```python
value = int(input("Enter input data:"))
print(type(value), isinstance(value, int))
```

```
Enter input data: 3

<class 'int'> True
```

```python
#Formatting Strings
#-------------------------------F-String in python 3.6
----------------------------
print(f"Entered value is {value+2:.2f}")  # with value, operator,
modifiers(using :)
print(f"{'Expensive' if value>20 else 'Cheap'}") #with ifelse
statement
print(f"{float(value*2)}") #with inbuilt-function and operator
def convert(a):
    return str(a)
print(f"Converted value to string {convert(value)}")

#-----------------------------format before python 3.6
----------------------------
print("Entered value is {0} and the value in double is
{1:.3f}".format(value, value))
#keep only modifiers inside place holder and we can use indexes to
ensure in correct order
age=25
name="fury"
print("My name is {1}, {1} is {0} years old".format(age, name))
print("My name is {name}, {name} is {age} years
old".format(name="gani", age=24))
#---------------------------using %
operator-----------------------------------------
print("Hello, %s! You are %.2f years old." %(name, age))  #Usr print
and use % for modifiers
```

```
Entered value is 5.00
Cheap
6.0
Converted value to string 3
Entered value is 3 and the value in double is 3.000
My name is fury, fury is 25 years old
My name is gani, gani is 24 years old
Hello, fury! You are 25.00 years old.
```

```python
#Variables
#Dynamically Typed
#Type Inference
name="fury"
age=25
x,y=2,3
```

```
a=b=c=4
print(a,b,c,x,y)
print(name, age)

fruits=["apple", "banana"]
m,n=fruits #Unpacking a collection (mainly we use tuple)
print(m,n)

print(x+y) #Here arthimetic operation as int
print(m+n) #Here concatenation as str
#print(x+m) #Error as 1 str and 1 int
height=180
def my_func():
    global height #To change value of global variable
    height=175
my_func()
print(height)
```

```
4 4 4 2 3
fury 25
apple banan
5
applebanan
175
```

```
#Data Types
#Multi-line commenting isnothing but multiline string not assigned to
any variable
"""
bytes, int, float, str, bool, complex (#no short, long, double, char
in python)
list, tuple, range
dict
set, frozenset
NoneType"""
```

```
'\nbytes, int, float, str, bool, complex (#no short, long, double,
char in python)\nlist, tuple, range\ndict\nset, frozenset\nNoneType'
```

```
#Conditional
a=int(input("Enter a number:"))
if a<20:
    print("kid")
elif a>20 and a<50:
    print("adult")
else:
    print("old")

if 3>2: print("good") # short hand if

#Multi condition ternary operator
```

```python
op="kid" if a<20 else "adult" if a>20 and a<50 else "old"
print(op)

name="fuRY"
#short hand if else (Ternary operators) (no use of : andindendation
here)
res=name if name==name.lower() else name.lower()
print(res)

if 2>1:
    pass

#Case statment
match name: #works in newer version
    case "fury":
        print("hi fury")
    case "fuRY":
        print("hi fuRY")
    case _:
        print("Ntg")

#Operators
"""
Arithmetic: +, -, *, /, %, //, **(Exponential)
Assignment: +, +=, -+, *=, /=, %=, //+, &=, |=, ^=, >>=, <<=, :=(used
for assigning not comparing)
Comparison: ==, !=, >, <, >=, <= (used in if conditions)
Logical: and, or, not
Identity: is, is not (compares objects references not values)
Membership: in, not in
Bitwise: &, |, ^, ~, <<, >>  (x>>y means x/2**y)
Boolean: bool("hi") - True (Capital T/F)
         bool(0), bool(""), bool(False), bool(None), bool(()),
bool([]), bool({}) - False
Operator Precedence: (), **, Unary, BODMAS, >>, <<, &, ^, |,
Comparison, identity, membership, not, and, or
"""

Enter a number: 30

adult
good
adult
fury
hi fuRY

#Loops
for k in range(0, 10, 3): #used for iterating over a sequence/iterable
object (like iterators in oops, but not traditional for loop)
    print(k)
```

```
else:
    print("executed for loop") #will not be executed if loop is
stopped by break statement

#here remember for each value of 1st loop, 2nd loop executes fully
for i in range(1,3): #Excluding 3
    for j in range(1,4):
        print(f"i={i}, j={j}")

i=4 #set variable
while i>0: #till condition is true
    print(i)
    if i==2:
        break
    i-=1 #increment/decrement
else:
    print("Executed while loop") #will run if we don't encounter break

i=4 #set variable
while i>0: #till condition is true
    i-=1 #increment/decrement
    if i==2:
        continue
    print(i) #statements to execute

#For loop with enumerate (getsboth index and value of elements during
iteration)
squares=[1,2]
for i, value in enumerate(squares): #gives tuples but we unpacking
here by i, value
    print(f"Index {i}: value {value}")

#Forloop with zip (allows iterator over multiple iterables in
parallel)(gets pairs of tuples for each iterable)
l1=[1,2,3]
l2=['a', 'b', 'c']
for i, j in zip(l1,l2): #gives tuples but we unpacking here by i, j
    print(f" Number: {i}, Character: {j}")

#---------------------------------List
Comprehensions------------------------
#List comprehension (concise way to create lists)
squares=[x**2 for x in range(0,5,1) if x%2==0 and x<5]
print(squares)

numbers=[1,2,3,4,5,6]
result=["Even" if e%2==0 else "Odd" for e in numbers]
print(result)

#here remember for each value of 1st loop, 2nd loop executes fully
```

```python
combinations=[(x,y) for x in range(3) for y in range(2)] #need to give (x,y) bcz it will get error
print(f"combinations is {combinations}")

matrix=[[1,2,3], [4,5,6], [7,8,9]]
flattened=[e for sublist in matrix for e in sublist]
print(f"flattened is {flattened}")

keys=['a', 'b', 'c']
values=[1,2,3]
dict_comprehension={keys[i]:values[i] for i in range(len(keys))}
print(dict_comprehension)

set_comprehension={x*2 for x in range(-3, 4)}
print(set_comprehension) #This removes duplicates

l1=[1,2,3]
l2=[4,5,6]
op=[x+y for x,y in zip(l1, l2)]
print(op)

#def is_prime(x):
''' primes=[x for x in range(20) if is_prime(x)]
print(primes)'''

0
3
6
9
executed for loop
i=1, j=1
i=1, j=2
i=1, j=3
i=2, j=1
i=2, j=2
i=2, j=3
4
3
2
3
1
0
Index 0: value 1
Index 1: value 2
 Number: 1, Character: a
 Number: 2, Character: b
 Number: 3, Character: c
[0, 4, 16]
['Odd', 'Even', 'Odd', 'Even', 'Odd', 'Even']
combinations is [(0, 0), (0, 1), (1, 0), (1, 1), (2, 0), (2, 1)]
```

```
flattened is [1, 2, 3, 4, 5, 6, 7, 8, 9]
{'a': 1, 'b': 2, 'c': 3}
{0, 2, 4, 6, -6, -4, -2}
[5, 7, 9]

' primes=[x for x in range(20) if is_prime(x)]\nprint(primes)'
```

```
#iterators
"""iterator is an object "that can be iterated upon", u can traverse
through all values in iterable
iterator object implements the iterator protocol, which consists of
__iter__(), __next__() methods

list, tuple, set, string are all iterable objects "which u can get an
iterator"
all these object have iter() method which is used to get an
iterator"""

name="ganesh"
myiter=iter(name)
print(next(myiter))
print(next(myiter))

for i in name: # for loop actually creates an iterator object and
executes the next() method for each loop
    print(i) # i is not iterator here, it issimply a variable that
hold each item returned by the iterators next() method
```

```
g
a
g
a
n
e
s
h
```

```
#Strings
name='bollapalli-ganesh' #String literal can be in siingle/double
quote
aliasname="fury:gani"
age=25
print(name) #Strings are array of bytes(no character datatype in
python, it will be string of length 1)
print("My name is 'Ganesh'") #will work as long as it won't affect
#Print("My name is Ganeh of age"+age) # will get error
print(f"My name is {name.lower()} and age {age-1:.2f}")
print("I am called \"Fury\"") #To insert illegal characters use \\\\\\
(\', \", \\(to print \), \\\n(to print \n), \t....)

