In [4]:
```
def outer(): #2

    def inner(): #4
        x=10 #local variable. declared in the body of current function
        print(x)

    inner() #3
```

In [5]:
```
outer() #1
```

```
10
```

In [6]:
```
#local variables are either passed to a method/functions or defined inside it
def outer(): #2

    def inner(y): #4
        x=10 #local variable. declared in the body of current function
        print(x, y , sep="\n")

    inner("local y") #3
```

In [7]:
```
outer()
```

```
10
local y
```

In [ ]:
```
#local
#enclosing
#global
#built-in
```

In [9]:
```
def outer(): #2

    z="enclosing z"

    def inner(y): #4
        x=10 #local variable. declared in the body of current function
        print(x, y , sep="\n")
        print(z)
    inner("local y") #3


outer()
```

```
10
local y
enclosing z
```

In [10]:
```
name="global variable" #global

def outer(): #2

    z="enclosing z"

    def inner(y): #4
        x=10 #local variable. declared in the body of current function
        print(x, y , sep="\n")
        print(z)
        print(name)
    inner("local y") #3


outer()
```

```
10
local y
enclosing z
global variable
```

In [12]:
```
#local
#enclosing
#global
#built-in

name="global variable" #global
```

```
def outer(): #2

    z="enclosing z"

    def inner(y): #4

        name="local name"

        x=10 #local variable. declared in the body of current function
#         print(x, y , sep="\n")
#         print(z)
        print(name)
        print(xyz)
    inner("local y") #3


outer()
```

```
local name
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-12-23ae16e6aca6> in <module>
     22
     23
---> 24 outer()

<ipython-input-12-23ae16e6aca6> in outer()
     19         print(name)
     20         print(xyz)
---> 21     inner("local y") #3
     22
     23

<ipython-input-12-23ae16e6aca6> in inner(y)
     18 #         print(z)
     19         print(name)
---> 20         print(xyz)
     21     inner("local y") #3
     22

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

In [13]:
```
#suppose , I want to create a list of first 100 natural numbers


l1=[]

for x in range(1,101,1):
    l1.append(x)


print(l1)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 3
1, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59
, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87,
88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]
```

In [14]:
```
#expression, conditions and range
#type 1 expression range

#s1 is a set of all elements x such that x is a natural number
#s1= {  x | x E N  }



#l1 is a list of all values x such that x is in the range of numbers 1 to 100
l1=[   x for x in range(1,101,1)           ]
print(l1)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 3
1, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59
, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87,
88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100]
```

In [15]:
```
#alphabets
```

```
alphabets=[   chr(x) for x in range(65,91,1)  ]
print(alphabets)
```

```
['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W
', 'X', 'Y', 'Z']
```

In [16]:
```
name="harshit shukla"


demo=[  letter for letter in name  ]
print(demo)
```

```
['h', 'a', 'r', 's', 'h', 'i', 't', ' ', 's', 'h', 'u', 'k', 'l', 'a']
```

In [17]:
```
#type 2   expression range if condition part


#list of numbers divisible by 4 between 1 to 100(both inclusive)


div_4=[  x for x in range(1,101,1) if x%4==0 ]
print(div_4)
```

```
[4, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48, 52, 56, 60, 64, 68, 72, 76, 80, 84, 88, 92, 96, 100]
```

In [19]:
```
#list of all numbers divisible by 7 and greater than 20 between 1 and 50 (both inclusive)

div_7=[  x for x in range(1,51,1) if x%7==0 and x > 20 ]
print(div_7)
```

```
[21, 28, 35, 42, 49]
```

In [20]:
```
#list of vowels from a given name

vowels=['a','e','i','o','u']

name="harshit"
vowel_chars=[  letter for letter in name if letter in vowels ]
print(vowel_chars)
```

```
['a', 'i']
```

In [22]:
```
#list of vowels from a given name

vowels=['a','e','i','o','u']

vowel_chars=[  letter for letter in input("Enter a name: ") if letter in vowels ]
print(vowel_chars)
```

```
Enter a name: harshit
['a', 'i']
```

In [23]:
```
#type 3  expression  if -else block  range

#list of values "Yes" and "No" where yes corresponds to even numbers and no corresponds to odd numbers between
#1 and 20


ans=[  "Yes" if num%2==0 else "No" for num in range(1,21,1)   ]
print(ans)
```

```
['No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Y
es', 'No', 'Yes']
```

In [27]:
```
class Student:


    #constructor! #dunder methods
    def __init__(self,name, sid):
        self.s_name=name
        self.s_id=sid
```

```
s1=Student(sid=1,name="Harshit")
s2=Student(sid=2,name="John")
s3=Student(sid=3,name="Smith")
```

In [29]:
```
print(  vars(s1)  )
print(  vars(s2))
```

