* Function: Breaking any big task into small module which is called function.
* Functional Programming: Completing a task without manipulating data with the use of function.
* Attribute: Property
* Object Oriented Programming:
* Method: Function in object-oriented programming is called method.
* ENCAPSULATION:
* ABSTRACTION:
* POLYMORPHISM:
* CLASS: Is a design or blueprint of object. 1.ATTRIBUTE: Variable 2. BEHAVIOUR: Method (Function)
* OBJECT: Before creating an object you have to create an class.
* In built type: Float, int,
* OOPS CONCEPT:
* HEAP MEMORY: CONTAINS ALL OBJECT,
* VARIABLES IN OOPS: If variables are defined inside the \_\_init\_\_called 1.INSTANCE VARIABLE, defined outside called 2.CLASS(STATIC) VARIABLE
* Class and static variables are same but methods are different
* ACCESOR: Fetch the value
* MUTATOR:
* Self-Similarity: the same geometrical figure infinite time. Its ease our work to solve the problem. 1. We know how to move two blocks from A to B. def Solve (K, Source, Target, Extra)  
  SOLVE HANOI(4, “A”, “B”, “C”)

If k<1 then return Magic Hanoi(k-1,source, extra, target) pri nt(f “move block {k} from {source} to { target}”) Magic Hanoi( k-1, extra, target, source)

* Recursion Function, Selection Sort, Binary Search, Pascals Triangle
* DrawS(count, size)
* If count<1 then return t.forward(size) t.right(90) drawS(count-1)
* T(n)=T(n-1)+(n-1) recurrence problem
* UNARY OPERATOR (1 OPERAND, ONLY NOT), BINARY OPERATOR (2, ALL)

ICP-PYTHON

PYTHON-TOOLS

*# ++++++++++++ PYTHON TOOLS ++++++++++++++++#  
  
  
  
  
  
#############################  
# CONSOLE INPUT/ CONSOLE OUTPPUT-INPUT BY KEYBOARD AND OUTPUT ON THE SCREEN  
#####  
# x,y,z=input("Enter three digits:").split()  
# print(x, y, z)  
# scores=[int(n) for n in input("Enter value").split()] # IT CREATES A LIST OF SCORES  
# print(scores, sep=",", end="!")  
# FORMATTED PRINTING  
# 1...USING FORMATTED STRING LITERAL  
#*R,B,H=1.233234234,90,120  
print(**f'1: radius={**R**} breadth={**B**} Height={**H**}'**)  
*# NB*n=**'man dhan tan yadav'  
for** x **in** n.split():  
 print(**f'{**x**:10}'**)  
*# 2...FORMAT(0) METHOD OF STRING OBJECT*name, age, gender=**"Manish"**, 19, **"m"**print(**'2:''name={}, age={}, gender={}'**.format(name, age, gender))  
print(**'3:''name={2}, age={0}, gender={1}'**.format(age, gender, name))  
print(**'name={0:10} gender={0:5}'**.format(name, gender))  
print(**"radius={0:2.2f}"**.format(R))  
*#####*print(**"==================================="**)  
  
  
  
  
  
*#############################  
# DECISION CONTROL INSTRUCTION  
#####*a=40  
b=30  
print(**"1.1:"**, 75 **and** a>=20 **and** b<=60 **and** 35, -30 **and** a>=20 **and** b< 15 **and** 35, -30 **and** a>=20 **and** 0 **and** 35)  
print(**"1.2:"**, 75 **or** a>=20 **or** 75, a<20 **or** 0 **or** 35)  
print(**"1.3:"**, **not** (a<=b))  
x=10  
print(**"1.4:"**, **"Biger than 5" if** x>=5 **else "Lower than 2" if** x<=2 **else 'GOOD'**)  
*#print("1.5:", 'true' if any(a,b,c,d))  
########*print(**"==================================="**)  
  
  
  
  
  
*#############################  
# REPEATITION CONTROL INSTRUCTION  
#####***for** i, ele **in** enumerate([10,20,30]):  
 print(i, ele)  
**for** i **in** range(5,10,2):  
*#for i in range(20, 5, -2):* print(i, end=**",,"**)  
*# INFINITE LOOP*i=1  
**while** 1:  
*# while True:* print(i)  
 i+=1  
 **if** i>10:  
 **break**a=1  
b=2  
c=3  
**for** x **in "abc"**:  
 print(x)  
**import** math  
length=10  
precision=3  
**for** n **in** range(1,10):  
 a=math.sqrt(n)  
 b=math.pow(n,1/3)  
 print(**f'{**n**:10}{**a**:{**length**}.{**precision**}}{**b**:{**length**}.{**precision**}}'**)  
  
*# MULTIPLE STRING*student=10  
qulified=5  
print(**f'\n{"STUDENT:":<1}{**student**:<10}'**,  
 **f'\n{"qulified:":<10}{**qulified**:<10}'**)  
  
details={**"Manish"**:6393241779, **"Deepak"**: 8787085500, **"Saloni"**:7081289995}  
**for** name, mono **in** details.items():  
 print(**f"{**name**:10}:{**mono**:20}"**)  
*#####*print(**"==================================="**)  
  
  
  
*#############################  
# ARGUMENTS IN PYTHON FUNCTION  
#####  
# 1.0: POSITIONAL/ REQUIRED ARGUMENTS***def** p(i,j,k):  
 print(i\*j, k.upper())  
print(**"1.0:"**, p(1,2,**"manish"**)) *# MUST BE IN THE ORDER AS PERFORM ACTION  
# 1.1: KEYWORD ARGUMENT***def** person(name,age):  
 **return** age-6,name  
print(**"1.2:"**, person(age=10,name=**'navin'** )) *# NO MATTER SEQUENCE, VARIABLES ARE DEFINED  
# 1.0 and 1.1 can be used together***def** person(i, int, str):  
 **return** i, int, str  
print(**"1.2:"**, person(10, int=10, str=**"mhgfds"** )) *# 1.0 MUST PRECEDE 1.1  
# 1.3.VARIABLE LENGTH POSITIONAL ARGUMENT***def** add(a,\*b): *# \*b means it can have multiple value* c=a  
 **for** i **in** b:  
 c=c+i  
 **return** c  
print(**"1.3:"**, add(10,20,300,300))  
*# 1.4: VARIABLE LEANGTH KEYWORD ARGUMENT  
#####***def** person(name,\*\*data): *# \*\*b means it can have multiple keyword ardument* **return** name, data  
print(**"2.1"**, person(**'manish'**,age=19,sex=**'male'**) ) *#having keyword  
#dic={"name":"manish", "data":"not available"}  
#print(\*\*dic)  
# COMINING ALL***def** all(i,j,\*args,x,y,\*\*kwargs):  
 print(i,j)  
 **for** var **in** args:  
 print(var,end=**","**)  
 print(x.y)  
 **for** name, value **in** kwargs.items():  
 print(name,value,end=**","**)  
print(10,20,100,200)  
*# DEFAULT VALUE.,6' ALREADY RECALLED IN FUNCTION***def** person(name,age=18):  
 **return** name,age-6  
print(**"1.2:"**, person(name=**'navin'** )) *#PUTTING AGE WILL OVERWRITE ASSIGNED VALUE  
# PANGRAM CHECKING***def** ispangram(s):  
 aplhaset={**"abcdefghijklmnopqrstuwxyz"**}  
 **return** aplhaset<=set(s.lower())  
print(ispangram(**" The quick brown fox jumps over the lazy dog"**))  
*# SORTING HYPHON SEPERATED SEQUENCE***def** sorting\_hyphon(s1):  
 item=[s **for** s **in** s1.split(**"-"**)]  
 item.sort()  
 s2=**"\_\_"**.join(item)  
 **return** s2  
print(**""**, sorting\_hyphon(**"manish-deepak-salman-"**))  
*# CHECKING PALINDROME***def** ispalindrome(s):  
 t=s.lower() *# IMMUTABLE STRING COLLECTED TO ANOTHER VARIABLE* left=0  
 right=len(t)-1  
 **while** right>=left:  
 **if** t[left]!=t[right]:  
 **return False  
 if** t[right]==**''**:  
 right-=1  
 **if** t[left]==**''**:  
 left+=1  
 left+=1  
 right-=1  
 **return True**print(ispalindrome(**"rats live on no eviln star"**))  
*# REMOVING DUPLICATE ITEMS***def** removing\_duplicate(s):  
 t=[ele **for** ele **in** s.split(**" "**) ]  
 **return " "**.join(sorted(list(set(t))))  
print(removing\_duplicate(**" ram is is good , good as it was never expected"**))  
*# COUNTING THE FREQUENCY OF WORD***def** frequency(s):  
 freq={}  
 **for** word **in** s.split():  
 freq[word]=freq.get(word,0)+11  
 **return** freq  
print(frequency(**"is he, what you think he is ?"**))  
*# CREATING SENTENCES***def** creatre\_sentence(a,b,c):  
 **return** [(x+**""**+y+**" "**+z) **for** x **in** a **for** y **in** b **for** z **in** c]  
print(creatre\_sentence([**"Ram"**,**"Shyam"**],[**"reads"**,**"plays"**],[**"books"**,**"nothing"**]))  
*########  
# RECURSION***def** prime\_factor(n):  
 i=2  
 **while** i<=n:  
 **if** n%i==0:  
 print(i)  
 n=n//i  
 **else**:  
 i+=1  
prime\_factor(17)  
*#########*print(**"==================================="**)  
  
  
  
  
  
*#############################  
# FUNCTIONAL PROGRAMING  
#####***def** prime\_factor(n):  
 i=2  
 **while** i<=n:  
 **if** n%i==0:  
 print(i)  
 n=n//i  
 **else**:  
 i+=1  
prime\_factor(17)  
**def** add(x,y,f):  
 print(x+y)  
 f()  
**def** fun():  
 print(**'Hey'**)  
f=fun  
a=add *# ASSIGNING A FUNCTION TO VARIABLE*a(1,2,f) *# PASSING FUNCTION AS ARGUMENT TO A FUNCTION  
  
# LAMBDA FUNCTION*print((**lambda** s:s.lstrip().rstrip().upper())(**"Manish"**))  
print((**lambda** s:s\*s\*s)(10))  
lst=[2,4,6]  
p=**lambda** l:sum(l)  
print((p)(lst))  
*# HIGH ORDER FUNCTIONS  
# 1.1 MAP***import** math  
print(list(map(math.factorial, lst)))  
*# FILTER***def** filt(n):  
 **if** n%5==0:  
 **return True  
 else**:  
 **return False**l=[10,22,30,2]  
lt=[**'A'**,**'B'**,2,3]  
print(list(filter(filt,l)))  
*# print(list(filter(str.isalpha,lt)) )  
# REDUCE***from** functools **import** reduce  
b=[4,8,9,10]  
**def** getsum(x,y):  
 **return** x+y  
print(reduce(getsum,b))  
*# USE IN ONE FUNCTION*print(reduce( **lambda** x,y:x+y,b)) *# INT OBJECT IS NOT ITRERABLE*print(list(filter(**lambda** n:n%5==0,b)))  
print(list(filter(**lambda** n:n%5==0,b)))  
print(**"1.2"**, list(filter(**lambda** n:n>10, map(**lambda** n:n\*n\*n,b) )))  
c=[1,2,3]  
d=[3,2,1]  
print(list(map(**lambda** n1,n2:n1+n2,c,d)))  
*######*print(**"==================================="**)  
  
  
  
  
  
*#############################  
# LOCAL AND GLOBAL VARIABLE  
#####*a=10 *#GLOBAL VARIABLE-work as outer***def** something():  
 a=3 *#LOCAL VARIABLE-of function* **return** a  
print(**"1.1:"**, a)  
print(**"1.2:"**, something())  
*#*a=10  
**def** something():  
 **global** a *# NO LOCAL VARIABLE- GLOBAL BECOMES LOCAL it work without writing* a=20  
 **return** a  
print(**"2.1:"**, something())  
print(**"2.2:"**, a)  
*# 1.3 CHANGING GLOBAL VARIABLE WITHOUT EFFECTING INNER VARIABLE*a=10  
**def** man():  
 a=12  
 x=globals()[**'a'**]  
 **return** x  
 globals()[**'a'**]=1  
print(**"3.1:"**, man())  
print(**"3.2:"**, a)  
*#*a=10  
**def** update(x):  
 print(id(x))  
 x=8  
 **return** id(x), x  
print(**"4.1:"**, update(a))  
print(**"4.2:"**, id(a),a)  
*#***def** update(lst):  
 lst[1]=25  
 **return** id(lst), lst  
lst=[10,20,30]  
print(**"5.1:"**, id(lst))  
print(**"5.2:"**, update(lst))  
*#######*print(**"==================================="**)  
  
  
  