print("ganesh" in name)
```

```
if "ganesh" in name: ##similarly not in
    print("Yes")

for c in name[1:3]: #looping on string via elements
    print(c)
for i in range(len(name)): #looping on string via index
    print(name[i])

print(name, name[1], name[2:5], name[-3:-1]) #5, -1 index values
excluding
print(len(name), ''.join(sorted(name)), ''.join(set(name)),
''.join(reversed(name))) #bcz sorted amd reversed will gives list
print(type(name), min(name), max(name))
print(name.find("sh", 2, 7), name.rfind("a"), name.index("sh", 2),
name.rindex("sh"), name.count("a", 1, 5))

print(name.strip(",.grt"), name.lstrip(",.asw"), name+" "+aliasname)
print(name.capitalize(), name.upper(), name.title(), name.swapcase())
print(name.ljust(20, "o"), name.rjust(20, "o"), name.center(20, "o"),
name.zfill(10)) #default " "(space)
print(name.split("-", 2), name.rsplit(",", 1), name.splitlines(True))
#True to keep line breaks in output
print(name.endswith("sh", 1, 10))
print(name.replace("g", "G", 1))

print(name.encode("ascii"), name,expandtabs(2), name.partition("ga"),
name.rpartition("ga"))
print(name.isalnum(), name.isalpha(), name.isascii(),
name.isdecimal(), name.isdigit(), name.islower(), name.istitile())
print(name.isspace(), name.isnumeri(), name.isprintable(),
name.isidentifier(), name.maketrans("shgane", "ipprad")
      name.translate(str.maketrans("sy", "km"))) #inside translate
giev dict or mapping table

'''
Scala                                   -     Python
name.forall(_.isLower)                  -     all(c.islower() for c in
name)
name.exists(_.isUpper)                  -     any(c.isupper() for c in
name)
name.find(_.isDigit)                    -     next((c for c in name if
c.isdigit), None)
name.splitAt(n)                         -     name[:n], name[n:]
isEmpty, nonEmpty                       -     s, not s
name.map(_.toUpper)                     -     ''.join(map(lambda c:
c.upper(), name))
name.flatMap(c=>List(c))                -     ''.join([x for c in name
for x in c]) #no flatmap in python
name.filter(_.isDigit)                  -     ''.join(filter(lambda x:
x.isdigit(), name))
```

```
name.fold("")((x,y)=>x+y)            -     reduce(lambda x,y: x+y,
name)
name.foldLeft("")((x,y)=>x+y)        -     reduce(lambda x,y: x+y,
name, '')  # name[::-1] for foldRight
name.foreach(println)                -     for c in name: println  #no
foreach hof in python
name.groupBy(_.isLetter)             -     {k:list(v) for k,v in
groupby(sorted(name))} #Sort, groupby to work
name.zip(name1)                      -      zip(name, name1)
name.zipWithIndex                    -     enumerate(name)
name.collect{case c if c.isDigit=>c} -     ''.join([c for c in name if
c.isdigit()])
name.take(3)                         -     name[:3]
name.drop(3)                         -     name[3:]
name.takeWhile(_.isLetter)           -
''.join(takeWhile(c.isalpha(), name))
name.partition(_.isDigit)            -    ''.join([c for c in name if
cond]), ''.join([c for c in name if not cond])
```

```
bollapalli-ganesh
My name is 'Ganesh'
My name is bollapalli-ganesh and age 24.00
I am called "Fury"
True
Yes
o
l
b
o
l
l
a
p
a
l
l
i
-
g
a
n
e
s
h
bollapalli-ganesh o lla es
17 -aaabeghillllnops slanbg-opieh hsenag-illapallob
<class 'str'> - s
-1 12 15 15 1
bollapalli-ganesh bollapalli-ganesh bollapalli-ganesh fury:gani
```

```
Bollapalli-ganesh BOLLAPALLI-GANESH Bollapalli-Ganesh BOLLAPALLI-
GANESH

#Collections
#--------------------------------List(Hetero)/Array(Homo)-----------
---------
#Hetero, mutable, ordered, dup
l=[1,"fury", 24, 5.10] #(or) list((1, "fury", 24, 5.10))
# [] (or) list(()) for empty list

print("fury" in l)
if "fury" in l: #similarly not in
    print("yes, fury exists")
for e in l: #looping through list
    print(e)
for i in range(len(l)): #looping thtough index
    print(l[i])

print(len(l), type(l), l[1], [2:4], l[-2])
l[1]="gani" #chnage value at index
#change values at range of index
l[1:3]=["fury","hero", 25] #inserting values>replacing index, then
remaiining items move accordingly (length will increase)
l[1:3]=["fury"] #inserting values<replacing index, then old values are
replaced with new low no of values ( length will descrease )

l1=["apple", "banana", "kite"]
l1.sort(reverse=True, key=str.lower) #sort list in place
sorted_list=sorted(l1) # which will generate a new list
l2.reverse() # don't place in print bcz it reverses in place and
