Python -IITM

▼ WEEK 1 - 1-14 LEC:

▼ PRINT

- Numbers are treated as numbers only when put without apostrophe/speech marks
- Strings require either single or double quotes
- Uses only round brackets `()`
- Uses comma to separate strings/variables

```
print("Hello world")
print('Hello world', 'Hello mom', 'Hello nietzsche')
print(10)
```

▼ VARIABLES

> 💡 Use well defined names for variables instead of a/b/c etc.

- Input commands

```
print("Enter a number")
num=int(input())
print(num)
```

```
age=int(input("Enter your age:"))
```

- `del` - Used to delete a variable

```
x = 10
del x
print(x)
#shows error x not defined
```

▼ DATA TYPES 1

- `int` - Integer
- `float` - Decimal number
- `str` - String- basically any letter or group of letters

- `list` -
  - Self explanatory
  - uses square bracket [ ]
  - uses comma to separate list items
  - indexing starts from 0 and not 1
  - negative indexing starts from -1 for the last element

```
list = [10,'Amaan', 6.8]
print(l[0]) #prints 10
print(l[2]) #prints 6.8 and not Amaaan
print(type(l[20])) #prints <class 'str'>
```

-

```
r= int(input("Enter the radius:"))
print(type(r))
#prints <class 'int'>
```

▼ DATA TYPES 2

- `bool` - Boolean - Takes either true or false as input. T of true and F of false must always be capital

```
b1 = True
b2 = False
```

Converting data types:

```
a = float(9)
b = int(2.4)
c = int('11')

print(a, type(a)) #prints 9.0 <class 'float'>
print(b, type(b)) #prints 2 <class 'int'>
print(c, type(c)) #prints 11 <class 'int'>
```

```
a = bool(100)
b = bool(-10.9)
c = bool(0)
d = bool('') #empty string

print(a, type(a)) #prints True <class 'bool>
print(b, type(b)) #prints True <class 'bool>
print(c, type(c)) #prints False <class 'bool>
print(d, type(d)) #prints False <class 'bool>
```

▼ OPERATORS AND EXPRESSIONS

Follows BODMAS

▼ *MATHEMATICAL*

- `+` - Sum/Union for lists
- `-` - Subtract
- `/` - Returns Float Quotient

```
print(7/3)
#prints 2.333333333333335
```

- `*` - Multiplication
- `//` - Floor Division - Returns Integer Quotient

```
print(7//3)
#prints 2
```

- `%` - Modulus - Remainder

```
print(7%3)
#prints 1
```

- `**` - Exponential

▼ ***RELATIONAL***

- `<`, `>`, `>=`, `<=` - Self Explanatory
- `==` - Compares the two operands and returns the value as either True or False

```
print(5==50)
#prints False
print(5==5)
#prints True
```

- `!=` - Not Equals to

```
print(5!=50)
#prints True
print(5!=5)
#prints False
```

▼ ***LOGICAL***

- `and` , `or` , `not` - Logic gates. Returns either True or False as value

▼ STRINGS

💡 Strings are treated like a list. Their letters are indexed just like lists. Starting from 0.

```
s = 'coffee'
t = 'bread'
print(s[1:3]) #1:3 - goes from 1 to 2
#prints of
print(s[1:5]) #1:5 - goes from 1 to 4
#prints offe
```

```
s='coffee'
t='bread'
print(s+t)
#prints coffeebread
```

💡 we can use only `+` and `*` operators on strings and not - and /

```
s = 'good'
print(s[0]*5)
#prints ggggg
```

```
print('apple' > 'one')
#prints False because a comes before o. it compares every letter of the string
```

```
print('abce'> 'abcd')
#prints True
```

```
s='huioiklopopopojhue'
print(len(s)) #prints length of the string
#prints 18
```

▼ WEEK 2 - 15-25 LEC:

- **Comment** - Starts with `#`. Isn't read by computer.

```
#this is a comment and it isnt read by the computer
```

▼ Short hand operator

- `+=` -

```
count = count + 1
```

```
count += 1 #should be read as count is equal to count + 1
```

- `-=`, `*=`, `/=` - Self explanatory

▼ Special operator

- `in` - In operator - used to find some specific value in a string or variable or list, etc.

```
print('alpha' in 'A variable name can only contain alpha-numeric characters and underscores')
#prints True as the string contains the word alpha
print('alpha' in 'A variable name must start with a letter or the underscore character')
#prints False as the string doesn't contain the word alpha
```

▼ Chaining operator

When two or more relational operators are used together.

```
x = 5
print(10<x<15)
#prints False as 5 is smaller than 10. Basically it acts like an OR logic gate. both statements need to be true to get the outp
```

▼ Escape characters

- `\` - Back slash

```
print('It's a beautiful day')
#returns error
print('It\'s a beautiful day')
#prints It's a beautiful day
print("We are from "IIT" Madras")
#returns error
print("We are from \"IIT\" Madras")
#prints We are from "IIT" Madras
```

- `\t` - Used to add tab space. Can be used repetitively. Ex - `\t\t\t`, etc

```
print('My name is Fulkit. \t I am from Rijnob'
#prints My name is Fulkit.   I am from Rijnob
```

- `\n` - New line in the same print command

```
print('I am not here. \n This isnt happening')
#prints I am not here
      # This isnt happening
```

▼ Quotes

- `''' '''` - Multi line string

```
z='''Amaan
AM
AP'''
print(z)
#prints Amaan
      #AM
      #AP
```

- `''' '''` - Multi line comment

```
'''comment1
comment2
comment3'''
```

▼ String Methods

Method	Description	Code x = 'pytHoN sTrIng mEthOds'	Output
`lower()`	converts a string into lower case	`print(x.lower())`	python string methods
`upper()`	converts a string into upper case	`print(x.upper())`	PYTHON STRING METHODS
`capitalize()`	converts the first character to upper case	`print(x.capitalize())`	Python string methods
`title()`	converts the first letter of each word to upper case	`print(x.title())`	Python String Methods
`swapcase()`	swaps upper to lower and lower to upper case in the string	`print(x.swapcase())`	PYThOn StRiNG MeTHoDs

Method	Description
`islower()`	Returns True if all characters in the string are lower case
`isupper()`	Returns True if all characters in the string are upper case
`istitle()`	Returns True if the string follows the rule of a title

Code	Output
`x='python' print(x.islower())`	True
`x='Python' print(x.islower())`	False
`x='PYTHON' print(x.isupper())`	True
`x='PYTHoN' print(x.isupper())`	False
`x='Python String Methods' print(x.istitle())`	True
`x='Python string methods' print(x.istitle())`	False

Method	Description
`isdigit()`	Returns True if all characters in the string are digit

Code	Output
`x='123' print(x.isdigit())`	True
`x='123abc' print(x.isdigit())`	False

Method	Description
`isalpha()`	Returns True if all characters in the string are in alphabets
`isalnum()`	Returns True if all characters in the string are alpha-numeric

Code	Output
`x='abc' print(x.isalpha())`	True
`x='abc123' print(x.isalpha())`	False
`x='abc123' print(x.isalnum())`	True
`x='abc123@*#$' print(x.isalnum())`	False

Method	Description	Code x='——Python ——'	Output
`strip()`	Returns a trimmed version of the string	`print(x.strip('-'))`	Python
`lstrip()`	Returns a left trim version of the string	`print(x.lstrip('-'))`	Python——
`rstrip()`	Returns a right trim version fo the string	`print(x.rstrip('-'))`	——Python

Method	Description
`startswith()`	Returns True if the string starts with the specified value
`endswith()`	Returns True if the string ends with the specified value

Code x='Python'	Output
`print(x.startswith('P'))`	True
`print(x.startswith('p'))`	False
`print(x.endswith('n'))`	True
`print(x.endswith('N'))`	False

Method	Description
`count()`	Returns the number of times a specified value occurs in a string
`index()`	Searches the string for a specified value and returns the position of where it was found
`replace()`	Returns a string where a specified value is replaced with a specified value

Code x='Python String Methods'	Output
`print(x.count('t'))`	3
`print(x.count('s'))`	1
`print(x.index('t'))`	2
`print(x.index('s'))`	20

`x=x.replace('S','s')`
`x=x.replace('M','m')`
`print(x)`
	Python string methods

▼ An interesting cipher: More on strings

- `alpha.index(n[i])` - finds index of the 'i'th letter of '`n`' in '`al`'
- `alpha.index(n[i])+1` - increases the index of that specific letter.
- we used `%26` to make sure the code doesn't break if our '`n`' has z in it.
- withouth `%26` it'll show index error as there's no letter ahead of z in '`al`'
- `%26` = we are calling out reminder after we divide our index number with 26 - so if 'z' i.e. 26 is divided by 26 it'll give 0 as reminder. This 0 = index 0 = 'a'
- `(al[(al.index(n[i])+1)%26])` - gives us the +1 shifted letter of n
- we store it in an empty string `t` and append it with the letters shifted `+1` of our `n`

```
al='abcdefghijklmnopqrstuvwxyz'
t=''
i=0
k=1
n='amaan'
#to print bnbbo - a letter shifter of my name
t+=(al[(al.index(n[i])+k)%26])
t+=(al[(al.index(n[i+1])+k)%26])
t+=(al[(al.index(n[i+2])+k)%26])
t+=(al[(al.index(n[i+3])+k)%26])
t+=(al[(al.index(n[i+4])+k)%26])

print(t)
```

https://replit.com/@weakgeek/ceaser-cipher?v=1

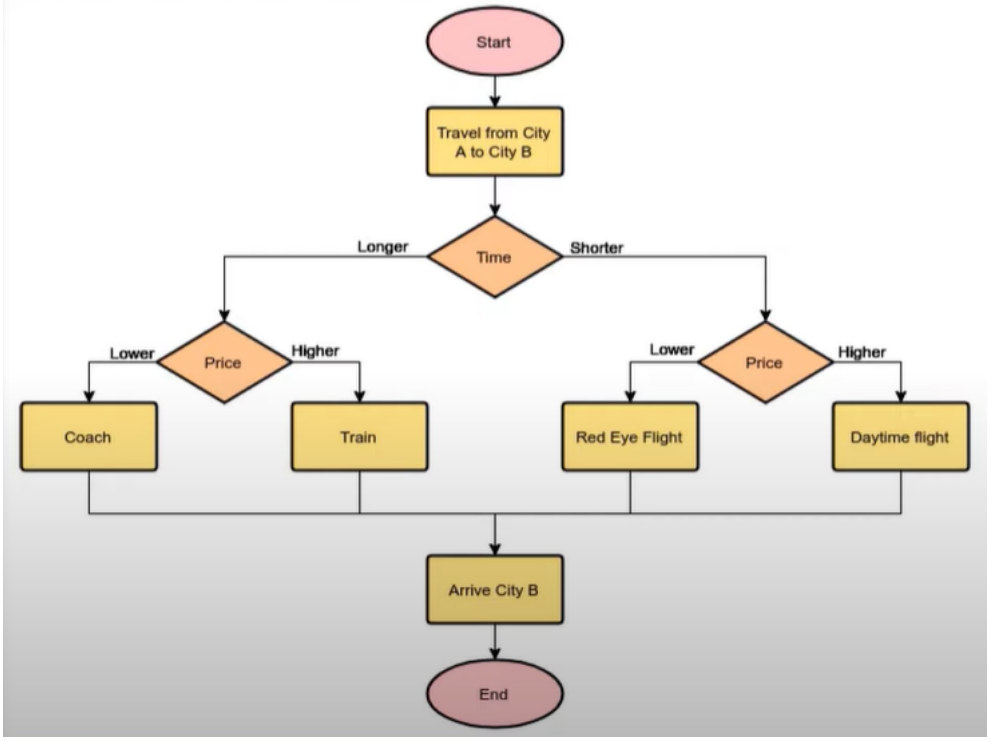
▼ If statement

```
dob=int(input("Enter your dob:"))
currentyear=2022
age=currentyear-dob
if (age<18):
  print("Sorry you can't enter")
else:
  print("Welcome aboard")
```

▼ if, else and elif(else-if)

```
marks=int(input("Enter marks obtained:"))
if(marks>=0 and marks <=100):
  if (marks>=90):
    print("A grade")
  elif (marks>=80):
    print("B grade")
  elif (marks>=70):
    print("C grade")
  elif (marks>=60):
    print("D grade")
  else:
    print("E grade")
else:
  print("Invalid output")
```

```
print('Travel from city A to city B')
time=int(input('Enter a time:'))
longer=int(input('Define longer:'))
if (time>=longer):
  price=int(input('Enter price:'))
  higher=int(input('Define higher:'))
  if (price>=higher):
    print('Train')
  else:
    print('Coach')
else:
  price=int(input('Enter price:'))
  higher=int(input('Define higher:'))
  if(price>=higher):
    print('Daytime Flight')
  else:
    print('Red eye Flight')
```



▼ WEEK 3 - 27-40 LEC:

▼ While loop

- It gets executed till the condition is met
- It quits when the condition is met

```
print("When did India got its independence?")
yr=int(input())

while(yr!=1947):
  print("You got it wrong. Enter once again.")
  yr=int(input())
print("You got it right!")
```

Compute Factorial:

```
n=int(input("Enter the number:"))
i=1
f=1
while(i<=n):
  f=f*i
  i=i+1
print("Factorial is:", f)
```

- When the first time while loop runs f=1*1 and i is incremented by 1. Hence i becomes 2.
- Now the second time runs i is 2. Hence f=1*2=2. Again i is incremented and becomes 2+1 = 3.
- This continues till i>n.

▼ Practice questions on While loop

- Find factorial of a given number:

Problem 1: Find the factorial of the given number

Test Cases		
Sr. No.	**Input**	**Expected output**
1	5	120
2	2	2
3	0	1
4	-7	Not defined

```
n=int(input("Enter the number:"))
i=1
f=1
if (n>0):
  while(i<=n):
    f=f*i
    i=i+1
  print("Factorial is:", f)
else:
  print("Invalid input")
```

- Find the number of digits in a given number

Problem 2: Find the number of digits in the given number

Test Cases		
Sr. No.	**Input**	**Expected output**
1	1234	4
2	1234567890	10
3	8	1
4	0	1
5	-4	1

```
n = abs(int(input("Enter the number:")))

digits = 1
while (n > 9):
    n = n // 10
    digits += 1
print("Number of digits are:", digits)
```

- Reverse the given number

Problem 3: Reverse the digits in the given number

Test Cases		
Sr. No.	**Input**	**Expected output**
1	1234	4321
2	123456789	987654321
3	8	8
4	0	0
5	-1234	-4321

```
n=int(input("Enter the number:"))
num=abs(n)
sum=0
while(num>0):
  r=num%10
  sum=sum*10+r
  num=num//10
if(n<=0):
  print(sum-(2*sum))
else:
  print(sum)
```

- Firstly, the num is divided by 10 to get its remiander - `%` is used for that. When we divide any number by 10 to get its remainder the remainder is always the last digit of the number. So **r** is the last digit of the input number now.
- `sum=sum*10+r` puts the last digit of the input number **r** as the first digit of the reverse number.
- `num=num//10` gives the int quotient of the initial number when divided by 10. Any number when divided by 10 gives the integer quotiet the nummber itself bar the last digit of it. i.e. 1234//10 gives 123.
- The `sum=sum*10+r` for 1234 looks like sum=4 for the first cycle. sum=4*10+3 for the second cycle. sum=43*10+2 for the third cycle. sum=432*10+1 for the last cycle.

- Find if the number is palindrome or not

Problem 4: Find whether the entered number is palindrome or not

Test Cases		
Sr. No.	**Input**	**Expected output**
1	12321	Palindrome
2	1221	Palindrome
3	1234	Not a palindrome
4	123421	Not a palindrome
5	9	Palindrome
6	-1221	Palindrome

```
n=int(input("Enter the number:"))
num=abs(n)
sum=0
while(num>0):
  r=num%10
  sum=sum*10+r
  num=num//10
if(sum==n or (sum-2*sum)==n):
  print('Palindrome')
else:
  print('Not a palindrome')
```

▼ For loop

- Adding first **n** numbers

```
n=int(input("Enter the number:"))
sum=0
for i in range(n):
  sum=sum+(i+1)
print(sum)
```

- Range has 3 parameters starting, ending and step
- Default **step** of range is 1
- Default **starting** of range is from 0

▼ Formatted Printing

- `end` - Attribute for the end of a command/string, etc. Default value is `\n` (new line)

```
for x in range(10):
  print(x, end = ' ')
#prints 0,1,2,3,4,5,6,7,8,9
```

- `sep` - Separator between two strings, etc. Default value is ‘ ‘ (space)

```
d=10,m=5,y=2022
print("Today's date is", d,m,y, end=' ', sep='-')
#prints Today's date is 10-5-2022
```

- **formatted printing** - Use to create a single string value

```
num=int(input("Enter the number:"))
for i in range(1,11):
  print(f'{num} X {i} = {num*i}')
```

- `.format` - In this method of formatting the print statement the variable are assigned indices and are later designated after the string.

```
num=int(input("Enter the number:"))
for i in range(1,11):
  print('{0} X {1} = {2}'.format(num,i,num*i)
```

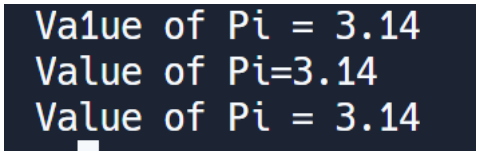
- **String modulo operator** - An old way of writing print statement. Was used by C language.
  - `%d` - integer
  - `%f` - float

```
num=int(input("Enter the number:"))
for i in range(1,11):
  print('%d X %d = %d' % (num,i,num*i)
```

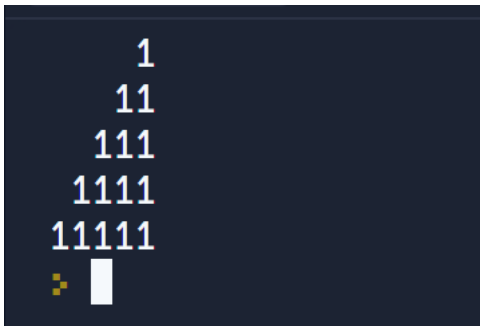
```
pi=22/7
print('value of Pi =%f' % (pi))
#shows only 6 digits after the point - 3.142857 because its an old method
```

- **format specifiers** - Used to describe the format of the output value

```
pi=22/7
print(f'Va1ue of Pi = {pi:.2f}')
print('Value of Pi = {0:.2f}'.format(pi))
print('Value of Pi = %.2f %(pi))
#prints 2 value after point i.e. 3.14
```




```
print('{0:5d}'.format(1))
print('{0:5d}'.format(11))
print('{0:5d}'.format(111))
print('{0:5d}'.format(1111))
print('{0:5d}'.format(11111))
```



the code above specifies that the computer must use atleast 5 characters in the print statement. **{0:5d}**

▼ Nested loops

- P3

Problem 3: Find the day wise total rainfall for the entered duration of time e.g. week, month, etc.