  
*#############################  
# CLASS and OOPS CONCEPT  
#####  
# COSTRUCTOR= define the memory of the class***class** computer:  
 **pass**c=computer() *#COMPUTER ARE CONSTRUCTOR*c1=computer() *# contructor decide the memory of that class*print(**"1.1:"**, id(c))  
print(**"1.2:"**, id(c1))  
*#####***class** computer:  
 **def** config(self):  
 print(**"15gb,i3,8"**)  
com=computer  
print(**"2.1:"**, com)  
print(**"2.2:"**, type(com))  
computer.config(com)  
*######***class** compuetr:  
 **pass** *# nothing print so use pass*com=compuetr  
print(**"3.1:"**, id(com))  
print(**"3.2:"**, com)  
*# \_\_init\_\_(self)  
# 1.***class** computer:  
 **def** \_\_init\_\_(self):  
 print(**"4.1:"**, **"in init"**)  
 **def** config(self):  
 print(**"4.2:"**, **"i5,16gb,1TB"**)  
com1=computer()  
com2=computer()  
com1.config()  
com2.config()  
*# 2.***class** computer:  
 **def** \_\_init\_\_(self,cpu,ram):  
 self.cpu=cpu  
 self.ram=ram *#value we passed assigned to those object* **def** config(self):  
 print(**"5.1: confif is:"**, self.cpu, self.ram) *# WE PASSING SELF TO USE IT FETCH THE VALUE*com1=computer(**'i5'**,16)  
com2=computer(**'Man'**,8)  
com1.config()  
com2.config()  
*#####*print(**"==================================="**)  
  
  
  
*#######################  
# 3. HOW TO CREATE A VARIABLE IN OBJECT  
#####***class** computer:  
 **def** \_\_init\_\_(self,name,age):  
 self.NAME=name  
 self.AGE=age *#value we passed assigned to those object* **def** update(self):  
 self.AGE=30  
 **def** compare(self, other):  
 **if** self.AGE==other.AGE:  
 **return "They are of the same age:"  
 else**:  
 **return False**c1=computer(**"MANISH"**,20) *# changing age will not print same REMOVE IT(SAME AGE)*c2=computer(**"RAHUL"**,20)  
print(**"1.1:"**, c1.compare(c2))  
print(**"1.2:"**, c1.NAME)  
print(**"1.3:"**, c2.NAME)  
*#####*print(**"==================================="**)  
  
  
  
  
*#############################  
# 4.HOW TO CHANGE VARIABLE WITHOUT EFFECTING OTHER  
#####***class** car:  
 **def** \_\_init\_\_(self):  
 self.mil=10  
 self.com=**"Tyota"**c1=car()  
c2=car()  
c1.mil=15 *#TO CHANGE MILAGE*c2.com=**"BMW"** *#TO CHANGE COMPANY*print(**"1:"**, c1.mil, c1.com)  
print(**"2:"**, c2.mil,c2.com)  
*# 5.CHANGING VARIABLE EFFECTING WITH OTHER(INSTANCE AND CLASS VARIABLES)  
########***class** car:  
 wheel=4 *# CLASS VARIABLE DEFINED OUSIDE \_\_INIT\_\_* **def** \_\_init\_\_(self):  
 self.mil=10 *#BOTH ARE INSTANCE VARIABLE* self.com=**"Tyota"**c1=car()  
c2=car()  
c1.mil=15 *#TO CHANGE MILAGE*c2.com=**"BMW"** *#TO CHANGE COMPANY*car.wheel=5 *#WE HAVE TO USE CLASS(car) BECAUSE THAT IS CLASS VARIABLE IT EFFECTS BOTH*print(**"3:"**, c1.mil, c1.com, c1.wheel)  
print(**"4:"**, c2.mil,c2.com, c2.wheel)  
*#####*print(**"==================================="**)  
  
  
  
  
*#####################  
# 7.error====CLASS INSIDE THE CLASS  
#####***class** student:  
 **def** \_\_init\_\_(self,name, rollno,):  
 self.n=name  
 self.r=rollno  
 self.l=self.laptop()  
 **def** show(self):  
 print(self.n, self.r)  
 self.l.show()  
 **class** laptop:  
 **def** \_\_init\_\_(self):  
 self.brand=**"hp"** self.pro=**'i5'** self.sto=8  
 **def** show(self):  
 print(self.brand,self.pro,self.sto)  
  
s1=student(**"MANISH"**,19)  
s1.show()  
*#####*print(**"==================================="**)  
  
  
  
*#############################  
# 6. SET/GET==ACCESSOR/MUTATOR  
#####***class** student:  
 school=**"JNV"  
 def** \_\_init\_\_(self,m1,m2,m3):  
 self.m1=m1  
 self.m2=m2  
 self.m3=m3  
 **def** avg(self):  
 **return** (self.m1+ self.m2+ self.m3)/3  
 **def** total\_avg(self,\*b):  
 **return** (self.m1+ self.m2+ self.m3)/3+s2.avg()  
 **def** get\_m1(self): *#GETTER; FETCH THE VALUE=ACCESSOR* **return** self.m1  
 **def** set\_m1(self): *#SETTER:CHANGE THE VALUE=MUTATOR* **return** self.m1  
s1= student(15,15,60)  
s2= student(30,30,30)  
*#s3= student(50,30,10) #It should be corrected*print(s1.avg())  
print(s1.total\_avg(s2,s3))  
*#####*print(**"==================================="**)

ARRAY

*# ++++++++++++ ARRAY ++++++++++++++++#  
  
  
  
  
  
#############################  
# SOME WORK PERFORMED IN ARRAY  
#####***from** array **import** \* *# ARRAY IMPORTED*arr= array(**"i"**,[10,20,-30,23]) *# ‘i’,(+/-) both can be use but in case ‘I’ used then only +ve*arr.remove(10)  
print(**"1:"**, arr) *# REMOVE SELECTED ELEMENT FROM ARRAY*arr.reverse()  
print(**"2:"**,arr)  
print(**"3:occurence:"**, arr.count(10)) *# NUMBER OF RECURRENCE OF X*print(**"4:ADDRESS & SIZE:"**, arr.buffer\_info()) *# shows address and size of -BUFFER\_INFO ()*arr.append(100)  
print(**"5:"**,arr) *# ADD A NEW DIGIT IN END OF ARRAY*arr.reverse()  
print(**"6:"**,arr) *# reversing array's value*print(**"7:NO OF INDEX 0:"**,arr[0]) *# PRINT ONE VALUE FROM ARRAY BY USING INDEX NO.*print(**"8:TYPE:"**,array.typecode) *#shows type of array-“TYPECODE”*print(**"9:"**,array(**"i"**,[1,23])+array(**"i"**,[9,7])) *# MIXING TWO ARRAYS*print(**"10:"**,sum(array(**"i"**,[1,23])+array(**"i"**,[9,7]))) *# SUM OF ALL THE DIGIT IN ARRAY*newArr=array(arr.typecode, (a\*a **for** a **in** arr)) *# CHANGING ARRAY INTO NEW ARRAY*print(**"11"**, newArr)  
print(**"12:"**, arr.itemsize)  
  
*#array.pop([i])  
#print("13:", arr.byteswap())  
#####*print(**"==================================="**)  
  
  
  
  
  
*#############################  
# CHECK WHETHER ALL ELEMENTS OF THE ARRAY ARE SAME OR NOT  
#######***def** check(A):  
 *#return all(i==A[0] for i in len(A))* **return** len(list(A))<=1  
 *#return A.count(A[0])==len(A)  
 #return A and [A[0]] \* len(A) == A  
 #return A[1:]==A[-1:]*print(**"1.1:"**, check([1,1,1]))  
print(**"1.2:"**, check([1,2,1]))  
print(**"1.3:"**, check([**"a"**,**"b"**,**"c"**]))  
*######*print(**"==================================="**)  
  
  
  
  
  
  
*#############################  
# FILTER  
#####  
# 1 EVEN***def** is\_even(n):  
 **return** n%2==0  
nums=[2,3,4,4,5,5]  
print(**"1:"**, list(filter(is\_even, nums)))  
  
*# 1 .ODD***def** is\_odd(n):  
 **return** n%2!=0  
nums=[2,3,4,4,5,5]  
print(**"2:"**, list(filter(is\_odd, nums)))  
  
*# 3 .DOUBLE WITH EVEN*nums=[2,3,4,5,6,7,8,9]  
c=list(filter(**lambda** n:n%2==0,nums))  
doubles=map(**lambda** n:2\*n,nums)  
print(**"3:"**, c)  
  
*#COUNTING EVEN ODD NUMBER IN LST***def** count(lst):  
 even=0  
 odd=0  
 **for** i **in** lst:  
 **if** i%2==0:  
 even+=1  
 **else**:  
 odd+=1  
 **return** even,odd  
lst={10,20,3,4,5}  
even,odd =count(lst)  
print(even)  
print(odd)  
*#####*

IF-ELSE-WHILE-PASS-CONTINUE-BREAK-ELIF

*#++++++++++++ CONDITIONS ++++++++++++++++#  
  
  
  
  
  
  
#############################  
# IF/ELSE/ELIF  
####*x=5  
**if** x==1:  
 print(**"one"**)  
**elif** x==2:  
 print(**"two"**)  
**elif** x==3:  
 print(**"three"**)  
**else**:  
 print(  
*#####"false")*print(**"==================================="**)  
  
  
  
  
*#############################  
# Print anything n times.  
#####  
# A.WHILE LOOP  
# 1.*i=1  
**while** i<=5:  
 print(**"manish"**,i)  
 i=i+1  
*# 2.*i=5  
**while** i>=1:  
 print(**"manish"**,i)  
 i=i-1  
*#####  
# PRINT SOMENTHING BESIDE  
#####*i=1  
**while** i<=5:  
 print(**"manish"**,end=**"\*\*"**)  
 j=1  
 **while** j<=4:  
 print(**"dada"**,end=**"\*\*"**)  
 j=j+1  
 i=i+1  
 print()  
*#####*print(**"==================================="**)  
  
  
  
  
*#############################  
# PRINT SEQ OR SET LINE BY LINE  
#####*x=[10,**"ram"**,20]  
**for** i **in** x:  
 print(i)  
**for** i **in** range (1,20):  
 **if** i%5!=0: *# leaving digit divisible by five* print(i)  
*# 1 T0 100 EXCEPT DIVISIBLE BY 15***for** i **in** range(1,31):  
 **if** i%3==0 **and** i%5==0:  
 **continue** print(i)  
*#####*print(**"==================================="**)  
  
  
  
  
*#############################  
#WENDING MACHINE  
#####*av=5  
x=4  
i=1  
**while** i<av:  
 print(**"Candy"**)  
 **if** i>=av:  
 print(**"out of stock"**)  
 print(**"avilable cande is"**,av)  
 i=i+1  
print(**"Bye"**)  
*#####*print(**"==================================="**)  
  
  
  
  
*#############################  
# USE OF PASS-CONTINUE-BREAK  
#####***for** i **in** range(5):  
 **if** i==3:  
 **continue** *# leave 3 chance and process forward* print(**"hello"**,i)  
**for** i **in** range(5):  
 **if** i==3:  
 **break** *# its leave 3 chance and end further process* print(**"GOOD"**,i)  
**for** i **in** range(1,11):  
 **if** i%2!=0: *# use= on placo of !to print only odd nnnmber* **pass** *# PASS =... OR NO-OP INSTRUCTION* **else**:  
 print(i)  
print(**"bye"**)  
*#####*print(**"==================================="**)  
  
  
  
  
*#############################  
# PRINT DESIRED DIVISIBLE NUMBERS FROM ARRAY  
#####***for** i **in** range(6):  
 **for** j **in** range(8):  
 print(**"#"**,end=**"\*"**) *#end="" to print # in same line* print() *#to print # into another line*nums=[10,58,67,78,80,76]  
**for** i **in** nums:  
 **if** i%5==0:  
 print(i)  
 **break  
else**:  
 print(**"not found"**)  
*# IDENTIFY PRIME*num=10  
**for** i **in** range(2,num):  
 **if** num%i==0:  
 print(**"Not prime"**)  
 **break  
else**:  
 print(**"Prime"**)  
*#####*print(**"==================================="**)  
  
  
  
  
*#############################  
# TAX CALCULATION HAVING DIFFERENT VALUE OF TAX AT DIFFERENT INCOME  
#####*n=100000  
t\_percent=30  
**if** n<=15:  
 **if** n>=12.5:  
 t\_percent=25  
 **if** n>10 **and** n<12.5:  
 t\_percent=20  
 **if** n>7.5 **and** n<10:  
 t\_percent=15  
 **if** n>5 **and** n<7.5:  
 t\_percent=10  
 **if** n>2.5 **and** n<5:  
 t\_percent=5  
 **if** n<2.5:  
 t\_percent=0  
tax= n\*t\_percent/100  
print(**"TAX:"**,tax,**'lakh'**)  
t\_Oldpercent=30  
**if** n<=15:  
 **if** n>12.5:  
 t\_Oldpercent=30  
 **if** n>10 **and** n<12.5:  
 t\_Oldpercent=30  
 **if** n>7.5 **and** n<10:  
 t\_Oldpercent=20  
 **if** n>5 **and** n<7.5:  
 t\_Oldpercent=20  
 **if** n>2.5 **and** n<5:  
 t\_Oldpercent=5  
 **if** n>2.5:  
 t\_Oldpercent=0  
Oldtax=n\*t\_Oldpercent/100  
print(**"Old Tax:"**,Oldtax)  
diftax=tax-Oldtax  
print(**"Difference in tax="**,diftax)  
*#####*print(**"==================================="**)  
  