returns None
reversed_list=list(reversed(l2))
distinct=list(set(l))

mylist=[1,2,3,4,5]
print(min(mylist), max(mylist), sum(mylist), any(mylist), all(mylist))
print(l.index(24), l.count(2))

l.insert(2, "King") # at specified index
l.append("hyd") # at end
l.extend([1,2,3]) #can be tuple, dict, set
l+l1
l.remove(3) #specific elemenst (if dup removes first occurence )
l.pop(6) # at specified index
l.pop() # last element
del l[5], del l, l.clear() #Removes elements in clear, but in del
removes list object completely
l3=l1.copy() #(or) list(l1) (or) l1[1:]

[x for x in l] #creates new list based on existing list
```

```
newList = [e for e in l if str(e).isnumeric()] #creaes newList keeping old list unchanged

'''
res=list(map(lambda x: x*2, list))
res=list(filter(lambda x: x%2==0, list))
res=reduce(lambda x,y: x+y, list)
res=any(e.isdigit for e in list)
res=all(e.isdigit list) # simple way
res=next((x%2==0 for x in list), None) # simple way
res=zip(l1,l2)
res-list(enumerate(list))
'''

#-----------------------------Dict(Hash Table)(keys are hashed and values stored)-----------------------
#Hetero, mutable, ordered, unq k - dup v
d={"name":"ganes", "age":25:, 1:2} #(or) dict(name="ganesh", age=25)
x=('k1', 'k2', 'k3')
d2=dict.fromkeys(x, None) #can use in lists to remove duplicates

if "name" in d:
    print(f"yes name exists")
for k,v in d.items(): #so ordered #items returnstuple of k,v in list
    print(k,v)
for k in d.keys(): #even if we give d will get keys only
    print(d[k])
for v in d.values():
    print(v)
if "ganes" in d.values():
    print("yes ganes exists")

print(d, d["name"], d.get("age", 0), tyep(d), d.keys(), d.values(), d.items()) #keys(), values(), items() are always updated
#print(dict(sorted(d.items())), dict(reversed(d,items()))) #sorting ad reversing
d["name"]="fury" #changing existing key value
d.update({"name":"gani"}) #can give other dict
d["city"]="chn" #adding new key, value
d.update({"city":"hyd"}) #it removes duplicates while adding, keeps last inserted one (so removes chn and keeps hyd)
d.pop("city")
d.popitem() #removes last inserted item
del d["name"] # del d deletes completes, d.clear() just cleares elements
d1=d.copy() #(or) dict(d)

'''
res=next(((k,v) for k,v in d.items() if condition)) #same as find in scala
```

```
res=dict(filter(lambda item: item[1]>5, d.items()))
similar for map, filter, ... but use d.items() at right

for k, group in groupby(lambda x:x[1]>5, sorted(dict.items(), lambda
x:x[1]):
    print(key, list(group))'''
#False:[('a', 1), ('b', 2)], True:[('c', 6)]


#------------------------------Sets----------------------------
#Hetero, Immutable, Unordered, Unq) - but u can add/remove items
s={0,1,"fury", 3, 1, False, True} #duplicates will be ignored
s1=Set((1,2,3,4,5))
print(s, len(s), type(s))
#Convert to list to reverse and sort
print(max(s), min(s), sum(s), any(s), all(s))

for i in s1:
    print(i)
if 1 in s:
    print("yes 1 exists")

s.add("hyd")
s.update(s1) #u can add list, tuple, dict #changes original set
s.remove(1) #if 1 not exsts returns error
s.discard(1) #will not raise error
s.pop() #removes randomly s.clear(), del s
s2=s.copy()

'''
res= s|s1|s2 (or) res=s.union(s1, s2, s3) [u can do with list and
tuples also]
res= s&s1    (or) res=s.intersect(s)   [only with sets]
s.intersectipn_update(s1) #changes original set in place
res= s-s1    (or) res=s.difference(s1) [only with sets]
s.difference_update(s1) #changes original set in place
res= s^s1    (or) res=s.symmentric_difference(s1) [elements no in
both]            s.symmetric_difference_update(s1) #changes original
set in place

issubset(), issuperset(), frozenset() (Immutable version of set)

res=set(map(lambda x:x),s)
res=set(filter(lambda x:x%2==0), s)
'''

#-------------------------------
Tuple---------------------------------------
#Hetero, Immutable, Ordered, Dup
t=(1,"fury", 25) #for tuple with 1 element ass , at the end
```

```
t1=typle(("apple", "banana"))

print("fury" in t)
if "orange" not in t1:
    print("yes orange doesn't exists")
for e in t: #loop on elements
    print(e)
for i in range(len(t)): #loop on index
    print(t[i])

print(t, t[1], len(t), t[2:3], t[-1], type(t))
# for sorting nf reverse convert to list
print(t.count(1), t.index(2))

k=list(t) # to change tuple
k[1]="gani"
z=tuple(k)

t+=t1 #concat 2 tuples
#del t[1], del t
(x,y,*z)=t #unpacking
r=z*2 #repeating
#Can apply hof map, filter....