Test Cases

Day	Input	Expected output
	7	
1	10 15 19 5 -1	49
2	12 13 31 -1	56
3	57 32 85 47 82 -1	303
4	2 5 8 2 4 8 8 2 -1	39
5	-1	0
6	21 73 90 3 7 -1	194
7	30 17 40 10 2 -1	99

```
nod=int(input("Enter number of days: "))
for i in range(1,(nod+1)):
  sum=0
  rain=int(input("Enter the rainfall: "))
  while(rain!=-1):
    sum=sum+rain
    rain=int(input("Enter the rainfall: "))
  print('Total rainfall for {0} days is {1}'.format(i,sum))
```

- P4

Problem 4: Find the length of longest word from the set of words entered by the user

Test Cases

Sr. No.	Input	Expected output
1	IITM online degree -1	6
2	Bachelor of science -1	8
3	Computational thinking -1	13
4	Introduction to python -1	12
5	Programming data structures and algorithms	11

```
word=input("Enter the word: ")
max=0
while(word!='-1'):
  count=0
  for let in word:
    count=count+1
  if(count>max):
    max=count
  word=input("Enter the word: ")
print(f'The length of the longest word is {max}')
```

▼ Break, Continue and Pass

- `break`

  Exits the loop under which it is written.

```
#to print username part of the input emailID
email=input("Enter the email ID: ")
for x in email:
  if(x=='@'):
    break
  print(x)
#prints, ex:
a
a
m
i
o
```

- `continue`

  Skips the iteration if the condition is met.

```
#to print username and email service used from the input emailID
email=input("Enter the email ID: ")
for x in email:
  if(x=='@'):
    print('')
    continue
  print(x, end='')
#prints, ex:
amanaskdk
gmail.com
```

- `pass`

  Does nothing. Acts as a null operation. Nothing happens when it is executed.

```
#to print the numbers less than 10 divisible by 3 and do nothing with the numbers that aren't
for i in range(11):
  if(i%3==0):
    print(i)
  else:
    pass
```

▼ WEEK 4 - 41-50 LEC:

▼ Warmup with Lists

- List is denoted by square brackets []

▼ Common Commands:

- `append` : adds an element to the list

```
l=[1,2,3,4,5]
l.append(6)
print(l)
'''Prints [1,2,3,4,5,6]'''
```

- `remove` : removes an element from the list

```
l=[1,2,3,4,5,6]
l.remove(6)
print(l)
'''Prints [1,2,3,4,5]'''
```

- `sort` : does obvious sort on a list i.e. small to big

```
l=[1,2,6,8,3,5,75,854,9]
l.sort()
print(l)
'''[1, 2, 3, 5, 6, 8, 9, 75, 854]'''
```

- Lists within list:

```
m=[]
m.append([1,2,3])
m.append([4,5,6])
m.append([7,8,9])
print(m)
'''prints [[1, 2, 3], [4, 5, 6], [7, 8, 9]]'''
print(m[0][0])
print(m[0][1])
print(m[2][2])
''''prints 1
          2
          3'''
#this is how we create a matrix
```

▼ Birthday Paradox

```
import random
l=[]
for i in range(60):
  l.append(random.randint(1,365))
l.sort()
print(l)
i=0
flag=0
while(i<len(l)-1):
  if(l[i]==l[i+1]):
    print("Repeats", l[i], l[i+1])
    flag=1
    pass
  i=i+1
if(flag==0):
  print("There is no repetition")
'''
Output example:
[10, 14, 22, 24, 25, 25, 25, 34, 58, 69, 83, 84, 88, 90, 92, 97, 100, 112, 114, 116, 120, 128, 131, 134, 154, 164, 168, 173, 17
Repeats 25 25
Repeats 25 25
Repeats 294 294'''
```

- First we import python's random library
- Then we initialize an empty list to store our DOBs
- Then we use the for loop to cycle through the random integers that randint parameter generates.
- randint is used in place of random.random because the latter generates float numbers too and DOBs cant be float numbers
- Then we sort the list in which we appended the random integers.
- We use the `flag` to make sure the while loop first checks all the numbers in the list and then displays if there was repetition or not
- We use the `pass` command to make sure our program enlists all the repetitive numbers/DOBs

▼ Naive Search in a List

```
import random
l=[]
for i in range(10000):
  l.append(random.randint(1,10000))
while(n>-1):
  n=int(input("Enter a number:, type -1 to exit")
  flag=0
  for i in range(len(l)):
    if (a==l[i]):
      print("Found it!")
      flag=1
```

```
      break
  if (flag==0):
    print("Not found!")
```

- We use this code to find any element in a list

▼ The Obvious Sort

```
l=[1,6,2,9,241,66,854,30,5,8]
m=[]
while(len(l)>0):
    min=l[0]
    for i in range(len(l)):
        if l[i]<min:
            min=l[i]
    m.append(min)
    l.remove(min)
print(m)
```

▼ Dot Product

```
l=[1,6,2,9,241,66,854,30,71]
y=[342,5,61,34,77,54,85,893,8]
sum=0
for i in range(len(l)):
    sum=sum+l[i]*y[i]
print(sum)
```

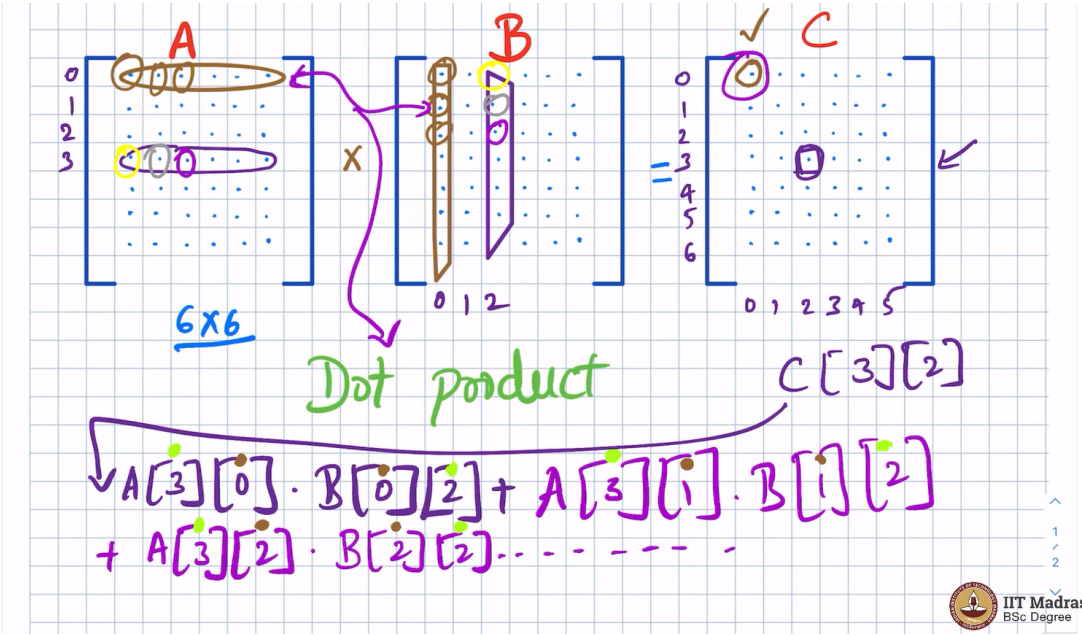
- Only works for square matrices of equal rows and columns

▼ Matrix Addition

```
dim=3
r1=[1,2,3]
r2=[4,5,6]
r3=[7,8,9]
s1=[2,5,1]
s2=[9,8,5]
s3=[5,3,6]

A=[]
A.append(r1)
A.append(r2)
A.append(r3)

B=[]
B.append(s1)
B.append(s2)
B.append(s3)

print(A)
print(B)

C=[[0,0,0],[0,0,0],[0,0,0]]

for i in range(dim):
  for j in range(dim):
    C[i][j] = A[i][j]+B[i][j]

print(C)
```

This code can be automated by asking for inputs from the user of the elements and dimensions of matrices.

▼ Matrix Multiplication



- This is how matrix multiplication happens.
- For C[3][2] we have to find the dot product of A[3][0].B[0][2] + A[3][1].B[1][2] + A[3][2].B[2][2] + A[3][3].B[3][2] + A[3][4].B[4][2] + A[3][5].B[5][2] + A[3][6].B[6][2]
- C[i][j] is the dot product of ith row of A and jth column of B
- We observe that the row of first matrix remains same and the column changes and the column of secod matrix remains same and its row changes.
- Hence we can use a loop to find the dot products and sum them up to get the product of two matrices

```
dim=3
r1=[1,2,3]
r2=[4,5,6]
r3=[7,8,9]
s1=[2,5,1]
s2=[9,8,5]
s3=[5,3,6]

A=[]
A.append(r1)
A.append(r2)
A.append(r3)

B=[]
B.append(s1)
B.append(s2)
B.append(s3)

print(A)
print(B)

C=[[0,0,0],[0,0,0],[0,0,0]]
for i in range(dim):
    for j in range(dim):
        for k in range(dim):
         C[i][j]=C[i][j]+A[i][k]*B[k][j]

print(C)
```

▼ WEEK 5 - 51-60 LEC:

▼ Introduction to Functions

```
def add(a,b):
  ans=a+b
  print(ans)
```

- This is a simple function to add two numbers that are inputed

```
def sub(a,b):
  ans=a-b
  print(ans)
```

- Prints the difference of two numbers

```
def disc(cost,d):
  ans=cos-(cost*(d/100))
  print(ans)
```

- Gives the discounted price of any product

```
def add(a,b):
  ans=a+b
  print(ans)
a=2
b=4
ans=add(a,b)+10
print(ans)
```

- This code will throw an error as we've used `print(ans)` instead of `return(ans)`

```
def add(a,b):
  ans=a+b
  return(ans)
a=2
b=4
ans=add(a,b)+10
print(ans)
```

- This would return the value of `ans` instead of printing `ans` so we can do more mathematical operations post running of the function on ans

```
def discount(cost,d):
  ans=cost-(cost*(d/100))
  return ans
print("Enter the cost price")
x=int(input())
print("Enter the discount")
y=int(input())
print("The final discount is:", discount(x,y))
```

- In this code we call the function by using `discount(x,y)` where `x` and `y` serve as `cost` and `d` of the function

▼ More Examples of Functions

```
def first_element(l):
  return l[0]
x=[1,2,3,4,5,6,7]
print(first_element)
'''prints 1'''
```

- A simple function to print the first element of a list.
- A function can take list as an input.

```
def list_min(l):
    mini=l[0]
    y=0
    for i in range(len(l)):
        if (l[i]<mini):
            mini=l[i]
    return mini
```

```
l=[12,23,53,74,21,97,8,374]
print(list_min(l))
```

- Prints the minimum element of a list

```
def list_maxi(l):
    maxi=l[0]
    for i in range(len(l)):
        if (l[i]>maxi):
            maxi=l[i]
    return maxi

l=[1,24,53,74,94,37,32,]
print(list_maxi(l))
```

- Prints the maximum element of a list

```
def list_apppendbefore(l,z):
    newl=[]
    for i in range(len(z)):
        newl.append(z[i])
    for i in range(len(l)):
        newl.append(l[i])
    return newl

l=[1,24,532,61]
z=[9,54,7,23,62,74]

print(list_apppendbefore(l,z))
```

- Appends a list before another in a new list

```
def list_apppend(l,z):
    newl=[]
    for i in range(len(l)):
        newl.append(l[i])
    for i in range(len(z)):
        newl.append(z[i])
    return newl

l=[1,24,532,61]
z=[9,54,7,23,62,74]

print(list_apppend(l,z))
```

- Appends a list after another in a new list

```
def list_avg(l):
    sum=0
    for i in range(len(l)):
        sum=sum+l[i]
    ans=sum/len(l)
    return ans

l=[1,2,533,63,72,84]
print(list_avg(l))
```

- Prints the avg of a list

▼ Sorting using Functions

```
def obvious_sort(l):
    m=[]
    while(len(l)>0):
        mini=l[0]
        for i in range(len(l)):
            if (l[i]<mini):
                mini=l[i]
        m.append(mini)
        l.remove(mini)
    return m
```

```
l=[24,632,7824,843,923,64,98,247,342,678,356]
print(obvious_sort(l))
```

- Performs obvious sort using a function

```
def list_min(l):
    mini=l[0]
    y=0
    for i in range(len(l)):
        if (l[i]<mini):
            mini=l[i]
    return mini

def obvious_sort(l):
  x=[]
  while(len(l)>0):
    mini=list_min(l)
    x.append(mini)
    l.remove(mini)
  return l

l=[24,632,7824,843,923,64,98,247,342,678,356]
print(obvious_sort(l))
```

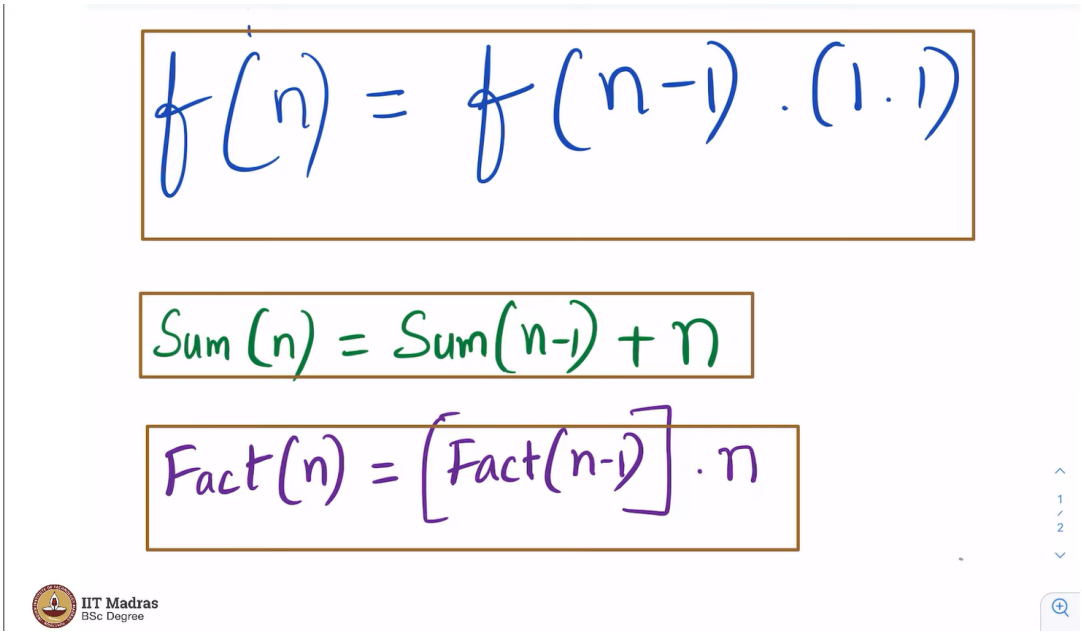
- Modular way of programming.
- Functions can be used to break your code into simpler chunks.

▼ Matrix Multiplication using Functions

```
#initialize c to zero
#i need to consider two matrices A and B
#i need to find the dot product of two lists
#i need to pick ith row and jth column in a matrix

def ini_mat(dim):
    C=[]
    for i in range(dim):
        C.append([])
    for i in range(dim):
        for j in range(dim):
            C[i].append(0)
    return C

def dot_pro(u,v):
    dim=len(u)
    ans=0
    for i in range(dim):
        ans=ans+(u[i]*v[i])
    return ans

def row(M,i):
    dim=len(M)
    l=[]
    for k in range(dim):
        l.append(M[i][k])
    return l

def column(M,j):
    dim=len(M)
    l=[]
    for k in range(dim):
        l.append(M[k][j])
    return l

def mat_mul(A,B):
    dim=len(A)
    C=ini_mat(dim)
    for i in range(dim):
        for j in range(dim):
            C[i][j]= dot_pro(row(A,i), column(B,j))
    return C

A=[[1,2,3],[4,5,6],[7,8,9]]
B=[[1,2,1],[3,1,7],[6,2,3]]

print(mat_mul(A,B))
```

- We’ve created a modular code to find the product of two square matrices using functions.

▼ Theoretical Introduction to Recursion



▼ Recursion - An Illustration

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

print(sum(10))
```

- Recursion - Repetition of functions by running the function inside of itself.
- The above code gives the sum of the **nth** numbers.
- Python lets you call the same function within the function.

```
#we are assuming the interest is 10% in this code

def comp(p,n):
    if (n==1):
        return p*(1.1)
#1.1 = (1+10/100)
    else:
        return comp(p,n-1)*1.1

print(comp(2000,3))
```

- Calculates value after compound interest of 10%
- We used `(n-1)` instead of `(n)` in the else statement because `f(3) =f(2)*1.1`

```
def fact(n):
    if (n==1):
        return 1
    else:
        return fact(n-1)*n

print(fact(5))
```

- Calculates the factorial using recursion

▼ Types of Function Arguments

- `def` - function definition.
- **function call** - this is the place where we are actually executing the defined function.
- **arguments** - the values which we pass along with the function call

```
def add(c,a,b): #def-function definition
    return(a+b-c) #function call
print(add(a=30,b=20,c=40)) #these are called keyword arguments
print(add(30,20,40)) #these are called positional arguments
```

```
def add(c,a=20,b=30):
    return(a+b-c)
print(add(40)) #these are called default arguments
print(add(40,10)) #10 replaces the default value of parameter a from 20 to 10
print(add(40,10,15) # replaces tha default value of a and b by 10 and 15 respectively.
```

```
def add(c,a=20,b=30):
    return(a+b-c)
print(add(40,a=50)) #this is a combination of default,positional and keyword arguments
```

▼ Scope of Variables

```
def myfunc1(x):
    x=x*2
    print("Value of x in function", x)

x=5
print("Value of x before function call",x) #5
myfunc1(x)
print("Value of x after function call",x) #10
```

- Value of function comes out to be 5 and not 10 (as we expected) this is because of **call by value** and **scope of variable**.

- **Call by value**:
  - Whenever we call a function along with an argument we are not actually passing this variable `x` to this function.
  - What we are actually doing is, only passing the value of that variable to this function.

- **Scope of variable:**
  - The variable inside the function is called a **local variable** while the variable outside the function is called **global variable**.

```
def myfunc():
    global x
    x=x*2
    print("Value of x in function", x)

x=5
print("Value of x before function call",x) #5
myfunc()
print("Value of x after function call",x*3) #30
```

  - This way we can use gloval variable inside the function instead of creating a local variable that takes in the value of the global variable.
  - This code prints:

    `Value of x before function call 5`

    `Value of x in function 10`

    `Value of x after function call 30`

▼ Types of functions

```
#Inbuilt functions
print(),input(),len()

#Library functions
math.log(),math.sqrt(),random.random(),random.randrange()
calender.calendar(),calendar.month()

#String methods (functions)
''.upper(),"".lower(),strip(),count(),index(),replace()

#User defined functions
def square(x):
  sqr=x**2
  return sqr
```

▼ Tutorial on functions

▼ Find number of different types of letters in a string

Problem 1: Write a Python code using functions which calculates the number of upper case letters, lower case letters, total number of characters and number of words

Test Cases

Sr. No.	Input	Expected output
1	Functions could have no parameters	1 29 34 5
2	FUNCTIONS COULD HAVE NO RETURN VALUE	31 0 36 6
3	Functions Could Have Multiple Return Statements, But The Moment The First Return Is Executed, Control Exits From The Function	19 86 125 19
4	FUNCTIONS have to be Defined before THEY can be called. the function Call Cannot come before the DEFINITION	26 63 107 18
5	fUNCTION CALLS COULD BE USED IN EXPRESSIONS	36 1 43 7

```
def num_of_chars():
    global x
    return len(x)

def num_of_words():
    global x
    j=1
    for i in range(len(x)):
        if (x[i]==' '):
            j=j+1
    return j

def num_of_lowercase():
    global x
    k=0
    alpha=['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z']
    for i in range(len(x)):
        for j in range(len(alpha)):
            if(x[i]==alpha[j]):
                k=k+1
    return k

def num_of_uppercase():
    global x
    k=0
    alpha=['A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z']
    for i in range(len(x)):
        for j in range(len(alpha)):
            if(x[i]==alpha[j]):
                k=k+1
    return k

x=input("Enter the sentence: ")
print("Total number of uppercase letters are:", num_of_uppercase())
print("Total number of lowercase letters are: ", num_of_lowercase())
print("Total number of characters are: ", num_of_chars())
print("Total number of words are: ", num_of_words())
```

- This is my code.

```
def upper(s):
    upper=0
    for c in s:
        if(c.isupper()):
            upper+=1
    return(upper)

def lower(s):
    lower=0
    for c in s:
        if(c.islower()):
            lower+=1
    return(lower)

def characters(s):
    chars=0
    for c in s:
        chars+=1
    return(chars)

def words(s):
    words=1
    for c in s:
        if(c==' '):
            words+=1
    return(words)

sentence=input('Enter the sentence: ')
uLetters = upper(sentence)
print(f'\nTotal number of upper case characters: {uLetters}')
lLetters = lower(sentence)
print(f'\nTotal number of lower case letters are: {lLetters}')
chars = characters(sentence)
print(f'\nTotal number of characters: {chars}')
words = words(sentence)
print(f'\nTotal number of words: {words}')llll ll
```

- This is instructor’s code.

```
x = input("Enter the sentence: ")

def num_of_chars():
    global x
    return len(x)

def num_of_words():
    global x
    return len(x.split())

def num_of_lowercase():
    global x
    return sum(1 for c in x if c.islower())

def num_of_uppercase():
    global x
    return sum(1 for c in x if c.isupper())

print("Total number of characters:", num_of_chars())
print("Total number of words:", num_of_words())
print("Total number of lowercase letters:", num_of_lowercase())
print("Total number of uppercase letters:", num_of_uppercase())
```

- This is chatgpt’s code.

▼ Find area and perimeter of circle and rectangle

Problem 2: Write a Python code using functions to calculate area and perimeter of circle and rectangle

Test Cases					
Sr. No.	**Input**	**Expected output**	**Sr. No.**	**Input**	**Expected output**
1	"circle" "area" 7	154 sq. units	2	"circle" "perimeter" 7	44 units
3	"rectangle" "area" 15 10	150 sq. units	4	"rectangle" "perimeter" 15 10	50 units
5	"exit"	*Stop execution*			

```
pi=22/7
def circle_area(x):
    return pi*(x**2)

def circle_peri(x):
    return 2*pi*x

def rect_area(x,y):
    return x*y

def rect_peri(x,y):
    return 2*(x+y)

def circle_cal(x):
    if (x=='area'):
        rad=float(input("Enter the radius:\n "))
        print("The area of the circle is",circle_area(rad), "sq. units")
    elif(x=='perimeter'):
        rad=float(input("Enter the radius: \n"))
        print("The perimeter of the circle is", circle_peri(rad))

def rect_cal(x):
    if (x=='area'):
        length=float(input("Enter length of the rectangle:\n "))
        breadth=float(input("Enter the breadth of the rectangle:\n "))
        print("The area of the rectangle is",rect_area(length,breadth),"sq. units")
    elif (x=='perimeter'):
        length=float(input("Enter lenght of the rectangle: "))
        breadth=float(input("Enter the breadth of the rectangle: "))
        print("The perimeter of the rectangle is",rect_peri(length,breadth))

polygon=''
calcu=''
while(polygon!='exit'):
    print("\nPOLYGONS\n \nEnter'circle' for circle\nEnter 'rectangle' for rectangle\nEnter 'exit' to exit\n")
    polygon=input("Choose the polygon type or exit: \n")
    if(polygon=='circle'):
        while(calcu!='exit'):
            x=input("Enter 'area' to calculate area  \n\n Enter 'perimeter' to calculate perimeter \n \n")
            circle_cal(x)
            break
    elif(polygon=='rectangle'):
        while(calcu!='exit'):
             x=input("Enter 'area' to calculate area \n \n Enter 'perimeter' to calculate perimeter \n\n ")
             rect_cal(x)
             break
```

- This is a menu driven program. The while loop acts as a menu.

▼ Find if the input coordinates form a triangle

Problem 3: Write a Python code using functions which checks whether the input coordinates form a triangle or not

Test Cases

Sr. No.	Input	Expected output
1	(0, 0), (0, 1), (1, 0)	Triangle
2	(-3, -5), (-7, 10), (0, 0)	Triangle
3	(1, 2), (1, 10), (5, 2)	Triangle
4	(1, 1), (2, 2), (3, 3)	Not a triangle
5	(2, 3), (3, 2), (2, 3)	Not a triangle
6	(0, 0), (10, 0), (0, 0.0001)	Triangle

```
def tri_check(a,b,c,d,e,f):
    area=0
    s1=abs(((c-a)**2)+((d-b))**2)**(0.5)
    s2=abs(((e-c)**2)+((f-d)**2))**(0.5)
    s3=abs(((e-a)**2)+((f-b)**2))**(0.5)
    s=abs((s1+s2+s3)/2)
    area=(s*(s-s1)*(s-s2)*(s-s3))**(0.5)
    if (area==0):
        print("Not a triangle")
    elif(area!=0):
        print("Triangle")

def coords():
    a=float(input("Enter a1: \n"))
    b=float(input("Enter a2: \n"))
    c=float(input("Enter b1: \n"))
    d=float(input("Enter b2: \n"))
    e=float(input("Enter c1: \n"))
    f=float(input("Enter c2: \n"))
    return tri_check(a,b,c,d,e,f)

coords()
```

- This is my code.