  
  
  
*#############################  
# ELSE/IF/ELIF-USE IN FAMILY TOUR PLANNING DECISION  
#####*print(**"0=NO AND 1=YES"**)  
f= int (input (print(**"Mr ROY:"**))) *#Mr Roy ’s choice*m = int (input (print(**"Mrs Roy:"**))) *#Mrs Roy ’s choice*g = int (input (print(**"Mrs Roy's Mother:"**))) *#Mrs Roy ’s mother ’s choice*d = int (input (print(**"Daughter:"**))) *#Daughter ’s choice  
#METHOD ONE***if** f == 1 **and** m == 1:  
 print(**"GO"**)  
**elif** f==1 **and** m==1 **and** g==1:  
 print(**"Go"**)  
**elif** f == 1 **and** m == 1 **and** d == 1:  
 print(**"GO"**)  
**elif** m == 1 **and** d == 1 **and** g == 1:  
 print(**"GO"**)  
**elif** f==1 **and** d==1 **and** g==1:  
 print(**"GO"**)  
**elif** f == 1 **and** m == 1 **and** g == 1 **and** d == 1:  
 print(**"GO"**)  
**else**:  
 print(**"0"**)  
*#METHOD TWO***if** f==1 **and** m==1:  
 print(**"GO"**)  
**else**:  
 **if** f==1 **and** m==1 **and** g==1:  
 print(**"Go"**)  
 **else**:  
 **if** f==1 **and** m==1 **and** d==1:  
 print(**"GO"**)  
 **else**:  
 **if** m==1 **and** d==1 **and** g==1:  
 print(**"GO"**)  
 **else**:  
 **if** f==1 **and** d==1 **and** g==1:  
 print(**"GO"**)  
 **else**:  
 **if** f==1 **and** m==1 **and** g==1 **and** d==1:  
 print(**"GO"**)  
 **else**:  
 print(**"0"**)  
  
*#####*print(**"==================================="**)  
  
  
  
  
*#############################  
# PRINTING ALL THE NON-REPEATING SEQUENCE FORMED BY DIGITS-WAYS OF NON REPEATING ARRANGEMENT  
######*A=[1,2,3]  
**for** i **in** A:  
 print(**"i"**, i)  
 **for** j **in** A:  
 print(**"j"**, j)  
 **for** k **in** A:  
 print(**"k"**, k)  
 **if** i!=j **and** j!=k **and** k!=i:  
 print(i,j,k)  
i=1  
**while** i<=3:  
 j=1  
 **while** j<=3:  
 k=1  
 **while** k<=1:  
 **if** i==j **or** j==k **or** k==i:  
 **continue  
 else**:  
 print(i,j,k)  
 k+=1  
 j+=1  
 i+=1  
*#####*print(**"==================================="**)  
  
  
  
  
*#############################  
# PRINT SEQ OR SET LINE BY LINE  
#####  
#####*print(**"==================================="**)  
  
  
  
  
*#############################  
# PRINT SEQ OR SET LINE BY LINE*

MATHEMATICAL OPERATION

*# ++++++++++++ MATHEMATICAL OPERATION ++++++++++++++++#  
  
  
  
  
  
#############################  
# ARITHMETIC OPERATIONS  
######*a=10  
b=2  
a+=b *# a=a+b*print(**"1:"**,a) *# a=12*a-=b *# a=a-b*print(**"2:"**,a) *# a=10*a/=b *# a=a/b*print(**"3:"**,a) *# a=5*a\*=b *# a=a.b*print(**"4:"**,a) *# a=10*a\*\*=b *# a=a^b*print(**"5:"**,a) *# a=100*a//=b *# FLOOR DIVISION--LARGEST INTEGER<=QUOTIENT*print(**"6:"**,a) *# a=50*a%=7 *# a-(b\*(a//b))*print(**"7:"**,a) *# a=1  
#####*a=10  
b=3  
print(**"1.1:"**, b+a)  
print(**"1.2:"**, b-a)  
print(**"1.3:"**, a/b) *# FLOOR DIVISION # GIVES EXACT FLOST VALUE OF QUOTIENT AFTER DEVISION*print(**"1.4:"**, a//b) *# GIVES INTEGER VALUE LOWER OR EQUAL OF QUOTIENT AFTER DEVISION*print(**"1.5:"**, a%b) *# a-(b\*(a//b)) # GIVES REMAINDER AFTER DEVISION*print(**"1.6:"**, a\*\*b)  
print(**"1.7:"**, a\*b)  
print(**"1.8:"**, 10//3\*\*2%5/2)  
print(**"1.9:"**, 2-10//3\*\*4%5/2+10) *# PEMDAS*print(**"1.9:"**, -10//3,10//-3,-10//-3)  
print(**"2.0:a-(b\*(a//b)):"**, -10%3, 10%-3, -10%-3)  
*#####*print(**"==================================="**)  
  
  
  
  
  
*##########################  
# CONVERSIONS*print(**"1:"**,int(23.3))  
print(**"2:"**,float(20))  
print(**"3:"**,bool(20))  
print(**"4:"**,str(100))  
print(**"5:"**,chr(32))  
print(**"6:"**,complex(23.3,10))  
print(**"7:"**, int(**True**))  
print(**"8:"**, int(**"100"**, 8), int(**'10'**, 2)) *# OCTAL/BINARY TO DECIMAL INT*n=23.1  
print(n)  
print(**"9: %d"**%(float(n))) *# CHANGING NUMBER INTO INTEGER*print(**"1.0: %.2f"**%(float(n))) *# A DECIMAL NUMBER HAVING TWO PLACE AFTER DECIMAL*print(**'1.1: %0.5d'**%(float(n))) *# INTEGER HAVING FIVE DIGIT LEADING WITH ZERO IF NEEDED*print(**'1.2: '**, round(float(n))) *# ROUND OF TO NEAR*n=str(**"13.45"**)  
print(**'1.2:'** ,int(n[0])) *# RETURNS FIRST DIGIT OF THE NUMBER (STRING)  
#####  
# APPLICATION*print(**"manish yadav"**+str(23.4))  
*#####*print(**"==================================="**)  
  
  
  
  
  
*#############################  
# ROOT VIA BINARY SEARCH  
####***def** nthroot(x, n):  
 x = float(x)  
 n = int(n)  
 low = 1  
 high = x  
 let = (low + high) / 2  
 **while** abs(let \*\* n - x) >= 0.00001:  
 **if** let \*\* n > x:  
 high = let  
 **else**:  
 low = let  
 let = (low + high) / 2  
 **return** let  
print(**"1:"**,nthroot(9,2)) *# x=input(print("Enter a number x>=1:")) # n=input(print("Enter a number n>0:"))  
# VIA SIMPLE FUNCTION***def** nthroot(x, n):  
 x = float(x)  
 n = int(n)  
 z=x\*\*1/n  
 **return** z  
print(**"2:"**, nthroot(8,2))  
*######*print(**"==================================="**)  
  
  
  
  
*#############################  
# FACTORIAL  
#####  
# 1.1***def** fac1(n):  
 **if** n==0:  
 **return** 1  
 **return** n\*fac1(n-1) *# RETURN BACK TO THIS POINT AGAIN WHENEVER 0 CONDITION WILL NOT REACH*print(**"1.1:"**,fac1(3))  
*# 1.2***def** fac2(n):  
 f=1  
 **for** i **in** range(1,n+1):  
 f=f\*i  
 **return**(f)  
print(**"1.2:"**,fac2(3))  
*######*print(**"==================================="**)  
  
  
  
  
*#############################  
# SQUARE OF A NUMBER  
#####  
# 2.1***def** sq(a):  
 c=a\*a  
 **return** c  
print(**"2.1:"**,sq(4))  
*# 2.2***def** sq(a):  
 **return** a\*a  
print(**"2.2:"**, sq(4))  
*# 2.3*f=**lambda** a:a\*a  
print(**"2.3:"**, f(9))  
*######*print(**"==================================="**)  
  
  
  
  
*#############################  
# FIBONACCI SEQUENCE FIND  
#####***def** fib(n):  
 a=0  
 b=1  
 **if** n==1:  
 print(a)  
 **else**:  
 print(a)  
 print(b)  
 **for** i **in** range(2,n):  
 c=a+b  
 a=b\*n  
 b=c  
 **return** c  
print(**"3.1:"**, fib(4))  
*######*print(**"==================================="**)  
  
  
  
  
*#############################  
# DIVISION  
#####***def** div1(a,b):  
 **if** a<b:  
 a,b=b,a  
 **return** a/b  
print(**"1:"**, div1(2,4))  
**def** div(a,b):  
 print(a/b)  
**def** smart\_div(func):  
 **def** inner(a,b):  
 **if** a<b:  
 a,b=b,a  
 **return** func(a,b)  
 **return** inner  
div=smart\_div(div)  
div(2,4)  
*######*print(**"==================================="**)  
  
  
  
  
*#############################  
# ADDITION  
#####  
  
# 1.***def** add\_sub(x,y):  
 c=x+y  
 d=x-y  
 **return** c, d  
result1,result2=add\_sub(5,4)  
print(result1,result2)  
*# 2.***def** add(x,y):  
 c=x+y  
 print(c)  
add(4,3)  
*# 3.***def** update(x):  
 print(id(x))  
 x=8  
 print(**"x"**,x)  
a=10  
update(a)  
print(id(a))  
print(**"a"**,a)  
*######*print(**"==================================="**)  
  
  
  
  
  
*#############################  
# BUILT IN FUNCTIONS IN MATH MODULE  
#####*print(**"1.1:"**, abs(-2.1))  
print(**"1.2:"**, pow(3,2))  
print(**"1.3:"**, min(2,3,1,10))  
print(**"1.4:"**, max(2,3,1,10))  
print(**"1.5:"**, divmod(4,2)) *# RETURN A//B AND A%B*print(**"1.6:"**, bin(10))  
print(**"1.7:"**, hex(10))  
print(**"1.8:"**, oct(10))  
print(**"1.9:"**, **"a:"**, round(2.59), **"b:"**, round(2.59851, 3))  
*#####*print(**"==================================="**)  
  
  
  
  
  
  
*# ++++++++++++ MATH MODULE++++++++++++++++#  
  
  
  
  
  
#############################  
# ARITHMETIC OPERATIONS  
######***from** math **import** \*  
*# MATHEMATICAL FUNCTION*print(**"1.0:"**, pi,e)  
print(**"1.1:"**, sq(2))  
print(**"1.2:"**, sqrt(4))  
print(**"1.3:"**, fac1(4), fac2(4), factorial(4))  
print(**"1.4:"**, fabs(-2.29) ) *# ABSOLUTE VALUE*print(**"1.5:"**, log10(10), log(10), log1p(10), log2(10) )  
print(**"1.6:"**, exp(0), expm1(0), frexp(0), )  
print(**"1.7:"**, trunc(5.9) ) *#*print(**"1.8:"**, ceil(-2.29) ) *# SMALLEST INTEGER>=X*print(**"1.9:"**, floor(-2.29) ) *# LARGEST INTEGER<=X*print(**"2.0:"**, modf(-2.29) ) *#  
# TRIGNOMENTRIC FUNCTION*print(**"2.1:"**, degrees(1) ) *# DEGREE TO RADIAN*print(**"2.2:"**, radians(180) ) *# RADIAN TO DEGREE*print(**"2.3:"**, sin(90), cos(90), tan(1) ) *# FUNCTION OF X RADIAN*print(**"2.4:"**, sinh(1), cosh(1), tanh(1) ) *# HYPERBOLIC SINE OF X*print(**"2.5:"**, asin(1), acos(1), atan(1) ) *# SINE INVERSE OF X, IN RADIAN  
#print("2.6:", asinh(1), acosh(1), atanh(1) ) #*print(**"2.7:"**, hypot(2,10) ) *# SQRT(X\*X+Y\*Y)  
##########*

NUMPY

*#++++++++++++ NUMPY ++++++++++++++++#  
  
  
  
