Starter shitzz

comments

Comments are some lines in a code used to increase the readibility of the code and also used when testing a code.

To add a comment of a single line to a program, we will use # [**single line comment**]
for example: #this is a comment

We can also write comment using """ to make it a multiline comment which are called [**Block Comment**].

For example:
"""
Comments bigger woooo?
woooooo
wooooo
"""

dir()

To get all possible function anything we can write
print(dir(set/tuple/list))

Help()

To understand what the functions do, we can write
help(set/tuple/etc)
*one might also give a library function inside of help to get the functions it can do

VARIABLE

*A container for a value (string, integer , float , boolean)
* It behaves as if it was the value it contains
*for strings the words need to be double quoted (" ")
*for integer, it can directly be assigned. [without quotes]
*for floats, it is also the same as integer

EX:
assignment: number=23
assignment: distance=23.57
assignment: first_name="Bro"
printing: printf(first_name)

printing of an "FSTRING"

*printing of a string along with texts and shit
*here we will need to first take f" " within a print statement
*then inside **f"**text text text **{Variable_name}** "
EX: print(f"hello {first_name}")
output hello bro

EX: age=25
print(f"you are buying {age} eggs")

BOOLEAN VALUES:
*For Boolean, it is either YES or NO.
*Assignment: is_student= True

*USAGE:

```
is_student = True                    #Assignment
if is_student :                      #condition check for True
        print("You are a student")
else:                                #condition check for False
        print("You are not a student")
```

OUTPUT: You are a student

Typecasting

*the process of converting a variable from one data type to another.
* str(), int(), float(), bool()
*the type of a variable can be found out with type()
EX:
name= "Bro Code"

```
print(type(name))

OUTPUT: <class 'str'>
name="Bro code"
age= 23
gpa = 4.2
is_gae = True

age= float(23)          # age=23.0 float conversion
gpa=int(gpa)             # gpa= 4 [the decimal part will be truncated]
age=str(age)             # "23" is a string here
num=float("2.8") #string is converted to float with this
num2= int(4.2)    # it will be converted to only 4
name= bool(name) # True will be the value for any number or character

name=""
name= bool(name)    # False    will be the value for every empty character

STRINGS
*String addition works different from integer addition.
        age="25"
        age+="1"                        #one can not add integer to a string. Only int to int and str to str
        print(age)

OUTPUT: 251
```

USER INPUT INTAKE

* It is done with the input() function
*Function that prompts the user to enter data
*returns the entered data as a string
EX:

- ```
  name= input("What is your name?: ")      # A string will be taken and stored to 'name'
  age=input("What is your age?: ")                  # A string input will be taken and stored to 'age'
  print(f"hello {name} you are {age} years old")
  ```

- ```
  IF WE WANT TO ADD TO A STRING
  age= input("What is your age?: ")
  age= int(age) +1                                                    #type casting should be
  ```

done
print(f"Happy birthday you have become {age} years old")

- IF WE WANT A INPUT of our preference
  age= int(input("What is your age?: ")) #Type casting not required

ARITHMATIC OPERATORS

* They are as the same as C.
*A new operator for Exponents is available " ** ". EX: friends= friends ** 2
* Modulus is also present (%) to find remainder
*There are two forms of use.
i)Normal: arc = arc +1
ii)Augmented+= 1 [for exponents friends ** = 2]

Many operations are present:

- *round() can be used to round up the value of a variable to the nearest int
  *round() can also be used to truncate extra decimal points
  * The format of usage is round(variable_name , number_of_decimals_we_need)
  EX: pi= 3.14
  print(round(pi))
  pi=3.1416.....
  res=round(pi, 2) # res=3.14

- abs() is absolute value.

- pow() function is also available.
  EX: result=pow(2,3)

- max() function is used to find the maximum value in between multiple values.
  EX: max(x, y, z)

- min() function is used to find the minimum value in between multiple values.
  EX: min(x, y, z)

IMPORTING OF FUNCTIONS

* import function_name
* different function might hold different constants & functions.

EX:

- For constants
  ```
  import math
  print(math.pi)          #to get the value of pi
  print(math.e)           #to get the value of e
  ```

- For Functions
  ```
  result=math.sqrt(value/variable)        #square root
  x= 2.1
  result = math.ceil(x)        #result will be 3. It will always be rounded to next integer
  result= math.floor(x)    #result will be 2 . The value will stripped off its decimal part
  ```

IF statement

*Do some code only IF some condition are True
*else do something else
* Else if (" elif ") can be used to put in a condition with another condition.
*In here "or", "and", "not"can be used as it is.

- ```
  age=int(input("What is your age bruh?: "))
  if age>= 18:
          print("You are signed up")
  elif age<0
          print("Dont joke bro")
  else:
          print("Nigga you are underaged")
  ```

- Condition should be checked if it enters the respective part with proper condition
  ```
  if age>=18:
          print("You are signed up")
  elif age>=100
          print("You are too old")
  # The else if statement is useless as it will never go to the second condition
  ```

  The valid statement should be
  ```

```
if age>=100:
        print("You are too old")
elif age>=18:
        print("You are eligible")
else age<0:
        print("Nigga you are underaged")
```

- Empty input can also be done:-

```
name= input("Enter Your Name: ")
if name=="":                    # symbolizing nothing
        print("No name==gay")
else:
        print(f"Hello! {name}")
```

- Booleans can also be used

```
for_sale = True
if for_sale:
        print("It is for sale")
else :
        print("It is not for sale")
```

- "and" & "not"

```
temp = int(input("What is the temperature today?: "))
is_sunny = True

if temp > 35 and is_sunny :
    print("It is Hot outside!!")
    print("It is sunny bish!")
elif 0 < temp < 35 and is_sunny :
    print("It is warm outside")
    print("It is sunny too")
elif 0 > temp and is_sunny :
    print("It is cold af")
    print("It is sunny")
if temp > 35 and not is_sunny:
        print("It is Hot outside!!")
        print("It is Cloudy")
elif 0 < temp < 35 and not is_sunny:
        print("It is warm outside")
        print("It is Cloudy")
elif 0 > temp and not is_sunny:
        print("It is cold af")
```

```
print("It is Cloudy")
```

Conditional statement: [Ternary Statement]

* A one line shortcut for the if-else statement (ternary operator)
* Print or assign one of two based on a condition
* execute_X if (condition) else execute_Y

USAGE:

- Here is a simple use of Conditional statement in case of printing

```
x=int(input("Type of number"))
print("Positive" if x > 0 else "Negative" )
```

- Another Use in case of assignment.

```
x=int(input("Type in a number"))
p= "divisible by 5" if x % 5==0 else "not divisible by 5"
print(f"The number is {p}")
```

- Conditions put accordingly gives us what we want

```
x=int(input("What is your age?: "))
p= "adult" if x >= 18 else "Tui to gugu gaga"
print(p)
```

STRING USAGE

- Len ( )

  The Length of a string can be found using len (string_name) function.

Code:

```
str= input("Write something")
string_length= len(str)
print(string_length)
```

- String_name.find("Letter")

*The first occurance of a letter within a string can be found with the help of this function.

*If the letter is missing it will return " -1 ".

Code:

```
str=input("write something")
res=str.find(" ")          #The position of first appearance of the letter given inside will be returned
print(res)
```

- String_name.capitalize()

*When a whole string is given, The FIRST LETTER will be capitalized.

Code:

```
str=input("write something")
res=str.capitalize( )          #The first letter of the string will be capitalized
print(res)
```

- String_name.upper ( )

*The whole string will be Capitalized altogether.

Code:

```
str=input("write something")
res=str.upper( )          #The whole string will be capitalised
print(res)
```

- String_name.lower()

*The whole string will be lower cased.

- String_name.isdigit()

*The string will be checked if it is a digit

*if Yes, then True will be returned, else False will be returned

- String_name.isalpha()

*The whole string will be checked if all of them are Alphanumerics.(A-Z, a-z)

*if Yes, then True will be returned, else False will be returned.

*if any digit is found we, will be returned with False.

0) String_name.count("letter")

*The whole string found for how many times it contains a letter/number/digit/sign

Code:

```
number= +8801556482486
```

```
res=number.count("6")
print(res)
```
Output:
2

- String_name.replace("a","b")

*The targetted letter (a) in the whole string will be replaced by the latter (b)
*An individual letter can be cutoff totally by putting "" in place of "b".

Code:
```
number=input("Enter phone number")        # input= "+880-155-648-2486"
number= number.replace( "-" , " " )
print(number)
```
Output:
+880 155 648 2486 # The dashes will be replaced by spaces

Code:
```
number=input("Enter phone number")        # input= "+880-155-648-2486"
number= number.replace( "-" , "" )
print(number)
```
Output:
+8801556482486 # Nothing in between remains.

Finding overlapping String

* The given will be found for overlapping repeats through out the whole string

Code: For finding overlapping substring in a string

```
import re


len = int(input())

text=input()

if len==2:
    print(text)
    exit()


i=2
m=0
```

```
max_coun = 0

for _ in range(len- 2):
    test_count=0
    for _ in re.finditer(f"(?={text[m:i]})",text):
        test_count+=1
    if(test_count>max_coun):
        max_coun = test_count
        sub = text[m:i]

    m += 1
    i += 1

print(f"{sub}")
```

- print(help(str))
  This will give us necessary functions which we can use in different cases to increase functionality.

Addition of string is also possible:
str= str1+ str2 +str3
Another thing can be done

```
import string                #we need to import the string function
str = string.punctuation+string.ascii_letters+string.digits
print(str)
```

OUTPUT
!"$&'()*+,-./:;<=>@[\]^_`{|}~abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ012345678

Use of ord() & chr()

ord(): If we pass in a character within this function we will get the ascii number of the letter.
ord('a') will return 97
ord('A') will return 65

chr(): If we pass in the unicode code point or the ASCII number we will get the letter letter we want.
char(65) will return "A"
char(97) will return "a"

Indexing

* It is accessing elements of a sequence using [ ] (Indexing Operator)
* [start : end : stop]

EX:
credit_number= "1234-5678-9012-3456"

- ## Printing only one letter of the string

  print(credit_number[0]) ----------> 1
  print(credit_number [-1])----------> 6
  print(credit_number [-3])----------> 4 # roll back counting from the other side

- ## Printing from one point to other

  print(credit_number[: 4]) ----------> 1234 # same thing as [0 : 4]
  print(credit_number)[5 : 9]----------> -5678
  print(credit_number)[5 :] ----------> -5678-9012-3456 # same thing as [5 : 19]

- ## Printing every nth string

  print(credit_number[: : 3]) ----------> 146-136
  # it will step up from 0--->3---->6 and so on

- ## Printing last n digits

  *we will take n as negative of any number of digits that we want

```
str = "1234-5678-9876"
last4dig= str[-4::]
print (last4dig)
```

- We get 9876 as output from 4th index to the end from the opposite side.

- ## REVERSING A STRING

  *To reverse a string,
  *In the sep part we will give a minus 1 so that it reverses the whole thing
  str= str[::-1]

- ## Printing only a part from the first / part

- *X[:-30] returns all elements of x except the last 30 [last 30]
  *X[-30:] returns all elements of x except the first 30 [first 30]

Format Specifiers

*This helps to Format a value based on what flags are inserted.
* { value : flags }

i. .(number)f = round to that many decimal places (fixed point)

ii. :(number) = allocate that many spaces

iii. :03 = allocate and zero pad that many spaces

iv. :< = Left justify

v. :> = Right justify

vi. :^ = Center allign

vii. :+ = use a plus sign to indicate positive values

viii. := = place sign to leftmost position

ix. : = insert a space before postive numbers

x. : , = comma seperator

USAGE:

- precision to decimal places

```
price1 = 3.14159
price2 = 987.12

print(f"(the price is {price1:.3f }))       # it will be rounded to 3 decimal places (3.142)
print(f"{the price is {price2:.3f}})        # it will have 3 decimal places forcefully putting 0 (987.120)
```

- Zero padding

```
price1 = 3.14159
price2 = -987.12

print(f"the price is {price1:.03f }") # (0003.14159)
print(f"the price is {price2:.03f}")    # (0000987.12)
# padding takes the decimal parts too into consideration
```

- left justifcation takes it to left and right to right and center to center

- To indicate a positive number {price1:+} [+3.14159]
  {price2:+} [-987.12] #it was negative to begin with

- Aligning them evenly

  *" : " needs to be used so that all numbers take a space in the beginning.

  print(f"The price is ${price1: }")
  print(f"The price is ${price2: }")
  print(f"The price is ${price3: }")

  OUTPUT:
  The price is $ 3.14159
  The price is $-987.65
  The price is $ 12.34

- To place commas in desired places. ":, "

  price1 = 3000.14159
  price2 = -9870.12
  price3= 1200.34
  print(f"The price is ${price1:,}")
  print(f"The price is ${price2:,}")
  print(f"The price is ${price3:,}")

  The price is $3,000.14159
  The price is $-9,870.65
  The price is $1,200.34

- combinations can also be made

  *but a specific sequence needs to be folllowed

  Where `format_spec` consists of optional components in this order:

  1. **Fill and alignment** (e.g., `<`, `^`, `>`, `=`).
  2. **Sign** (e.g., `+`, `-`, or a space).
  3. **Width** (e.g., a number specifying minimum field width).
  4. **Comma as a thousand separator** (`,`).
  5. **Precision** (e.g., `.2f` for two decimal places).
  6. **Type** (e.g., `f` for floating-point, `d` for integer).

  CODE:
  price1 = 3000.14159
  price2 = -9870.12
  price3= 1200.34
  print(f"The price is ${price1:+,.2F}")
  print(f"The price is ${price2:+,.2F}")

```
print(f"The price is ${price3:+,.2F}")
```

OUTPUT:
The price is $+3,000.14
The price is $-9,870.65
The price is $+1,200.34

WHILE LOOPS

*executes some code while some conditions are true

USAGE:

- For empty input we will take repeat:

```
name=input("Enter your name: ")
        while name=="""":
                print("You did nto enter your name")
                name=input("Enter your name: ")

print("Hello {name}")          #identation should be maintained
```

- Again for a specific if we want to exit the loop:

```
food=input("Enter a food you like (q to quit): ")

while not food=="q" :
        print(f"You like {food}")
        food= input("Enter another food you like (q to quit)")
print("bye")
```

- Multiple conditions in a while loop

```
num= int(input("Enter a number between 1-10"))

while num<1 or num>10:
        print("your num {num} is invalid number")
        num=int(input("Enter a valid number"))
print("Your number is {num}")
```

for loops

* Executes a block of code a fixed numbers of times.

* We can iterate over a range, string, sequence etc.

- BASIC

```
for x in range (1,11):
        print(x)                # 1 2 3 4 5..... 9
```

- REVERSED

```
for x in reversed( range (1,11) ) :
        print(x)                 # 10 9 8 7 6 ..... 1
```

- Printing in STEPS

```
for x in range (1, 11, 2) :          #it prints number after skipping 2
        print(x)                # 1 3 5 7 9
```

- Printing a String sequentially

```
p= "buka lula"
for x in p :
        print x
```

import time

- time.sleep(seconds)

```
time.sleep(3)
print("TIME'S UP")
```

- A REAL TIME COUNTER

```
my_time= int(input("Enter the time in seconds: "))
for x in range (0, my_time):
        print(x)
        time.sleep(1)        #1 second time lag for every printing of x
print("Times up")
```

- REVERSE COUNTER

```
my_time= int(input("Enter the time in seconds: "))
for x in range (my_time, 0, -1):        [step actually gets added to the total time]
        print(x)
        time.sleep(1)        #1 second time lag for every printing of x
print("Times up")
```

- HEAVILY IMPROVED TIMER

```
my_time= int(input("Enter the time in seconds: "))
for x in range (my_time, 0, -1):
        seconds= int (x % 60)
        minutes= int (x / 60) % 60
```

```
            hours= int(x / 3600) % 3600
            print(f"{hours:02}:{minutes:02}:{seconds:02}")
            time.sleep(1)
    print("Times up")
```

NESTED LOOPS

*Basically a loop within another (outer, inner)
*Typical format: outer loop:
inner loop:

USAGE:

- Print style can be personalized

```
for x in range ( 1 , 10 ):
        print(x, end="")        # 123456789 [exact output]

for x in range (1,10):
        print(x, end=" ")          #1 2 3 4 5 6.....    [with a space]
```

- Running a loop a specific number of times:

```
for x in range (3):          #whole thing will run 3 times
        for y in range(1,10)
                print(y, end="") #123456789123456789.....789 [beside each other]

for x in range (3):
        for y in range(1,10)
                print(y, end="")
        print()    #This will print a line break after every iteration of the outer loop
```

- Printing a Rectangle [whole filled]

```
rows=int(input("Enter the number of rows: "))
columns=int(input("Enter the number of columns: "))
sym=input("Enter a symbol: ")

for x in range (rows):
        for y in range (column):
                print(sym, end="")      #should always be used in loops
        print()
```

Printing a number rectangle:

```
num =1


for x in range(3):
    for y in range(3):
        print(num, end=" ")
        num=num+1
    print()
```

- OUTPUT:
  1 2 3
  4 5 6
  7 8 9
-

Printing a Star rectangle:

```
lulu =int(input("Enter the number of rows you want: "))
    mulu = int(input("Enter the number of column you want: "))


    for x in range(lulu):
        for y in range(mulu):
            print("*", end=" ")
        print()
```

OUTPUT:

Enter the number of rows you want: 5

Enter the number of column you want: 6

* * * * * *

* * * * * *

* * * * * *

FOR Star pyramid:

```
lulu =int(input("Enter the number of rows you want: "))
    mulu =1


    for x in range(lulu):
        for y in range(mulu):
            print("*", end=" ")
```

```
            mulu+=1
            print()
```

OUTPUT:

Enter the number of rows you want: 5

*

* *

* * *

* * * *

* * * * *

-

Collection

* Single variable used to store multiple values.
    List = [ ] ordered and changeable. Duplicates OK
    set = { } unordered and immutable, but Add/Remove OK, NO duplicates
    Tuple= ( ) ordered and unchangeable. Duplicates Ok. FASTER

LIST

- Printing a **whole** list:

```
fruits = ["apples", "orange" , "banana", "coconuts"]
print(fruits)

# [ 'apples', 'orange' , 'banana', 'coconuts' ]    [exact]
```

- Printing a **single** element:

```
print(fruits[0])
#apple
```

- Printing **Multiple** elements:

```
print(fruit[0:3])
# [ 'apple', 'orange', 'banana' ]
```

- Printing every **nth** element:

```
print(fruit[::2])
#['apple' , 'banana' ]        # 0--->2
```

- Printing **Reversed**

```
print(fruits[::-1])
# ['coconuts', 'banana','orange','apple']
```

- Printing **sequentially** with line breaks

```
for fruit in fruits:
        print(fruit)
```

- Elements of a list can be found with the **help of len()** function

```
fruits=["guava","mango","apple","banana"]
len(fruits)            # this will be 4
```

- To **identify** if an element is **present** in a list

```
# answer will be found in boolean

print("apple" in fruits) #as it is present
Output:
True

print("pineapple" in fruits) #as it is not present
Output:
False
```

- We can **reassign** the elements in a list:

```
fruit[0]= "pineapple"
for fruit in fruits:
        print(fruit)

#       pineapple
        orange
        banana
        coconut
```

- We can **append** elements to a list:

```
fruit.append("jambura")

print(fruits)

#['apple', 'orange', 'banana', 'coconut', 'pineapple', 'jambura']
```

- To **remove** an element

```
fruit.remove("apple")

print(fruit)

#['orange','banana','coconut','pineapple']
```

- **To assign to a specific part of the list we can also use this function**

```
fruit.insert(0, "pineapple")
```

- To sort in **alphabetical** order:

  # Sort the characters in the string
  sorted_fruits = ''.join(sorted(fruits))
  print(sorted_fruits)

- To reverse the **sequence** of elements in a list:

  fruit.reverse()

- To **clear** a list

  fruits.clear()

- To **convert** a list to string:

  test=['x','y','z']
  test= "".join(test)

- To **return** the **index** of an element

  print(fruit.index("apple"))
  # we will get 0 as output as the index number is 0.

- To find how many **times** an **element** are there in a **list**

  print(fruit.count(banana))
  #1 as one tme its present

  print(fruit.count(pineapple))
  #0 as it is not present

- To include **characters** of a string to a list

```
import string
  char="meowmeowbitch"
  char=list(char)
  print(char)

  OUTPUT:
  ['m', 'e', 'o', 'w', 'm', 'e', 'o', 'w', 'b', 'i', 't', 'c', 'h']
```

To copy **a list** to another list

```
import string              #we need to import the string function
str = string.punctuation+string.ascii_letters+string.digits
```

```
str=list(str)
key=chars.copy()
```

To sort **a list** of integers

For Ascending sort:

```
numbers = [5, 2, 9, 1, 5, 6]
numbers .sort()
print(numbers)          # Output: [1, 2, 5, 5, 6, 9]
```

For saving sorted list in another list:

```
numbers = [5, 2, 9, 1, 5, 6]
sorted_numbers = sorted(numbers)       #returns a new list
print(sorted_numbers)    # Output: [1, 2, 5, 5, 6, 9]
```

For descending sort :

```
numbers.sort( reverse = True )
```

SET

*values are Unordered and Immutable
*No duplicates can be included [if duplicate is given only the first will be taken]
*We cannot use indexing here

Adding an element
fruits.add("pineapple")

Removing an element
fruits.remove("apple")

Removal of any element at first but the process is random
fruits.pop()

Clearance of the whole set
fruits.clear()

TUPLE

*Ordered & Unchangeable
*It is faster than a list
*It is present in a ()

*index can be found here
*count of an element can be found
*We can iterate over them

A simple shoping cart program with loops and list

```
foods = []
prices = []
total = 0

while True :
        food= input("What type of food do you want? (q to quit)")
        if food.lower() == "q":
                break
        else :
```

```
            price= float(input (f"What is the price of {food}"))
            foods.append(food)
            prices.append(price)

print("----------YOUR CART----------")
for food in foods :
        print(food, end=" ")
for price in prices :
        total+=price
print()
print(f"Your total is {total}")
```

2D lists

*To make a 2D list we will need to add lists to another list
*We can make a list with sets
*We can also make a tuple with sets
*We can make tuple made up of tuples

EXAMPLE:

- Assigning values of a 2D list

```
fruits =    ["apple","orange","banana","coconut"]
vegs=       ["celery","carrots","potatoes"]                    # for the visualization of an array
meats =     ["chicken","fish","turkey"]

groceries = [ fruits, vegetables, meats]

# the whole thing will be laid flat one after the another
```

- Assigning values of a 2D list [altogether]

```
groceries=[["apple","orange","banana","coconut"], ["celery","carrots","potatoes"],
["chicken","fish","turkey"] ]

#for better visibilty we can write

groceries= [ ["apple","orange","banana","coconut"],
             ["celery","carrots","potatoes"],
             ["chicken","fish","turkey"] ]
```

- print(groceries [0])

  # The whole fruits list will be printed

- If we need a specific element from the 2D list

  ```
  print(groceries[0][0])                    # apple
  print(groceries[0][2])                    #banana
  print(groceries[1][2])                    #potatoes
  ```

- Iteration through the elments of the list:

  ```
  for collection in groceries :          # to get as a list
          print(collection)

  # we will get
  ["apple","orange","banana","coconut"],
  ["celery","carrots","potatoes"],
  ["chicken","fish","turkey"]


  for collection in groceries:
          for food in collection:
                  print(food, end=" ")
  # In this way we will get output like this
  apple
  orange
  banana
  coconut
  celery
  carrots.....
  ```

- **A SIMPLE PROGRAM PRINTING NUM PAD**

  ```
  num_pad=((1,2,3),
  ```

```
                    (4,5,6),
                    (7,8,9),
                    ("#",0,"*")
                    )
for num in num_pad :
        for mum in num :
                print(mum, end=" ") # printing of each element
        print()                                                 # line break after each
iteration
```

- **A QUIZ TEST PROGRAM**

```
questions=( "What is the mitochondria name?",
                "How many asmani kitabs are there?",
                "How many flags are there at IUT?",
                "Which year did Bangladesh actually gain independence?",
                )

options=(("A. Power House of the cell", "B. Gyatt of the cell", "C. Traffic police", "D. Rizzler
organelle"),
            ("A.4","B.5", "C.6", "D.3"),
            ("A.none","B.61", "C.62", "D.65"),
            ("A.1971","B.1947", "C.2023", "D.2024"))
answers=("A","A","C","D")
qnum=0
guesses=[]
score=0

for question in questions :
    print("---------------------------")
    print(question)
    print(options[qnum])
    guess=input("Your guess: ")
    guesses.append(guess)
    if guesses[qnum]==answers[qnum] :
        print("Correct!!")
        score+=1
    else :
        print("Incorrect!!")
    qnum=qnum+1
print("The guesses are: ")
for guess in guesses :
```

```
        print(guess, end=" ")
print()
print("The Answers are:")

for answer in answers :
        print(answer, end=" ")
print()
perc= float((score/len(questions))*100)

print(f"You were {perc}% correct in the test")
```

Dictionaries

* A **collection** of {**key:value**} **pairs**
*Ordered and Changable
*No duplicates allowed

Examples:

- Assigning values to a dictionary:

```
capital = {"USA":"Washington",
                "India":"delhi",
                "Russia":"Moscow",}
```

- **To get the capital of USA [Value]**

```
print(capital.get("USA"))
#we will get Washington as an output
# Here USA is a key and Washington is the value to that key
```

- **If a non-existing key is used to get a value**

```
print(capital.get("Japan"))
#Output will be "none"
```

- **if-else statement usage [boolean return for key values]**

```
if capital.get("USA"):
        print("Capital exists")
else :                                                  # If exist we will get "True" else
```

"False"
print("Capital doesnt exist")

- **Capitals.update({ Key : Value }) [Updating value]**

*Used to change an existing value
*Also to update a value
EX:
capitals.update({ "Germany" : "Berlin" }) #for an update
capitals.update({ "USA" : "Detroit" }) #for a change

- **Captals.pop(key) [removal of a key]**

*To remove a Key value pair

EX:
capitals.pop("Russia")

- **Capitals.popitem() [Removal of latest item]**

*To remove the latest updated {key:value} pair

- **Capitals.clear() [For Clearance of the whole list]**

*To clear the whole dictionary

- **Capital.keys() [To get the keys of key value pair]**

*This function wil return all the keys of a dictionary
*They are also iterable.

For key in Capitals.key():
print(key)

- **Capitals.value() [To get the values of key value pair]**

*We will get all the values
print(Capital.value())

Capitals.items() [to get both the key and value]

*To get all the items both keys and values in a 2D list of Tuples.
Capitals.items()
print(items)

*It is also iterable.
for key, value in Capitals.item():
print(f"{key}:{value}")
#output will be
USA:Washington
India:Delhi
China:Beijing.....

To add keys & values with user input

Taking key and value individually:

```
dict={}

for num in range(4):
        key = input("Enter your name : ")
        value = int(input("Enter your score: "))
        dict[key] = value
print(dict)
```

Taking key and value together:

```
n = int(input("Enter number of elements: "))
my_dict = {}

for _ in range(n):
        key, value = input("Enter key and value separated by space: ").split()
        my_dict[key] = int(value)    # Convert value to integer

print("Dictionary:", my_dict)
```

***But get only the values from a dictionary or only values from a dictionary**

```
for value, key in kwargs.item():#here both the key and value needed
                    print(key)                    #this will give us just keys


for value,key in kwargs.items():
        print(value)                                                    #for only values
```

import random

- To get a random number: **[random.randint(from,to)]**
  *We can get a random number in a specific range from that number to the end we can use this

functions.
*You can also place inside this function as long as they contain numbers.

```
low=1
high=100
number=random.randint( low, high)      #same thing as random(1, 100)
print(number)
```

- To get a random floating point number [**random.random()**]

```
number=random.random()
print(number)
#number = 0.2323342524345 like this
```

- To get a random choice [**random.choice(tuple)**]
*to take a random choice from a list.

```
options =("rock","paper" ,"scissors" )
option=random.choice(option)
#we will get any of the things here
```

- To get a random choice within strings and numbers [**random.shuffle()**]

```
cards=["1","2","K","Q","J","8","9"]

cards= random.shuffle(cards)
print(cards)
```

- **Rock Paper Scissors program**

```
import random
m=("Rock","Paper","Scissors")
x=5
count=0
win=0
cwin=0
while count<5 :
        a = random.choice(m)
        count+=1
        c =input("Rock paper scissors!! Go!!!: ")
        b=c.capitalize()
        if    b=="Rock" and a=="Scissors" :
                print(f"Computer goes {a}")
                print("You win!!")
                win+=1
```

```
        elif b=="Paper" and a=="Rock" :
            print(f"Computer goes {a}")
            print("You win!!")
            win+=1
        elif b == "Scissors" and a == "Paper":
            print(f"Computer goes {a}")
            print("You win!!")
            win+=1
        elif b == "Paper" and a == "Scissors":
            print(f"Computer goes {a}")
            print("You lose!!")
            win+=1
        elif b == "Scissors" and a == "Rock":
            print(f"Computer goes {a}")
            print("You lose!!")
            cwin+=1
        elif b == "Rock" and a == "Paper":
            print(f"Computer goes {a}")
            print("You lose!!")
            cwin+=1
        elif b==a :
            print("Its a draw")
        else:
            print(f"{a}")
print(f"The score is {win}-{cwin}")
if win<cwin :
    print("Computer wins")
else :
    print("You win")
```

- **Dice program**

```
import random
dice_art={
    1:("┌─────────┐",
       "│         │",
       "│    ●    │",
       "│         │",
       "└─────────┘"),
    2:("┌─────────┐",
       "│ ●       │",
       "│         │",
       "│       ● │",
       "└─────────┘"),
    3:("┌─────────┐",
       "│ ●       │",
```

```
            "|       ●     |",
            "|           ● |",
            "└─────────────┘"),
        4:("┌─────────────┐",
            "| ●       ● |",
            "|           |",
            "| ●       ● |",
            "└─────────────┘"),
        5:("┌─────────────┐",
            "| ●       ● |",
            "|       ●     |",
            "| ●       ● |",
            "└─────────────┘"),
        6:("┌─────────────┐",
            "| ●       ● |",
            "| ●       ● |",
            "| ●       ● |",
            "└─────────────┘")
}
dice=[]
total=0
num=int(input("What are the number of dices you want?: "))
for die in range(num):
    dice.append(random.randint(1,6))
for die in range(num):
    for line in dice_art.get(dice[die]):
        print(line)
for die in dice:
    print(die)
    total+=die
print(f"The total is {total}")
```

Function

*A block of reusable code
*place **(parameter(if any))** after the function name to invoke it.
*We make a function with **def function_name(parameter) :**
*return statement is used to end a function and send a result to the caller of the function.

USAGE:

- Calling a function using parameters

```
def happy_birthday(name, age):

# Making    of the function & putting in parameter

        print(f"Happy Birthday to {name}!")
        print(f"You are {age} old!!")
        print(f"Happy birhtday to {name}!")
        print()
happy_birthday("bro",20)

#Invoking of the function with parameter

happy_birthday("steve",40)

#positioning of the parameters also matter

happy_birthday("miiii",50)

# as a string parameter is used, we put in string too
```

- Using of Return function

  *the returned variable should also be a parameter within the parenthesis

- **A Program returning the addition of two numbers**

```
def add(x,y):
    return(x+y)
print(add(2,3))
```

- **A program returning capitalized names**

```
def capname(name,nami):
        return name.capitalize()+" "+nami.capitalize()
# YES THIS SORT OF RETURN TYPE IS POSSIBLE
print(capname("nowfel","hossain"))
```

DEFAULT ARGUMENTS (within function)

*A default value for certain parameters
*default is used when that argument is omitted
*making the fucnton more flexible, reduces numbers of arguments
*TYPES OF ARGUMENTS (4):
    i)positional
    ii)default
    iii)keyword
    iv)arbitrary

To understand the concept we will use a simple program finding the price of an item:

```
def price(rp,disc,tax):
```

```
        return rp*(1-(disc/100))*(1+(tax/100))

print(f"{price(100,50,10):.2F}")
```

Here there are **3 arguments all** which **can be changed**.

But, let us say normally there is **no tax** or **no discount** for objects and **only for special cases there are discount and tax.**

1

```
def price(rp,disc=0,tax=0):
                        # Here the discount and tax has been given to zero
        return rp*(1-(disc/100))*(1+(tax/100))

print(f"{price(100):.2F}")
                        # Here the price is to given only to ger the default price
                        # The discount & tax has no need to be mentioned
```

2

*Let us say, for a specific case, the discount is 5% and tax is 3%. Then in this case, wven if there is a default case, the case willl be overwritten with newer values.

```
def price(rp,disc=0,tax=0):

        return rp*(1-(disc/100))*(1+(tax/100))

print(f"{price(100,5,3):.2F}")
```

3

*The default arguments should be mentioned after the non-default arguments
*else error will pop up

```
import time
```

```
def timer(end,start=0):          # The mentioning needs to be done to after

        for num in range(start,end+1):
                print(num)
                time.sleep(1)
        print("end of timer!!")

timer(100)
```

Keyword & Positional

arguments

Positional Argument

These are positional arguments, where the position matters:

```
def open(greetings,title,first,last):
        print(f"{greetings} {title} {first} {last}")
open("hi","My bro","Dudu","Miya")
```

Keyword Argument

*An argument which is **preceded** **by an identifer**.
*helps with readability
*orders of argument do not matter

But, if we want the order of the arguments to not matter then keyword arguments are used:

```
def open(greetings,title,first,last):
        print(f"{greetings} {title} {first} {last}")
open("hi",title="My bro",last="Miya",first="Dudu")
```

 # Here title, last, first are identifiers.

***What we can derive from this code is that:**
1) we have to mention the positional arguments at first (if it needs to be mentioned) then we can use the keyword arguments.
2) If all keyword arguments are used, the position does not matter.

- Within print function, we can use keyword arguments:

```
print("1","2","3","4","5","6","7","8","9","10", sep="*")

#1*2*3*4*5*6*7*8*9*10
```

Arbirtrary arguments

----> *arg allows you to pass multiple non-key arguments
*for args it is always tuples

-----> **kwargs allows you to pass multiple key-arguments

* for kwargs its always dictionaries {key}:{value} pairs

-----> * is unpacking operator

Working Mechanism:

* Here, the arguments that have been entered within the function for *args are all included in a tuple.
* Again, the arguments that have been entered within the function for *kwargs are all included in a dictionary.
* From the tuple, the values are taken and used for later purposes.

IDENTIFYING CODE:

```
def add(*args):
        print(type(args))
add(1,2,3,4,5,6,7)

def add(**kwargs):
        print(type(kwargs))
add (siu="lu",
        liu="sulu",
        sodu="chau",
        kodu="nouu",
        lau="siuu",
        mewed="Lau",
        weed="mau",
        street="siuuuu")
```

OUTPUT:

```
<class 'tuple'> #args
<class 'dict'>      #kwargs
```

USAGE [*args] :

- **A simple code adding multiple numbers**

```
def add(*nums): # Parameter names might differ
        total=0
        for num in nums:      # we iterate through the arguments in nums
                total+=num
        return total
```

```
sum=add(1,2,3)
print(sum)
```

- **A simple code printing names sequentially**

```
def name(*names):
        for name in names:
                print(name, end=" ")
name("Subaru","Shorkar","Jalil")
```

USAGE [**kwargs] :

- **A simple program printing all the items in the dictionary kwargs by iteration:**

```
def add(**kwargs):
        for key,value in kwargs.items():
                print(f"{key}:{value}")
add (siu="lu",
        liu="sulu",
        sodu="chau",
        kodu="nouu",
        lau="siuu",
        mewed="Lau",
        weed="mau",
        street="siuuuu")
```

MIXED USAGE OF KWARGS AND ARGS:

- Sequence Handling

*Here, the sequencing of the inputs need to be handled properly.
*As args and kwargs are first and second respectively, we have to place the arguments first in the parameter part and then the keyword arguments later.
*If kwargs were mentioned first, we had to put the keyword parameters at first and arguments later.