```
import math

def slope(xi,yi,xj,yj):
    if(xi==xj):
        return(math.inf)
    else:
        return((yj-yi)/(xj-xi))

x1=float(input("Enter x coordinate of 1st point: "))
y1=float(input("Enter y coordinate of 1st point: "))
x2=float(input("Enter x coordinate of 2nd point: "))
y2=float(input("Enter y coordinate of 2nd point: "))
x3=float(input("Enter x coordinate of 3rd point: "))
y3=float(input("Enter y coordinate of 3rd point: "))
s1=slope(x1,y1,x2,y2)
print(f'\nSlope of the line connecting points ({x1},{y1}) and ({x2},{y2})=',s1)
s2=slope(x2,y2,x3,y3)
print(f'\nSlope of the line connecting points ({x2},{y2}) and ({x3},{y3})=',s2)
if (s1!=s2):
    print('\nTriangle')
else:
    print('\nNot a triangle')
```

- This is instructor's code.

▼ WEEK 6 - 61-67 LEC:

▼ Lists and Sets

- Set - No repetition allowed. Denoted by `{ }`
- `in` - Checks for values in a list, set, dictionary

```
y={1,2,3,4,5,73,325,63,23,15,85,79,98}
type(y)
set
79 in y
True
78 in y
False
```

List	Set
Uses less memory	Uses more memory than a list
Is indexed	Isn't indexed
Naive search takes a lot of time	Naive search doesn't take a lot of time
Each individual element can be referred.	Elements can't be referred.

```
import sys
l=list(range(100))
s=set(range(100))
print(sys.getsizeof(l)) #856
print(sys.getsizeof(s)) #8408
print(l[1]) #1
print(s[1]) #throws error - 'set' object is not subscriptable
```

▼ Dictionaries

```
d={}
d['sud']=989898
d['am']=99108
d[123]='amaan'

print(d)
```

```
malgudi=['It', 'was', 'Monday', 'morning' , 'Swaminathan', 'was', 'reluctant', 'to', 'open', 'his', 'eyes', 'he', 'considered',
print(len(malgudi))
s=set(malgudi)
print(len(s))

d={}

for x in s:
    d[x]=0 #this stores every word in set s as keys and assigns the value 0 to them
          #like this - d=['It'=0,'was'=0...]
for x in malgudi:
    d[x]=d[x]+1 #this adds +1 to the value of the keywords (words of the sentence) every time they are iterated

print(d)

max=0
answer_word=''
d={}

for x in s:
    d[x]=0

for x in malgudi:
    d[x]=d[x]+1
    if (d[x]>max):
        max=d[x] #this stores the value of most occurred word in max
        answer_word=x #this stores the keyword (word of the sentence) of the most occurred word

print(max,'-',answer_word)
```

- We've created a program to check the occurrence of words in a given sentence using dictionaries.

▼ Tuples

- `()` - used for tuples.

- Tuples are immutable (i.e. can't be edited).
- Uses less space than a list.

```
import string
s=string.ascii_letters
a='My name is aNtOnY %12 g0nse1v1e2 @176  0^^'
b=list(a)
alpha = tuple(list(s))
r=[]
for p in b:
    if p in alpha:
        r.append(p)
print(r) #this prints only the letters of our inpu sentence and removes everything else
```

▼ More on lists

```
l1=[1,2,3]
l2=[10,20,30]
l12=l1+l2
l21=l2+l1
print(l12,l21) #prints [1,2,3,10,20,30] [10,20,30,1,2,3]
```

```
l1=[0,0,0,0,0]
l2=[0]*5
print(l1,l2) #both give same output
l3=[1,2,3]*5
print(l3) #prints [1,2,3,1,2,3,1,2,3,1,2,3,1,2,3]
```

```
l1=[1,2,3]
l2=[1,2,3]
l3=[1,3,2]
print(l1==l2) #prints True
print(l2==l3) #prints False
print(l2<l3) #prints True
```

```
print([1,2]<[2,1]) #prints True
print([1]<[1,2,3]) #prints True
print([2,3]<[3]) #prints True
print([]<[1]) #prints True
```

```
l=[1,2,4]
print(l) #prints [1,2,4]
l[2]=3
print(l) #prints [1,2,3]
```

```
l1=[1,2,3]
l2=l1
l1[0]=100
print(l1,l2) #prints [100,2,3] [100,2,3]

'''This happens because instead of creating a new memory location,
python simply adds another name to the same memory location created
earlier'''
```

```
'''To create a new memory location for a list there are 3 ways'''
l1=[1,2,3]
l2=list(l1)
l3=l1[:]
l4=l1.copy()
l5=l1

l2[0]=100
l3[1]=200
l4[2]=300
print(l1,l2,l3,l4)
#[1,2,3] [100,2,3] [1,200,3]
```

```
'''To check if python has alloted different or same memory location
we use 'is' operator'''

print(l1 is l2) #False
print(l1 is l3) #False
print(l1 is l4) #False
print(l1 is l5) #True
```

```
def add(x):
    x.append(1)
    return x
x=[5]
print(add(x)) #prints [5,1]
print(x) #prints [5,1]

'''If the function argument is of mutable type then it is called
by reference otherwise it is called by value'''
'''Hence in case of list it is called by reference and in case of
integer it is called by value'''
```

```
'''LIST METHODS'''
l=[1,2,3]
print(l)

l.append(4) #Takes in only 1 value #It also adds the element at the end of the list
print(l)

l.insert(2,9) #Takes in multiple values #It adds the element at the second index of the list
print(l)

l.remove(2) #removes the value 2 irrespective of the index
print(l)

l.pop(0) #removes the element at 0 index of the list
print(l)

l=[2,6,1,50,3,7,5]
l.sort() #sorts in ascending order
print(l)

l=[2,6,1,50,3,7,5]
l.reverse() #prints the list in reverse order
print(l)
```

▼ More on Tuples

```
t = 1, 2, 3
print(t,type(t)) #prints (1,2,3) <class 'tuple'> #this is packing

x,y,z = t
print(x,y,z) #prints 1,2,3 #this is unpacking
```

```
l=[10]
print(l,type(l)) #prints [10] <class 'list'>

t=(10)
print(t,type(t)) #prints 10 <class 'int'>

'''To make the type of t=(10) tuple we've to put a comma after 10'''

t=(10,)
print(t,type(t)) #prints 10 <class 'tuple'>
```

```
t=([1,2], ['a','b']) #two lists inside a tuple
t[0] = [10,20]
print(t) #throws an error
t[0][0] = 10
print(t) #prints ([10,2], ['a','b']) #Hence we can edit a list inside a tuple
'''We can't modify tuple values but if a value inside a tuple is a list,
then we can modify the values inside that list'''
'''This particular principle is called hashable'''
```

```
t1=(1,2,3) #If values inside tuple are immutable it is referred as hashable
t2=([1,2,3],) #If values inside tuple are mutable it is referred as non-hashable
```

▼ More of Dictionaries

```
d={'key':'value'}
```

- We can use immutable data types as key. Hence, integer, float, boolean, string but not a list or dictionary.
- Value has no restrictions, we can use integer, float, boolean, string, list, tuple, dictionary, etc.
- Key has to be unique the value can be repeated or duplicated.
- If we pass dictionary as an argument to a function, it will be passed as a reference.

```
d={0:0,1:1,2:4,3:9,4:16}
print(d)
for key in d:
    print(key,d[key])
```

```
'''DICTIONARY METHODS'''
d={0:0,1:1,2:4,3:9,4:16}
print(d.keys()) #prints dict_keys([0,1,2,3,4])
print(d.values()) #prints dict_values([0,1,4,9,16])
print(d.items()) #prints dict_items([(0,0),(1,1),(2,4),(3,9),(4,16)])
                 #every element in the list is a tuple
```

▼ More on Sets

```
st = {1,2,3,4,5,1,2,3,4,5}
print(st) #prints {1,2,3,4,5}
```

- set doesn't allow repetitive elements.

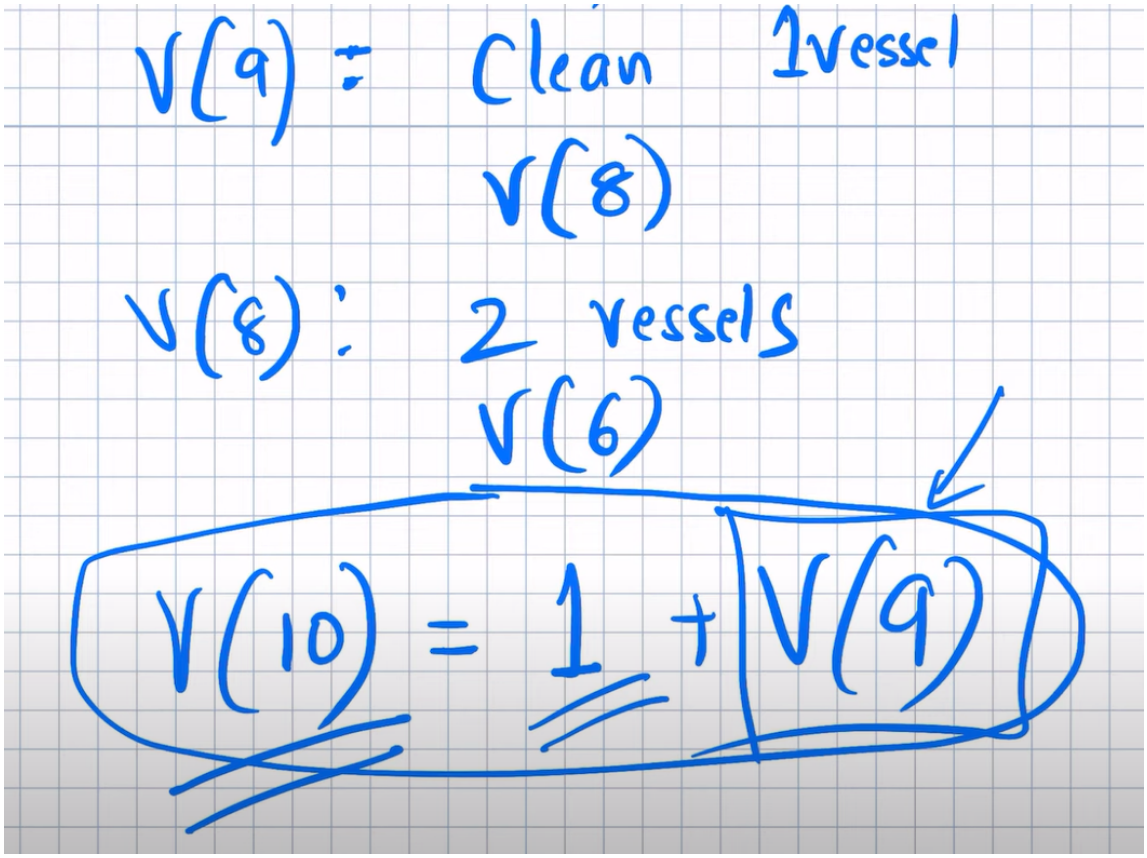
```
st = {1,2,3,4,5,1,2,3,4,5}
print(st[0])
```

- set is an unordered entity. It has no index.
- Set is mutable but every value inside it has to be immutable and hashable which means you can add values into a set but those values have to be immutable. Hence, we cannot add a dictionary or list in a set.

```
'''SET METHODS'''

A={1,2,3}
B={1,2,3,4,5}
print(A.issubset(B)) #prints True
print(A.issuperset(B)) #prints False

A={1,2,3}
B={3,4,5}
C1=A.union(B)
C2=A | B
print(C1,C2)

A={1,2,3}
B={3,4,5}
C1=A.intersection(B)
C2=A&B
print(C1,C2)

A={1,2,3}
B={3,4,5}
C1=A.difference(B)
C2=A-B
print(C1,C2)
```

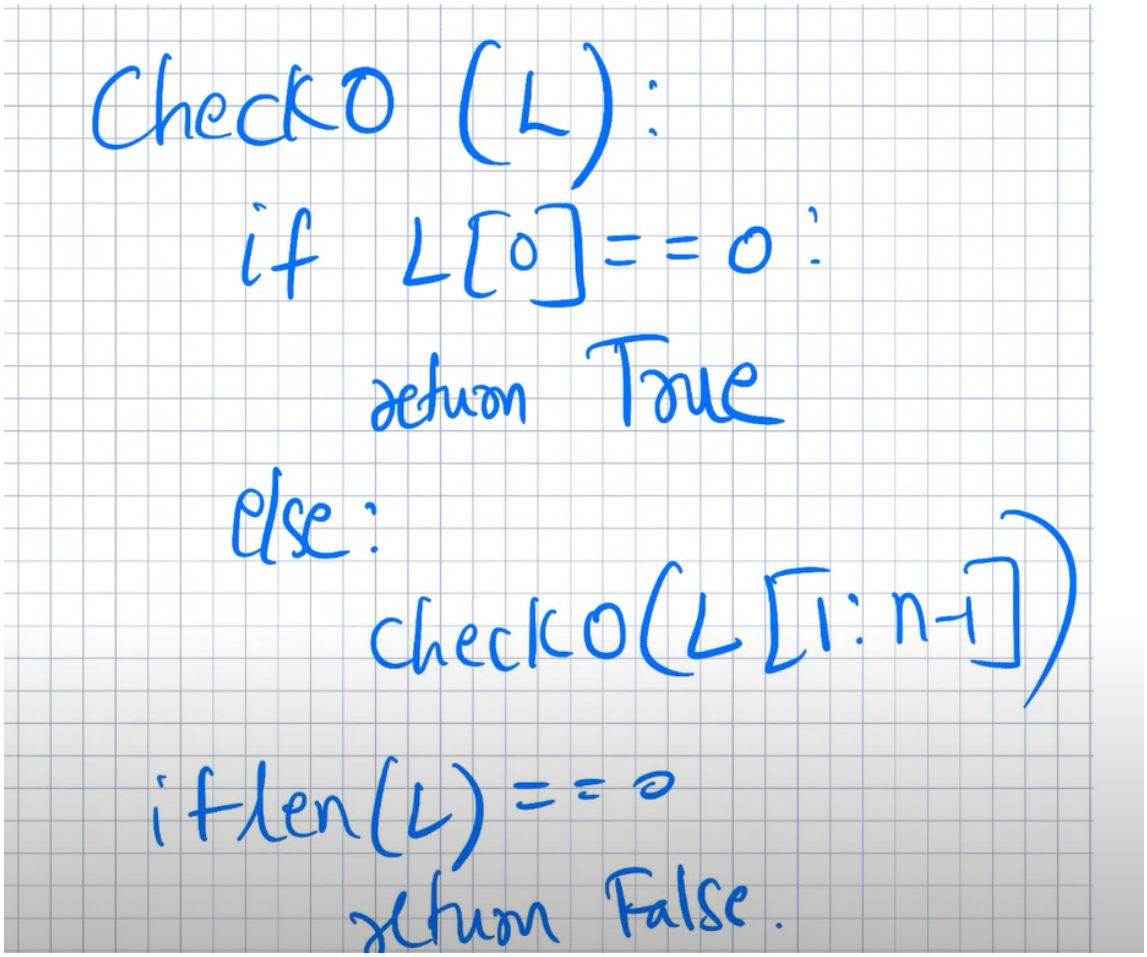
▼ WEEK 7 - 68-75 LEC:

▼ Introduction to Recursion

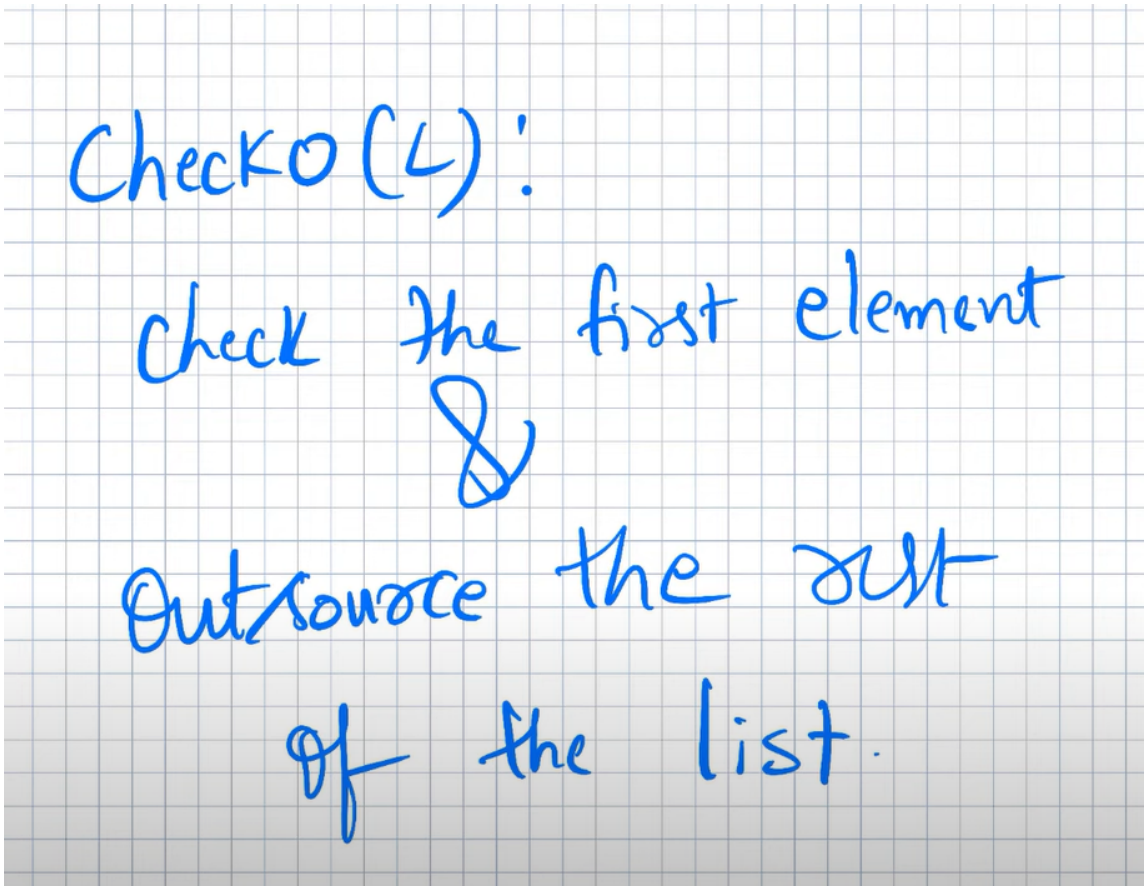


- Same thing as **F(n) = F(1)+F(n-1)**

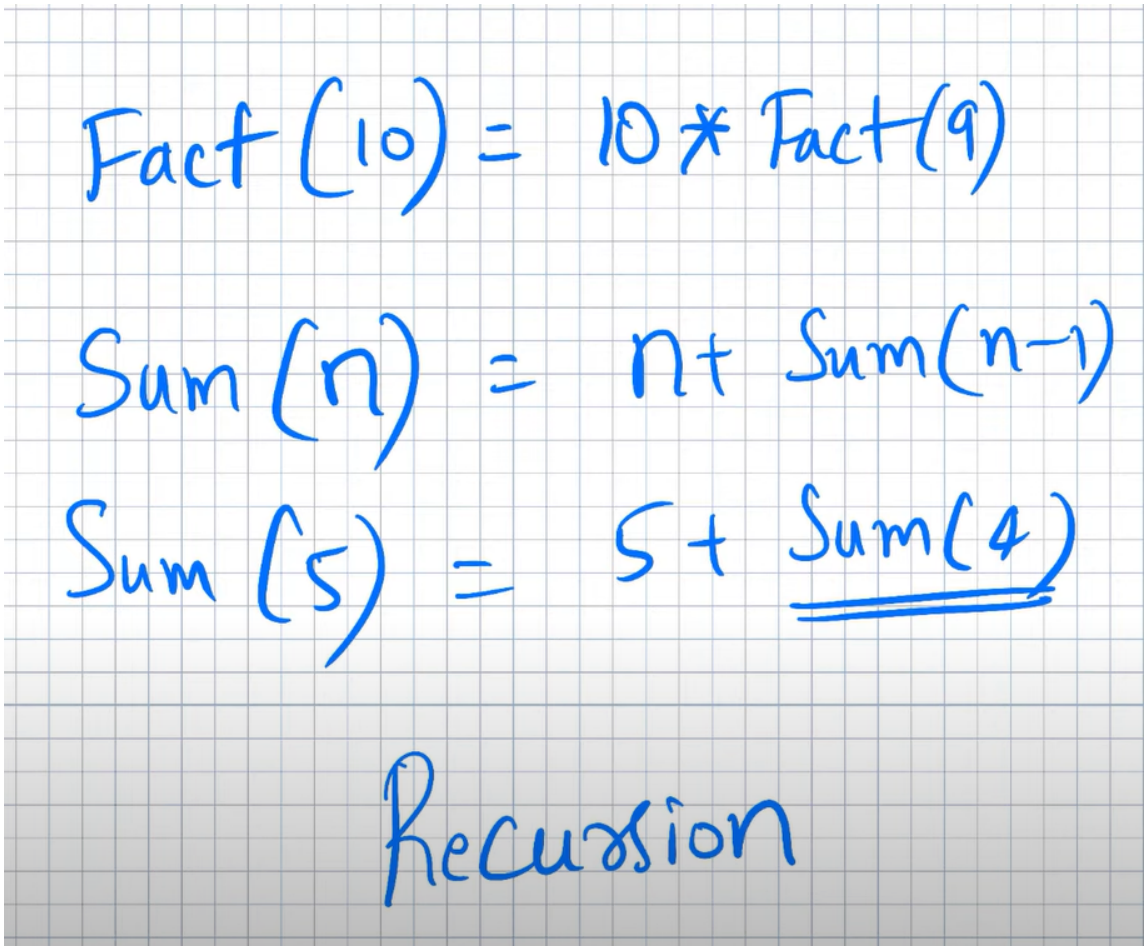
▼ Recursion: A simple question



This isn't in correct python syntax. This program checks if a list has 0 in it and if it does it returns 1 or else 0.



This is the logic behind the above pseudocode.



Revision of concepts done before.

▼ Recursion: Find 0 in a list

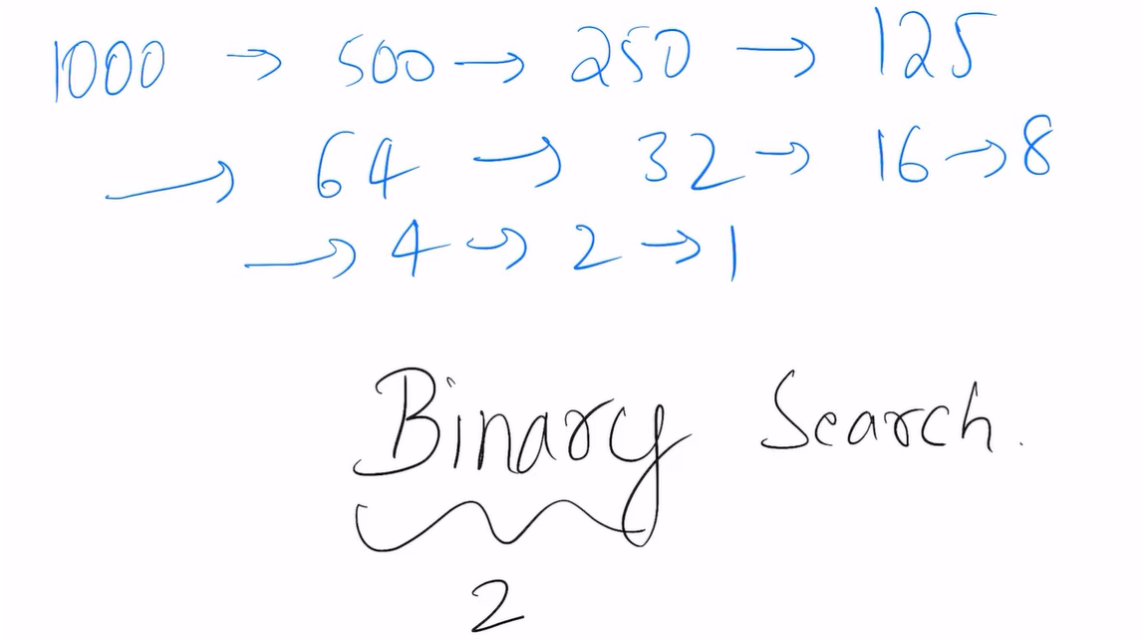
```
def check0(l):
    #if the list is empty, return False
    if (len(l)==0):
        return 0
    if(l[0]==0):
        #if the first element is zero then return 1
        return 1
    else:
        return check0(l[1:len(l)])
        #the above code simply outsources
print(check0([1,2,3,4,5,532,0,24,16,74,8]))
```

▼ Sorting Recursively