  
#############################  
#CREATING DSEIRE NUMBER OF MATRIX  
####***from** numpy **import** \*  
print(**"1:"**,sqrt(array([16,36,64,80])))  
print(**"2:"**,array([10,20,30])+array([10,20,30])) *# adding two array*print(**"3:"**,array([16,36,64,80])+5) *# Adding single digit to whole array*print(**"4:"**,zeros(5)) *#ZEROS/ONES*print(**"5:"**,ones(5)) *# ONES*print(**"6:"**,arange(1,15,2)) *# ARRANGING ARRAY FROM1 TO 15 HAVING 2 GAP*print(**"7:"**,logspace(1,15,2))  
print(**"8:"**,sin(array([16,36,64,80]))) *# GIVES SIGN VALUE OF ARRAY*arr1=array([16,36,64,80])  
print(**"9:"**,arr1.dtype) *# PRINT DATA TYPE INT/FLOAT*arr2=array([1,3,5],float) *# CHANGE ARRAY TYPE*print(**"10:"**,arr2.dtype)  
print(**"11:"**,arr2)  
arr2=array(arr2,int)  
print(**"12:"**,concatenate([arr1,arr2])) *# THAT ALSO ADD ARRAY  
######*print(**"==================================="**)  
  
  
  
  
*#############################  
# DEEP COPY***from** numpy **import** \*  
arr1=array([10,20])  
arr2=arr1.copy()  
print(arr1)  
arr1[1]=7  
print(arr2)  
print(id(arr1))  
print(id(arr2))  
*# SHALLOW COPY***from** numpy **import** \*  
arr1=array([10,20])  
arr2=arr1.view()  
arr1[1]=7  
print(arr1)  
print(arr2)  
print(id(arr1))  
print(id(arr2))  
*######*print(**"==================================="**)  
  
  
  
  
*#############################  
# MATRIX  
#####***from** numpy **import** \*  
arr=array([[1,2,3,8,5,6],[4,9,0,5,6,2]])  
print(**"1:"**,arr.shape)  
print(**"2:"**,matrix([[1,2,3,8,5,6],[4,9,0,5,6,2]]))  
print(**"3:"**,matrix(**'1,2;4,5,;9,8'**))  
print(**"4:ARRAY SIZE="**, arr.size) *#SIZE-GIVES TOTAL ELMENT OF BOTH*print(**"5:"**,matrix(**'1,4;4,5,'**)\*matrix(**'1,4;4,5,'**))  
print(**"6:"**,arr.flatten()) *# FLATTEN-PRINT TWO ARRAYS IN SINGLE*print(**"7:"**,arr.reshape(3,4)) *# MATRIX OF THREE ROW AND FOUR COLOUMN*print(**"8:"**,arr.reshape(2,2,3)) *# TWO MATRIX EAXH WITH TWO ROW AND THREE COLOUMN  
######*

STRING-DICTIONARY-TUPLE-LIST

*# ++++++++++++ STRING-DICTIONARY-TUPLE-LIST ++++++++++++++++#  
  
  
  
  
  
  
  
#############################  
# STRING  
#####*p=**"codeblocks"**string=**'youtube'**print(**"1.0:"**, **'string'**+p)  
print(**"1.1:"**, string[-2])  
print(**"1.2:"**, string[0:])  
print(**"1.3:"**, string[1:5], p[0:9:2])  
print(**"1.4:"**, string[2:]+**'p'**)  
print(**"1.5:"**, **'A''B'**)  
print(**"10z.6:"**, **'ce' in 'race'**)  
print(**"1.7:"**, **'-'**\* 10, **'MANISH'**\*5)  
*# CONVERSION*print(**"1.8:"**, **"MANISH yadav"**.upper(), p.upper())  
print(**"1.9:"**, **"Hello"**.lower(), p.lower())  
print(**"2.0:"**, **"manish"**.capitalize())  
print(**"2.1:"**, **"MAN123@,/=9ish"**.swapcase()) *# SWAP LOWER CASE LETTERS INTO UPPERCASE AND VICE-VERSA  
# CONTENT TEST FUNCION*print(**"2.2:"**, **"hello"**.islower(), **"HELLO"**.isupper()) *# CHECKS ALL CHARACTERS ARE LOWER/UPPER CASE*print(**"2.3:"**, **"12HELLO"**.casefold() ) *# CHECKS ALL CHARACTERS ARE*print(**"2.4:"**, **"Hello"**.isalpha()) *# CHECKS ALL CHARACTERS ARE*print(**"2.5:"**, **"Hello23"**.isascii()) *#*print(**"2.6:"**, **"Hell123"**.isalnum()) *# ALPHANUMERIC--CHECKS ALL CHARACTERS ARE DIGITS OR ALPHABETS*print(**"2.7:"**, **"123HEKL"**.isdigit()) *# CHECKS ALL CHARACTERS ARE DIGITS*print(**"2.8:"**, **"MANISH"**.startswith(**"MA"**)) *# CHECKS IF STARTS WITH A VALUE RESPECTIVELY*print(**"2.9:"**, **"MANISH"**.endswith(**'ISH'**)) *# CHECKS IF END WITH A VALUE RESPECTIVELY  
# SEARCH AND REPLACE*print(**"3.0:"**, **"Manish\_\_Yadav"**.split(**'\_\_'**)) *# ['Manish', 'Yadav']*print(**"3.1:"**, **"Manish\\Yadav"**.partition(**'\\'**)) *# ('Manish', '\\', 'Yadav')*print(**"3.2:"**, **"ABC"**<**"ABD"**, **"ABC"**<**"B"**, **"e"**<**"ABC"**) *# ALPHABETIC ORDER CHECKED, LOWERCASE>UPPERCASE*print(**"3.3:"**, ord(**'A'**), ord(**'B'**), ord(**'Y'**), ord(**'Z'**)) *# GIVES UNICODE VALUE*print(**"3.4:"**, chr(65), chr(1), chr(2), chr(90) )  
print(**"3.5:"**, **"MANish YADav"**.lstrip())  
print(**"3.6:"**, **"MANish YADav"**.rstrip())  
print(**"3.7:"**, **"ManishYadav"**.find(**'Y'**) )  
print(**"3.8:"**, **"ManishYadav"**.replace(**'Yadav'**,**'Kumar'**))  
print(**"3.9:"**, len(**'MANISH'**))  
print(**"4.0:"**, **'MANISHYADAV'**.count(**'A'**))  
x=**"MANISH"**print(**"4.1:"**, [\*x])  
*# MULTILINE STRING*msg1=**"good"**msg1=msg1+**'man'**print(msg1)  
print(**'He said,\'are you ok?\'.'**, **'PyPy:\\D\\drive'**) *# ESCAPE SEQUENCE  
# RAW STERING PREPEND R*print(**r'PyPy:\D\drive'**)  
print(**'Go,\ run'**)  
print(**"""See, that's me."""**)  
print((**"What!" "that's done"**))  
*#####*print(**"==================================="**)  
  
  
  
  
  
*#############################  
# DICTIONARY  
#####*dictionary={**'navin'**:**'samsng'**,**'manish'**:**'redmi'**}  
dic={**'manish'**:{**"phone"**:**'samsng'**,**"colour"**:**'blue'**},**"deepak"**:{**"phone"**:**'redmi'**,**"colour"**:**'black'**}}  
print(dic)  
d=dictionary  
**if** bool(d): *# IF NOT BOOL(D)* print(**"dic is not empty"**)  
print(**"1:"**, d)  
print(**"2:"**, d.keys()) *# dictionay.keys()*print(**"3:"**, d.values()) *# RETURNS VALUE SAMSANG AND REDMI*print(**"4:"**, d[**'manish'**]) *# RETURNS VALUE OF KEY MANISH*print(**"5:"**, d.get(**'navin'**)) *# RETURNS VALUE OF KEY MANISH*dictionary[**"Deepak"**]=**"realmi"** *# ADD NEW KEY-VALUE*print(d.get(**"rahul"**, **"anything"**)) *# RETURNS VALUE ANYTHING IF THE GIVEN KEY IS NOT IN THRE DIC*dictionary[**"manish"**]=**"redminote 7s"** *# MODIFYT THE VALUE FOR A KEY***del**(d[**"navin"**]) *# DELETE KEY-VALUE,*print(d)  
print(len(d))  
print(max(d)) *# CHECKS ALPHABETICAL ORDER OF KEYS*print(min(d))  
print(**"manish" in** d) *# CHECKS WHETHER THIS KEY/VALUE IS IN D OR NOT*lst=[2,3,4,4,4,4,4,4,4,4] *# DICTIONARY DOESN'T CONTAIN DUPLICATE ITEMS*a=dict.fromkeys(lst,25) *# IT SETS VALUE 25 TO EACH KEY*print(a)  
dic={**'manish'**:{**"phone"**:**'samsng'**,**"colour"**:**'blue'**},**"deepak"**:{**"phone"**:**'redmi'**,**"colour"**:**'black'**}}  
print(dic)  
print(dic.values())  
**for** v **in** dic.values():  
 print(v[**"phone"**])  
*# WAYS OF RECALLING DICTIONARY*d1={**"a"**:1,**"b"**:2}  
d2={**"y"**:25,**"z"**:26}  
d={}  
  
**for** i **in** (d1,d2):  
 d.update(i)  
print(d)  
d3={\*\*d1,\*\*d2}  
print(d3) *# UNPACKS DICTIONARY*print(\*d1,\*d2) *# UNPACKS VALUES***for** a,b **in** d.items(): *# RETURNS KEYS AND VALUES* print(a,b)  
**for** k **in** d.keys(): *# RETURNS KEYS* print(k)  
**for** v **in** d: *# RETURNS KEYS* print(v)  
**for** v **in** d.values(): *# RETURNS VALUE* print(v)  
*#del(d) # delete all  
#dictionary.clear()*dic={**"A"**:3,**"B"**:2,**"C"**:1}  
print(max(list(dic.keys())))  
print(dic[max(dic.keys(),key=(**lambda** k: dic[k]))])  
*#####*print(**"==================================="**)  
  
  
  
  
  
  
*#############################  
# SET  
#####*set={1,**"MANISH"**,2,**"DEEPAK"**,3,**"CHANDAR"**} *# 2*print(**"1.2:SET:"**,set) *# SET IS NOT COLLABLE*set.remove(**"CHANDAR"**)  
print(**"3.2:SET:CHANDAR IS REMOVED:"**,set) *# tup.remove(30)*set.add( **"CHANDAR"**) *# tup.add(30) # # set.add[2, 35]*print(**"4.2: CHANDAR ADDED:"**,set)  
print(**"5.2(LEN SET):"**,len(set))  
print(**"7.2:"**,{**'nums'**+**'values'**})  
print(**"9.2:"**, set.pop()) *#print("9.1:", tup.pop(1))*print(**"12.2:"**, sum({2,3,10}))  
*# print({'navin','kiran','john'}+{3,4})*s={**"ram"**, **"good"**}  
a={1,21}  
print(s.update(a))  
s.discard(**"ram"**)  
s.remove((1))  
print(s)  
s.clear()  
print(s)  
*# MATHEMATICL SET OPERATION*x={50,60,70,80}  
y={60,7,1}  
print(x.isdisjoint(y))  
print(y.issubset(x))  
print(x.issuperset(y))  
print(x|y) *# UNION*print(x>=y) *# PRINT TRUE IF X IS SUPERSET OF Y*print(x<=y) *# PRINT TRUE IF X IS SUBSET OF Y*print(x & y) *# INTERSECTION*print(y-x) *# DIFFERENCE*print(x^y) *# SYMMETRIC DIFFERENCE*x|=y *# UPDATE A TO A|B, lIKE THAT, x&=y, x^=y,x-=y*print(x)  
  
  
  
  
*#####*print(**"==================================="**)  
  
  
  
  
*#############################  
# TUPLE  
####*tup = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 *# OR (1,2,3,4,5,6,7,8,9,10) # ONE ITEM, A=(10,) AND ZER ITEM A=()  
# x=([4,1,7],"ram") IT ALSO A TUPLE*print(**"1:TUP:"**, tup)  
print(**"2(LEN TUP):"**, len(tup))  
print(tup[1]) *# CAN BE INDICES...  
# tup[0]=12 SHOWS ERROR SINCE IT'S IMMUTABLE*print(tup[3:7]) *# SLICED*print(tup.index(10))  
print(max(tup), min(tup))  
print(sum(tup), sorted(tup))  
print(tuple(reversed(tup))) *# REVERSED*print(tuple(tup[::-1])) *# REVERSED*n=**"MANISH"**print(\*n)  
  
  
print(tuple([1, 2]),tuple({10, 20})) *# TUPLE FUNCTION-SET/LIST TO TUPLE*print(tuple((x,x\*\*2) **for** x **in** range(5)))  
print(**"3:"**, (**'navin'**, **'kiran'**, **'john'**) + (10, 20))  
print(**"4:"**, (**'nums'** + **'values'**))  
print(**"5:"**, sum(tup))  
  
X=(10,20,30)  
Y=(30,20,10)  
a=(X,Y)  
print(a[0][0])  
print(2,X,3,Y)  
print(2,\*X,2)  
*# DISPOSABLE VARIABLES*tup1=(2,14,22,48,23)  
x,\_,\_,\_,y=tup1  
i,\*\_,j=tup1 *# INTO SINGLE DISPOSABLE VARIABLES*print(x,y,\_)  
print(i,j,\_)  
*#*lst=(**"a"**,**"b"**,(**"d"**),**"q"**)  
**for** i **in** lst:  
 **if** isinstance(i,tuple):  
 print(i)  
details = ((**"Manish"**, 19, **"m"**), (**"Reena"**, 18, **"f"**))  
**for** n, a, s **in** details:  
 print(n,a,s)  
*######*print(**"==================================="**)  
  
