q2 - Create a class pets from class animals and create class dogs from pets.Add a method bark to class Dog

In [4]:
```
class Animals:
    animalType="Mammal"


class pets(Animals):
    color="white"


class Dogs(pets):
    @staticmethod
    def bark():
        print("bowww.....bowww....")


ani=Animals()
pet=pets()
dog=Dogs()

print(dog.bark())
```

```
bowww.....bowww....
None
```

Q3 - Create a class Employee and add properties salary and increment.write a method salaryafterIncrement with a property decorator with a setter which changes the value of increment based on the salary

In [23]:
```
class Employe:
    salary=1000
    increment=1.5

    @property
    def salaryAfterIncrement(self):
        return self.salary * self.increment
        #return self.salary

    @salaryAfterIncrement.setter
    def salaryAfterIncrement(self,sai):
        self.increment= sai/self.salary
        #return self.increment


e=Employe()
print(f"salary is {e.salary}")
print(f"increment is {e.increment}")
print(f"salary after increment is {e.salaryAfterIncrement}")
e.salaryAfterIncrement=2000
print(f"increment after setting the new salary is {e.increment}")
```

```
salary is 1000
increment is 1.5
salary after increment is 1500.0
increment after setting the new salary is 2.0
```

Q4 - Writa a class vector representing vector of n dimension. Overload the + and * operator which calculates the sum and dot product of it

In [34]:
```
class vector:
    def __init__(self,vec):
        self.vec=vec

    def __str__(self):
        str1=""
        index=0
        for i in self.vec:
            str1+=f"{i}a{index} +"
            index+=1
        return str1[:-1]

    def __add__(self,vec2):
        newlist=[]
        for i in range(len(self.vec)):
            newlist.append(self.vec[i]+vec2.vec[i])
        return vector(newlist)

v1=vector([1,4])
v2=vector([1,6])
print(v1)
print(v2)
print(v1+v2)
```

```
1a0 +4a1
1a0 +6a1
2a0 +10a1
```

Project: Write a program that generates a random number and asks user to guess it and display the message accordingly if guessed less, more or exact and the number of guesses it to answered it correctly

In [49]:
```
import random
randNum=random.randint(1,10)
#print(randNum)
guess=1

userGuess=None
while userGuess!=randNum:
    userGuess=int(input("Enter your guess : "))
    if userGuess==randNum:
        print(f"You guessed it right and in {guess} attempt")
    else:
        if userGuess>randNum:
            print("you guessed it wrong, try a lesser number")
        else:
            print("you guessed it wrong, try a higher number")
    guess+=1
```

```
Enter your guess : 3
you guessed it wrong, try a higher number
Enter your guess : 4
you guessed it wrong, try a higher number
Enter your guess : 5
You guessed it right and in 3 attempt
```

Labels: python (https://ratzraturi.blogspot.com/search/label/python)

COMMENTS