```
{'s_name': 'Harshit', 's_id': 1}
{'s_name': 'John', 's_id': 2}
```

In [30]:
```
print(s1)
```

```
<__main__.Student object at 0x7f9c786aadf0>
```

In [31]:
```
class Student:


    #constructor! #dunder methods
    def __init__(self,name, sid):
        self.s_name=name
        self.s_id=sid


    def __repr__(self):
        return f"{vars(self)}"



s1=Student(sid=1,name="Harshit")
s2=Student(sid=2,name="John")
s3=Student(sid=3,name="Smith")

print(s1) #works now!
```

```
{'s_name': 'Harshit', 's_id': 1}
```

In [34]:
```
#<------student{'s_name': 'Harshit', 's_id': 1}------>

class Student:


    #constructor! #dunder methods
    def __init__(self,name, sid):
        self.s_name=name
        self.s_id=sid


    def __repr__(self):
        return f"<------{vars(self)}------>"



s1=Student(sid=1,name="Harshit")
s2=Student(sid=2,name="John")
s3=Student(sid=3,name="Smith")

print(s1) #works now!
```

```
<------{'s_name': 'Harshit', 's_id': 1}------>
```

In [37]:
```
#<------student{'s_name': 'Harshit', 's_id': 1}------>

class Student:


    #constructor! #dunder methods
    def __init__(self,name, sid):
        self.s_name=name
        self.s_id=sid


    def __repr__(self):
        return f"<------Name: {self.s_name}, ID: {self.s_id}------>"



s1=Student(sid=1,name="Harshit")
s2=Student(sid=2,name="John")
```

```
s3=Student(sid=3,name="Smith")

print(s1) #works now!
```

```
<------Name: Harshit, ID: 1------>
```

In [40]:

```
#<------student{'s_name': 'Harshit', 's_id': 1}------>
#s_gpa
class Student:


    #constructor! #dunder methods
    def __init__(self,name, sid,gpa):
        self.s_name=name
        self.s_id=sid
        self.s_gpa=gpa


    def __repr__(self):
        return f"<------Name: {self.s_name}, ID: {self.s_id} GPA: {self.s_gpa}------>"


    def changeName(self,new_name):
        self.s_name = new_name



s1=Student(sid=1,name="Harshit",gpa=3.6)


print(s1) #works now!


s1.changeName("Harshit Pradeep Shukla")

print("After changing name",s1,sep="\n")
```

```
<------Name: Harshit, ID: 1 GPA: 3.6------>
After changing name
<------Name: Harshit Pradeep Shukla, ID: 1 GPA: 3.6------>
```

In [ ]:

```
# create a class Employee which has 3 properties ename, eid and esalary

# add a suitable representation mechanism of your own choice

# create a method raiseSalary that increases the salary of the current object by 10 % of its original value
```

In [47]:

```
class Student:

    #class variable
    School_Name="TJSC" #common for the entire class
    landline_number=27547765


    #special dunder method
    def __init__(self,name, sid,gpa): #constructor

        #instance variables
        self.s_name=name
        self.s_id=sid
        self.s_gpa=gpa


    #special dunder method
    def __repr__(self):
        return f"<------Name: {self.s_name}, ID: {self.s_id} GPA: {self.s_gpa}------>"


    #instance methods
    def changeName(self,new_name):
        self.s_name = new_name



s1=Student(sid=1,name="Harshit",gpa=3.6)
s2=Student(sid=2,name="John",gpa=3.7)

print(s1.landline_number)
```

```
27547765
```

In [48]:

```
Student.landline_number=22345678
```

In [48]:
```
Student.landline_number = 33345678
```

In [49]:
```
print(s1.landline_number)
```

```
33345678
```

In [ ]:
```
#class methods
```

In [55]:
```
class Employee:

    factor=0.1

    def __init__(self,name,eid,sal):
        self.e_name = name
        self.e_eid = eid
        self.e_sal = sal
    def __repr__(self):
        return f"1..Name: {self.e_name}, 2..ID: {self.e_eid}, 3..Salary: {self.e_sal}"

    #instance method
    def raiseSalary(self):
        self.e_sal = self.e_sal + self.e_sal*Employee.factor


    @classmethod
    def changeFactor(cls,new_factor):
        cls.factor=new_factor

e1= Employee(eid=1,name="Sukesh", sal=10000)
print(e1)
e1.raiseSalary()
print(e1)
```

```
1..Name: Sukesh, 2..ID: 1, 3..Salary: 10000
1..Name: Sukesh, 2..ID: 1, 3..Salary: 11000.0
```

In [56]:
```
Employee.changeFactor(0.4) #changing attribute of the class!