  
  
  
  
*#############################  
# PRINT SEQ OR SET LINE BY LINE  
#####  
  
#####*print(**"==================================="**)  
  
  
  
  
*#############################  
# LIST  
#####*list=[1,**"MANISH"**,2,**"DEEPAK"**,3,**"CHANDAR"**]  
print(**"1.3:LIST"**,list)  
print(**"2.3:list[1]"**,list[1])  
list.remove(**"CHANDAR"**)  
print(**"3.3:LIST: CHANDAR IS REMOVED"**, list)  
list.append(**"CHANDAR"**)  
print(**"4.3:CHANDAR APPENDED:"**,list)  
print(**"5.3(LEN LIST):"**,len(list))  
print(**"6.3:LIST ADDED:"**,[1,2]+[5,6])  
print(**"7.3:"**,[**'nums'**+**'values'**])  
list.insert(2,77)  
print(**"8.3: LIST insert(2,77):"**, list) *# print("8.2:",set.insert(2,77)) # print("8.1:",tup.insert(2,77))*print(**"9.3:"**, list.pop(2)) *# it removed number at index 2...pop() removed lastt element...leftpop() removed left*print(list)  
**del** list[2:]  
print(**"10.3: "**, list)  
list.extend([10,20,30])  
print(**"11.3::"**, list)  
*#print("12.3:", sum(list)) # max(list) # sum(list)*a=[2,4,1,5,2,10]  
a.sort() *# set.sort() tup.sort() ---NO SORT PERFORMED*print(**"13.3:"**, a)  
**del**(list[2:4]) *# NOT del(set[2:4]) del(tup[2:4])*print(**"13.4:"**, list)  
list[:]=[] *# DELETE ELEMENTS ...NOT set[:]=[] tup[:]=[]*print(**"13.5"**, list)  
list[0:3]=10,20,30,40  
print(**"13.5"**,list)  
**del** list  
print(**"13.5"**,list)  
*#*x=[**"a"**,**"b"**,**"c"**]  
y=[9,10,11]  
c=x+y  
print(**" "**,c[0][0])  
*# SHALLOW COPY # CHANGING ONES VALUE CHANGES THE OTHER*lst1=[1,2,3]  
lst2=lst1  
lst1[1]=5  
print(**"1"**, lst1, lst2)  
*# DEEP COPY # CHANGING ONES VALUE DOESN'T CHANGE THE OTHER*lst1=[1,2,3]  
lst2=[]  
lst2=lst2+lst1  
lst1[1]=5  
print(**"2"**, lst1, lst2)  
*# ADDING TWO MATRIX*m1=[[1,1,1],[2,2,2],[3,3,3]]  
*#print(len(m1))  
#print(len(m1[0]))  
#print(m1[1][2])*m2=[[3,3,3],[2,2,2],[3,3,3]]  
m= [[0,0,0],[0,0,0],[0,0,0]]  
**for** i **in** range(len(m1)):  
 **for** j **in** range(len(m1[0])):  
 m[i][j]=m1[i][j]+m2[i][j]  
print(m)  
*#####*print(**"==================================="**)  
  
  
  
  
*#############################  
# COMPREHENSION  
#####  
# LIST  
# a=[random.randint(10,20) for n in range(5)]*print(**"1.1:"**, [(x,x\*\*2,x\*\*3) **for** x **in** range(5)])  
print(**"1.2:"**, [int(x) **for** x **in** [**"10"**,**"2"**, **"1"** ]] )  
print(**"1.3:"**, [n **for** n **in** range(5,10) **if** n%2!=0])  
print(**"1.4:"**, [ n **for** n **in** range(15,35) **if** n<20 **or** n>30])  
print(**"1.5:"**, [ **"!" if** alphabet **in "aeiou" else** alphabet **for** alphabet **in "Techincal"**])  
print(**"1.6"**, [a+b **for** a **in** [1,2,3] **for** b **in** [3,2,1]])  
print(**"1.7:"**, [[a+b **for** a **in** [1,2,3]] **for** b **in** [3,2,1]])  
print(**"1.8:"**, [(i,j,k) **for** i **in** [1,2,3] **for** j **in** [1,2,3] **for** k **in** [1,2,3] **if** i!=j **and** j!=k **and** k!= i])  
arr=[[1,2,3],[4,5,6]]  
print(**"1.9:"**, [ n **for** ele **in** arr **for** n **in** ele] )  
print(**"2.0:"**, [\*arr[0],\*arr[1]])  
mat1,mat2=[[1,2,3,4],[5,6,7,8]],[[8,7,6,5],[4,3,2,1]]  
print(**"2.1:"**,[[mat1[i][j]+mat2[i][j] **for** j **in** range(len(mat1[0])) **for** i **in** range(len(mat1))]])  
lst=[(),10,(**""**),20,(1,2),3]  
print(**"2.2:"**,[tpl **for** tpl **in** lst **if** tpl]) *# TPL FUNCTION RETURNS TRUE IF TUPLE IS NOT EMPTY*s=**' do dreams otherwise you will never fail'**print(**"2.3:"**, [**""**.join(w.capitalize() **for** w **in** s.split())])  
print(**"2.4:"**, {**""**.join(alpha **for** alpha **in** k **if** alpha **not in "aeiou"**):v **for** (k,v) **in** d.items()})  
  
*# SET*print(**"2.5:"**, { x\*\*3 **for** x **in** range(1,5)})  
print(**"2.6:"**, { n **for** n **in** range(15,35) **if** n<20 **or** n>30})  
  
*# DICTIONARY COMPREHENSION*d={**"x"**:1,**"y"**:2,**"z"**:3}  
print(**"1.1:"**, {p:v\*10 **for** (p,v) **in** d.items()})  
print(**"1.2:"**, {k:(**"even" if** v%2==0 **else "odd"**) **for** (k,v) **in** d.items()})  
print()  
students={**"A93"**:{**"name"**:**"manish"**, **"phone"**:6393241779}, **"B94"**:{**"name"**:**"krma"**, **"phone"**:8787085500}}  
print(**"1.3:"**,students[**"A93"**][**"name"**])  
print(**"1.4:"**,{ k:v **for** (k,v) **in** students.items() **if** k.startswith(**"A"**)})  
print(**"1.5:"**,{k:v[**"name"**] **for** (k,v) **in** students.items() **if** v[**"name"**].startswith(**"m"**)})  
*#######*print(**"==================================="**)

SYS-MODULE

*# ++++++++++++ SYS-MODULE ++++++++++++++++#  
  
  
  
  
#############################  
# SETRECURSSION/GETRECURSSION-INCREASE LIMIT OF WRITING WORD  
####***import** sys  
sys.setrecursionlimit(2000)  
print(sys.getrecursionlimit())  
i=1  
**def** fun():  
 **global** i  
 i+=1  
 print(**"hello"**,i)  
 fun()  
fun()  
*######*

TURTLE-MODULE

*#++++++++++++ TURTLE ++++++++++++++++#  
  
  
  
  
  
#############################  
# TURTLE MODULE AND COADING  
#####***import** turtle  
t=turtle.Turtle()  
turtle.bgcolor(**"yellow"**) *# CHANGES BACKGROUND COLOUR*t.speed(1) *# CONTROL SPEED OF TURTLE*t.fd(100) *# MOVE FORWARD*t.color(**'blue'**) *# TURTLE COLOR START FROM WHERE YOU PUT IT*t.bk(20) *# MOVE BACK ON SAME PATH*t.rt(70) *# CHANGE ANGLE TO RIGHT*t.fd(-200) *# OPPOSITE DIRECTION OF REAL EXPECTATION*t.seth(to\_angle=-90) *# SET TURTLE HEADING INTO GIVEN DIRECTOIN wrt ORIGIN*t.home() *# BRING INITIAL TURTLE TO THE STARTING POINT DRAWING UNNECESSARY LINE.*turtle.done()  
  
  
**import** turtle  
t=turtle.Turtle()  
*# t.pen(fillcolor="black", pencolor="red", pensize=10)  
# PEN-SIZE= t.pensize(10)  
# PEN COLOR= t.pencolor("red") or t.color("red")  
# TURTLE TRIANGLE COLOR= t.fillcolor("yellow")*t.setx(100) *# SET THE POSITION OF X CORDINATE LEAVING Y # t.goto(100,0) or t.setpos(100,0)*t.right(-90)  
t.fd(50)  
*#turtle.filling('red')  
#t.begin\_fill()*t.dot(10, **"blue"**) *# POINT(SIZE,COLOUR) FORM WHERE YOU PUT IT*t.lt(60) *# CHANGE DIRECTION TO LEFT*t.fd(200)  
*#t.end\_fill()*turtle.done() *# TURTLE HAS DONE, NOW STOP WORKING  
#####*print(**"==================================="**)  
  
  
  
  
*#############################  
# SQUARE FORMATION  
#####***import** turtle  
T = turtle.Turtle()  
T.speed(0)  
**for** i **in** range(4):  
 T.forward(100)  
 T.right(90)  
turtle.done()  
*# SQUIRAL-SPIRAL***from** turtle **import** \*  
**for** i **in** range(200):  
 fd(i)  
 lt(93)  
 fd(i)  
*#####*print(**"==================================="**)  
  
  
  
  
*#############################  
# CIRCLE  
####  
# CICLE HAVING LENGTH OF THE RADIUS AND ANGLE OF THE ARC***import** turtle  
t = turtle.Turtle()  
t.circle(100,360)  
turtle.done()  
*# SPIRAL***for** i **in** range(50):  
 t.circle(i,45)  
turtle.done()  
*# CONCENTRIC CIRCLE***for** j **in** range(100):  
 t.pendown()  
 t.circle(2\*j)  
 t.penup()  
turtle.done()  
*# CONCENTRIC CIRCLE***for** i **in** range(50):  
 t.circle(10 \* i)  
 t.up()  
 t.sety(-(10 \* i) )  
 t.down()  
turtle.done()  
*# CIRCLE WITH WHILE***while**(count <100):  
 turtle.forward(5)  
 turtle.left(4)  
 count = count + 1  
turtle.done()  
*#####*print(**"==================================="**)  
  
  
  
*#############################  
# 4.STAR HAVING PENTAGON  
####***import** turtle  
T=turtle.Turtle()  
T.speed(0)  
*#draw a star spiral and overlaping pentagon***for** i **in** range(50):  
 T.forward(50+5\*i)  
 T.right(144)  
T.penup() *#to stop drawing*T.home() *#reach starting point and draw ongoing statement*T.pendown() *#start drawing  
#To draw pentagon***for** i **in** range(50):  
 T.forward(50+5\*i)  
 T.right(72)  
turtle.done()  
*#####*print(**"==================================="**)  
  
  
  
*#############################  
# 5.OVERLAPPING SPIRAL  
####***import** turtle  
T=turtle.Turtle()  
T.speed(0)  
*#draw overlapping spiral***for** i **in** range(360):  
 T.forward(100+5\*i)  
 T.right(179)  
turtle.done()  
*#####*print(**"==================================="**)  
  
  
  
  
*#############################  
# 6.RECTANGULAR SPIRAL  
####***import** turtle  
T=turtle.Turtle()  
T.speed(30)  
**for** i **in** range(0,100):  
 i=i+1  
 T.forward(i\*10)  
 T.right(90)  
 T.forward(i\*10)  
 T.right(90)  
turtle.done()  
  
*#####*print(**"==================================="**)  
  
  
  
*#############################  
# #1.b COLOURFULL HISTOGRAM  
#####***import** turtle  
T = turtle.Turtle()  
T.hideturtle()  
turtle.tracer(0, 0)  
xzoom = 30  
yzoom = 3  
**def** gradecolor(x):  
 c = **'green'  
 if** x <= 20:  
 c = **'red'  
 elif** x <= 50: *# this implies if x>20 and x<=50:* c = **'orange'  
 elif** x <= 80: *# this implies if x>50 and x<=80:* c = **'yellow'  
 return** c  
**def** drawHistogram(A):  
 **for** i **in** range(len(A)):  
 T.penup()  
 T.setpos(i \* xzoom, 0)  
 T.pendown()  
 T.setpos(i \* xzoom, A[i] \* yzoom)  
 T.dot(xzoom // 2)  
**def** drawBetterHistogram(Arr):  
 **for** i **in** range(len(Arr)):  
 T.penup()  
 T.color(gradecolor(Arr[i]))  
 T.begin\_fill()  
 T.setpos(i \* xzoom, 0)  
 T.pendown()  
 T.setpos(i \* xzoom, Arr[i] \* yzoom)  
 T.setpos(i \* xzoom + xzoom // 2, Arr[i] \* yzoom)  
 T.setpos(i \* xzoom + xzoom // 2, 0)  
 T.setpos(i \* xzoom, 0)  
 T.end\_fill()  
**def** main():  
 A = [37, 50, 25, 48, 2, 10, 88, 44, 33, 57, 60, 30, 99, 34, 12, 8, 50]  
 drawBetterHistogram(A)  
 *# <=20 == Red  
 # 21 to 50 == orange  
 # 51 to 80 = yellow  
 # >80 = green*main()  
turtle.update()  
turtle.done()  
*#####*print(**"==================================="**)  
  