```
def mini(l):
    mini=l[0]
    for x in l:
        if(x<mini):
            mini=x
    return mini

def sort(l):
    '''recursively sort the list l'''
    if(l==[])or(len(l)==1):
        return l
    #if the list is empty, there is nothing to sort hence we return the list as it is
    m=mini(l)
    #m now contains the minimum most element in l
    l.remove(m)
    #we remove that element from l
    return [m]+sort(l)
```

```
    #we recursively sort the smaller list

l=[5,7,2,9,1,6,3]
print(sort(l))
```

▼ Introduction to Binary Search



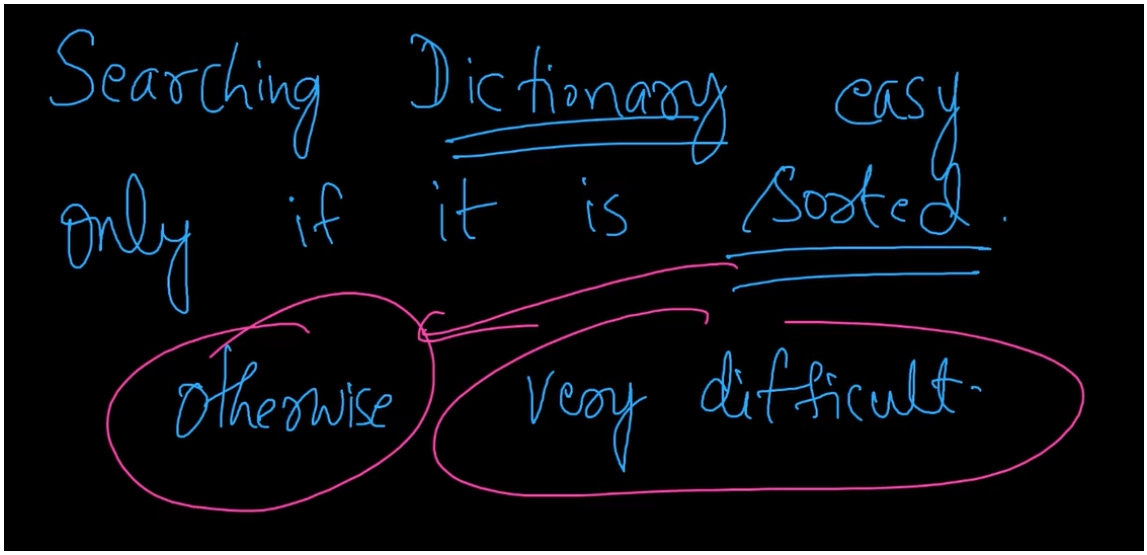
For example - Lets consider you've to find the word 'serendipity' in a dictionary. You start out with opening the middle page of the dictionary. If it's of a letter that comes before the letter S you can discard half of the dictionary. Repeating this method until we find our word is called Binary Search.

▼ Warm up for Binary Search

```
def obvious_search(L,k):
    for x in L:
        if x==k:
            return 1
    return 0
    #code verified.
L=list(range(100000000))
import time
a=time.time()
#Measures time from an epoch to this moment.
print(obvious_search(L,9999999999))
b=time.time()
print(b-a)
```

- When we search for a closer number to the limit of the big list it takes a considerable time for our code to find that number and get the output.
- This is obvious search. We'll use binary search next.

▼ Binary Search Implementation




```
import time

def binarysearch(L,k):
    '''This function is an alternative fot the obvious search.
        It does exactly what is ecpected from the obvious search,
        but in an efficient way.
        This method is popularly called the binary search'''
        #We want to shrink our list.
        #We will do that using a while loop.

    begin = L[0]
    end=(len(L)-1)

    #Use a while loop to look at the list and keep halving it.
    while((end-begin)>1):

        #Compute the mid which is the mid point of begin to end.
        mid=(begin+end)//2

        #If mid is indeed k, then we return True and stop the code.
        if(L[mid]==k):
            return 1

        '''If the middle element is greater than k,
        then cut the right side and retain the left side.'''
        if (L[mid]>k):
            end=mid-1

        '''If the middle element is less than k,
        then cut the left side and retain the right side.'''
        if (L[mid]<k):
            begin=mid+1

        '''This is outside the while loop. If we are here, it means that we
        haven't found the element. Also, if we are here, it means that the
        while condition is violated. Which means end-begin is less than or equal to 1.'''

        #If it is equal to 1, then there are exactly two elements.
    if(L[begin]==k) or (L[end]==k):
        return 1
    else:
        return 0

L=list(range(100*100*100))
a=time.time()
print(binarysearch(L,-1))
b=time.time()
print(b-a)
```

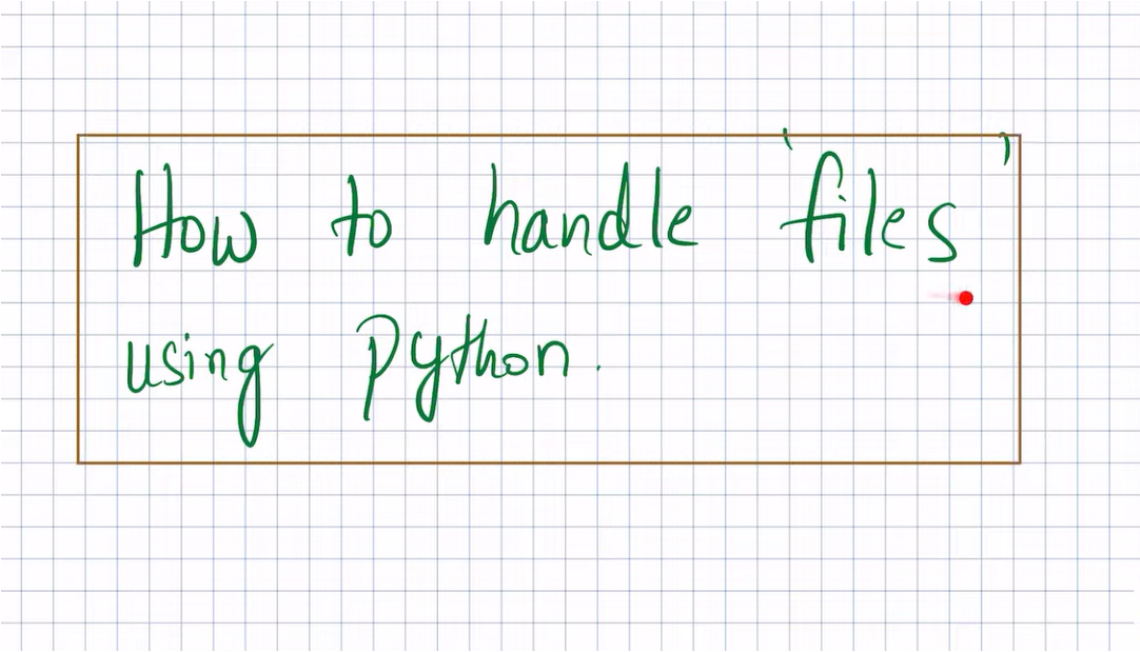
▼ Binary Search Recursion Way

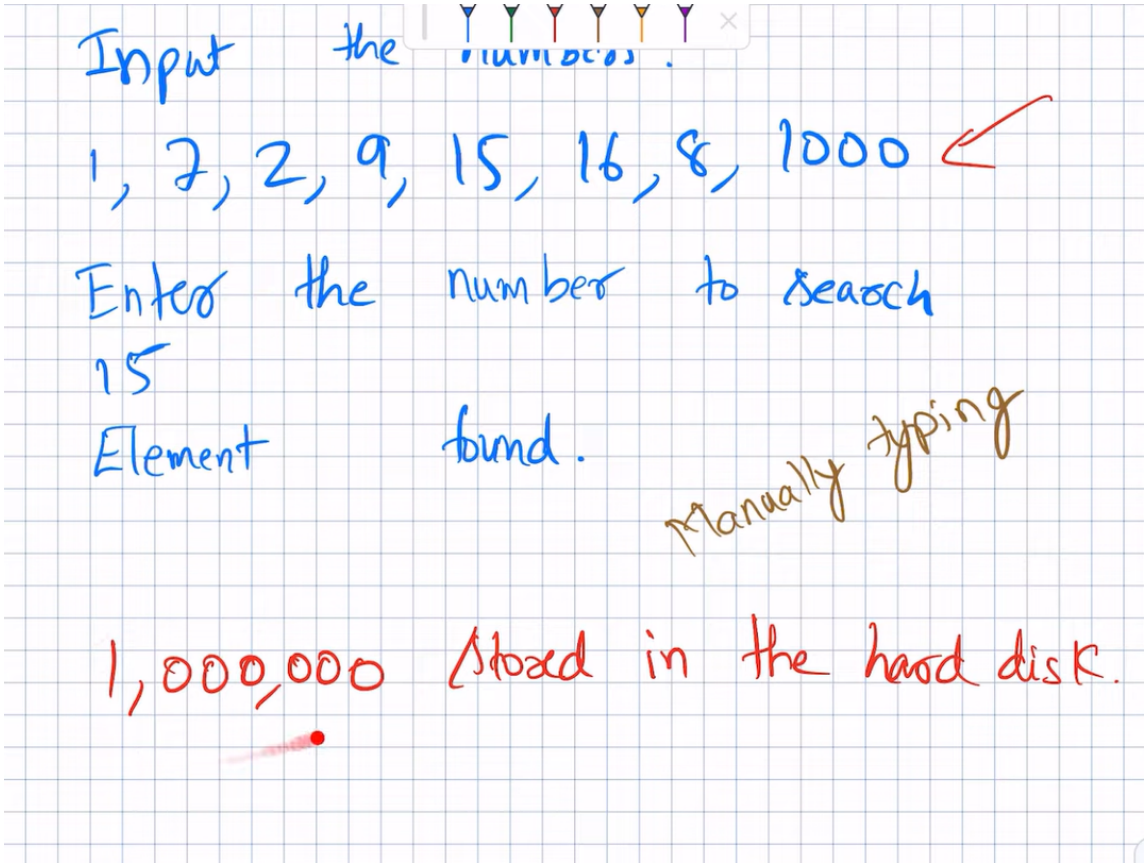
```
#Binary search but recursion way

def rbinarysearch(L,k,begin,end):
    '''This will recursively compute binary search'''
```

```
    '''If begin and end are the same, then we need to just
    check L[begin]'''
    if (begin==end):
        if (L[begin]==k):
            return 1
        else:
            return 0
    '''If begin and end are consecutive, then check them
    individually'''
    if (end-begin==1):
        if (L[begin]==k) or (L[end]==k):
            return 1
        else:
            return 0
    '''If end-begin>1'''
    if (end-begin>1):
        mid=(begin+end)//2
        if (L[mid]>k):
            #discard the right and retain the left.
            end=mid-1
        if (L[mid]<k):
            #discard the left and retain the right.
            begin=mid+1
        if(L[mid]==k):
            return 1
        if(end-begin<0):
            return 0

    return rbinarysearch(L,k,begin,end)

L=list(range(10000))
k=-33
begin=L[0]
end=len(L)-1
print(rbinarysearch(L,k,begin,end))
```

▼ WEEK 8 - 76-83 LEC:

▼ Introduction to File Handling



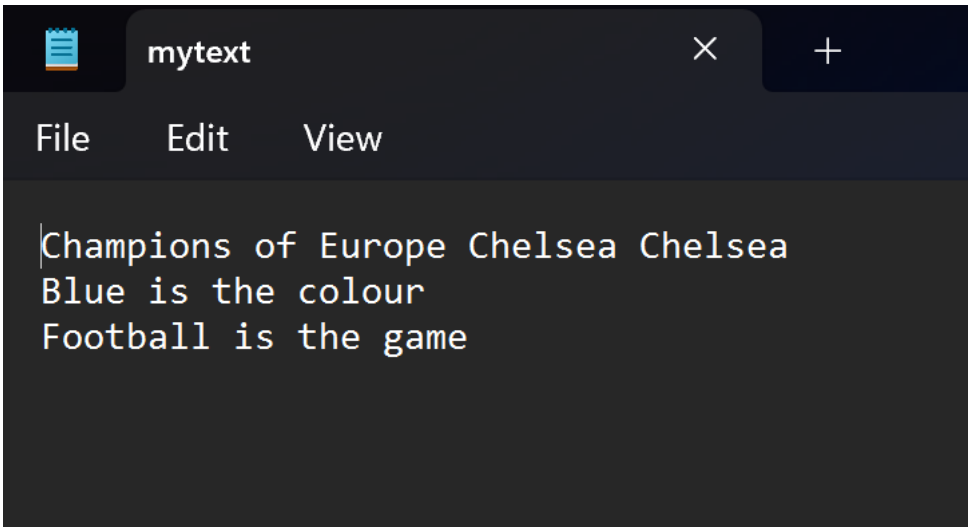


▼ Reading and Writing to a File

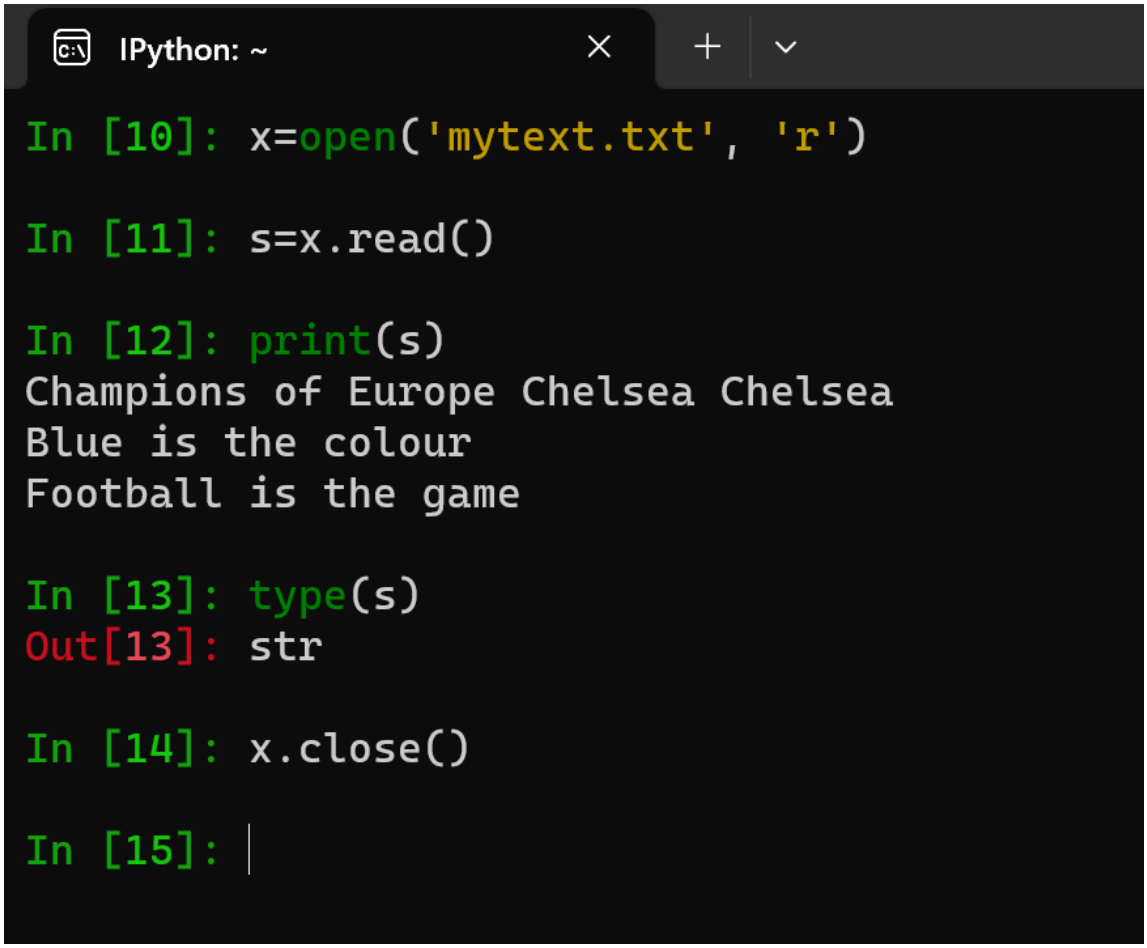
```
Python 3.11.4 (tags/v3.11.4:d2340ef, Jun  7 2023, 05:45:37) [MSC v.1934 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.14.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: cd
C:\Users\amaan

In [2]: f=open('mytext.txt', 'w')

In [3]: f.write('Champions of Europe ')
Out[3]: 20

In [4]: f.write('Chelsea Chelsea ')
Out[4]: 16

In [5]: f.write('\n')
Out[5]: 1

In [6]: f.write('Blue is the colour')
Out[6]: 18

In [7]: f.write('\n')
Out[7]: 1

In [8]: f.write('Football is the game')
Out[8]: 20

In [9]: f.close()
```

- Here, f is a variable to which we've assigned the open command. `'mytext.txt'` is the name of the file and `'w'` stands for write.
- We then use the variable f to call the file and write in it using `f.write()` command.

- `'\n'`
- When we are done writing we close the file using `f.close()`.



- Our text got written in the mytext.txt file.



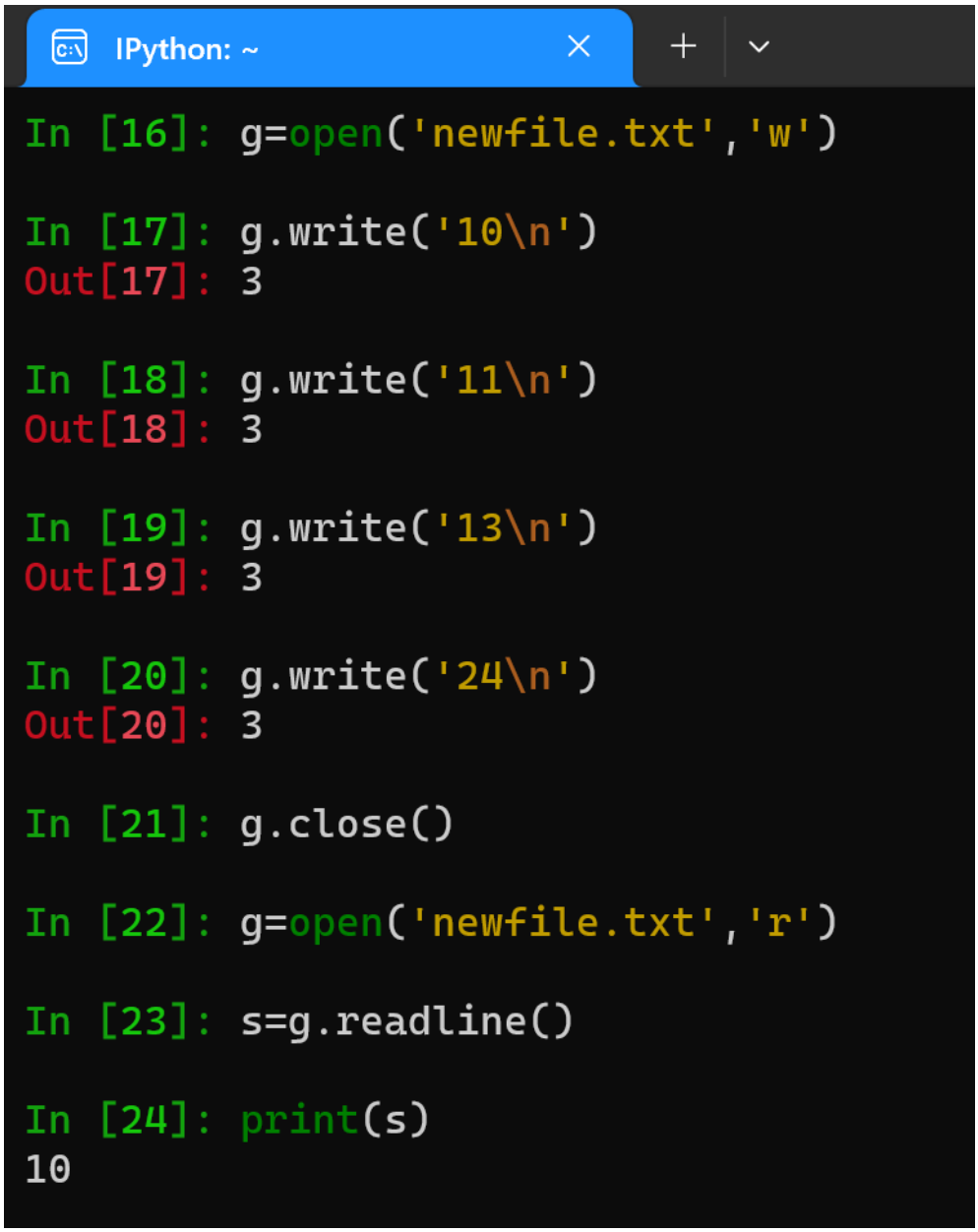
- Here, we assign the open command of our file to the variable `x`.
- Then, we assign the read command of `x` file on the variable `s`.
- `print(s)` prints whatever is in the mytext.txt file.

- `type(s)` prints the type of the variable `s` which is string in this case.
- `x.close()` closes the file.

> 💡 To clear the terminal use CTRL+L.

> 💡 `'w'` writes over the file. `'a'` appends to the existing file.

▼ Big text file handling



- Here, we first create a new file by the name `newfile.txt` then we add text to it.
- `readline` command is used to read only 1 line of the file.