```

```
  Cell In[21], line 139
    print(t, t[1], len(t), t[2:3], [t[-1], type(t))
                                                   ^
SyntaxError: closing parenthesis ')' does not match opening
parenthesis '['
```

```
#Functions
def add(z, x:int, y:int=4)->int: #None is default return tyep (small
int)
    #Non-default before default parameter
    return x+y+z #return keyword mandatory or will get None
print(add(3, y=2, x=3)) #pos before keyword args

#--------------Keyword>Positional>Default-------
def mul(*n:int, **m:int)->int: #gets tuples of arguments *n - pos
args, **m - keywors args
    sum=0
    for i in n:
        sum+=i
    return sum
res=mul(2,3,4,5,l=3, m=4) #we can send any datatype and function
interprets it
print(res)

#Lambda Function
```

```python
a=lambda x,y:x+y #n args but 1 expression
print(a(3,4))
#Best used with other functions

'''
Position only arguments , /
def fun(x, /):
//Logic
fun(3) correct, fun(x=3) gives error

Keyword only arguments * ,
def fun(*, x):
//Logic
fun(x=3), fun(3) gives error

def fun(x,y, /, *, a, b):
//Logic
fun(2,3,n=7, m=8) correct
'''
```

```
8
14
7

'\nPosition only arguments , /\ndef fun(x, /):\n//Logic\nfun(3)
correct, fun(x=3) gives error\n\nKeyword only arguments * ,\ndef
fun(*, x):\n//Logic\nfun(x=3), fun(3) gives error\n\ndef fun(x,y, /,
*, a, b):\n//Logic\nfun(2,3,n=7, m=8) correct\n'
```

```python
#---------------------------------Exception
Handling---------------------------
def fun1(x:int):
    assert x>0, "Give +ve number"
    print(x)
def fun2(y:int):
    if(x<0):
        raise ValueError("Send +ve number")
    else:
        print(x)
try:
    fun1(2)
    fun2(3)
except ValueError as e:
    print("Value Error")
except Exception as e:
    print("error")
finally:
    print("Executed")

#---------------------Regex-------------------------------------
```

```python
import re
s="123abc456"
#pattern we give as groups "(\d{3})-(\d{2})-(\d{4})"
#group(0) - entire match, (1), (2) - specific captured groups
print(re.search("\d+",s).group())
print(re.match("\d+",s).group())
print(re.sub("\d+",'', s))
print(re.findall("\d+", s)) #Gives list
print(re.subn("\d+", '', s))
print(re.split("\d+", s))

#----------------------File Handling-----------------------
#No need to close file descriptor if we use with
with open("file", "r") as f:
    f.read() #read whole file content
    f.read(5) #read first 5 bytes of file
    lines=f.readLines() #reads all lines and returns as list
    line=f.readLine() #read line by line
    while line:
        print(line)
        line=f.readLine()

with open("file", "w")as w:
    w.write("adding new line")
    w.writelines("lines")
```

```
2
2
Executed
123
123
abc
['123', '456']
('abc', 2)
['', 'abc', '']

<>:25: SyntaxWarning: invalid escape sequence '\d'
<>:26: SyntaxWarning: invalid escape sequence '\d'
<>:27: SyntaxWarning: invalid escape sequence '\d'
<>:28: SyntaxWarning: invalid escape sequence '\d'
<>:29: SyntaxWarning: invalid escape sequence '\d'