  
  
  
*#############################  
# 1.c SORTING HOSTOGRAM ANIMATING TO MAKE ALL HISTOGRAM BLOCKS IN PRECEDING SEQUENCE  
#####***import** turtle  
T = turtle.Turtle()  
T.hideturtle()  
*# turtle.tracer(0,0)*T.speed(0)  
xzoom = 10 *# BREADTH OF THE BARS*yzoom = 1 *# LENGTH OF THE BARS***def** drawBar(x, y):  
 T.penup()  
 T.begin\_fill() *# TO FILL BAR WITH COLOUR* T.setpos(x \* xzoom,0 )  
 T.pendown()  
 T.setpos(x \* xzoom, y \* yzoom)  
 T.setpos(x \* xzoom + xzoom // 2, y \* yzoom)  
 T.setpos(x \* xzoom + xzoom // 2, 0)  
 T.setpos(x \* xzoom, 0)  
 T.end\_fill()  
**def** drawThing(Arr):  
 **for** i **in** range(len(Arr)):  
 drawBar(i, Arr[i])  
**def** swap(Arr, i, g):  
 *# Swap elements at positions i and g  
 # 1. Delete the old bars* T.color(**"white"**)  
 drawBar(i, Arr[i])  
 drawBar(g, Arr[g])  
 *# 2. Actually swap the values* Arr[i], Arr[g] = Arr[g], Arr[i]  
 *# 3. Draw the new bars* T.color(**"black"**)  
 drawBar(i, Arr[i])  
 drawBar(g, Arr[g])  
**def** selectionSort(Arr):  
 **for** i **in** range(len(Arr)):  
 *# g = location of minimum element from i onwards* g = i  
 **for** j **in** range(i, len(Arr)):  
 **if** Arr[g] > Arr[j]:  
 g = j  
 swap(Arr, i, g)  
**def** main():  
 A = [37, 50, 25, 48, 2, 10, 88, 44, 33, 57, 60, 30, 99, 34, 12, 8, 50]  
 *# <=20 == Red  
 # 21 to 50 == orange  
 # 51 to 80 = yellow  
 # >80 = green* drawThing(A)  
 selectionSort(A)  
 print(A)  
main()  
turtle.update()  
turtle.done()  
*#####*print(**"==================================="**)  
  
  
  
  
*#############################  
# 9.CAGE  
#####***import** turtle  
t=turtle.Turtle()  
t.speed(10)  
**for** i **in** range(5):  
 t.forward(10)  
 t.right(90)  
 t.forward(100)  
 t.left(90)  
 t.forward(10)  
 t.left(90)  
 t.forward(100)  
 t.right(90)  
t.penup()  
t.home()  
t.pendown()  
**for** i **in** range(5):  
 t.forward(100)  
 t.right(90)  
 t.forward(10)  
 t.right(90)  
 t.forward(100)  
 t.left(90)  
 t.forward(10)  
 t.left(90)  
t.penup()  
t.home()  
t.pendown()  
t.right(90)  
t.forward(100)  
t.left(90)  
t.forward(100)  
turtle.done()  
*#####*print(**"==================================="**)  
  
  
  
*#############################  
# 10. PERFECT SPIRAL  
#####***import** turtle  
t=turtle.Turtle()  
turtle.bgcolor(**'black'**)  
t.speed(0)  
t.pencolor(**'yellow'**)  
t.pensize(2)  
**for** i **in** range(1000):  
 t.forward(1)  
 t.forward(0.01+0.1\*i)  
 t.right(18)  
t.penup()  
t.home()  
turtle.done()  
  
*#####*print(**"==================================="**)  
  
  
  
  
*#############################  
# COLOURFUL ILLUSSING TUNNEL  
#####***import** turtle  
colors = [**'red'**, **'purple'**, **'blue'**, **'green'**, **'orange'**, **'yellow'**]  
t = turtle.Pen()  
turtle.bgcolor(**'black'**)  
**for** x **in** range(360):  
 t.pencolor(colors[x%6]) *# [x%6] 6 IS SAME AS THE NUMBER OF COLOUR IN THE LIST* t.width(x/100 + 1) *# TO INCREASE THE WIDTH OF THE PEN* t.forward(x)  
 t.left(59)  
*#####*print(**"==================================="**)  
  
  
  
  
*#############################  
# DRAWING MANY SQUARE WITH SLIGHTLY ANGLE CHANGING  
#####***import** turtle  
t=turtle.Turtle()  
**for** i **in** range(10):  
 turtle.fd(50)  
 turtle.lt(80)  
**for** i **in** range(20):  
 turtle.undo() *#ITS REVERSE THE PROCESS*t.hideturtle()  
**def** draw\_square():  
 turtle.forward(50)  
 turtle.left(90)  
 turtle.forward(50)  
 turtle.left(90)  
 turtle.forward(50)  
 turtle.left(90)  
 turtle.forward(50)  
 turtle.left(90)  
turtle.color(**'green'**)  
**for** i **in** range(360):  
 draw\_square()  
 turtle.left(i)  
turtle.done()  
*#####*print(**"==================================="**)  
  
  
  
  
*#############################  
# NOT WORKING  
#####***import** turtle  
**from** turtle **import** Turtle, Screen  
t=Turtle(**"turtle"**)  
screen=Screen()  
t.speed(-1)  
**def** dragging(x,y):  
 t.ondrag(**None**)  
 t.setheading(t.toward(x,y))  
 t.goto(x,y)  
 t.ondrag(dragging)  
**def** clickright():  
 t.clear()  
**def** main():  
 turtle.listen()  
 t.ondrag(dragging)  
 turtle.onscreenclick(clickright,3)  
 screen.mainloop()  
main()  
*#####*print(**"==================================="**)  
  
  
  
  
*#############################  
# STAR  
#####***from** turtle **import** \*  
color(**'red'**, **'blue'**)  
begin\_fill() *# START FILLING COLOUR***while True**:  
 forward(200)  
 left(170)  
 **if** abs(pos()) < 1: *#TO STOP IF TURTLE CUTS ITS OWN LINE* **break**end\_fill() *# END FILLING COLOUR*done()  
*#####*

ICP-CLASS

SORTING

*#++++++++++++ SORTING ++++++++++++++++#  
  
  
  
  
  
#############################  
# INSERTIONSORT  
######***def** swap(A,x,y):  
 A[x],A[y]=A[y],A[x]  
A=[10,80,30,40,15,90,10,80,30,90,63,-2]  
**def** insertionsort(arr):  
 B = []  
 **for** i **in** range(len(arr)): *# N element so length n but index upto n-1* **for** j **in** range(i):  
 **if** arr[j] > arr[i]:  
 swap(arr, j, i)  
 print(**"SORTED ARRAY USING INSERTIONSORT IS:"**, A)  
insertionsort(A)  
*######*print(**"==================================="**)  
  
  
  
  
*#############################  
# BUBBLESORT  
######***def** swap(A,x,y):  
 A[x],A[y]=A[y],A[x]  
A=[10,80,30,40,15,90,10,80,30,90,63,-2]  
**def** BubbleSort(arr):  
 **def** Sort(arr):  
 B = []  
 **for** i **in** range(len(arr) - 1): *# N element so length n but index upto n-1  
 #print(arr)* **if** arr[i] > arr[i + 1]:  
 B.append(arr[i])  
 swap(arr, i, i + 1)  
 **return** len(B)  
 n=Sort(arr)  
 **if** n!=0:  
 BubbleSort(arr)  
BubbleSort(A)  
print(**"SORTED ARRAY USING BUBBLESORT IS:"**, A)  
*#####*print(**"==================================="**)  
  
  
  
*#########################################  
# QUICKSORT  
######***def** swap(A,x,y):  
 A[x],A[y]=A[y],A[x]  
arr=[10,80,30,40,15,90,10,80,30,90,63,-2]  
**def** quicksort(arr, left, right):  
 *# DIVIDING ARRAY INTO TWO PARTS* **def** partition(arr, left, right):  
 x = arr[right] *# RIGHT MOST DIGIT VALUE* wall = left *# LEFT MOST DIGIT PLACE IN ARRAY IS WALL* **for** i **in** range(left, right):  
 **if** arr[i] < x:  
 swap(arr, i, wall)  
 wall = wall + 1 *# INCREASE THE PLACE OF WALL* swap(arr, wall, right) *# SWAP RIGHT MOST DIGIT TO CURRENT WALL PLACE* **return** wall  
 *# MAIN CODES OF THE QUICKSORT* **if** left<right:  
 p=partition(arr, left, right)  
 quicksort(arr, left, p-1)  
 quicksort(arr, p+1, right)  
n = len(arr)  
quicksort(arr, 0, n-1)  
print(**"SORTED ARRAY USING QUICKSORT IS:"**, arr)  
*#####  
# MODIFIED TO COUNTING NUMBER OF STEPS  
# Function to swap the elements***def** swap(A,x,y):  
 A[x],A[y]=A[y],A[x]  
arr=[2,1,3]  
*#*B = [] *# empty array to store the number of swaping performed***def** partition(arr, left, right):  
 x=arr[left] *# LEFT MOST DIGIT VALUE* wall=right *# WALL IS THE RIGHT MOST DIGIT PLACE IN ARRAY* **for** i **in** range(left, right):  
 **if** arr[right-i]<x:  
 swap(arr,right-i,wall)  
 **if** wall>right-i: *# TO ADD NUMBER OF SWAP PERFORMED IN EACH PARTITION* B.append(arr[wall])  
 wall=wall-1 *# INCREASE THE PLACE OF WALL* swap(arr,wall,left) *# SWAP RIGHT MOST DIGIT TO CURRENT WALL PLACE* **if** wall>left:  
 B.append(arr[wall])  
 **return** wall  
*#*D = []  
C=[]  
**def** quicksort(arr, left, right):  
 **if** left<right:  
 p=partition(arr, left, right)  
 **for** i **in** range(left,right+1):  
 D.append(i)  
 C.append(len(B))  
 quicksort(arr, left, p-1)  
 quicksort(arr, p+1, right)  
 **return**(D,C)  
  
quicksort(arr, 0, len(arr)-1)  
print(**"Number of Comparison"**,len(D))  
print(**"Number of Swaps Perforned:"**,sum(C))  
print(**"Sorted array is:"**)  
**for** i **in** range(len(arr)):  
 print(**"%d"** % arr[i])  
*#  
# SWAP FUNCTION TO SWAP ELEMENTS OF THE ARRAY***def** swap(A,x,y):  
 A[x],A[y]=A[y],A[x]  
arr=[2,1,3]  
  
*# DIVIDING ARRAY INTO TWO PARTS*B = [] *# EMPTY ARRAY DEFINED TO ADD THE NUMBER OF SWAPING***def** partition(arr, left, right):  
 x=arr[right] *# THE VALUE OF RIGHT MOST DIGIT* wall=left *# LEFT MOST DIGIT PLACE IN ARRAY IS WALL* **for** i **in** range(left, right):  
 **if** arr[i]<x:  
 swap(arr,i,wall)  
 **if** wall<i: *# SWAPING IS NOT PERFORMED IN THE CASE, WALL=i* B.append(arr[wall])  
 wall=wall+1 *# INCREASE THE PLACE OF WALL* swap(arr,wall,right) *# SWAP RIGHT MOST DIGIT TO CURRENT WALL PLACE  
 # THE LAST SWAP HAPPENED IN THE CASE OF RIGHT MOST ELEMENTS* **if** wall<right:  
 B.append(arr[wall])  
 **return** wall  
*#SORTING ELEMENTS OF THE DIVIDED PARTS*A = []  
C=[]  
D=[]  
**def** quicksort(arr, left, right):  
 **if** left<right:  
 p=partition(arr, left, right)  
 **for** i **in** range(left,right+1):  
 D.append(i)  
 C.append(len(B))  
 A.append(p) *# TO ADD THE VALUE OF P, LATER IT'S USED TO FIND NUMBER OF PARTITION* quicksort(arr, left, p-1)  
 quicksort(arr, p+1, right)  
 **return**(A,C,D)  
quicksort(arr, 0, len(arr)-1)  
print(**"Number of Comparison:"**,len(D))  
print(**"Number of Partition:"**,len(A))  
print(**"Number of Swaps Perforned:"**,sum(C))  
print(**"Sorted array is:"**)  
**for** i **in** range(len(arr)):  
 print(**"%d"** % arr[i])  
  
*#####*print(**"==================================="**)  
  
  
  