e1.raiseSalary()
print(e1)
```

```
1..Name: Sukesh, 2..ID: 1, 3..Salary: 15400.0
```

In [ ]:
```
#Note

#Class variables MUST be modified by class methods
#instance variables need to be modifed by instance method
```

In [57]:
```
class Student:

    #class variable
    School_Name="TJSC" #common for the entire class
    landline_number=27547765


    #special dunder method
    def __init__(self,name, sid,gpa): #constructor

        #instance variables
        self.s_name=name
        self.s_id=sid
        self.s_gpa=gpa


    #special dunder method
    def __repr__(self):
        return f"<------Name: {self.s_name}, ID: {self.s_id} GPA: {self.s_gpa}------>"


    #instance methods
    def changeName(self,new_name):
        self.s_name = new_name



s1=Student(sid=1,name="Harshit",gpa=3.6)
s2=Student(sid=2,name="John",gpa=3.7)

print(s1.landline_number,s2.landline_number,sep="\n")
```

```
27547765
```

```
27547765
```

In [58]:
```
#change class variable, such that ALL OBJECTS SEE THE NEW CHANGE

Student.landline_number = 33456789 #IMP


print(s1.landline_number,s2.landline_number,sep="\n")
```

```
33456789
33456789
```

In [59]:
```
s1.landline_number = 11111111

print(s1.landline_number,s2.landline_number,sep="\n")
```

```
11111111
33456789
```

In [60]:
```
print(vars(s1))
print(vars(s2))
```

```
{'s_name': 'Harshit', 's_id': 1, 's_gpa': 3.6, 'landline_number': 11111111}
{'s_name': 'John', 's_id': 2, 's_gpa': 3.7}
```

In [ ]:
```
Employee

Manager Directors Executives
```

In [71]:
```
class Employee:

    factor=0.1

    def __init__(self,ename,eid,esal):
        print("INSIDE PARENT")
        self.e_name = ename
        self.e_eid = eid
        self.e_sal = esal
    def __repr__(self):
        return f"1..Name: {self.e_name}, 2..ID: {self.e_eid}, 3..Salary: {self.e_sal}"

    #instance method
    def raiseSalary(self):
        self.e_sal = self.e_sal + self.e_sal*Employee.factor


    @classmethod
    def changeFactor(cls,new_factor):
        cls.factor=new_factor

class Manager(Employee):
    factor=0.1

    def __init__(self,department,eid,esal,ename):
        print("INSIDE CHILD")
        self.department = department
        print("Department set")
        super().__init__(ename=ename,esal=esal, eid=eid )


    def __repr__(self):
        return f"Department: {self.department} {super().__repr__()}"

   """
   method overriding-->concept of changing an inherited method inside the child
   """
    def raiseSalary(self):
        data=float(input("Enter a factor: "))
        self.e_sal = self.e_sal + self.e_sal*data


m1=Manager(department="IT",eid=10,esal=29000,ename="john")
m1.raiseSalary()
print(m1.e_sal)
print(m1)
```

```
INSIDE CHILD
Department set
```

```
INSIDE PARENT
Enter a factor: 0.6
46400.0
Department: IT 1..Name: john, 2..ID: 10, 3..Salary: 46400.0
```

In [ ]:
```
"""
Types of inheritance
1)single inheritance----> only one parent, 2 levels--> parent and child

2) Multi-level inheritance---> more than 2 levels. Possibility of parent, grand parent, great grandparent and so
"""
```

In [79]:
```
class Employee:

    factor=0.1

    def __init__(self,ename,eid,esal):
        print("INSIDE PARENT")
        self.e_name = ename
        self.e_eid = eid
        self.e_sal = esal
    def __repr__(self):
        return f"1..Name: {self.e_name}, 2..ID: {self.e_eid}, 3..Salary: {self.e_sal}"

    #instance method
    def raiseSalary(self):
        self.e_sal = self.e_sal + self.e_sal*Employee.factor


    @classmethod
    def changeFactor(cls,new_factor):
        cls.factor=new_factor

class Manager(Employee):
    factor=0.1

    def __init__(self,department,eid,esal,ename):
        print("INSIDE CHILD")
        self.department = department
        print("Department set")
        super().__init__(ename=ename,esal=esal, eid=eid )


    def __repr__(self):
        return f"Department: {self.department} {super().__repr__()}"


    def raiseSalary(self):
        data=float(input("Enter a factor: "))
        self.e_sal = self.e_sal + self.e_sal*data


class Director(Manager):


    def __init__(self, secretary_name, equity_percent,department, eid, esal, ename):
        self.secretary_name = secretary_name
        self.equity_percent = equity_percent
        super().__init__( department, eid, esal, ename)

    def __repr__(self):
        return f"{super().__repr__()} Secretary:{self.secretary_name} Equity:{self.equity_percent}"


d1=Director(secretary_name="Ajay", equity_percent= 45.00, department="IT",eid=101,esal=101000, ename="Joseph")
print(d1)
```