```
def shiplabel(*args,**kwargs):
     for arg in args:
          pass
shiplabel("Dr.", "Spongebob", "Squarepants", "III",
          street="fake street 123",
          apt="100",
          city="Detroit",
```

```
                state="MI",
                zip="54321"
                )
```

- A single value or key may also be obtained

```
def shiplabel(*args,**kwargs):
    for arg in args:
        print(arg, end=" ")
    print()
    for value,key in kwargs.items():
        print(f"{value}", end=" ")
    print()
    print( f"{kwargs.get('street') }")

shiplabel("Dr.", "Spongebob", "Squarepants", "III",
                street="fake street 123",
                apt="100",
                city="Detroit",
                state="MI",
                zip="54321"
                )
```

- A condition might be checked within the function

```
def shiplabel(*args,**kwargs):
    for arg in args:
        print(arg, end=" ")
    print()
    if "apt" in kwargs:
        print( f"{kwargs.get('street')} {kwargs.get('apt')}")
        print(f"{kwargs.get('city')} {kwargs.get('zip')}")
    else:
        print(f"{kwargs.get('street')}")
        print(f"{kwargs.get('city')} {kwargs.get('zip')}")
shiplabel("Dr.", "Spongebob", "Squarepants", "III",
                street="fake street 123",
                city="Detroit",
                state="MI",
                zip="54321"
                )
```

Iterables

* An object / collection that can return it element one at a time.
*allowing to be iterated over in a loop

LISTS ARE ITERABLE

- Sequential iteration through for loop:

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

for number in numbers:
        print(number)
```

- Reversed iteration through for loop:

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

for number in reversed(numbers):
        print(number)
```

SETS [non sequentially iterability]

- So they are not reversibily iterable.
- Normal iterations are fine

STRINGS [sequential iterability]

- Code:

```
name="Bro code"

for character in name:
        print(character, end=" ")
```

DICTIONARY [Iterability]

- **For Just Keys:**

```
my_dictionary={
        "A":67,
        "B":68,
        "C":69
}
for key in my_dictionary.keys():
        print(f"The ascii value of {key}")
```

- **For Just values:**

```
my_dictionary={
```

```
        "A":67,
        "B":68,
        "C":69
    }
    for value in my_dictionary.values():
        print(f"The ascii value of {value}")
```

- **For both keys & values:**

```
my_dictionary={
    "A":67,
    "B":68,
    "C":69
}
for key,value in my_dictionary.items():
    print(f"The ascii value of {key} is {value} ")
```

Membership operators

*Used to test whether a value or variable is found in a sequence
*may include string, list, tuple, set or dictionary
*They are i)"in" & ii)"not in"

- **For a single character checkup**

- word="Sagol"
  a=input(guess the word)

  if a not in word:
      print(f"It's wrong you {word}")
  else:
      print("Yes it is there")

- **For a word in a list**

- words=["skibidi","gyatt","rizz","sigma","gigachad"]
  guess=input("Guess that Gen-z term")

  if guess in words:
  print(f"Yes sir you are a {words[0]} {words[4]}")
  else:
  print(f"You have no {words[2]} sir")

For Dictionaries

- grades= {"Fanum":"A",
  "Kai Cenat":"B",
  "Speed":"F 🗣️ 🐶🗣️🐶🗣️🐶",
  "KSI":"From the screen to the ring to the pen to the king 🗣️📣📣📣🔥🔥🔥"
  }
  student=input("Enter the name of the student: ")

  if student in grades:
  print(f"The Grade of {student} is {grades.get(student)}")
  #we can also write grades[student]
  else:
  print(f"The student {student} does not exist")

LIST COMPREHENSION

* A concise way to create lists in Python.
*Compact and easier to read traditional loops

FORMAT : [**expression** **for** value **in** iterable **if** condition]

WITH NUMBERS

CODE: [Finds the double of the ranged numbers]
double[x*2 for x in range (1,11)]
print(double)

OUTPUt:
{2,4,6,8,.....,20}

WITH STRINGS

CODE [Capitalizes the whole thing]

```
fruits=["apple","orange","banana","coconut"]
fruits=[fruits.upper() for fruit in fruits]
print(fruits)
```

We can also write:

```
fruits=[fruits.upper() for fruit in ["apple". ""orange, "banana"," coconut"]]
print(fruit)
```

OUTPUT::
['APPLE','ORANGE','BANANA','COCONUT']

CODE [To get the First letter of all the elements in the list]

```
fruits=["apple","orange","banana","coconut"]
fruits=[fruit[0] for fruit in fruits]
print(fruit)
```

OUTPUT:
['a','o','b','c']

USAGE OF IF CONDITION

CODE [Changes the sign of the number]

```
numbers = [1, -2, -3, 4, -5, 6]
pos_num = [number for number in numbers if number>=0]
neg_num= [number for number in numbers if number<0]
ev_num=[number for number in numbers if number%2==0]
od_num=[number for number in numbers if number%2>0]
print(f"{pos_num} are the positive numbers")
print(f"{neg_num} are the negative numbers")
print(f"{od_num} are the odd numbers")
print(f"{ev_num} are the even numbers")
```

OUTPUT
[1, 4, 6] are the positive numbers
[-2, -3, -5] are the negative numbers
[1, -3, -5] are the odd numbers
[-2, 4, 6] are the even numbers

PRACTICAL USE

TO GET DICTIONARY VALUES

```
EX={"math":64,
    "english":80,
    "Agri":95,
    "Bent":78,
```

```
            "bom":89,
            "bangla":49,
            "Religion":30}

pasub={key: value for key,value in EX.items() if value>=50}
print(f"The subjects that he passed in are {pasub}")
fasub={key: value for key,value in EX.items() if value<50}
print(f"The subjects that he passed in are {fasub}")
```

TO GET ONLY SUBJECT

```
EX={"math":64,
            "english":80,
            "Agri":95,
            "Bent":78,
            "bom":89,
            "bangla":49,
            "Religion":30}

pasub=[key for key,value in EX.items() if value>=50]
print(f"The subjects that he passed in are {pasub}")
fasub=[key for key,value in EX.items() if value<50]
print(f"The subjects that he passed in are {fasub}")
```

TO ONLY GET VALUE

```
EX={"math":64,
            "english":80,
            "Agri":95,
            "Bent":78,
            "bom":89,
            "bangla":49,
            "Religion":30}

pasub=[value for key,value in EX.items() if value>=50]
print(f"The subjects that he passed in are {pasub}")
fasub=[value for key,value in EX.items() if value<50]
print(f"The subjects that he passed in are {fasub}")
```

Match Case Statement

* They are basically the switch case in C
*An alternative to using too many 'elif' statement
*Execute some code if a value matches with a 'case'
*Benefits? cleaner + readable

```
wili=input("What day is it today?: ")

match wili :
        case"sunday"|"saturday":        # here "|" acts as "OR"
                print("Its a holiday!!!")
        case "monday"|"tuesday"|"wednesday"|"thursday"|"friday":
                print("Wakey wakey wakey wakey!!! ITS TIME FOR SCHOOL BITCH!!!")
```

MODULE

* A file containing code you want to include in your program.
*use 'import' to include a module (built-in or your own)
*useful to break up a large program reusable seperate files.

USAGE:

*TO INCLUDE A MODULE We write **import.module_name**
***To give a module a nickname, we write: **import math as m**

- FOR BORROWING A **VALUE**:

  If we need to borrow a value from a module:

  we write for example:
  import math
  math.pi #for the value of pi

  *We can also write
  import math as m
  m.pi #for the same value of pi

  FOR BORROWING AN **INDIVIDUAL VALUE**:

  from math import pi
  print(pi)

  ISSUE 1:

```
import math
a,b,c,d=1,2,3,4
print(math.e)
```

Here e is value in the math module. But if:

```
import math
a,b,c,d,e=1,2,3,4, 5
print(e**e)    #only takes the assigned value not the mathematical one
print(math.e**e)    #here there are 2 e's. One is mathematical and one is assigned e
```

We can import a library with a specific name too:

```
import numpy as np
```

[we can use this library as np later whenever we want]

- **FOR MAKING YOUR OWN MODULE:**

  i) Open a new python file .
  ii) Put in self defined functions there.
  iii) Then call for that module as it is the file name.
  iv) Then revoke the functions you found there.

***** To import **everything** from a **script**, write :

from script2 import * **[Yes including the asterisk]**

Variable Scope

*variable scope is where a variable is visible and accessible.

*Order of variables are explained here below:

**scope resolution (LEGB)= Local - > Enclosed - > Global - > Built-in

*The variables within a function cannot be accessed by other functions.

*different functions can use the same variable. [local variables]

LOCAL VARIABLES: ***Here x in both functions are local variables**

```
def func1():
    x=1
    print(x)
def func2():
    x=2
    print(x)
func1()
func2()
```

GLOBAL VARIABLES: ***x as a global variable**

```
def func1():
    print(x)
def func2():
    print(x)
x=3
func1()
func2()
```

BUILT IN VARIABLES: ***e as a Built in variable**

```
from math import e
def func1():
    print(e)
def func2():
    print(e)
e=3
func1()
func2()
```

BUT if it is mentioned again within the main function..... then that values is considered as our original value. But still e here with import math is a GLOBAL VARIABLE.

Enclosed Variables:
When a function is defined inside another function:

The inner function has access to variables defined in its enclosing function's scope (but not in the global scope unless specified).

These variables are called enclosed variables in the context of the inner function.

-

```
def outer_function():
        enclosed_variable = "I'm enclosed"

        def inner_function():
            print(enclosed_variable)    # Accessing the enclosed variable

        inner_function()
```

```
outer_function()
```

if __name__==' __main__'

- * This script can be imported OR run standalone
  *functions and classes in this module can be reused without the main block of code executing.

  *Good practice as :
  i)Code is modular
  ii)helps readability
  iii)leaves no globar variables
  iv)avoids unintended execution

ex.library

- *import library for functionality
  *when running library directly, display a help page.

  STEP for experiment:
  1) First make 2 scripts of python file.
  2)Go to run to edit run configuration.
  3)The run config should be made to script1 for script1 and same for script2 for script2.
  4)The file type should be given python.

LEARN HOW [if __name__=="__main__"] WORKS

1) Now to get the list of functionality we will have to print the directory.
   print(dir())
2)Again, if we print(__name__), then we will get __main__

3)Next type [The run type is script1 meaning we are runnning script 1 directly]
```
from script2 import *
```

We bring in the functionalities of script2 within script1 and then printing __name__, would give us:

script2
__main__

4) Now, [The run type is script2 meaning we will be running script 2 directly]

```
from script1 import *

print(__name__)
```

We bring in the functionalities of script1 within script2 and then printing __name__, would give us:

script1
__main__

5) Here, " if __name__=="__main__": can be used to understand which script is running directly. This if statement is checked in the beginning.

```
def fav_food(food):
    print(f"Your favourite food is {food}")
def main():
    print("Hello to script1")
    fav_food("pizza")
    print("Goodbye")

if __name__ == '__main__':
    main()
```

This will give us an output of whatever we want as the script 1 is our main script right now. That is: [We run a function if it is ran directly only]

Hello to script1
Your favourite food is pizza
Goodbye

6) Now, even if we import script1 to script2, it wont run, as the if statement is checked and the script2 is not a main function.

7)But if the if statement is removed then, we can get an output in the script2 as the script2 is not checked for a main function.

script1

```
def fav_food(food):
    print(f"Your favourite food is {food}")

print("Hello to script1")
fav_food("pizza")
print("Goodbye")
```

script2

```
from script1 import*

#then the script2 will be run even if the run configuration is script2
```

Hello to script1
Your favourite food is pizza
Goodbye

PRACTICE [Bank application]
DEPOSITION & WITHDRAWAL

```
def balance(bal):
    print(bal)
def withdraw(amount):
    is_running= True
    while is_running:
        if amount < 0:
            print("Invalid withdrawal amount sir")
            print("Enter a valid amount")
        elif amount>0:
            return amount
            is_running= False

def deposit(amount):
    is_running= True
    while is_running:
        if amount < 0:
            print("Invalid deposit amount sir")
            print("Enter a valid amount")
        elif amount > 0:
            is_running= False
            return amount

is_running= True
```

```
bal=0

print("Type in from 1-4 to get your query")
print("-------------------------------------")
print("1) Show Balance")
print("2) Withdraw Money")
print("3) Deposit Money")
print("4) Quit")
print("-------------------------------------")

while is_running:
    mi = int(input("Which do you want done: "))

    if mi==1:
        print(f"Your banlance is {bal} tk")
    elif mi==2:
        x=int(input("Enter the amount of money for withdrawal: "))
        if x > bal:
            print("Insufficient balance.")
        else:
            bal-=withdraw(x)
    elif mi==3:
        y=int(input("Enter the amount of money for withdrawal: "))
        bal+=deposit(y)
    elif mi==4:
        is_running= False
    else:
        print("Invalid input sir.")

print("Have a good day!!")
```

PRACTICE(SPINNING LOTTERY program)

```
import random


def sp_row():
    list = ['🍁', '🍆', '🍋', '🍉', '🍎']
    spin = [random.choice(list) for _ in range(3)]
    return spin


def payout(spin, bet):
    if spin[0] == spin[1] == spin[2]:
        if spin[1] == '🍁':
```

```
                return bet * 5
            elif spin[1] == '🍌':
                return bet * 4
            elif spin[1] == '🍉':
                return bet * 3
            elif spin[1] == '🍎':
                return bet * 2
            elif spin[1] == '🍉':
                return bet * 10
        else:
            print("\nYOU LOST!!!")
            return (-bet)



def balance(bal,add):
            bal += add
            return bal


def main():
        print("************************************")
        print("SPIN THE WHEEEEELLLLL !!!!!!!")
        print("************************************")
        print("🍁 🍌 🍋 🍉 🍎")
        print("************************************")
        is_running= True
        bal=100
        while is_running:
            quer=input("Do you want to spin the wheel? (Y/N): ").capitalize()
            if quer.isdigit():
                print("INVALID INPUT")
                continue
            spin=[]
            if quer=="Y":
                bet=int(input("Place your bets: "))
                if bet>bal :
                    print("Insufficient Balance")
                    continue
                if bal<=0:
                    print("Insufficient Balance. You are broke sir.")
                    is_running= False
                spin = sp_row()
                for element in spin:
```

```
                print(f"{element}",end=" | ")

        elif quer=="N":
            is_running= False
            continue
        miu=balance(bal,payout(spin,bet))
        bal=miu
        if miu>0:
            print(f"Your current balance is {miu}")
        else:
            print("Insufficient balance")
            is_running= False

if __name__ == "__main__" :
    main()
```

Text Encryption program

```
import string
import random
#we need to import the string function
#for shuffle function too we will need random
str = string.punctuation+string.ascii_letters+string.digits
str1= input("Enter the OG text: ")
str=list(str)
key=str.copy()
encryptxt=""
random.shuffle(key)
for letters in str1:
    index=str.index(letters)
    encryptxt+=key[index]
print(f"Your OG text is {str1}")
print(f"Your encrypted text is {encryptxt}")
```

Object **Oriented** Program

What is an **Object?**

A "bundle" of related attributes (variables) and methonds (functions)

ex: phone, cup, phone.
You need a class to create many objects

What is a **Class?**

Blueprint which is used to design the structure and layout of an object.

*What does self refer to?
It refers to the state of the object

[1]How to make a Class

1) To make a class, we will need to name the class first.
class class_name:

2) Then, we need to initialize to self with dunder init **[CONSTRUCTOR]**
def __init__(self):

3) Then next step is to add multiple attributes within the def__init__(self,__,__,__) part
def__init__(self, name, year, model):

4) Then we will have to add t the attributes according to their names with the . (attribute access operator). we will need to connect the attributes to the self and then equalize them to the attributes we took.

def__init__(self,name,year,model):
self.name=name
self.year=year
self.model=model

FULL INSTANCE

```
class Car:
        def __init__ (self, model, year, for_sale):
            self.model=model
            self.year=year
            self.for_sale=for_sale
```

[2]How to make Class of your own

1) Take a class variable name and initialize it to the Main Class name

class_var=class_name("string",12,True)

2)The values within maybe string, integer, float, Boolean anything.

FULL INSTANCE

```
car1=Car("merc", 2023, True)
car2=Car("benz", 2002, False)
```

[3]How to use Class values

Take the class variable and use the attribute access operator to access the value of the attribute.

class_var . attribute_name

FULL INSTANCE

```
car1=Car("merc", 2023, True)
car2=Car("benz", 2002, False)


print(f"There are {car1.model} cars")
```

Output

There are merc cars

[4]How to make and use Class functions

Define a function using self as attribute and use function name like:

def func_name(self):

Within the function we can call for the attributes of the class as we wish:

print(f"Drive the {self.attribute_name}")

Then, we can call for the function within the main() function by calling on to the Class name and then putting the attribute as the Class name as you want within the empty parentheses

Class_name . Function_name (Class_var_name)

FULL INSTANCE

```
class Car:
    def __init__ (self, model, year, for_sale):
        self.model=model
        self.year=year
        self.for_sale=for_sale


    def drive(self):                # making of the function
        print(f"Drive the {self.model}")



car1=Car("merc", 2023, True)
car2=Car("benz", 2002, False)


Car.drive(car1)                     # calling of the function
Car.drive(car2)
```

OUTPUT:

```
Drive the merc
Drive the benz
```

Class Varaibles

What is **Class Variable?**

A Variable within a class which is shared within all the instances of a class

*It is defined outside the constructor

*It allows you to share data among all objects created from that class

What is **Instance Variable?**

The variables which are defined only within the definite function

*defined within the constructor

*It can be used within a specific function only

FULL INSTANCE

```
class Car:

        num_cars=4          # This is a Class variable

        def __init__ (self, model, year, for_sale):
            self.model=model
            self.year=year
            self.for_sale=for_sale
        def drive(self):

                num=5               # This is a instance variable
            print(f"Drive the {self.model}")

car1=Car("merc", 2023, True)
car2=Car("benz", 2002, False)
```

Call for class variable values

We can also call for the values using any of the of the Class_names of our own and also the main Class name but we should use the main class name to make the reader of the code understand that it is a Class variable.