```
Console 1/A

In [11]: g=open('newfile.txt','r')

In [12]: s=g.readline()

In [13]: print(s)
10


In [14]: s=g.readline()

In [15]: print(s)
11


In [16]: s=g.readline()

In [17]: print(s)
13


In [18]: s=g.readline()

In [19]: print(s)
24


In [20]: s=g.readline()

In [21]: print(s)


In [22]: s==''
Out[22]: True

In [23]:
```

- When we reach the end of any file it is represented by `''` (null character). Hence `s==''` outputted True as we've reached the end of the file.

```
C:\Users\amaan\findnum.py

findnum.py

1  f=open('newfile.txt', 'r')
2  flag=0
3  s='0'
4  while(s!=''):
5      s=f.readline()
6      if(s!=''):
7          n=int(s)
8          if(n==10):
9              print("The number was found.")
10             flag=1
11             break
12 if (flag==0):
13     print("The number was not found.")
```

```
Console 1/A

In [38]: run findnum.py
The number was found.

In [39]:
```

- We run this program to find a number in our text file. `run` command is followed by the name of the file which we want to run. (We need to make sure the file we have to run is in the same directory as our text file.)

```
f=open('newfile.txt', 'r')
flag=0
s='0'
while(s!=''):
    s=f.readline()
    if(s!=''):
        n=int(s)
        if(n==10):
            print("The number was found.")
            flag=1
            break
if (flag==0):
    print("The number was not found.")
```

▼ Very big files - a tip

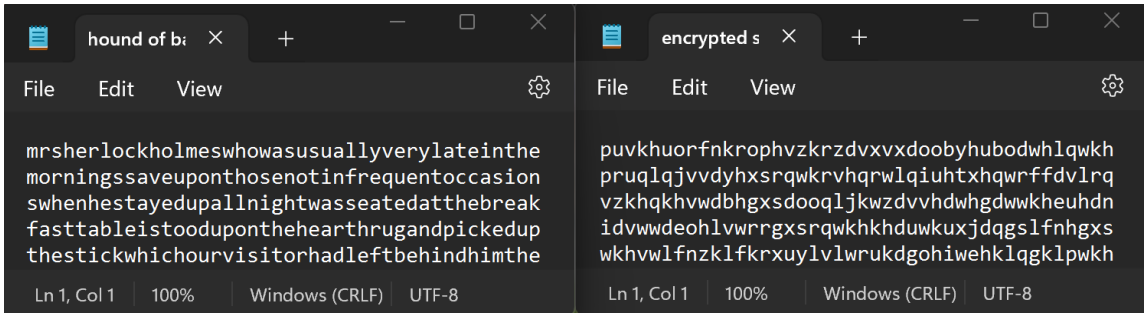
- File handling is possible irrespective of how big the file is.
- Just that it will take a long time to process it.

▼ Caesar Cipher


```
'''This program considers an input file and
encrypts it by using caesr cipher. By that we
mean, we shift the letters by 3 units. For
example a becomes d, b becomes e and so on..
w becomes z, x becomes a, y becomes b and z
becomes c'''

import string

def create_caesar_dictionary():
    l=string.ascii_lowercase
    l=list(l)
    d={}
    for i in range(len(l)):
        d[l[i]]=l[(i+3)%26]
    return d

f=open('hound of baskervilles.txt', 'r')
g=open('encrypted sherlock.txt', 'w')

d=create_caesar_dictionary()
c=f.read(1)
while (c!=''):
    g.write(d[c])
    c=f.read(1)
```

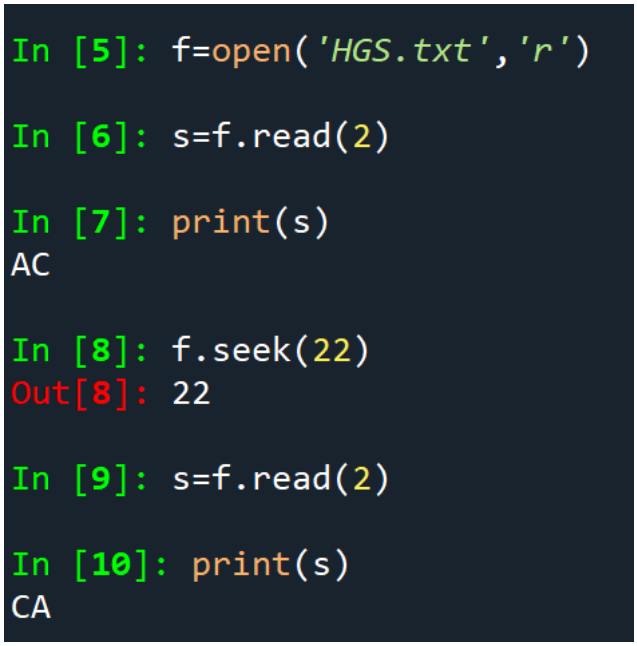
```
f.close()
g.close()
```

- The code begins by importing the `string` module, which provides various useful string-related functions and constants.
- The `create_caesar_dictionary()` function is defined. It initializes an empty dictionary `d` and a list `l` containing all the lowercase letters of the alphabet.
- The function then iterates over the lowercase letters and assigns a new value to each letter in the dictionary `d`. The new value is obtained by shifting the letter three units to the right (using the modulo operator `%` to wrap around to the beginning of the alphabet if needed).
- After defining the `create_caesar_dictionary()` function, the program opens the input file 'hound of baskervilles.txt' in read mode using the `open()` function and assigns it to the file object `f`.
- It also opens a new file 'encrypted sherlock.txt' in write mode using `open()` and assigns it to the file object `g`.
- The program then calls the `create_caesar_dictionary()` function to create the Caesar cipher dictionary and assigns it to the variable `d`.
- The program enters a loop that reads one character at a time from the input file `f` using `f.read(1)`. The loop continues until the end of the file is reached (denoted by an empty string `''`).
- Inside the loop, the program writes the corresponding encrypted character for the current character to the output file `g` using `g.write(d[c])`.
- Once the loop is finished, the program closes both the input and output files using `f.close()` and `g.close()` respectively, ensuring that all changes are saved.



▼ File handling, Genetic Sequences

- `f.seek()` command is used to seek characters ahead or backwards. It isn't efficient as it moves linearly.



- In the following code:
  - First, we opened our file containing Human Genome Sequence.
  - Secondly, we made a string `seq` that will contain whole of the text of our file using `seq=f.read()` command.
  - Then, we checked if our `seq` contains the patterns we want to check and it returned either False or True.

> 💡 `f.read()` works fine for smaller files but takes a lot of time for bigger files. Hence we use KMP(Knuth Morris Pratt) algorithm to find a substring in a given big string.

Console 1/A

```
In [12]: f=open('HGS.txt', 'r')

In [13]: seq=f.read()

In [14]: seq[1]
Out[14]: 'C'

In [15]: seq[0]
Out[15]: 'A'

In [16]: diab='GTATAG'

In [17]: diab in seq
Out[17]: True

In [18]: bp='GAAGTCA'

In [19]: bp in seq
Out[19]: False

In [20]: 'AAAA' in seq
Out[20]: True

In [21]: 'AGAGT' in seq
Out[21]: True

In [22]: 'AAAAAAAA' in seq
Out[22]: False
```

▼ **Why Pandas**

main (copy).py

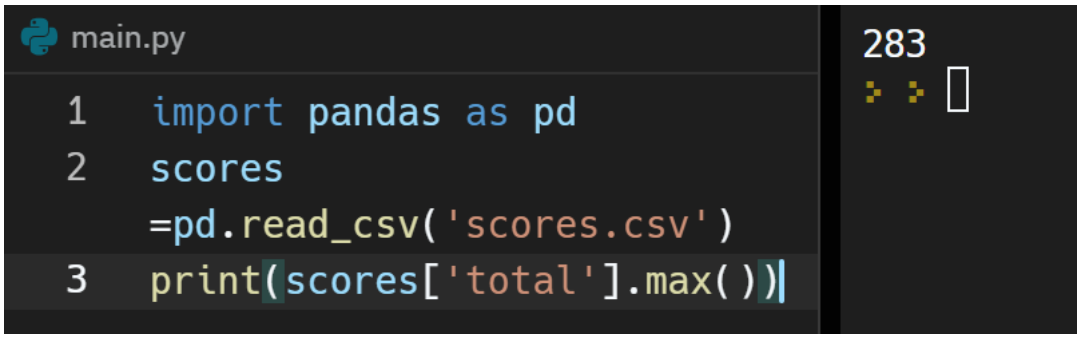
```
1  f=open('scores.csv','r')
2  scores=f.readlines()[1:]
3  max=0
4  for record in scores:
5      fields=record.split(',')
6      if int(fields[8])>max:
7          max=int(fields[8])
8  print(max)
```

Console

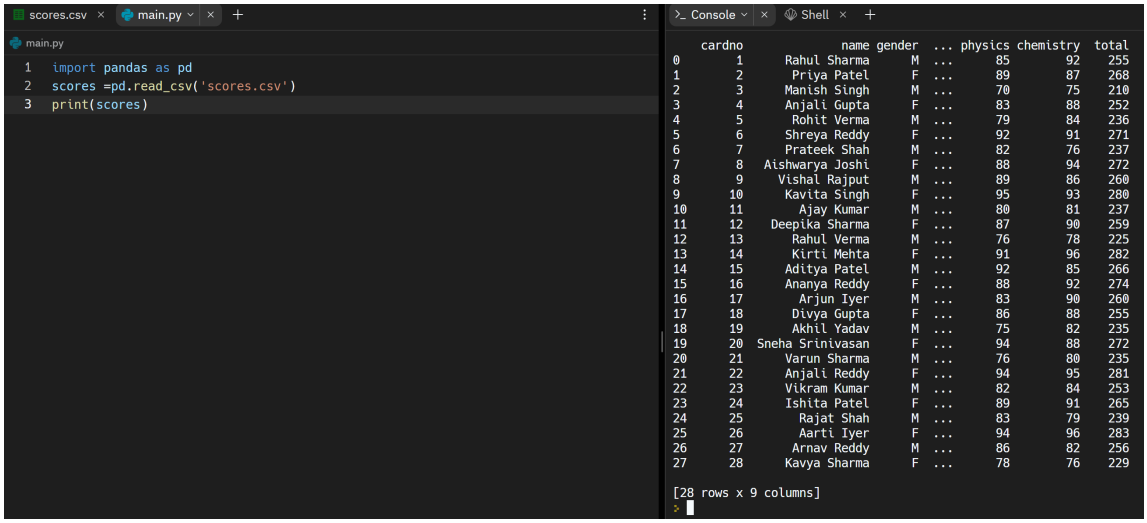
```
283
```

- `[1:]` is a string slicing operation that skips over the header fields in our csv file.

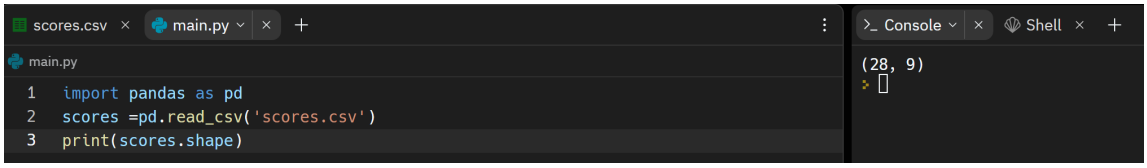
```
f=open('scores.csv','r')
scores=f.readlines()[1:]
max=0
for record in scores:
  fields=record.split(',')
  if int(fields[8])>max:
    max=int(fields[8])
print(max)
```



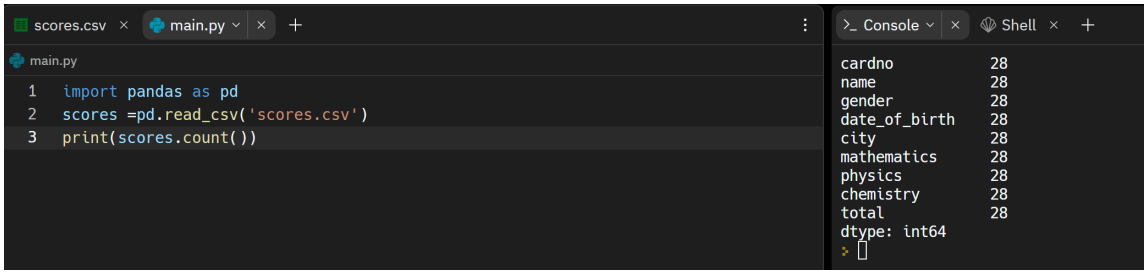
- The above code does what we did with 8 lines of our previous code.



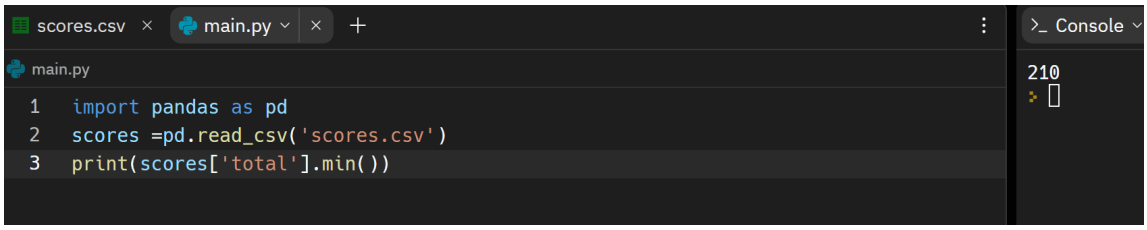
- `print(scores)` prints the scores using pandas. As we can notice, pandas add an additional index column before our first column.
- Another thing to notice is that all fields aren't displayed.
- At the bottom we can see (28 rows x 9 columns) this shows that we have 28 entries in 9 fields.



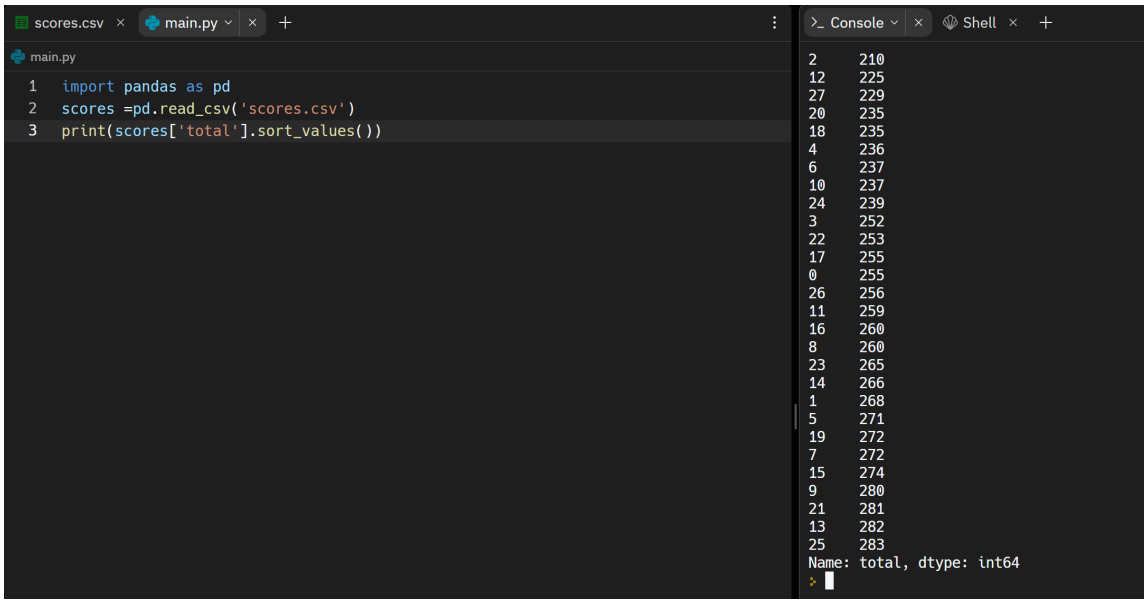
- `print(scores.shape)` returns the shape of our table.



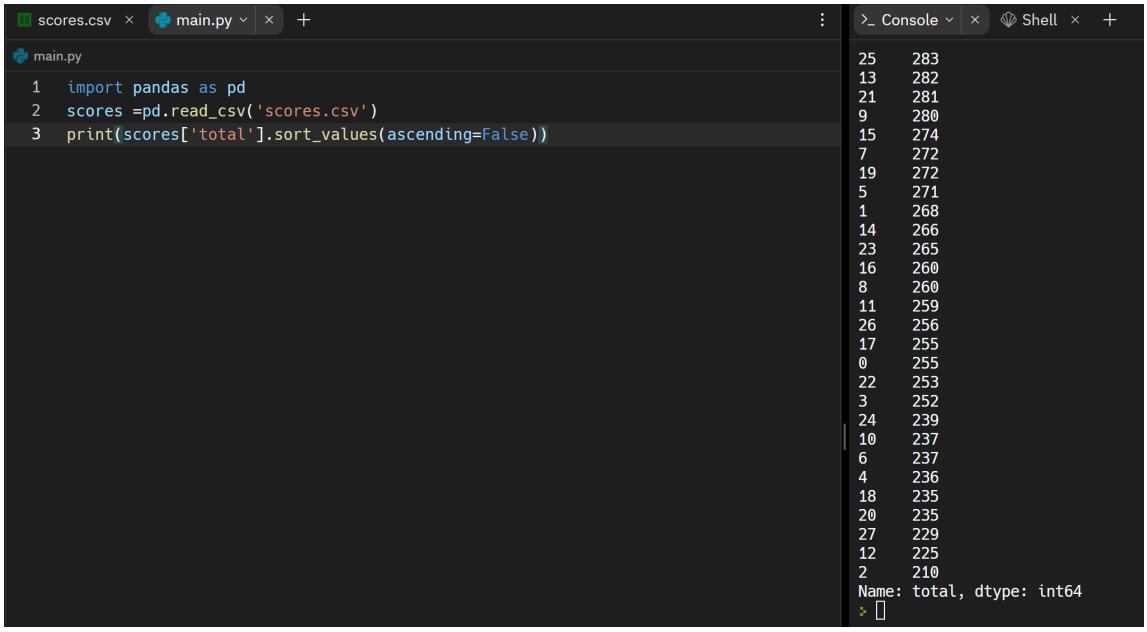
- `print(scores.count())` shows all the values a coulmn contains.



- `print(scores['total'].min())` returns the minimum value of the total column. It works in a similar fashion like the max function.
- Similarly, we can use `.sum()`, `.mean()` etc.



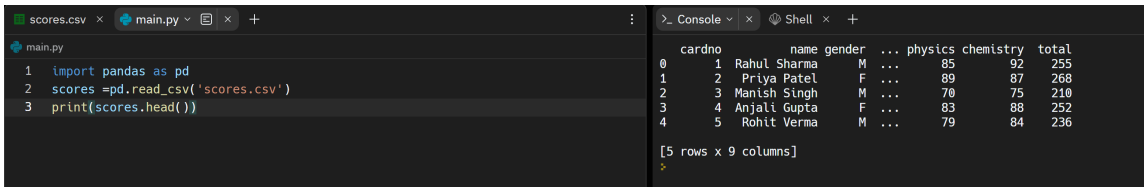
- `.sort_values()` returns the sorted values in ascending order.



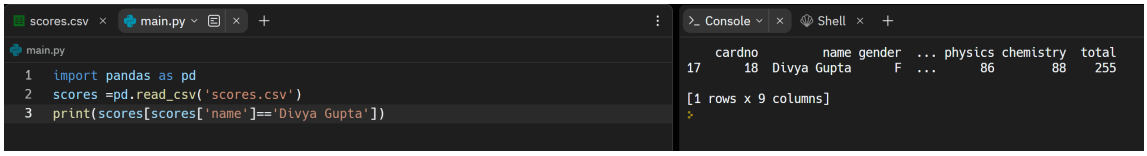
- `.sort_values(ascending=False)` returns the sorted values in descending order.

▼ Pandas Series, DataFrame and more

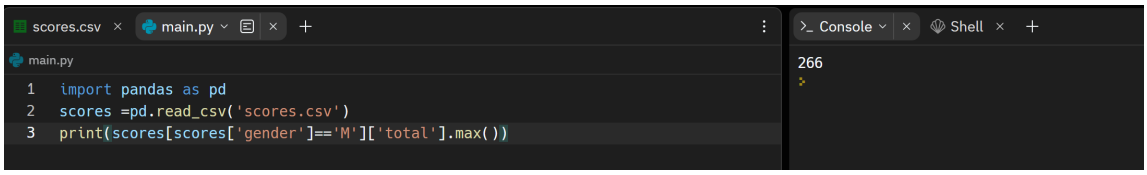
- Scores.csv we used earlier is called a **dataframe** in pandas and every column in it is called a **series**.



- `scores.head()` returns the first 5 rows from the given dataset.
- Similarly `scores.tail()` returns the last 5 rows.



- We can access entire data of a particular individual in a row by using the command `print(scores[scores['name']=='name of individual'])`.
- This way we can access data row-wise.



- `print(scores[scores['gender']=='M']['total'].max())` using this command we can get the boy with the maximum marks.
- Similarly we can find the girl with the most marks too.