<>:30: SyntaxWarning: invalid escape sequence '\d'
<>:25: SyntaxWarning: invalid escape sequence '\d'
<>:26: SyntaxWarning: invalid escape sequence '\d'
<>:27: SyntaxWarning: invalid escape sequence '\d'
<>:28: SyntaxWarning: invalid escape sequence '\d'
<>:29: SyntaxWarning: invalid escape sequence '\d'
<>:30: SyntaxWarning: invalid escape sequence '\d'
C:\Users\Dell\AppData\Local\Temp\ipykernel_12484\3866192383.py:25:
SyntaxWarning: invalid escape sequence '\d'
```

```
  print(re.search("\d+",s).group())
C:\Users\Dell\AppData\Local\Temp\ipykernel_12484\3866192383.py:26:
SyntaxWarning: invalid escape sequence '\d'
  print(re.match("\d+",s).group())
C:\Users\Dell\AppData\Local\Temp\ipykernel_12484\3866192383.py:27:
SyntaxWarning: invalid escape sequence '\d'
  print(re.sub("\d+",'', s))
C:\Users\Dell\AppData\Local\Temp\ipykernel_12484\3866192383.py:28:
SyntaxWarning: invalid escape sequence '\d'
  print(re.findall("\d+", s)) #Gives list
C:\Users\Dell\AppData\Local\Temp\ipykernel_12484\3866192383.py:29:
SyntaxWarning: invalid escape sequence '\d'
  print(re.subn("\d+", '', s))
C:\Users\Dell\AppData\Local\Temp\ipykernel_12484\3866192383.py:30:
SyntaxWarning: invalid escape sequence '\d'
  print(re.split("\d+", s))
C:\Users\Dell\AppData\Local\Temp\ipykernel_12484\3866192383.py:25:
SyntaxWarning: invalid escape sequence '\d'
  print(re.search("\d+",s).group())
C:\Users\Dell\AppData\Local\Temp\ipykernel_12484\3866192383.py:26:
SyntaxWarning: invalid escape sequence '\d'
  print(re.match("\d+",s).group())
C:\Users\Dell\AppData\Local\Temp\ipykernel_12484\3866192383.py:27:
SyntaxWarning: invalid escape sequence '\d'
  print(re.sub("\d+",'', s))
C:\Users\Dell\AppData\Local\Temp\ipykernel_12484\3866192383.py:28:
SyntaxWarning: invalid escape sequence '\d'
  print(re.findall("\d+", s)) #Gives list
C:\Users\Dell\AppData\Local\Temp\ipykernel_12484\3866192383.py:29:
SyntaxWarning: invalid escape sequence '\d'
  print(re.subn("\d+", '', s))
C:\Users\Dell\AppData\Local\Temp\ipykernel_12484\3866192383.py:30:
SyntaxWarning: invalid escape sequence '\d'
  print(re.split("\d+", s))

---------------------------------------------------------------------
-----
FileNotFoundError                         Traceback (most recent call
last)
Cell In[27], line 34
     30 print(re.split("\d+", s))
     32 #----------------------File Handling-----------------------
     33 #No need to close file descriptor if we use with
---> 34 with open("file", "r") as f:
     35     f.read() #read whole file content
     36     f.read(5) #read first 5 bytes of file

File ~\AppData\Local\Programs\Python\Python313\Lib\site-packages\
IPython\core\interactiveshell.py:325, in _modified_open(file, *args,
**kwargs)
```

```
    318 if file in {0, 1, 2}:
    319     raise ValueError(
    320         f"IPython won't let you open fd={file} by default "
    321         "as it is likely to crash IPython. If you know what
you are doing, "
    322         "you can use builtins' open."