```
class Car:

        num_cars=2

        def __init__ (self, model, year, for_sale):
            self.model=model
            self.year=year
```

```
        self.for_sale=for_sale
    def drive(self):
        num=5
        print(f"Drive the {self.model}")

car1=Car("merc", 2023, True)
car2=Car("benz", 2002, False)


print(f"{Car.num_cars}")        # call for class variable value
```

Use of the value

* We can change the value of the class variable as we wish so with every calls how we wish to by putting the work within the initialization phase of the making of the class

Full instance

```
class Car:

    num_cars=0

    def __init__ (self, model, year, for_sale):
        self.model=model
        self.year=year
        self.for_sale=for_sale
        Car.num_cars+=1
    def drive(self):
        num=5
        print(f"Drive the {self.model}")

car1=Car("merc", 2023, True)
car2=Car("benz", 2002, False)


print(f"{car1.num_cars}")
```

Inheritance

What does **inheritance?**

It allows a class to inherit attributes and methods from another class

It helps with code reusability and extensibility

class Child(Parent)

[1]How to inherit a class?

Step1:

We will need to make the Main Class first and then include the functions we need and the constants we need. We will need to initialize it all properly in their places.

For example:

```
class Animal:
    def __init__(self,name):
        self.name=name
        self.is_alive= True
    def eat(self):
        print(f"{self.name} is eating.")
    def sleep(self):
        print(f"{self.name} is sleeping.")
```

Step2: Make own class and pass in Main class name as Attribute.

For example:

```
class dog(Animal):      # include the main function name within the latter created function
    pass
class cat(Animal):      # we will be able to inherit the functonality of Animal class here too
```

```
        pass
    class mouse(Animal): # Reusing and extensive usage is possible
        pass
```

[2]How to use inherited classes?

Step1: Store it in a varible of such after enteriing in the attributes if there is any.

var_name= class_name("Attribute_name")

```
d=dog("Spike")
c=cat("Tom")
m=mouse("jerry")
```

Step 2: Then, we will need to call the variable, where we stored the class:

var_name.func_name()

```
 d.eat()                    # We will need to use the attr access operator
d.sleep()
```

Full Instance

```
class Car:
        wheel=4
        def __init__(self, model,year,mileage):
            self.model=model
            self.year=year
            self.mileage=mileage
        def start_car(self):
            print(f"You started the car {self.model}")
        def stop_car(self):
            print(f"You stopped the car {self.model}")
        def check_mileage(self):
            print(f"The mileage of the car {self.model} is {self.mileage}")
    class car1(Car):
        pass
    class car2(Car):
        pass
    class car3(Car):
        pass

    d=car1("BMW",2023,20000)
```

```
c=car2("Merc",2007,40000)
e=car3("Bentley", 2016, 50000)

d.start_car()
print(f"The mileage of the car is {d.mileage} with its {Car.wheel} wheels")
d.stop_car()
e.check_mileage()
c.check_mileage()
d.check_mileage()
```

Output:

```
You started the car BMW
The mileage of the car is 20000 with its 4 wheels
You stopped the car BMW
The mileage of the car Bentley is 50000
The mileage of the car Merc is 40000
The mileage of the car BMW is 20000
```

Multiple Inheritance

What is multiple inheritance?

To inherit from more than one parent class is called multiple inheritance.

format: C(A,B)

What is multilevel inheritnace?

inherit from a parent which inherits from another parent

format: C(B) <---- B(A) <--- A

Full instance [multiple inheritance]

Here, we can see in fish, multiple inheritance has taken place.

```
The hawk is a predator so it can only chase

The rabbit is a prey so it can run away
```

BUT the fish might have bigger fishes which chase and the smaller get preyed on so it gets both the characteristics of a prey and also a predator.

```
class prey:
    def run(self):
        print("The animal is running")
class predator:
    def chase(self):
        print("The animal is chasing the prey")
class hawk(predator):
    pass
class rabbit(prey):
    pass
class fish(predator, prey):#This is a children class which can inherit multiple classes
    pass

f=fish()
r=rabbit()
h=hawk()

h.chase()
r.run()

f.run()
f.chase()
```

A parent can also inherit from other parents too. **3rd degree inheritance**

```
class Animal:
    def __init__(self,name):
        self.name=name

    def sleep(self):
        print(f"The {self.name} is sleeping")
    def eat(self):
        print(f"The {self.name} is eating")
class prey(Animal):
    def run(self):
        print(f"The {self.name} is running")
class predator(Animal):
    def chase(self):
        print(f"The {self.name} is chasing the prey")
class hawk(predator):
    pass
```

```
class rabbit(prey):
        pass
class fish(predator, prey):
        pass

f=fish("fish")
r=rabbit("rabbit")
h=hawk("hawk")

f.sleep()
r.eat()
h.sleep()
f.chase()
```

Super()

Where is the **Super()** function used?

*Function used in a child class to call methods from a parent class (superclass)
*It allows you to extend the functionability of the inherited methods

*The child class is called the sub class
*The parent class is called the super class

* if there are 2 class version of the same name, we will use the child version instead of the parent version
*if both the version needs to be used together then we will need to write within the child version
super().class_name

How to use it?

1) If attributes required before are not required afterwards, the attributes still need to be written in the latter portion of the code when writing the child class.

2) Then again, after writing the child class we will need to initialize the super() function and give in the attributes which were required in the Parent class. [The attributes in the parent class needs to be initialized within the parent class]

super()__init__(attr_main_class1 , attr_main_class2)

3) Then, after initializing the newer attributes of the newer class afterwards, we will be ready to use

it.

FULL INSTANCE

```
class Shape:
        def __init__(self, color ,is_filled ):
            self.color= color
            self.is_filled= is_filled
    class Triangle(Shape):
        def __init__(self,color,is_filled, width, height):
            super().__init__(color,is_filled)
            self.width=width
            self.height=height
    class Square(Shape):
        def __init__(self,color,is_filled,width):
            super().__init__(color, is_filled)
            self.width=width
    class circle(Shape):
        def __init__(self, color, is_filled, radius):
            super().__init__(color, is_filled)
            self.radius=radius


    C=circle("Blue", True, 5)
    S=Square("red",True,25)
    T=Triangle("Purple", False , 10, 25)


    print(T.is_filled)
```

Adding function to the parent class

```
class Shape:
        def __init__(self, color ,is_filled ):
            self.color= color
            self.is_filled= is_filled
        def describe(self):          # The function in question
            return (f"It is {self.color} and is {"filled" if self.is_filled else "Not filled" }")
    class Triangle(Shape):
        def __init__(self,color,is_filled, width, height):
            super().__init__(color,is_filled)
            self.width=width
```

```
            self.height=height
class Square(Shape):
        def __init__(self,color,is_filled,width):
            super().__init__(color, is_filled)
            self.width=width
class circle(Shape):
        def __init__(self, color, is_filled, radius):
            super().__init__(color, is_filled)
            self.radius=radius


C=circle("Blue", True, 5 )
S=Square("red",False,25)
T=Triangle("Purple", True, 10, 25)


print(T.describe())
print(S.describe())
print(C.describe())
```

OUTPUT:

```
It is Purple and is filled
It is red and is Not filled
It is Blue and is filled
```

Polymorphism

What is **polymorphism?**

A program can take many forms, and this character is called polymorphism.

*poly = many

*morphe = form

Ways to achieve polymorphism:

i) **Inheritance**: An object could be treated of the same type as a parent class

ii) "**Duck Typing**" = Object must have necessary attributes

Achieving polymorphism

Step 1 : **Import library**:

As the main idea of polymorphism is making a program or a class being under a same Parent but also being different, we need to import a Library.

From abc library we will need to import: @abstractmethod and ABC like:

from abc import ABC, @abstractmethod

Step 2 : **"Abstract method"**:

We will need to call for the abstract method within the main class. Then under the abstract method, we need to define the function which will be common for all the children classes BUT it will act different under different classes. Instances:

```
class main_class:
        @abstractmethod        # Abstract method called for
        def common_func(self):          # method called for
                pass
```

Step 3 : **"Inheritance"**:

We will need to inherit the main class within the sub classes and define out common function under every sub class or child class. In this way, we can inherit the common function for all sub classes which can be different for each cases.

It will be shown in the full instance part.

Full instance

```
from abc import ABC,abstractmethod


    class Shape:
        @abstractmethod
```

```
    def area(self):
        pass

class Triangle(Shape):
    def __init__(self,base,height,name):
        self.height=height
        self.base=base
        self.name = name
    def area(self):
        return self.height*self.base*.5

class Square(Shape):
    def __init__ (self, side, name):
        self.side=side
        self.name=name
    def area(self):
        return self.side * self.side
class Circle(Shape):
    def __init__ (self, radius,name):
        self.radius=radius
        self.name=name
    def area(self):
        return 3.1412*(self.radius**2)
class Pizza(Circle):
    def __init__(self,radius,topping,name):
        self.topping=topping
        super().__init__(radius,name)


shapes=[Triangle(4,5, "triangle"),Circle(radius=4,name="circle"),Square(side=5,
name="square"),Pizza(25,"Cheese","pizza")]

for shape in shapes:
    print(f"the area of the {shape.name} is {shape.area()} sq.cm")
```

Static method

&

Instance method

What is a **Static method?**

-> A method that belong to a class rather than any object from that class (instance)

*Usually used for a general utility functions.

*without borrowing info from a class.

**For this you will need to only access the class.

EX:

```
@staticmethod
def km_to_miles(kilometers):
    return kilometers * 0.621371
```

What is a **Instance method?**

-> A method belonging to a class and more precisely from the objects from that class

* Best for operations on instances of the class (objects)

**For this you need to ACCESS the object first and then call for the method.

EX:

```
def get_info(self):
    return f"{self.name}={self.postion}"
```

Full Instance

```
class Employee:

    def __init__(self, name,position):
        self.name=name
        self.position=position

    def get_info(self):                    # defining a instance method
        return f"{self.name} = {self.position}"

    @staticmethod
    def is_valid_employee(position):              # defining a static method
        valid_position=["cook", "manager","janitor","cashier"]
```

```
        if position in valid_position:
            return ("It is a valid position")
        else:
            return ("It is not a valid position")

print(Employee.is_valid_employee("janitor")) #calling a static method


employee1= Employee("Mr.Crabs","manager")
employee2= Employee("Spongebob","cook")          #initializing a class objects
employee3=Employee("Eugene","janitor")
employee4=Employee("Squidwards","cashier")

print(employee1.get_info())
print(employee2.get_info())                      #calling a instance method
print(employee3.get_info())
print(employee4.get_info())
```

OUTPUT:

It is a valid position
Mr.Crabs = manager
Spongebob = cook
Eugene = janitor
Squidwards = cashier

Class Method

What is a Class method?

The method which allows operations related to the class itself
Take (cls) as the first parameter, which represents the class itself.

How to make a Class method?

1) At first under a CLASS, call for the class method by

"@classmethod"

2)The under the class method define your method.

3)The class name should be defined as:

def class_name():

4) An attribute should be passed in which is "cls": [definitely without the quotations]

def class_name(cls):

5) The constants within the main class can be called on to:

def class_name(cls):

return (cls.const_name)

FULL INSTANCE

```
class Student:
        count=0
        total_gpa=0

        def __init__(self,    name, gpa):
            self.name=name
            self.gpa=gpa
            Student.count+=1
            Student.total_gpa+=gpa

        def get_info(self):
            return f"{self.name} got a GPA of {self.gpa}"

        @classmethod
        def get_avg(cls):
            if cls.total_gpa==0:
                return 0
            else:
                return f"The average Gpa of the class is {cls.total_gpa/cls.count:.2F}"

    student1= Student("Arkam",2.3)
    student2=Student("Milu", 3.5)
    student3=Student("Sayuri", 5)

    print(student1.get_info())
    print(student2.get_info())
    print(student3.get_info())
```

```
print(f"The number of students in the class is {Student.count}")
print(Student.get_avg())
```

Decorator

What is a **Decorator?**

-> A function that extends the behavior of another function is called a decorator.

*w/o modifying the base function

*Pass the base function as an argument to the decorator

*We are adding something to a base function without changing it

How to create a Decorator?

1) At first make a base function. EX:

```
def base_func():
    print("Here is your base function")
```

2) Then define a function other than the base function before the main function. EX:

```
def Decorator():
    print("Here is your decorator.")
```

3) Then you will need to call the base function above the main function. EX:

```
def Decorator():
    print("Here is your decorator.")
@Decorator      #addition of the decorator
def base_func():
    print("Here is your base function")
```

4) One thing you will need to do is call for function within another function under the decorator. Then also remember to return it.

```
def Decorator(base_func):
    def func_within():
        print( "Function within decorator." )
        base_func()
    return func_within
@Decorator      #addition of the decorator
def base_func():
    print("Here is your base function.")
```

base_func()

OUTPUT:

Function within decorator.
Here is your base function.

FULL INSTANCE

```
def sprinkle(base_func):
    def func_within(*args, **kwargs):

                        #to show that it takes any number of arguments

        print("You used sprinkles ;)    ")
        base_func(*args, **kwargs)

                        #for the base function it is already required

    return func_within

def fudge(base_func):
    def func_within(*args, **kwargs):
        print("You also used fudge    :O ")
        base_func(*args, **kwargs)
    return func_within


@sprinkle    # addition of the decorator
@fudge
def base_func(flavor):
    print(f"Here is your {flavor} ice cream")


base_func("vanilla")
```

** A new discovered tric from here is if a function takes multiple arguments or one argument, for the other functions to run they will have to have the attribute *args, **kwargs if the main function has multiple arguments to it. Whenever the main function is called within the decorators they also have to have these arguments.

What are **Magic Methods?**

Magic methods are Dunder methods (double underscore) __init__, __str__, __eq__

*They are automatically called by many Python's built in operations.

*They allow developers to define or customize the behaviour of the objects when using built in operations.

1) __init__:

This method

*initializes an object when it is created.

For example, in a Book class, it can set attributes like title, author, and number of pages.

2) __str__:

This method

*defines the string representation of an object.

*Instead of returning a memory address, it can return a formatted string, such as "The Hobbit by J.R.R. Tolkien".

USAGE:

CODE WITHOUT USE OF STR

```
class Book:
    def __init__(self, title, writer_name, pages):
        self.title = title
        self.writer_name = writer_name
        self.pages = pages


book1 = Book("The hobbit", "Hobbit er baap", 32)
book2 = Book("Harry Potter and the Deathly Hollows", "J.K Rowlings", 45)
book3 = Book("Haat kaata robin", "Jafor Iqbal", 354)

print(book1)
```

OUTPUT: it returns the address of the class
<__main__.Book object at 0x000001234FF60FD0>

but this can be modified in such a way that the print statement acts differently that is modifying it.

CODE WITH USE OF STR

```
class Book:
    def __init__(self, title, writer_name, pages):
        self.title = title
        self.writer_name = writer_name
```

```
        self.pages = pages
    def __str__(self):
        return f"{self.title} by {self.writer_name}"

book1 = Book("The hobbit", "Hobbit er baap", 32)
book2 = Book("Harry Potter and the Deathly Hollows", "J.K Rowlings", 45)
book3 = Book("Haat kaata robin", "Jafor Iqbal", 354)

print(book1)
print(book2)
```

OUTPUT: **#we have modified in such a way that it prints out the attributes of the class how we want**

The hobbit by Hobbit er baap
Harry Potter and the Deathly Hollows by J.K Rowlings

3) __eq__:

This method

*checks for equality between two objects.
*It can be customized to compare specific attributes,
like title and author, while ignoring others like the number of pages.

USAGE:

Without use of the eq method the comparision will always be returned false however it is equal (most likely as the address is never equal)

With use of __eq__

CODE:

```
class Book:
    def __init__(self, title, writer_name, pages):
        self.title = title
        self.writer_name = writer_name
        self.pages = pages
  def __eq__(self, other):       # identation problems might occur so identate pls
    return self.title==other.title and self.writer_name==other.writer_name

book1 = Book("The hobbit", "Hobbit er baap", 32)
book2 = Book("Harry Potter and the Deathly Hollows", "J.K Rowlings", 45)
book3 = Book("Haat kaata robin", "Jafor Iqbal", 354)
book4 = Book("Haat kaata robin", "Jafor Iqbal", 354)
print(book3==book4)
```

This checks if the name of the author and the title of the books are equal which is the modification we made. By printing if book3==book4, we get either a true or false. As the name and the author are both the same it returns true.

4) **__lt__(less than)** & **__gt__(greater than)**:

These methods

*allow comparison operations (less than and greater than) between objects based on specific criteria,

such as the number of pages in the books.

USAGE:

Without use of lt or gt, we will always get an error that between the instances of 2 classes, there can not be a comparision.

WITH USE OF LT and GT

```
class Book:
    def __init__(self, title, writer_name, pages):
        self.title = title
        self.writer_name = writer_name
        self.pages = pages
    def __str__(self):
        return f"{self.title} by {self.writer_name}"
    def __eq__(self, other):
        return self.title==other.title and self.writer_name==other.writer_name
    def __gt__(self, other):
        return self.pages > other.pages
    def __lt__(self, other):
        return self.pages < other.pages

book1 = Book("The hobbit", "Hobbit er baap", 32)
book2 = Book("Harry Potter and the Deathly Hollows", "J.K Rowlings", 45)
book3 = Book("Haat kaata robin", "Jafor Iqbal", 354)
book4 = Book("Haat kaata robin", "Jafor Iqbal", 354)
print(book2 < book4)
print(book3 > book4)
print(book3<book4)
print(book1 > book2)
```

OUTPUT:

```
True          #as the value is smaller
False       # as the values are equal
False         # same reason as above
False         # as it is smaller
```

5) **__add__**: This method

*allows the addition of two objects,
such as summing the number of pages from two book objects.

USAGE:

```
def __add__(self, other):
      return self.pages + other.pages

print((book3+book1))
```

OUTPUT:

```
386       # summation of the page numbers of both the books
```

6) **__contains__**: This method

*checks if a keyword exists within an object's attributes,
such as searching for a word in the title or author.

Without the use of modification it will return that it is not iterable.

USAGE

```
def __contains__(self, keyword):
      return keyword in self.title or keyword in self.author

print("Potter" in book2)
```

OUTPUT:

True

7)**__getitem__**: This method

*allows access to an object's attributes using indexing,
*enabling retrieval of specific attributes

like title or author by key.

Without use it will return that object is not subscriptable.

With use of the __getitem__:

USAGE

```
def __getitem__(self,key):
    if key=='title':
        return f"The title of the book is {self.title}"
    elif key=='page':
        return f"The number of pages of this book is {self.pages}"
    elif key=='author':
        return f"The author of the book is {self.writer_name}"
    else :
        return f"The attribute {key} is missing"
```

OUTPUT:

The title of the book is The hobbit
The author of the book is Hobbit er baap
The number of pages of this book is 32
The title of the book is Harry Potter and the Deathly Hollows
The author of the book is J.K Rowlings
The number of pages of this book is 45
The title of the book is Haat kaata robin
The author of the book is Jafor Iqbal
The number of pages of this book is 354
The attribute summary is missing

Property

What is **Property?**

Decorator used to define a method as a property (it can be accessed like an attribute)

*Beneift: Adding additional logic when read, write or delete attributes
*Gives getter, setter and deleter method.

1) Getter Method [to read]:

You can define a method that

*retrieves the value of a private attribute.

*By using the @property decorator, you can access this method as if it were an attribute.

For example, if you have a private attribute _width, you can create a method width that returns its value formatted to one decimal place.

2) Setter Method[to write]:

You can define a method

*to set the value of a private attribute using the @<attribute_name>.setter decorator.

*This allows you to add validation logic, such as ensuring the new value is greater than zero before assigning it to the attribute.

3) Deleter Method[to delete]:

You can define a method

*to delete an attribute using the @<attribute_name>.deleter decorator.

*This can be useful for cleanup or to prevent access to certain attributes.

Encapsulation: By using private attributes (prefixed with an underscore),

*you can control access to these attributes

*enforce rules on how they are modified or accessed.

Example: In a Rectangle class, you might have private attributes _width and _height. You can create getter methods to return these values with a specific format, setter methods to validate the input, and deleter methods to remove the attributes if needed.

Use of property

1) To create protected members:

IF we do not want an attribute to be accessed directly from the internals and we want the values we can get after some property procedures then,

During the defining of an attribute we will need to put an underscore,

For example:

```
class class_name:
        def __init__(self, att1,att2):
                self._att1=att1         # we will put in a underscore before attribute
                self._att2=att2
```

Here if we call the att1 within the main function, then we will get the changed or modified version of it. But, (not needed usually) if we want the internal value of the attribute we will have to put the underscore while printing.
For Example:
class_name._att1

2) Calling for property:

Then for calling the attribute, we will need to call the "@property" before calling on to the property

USING OF GETTER METHODS

For example:

```
        @property
        def att1(self):
                pass
```

Full Instance of usage of Getter methods

CODE:

```
class rectangle:

    def __init__(self, height, width):
        self._width=width
        self._height=height
    @property
    def width(self):
        return f"{self._width:.1f} CM"
    @property
    def height(self):
        return f"{self._height:.1f} CM"

R = rectangle(5,4)
print(R.width)
print(R.height)
```

OUTPUT: #printing out modified values (not actually but ig so)

4.0 CM
5.0 CM

3) Calling of setter:

Calling for a setter method is done to change or assign new values to old attributes.

Use of Setter

```
class rectangle:

    def __init__(self, height, width):
        self._width=width
        self._height=height
    @property
    def width(self):
        return f"{self._width:.1f} CM"
    @property
    def height(self):
        return f"{self._height:.1f} CM"

    @width.setter
    def width(self, new_width):
        if new_width > 0:
            self._width = new_width
        else:
            return print("width must be greater than 0")
    @height.setter
    def height(self, new_height):
        if new_height > 0:
            self._height = new_height
        else:
            return print("width must be greater than 0")

R = rectangle(5,4)
print(R.width)
print(R.height)
R.width=-1
R.height=-24
print(R.width)
print(R.height)
R.height=9
R.width=9
print(R.width)
print(R.height)
```

OUTPUT:

4.0 CM
5.0 CM
width must be greater than 0
width must be greater than 0
4.0 CM
5.0 CM
9.0 CM
9.0 CM

4) Calling of Deleter: [useless but exists]

1) The delete method should be used after the class declaration under the deleter method call like:
@attr_name.deleter
2) Then within it we should define the attribute which needs deletion. The original internal value needs to be addressed for deletion with the help of an underscore like: _attr_name
@attr_name.deleter
def attr_name(self):
del self._attr_name

3)It can be called within the main function by the help of
del class.attr_name
Then the required attribute will be deleted.

FULL INSTANCE

CODE:

```
@width.deleter
def width(self):
    del self._width
    print("The width has been deleted successfully")
@height.deleter
def height(self):
    del self._height
    print("The height has been deleted successfully")

R = rectangle(5,4)
del R.width
del R.height
```

OUTPUT:

The width has been deleted successfully
The height has been deleted successfully

Exception

What are **Exception?**

-> A event that interrupts the Flow of a program.
(ZeroDivisionError, TypeError, ValueError)
1) try 2) except 3)finally

Types of Exceptions: There are various types of exceptions in Python, including:

ZeroDivisionError: Occurs when attempting to divide a number by zero.

EX:
1/0

TypeError: Raised when an operation is performed on an object of an inappropriate type, such as adding an integer to a string.

EX:
1+"1" [addition of integer with string]

ValueError: Happens when a function receives an argument of the right type but an inappropriate value, such as trying to convert a non-numeric string to an integer.

EX: #this happens specially during typecasting
int("pizza") #here pizza is a string clearly and can never be a integer

Exception Handling

We have to do exception handling if we find an exception insteading of running and crashing it. If we run into exceptions which might cause any of the 3 error here, it will completely break down and will not run no more. To handle it gracefully, we will have to handle them properly:

In the following steps of 3, it can be done:

1) Try:
Firstly, put in the program in try, where the values will be tried to be put in an operation

2) Exception:
Secondly, put the errors in the part for all 3 types of error and how you want them to be handled within the except part.

If on general all problems need to be dealth with then type:
except Exception:

3) Finally:
Put in what you want to do at the end of the program. This part of the code will run disregarding whether the code runs into an error or not

Full instance

```
try:    #code needed to be run
        number= int(input("Enter a number: "))
        print(1/number)
except TypeError: #exception handling
        print("Dude writing pizza or someshit wont help ya, Enter a damn number")
except ValueError:
        print("You need to enter a integer you stupid ass")

except ZeroDivisionError:
        print("Dude lemme ask you something...How can you divide a number with 0?")
        print("Do it yourself pls... I aint built for this")
except Exception: #all exceptions other than the ones mentioned are handled here
        print("Oops something went wrong")
finally: #will run regardless of the error or no error
        print("Thats allll folks")
```

Python File detection

To detect file paths we first need to deal with relative file path and absolute file path.
IF WANTED WE CAN USE / INSTEAD OF \ AS IT WILL BE NEEDED TO USE DOUBLE TIME

Relative file path:
If the files are within the same folder, then mentioning the file name is enough to find the file.

For example:
1) The os folder should be imported with "import os"
2) After making the file within the same folder, the folder can be called on with just the name.
3) We can check if the folder exists like:

```
if os.path.exists(file_path):
        print(f"The location {file_path} exists ")
```

4) If it stays in a folder around the same place, let us say the folder name is stuff

file_path= stuff\\test.txt

Full Instance

```
import os
file_path= "test.txt"
if os.path.exists(file_path):
        print(f"The file path {file_path} exists.")
else:
        print(f"The file path {file_path} does not exist")
```

OUTPUT:

The file path test.txt exists.

Absolute file path:

The actual file path of the file needs to be used for this case.
Here the actual file path needs to be added along the file name that needs to be chekced for detection.

Steps:

1)file_path="C:/Users/tahsin/OneDrive/Desktop/mox.txt"
[/ this slash should be used]
2) To find path import os at first.
3)Then afterwards, use "os.path"
4)Then again use "os.path.isfile(file_name)" to check if it is a file
5)Then again you can also use "os.path.isdir(file_name)" to check if it is a directory

Full instance

```
import os
file_path="D:/HUM 4147.rtf"
if os.path.exists(file_path):
        print(f"The file path {file_path} exists.")
        if os.path.isfile(file_path):
                print(f"{file_path} is a file path")
        elif os.path.isdir(file_path):
                print(f"{file_path} is a file directory")
else:
        print(f"The file path {file_path} does not exist")
```

Writing files

Step1:

take your file path in a variable. [both relative and absolute file can be used]

Step2:

with statement usage

With statement if it opens a file it closes it too. The code of block will be both opened or runned and also closed.

Step3:

opening it

Within the "with" statement just beside it we can write the open statement within which we will give our file path (stored in variable) and also the command that we want accomplished. Also, we will need to include the as "something" afterwards to name it as something for later use.

with open (file_path_variable_name, "command") as file:

We can also write them as KEYWORD ARGUMENTS to make it easier to understand:

with open(file=file_path, mode="w") as file:

Step4:

Giving modes

There are multiple possible modes which we can use. They all need to be written within the double quotes. Some of the modes are:

i) "w" for write if a file exists
ii) "x" for writing if a file does not exist [it will give an error if the file DOES EXIST]
iii)"r" for reading
iv)"a" for appending

Full Instance of writing

```
txt_data="Wiwu wiwu sulumulu lala mala"

file_path="test.txt"

with open (file_path,"w") as file:
    file.write(txt_data)
    print("Successful writing of a file was done")
```

A file with the name test.txt was created and written upon
If "x" was used and if the file existed, then it would give us an error. x should be used only when old file does not exist.

APPENDING: just put an "a" instead of "w" to append text or whatever. Then more and more text will be racked up

one over the other.

WRITING: just putting a "w" will help overwrite over the text file all over to only to what has been given to us.

Just by doing ("\n" + txt_data) within the file.write() part, we can add a line break everytime we append data to it

OUTPUT:

```
Wiwu wiwu sulumulu lala mala
Wiwu wiwu sulumulu lala mala
Wiwu wiwu sulumulu lala mala
Wiwu wiwu sulumulu lala mala
Wiwu wiwu sulumulu lala mala
Wiwu wiwu sulumulu lala mala
```

LIST INPUTTING

CODE:

```
list_data=["Mueej", "Al", "Baseet" , "Arkam", "Hossain", "Saad", "Tawfiq", "Ahmed", "Tahsin"]

file_path="test.txt"
with open (file_path,"w") as file:
        for name in list_data:
                file.write(name+ " ")


        print("Successful writing of a file was done")
```

OUTPUT:

```
Mueej Al Baseet Arkam Hossain Saad Tawfiq Ahmed Tahsin
```

DICTIONARY INPUTTING (.json files)

The text data and the file will need to be put within the parentheses of json.dump() and then after the file and the text data are put in the indent and the amount of space which needs to be put in after each key : value pair.

CODE:

```
#ALWAYS IMPORT JSON FIRST


import json

list_data={
```

```
        "name11":"Mueej",
        "name12":"Al",
        "name13":"Baseet",
        "name21":"Arkam",
        "name22":"Hossain",
        "name23":"Saad",
        "name31":"Tawfiq",
        "name32":"Ahmed",
        "name33":"Tahsin"
}

file_path="test.txt"
with open (file_path,"w") as file:
        json.dump(list_data, file, indent=5)

        print("Successful writing of a file was done")
```

Multi Dimensional Lists (.csv files)

```
# The csv library needs to be imported
# writer method needs to be used here
## Step1: saving the file within a variable
## Step2: using csv.write(file_name) within the variable
##Step3: Iterating for every row (list) within the list
##Step4: Using var_name.writerow(row) to     write in rows



import csv
list_data=[
        ["name","age","rizz level"],
        ["biggyatt fahim","21","89"],
        ["smolgyatt rhyme","21","10"],
]

file_path="test.txt"
with open (file_path,"w",newline="") as file:
        writer = csv.writer(file)
        for row in list_data:
                writer.writerow(row)
        print("Successful writing of a file was done")
```

Python reading files

Step1:

Taking file path within a variable

Step2:

with open(file_path_var, "r") as file: [r as the mode within the open]

Step3:

content= file.read() [storing the file as a variable]

Then we can print the content or whatever we want to do.

While reading files we may have to face multiple errors:
i) FilenotFoundError [when the file was not found]
ii) PermissionError [when no permission is given for opening the file]

****Always remember to bring a code [which has risks] within try and handle exceptions****

Full instance

CODE: [reading txt file]

```
file_path="test.txt"
try:
    with open (file_path,"r") as file:
        read= file.read()
        print(read)
except FileNotFoundError:
    print("The file path was not found.")
except PermissionError:
    print("You have no permission to open the file.")
```

CODE: [reading json file]

```
import json
# include the json file within a variable
file_path="test.txt"
try:
    with open (file_path,"r") as file:
        read= json.load(file)
```

```
            print(read)                              # for printing the whole of it
                print(read["key_name"])      # for printing a value of a key
except FileNotFoundError:
    print("The file path was not found.")
except PermissionError:
    print("You have no permission to open the file.")
```

CODE: [reading csv file]

```
import csv

file_path="test.csv"
try:
    with open (file_path,"r") as file:
        read= csv.reader(file)
        for line in read:
            print(line)                          #iterate to print the whole thing too
            print(line[0])       #iterate to print a specific line
except FileNotFoundError:
    print("The file path was not found.")
except PermissionError:
    print("You have no permission to open the file.")
```

Date & Time

First import datetime

To put in a date:

1) Take a variable to store the date
2) Call the library with datetime.
3) Call for date within the library datetime.date()
4)Then put the date reversely from year to month to day

storing_date_variable = **datetime.date(year , month , day)**

To store the date of today:

1) Take a variable and equalize it to the date today
2) Call for the library with datetime.day.
3) Call for today object with datetime.day.today()

printing the variable will give us the date today

To put in a time:

1) Take a variable
2) Call for time object datetime.time()
3) Put in the time hour and then minute and then second like:
    datetime.time(12, 30, 0)---------> daytime.time(hour, minute, second)

TO PUT IN BOTH DATE &TIME:

IF put all together, the date along the time,
    datetime.datetime(year, month, day, hour, minute, second)

To Get time of now:

1) same
2) Call for daytime.daytime [yes it is what it is]
3) Call for the now object like:
    datetime.datetime.now()

Printing the variable will give us the time of the exact moment it was then and also the date of that day.

OUTPUT:
2025-01-12 23:53:37.365620

But the format of the output can be handled.

FORMATTING TIME:

we have to call the strftime after storing the time within a variable first

now = datetime.datetime.now()
now = datetime.strftime()

Then, within the parentheses we will put the format how we want like:

%H, %M, %S [Higher case letters for Hour, Minute & Second]
%m, %d, %Y [m and d for month and day respectively and Y for the year]

CODE:

```
import datetime

now=datetime.datetime.now()
now=now.strftime("%H: %M: %S \n%d\ %m\ %Y")
print(now)
```

OUTPUT:

00: 02: 49
13\ 01\ 2025

To check for a target time:

We can check if a time is before or after our given time.

CODE:

```
import datetime

now=datetime.datetime.now()
then=datetime.datetime(2025, 1, 14, 23, 0, 45) #mention datetime object

if now > then:
    print("The time has already passed")
else:
    print("The time has not passed")
```

ALARM CLOCK PROGRAM

**We will need to import

1)time
2)datetime
3)pygame

** TO USE PYGAME FOR RUNNING MUSIC:

i) We will definetely need to import it duh
ii) We will have to initialize the mixer with pygame.mixer.init()
iii)We will have to load it pygame.mixer.music.load(filename)
iv)We will have to then play it pygame.mixer.music.play()
v)To run it until it ends while pygame.mixer.music.get_busy():
time.sleep(1)

**We will need a music file

```
import pygame
import time
import datetime

import pygame.mixer
```

```
def set_alarm(alarm_time):
    print(f"The alarm has been set for {alarm_time}")
    music_file = "C:/Users/tahsin/Downloads/danda.mp3"
    is_running = True
    while is_running:
        current_time= datetime.datetime.now().strftime("%H:%M:%S")
        print(current_time)

        time.sleep(1)

        if current_time==alarm_time:
            print("Wake Up!!!!")
            pygame.mixer.init()
            pygame.mixer.music.load(music_file)
            pygame.mixer.music.play()

            while pygame.mixer.music.get_busy():
                time.sleep(1)

            is_running=False


if __name__ == "__main__":
    alarm_time= input("Enter the time for setting the alarm: HH:MM:SS: ")
    set_alarm(alarm_time)
```

Multithreading

What is **Multithreading?**

*Used to perform multiple tasks concurrently (multitasking)
*Good for I/O bound tasks like reading files and fetching data from APIs
*Threading. Thread(target = my_function)

Purpose: Multi-threading is used to perform multiple tasks simultaneously, which can improve the efficiency of programs, especially for I/O-bound tasks such as reading files, fetching data from APIs, or handling user input.

Threading Module: To use multi-threading in Python, you import the threading module. This module provides a way to create and manage threads.

Creating Threads:

**You can create a thread by instantiating the Thread class from the threading

module.

**You need to pass a target function that the thread will execute.

STEPS:
1) Import the thread function.
2) Define a self defined function
3) Reinstate the threading library and call for the Thread object. Like:

thread_variable = threading.Thread (target = self_def_func_name)

For example:

```
import threading

def task():
    print("Task is running")


thread = threading.Thread(target=task)
thread.start()
```

IF thread required arguments:

Step1: Defining a self defined function having a argument requirement
Step2: While calling for a thread we need to pass in arguments [in tuple]

*To show that the arguments are in a tuple, we will need to call for the "args=" and then we need to put in a comma after our argument just to show that it is a tuple.

thread.Thread(target=task, args=("Argument name",))

 # if multiple arguments are present
[it is already a tuple when having multiple elements in first bracket]
thread.Thread(target=task, args=("Argument name1", "Argument name2"))

FULL INSTANCE

```
import threading
import time
def walk_dog(dog):
    time.sleep(4)
    print(f"You finished walking {dog}")
chore1=threading.Thread(target= walk_dog, args=("Shishumanu",))
chore1.start()
```

OUTPUT:
You finished walking Shishumanu

Concurrency:

*Threads run concurrently, which means they can execute at the same time.

*This is particularly useful for tasks that involve waiting for external resources, *allowing other threads to run while one is waiting.

Creating Multiple Threads:

Let us say, we want multiple threads to run together at the same time. All start at the s ame time and end when they are supposed to. Then we will have to write the code in this way:

#Getting them to work altogether

```
import threading
import time
def walk_dog():
    time.sleep(4)
    print("You finished walking the dog")

def take_trash():
    time.sleep(3)
    print("You took out the trash")

def get_mail():
    time.sleep(2)
    print("You got the mail")

chore1=threading.Thread(target= walk_dog)
chore1.start()
chore2=threading.Thread(target= take_trash)
chore2.start()
chore3=threading.Thread(target = get_mail)
chore3.start()
```

Thread Synchronization:

**When multiple threads access shared resources, synchronization is necessary to prevent data corruption.

*Python provides several mechanisms for synchronization:

1) **Locks**: *allows only one thread to access a resource at a time.

2) **Semaphores**: *controls access to a resource by multiple threads.

3) **Events**: *a simple way for one thread to signal another that a certain condition has occurred.

4) **Joining Threads:** **We can use the join() method to wait for a thread to finish its execution before proceeding with the rest of the program.
**This is useful when you need to ensure that all threads complete their tasks before moving on.

To start together and end with a ending message altogether after the threads are run, we will need to use the join method. All of them will be chained together and run and then after the end, a ending message will be printed. It will be done like:

CODE:

```
import threading
import time
def walk_dog():
    time.sleep(4)
    print("You finished walking the dog")

def take_trash():
    time.sleep(3)
    print("You took out the trash")

def get_mail():
    time.sleep(2)
    print("You got the mail")

chore1=threading.Thread(target= walk_dog)
chore1.start()
chore2=threading.Thread(target= take_trash)
chore2.start()
chore3=threading.Thread(target = get_mail)
chore3.start()

chore1.join()
chore2.join()    #tasks joined will start together
chore3.join()

print("All tasks are complete")    #after the threads are taken care of, the last printing will
                                    occur.
```

5) **Thread Lifecycle:** **A thread can be in several states: new, runnable, blocked, waiting, or terminated.
**Understanding these states helps in managing thread behavior effectively.

Limitations: Python's Global Interpreter Lock (GIL) can limit the effectiveness of multi-threading for CPU-bound tasks, as it allows only one thread to execute Python bytecode at a time. For CPU-bound tasks, multi-processing (using the multiprocessing module) may be more effective.

Example: Here's a simple example demonstrating multi-threading:

```
import threading
import time

def walk_dog():
    time.sleep(8)
    print("Finished walking the dog")

def take_out_trash():
    time.sleep(2)
    print("Finished taking out the trash")



def get_mail():
    time.sleep(4)
    print("Finished getting the mail")

# Create threads
threads = []
threads.append(threading.Thread(target=walk_dog))threads.append(threading.Thread(target=take_out
_trash))threads.append(threading.Thread(target=get_mail))



# Start threads

for thread in threads:
        thread.start()



# Wait for all threads to complete

for thread in threads:
        thread.join()
print("All chores are complete.")
```

Use Cases: Multi-threading is particularly useful in applications that require responsiveness, such as GUI applications, web servers, and network applications, where tasks can be performed in the background without blocking the main thread.

Connecting to an API using Python

Connecting to an API (Application Programming Interface) in Python typically involves making HTTP requests to interact with a web service. Here are the key points on how to connect to an API:

pip install requests

Making a GET Request:

*****To retrieve data from an API, you use a GET request. Here's a basic example:

```
import requests
response = requests.get('https://api.example.com/data')

if response.status_code == 200:
    data = response.json()    # Parse the JSON response
    print(data)

else:
    print("Error:", response.status_code)
```

Making a POST Request:

*****To send data to an API, you typically use a POST request. This is common for creating new resources. Example:

```
import requests
payload = {'key1': 'value1', 'key2': 'value2'}
response = requests.post('https://api.example.com/data', json=payload)

if response.status_code == 201:
    print("Resource created:", response.json())
else:
    print("Error:", response.status_code)
```

Headers and Authentication:

*Many APIs require authentication and specific headers. You can include headers in your requests like this:

```
headers = {'Authorization': 'Bearer YOUR_API_TOKEN'}
response = requests.get('https://api.example.com/data', headers=headers)
```

Error Handling:

*Always check the response status code to handle errors appropriately. Common status codes include:

200: OK

201: Created

400: Bad Request

401: Unauthorized

404: Not Found

500: Internal Server Error

Rate Limiting: Be aware of the API's rate limits, which restrict the number of requests you can make in a given time period. Exceeding these limits can result in temporary bans.

Documentation: Always refer to the API's documentation for specific endpoints, required parameters, and response formats. This documentation is crucial for understanding how to interact with the API effectively.

Example of Connecting to a Public API: Here's an example of connecting to a public API (e.g., JSONPlaceholder):

```
import requests


response = requests.get('https://jsonplaceholder.typicode.com/posts')

if response.status_code == 200:

    posts = response.json()

    for post in posts:
```

```
print(post['title'])
```

Asynchronous Requests: For better performance, especially when dealing with multiple API calls, consider using asynchronous requests with libraries like aiohttp or httpx.

Connecting to APIs allows you to access and manipulate data from various services, enabling you to build powerful applications that leverage external data sources.

FULL INSTANCE

```
import requests

base_url="https://pokeapi.co/api/v2/"

def get_pokemon_info(name):
    url = f"{base_url}/pokemon/{name}"
    response= requests.get(url)      #using a GET method to retrieve data

    if response.status_code == 200: #checking if status code is fine [200 for OK]
        pokemon_data = response.json()    #storing data in json format
        return pokemon_data
    else:
        print(f"Error {response.status_code} while fetching data")

pokemon_name= input("What pokemon do you want info on?").lower()
pokemon_info = get_pokemon_info(pokemon_name)

if pokemon_info:
    print(f"Name:      {pokemon_info['name'].capitalize()}")
    print(f"Height: {pokemon_info['height']}")
    print(f"Id:          {pokemon_info['id' ]}")
    print(f"Weight: {pokemon_info['weight']}")
```

OUTPUT:

What pokemon do you want info on?snorlax
Name: Snorlax
Height: 21
Id: 143
Weight: 4600

GUI [Graphics User Interface]

*At first import the sys library
*then from PyQt5.QWidgets import QMainwindow, QApplication

Getting a basic window:

CODE:

```
import sys
from PyQt5.QtWidgets import QMainWindow,QApplication

class MainWindow(QMainWindow):
        def __init__(self):
                super().__init__()

def main():
        app=QApplication(sys.argv)
        window = MainWindow()        # window initialization
        window.show()                    # showing window
        sys.exit(app.exec_())        # waiting until it is closed by user

if __name__ == "__main__" :
        main()
```

To get title on the titlebar of the application we are gonna make, we will have to:

*Set **self.setWindowTtile** equals to the string of the name which we want to set the title

```
class MainWindow(QMainWindow):
        def __init__(self):
                super().__init__()
                self.setWindowTitle("My first GUI")
```

To set where the window will appear, we will have to:

*Set self.setGeometry() and inside we will need to put ion the arguments.

like :

```
self.setGeometry(x,y,widgth,height)
```

x=0, y=0 is exactly at the upper left corner of the screen. From there the axis measurement starts. Then the width & height of the window can be set too as we want to in measurement of pixels like:

```
class MainWindow(QMainWindow):
```

```
    def __init__(self):
        super().__init__()
        self.setWindowTitle("My first GUI")
        self.setGeometry(700,250,500,500)
```

To set the icon of the application:

* First we will need to import QIcon from PyQt5.QGui
* Then for setting the icon of the application write: self.setWindowIcon()
*Afterwards inside of the parentheses, self.setWindowIcon(QIcon())
*Then within the empty parentheses of QIcon put in the file path of the image like:
self.setWindowIcon(QIcon(D\Downloads\image.jpg))

FULL INSTANCE [Creation of a window having title & Icon]

```
import sys
from PyQt5.QtWidgets import QMainWindow,QApplication
from PyQt5.QtGui import QIcon

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("My first GUI")
        self.setGeometry(700,250,500,500)
        self.setWindowIcon(QIcon("C:/Users/tahsin/Downloads/bob.png"))

def main():
    app=QApplication(sys.argv)
    window = MainWindow()    # window initialization
    window.show()    # showing window
    sys.exit(app.exec_())    # waiting until it is closed by user

if __name__ == "__main__" :
    main()
```

PYQT5 QLabels

For labels, first import library equipments,

We will need to import:

```
import sys
from PyQt5 import QMainWindow, QApplication, QLabel      # Qlabel for labels
from PyQt5.QtGui import Qfont    #for importing fonts
```

To add a label to the program:

# To add label, we need to take a variable for the label and then store the label inside of it like:
label = Qlabel("Hello" , self)

```
class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("My first GUI")

        label = QLabel("Hello", self)
```

 # first argument will be the string and second argument will be attributed to self which redirects to the MainWindow.

To add font type & size to the label:

Within the Main Window Class, we will need to write the method:
label_name.setFont()

Now within the method, we will need to call the constructor QFont:
label_name.setFont(QFont())

Then afterwards we will need to pass the font name and size as arguments:
label_name.setFont(Qfont("Agency FB", 40))

To set geometry of the label:

* This has to be done in the same manner as setting geometry before.

label_name.setGeometry(x, y, height, width)

To set stylesheet of the label:

*To set style to a label, we will write:
label.setStyleSheet()

SETTING COLOURS

* Then, within double quotes, we will write the color we want or anything:
label_name.setStyleSheet("color : blue;") [these properties should end with a semicolon]

We can also use hexadecimal colours if we want within the colour part:
label_name.setStyleSheet("color : #FFFFFF;") # for black

SETTING BGCOLOURS:* label_name.setStyleSheet("background-color: #6fdcf7;")

SETTING BOLDNESS: *label_name.setStyleSheet("font-weight: bold;")
#it can be light too if wanted

**SETTING STYLE: ***label_name.setStyleSheet("font-style: italic;")

**SETTING DECORATION: ***label_name.setStyleSheet("text-decoration: underline;")

SETTING ALLIGNMENTS:

*First you will need to import Qt from PyQt5.QtCore which provides allignments

Singular Flag

Top Allignment	label_name.setAlignment(Qt.AlignTop)
Bottom	Qt.AllignBottom
Center [Vertical]	Qt.AlignVCenter [by default]
Right	Qt.AlignRight
Left	Qt.AlignLeft
Center [Horizontal]	Qt.AlignHCenter
Center [abs center]	Qt.AlignCenter

Combining Flags:

Combining Multiple Flags together is also possible like this:
label_nameAlignment(Qt.AlignCenter | Qt.AlignTop) # | acts as he or operator

Combined Flags

Center & Top	Qt.AlignCenter | Qt.AlignTop
Center & Bottom	Qt.AlignCenter | Qt.AlignBottom
Center & Center	Qt.AlignVCenter | Qt.AlignHCenter

PyQt5 images

*We wil need
QApplication, QMainWindow, QLabel from the PyQt5.QtWidgets

***FOR IMAGES WE WILL NEED,**

QPixmap from the PyQt5.QtGui

How to add pictures to labels?

Step1: Opening a label

Step2: **Setting label Geometry**

Setting the geometry of the label properly

Step3: **Storing of image to variable**

Storing the image in a variable using QPixmap("pic.jpg")

Step4: **Inserting image to the label**

Calling the label to set the image in it:

label.setPixmap(image_variable_name)

Step5: **Scaling of Image**

The image might not show fully for the label size we have mentioned. So It needs scaling to be done. The image size will be dialed down as like as the label size given.

label.setScaledContent(True)

FULL INSTANCE

```
import sys
from PyQt5.QtWidgets import QMainWindow,QApplication,QLabel
from PyQt5.QtGui import QPixmap

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("My first GUI")
        self.setGeometry(700,300,250,250)
        label = QLabel(self)
        label.setGeometry(0,0,250,250)
        pixmap= QPixmap("C:/Users/tahsin/Downloads/bob.png")
        label.setPixmap(pixmap)
        label.setScaledContents(True)
        label.setGeometry(0,0,label.height(),label.width())



def main():
    app=QApplication(sys.argv)
```

```
    window = MainWindow()    # window initialization
    window.show()    # showing window
    sys.exit(app.exec_())    # waiting until it is closed by user

if __name__ == "__main__" :
    main()
```

How to move the picture into positions as we want?

After setting the geometry from before, we have to set the geometry again later after scaling the image.

We can call for the height of a label we have used before by:

label.width()
label.height()

Again, at the same time we can also call for the Main window height and width too:

self.height()
self.width()

Placing of the image:

After the scaling of the image we will again take the label and set its geometry.

For Left justified image

it will stay as it is

For Right justified image:

label.setGeometry(self.width()-label.height(),0,label.width(),label.height())
Explanation:

1) When we move to the right with positive self.width() to the right side, we are starting the image from basically outside the panel as the width of the main window ends there.

2) Then if we move to the left by subtracting the label.width() from the self.width(), then we get in exactly inside by the width of the image i.e the whole image is fit touching the boundary.

For bottom left:

label.setGeometry(0, self.height()-label.height(), label.height(), label.width())

For bottom right:

label.setGeometry(self.width()-label.width(), self.height()-label.height(), label.height(), label.width())

For Center:

label.setGeometry(self.width()-label.width() // 2 , self.height()-label.height() // 2, label.width() , label height())

Explanation:
** **We can get direct division with the help of "/" but with "//" we get a rounded up division without the decimal parts as PIXELS CAN NEVER BE IN DECIMALS.#**

PyQt5 layouts

We will need:

```
import sys
from PyQt5.QtWidgets import
                    (QApplication,
                    QMainWIndow,
                    QWidget,
                    QVBoxLayout,
                    QHBoxLayout,
                    QLayout)
```

i) Doing everything within the MainWindow Class will cause the program to be totally unarranged, so to make things better, we will need to initialize the User Interface as a seperate function which wil have no arguments accept "self" . Like:

```
def initUI (self):
        pass
```

ii) **UI initialization:**

Now, when we construct a window object, we wil call self.initUI() within the MainWindow class. Like:

```
class MainWindow(QMainWindow):
        def __init__(self):
                super().__init__()
                self.setGeometry(700, 300, 500, 500)
                self.initUI()
        def initUI(self):
                pass
```

iii) **Adding a Central Widget:**

Normally, we cant add a layout manager within the main window as main window widgets have a specific design and layout structure which is incompatible with the layout managers.

So, we will need to create a generic widget and add a layout manager to that widget. Then, we will add the widget to the main window to display the layout within our method to initialize our user interface. That generic widget will be our Central Widget.

So, we will call for the central_widget within the def initUI(self) like:

```
        def initUI(self):
                central_widget = Qwidget()       #setting up of a widget
                self.setCentralWidget(central_widget)      # adding the central widget
```

FULL INSTANCE

```
import sys
from PyQt5.QtWidgets import (QMainWindow,QApplication,QLabel,
                                        QWidget,QHBoxLayout,QVBoxLayout,
                                         QGridLayout)

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setGeometry(700,300,250,250)
        self.initUI()

    def initUI(self):
        central_widget= QWidget()
        self.setCentralWidget(central_widget)



def main():
    app=QApplication(sys.argv)
    window = MainWindow()    # window initialization
    window.show()    # showing window
```

```
    sys.exit(app.exec_())    # waiting until it is closed by user

if __name__ == "__main__" :
    main()
```

iii) Adding Layout:

 # STEP1: Making Labels within the defined initUI method

```
def initUI(self):
    central_widget= QWidget()
    self.setCentralWidget(central_widget)
    label1= QLabel("1", self)
    label2= QLabel("2", self)
    label3= QLabel("3", self)
    label4=QLabel("4",self)
    label5=QLabel("5", self)
```

 # STEP2: Adding Stylesheet # Unnecessary but important to understand layout

```
    label1.setStyleSheet("background-color: blue;")
    label2.setStyleSheet("background-color: red;")
    label3.setStyleSheet("background-color: green;")
    label4.setStyleSheet("background-color: indigo;")
    label5.setStyleSheet("background-color: purple;")
```

 # STEP3: To store the layout within a variable & adding to widget:

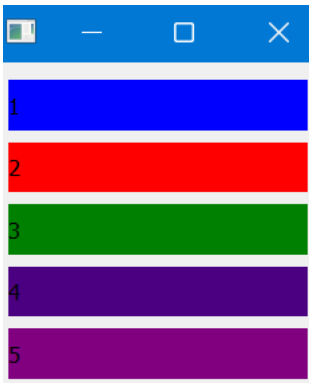
```
        vbox= QVBoxLayout()

    vbox.addWidget(label1)
    vbox.addWidget(label2)
    vbox.addWidget(label3)
    vbox.addWidget(label4)
    vbox.addWidget(label5)
```

 # STEP4: Applying the Layout to Central Widget:

```
    central_widget.setLayout(vbox)
```

OUTPUT:



Horizontal Layout [FULL INSTANCE]

```
import sys
from PyQt5.QtWidgets import (QMainWindow,QApplication,QLabel,
                                        QWidget,QHBoxLayout,QVBoxLayout,
                                         QGridLayout)

class MainWindow(QMainWindow):
      def __init__(self):
            super().__init__()
            self.setGeometry(700,300,250,250)
            self.initUI()

      def initUI(self):
            central_widget= QWidget()
            self.setCentralWidget(central_widget)
            label1= QLabel("1", self)
            label2= QLabel("2", self)
            label3= QLabel("3", self)
            label4=QLabel("4",self)
            label5=QLabel("5", self)

            label1.setStyleSheet("background-color: blue;")
            label2.setStyleSheet("background-color: red;")
            label3.setStyleSheet("background-color: green;")
            label4.setStyleSheet("background-color: indigo;")
            label5.setStyleSheet("background-color: purple;")
            hbox= QHBoxLayout()

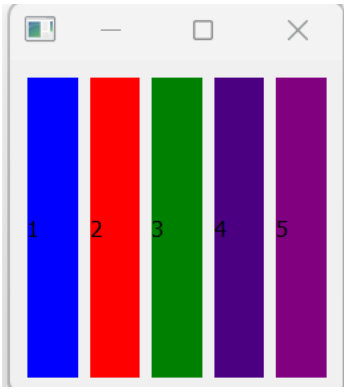
            hbox.addWidget(label1)
            hbox.addWidget(label2)
            hbox.addWidget(label3)
            hbox.addWidget(label4)
```

```
            hbox.addWidget(label5)

            central_widget.setLayout(hbox)
def main():
    app=QApplication(sys.argv)
    window = MainWindow()    # window initialization
    window.show()    # showing window
    sys.exit(app.exec_())    # waiting until it is closed by user

if __name__ == "__main__" :
    main()
```

OUTPUT:



Grid Layout

For the Grid Layout Arrangement, the row and coloum number needs to be mentioned when we add the layouts. Like:

grid.addWidget(label_name, row, column)

Full instance

```
import sys
from PyQt5.QtWidgets import (QMainWindow,QApplication,QLabel,
                                        QWidget,QHBoxLayout,QVBoxLayout,
                                         QGridLayout)

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setGeometry(700,300,250,250)
        self.initUI()
```

```
        def initUI(self):
                central_widget= QWidget()
                self.setCentralWidget(central_widget)
                label1= QLabel("1", self)
                label2= QLabel("2", self)
                label3= QLabel("3", self)
                label4=QLabel("4",self)
                label5=QLabel("5", self)

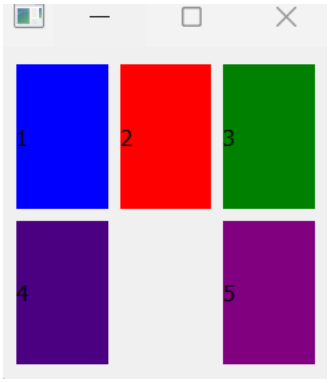
                label1.setStyleSheet("background-color: blue;")
                label2.setStyleSheet("background-color: red;")
                label3.setStyleSheet("background-color: green;")
                label4.setStyleSheet("background-color: indigo;")
                label5.setStyleSheet("background-color: purple;")
                gbox= QGridLayout()

                gbox.addWidget(label1, 0,0)
                gbox.addWidget(label2, 0,1)
                gbox.addWidget(label3, 0,2)
                gbox.addWidget(label4, 1,0)
                gbox.addWidget(label5, 1,2)

                central_widget.setLayout(gbox)
def main():
        app=QApplication(sys.argv)
        window = MainWindow()    # window initialization
        window.show()    # showing window
        sys.exit(app.exec_())    # waiting until it is closed by user

if __name__ == "__main__" :
        main()
```

OUTPUT:



Creating Push Buttons

How to create Pushing Buttons?

STEP 1: Importing Necessary Library Equipments

```
import sys
from PyQt5.QWidgets import QMainWindow, QApplication, QPushButton, Qlabel
```

STEP 2: Saving the Button in a Variable [will be done under the initUI part]

****ANY INSTANCE OF BUTTON WE WILL PREFIX IT WITH self** [as it is the button of the main window class]**

```
        self.button_var= QPushButton()
```

This PushButton function has two attributes:

1) What is written on the button [A string]
2) Then, another attribute (self)

The format for attribute input that is followed will be:

```
        self.button_var= QPushButton( "Writting on the button", self)
```

STEP 3: Setting the command which occurs on click:

1) We will need to set a function, which occurs on click of the button.
2) Then after doing so we will need to connect the click to that function.
3) We will call the clicked object of the button variable and the call the connect function
4)Then within the function we will put in self.func_name

```
def initUI(self):
        self.button=QPushButton("Button er lekha", self)
        self.button.setGeometry(150,200,200,100)
        self.button.setStyleSheet("font-size: 30px;")
        self.button.clicked.connect( self.click_func)      #connecing the function to button

def click_func():      # the connected function
        print("The button was clicked")
```

MISCALLENOUS STEP 4: Changing the button text after Clicking:

Within the click function we will need to write,

```
        self.button.setText("Clicked!!")      #to change the text of the button
```

MISCALLENOUS STEP 5: Disabling the button after a click:

Within the click function we will need to write,

```
        self.button.setDisable(True)                    #Disabling button
```

MISCALLENOUS STEP 6: Changing Label Text after a click:

Within the click function we will need to write,

```
        self.label.setText("Goodbye")
```

FULL INSTANCE

```
import sys
from PyQt5.QtWidgets import (QMainWindow,QApplication,QLabel,
```

```
                                        QPushButton)

class MainWindow(QMainWindow):
        def __init__(self):
                super().__init__()
                self.setGeometry(700,300,250,250)
                self.initUI()
                self.label=QLabel("Mew",self)
                self.label.setGeometry(0,0,1000,20)
        def initUI(self):
                self.button=QPushButton("Click me to kill the cat!!", self)
                self.button.setGeometry(150,200, 300, 300)
                self.button.setStyleSheet("font-size: 15px;")
                self.button.clicked.connect(self.click_func)
        def click_func(self):
                self.button.setText("Clicked")
                self.button.setStyleSheet("background-color : red;")
                self.button.setText("The cat is dead :(")
                self.label.setText("No more mew mew")
                self.button.setDisabled(True)



def main():
        app=QApplication(sys.argv)
        window = MainWindow()    # window initialization
        window.show()    # showing window
        sys.exit(app.exec_())    # waiting until it is closed by user

if __name__ == "__main__" :
        main()
```

CheckBoxes [PyQt5]

How to make checkboxes?

STEP1: Import the necessary Libraries:

import sys
from PyQt5.QtWidgets import QApplication, QMainWindow, **QCheckBox**
from PyQt5.QtCore import Qt

 # The QtCore module in PyQt5 (or PySide2) is a fundamental part of the Qt framework, providing core non-GUI functionality.

STEP2: Call for QCheckBox class:

1) We save the checkbox in a variable under the initialization part where we call on to the function QCheckBox class like:

```
self.checkbox = QCheckBox()
```

2) Then, we will need to give as the positional arguments within the empty parentheses of the QCheckBox class which are the text beside it and the self, like:

self.checkbox_variable("any text" , self)

FULL INSTANCE

```
class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setGeometry(700,300,250,250)
        self.checkbox= QCheckBox("Do you BLEED?", self)
```

STEP3: Initialize the User Interface for the Checkbox:

1) Define a new function "initUI" and pass in "self" as the attribute & under the function set stylesheet, font size as you like. Ex:

```
def initUI(self):
    self.checkbox.setGeometry(10,10,100,20)
    self.checkbox.setStyleSheet("background-color: 30px;"
                                "font-family: Agency FB;")
```

2) Then, Choose the state of the checkbox as you want with the "setChecked" class within the initialization the User Interface. It can be Either "False" or "True". True for checkmarked and False for the opposite :

```
self.checkbox.setChecked(False)
```

STEP4: Connecting to a Function [when State Changed] :

1) First Define a function under the mainWindow class and the pass in the Arguments "self" and state respectively. State defines if the checkbox is marked and self attributes it to itself.

```
def click_func(self, state):
```

2) Then Within the Function, Write whatever you want to happen if the state is changed. But, when state is changed there are two things that occur. It has two integer values :

i) When "True":

The state will be "2"

ii) When "False":

The state will be "0"

This value can also be called onto as "Qt.Checked" which is also returns integer value "2" . For better readability it is used to check the state of the checkbox.

INSTANCE

```
def click_func(self, state):
    if state == Qt.Checked:
```

```
            print("You do be bleeding")
            print("checked")
        else:
            print("ILL MAKE YOU BLEED")
            print("unchecked")
```

3) **CONNECTING** THE FUNCTION:

```
def initUI(self):
        self.checkbox.setGeometry(10,10,100,20)
        self.checkbox.setStyleSheet("background-color: 30px;"
                                    "font-family: Agency FB;")
        self.checkbox.setChecked(False)
        self.checkbox.stateChanged.connect(self.click_func)
```

FULL INSTANCE

```
import sys
from PyQt5.QtWidgets import (QMainWindow,QApplication,QCheckBox)
from PyQt5.QtCore import Qt

class MainWindow(QMainWindow):
        def __init__(self):
            super().__init__()
            self.setGeometry(700,300,250,250)
            self.checkbox= QCheckBox("Do you BLEED?", self)
            self.initUI()

        def initUI(self):
            self.checkbox.setGeometry(10,10,100,20)
            self.checkbox.setStyleSheet("background-color: 30px;"
                                        "font-family: Agency FB;")
            self.checkbox.setChecked(False)
            self.checkbox.stateChanged.connect(self.click_func)

        def click_func(self, state):
            if state == Qt.Checked:
                print("You do be bleeding")
                print("checked")
            else:
                print("ILL MAKE YOU BLEED")
                print("unchecked")

def main():
        app=QApplication(sys.argv)
        window = MainWindow()    # window initialization
        window.show()    # showing window
        sys.exit(app.exec_())    # waiting until it is closed by user

if __name__ == "__main__" :
```

```
main()
```

PyQt5 radio buttons

How to make **radio buttons?**

STEP1: Import the required Libraries:

```
import sys
from PyQt5.QtWidgets import (QMainWindow,QApplication, QRadioButton,
        QButtonGroup,QLabel)

# This is to get the Radio buttons [QRadioButton]
# This is to make groups of buttons [QButtonGroup]
```

STEP2: To initialize the radio buttons:

1) Calling on to the RadioButton & storing it within a variable.
2) Then, passing the text within and self as attributes.
Format: self.radio_name=QRadioButton("TEXT", self)

FULL INSTANCE

```
class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setGeometry(700,300,250,250)
        self.label1 = QLabel("Choose a Payment option:",self)
        self.radio1 = QRadioButton("Visa", self)
        self.radio2 = QRadioButton("MasterCard", self)
        self.radio3 = QRadioButton("Gift Card", self)
        self.label2= QLabel("Enter Pick up method:",self)
        self.radio4 = QRadioButton("Store pick", self)
        self.radio5 = QRadioButton("Home delivery", self)
```

STEP3: To initialize groups of button:
Within the constructor, The buttons need to be initialized:

```
self.button_group_name1 = QButtonGroup(self)
self.button_group_name2 = QButtonGroup(self)
```

STEP4: To initialize the UI [User Interface]:

1) Within the UI initialization, enter the label's Geometry as far needed:

```
self.label1.setGeometry(0,0,300,50)
self.radio1.setGeometry(0,20,300,50)
self.radio2.setGeometry(0,40,300,50)
self.radio3.setGeometry(0,60,300,50)
self.label2.setGeometry(0,80,300,50)
```

```
self.radio4.setGeometry(0,100,300,50)
self.radio5.setGeometry(0,120,300,50)
```

2) Set a stylesheet:

 # A more efficient way of doing this if multiple stylesheet are added at once

```
self.setStyleSheet("""
        QLabel {
                font-size:20px;
                font-family: Agency FB;
                padding: 10px;
                }
        QRadioButton{
                font-size:20px;
                font-family:Agency FB;
                padding:10px;
                }
        """)
```

3) Adding the buttons to their Respective groups: [IMPORTANT]

This will be done within the USER INTERFACE initialization.

FORMAT:

self.button_group_name1.addButton(self.radio_button_name)

FULL INSTANCE

```
self.button_group_1.addButton(self.radio1)
self.button_group_1.addButton(self.radio2)
self.button_group_1.addButton(self.radio3)
self.button_group_2.addButton(self.radio4)
self.button_group_2.addButton(self.radio5)
```

STEP4: Setting action if **Toggled**:

1)Define a function and attribute it to self.

2) Save the button name if toggled:

TO get the button name which was selected/deselected we use " sender() "

```
def radio_button_changed(self):
        radio_button = self.sender()    #To check which button was toggled
```

3) Setting action for when it is toggled:

To CHECK the button which was CHECKMARKED we use "radio_button.isChecked()**"**
To Check the button name which was clicked we use "radio_button.text()**"

TO RETURN SOMETHING IF CLICKED:

```
if radio_button.isChecked():
        print(f"{radio_button.text()} is selected")
```

FULL INSTANCE

```
import sys
from PyQt5.QtWidgets import (QMainWindow,QApplication, QRadioButton, QButtonGroup,QLabel)

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setGeometry(700,300,250,250)
        self.label1 = QLabel("Choose a Payment option:",self)
        self.radio1 = QRadioButton("Visa", self)
        self.radio2 = QRadioButton("MasterCard", self)
        self.radio3 = QRadioButton("Gift Card", self)
        self.label2= QLabel("Enter Pick up method:",self)
        self.radio4 = QRadioButton("Store pick", self)
        self.radio5 = QRadioButton("Home delivery", self)

        self.button_group_1 = QButtonGroup(self)
        self.button_group_2 = QButtonGroup(self)

        self.initUI()

    def initUI(self):
        self.label1.setGeometry(0,0,300,50)
        self.radio1.setGeometry(0,20,300,50)
        self.radio2.setGeometry(0,40,300,50)
        self.radio3.setGeometry(0,60,300,50)
        self.label2.setGeometry(0,80,300,50)
        self.radio4.setGeometry(0,100,300,50)
        self.radio5.setGeometry(0,120,300,50)
        self.setStyleSheet("""
            QLabel {
                font-size:20px;
                font-family: Agency FB;
                padding: 10px;
                }
            QRadioButton{
```

```
                    font-size:20px;
                    font-family:Agency FB;
                    padding:10px;
                    }
                """)
        self.button_group_1.addButton(self.radio1)
        self.button_group_1.addButton(self.radio2)
        self.button_group_1.addButton(self.radio3)
        self.button_group_2.addButton(self.radio4)
        self.button_group_2.addButton(self.radio5)

        self.radio1.toggled.connect(self.radio_button_changed)
        self.radio2.toggled.connect(self.radio_button_changed)
        self.radio3.toggled.connect(self.radio_button_changed)
        self.radio4.toggled.connect(self.radio_button_changed)
        self.radio5.toggled.connect(self.radio_button_changed)

    def radio_button_changed(self):
        radio_button = self.sender()    #To check which button was toggled

        if radio_button.isChecked():
            print(f"{radio_button.text()} is selected") #to get the radio button name

def main():
    app=QApplication(sys.argv)
    window = MainWindow()    # window initialization
    window.show()    # showing window
    sys.exit(app.exec_())    # waiting until it is closed by user

if __name__ == "__main__" :
    main()
```

Line Edit Widgets (text boxes)

How to make **line boxes?**

STEP1: Import necessary libraries:

```
import sys
from PyQt5.Widgets import QApplication, QMainWindow, QLineEdit
```

STEP2: Call for the QLineEdit function:

Store the Line Edit constructor within a variable & pass in self as attribute.
Like:

```
self.line_edit = QLineEdit(self)
```

STEP3: Setting UI for the Line Edit Widget:

We can put in whatever text we want within the text box before the input with the help of **setPlaceholder()**

```
self.line_edit.SetGeometry(10,10,200,40)
self.line_edit.SetStyleSheet("""
                                    font-size : 30 px;
                                    font-family : Agency FB;
                                      """)
self.line_edit.setPlaceholder("text")
```

What we can do,

Call for the text within the textbox which will be written after the execution of the program.

FULL INSTANCE

```
import sys
from PyQt5.QtWidgets import (QMainWindow,QApplication, QLineEdit,QPushButton)

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setGeometry(700,300,250,250)
        self.line_edit = QLineEdit(self)
        self.button=QPushButton("submit",self)
        self.initUI()

    def initUI(self):
        self.line_edit.setGeometry(10, 10, 200, 40)
        self.button.setGeometry(210,10,50,50)
        self.setStyleSheet("""
                            QLineEdit{
                                font-size : 30 px;
                                font-family : Agency FB;}
```

```
                                        QPushButton{
                                        font-size : 30;
                                        font-family : Agency FB;
                                        }
                                          """)
            self.line_edit.setPlaceholderText("Enter your name")
            self.button.clicked.connect(self.submit)
        def submit(self):
            print(f"Welcome {self.line_edit.text()}")



def main():
        app=QApplication(sys.argv)
        window = MainWindow()    # window initialization
        window.show()    # showing window
        sys.exit(app.exec_())    # waiting until it is closed by user

if __name__ == "__main__" :
        main()
```

PyQt5 STYLESHEET

hover option: In CSS, the :hover pseudo-class is used to apply styles to an element when the user hovers over it with a pointing device
We can set the name of an Object by setObjectName() and then passing the name as a string within the empty parentheses.
To access a specific object we can call for the main class and then call for the sub class Like: QPushButton#button1 within a stylesheet

```
import sys
from PyQt5.QtWidgets import (QMainWindow,QApplication, QGridLayout,QPushButton)

class MainWindow(QMainWindow):
        def __init__(self):
            super().__init__()
            self.setGeometry(700,300,250,250)
            self.button1=QPushButton("#1",self)
            self.button2=QPushButton("#2",self)
```

```
        self.button3=QPushButton("#3",self)
        self.button4=QPushButton("#4",self)
        self.initUI()

    def initUI(self):
        self.button1.setObjectName("button1")
        self.button2.setObjectName("button2")
        self.button3.setObjectName("button3")
        self.button4.setObjectName("button4")

        self.button1.setGeometry(0,0,500,500)
        self.button2.setGeometry(0,500,500,500)
        self.button3.setGeometry(500,500,500,500)
        self.button4.setGeometry(500,0,500,500)

        self.setStyleSheet("""
                                QPushButton{
                                font-size : 50 px;
                                font-family : Cursive;
                                margin : 20 px;
                                border : 3px solid;
                                border-radius : 15 px;
                                }
                                QPushButton#button1:hover{
                                background-color : hsv(169, 90%, 51%);
                                }
                                QPushButton#button2:hover{
                                background-color : hsl(354, 77%, 41%);
                                }
                                QPushButton#button3:hover{
                                background-color : hsl(68, 92%, 50%);
                                }
                                QPushButton#button4:hover{
                                background-color : hsl(145, 96%, 43%);
                                }

                                    """)



def main():
    app=QApplication(sys.argv)
    window = MainWindow()    # window initialization
    window.show()    # showing window
    sys.exit(app.exec_())    # waiting until it is closed by user
```

```
if __name__ == "__main__" :
    main()
```

DIGITAL CLOCK [PROGRAM]

The QTime module helps us to get the current time

The ToString use helps us to get the time in a specific format

The QTimer module here helps us to count a timeout such that after sometime the function will be invoked again

** The timeout attribute helps us do that**

like: timer.timeout.connect(self.self_def_func)

timer.start(1000) helps us to give the time after which we want the timeout here after every 1000 mili seconds or 1 second the current time function will be invoked.