```
import pandas as pd
scores =pd.read_csv('scores.csv')
print(scores[scores['physics']>85])
print(scores[scores['physics'].between(70,85)])
print(scores[scores['physics'].between(60,70)])
print(scores[scores['physics']<60])
```

```
    cardno             name gender  ... physics chemistry  total
1        2      Priya Patel      F  ...      89        87    268
5        6     Shreya Reddy      F  ...      92        91    271
7        8   Aishwarya Joshi      F  ...      88        94    272
8        9     Vishal Rajput      M  ...      89        86    260
9       10      Kavita Singh      F  ...      95        93    280
11      12    Deepika Sharma      F  ...      87        90    259
13      14       Kirti Mehta      F  ...      91        96    282
14      15      Aditya Patel      M  ...      92        85    266
15      16      Ananya Reddy      F  ...      88        92    274
17      18       Divya Gupta      F  ...      86        88    255
19      20  Sneha Srinivasan      F  ...      94        88    272
21      22      Anjali Reddy      F  ...      94        95    281
23      24      Ishita Patel      F  ...      89        91    265
25      26        Aarti Iyer      F  ...      94        96    283
26      27       Arnav Reddy      M  ...      86        82    256

[15 rows x 9 columns]
    cardno            name gender  ... physics chemistry  total
0        1   Rahul Sharma      M  ...      85        92    255
2        3   Manish Singh      M  ...      70        75    210
3        4   Anjali Gupta      F  ...      83        88    252
4        5    Rohit Verma      M  ...      79        84    236
6        7   Prateek Shah      M  ...      82        76    237
10      11     Ajay Kumar      M  ...      80        81    237
12      13    Rahul Verma      M  ...      76        78    225
16      17     Arjun Iyer      M  ...      83        90    260
18      19    Akhil Yadav      M  ...      75        82    235
20      21   Varun Sharma      M  ...      76        80    235
22      23   Vikram Kumar      M  ...      82        84    253
24      25     Rajat Shah      M  ...      83        79    239
27      28   Kavya Sharma      F  ...      78        76    229

[13 rows x 9 columns]
   cardno           name gender  ... physics chemistry  total
2       3   Manish Singh      M  ...      70        75    210

[1 rows x 9 columns]
Empty DataFrame
Columns: [cardno, name, gender, date_of_birth, city, mathematics, physic
, chemistry, total]
Index: []
```

- Panda provides us with a `.between()` command that acts as a relational operator

```
import pandas as pd
scores =pd.read_csv('scores.csv')
print(scores[scores['physics']>85].shape[0])
print(scores[scores['physics'].between(70,85)].shape[0])
print(scores[scores['physics'].between(60,70)].shape[0])
print(scores[scores['physics']<60].shape[0])
```

```
15
13
1
0
```

- `.shape` returns a tuple containing the number of rows and columns of that dataframe.
- So, `shape[0]` is used since we only need rows and the index of rows is 0 in the tuple.

```
import pandas as pd
scores =pd.read_csv('scores.csv')
print(scores[(scores['physics']>85) & (scores['gender']=='F')])
```

```
    cardno             name gender  ... physics chemistry  total
1        2      Priya Patel      F  ...      89        87    268
5        6     Shreya Reddy      F  ...      92        91    271
7        8   Aishwarya Joshi      F  ...      88        94    272
9       10      Kavita Singh      F  ...      95        93    280
11      12    Deepika Sharma      F  ...      87        90    259
13      14       Kirti Mehta      F  ...      91        96    282
15      16      Ananya Reddy      F  ...      88        92    274
17      18       Divya Gupta      F  ...      86        88    255
19      20  Sneha Srinivasan      F  ...      94        88    272
21      22      Anjali Reddy      F  ...      94        95    281
23      24      Ishita Patel      F  ...      89        91    265
25      26        Aarti Iyer      F  ...      94        96    283

[12 rows x 9 columns]
```

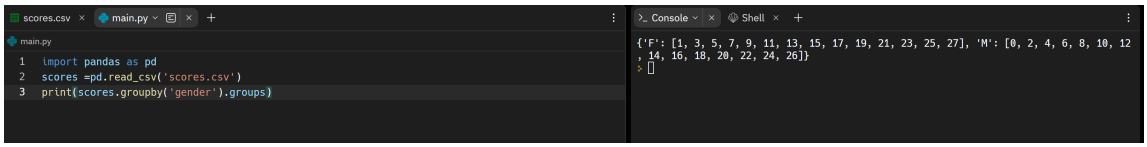
- Pandas has its own **and** operator and it is '**&**'.
- It can be used to condtionalize our command to check two conditions from two different columns or panda series.

```
import pandas as pd
scores =pd.read_csv('scores.csv')
subject = ['mathematics', 'physics', 'chemistry']
for sub in subject:
  print('Above 85 in', sub)
  print(scores[(scores[sub]>85) & (scores['gender']=='M')].shape[0])
  print(scores[(scores[sub]>85) & (scores['gender']=='F')].shape[0])
```

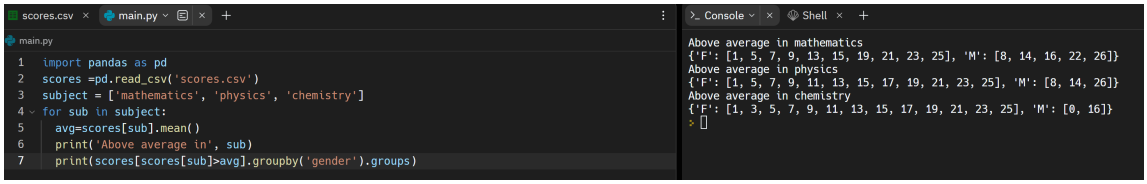
```
Above 85 in mathematics
4
9
Above 85 in physics
3
12
Above 85 in chemistry
3
13
```

- We can iterate over subjects using this code.
- It returns the number of students, male and female who scored above 85 in each subjects.

```
import pandas as pd
scores =pd.read_csv('scores.csv')
subject = ['mathematics', 'physics', 'chemistry']
for sub in subject:
  avg=scores[sub].mean()
  print('Above average in', sub)
  print(scores[(scores[sub]>avg) & (scores['gender']=='M')].shape[0])
  print(scores[(scores[sub]>avg) & (scores['gender']=='F')].shape[0])
```



- `.groupby` groups the entries on the basis of a given parameter.
- In these dictionaries the numbers are the cardno of the female and male students.



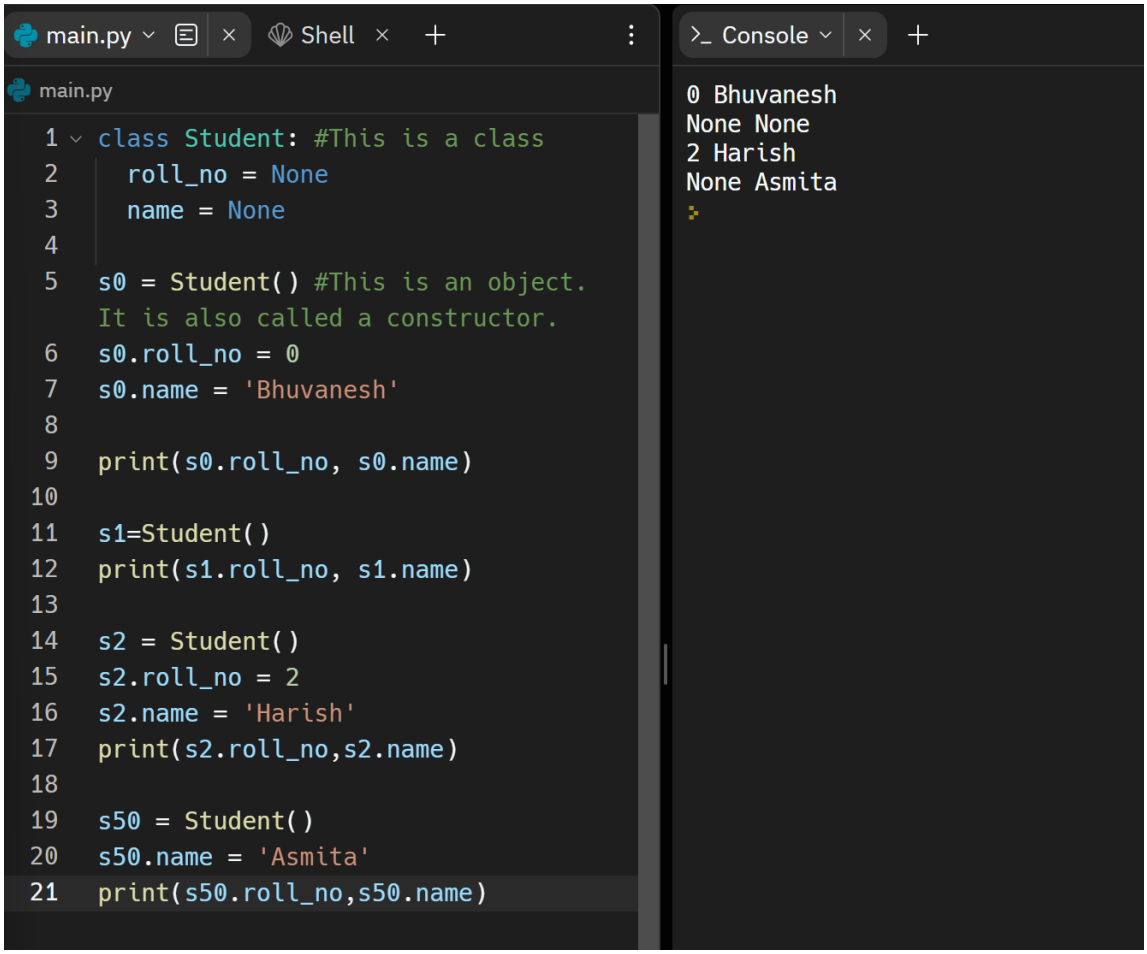
- We can make use of the `.groupby` method to easily make the previous code.

▼ WEEK 9 - 84-89 LEC:

▼ Introduction to Object Oriented Programming

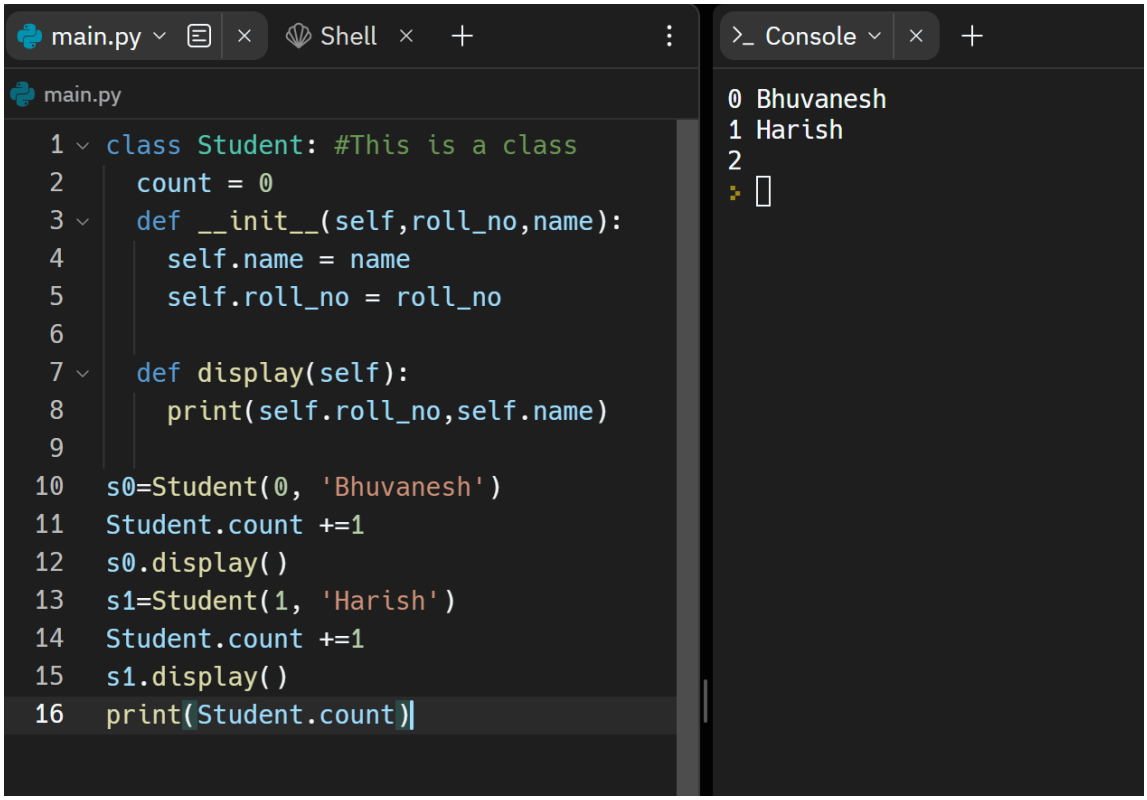
- Everything is an object, every object has attributes and behaviour
  Eg1: A person (object); name, age (attributes); singing, driving (behaviour)
  attributes -> variables; behaviour ->functions

▼ Classes and Objects

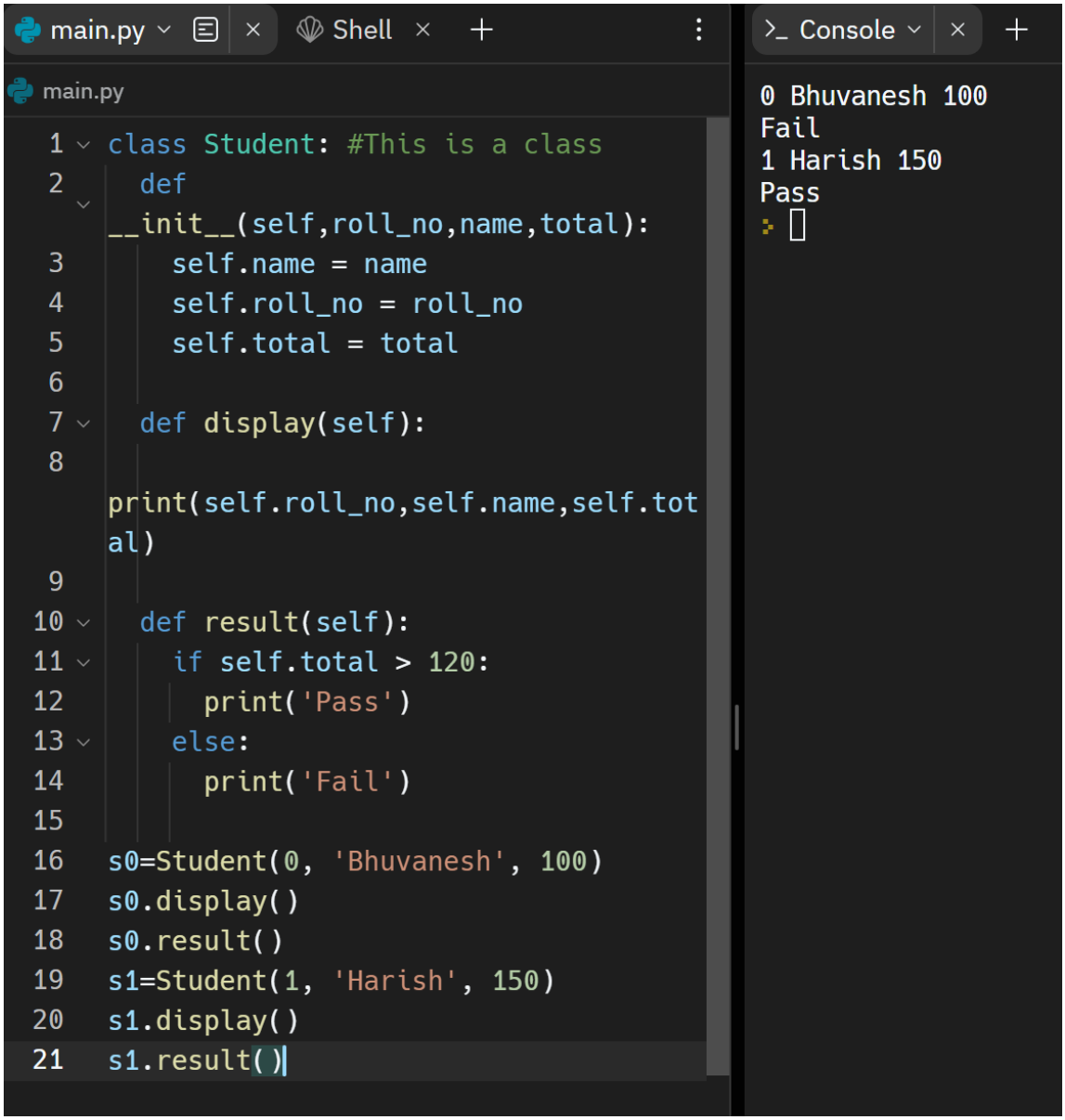


- Here, `Student` is the class, `roll_no`, `name` are attributes and `s0` is an object of the class `Student`.
- We use a dot operator `.` to access the variables of the class in an object.

▼ Attributes and Methods



- Here, `__init__(self, roll_no,name)` is a special method or function. The self parameter stores values like s0, s1, etc.
- If we have some parameters like `roll_no`, `name` which we want to initialize against some variables which are available in the class, then we have to use a constructor with parameters, and whenever we call such a constructor, internally python executes this particular function called `init`.Therefore we don't have to explicitly call this function like other functions.
- `self.name` and `self.roll_no` are object variables which are nothing but attributes of these two objects `s0` and `s1`.
- The variable `count` is owned by the class `Student` and is shared by objects `s1` and `s2`. `s1` and `s2` will not have their own copy of this variable. Hence, `count` is a class attribute and the variables `roll_no` and `name` are object attributes.



- Here, the method `result()` acts as behaviour as discussed in the introduction of this week's lectures.
- When functions belong to a specific class then they are referred to as methods.

▼ Inheritance and Method Overriding

Inheritance

main.py