```
INSIDE CHILD
Department set
INSIDE PARENT
Department: IT 1..Name: Joseph, 2..ID: 101, 3..Salary: 101000 Secretary:Ajay Equity:45.0
```

In [ ]:
```
"""
Types of inheritance
1)single inheritance----> only one parent, 2 levels--> parent and child

2) Multi-level inheritance---> more than 2 levels. Possibility of parent, grand parent, great grandparent
and so on

3) Multiple Inheritance---> More than 1 immediate parent classes for a child
"""
```

In [81]:
```
class Mother:
```

```
    mother_tongue= "Hindi"


class Father:
    lastName="Singh"


class Child(Mother, Father):

    def __repr__(self):
        return f"Language: {Child.mother_tongue} Surname: {Child.lastName}"


c1=Child()

print(c1)
```

```
Language: Hindi Surname: Singh
```

In [82]:

```
#exception handling

print("hello")

print( 10+20  )


print( 10 + "harshit")


print(  20 + 30 )
```

```
hello
30
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-82-f00763f35634> in <module>
      4
      5
----> 6 print( 10 +"harshit")
      7
      8

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

In [83]:

```
print( 10 / 0 )
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-83-8086fce235bb> in <module>
----> 1 print( 10 / 0 )

ZeroDivisionError: division by zero
```

In [84]:

```
l1=[1,2,3,4,5]

l1.split()
#string class function on list object?

#when you call a method / access a property which is not applicable for given object
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-84-cac7627171c6> in <module>
      1 l1=[1,2,3,4,5]
      2
----> 3 l1.split()
      4 #string class function on list object?

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

In [85]:

```
int(input("Enter a number "))
```

```
Enter a number hhjkkdslfjljgldskjg
---------------------------------------------------------------------------
```

```
ValueError                                Traceback (most recent call last)
<ipython-input-85-4d45652f3b2b> in <module>
----> 1 int(input("Enter a number "))

ValueError: invalid literal for int() with base 10: 'hhjkkdslfjljgldskjg'
```

In [86]:
```
ord('A')
```

Out[86]: 65

In [90]:
```
#write a program to take 2 numbers from the user. Print the result for division of the 2 numbers

# 4 keyswords for exception handling - try, except, else and finally block

#added feature--->Creating & throwing custom exceptions

"""
give users unlimited chances for giving a correct input. If something goes wrong, explain to the user.
Allow then to Try again
"""
flag = True
while flag:
    try:
        n1= int(input("Enter a number: "))
        n2 = int(input("Enter second number"))

    except ValueError:
                print("sorry. This is not valid. Try again!")


    else:
        try:
            print(n1/n2)
            flag=False #make my loop stop

        except ZeroDivisionError:
            print("cannot divide by zero. please try again")
```

```
Enter a number: hhh
sorry. This is not valid. Try again!
Enter a number: kk
sorry. This is not valid. Try again!
Enter a number: 10
Enter second number20
0.5
```

In [ ]:
```
"""
try----->write code that MAY throw exception here
except--->these blocks handle exception that MAY arise inn try block. You can write multiple except blocks
for every try
else-----> we write the steps to be taken IF NO EXCEPTIONS ARISE IN try inside this else block
finally----> any activity that needs to be performed in both cases, exception or no exception
"""
```

In [93]:
```
flag=True
while flag:
    try:
        n1= int(input("Enter a number: "))
        n2 = int(input("Enter second number"))
    except ValueError:
                print("sorry. This is not valid. Try again!")

    else:
        try:
            print(n1/n2)
            flag=False #make my loop stop

        except ZeroDivisionError:
            print("cannot divide by zero. please try again")


    finally:
        print("CONNECTION CLOSE. GOOD BYE!")
```

```
Enter a number: 10
Enter second number0
```

```
cannot divide by zero. please try again
CONNECTION CLOSE. GOOD BYE!
Enter a number: 10
Enter second number2
5.0
CONNECTION CLOSE. GOOD BYE!
```

In [95]:

```python
#verify if number given by user is valid

import re



pattern = re.compile( r"^\+91[6-9][0-9]{9}$" )



data=input("Enter your mobile number: ")

if re.search(pattern, data ):
    print("SUCCESS")
else:
    print("failure")
```

```
Enter your mobile number: +18779092028
failure
```

In [ ]:

```python
"""
email ID should be at least 6 characters excluding the domain.
Domain should be gmail.com
IT cannot start with a number
special symbols allowed in id are _ - and #

(underscore, hyphen and hash)
"""
```