*#########################################  
# SORTING MULTIPLE KEYS- USING OPERATOR MODULE  
######***import** operator  
A=[(**"RAHUL"**,16,2),(**"SHIV"**,3,22),(**"RAHUL"**,16,3)]  
print(sorted(A,key=operator.itemgetter(0,1,2)))  
print(sorted(A, key=**lambda** tpl:(tpl[0],tpl[1],tpl[2]))) *# BOTH ARE SAME*

HANOI PROBLEM

1…

**def** anoi(height, fromPole, toPole,withPole):  
 **if** height>= 1:  
 anoi(height-1, fromPole, withPole, toPole)  
 print (**‘Move disk={} from {} to {}.’**.format(height,fromPole, toPole))  
 anoi(height-1, withPole, toPole, fromPole)  
anoi(3, **“A”**, **“C”**, **“B”**)

2…

**def** anoi(disks, fromPole, withPole, toPole):  
 **if** disks == 1:  
 print(**‘Move disk 1 from {} to {}.’**.format(fromPole, toPole))  
 **return** anoi(disks – 1, fromPole, toPole, withPole)  
 print(**‘Move disk {} from peg {} to peg {}.’**.format(disks,fromPole, toPole))  
 anoi(disks – 1, withPole,fromPole,toPole) *# IT MOVES DISK FROM WP TO TP*disks = int(input(**‘Enter number of disks: ‘**))  
anoi(disks, **‘A’**, **‘B’**, **‘C’**)

3…

**def** moveTower(height,fromPole, toPole, withPole):  
 **if** height >= 1:  
 moveTower(height-1,fromPole,withPole,toPole)  
 moveDisk(fromPole,toPole)  
 moveTower(height-1,withPole,toPole,fromPole)  
**def** moveDisk(fp,tp):  
 print(**“moving disk from”**,fp,**”to”**,tp)  
height = int(input(**‘Enter number of disks: ‘**))  
moveTower(height,**”A”**,**”C”**,**”B”**)

TURTLE SELF SIMILARITY

**1…PENTAGON**

t.circle(1,2,1)  
t.seth(to\_angle=180) *# SET TURTLE HEADING INTO GIVEN DIRECTOIN wrt ORIGIN  
# t.setx(100) # SET THE POSITION OF X CORDINATE LEAVING Y # t.goto(100,0) or t.setpos(100,0) # MOVES or SET POSITION OF TURTLE TO GIVEN COORDINATE  
# t.home() # BRING INITIAL TURTLE TO THE STARTING POINT DRAWING UNNECESSARY LINE.*turtle.home() *# NEW TURTLE FROM STARTING POINT*turtle.right(79)  
turtle.color(**“red”**)  
turtle.fd(50)  
turtle.dot(10, **“blue”**) *# POINT(SIZE,COLOUR) FORM WHERE YOU PUT IT*turtle.lt(60) *# CHANGE DIRECTION TO LEFT*turtle.fd(200)  
turtle.done() *# TURTLE HAS DONE, NOW STOP WORKING*

**2…TREE**

**import** turtle  
**def** tree(length, n):  
 **if** length < (length / n):  
 **return** turtle.forward(length)  
 turtle.left(45)  
 tree(length \* 0.5, length / n)  
 turtle.left(20)  
 tree(length \* 0.5, length / n)  
 turtle.right(75)  
 tree(length \* 0.5, length / n)  
 turtle.right(20)  
 tree(length \* 0.5, length / n)  
 turtle.left(30)  
 turtle.backward(length)  
 **return**turtle.left(90)

turtle.backward(30)  
tree(200, 2)

**3…TRIANGLE (SEIPWNSKI)**

**import** turtle  
**import** random  
T= turtle.Turtle()  
T.speed(0)  
turtle.tracer(0,0)  
T.penup() *# TO AVOID LINE DRAWING*T.hideturtle()  
**def** halfway\_there(a, b):  
 x = (a[0]+b[0])/2  
 y = (a[1]+b[1])/2  
 **return** (x,y)  
color = [**'red'**,**'green'**,**'blue'**]  
p = [**None**]\*3 *# NUMBER OF POINTS*T.right(60)  
T.setpos(100,300) *# TO FIX POSITION OF IMAGE***for** i **in** range(3):  
 T.pensize(3)  
 T.forward(300)  
 T.right(120)  
 p[i] = T.pos()  
 T.color(color[i])  
 T.dot(5)  
**for** i **in** range(10000):  
 choice = random.randint(0,2) *# randomly choosing red, green, blue* middle = halfway\_there(T.pos(), p[choice])  
 T.setpos(middle) *# FILLING POSITION OF OTHER POINTS DEFINED IN MIDDLE* T.color(color[choice]) *# FILLING COLOUR IN POINTS* T.dot(2)  
turtle.update()  
turtle.done()  
turtle.done()

4…STAR

**import** turtle  
**import** random  
T= turtle.Turtle()  
T.speed(0)  
turtle.tracer(0,0)  
T.penup()  
T.hideturtle()  
**def** halfway\_there(a, b):  
 x = (a[0]+b[0])/2  
 y = (a[1]+b[1])/2  
 **return** (x,y)  
color = [**'red'**,**'green'**,**'blue'**,**'yellow'**,**'black'**]  
p = [**None**]\*5  
T.right(72)  
T.setpos(100,300) *# TO FIX POSITION OF IMAGE***for** i **in** range(5):  
 T.forward(300)  
 T.right(72)  
 p[i] = T.pos()  
 T.color(color[i])  
 T.dot(5)  
**for** i **in** range(10000):  
 *##randomly choosing red, green, blue* choice = random.randint(0,4)  
 middle = halfway\_there(T.pos(), p[choice])  
 T.setpos(middle) *# FILLING POSITION OF OTHER POINTS DEFINED IN MIDDLE* T.color(color[choice]) *# FILLING COLOUR IN POINTS* T.dot(6)  
turtle.update()  
turtle.done()

**FORMING SPIRAL WITH CIRCLES**

**import** turtle *#Drawing tool***import** math  
zoom = 0.02  
T = turtle.Turtle() *#Initiates turtle*T.hideturtle() *#Hides turtle cursor*turtle.tracer(0,0)  
T.penup()  
**for** v **in** range(10000):  
 x = v\*math.cos(v)\*zoom  
 y = v\*math.sin(v)\*zoom  
 T.setpos(x,y)  
 T.pendown()  
 T.dot(2)  
 T.penup()  
turtle.update()  
turtle.done()

**ZIGZAG LINE**

**import** turtle  
**import** time  
T = turtle.Turtle()  
T.hideturtle() *# Hiding Turtle  
# turtle.tracer(0,0) # To direct print line and point without the line movement.*xzoom = 40 *# Y axis length*yzoom = 2 *# Y axis length***def** display(A):  
 T.penup() *# To avoid a line at the start* T.setpos(0,A[0]\*yzoom) *# It fixes the turtle position x=0 and y=first digit in array\*yzoom length* T.pendown() *# To draw the line* **for** i **in** range(len(A)):  
 T.setpos(i\*xzoom,A[i]\*yzoom)  
 T.dot(10) *# Dot size of the point***def** main():  
 A = [37,50,25,48,2,10,88,44,33,57,60,30,99,34,12,8,50]  
 display(A)  
 turtle.update()  
 time.sleep(0) *# Time gaap between upadating and drawing another line* A[0],A[1]=A[1],A[0] *# Swamping the first two value of array* T.clear() *# To clear what we have drawn before.* display(A)  
 turtle.update()  
main()  
turtle.update()  
turtle.done()

ANIMATING RANDOM POINT

**import** turtle *# Drawing tool***import** time *# Used for functions related to time***import** random *# Used to generate random numbers/sequences*T = turtle.Turtle() *# Initiates turtle*T.hideturtle() *# Hides turtle cursor*turtle.tracer(0, 0)  
xzoom = 20  
yzoom = 100  
**def** display(A):  
 x\_shift = 1000 *# Pixel or point to cover* T.penup() *# To avoid a line at the start* T.setpos(0 - x\_shift, A[0] \* yzoom)  
 T.pendown()  
 **for** i **in** range(len(A)):  
 color = **"red" if** A[i] < A[i - 1] **else "green"** T.color(color)  
 T.setpos((i \* xzoom) - x\_shift, A[i] \* yzoom)  
 T.dot(5)  
 turtle.update() *# updates the screen with new graph* time.sleep(0.1) *# timed delay  
# x\_new = r \* x\_old \* (1-x\_old)***def** main():  
 A = [0.5] \* 60  
 **for** i **in** range(1000):  
 T.clear()  
 display(A)  
 x\_old = A[-1]  
 x\_new = 3.99 \* x\_old \* (1 - x\_old)  
 A.append(x\_new) *# Adds an element/list to the end of a list* A.pop(0) *# Deletes the element at passed index (in this case, 0)*main()  
turtle.update()  
turtle.done()

27/04/2020

\_init\_() / CLASS

**import** math  
**class** Complex:  
 *# stuff defined here is shared (weird, right??)  
 # try this with an array and see* **def** \_\_init\_\_(self, realIn=0, imaginaryIn=0):  
 **if** isinstance(realIn, Complex):  
 realIn = realIn.real  
 imaginaryIn = realIn.imaginary  
 *# this is specific to each instance* self.real = realIn  
 self.imaginary = imaginaryIn  
 *# ====================* **def** \_\_str\_\_(self):  
 **if** self.imaginary < 0:  
 **return f"{**self.real**} - {**-self.imaginary**}i"** *# you reach this point only in the else condition* **return f"{**self.real**} + {**self.imaginary**}i"** *# ====================* **def** \_\_repr\_\_(self): *# don't worry about this rn* **return** str(self)  
 *# ====================* **def** \_\_add\_\_(self, other):  
 *# just in case other is not a complex number* **if** isinstance(other, (int, float)):  
 other = Complex(other, 0)  
 r = self.real + other.real  
 i = self.imaginary + other.imaginary  
 **return** Complex(r, i)  
 *# ====================* **def** \_\_sub\_\_(self, other):  
 *# just in case other is not a complex number* **if** isinstance(other, (int, float)):  
 other = Complex(other, 0)  
 r = self.real - other.real  
 i = self.imaginary - other.imaginary  
 **return** Complex(r, i)  
 *# ====================* **def** \_\_mul\_\_(self, other):  
 *# just in case other is not a complex number* **if** isinstance(other, (int, float)):  
 other = Complex(other, 0)  
 *# (a+ib)(c+id) = (ac - bd) + i(bc + ad)* r = self.real \* other.real - self.imaginary \* other.imaginary  
 i = self.imaginary \* other.real + self.real \* other.imaginary  
 **return** Complex(r, i)  
 *# ====================* **def** \_\_abs\_\_(self):  
 **return** math.sqrt(self.real \*\* 2 + self.imaginary \*\* 2)   
c1 = Complex(3, 4) *# 3 + 4i*print(c1)  
print([c1])  
print(**f"c1={**c1**}"**) *# print tells c1: what do you look like as a string??*c2 = Complex(5, -7) *# 5 - 7i ==> naively, you'll print 5 + -7i*print(**f"c2={**c2**}"**)  
c3 = (c1 + c2) \* (c1 + 5) *# c1+c2 => +(c1, c2) => c1.\_\_add\_\_(self, c2)  
# c3 = mult( add(c1, c2), add(c1, complexify(5,0))))))))))))*print(**f"c3 = {**c3**}"**)

**1/04/2020**

***# 0.) =>> see generateword.py  
# 1. build "persons"  
# 3. simulate!  
# map: n  
# 0,n....n,n  
# 0,0 ...n,0***

# *# ADDING NUMBER IN A ARRAY  
# METHOD 1*A=[]  
**for** i **in** range(0,10):  
 A.append(i)  
print(A)  
*# METHOD 2*B=[3\*i **for** i **in** range(0,5)] *# WE CAN ALSO USE INSTEAD OF 3\*i=3+i, 3i etc*print(B)  
**def** f(i):  
 **return** 2\*i  
B=[f(i) **for** i **in** range(0,5)]  
print(B)

***Fun: Generate random names (and learn how to make passwords!***

**import** random  
**import** string  
**def** GenerateWordOld(wordlength):  
 box=string.ascii\_letters *#ascii\_lowercase* w=**''  
 for** i **in** range(0, wordlength):  
 w=w+random.choice(box)  
 **return** w  
**def** GenerateWord(wordlength):  
 *#box=string.ascii\_letters #ascii\_lowercase* box = string.ascii\_letters+string.digits  
 **return ''**.join( random.choice(box) **for** i **in** range(0, wordlength) )  
 *#return string.plus(random.choice(box) for i in range(0, wordlength))  
 #return string+''.join(random.choice(box) for i in range(0, wordlength))  
 #return [(random.choice(box) for i in range(0, wordlength))]***def** GenerateName(n):  
 box = string.ascii\_lowercase  
 upper=string.ascii\_uppercase  
 s=random.choice(upper)  
 **return** s+**''**.join(random.choice(box) **for** i **in** range(0,n))  
**def** GenerateNameNew1(n):  
 **return** random.choice( string.ascii\_uppercase) + **''**.join( random.choice(string.ascii\_lowercase) **for** i **in** range(1,n))  
**def** GenerateNameNew2(n):  
 box = string.ascii\_lowercase  
 s = random.choice(string.ascii\_uppercase)  
 **return** s + **''**.join(random.choice(box) **for** i **in** range(0, n))  
w1=GenerateWordOld(12)  
w2=GenerateWord(12)  
w3=GenerateName(12)  
w4=GenerateNameNew2(12)  
print(w1,**'\n'**,w2)  
print(w3, w4, end=**' Thanks'**)  
*#print(string.ascii\_letters)  
#print(string.digits)*