    323     )
--> 325 return io_open(file, *args, **kwargs)

FileNotFoundError: [Errno 2] No such file or directory: 'file'
```

```
#-------------------------------------OOps---------------------------
class One:
    print("running parent class")
    def __init__(self, name, age):
        #Constructor used to initialize values for obj
attributes(called automatically when obj created)
        self.name=name
        self.age=age
    def show(self):#refers to current instance, allows to access
fields, methods within class
        #used as 1st parameter to distinguish btw instance vraibales
and local variables
        height=175
        println(f"my name is {self.name} and age is {self.age} and
height {height}")
        #for local variable self is not required
class Two(One):
    print("running child class")
    def __init__(self, name, age, loc): #it will override base class
init method
        super().__init__(name, age) #in scala it runs automatically
here we need to mention this line (can use One. as well)
        self.loc=loc
    '''if don;t want to pass name, age here give default values
    def __init__(self, loc):
        super().__init__("fury", 25) / One.__init__("fury", 25)
        self.loc=loc
    '''
    @staticmethod
    #use thisdecorator(independant of both class and instance)(utility
fun related to cls but no requiring acess to it's attributes)
    def my_method(p1, p2): #no self keyword
        print(f"static method called with {p1} and {p2}")

    def show(self):
        print(f"my name is {self.name} and age is {self.age} and loc
is {self.loc}") #use self
    def greet(self, country):
        print(country) # ***don't give self.country*** (bcz it doesn't
```

```
belongs to One (or) Two class, just passing to method directly,
similar to local variable

o=Two("Fury", 25, "hyd") #Need to give () definetly
o.show()
o.greet("india")
Two.my_method(2,3) #static method called on class itself

running parent class
running child class
my name is Fury and age is 25 and loc is hyd
india
static method called with 2 and 3

#-------------------------Generators-----------------------------
'''Allows u to iterate over a sequence of values lazily(one value at a
time)
Uses yield to produce value 1 by 1 and maintain their state between
each yield'''
def generator(n):
    a=0
    while a<n:
        yield a
        a+=1

for i in generator(5):
    print(i)

def fibanocci_series(n):
    a,b=0,1
    x=0
    while x<n:
        yield a
        a,b=b,a+b
        x+=1
for i in fibanocci_series(5):
    print(i)

0
1
2
3
4
0
1
1
2
3
```

```python
#----------------------------Decorators-------------------------
'''Modify/Enhance behaviour of functions/methods without changing
their actual code'''
def dec(fun):
    def wrapper():
        print("hi")
        fun()
        print("bye")
    return wrapper
@dec
def sayhello():
    print("hello")
sayhello()

def repeat(n):
    def decorator(fun):
        def wrapper(*args, **kargs):
            for i in range(0,n):
                fun(*args, **kargs)
        return wrapper
    return decorator
@repeat(3)
def details(name):
    print(name)
details("fury")
```

```
hi
hello
bye
fury
fury
fury
```

```python
import copy
l=[1,2,[3,4]]
l1=copy.copy(l) #shallow copy(doesn't recusrively copies the objects
within)(copies references)
l1[2][0]=99 #when we change l1, values in l also will change
print(f"original {l}, copied {l1}") #bcz nested list is shared by both

m=[1,2,[3,4]]
n=copy.deepcopy(m) #deep copy(recursively copies all objects)(no
references are shared)
n[2][0]=99
print(f"original {m}, copied {n}")

#shallow equality
l=[1,2,[3,4]]
print(l is m) #compares object memory
```

```
#Deep equality
print(m==n)

original [1, 2, [99, 4]], copied [1, 2, [99, 4]]
original [1, 2, [3, 4]], copied [1, 2, [99, 4]]
False
False

'''
--------------------Python Features--------------------------------
Python: Interpreted, Dynamically Typed, Highlevel(multi paradigm)
Scala: Compiled, Statically Typed, multiparadigm(pure oops + FPL)

python 3.x over 2.x (print is function, integer divison not rounding
in 3.x, range, unicode)

Python optimization: List comprehension, garbage collection, built-in
functions, avoding global variables, multi-threading

Python singleton object (__new__)

--------------------Garbage Collection------------------
Memory Management
removes unused objects
using reference counting (or) cyclic garbage collector
impprt gc
gc.collect
#It does automatically

------------------Module vs Package-----------------------
Module(file): Single .py file with func, cls in it (use dir() to list
them all)
Package(dir): Contains many .py modules/sub-packages, must contain
__init__

------------------Range vs XRange---------------------------
Range - returns list, stores list
XRange - returns iterator, done on demand (Removed in python 3.x)
range(5) -> [0,1,2,3,4]
Xrange(5) -> Xrange(5)

------------------With statement(context management)-------------
AUtomatic cleanup(closing a file, releasing lock), Code cleaner and
more readable

-----------------MultiThreading, GIL--------------------------
MultiThreading: Multiple threads(process) to run concurrently
GIL: Mutex(lock) that allows only one thread to execute at a time
#To run multi-threading for CPU boudning task and avoid GIL, we can
have seperate namepsace for each process
```

```
__str__: user-friendly string representation of object "print...."
__repr__:official string representation "MyClass()"
obj=MyClass()
print(str(obj)) #uses __str__
print(repr(obj)) #uses __repr__
'''
```