```python
class Person:
  def __init__ (self, name, age):
    self.name=name
    self.age=age
  def display(self):
    print(self.name, self.age)

class Student(Person):
  def __init__ (self, name, age, marks):
    super().__init__(name, age)
    self.marks=marks
  def display(self):
    super().display()
    print(self.marks)


s= Student('Rida', 20, 250)
s.display()

class Employee(Person):
  def __init__(self, name, age, salary):
    super().__init__(name, age)
    self.salary=salary
  def display(self):
    super().display()
    print(self.salary)


e= Employee('Harsh', 30, 30000)
e.display()
```

```
Rida 20
250
Harsh 30
30000
```

- Here the class `Person` is the parent or super class and the classes `Student` and `Employee` are children classes.
- This way, we can avoid repetition and assign the common attributes to the parent class.

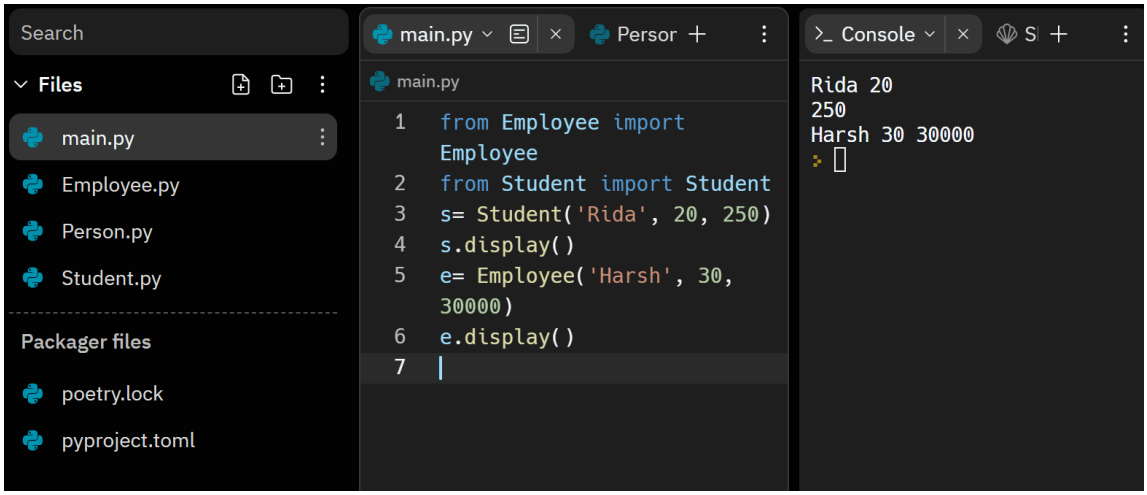
Method Overriding

main.py

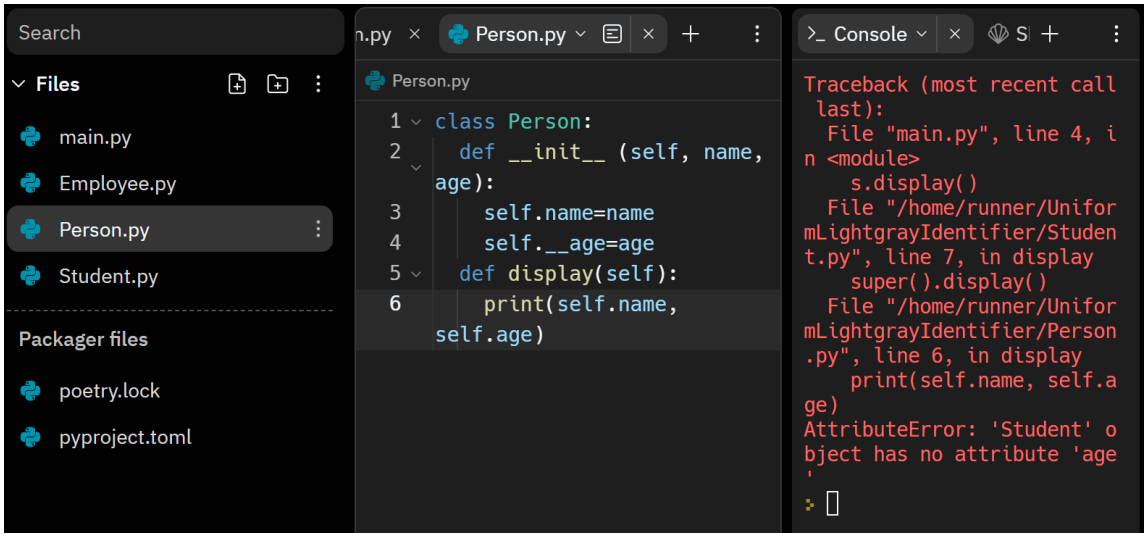
```python
    def display(self):
        print(self.name, self.age)

class Student(Person):
    def __init__ (self, name, age, marks):
        super().__init__(name, age)
        self.marks=marks
    def display(self):
        super().display()
        print(self.marks)


s= Student('Rida', 20, 250)
s.display()

class Employee(Person):
    def __init__(self, name, age, salary):
        super().__init__(name, age)
        self.salary=salary
    def display(self):
        print(self.name,self.age,self.salary)


e= Employee('Harsh', 30, 30000)
e.display()

```

```
Rida 20
250
Harsh 30 30000
```

- This is called **Method Overriding**. As the class employee uses its own function/method instead of the parent class' method.



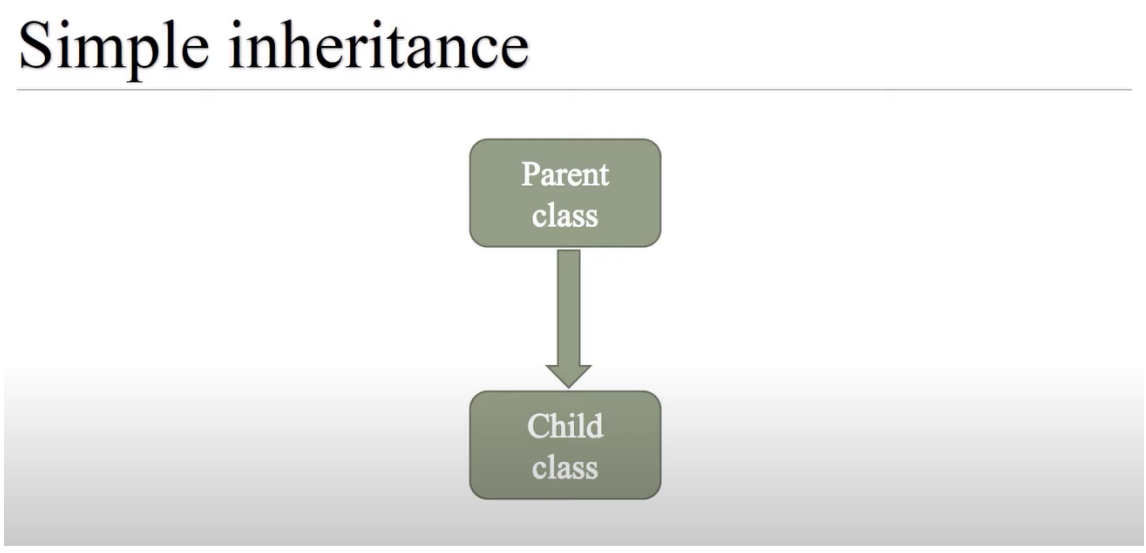
- We can divide our code and make it run this way too in replit.



- This way we can make any variable secret by adding underscores in it. These variables are referred to as **private members**.
- Without the underscores in the variable it becomes accessible to all hence it's called a **public member**.

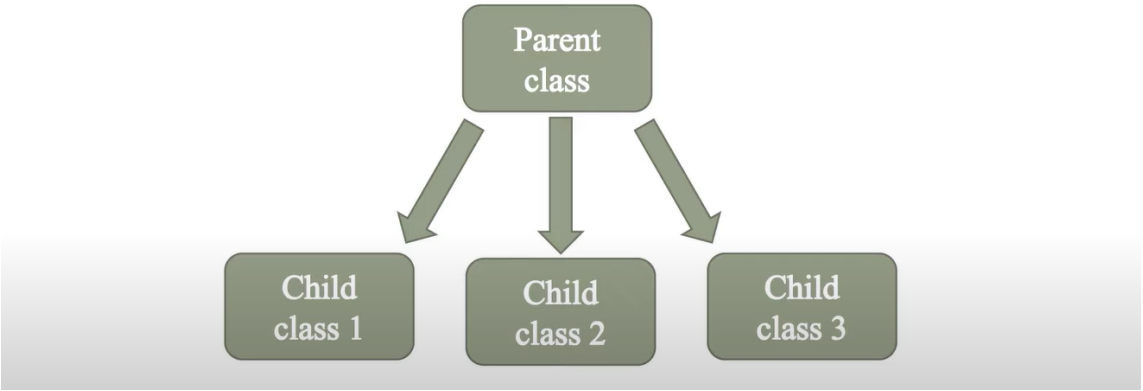
Types of inheritance:

- Simple Inheritance - Single parent and child class.



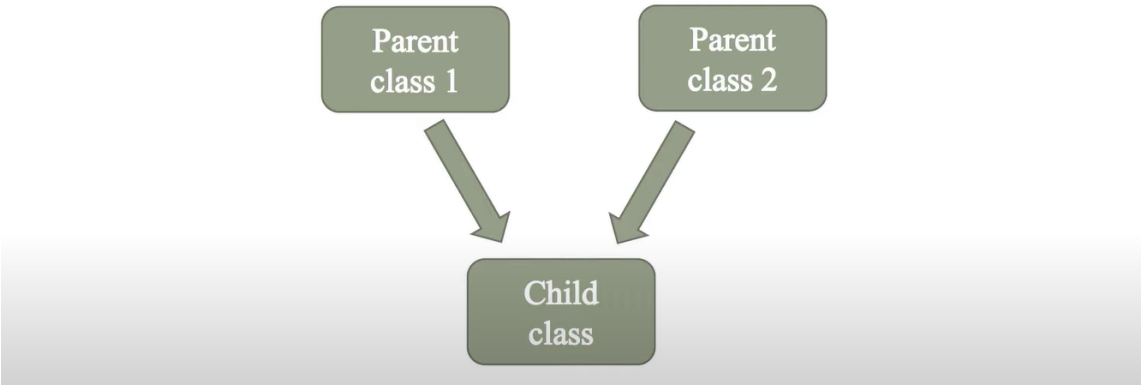
- Hierarchical Inheritance - Multiple children classes. (Ex - Person, Employee and Student classes we worked on previously.)

Hierarchical inheritance



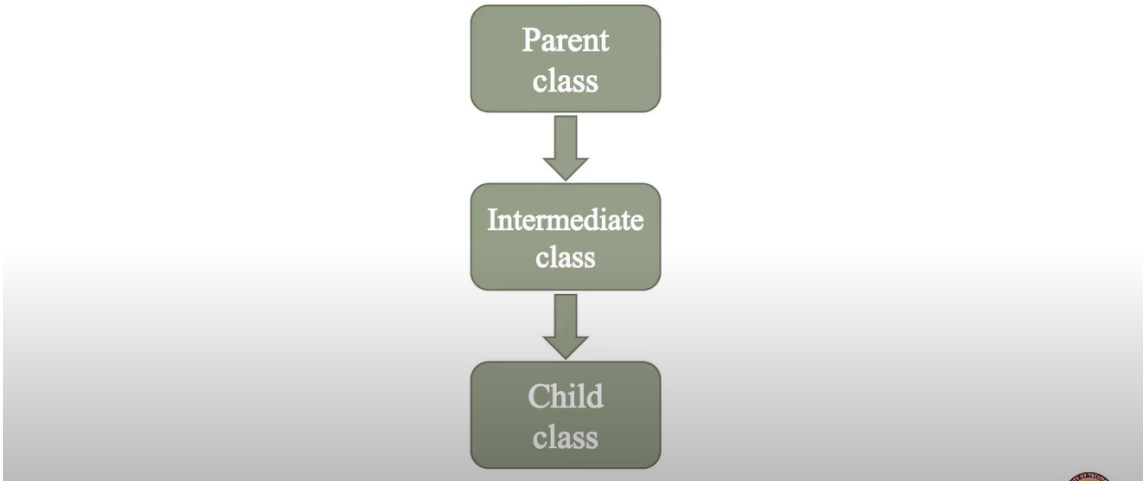
- Multiple Inheritance - Multiple parent classes.

Multiple inheritance



- Multilevel Inheritance - Grandparent, parent and child classes.

Multilevel inheritance

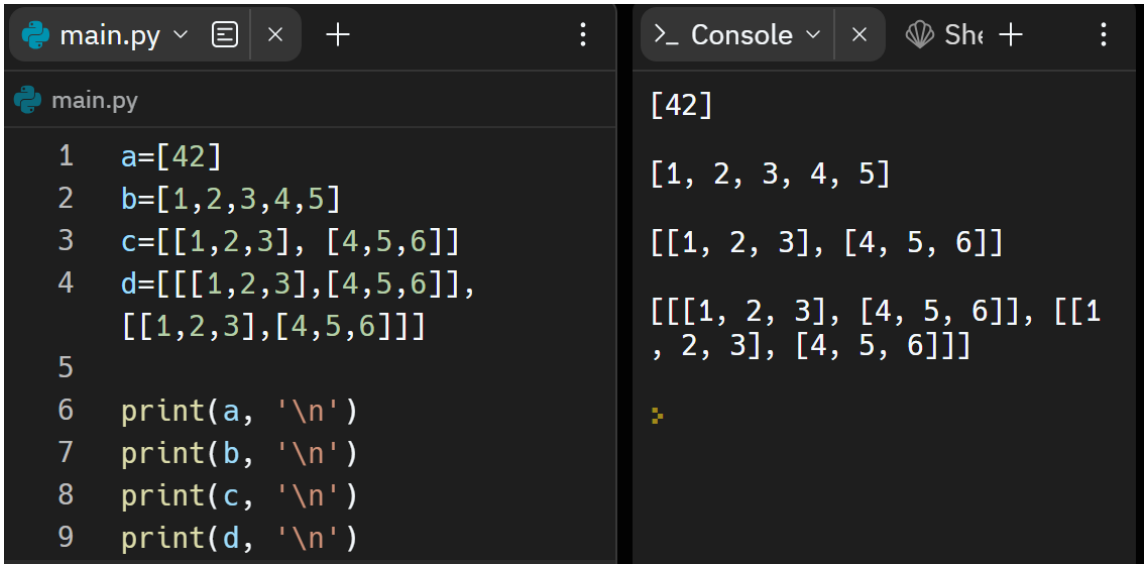


- Hybrid Inheritance - Variation of previous defined inheritances.

Hybrid inheritance

▼ **Introduction to Numpy library**

Parameters	Python list	NumPy array
Installation and importing	Not required	Required
Type of elements	Heterogenous	Homogenous
Dimension of elements	No restriction	Has to be same
Memory allocation	Non-contiguous	Contiguous
Size	Requires more space	Requires less space
Performance	Slower	Faster
Element wise operations	Not possible	Possible
Functionality	Can not handle arithmetic operations	Can handle arithmetic operations



- Python considers all these as one dimensional lists.

whereas,

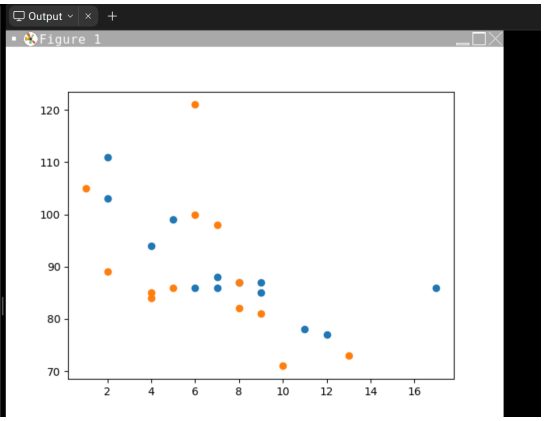
```
main.py
1   import numpy as np
2   a=np.array(42)
3   b=np.array([1,2,3,4,5])
4   c=np.array([[1,2,3],
    [4,5,6]])
5   d=np.array([[[1,2,3],
    [4,5,6]], [[1,2,3],
    [4,5,6]]])
6
7   print(a, a.ndim, '\n')
8   print(b, b.ndim, '\n')
9   print(c, c.ndim, '\n')
10  print(d, d.ndim, '\n')
```

```
42 0

[1 2 3 4 5] 1

[[1 2 3]
 [4 5 6]] 2

[[[1 2 3]
  [4 5 6]]

 [[1 2 3]
  [4 5 6]]] 3
```

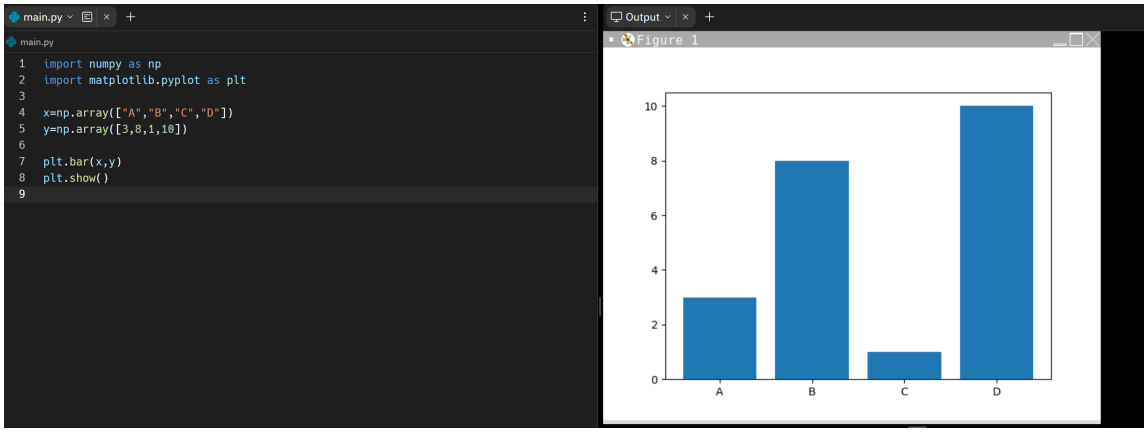
- Using numpy array, python considers these as matrix with their respective dimension.

▼ Introduction to Matplotlib library

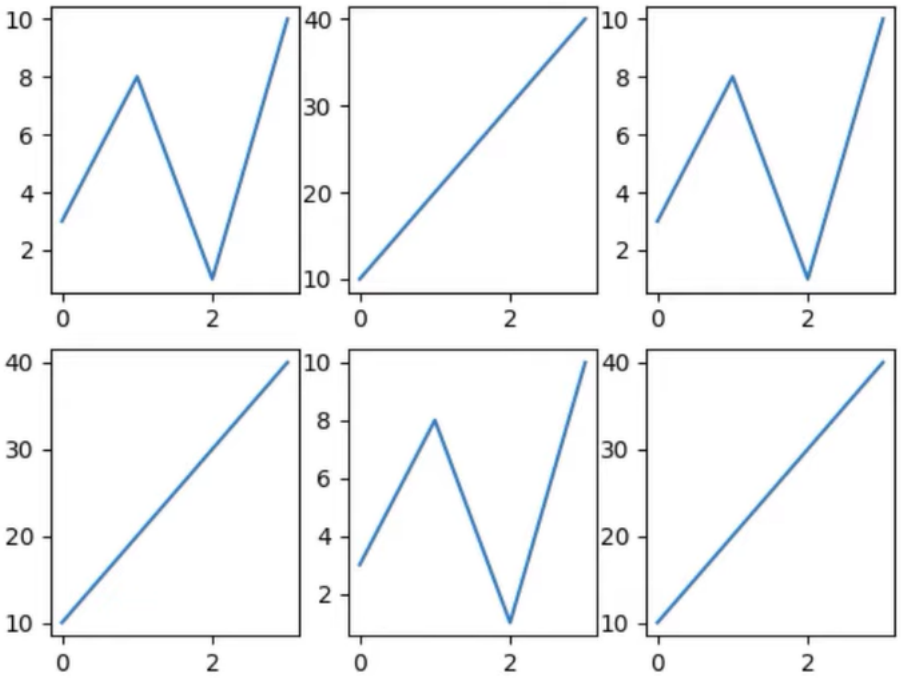
```
main.py
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  x=np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
5  y=np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
6  plt.scatter(x,y)
7  x=np.array([6,4,9,2,6,4,1,5,7,13,10,8,8])
8  y=np.array([100,84,81,89,121,85,105,86,98,73,71,87,82])
9  plt.scatter(x,y)
10 plt.show()
11
```



- `plt.scatter()` is used to make a scatter chart.



- `plt.bar()` is used to create bar chart.
- Similarly, `.hist()` is used to make histogram, `.pie()` for pie chart.



- We can create subplots like these using `plt.subplot()`. This command takes in 3 attributes - row (x axis), column (y axis) and position in the chronological order.
- So to create the above subplots we used `plt.subplot(2,3,1)`, `plt.subplot(2,3,2)`, `plt.subplot(2,3,3)`, `plt.subplot(2,3,4)`, `plt.subplot(2,3,5)`, `plt.subplot(2,3,6)`.

▼ WEEK 10 - 90-95 LEC:

▼ Introduction

We'll learn about **Exception Handling** and **Functional Programming** concepts i.e. :

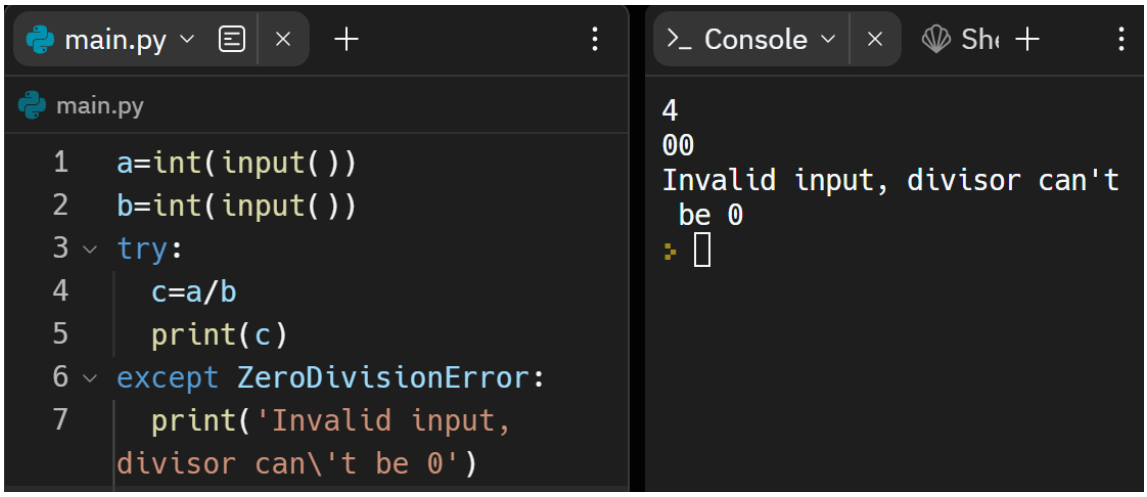
- Iterator
- Generator
- In line statements
- List comprehension

- Lambda function
- Enumerator
- Zip
- Map
- Filter

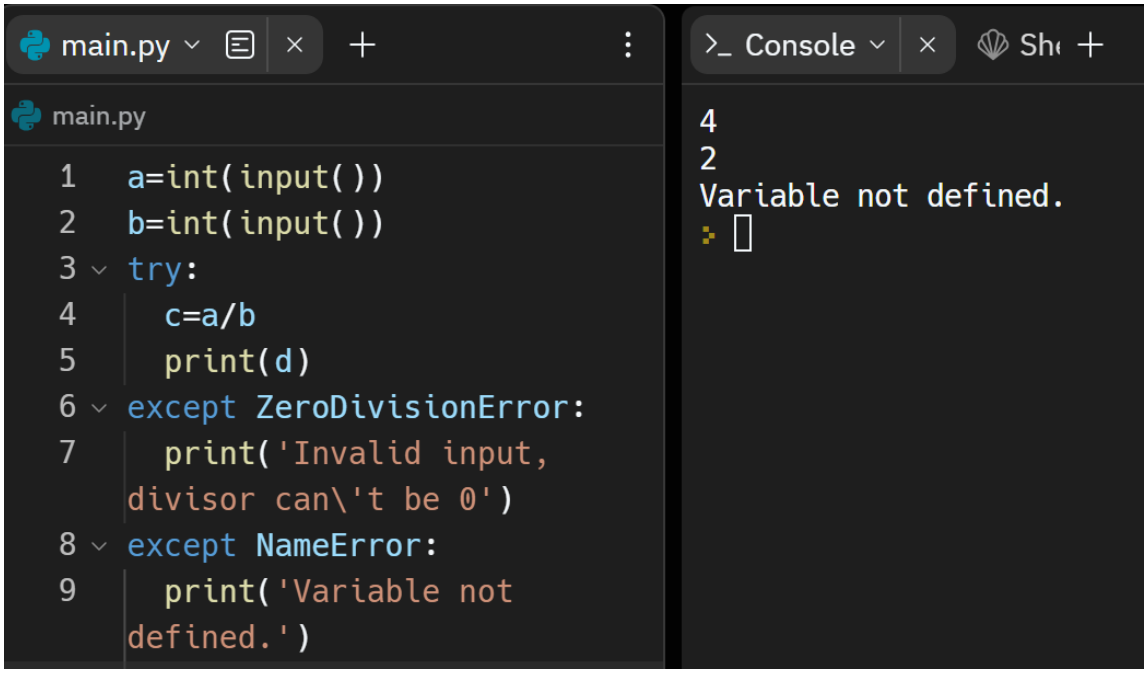
▼ Exception handling

We use exception handling to print out meaningful error messages that may help someone encountering them solve them.

It is considered as an industry standard.



- The `try:` command first tries if the code under it can run without error.
- If it doesn't, the `except:` command runs the code under it.



- This way we can handle **NameError**.
- We can stack up multiple `except` commands.

main.py