***# 2. make the persons move around***

**import** random  
mapSize = 1000  
**class** Person:  
 idval = [0] *# inside an array, we have shared memory* **def** \_\_init\_\_(self, X=**None**, Y=**None**):  
 **if** X **is None**:  
 X = random.randint(0, mapSize)  
 **if** Y **is None**:  
 Y = random.randint(0, mapSize)  
 self.x = X  
 self.y = Y  
 self.id = self.idval[0]  
 self.idval[0] = self.idval[0] + 1  
 **def** moveRandom(self):  
 X = random.randint(0, mapSize)  
 Y = random.randint(0, mapSize)  
 **def** \_\_str\_\_(self):  
 s = **f"{**self.id**}'s home is located at x={**self.x**}, y={**self.y**}"  
 return** s  
numPersons = 5  
P = [Person() **for** i **in** range(0, numPersons)]  
*# print their home locations***for** x **in** P:  
 print(x)  
**for** x **in** P:  
 x.moveRandom()

3/04/2020

INFECTED PEOPLE

**import** random  
**import** math  
mapSize = 100  
maxDist = 10  
infectionDist = 5  
**class** Person:  
 *# location 0 holds the id counter  
 # location 1 holds the number of people currently infected* mem = [0, 0] *# inside an array, we have shared memory* **def** \_\_init\_\_(self, X=**None**, Y=**None**):  
 **if** X **is None**:  
 X = random.randint(0, mapSize)  
 **if** Y **is None**:  
 Y = random.randint(0, mapSize)  
 self.x = X  
 self.y = Y  
 self.homex = X  
 self.homey = Y  
 self.status = 2 *# recovered=0, okay=2, infected=5* self.id = self.mem[0]  
 self.mem[0] = self.mem[0] + 1  
 **def** moveRandom(self):  
 *# this person, "self" has a home location (homex, homey)  
 # randomly move this person to a new location (x,y)  
 # within distance maxDist from home* temp = self.homex + random.randint(-maxDist, maxDist)  
 **if** temp < 0:  
 temp = 0  
 **if** temp > mapSize:  
 temp = mapSize  
 self.x = temp  
  
 temp = self.homey + random.randint(-maxDist, maxDist)  
 **if** temp < 0:  
 temp = 0  
 **if** temp > mapSize:  
 temp = mapSize  
 self.y = temp  
  
 **def** infect(self):  
 self.status = 5  
 self.mem[1] = self.mem[1] + 1  
  
 **def** \_\_str\_\_(self):  
 s = **f"{**self.id**} is located at x={**self.x**}, y={**self.y**}"  
 return** s  
*# code***def** distanceBetween(A, B):  
 **return** math.sqrt((A.x - B.x) \*\* 2 + (A.y - B.y) \*\* 2)  
  
**def** transmission(P):  
 n = len(P)  
 **for** i **in** range(0, n):  
 **for** j **in** range(i + 1, n):  
 d = distanceBetween(P[i], P[j])  
 **if** d < infectionDist:  
 *# this is subtly wrong  
 # if either one is infected, then both get infected  
 # recovered=0, okay=2, infected=5* **if** P[i].status + P[j].status > 6 **and** P[i].status + P[j].status < 10:  
 P[i].infect()  
 P[j].infect()  
  
**def** countInfected(P):  
 c = 0  
 **for** x **in** P:  
 **if** x.status == 5:  
 c = c + 1  
 **return** c  
  
numPersons = 100  
P = [Person() **for** i **in** range(0, numPersons)]  
  
bad = random.choice(P)  
bad.infect()  
  
**for** day **in** range(50):  
 *# move each person to a new x,y* **for** person **in** P:  
 person.moveRandom()  
 *# check transmission* transmission(P)  
 *# print how many people are infected* print(**f"On day {**day**}, the number of infected people is {**P[0].mem[1]**}"**)  
*# for x in P:  
# x.moveRandom()*

8/04/2020

**import** random  
**import** math  
mapSize = 200 *# this is our world*maxDist = 10 *# this is how far from home you can go*infectionDist = 7 *# how close you need to be to get infected*numPersons = 100 *# number of people in our world*numInfected = 5 *# number of initial infections***class** Person:  
 *# location 0 holds the id counter  
 # location 1 holds the number of people currently infected  
 # location 2 holds number of total infections (ever)* mem = [0, 0, 0] *# inside an array, we have shared memory  
 # these are helper functions* **def** buildNextID(self):  
 self.mem[0] = self.mem[0] + 1  
 **return** self.mem[0]  
 **def** increaseActiveInfections(self):  
 self.mem[1] = self.mem[1] + 1  
 **def** decreaseActiveInfections(self):  
 self.mem[1] = self.mem[1] - 1  
 **def** increaseTotalInfections(self):  
 self.mem[2] = self.mem[2] + 1  
 **def** \_\_init\_\_(self, X=**None**, Y=**None**):  
 **if** X **is None**:  
 X = random.randint(0, mapSize)  
 **if** Y **is None**:  
 Y = random.randint(0, mapSize)  
 self.x = X  
 self.y = Y  
 self.homex = X  
 self.homey = Y  
 self.age = random.randint(5, 95)  
 self.status = 2 *# recovered=0, susceptible=2, infected=5* self.timeToRecover = 0 *# how many days you need to recover once infected* self.id = self.buildNextID()  
 *# we assume this is called exactly once each day* **def** moveRandom(self):  
 *# this person, "self" has a home location (homex, homey)  
 # randomly move this person to a new location (x,y)  
 # within distance maxDist from home* temp = self.homex + random.randint(-maxDist, maxDist)  
 **if** temp < 0:  
 temp = 0  
 **if** temp > mapSize:  
 temp = mapSize  
 self.x = temp  
 temp = self.homey + random.randint(-maxDist, maxDist)  
 **if** temp < 0:  
 temp = 0  
 **if** temp > mapSize:  
 temp = mapSize  
 self.y = temp  
 **if** self.status == 5:  
 *# if I am infected* self.timeToRecover = self.timeToRecover - 1  
 **if** self.timeToRecover <= 0:  
 self.status = 0 *# now I am recovered!* self.decreaseActiveInfections() *# we need to update number of active infections* **def** infect(self):  
 *# if I am susceptible* **if** self.status == 2:  
 self.status = 5 *# then "self" is infected* self.increaseActiveInfections() *# number of active infections +1* self.increaseTotalInfections()  
 **if** self.age > 70:  
 self.timeToRecover = 28  
 **elif** self.age > 50:  
 self.timeToRecover = 21  
 **else**:  
 self.timeToRecover = 14 *# now you need 14 days to recover* **def** \_\_str\_\_(self):  
 s = **f"{**self.id**} is located at x={**self.x**}, y={**self.y**}"  
 return** s  
**def** distanceBetween(A, B):  
 **return** math.sqrt((A.x - B.x) \*\* 2 + (A.y - B.y) \*\* 2)  
**def** transmission(P):  
 n = len(P)  
 **for** i **in** range(0, n):  
 **for** j **in** range(i + 1, n):  
 d = distanceBetween(P[i], P[j]) / infectionDist  
 **if** d < 1:  
 *# this is subtly wrong  
 # the smaller d is, the more likely infection should be* r = random.random() *# this a number between 0 and 1* **if** d < r:  
 *# if either one is infected, then both get infected  
 # recovered=0, susceptible=2, infected=5* **if** P[i].status + P[j].status == 7: *# one infected, one susceptible* P[i].infect()  
 P[j].infect()  
*# first creating numPersons people*P = [Person() **for** i **in** range(0, numPersons)]  
*# initial infections  
# we infect some random person  
#* ***TODO: start with numInfected people****# how do we do this?  
# google "randomly choose some items from a list"  
# random.sample(P, 5)  
# then I just call .infect on each of these individually*bad = random.sample(P, numInfected)  
**for** x **in** bad:  
 x.infect()  
**for** day **in** range(1, 50):  
 *# =============================================  
 # this happens on each day  
 # =============================================  
 # move each person to a new x,y* **for** person **in** P:  
 person.moveRandom()  
 *# check if any two people spread the infection to each other* transmission(P)  
 *# print how many people are infected* print(**f"Day {**day**}: {**P[0].mem[1]**} active infections"**)  
 *# =============================================*print(**f"{**P[0].mem[2]**} total infections"**)  
*# for x in P:  
# x.moveRandom()*

10/04/2020

# CUSTOM EMPLOYEE -CLASS/DATA TYPE

**class** Employee:  
 *# this function is called automatically  
 # when you do x = Employee(...)* **def** \_\_init\_\_(self, eName, eDesig, eDept, salary, yearOfBirth, yearJoined, email, phone, office, boss):  
 self.employeeName = eName  
 self.designation = eDesig  
 self.department = eDept  
 self.prevSalaries = []  
 self.salary = salary  
 self.yearOfBirth = yearOfBirth  
 self.yearJoined = yearJoined  
 self.email = email  
 self.phone = phone  
 self.office = office  
 self.boss = boss  
 **def** getSalary(self):  
 **return** self.salary  
 **def** setSalary(self, newSalary):  
 self.prevSalaries.append(self.salary)  
 self.salary = newSalary  
 *# what does this employee look like as a string  
 # this is what print(e) calls!* **def** \_\_str\_\_(self):  
 **return f"\  
========================================\n\  
Name: {**self.employeeName**}\n\  
Designation: {**self.designation**}\n\  
Department: {**self.department**}\n\  
Salary: {**self.salary**}\n\  
Contact: {**self.email**}, {**self.phone**}"***#================================================  
# code (wish list)  
#================================================*e1 = Employee(**"Debayan Gupta"**, **"Bucket"**, **"Cleanup"**, 100, 1988, 2020, **"dg@cle.an"**, **None**, **"India"**, **None**)  
e1.phone = **"+91 99999 88888"**print(e1) *# e.\_\_str\_\_(e)*e2 = Employee(**"Erik Demaine"**, **"Vaccuum"**, **"Cleanup"**, 100, 1981, 2020, **"ed@cle.an"**, **None**, **"US"**, **None**)  
print(e2)  
*#e1 = Employee("Employee Name", "Designation", "Dept", salary, dob, doe, email, phone, office, boss)  
# print(e1) => ?  
# we want the system to automatically assign an EmpID  
# acccess and edit functions  
#e1.setSalary(newval)  
#e1.getSalary()*

ICP 15-April-2020

# Custom "Employee" class/data type

*# Custom "Employee" class/data type***class** Employee:  
 *# this function is called automatically  
 # when you do x = Employee(...)* **def** \_\_init\_\_(self, eName, eDesig, eDept, salary, yearOfBirth, yearJoined, email, phone, office, boss):  
 self.employeeName = eName  
 self.designation = eDesig  
 self.department = eDept  
 self.prevSalaries = []  
 self.salary = salary  
 self.yearOfBirth = yearOfBirth  
 self.yearJoined = yearJoined  
 self.email = email  
 self.phone = phone  
 self.office = office  
 self.boss = boss  
 **def** getSalary(self):  
 **return** self.salary  
 **def** setSalary(self, newSalary):  
 self.prevSalaries.append(self.salary)  
 self.salary = newSalary  
 *# what does this employee look like as a string  
 # this is what print(e) calls!* **def** \_\_str\_\_(self):  
 **return f"\  
========================================\n\  
Name: {**self.employeeName**}\n\  
Designation: {**self.designation**}\n\  
Department: {**self.department**}\n\  
Salary: {**self.salary**}\n\  
Contact: {**self.email**}, {**self.phone**}"***#================================================  
# code (wish list)  
#================================================*e1 = Employee(**"Debayan Gupta"**, **"Bucket"**, **"Cleanup"**, 100, 1988, 2020, **"\"dg@cle.an"**, **None**, **"India"**, **None**)  
e1.phone = **"+91 99999 88888"**print(e1) *# e.\_\_str\_\_(e)*e2 = Employee(**"Erik Demaine"**, **"Vaccuum"**, **"Cleanup"**, 100, 1981, 2020, **"ed@cle.an"**, **None**, **"US"**, **None**)  
print(e2)  
*#e1 = Employee("Employee Name", "Designation", "Dept", salary, dob, doe, email, phone, office, boss)  
# print(e1) => ?  
# we want the system to automatically assign an EmpID  
# acccess and edit functions  
#e1.setSalary(newval)  
#e1.getSalary()*