```
a=int(input())
b=int(input())
try:
  c=a/b
  print(c)
  f=open('abc.txt','r')
except ZeroDivisionError:
  print('Invalid input, divisor can\'t be 0')
except NameError:
  print('Variable not defined.')
except FileNotFoundError:
  print('Invalid file name. Please check again')
except:
  print('Something went wrong.')
```

Console

```
4
9
0.4444444444444444
Invalid file name. Please check again
```

- This way we can handle the **FileNotFoundError**.
- The last except block is for unforeseen errors apart from the above 3 errors that may arrive.

main.py abc.txt

```
a=int(input())
b=int(input())
try:
  f=open('abc.txt','r')
  c=a/b
  print(c)
except ZeroDivisionError:
  print('Invalid input, divisor can\'t be 0')
except NameError:
  print('Variable not defined.')
except:
  print('Something went wrong.')
finally:
  f.close()
  print('From finally block')
```

Console

```
5
2
2.5
From finally block
```

- The `finally` block is always executed before termination of the program.

main.py abc.txt

```
a=int(input())
if a<18:
  raise Exception('You are underage, can not vote.')

```

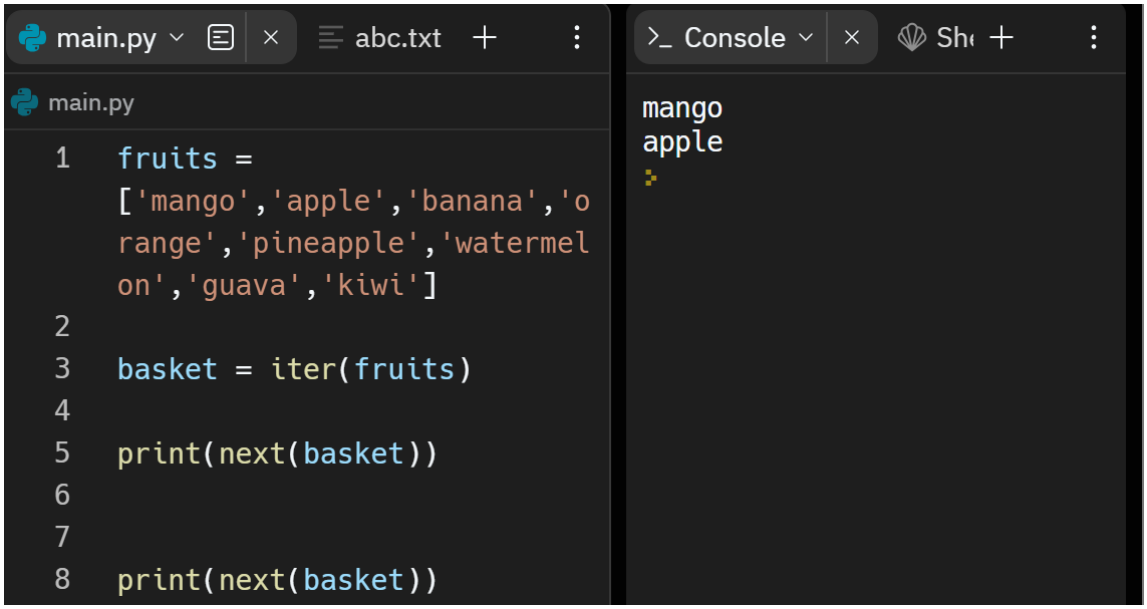
Console

```
6
Traceback (most recent call last):
  File "main.py", line 3, in <module>
    raise Exception('You are underage, can not vote.')
Exception: You are underage, can not vote.
```

- We can create our own exception warnings using `raise Exception` command.

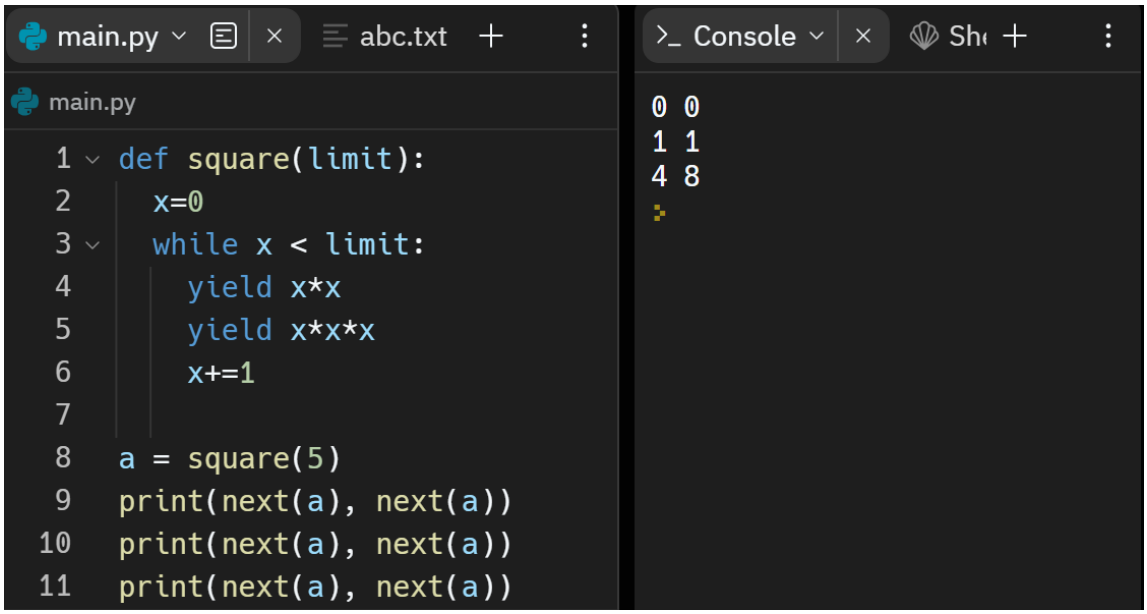
▼ Functional Programming (Part 1)

Iterator:



- `iter` is used to iterate over any iteratable item i.e. string, list, dictionary, tuple etc.
- `next` is used to iterate over that particular iterator if and when required.

Generator:



- `yield` function is called generator. It generates an iterator.
- Here, `a` is an iterator and we can iterate over the iterator using same `next` function.

▼ Functional Programming (Part 2)

In line statements:

```
a=10
b=20

if a<b:
  small=a
else:
  small=b
```

```
a=10
b=20

small = a if a < b else b

print(small)
```

```
print(small)
```

- The **in line statement** does the same job of an if else statement.
- We can reduce the length of our code by using this everywhere possible.

```
a=5
while a>0:
  print(a)
  a-=1
```

```
a=5
while a>0: print(a); a-=1
```

- Python reads both the codes in equal time so there's no time advantage in running either of the code over the other.
- However, our code may become difficult to understand if its big and we write commands in a single line.

List Comprehension:

```
fruits=['mango','apple','pineapple',
'orange', 'watermelon', 'guava',
 'banana', 'kiwi']

newlist=[]
for fruit in fruits:
  if 'n' in fruit:
    newlist.append(fruit.capitalize())

print(newlist)
```

```
fruits=['mango','apple','pineapple','orange', 'watermelon', 'guava',
 'banana', 'kiwi']

newlist=[fruit.capitalize() for fruit in fruits if 'n' in fruit]
print(newlist)
```

- Both of these code give the same output but the second code uses **list comprehension**.
- In it we used **inline for**, **inline if**, and putting them together in a list creating a new list. This is called list comprehension.

▼ Functional Programming (Part 3)

Lambda Function:

Functions that are anonymous i.e. without any function name.

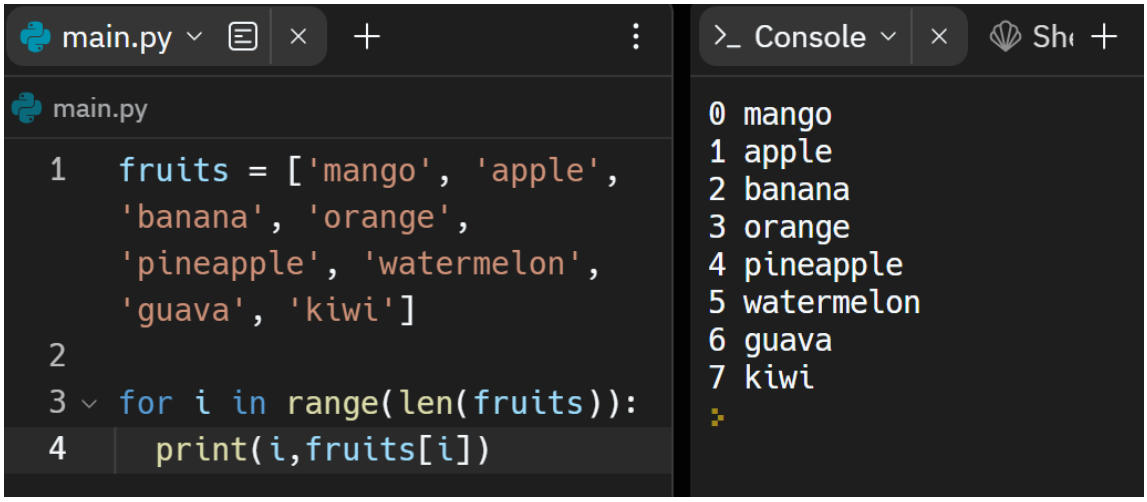
```
def add(x,y):
  return x+y

def sub(x,y):
  return x-y

def mul(x,y):
  return x*y

def div(x,y):
  return x/y

print(add(10,20))
print(sub(10,20))
print(mul(10,20))
print(div(10,20))
```

```
add = lambda x,y: x+y
sub = lambda x,y: x-y
mul = lambda x,y: x*y
div = lambda x,y: x/y

print(add(10,20))
print(sub(10,20))
print(mul(10,20))
print(div(10,20))
```

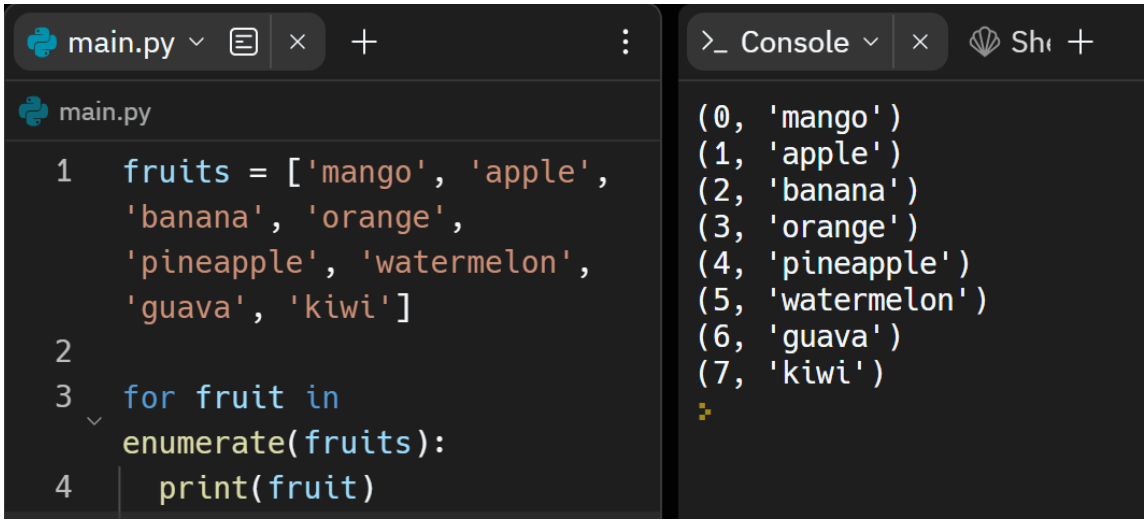
- In the second code we use **lambda functions** instead of common functions.
- Both of the codes give the same output.

- Advantage of using lambda functions is they reduce the length of our program.

Enumerator Function:

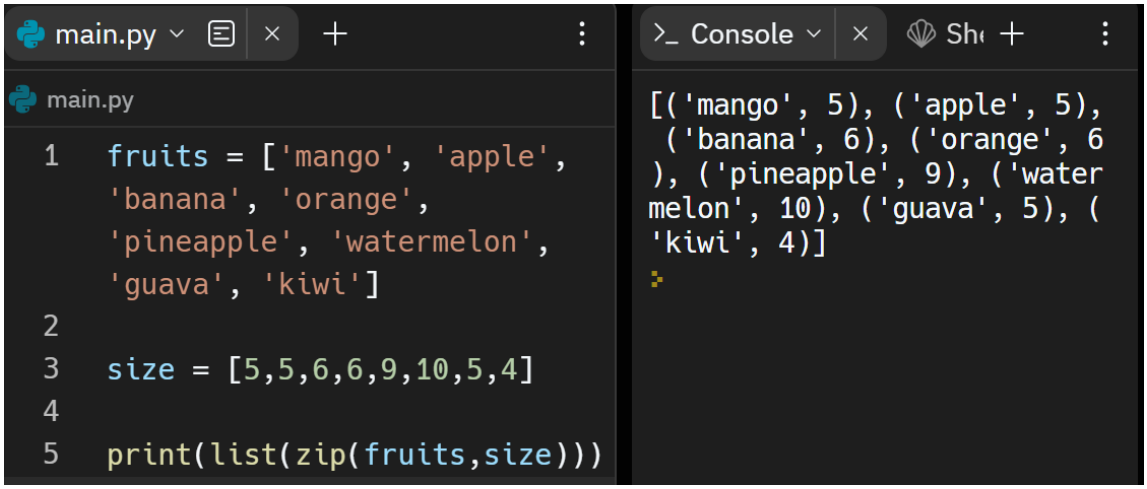


- In this code the index values and the fruits in fruits list aren't coupled.
- The index and the name of the fruit are two different entities here.



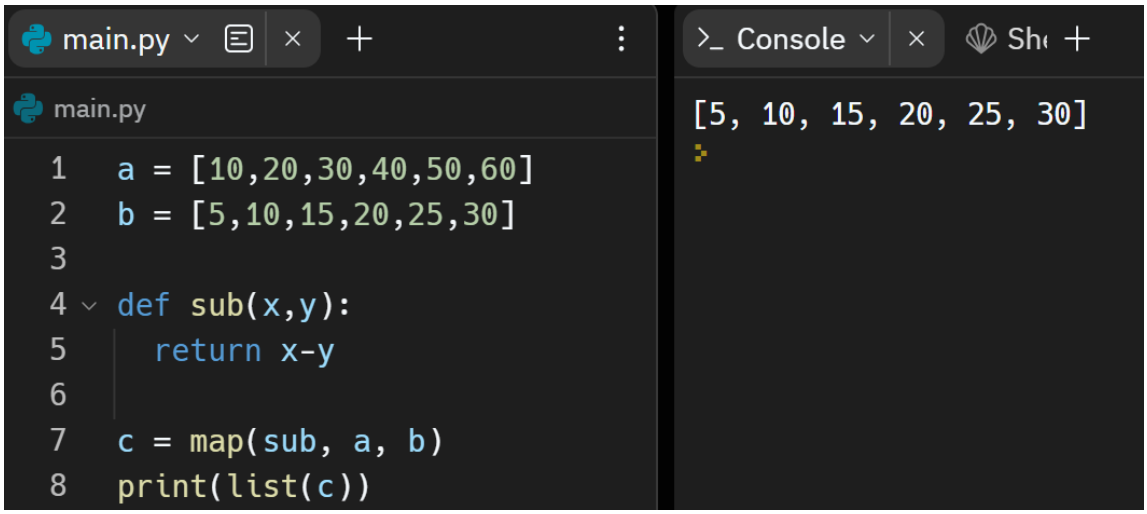
- Using the `enumerate` function, we've coupled the indexes and the name of the fruits. They've become a single entity.

Zip Function:

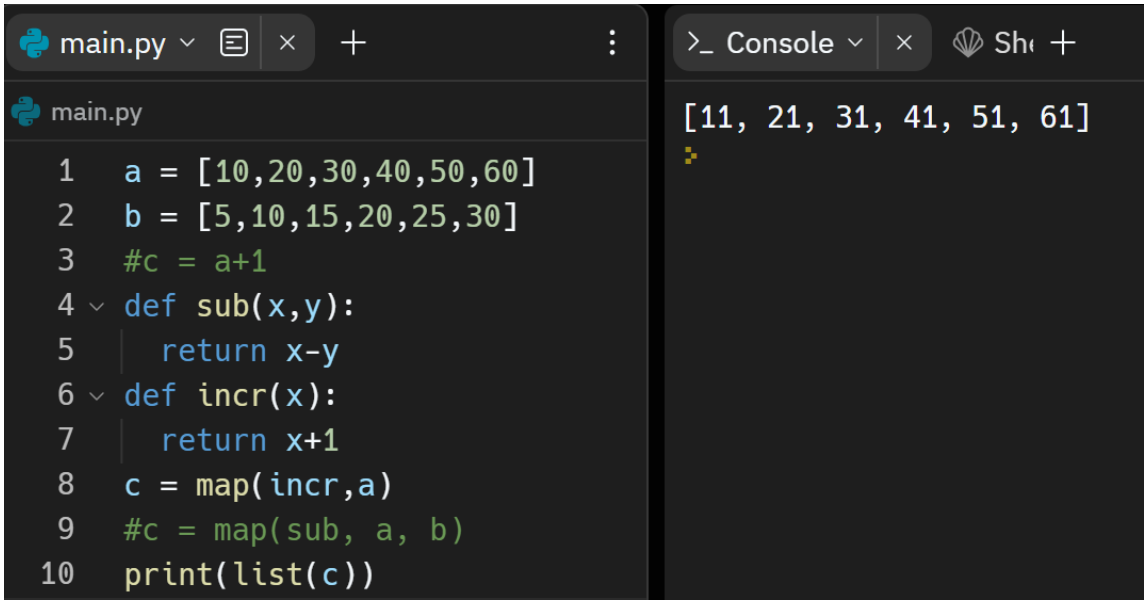


- Using the `zip` function we can couple two different lists.
- Instead of getting our output as list we can get it as a dictionary too by using `dict` instead of `list` before the zip function.

Map Function:

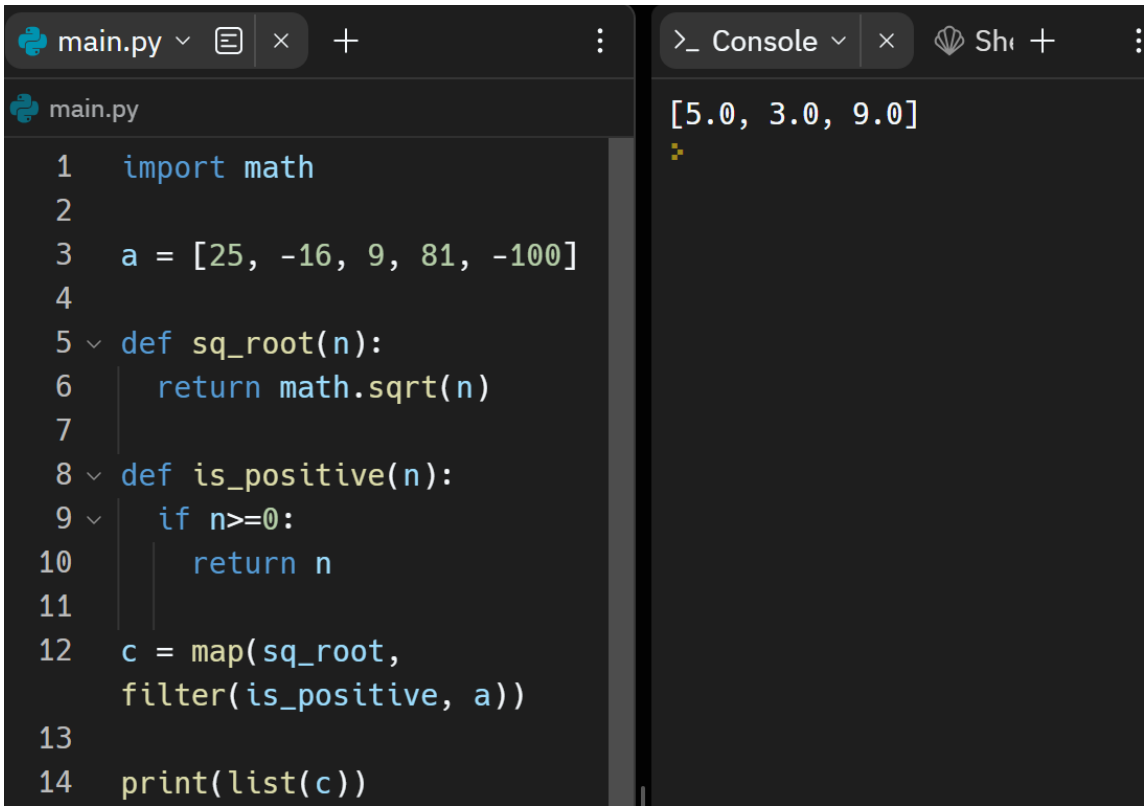


- Using the `map` function we can map a function with its iterators.
- In this case this `map` function will iterate over these two lists and at the same time it will exeecute this fucntions for every single entry inside these two lists.



- We incremented every element in list a by 1 using map function.

Filter Function:



- We created a filter here to filter out negative numbers from the list a using `filter` function.