```
import sys
from PyQt5.QtWidgets import QWidget, QLabel, QVBoxLayout,QApplication
from PyQt5.QtCore import QTime,QTimer,Qt


class Main_window(QWidget):
    def __init__(self):
        super().__init__()
        self.Clock=QLabel(self)
        self.Timer=QTimer(self)
        self.initUI()

    def initUI(self):
        self.setWindowTitle("Digital clock")
        self.setGeometry(600,400,300,100)
        self.Clock.setStyleSheet("color:red;"
                                 "font-size:150px;"
                                 "font-family:Agency FB"
                                 )
        self.setStyleSheet("background-color:black")
        vbox= QVBoxLayout()
        vbox.addWidget(self.Clock)
        self.setLayout(vbox)
        self.Clock.setAlignment(Qt.AlignCenter)
        self.Timer.timeout.connect(self.current_time)
        self.Timer.start(1000)
```

```
    def current_time(self):
        current_time= QTime.currentTime().toString("h:m:s")
        self.Clock.setText(current_time)

if __name__ == "__main__":
    app= QApplication(sys.argv)
    clock = Main_window()
    clock.show()
    sys.exit(app.exec_())
```

Stopwatch Program

```
import sys
from PyQt5.QtWidgets import QApplication,QWidget,QHBoxLayout,QVBoxLayout,QLabel,QPushButton
from PyQt5.Qt import QTimer,QTime,Qt

class Main_window(QWidget):
    def __init__(self):
        super().__init__()
        self.time=QTime(0,0,0,0)
        self.timer=QTimer(self)
        self.clock=QLabel("00:00:00.00", self)
        self.button_start=QPushButton("Start")
        self.button_stop=QPushButton("Stop")
        self.button_reset=QPushButton("Reset")
        self.initUI()
    def initUI(self):

        self.clock.setStyleSheet("font-weight:bold;"
                                 "font-family:Agency FB;"
                                 "font-size:100px;"
                                 "color:Red")
        self.clock.setAlignment(Qt.AlignCenter)


        self.setStyleSheet("QWidget{"
                           "background-color:white}"
                           "QPushButton{"
                           "color:red;"
                           "font-size: 30 px"
                           "}"
                           )
        vbox=QVBoxLayout()
        vbox.addWidget(self.clock)
```

```
            self.setLayout(vbox)
            hbox=QHBoxLayout()
            hbox.addWidget(self.button_stop)
            hbox.addWidget(self.button_reset)
            hbox.addWidget(self.button_start)
            vbox.addLayout(hbox)
            self.button_start.clicked.connect(self.start)
            self.button_stop.clicked.connect(self.stop)
            self.button_reset.clicked.connect(self.reset)
            self.timer.timeout.connect(self.update_time)
        def start(self):
            self.timer.start(10)
        def stop(self):
            self.timer.stop()
        def reset(self):
            self.timer.stop()
            self.time = QTime(0,0,0,0)
            self.clock.setText("00:00:00:000")
        def update_time(self):
            self.time = self.time.addMSecs(10)
            self.clock.setText(self.format_time(self.time))
        def format_time(self,time):
            hours = time.hour()
            minute = time. minute()
            second = time.second()
            milisecond =time.msec()
            return f"{hours:02}:{minute:02}:{second:02}:{milisecond:02}"
if __name__ == '__main__':
    app=QApplication(sys.argv)
    Stop_watch= Main_window()
    Stop_watch.show()
    sys.exit(app.exec_())
```

NUMPY

** It is very space sufficient**

USAGE STARTER:

i)Go to the terminal and write jupyter notebook and click enter.

ii)Then it will automatically move us to the website to jupyter notebook.

iii)Then we will click on new and rename the file as numpy_modules which automatically gives us an ipynb file.

iv)Then open jupyter lab and open the file we just made.

We can get multidimensional arrays:

i) First importing numpy is required.
[we imported numpy as np]

ii) We can make 1D 2D 3D and 4D arrays too. Here, we will need to call the numpy library and call the array function. Then within the empty parentheses, we will call the types of array as we want.
iii) Before that we will need an object for us to store the array in.

For a 1D array: np.array([ ]) # within parentheses, 3rd brackett used once.

For a 2D array: np.array([[1,2,3] # 1 third brackett extra and rest for the arrays
[4,5,6]])
For a 3D array: np.array([[[1,2,3,4] #2 extra and rest for arrays
[5,6,7,8]
[9,10,11,12]]])

```
a=np.array([1,2,3,4,5])
print(a)
b=np.array([[[1,2,3],
            [4,5,6],
            [7,8,9]]])
print(b)
c=np.array([[[[1,2,3],
            [4,5,6],
            [7,8,9],
            [10,11,12]]]])
print(c)
```

We can get size of arrays:

We can print the size of the arrays, with:
Object_name.size

```
5
9
12
```

```
[ ]:
print(a.size)
print(b.size)
print(c.size)
```

We can get shape of arrays:

We can print the shape of arrays with:
Object_name.shape

```
[19]:
print(a.shape)
print(b.shape)
print(c.shape)
```

```
(5,)
(1, 3, 3)
(1, 1, 4, 3)
```

We can get data type of arrays:

We can print the data type of arrays with:
Object_name.dtype

```
[21]:
print(a.dtype)
```

```
int64
```

We can shuffle data all over:

*This is to not overfit the model.

np.random.seed()

linspace() :

* A function which **generates evenly spaced numbers** over a specific **interval**.

SYNTAX

numpy.linspace(**start**, **stop**, **num**=50, **endpoint**=True, **retstep**=False, **dtype**=None, **axis**=0)

start [**The starting value of the sequence**]
stop [The **end value** (**included** if **endpoint=True**).]
num [**Number of samples** to **generate** (**default=50**)]
endpoint [If **True (default)**, includes the **stop value**. If **False**, excludes it.]
retstep [If True, returns the **step size between values**.]
dtype [**data type** of output]
axis [Axis along which to **store the samples (advanced)**]

USAGE:

i) Basic

If we want an array from a specific point to another **a specific range** , amount of elements we want can also be included , which gives us an array which has **that many number of elements** in that **range**.

```
[3]: import numpy as np
     arr = np.linspace(0,10,5)
     print(arr)

     [ 0.   2.5  5.   7.5 10. ]
```

ii) Excluding / Including endpoint

We can **generate** either along with endpoint or without it but still we will generate as many elements as per request.

If we want it, **endpoint=True** else **endpoint=False**

```
[5]: import numpy as np
     arr = np.linspace(0,10,5, endpoint=False)
     print(arr)

     [0. 2. 4. 6. 8.]
```

iii) Getting the step size

We can find how much we jumped from previous to the next value. and print it too. Here, we unpack the array and the steps into different variables. So, as **by default retstep is false it isn't unpacked** . So to get steps, we will need to set **retstep=true.**

Set value is calculated **by dividing the number of elements by the number of intervals (num_points - 1)**.

step = number_of_elements / number_of_interval

```
[12]: import numpy as np
      arr, step=np.linspace(0,10,5,retstep=True)
      print(step)
      print(arr)

      2.5
      [ 0.   2.5  5.   7.5 10. ]
```

iv) Setting up of data type

By default, the data return type is a **one decimal placed number**. But if we want to set the numbers as we want. This function is to be used.

```
[13]: import numpy as np
      arr, step=np.linspace(0,10,5,retstep=True,dtype=int)
      print(step)
      print(arr)

      2.5
      [ 0  2  5  7 10]
```

Matrix operations

Addition of arrays:

sum_array = array1+array2
print(sum_array)

Transposing arrays:

1 dimensional arrays cannot be transposed

1) We can transpose using,
array_object.T

[13]:

```
b=np.array([[[1,2,3],
             [4,5,6],
             [7,8,9]]])
print(b)

print(b.T)
```

```
[[[1 2 3]
  [4 5 6]
  [7 8 9]]]
[[[1]
  [4]
  [7]]

 [[2]
  [5]
  [8]]

 [[3]
  [6]
  [9]]]
```

2) We can also transpose using:
transposed_matrix = np.transpose(matrix)

Filling up matrix:

Filling up matrices with the number 0 or 1

1) Filling the array with ones.
2) We can **define the amount of times** we want that specific number within the array after calling the number and within the empty parentheses putting in a specific number.

array_name = np.**ones(rows)**

This will take 1 the row of times we wanted, BUT we willl get float type 1 inside the array by default.

[7]:

```
c = np.ones(6)
c
```

[7]:

```
array([1., 1., 1., 1., 1., 1.])
```

To get integer type array:

** After entering the number, within the empty parentheses we will take the type with the help of dtype, where dtype = data_type_required

array_name = np.(num, dtype= type_we_want)

[8]:

```
c = np.ones(6, dtype=int)
c
```

[8]:

```
array([1, 1, 1, 1, 1, 1])
```

WE CAN ALSO DO THE SAME WITH **0** with

np.zeros(num, dtype=....)

[12]:

```
c = np.zeros(6, dtype=int)
c
```

[12]:

```
array([0, 0, 0, 0, 0, 0])
```

**** If we want multidimensional arrays of 0 or 1, then we can write:**

c= np.zeros((row ,column), dtype=int)

```
c = np.zeros((6,2),dtype=int)
c
```

```
[14]:

array([[0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0],
       [0, 0]])
```

Filling up with other numbers:

For filling up with other numbers, we use np.full. Like:

ar = np.full((dimension1, dimension2) , number_we_want)

For example:

```
import numpy as np
# Fill an array with the number 7
array_with_sevens = np.full((3, 3), 7)
print(array_with_sevens)
# Fill an array with the number 42
array_with_forty_two = np.full((2, 4), 42)
print(array_with_forty_two)
```

```
D:\PythonProject\.ve
[[7 7 7]
 [7 7 7]
 [7 7 7]]
[[42 42 42 42]
 [42 42 42 42]]
```

Filling up sequentially

We can fill up numbers sequentially with a One dimensional array with all the number that we want by writing:

a= np.arange(From_num , (To_num))

Yes, arrange with **one r less**. It was **not a typo**.
But this always ends at **1 number less than the final number given.**

Example:

```
[16]: c = np.arange(1,20)
      c
```

```
[16]: array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
             18, 19])
```

SKIPPI

NG SEQUENCE

To only print even number or odd number, we might need to skip some numbers in the middle. What we can write for this is:

np.arange(**From_num**, **To_num**, **skip_sequence_num**)

For example:

Even number printup:

```
]: c = np.arange(2,21, 2)
   c
```

```
]: array([ 2,  4,  6,  8, 10, 12, 14, 16, 18, 20])
```

Odd number printup:

```
[18]: c = np.arange(1,21, 2)
      c
```

```
[18]: array([ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19])
```

Both of these cases skip 2 number forward moving sequentially

Reshaping Array:

We can reshape an array after filling it up too, or giving it random numbers too by writing :

c= c.**reshape**(**rows**, **columns**)

One thing should always be kept in mind while reshaping arrays that we need to reshape the arrays in such a way that : **row * column = element_number else we will find error while running it **

For Example:

1) row * column == **element number**
   1-21 elements in odd numbers are 10
   2-21 elements in even number are 10

```
[20]: c = np.arange(1,21, 2)
      c = c.reshape(2,5)
      c
```

```
[20]: array([[ 1,  3,  5,  7,  9],
             [11, 13, 15, 17, 19]])
```

```
[20]: c = np.arange(1,21, 2)
      c = c.reshape(2,5)
      c
```

```
[20]: array([[ 1,  3,  5,  7,  9],
             [11, 13, 15, 17, 19]])
```

2) If **number of elements** are **prime number**, then **row column can only be**:

1 * prime_number = prime_number

For example:

Here, 19 elements present so
reshape can be only (1,19) or (19,1)

```
a=np.arange(0,19)
a=a.reshape(19,1)
b=np.arange(20,39)
b=b.reshape(19,1)

print(a+b)
```

```
[[20]
 [22]
 [24]
 [26]
 [28]
 [30]
 [32]
 [34]
 [36]
 [38]
 [40]
 [42]
 [44]
 [46]
 [48]
 [50]
 [52]
 [54]
 [56]]
```

PANDA

Pandas is an open-source library built on top of NumPy, providing data structures like **DataFrames** and **Series** that are ideal for **handling** and **analyzing structured data**.

*It's commonly used in **data science**, **data cleaning**, and **preprocessing tasks**.
*It brings **any sort of data in csv** format **mainly**.

Making a DataFrame:

*Using dataframe we can create a **table** ourselves.

For example:
*We will get a table, where 2 is from "0" and 4 is from "1".
*Same way we will get 4 for "str1" and 5 for "str2".
*Both of the data under the str1 and str2

pd.DataFrame({'str1':[2,4], 'str2':[4,5]})

	str1	str2
0	2	4
1	4	5

Making a Series:

*Using this we can get a table too. But the elements will be shown alongside their indices when we pass in an array inside the Series function.

* We can also get the data type of the array too.

For example:

pd.Series([1,2,3,4,5])

```
[40]:
pd.Series([1,2,3,4,5])

[40]:
0    1
1    2
2    3
3    4
4    5
dtype: int64
```

Getting info() from a CSV file:

Using the info() function, we can get the summary of the total dataset about the :
i) Datatype
ii) count of the non null values
iii) column

1) To get info from a csv, you will need to at first read the csv file and store it in a variable.

var_name = pd.read_csv("File_name")

2) Then we will need to write this to get the info from the csv file using **panda library**.

var_name.info()

FOR EXAMPLE:

```
housing=pd.read_csv("Housing.csv")
```

```
housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 545 entries, 0 to 544
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   price             545 non-null    int64 
 1   area              545 non-null    int64 
 2   bedrooms          545 non-null    int64 
 3   bathrooms         545 non-null    int64 
 4   stories           545 non-null    int64 
 5   mainroad          545 non-null    object
 6   guestroom         545 non-null    object
 7   basement          545 non-null    object
 8   hotwaterheating   545 non-null    object
 9   airconditioning   545 non-null    object
 10  parking           545 non-null    int64 
 11  prefarea          545 non-null    object
 12  furnishingstatus  545 non-null    object
dtypes: int64(6), object(7)
memory usage: 55.5+ KB
```

Describe()

Same process to save the csv file within an object and then describing it will give us overall analysis of the csv file.

[44]:
```
housing.describe()
```

[44]:

	price	area	bedrooms	bathrooms	stories	parking
count	5.450000e+02	545.000000	545.000000	545.000000	545.000000	545.000000
mean	4.766729e+06	5150.541284	2.965138	1.286239	1.805505	0.693578
std	1.870440e+06	2170.141023	0.738064	0.502470	0.867492	0.861586
min	1.750000e+06	1650.000000	1.000000	1.000000	1.000000	0.000000
25%	3.430000e+06	3600.000000	2.000000	1.000000	1.000000	0.000000
50%	4.340000e+06	4600.000000	3.000000	1.000000	2.000000	0.000000
75%	5.740000e+06	6360.000000	3.000000	2.000000	2.000000	1.000000
max	1.330000e+07	16200.000000	6.000000	4.000000	4.000000	3.000000

EQUAL DISTIBUTION of data

head()

* By default shows the **first 5 rows of a DataFrame.**
* head(n) shows the info on **n rows**.

USEFULNESS:

* Helps in quickly inspecting the structure of the data.
* Useful for debugging and data exploration.

```
#understand the data
data.head()  #for understanding the data
```

[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction
0	6	148	72	35	0	33.6	0.627
1	1	85	66	29	0	26.6	0.351
2	8	183	64	0	0	23.3	0.672
3	1	89	66	23	94	28.1	0.167
4	0	137	40	35	168	43.1	2.288

unique()

* Returns **unique values in a column**.
* To find the number of unique values in different cases
* Some datas having **2 unqiue values** are **binary classifications**
* Others having more than that are **multiclass variations**

[16]:

```
data['Age'].unique() #Will show what the Age data contains
                     # How many types of age it contains will be shown here
                     # SHORTLY ALL UNIQUE VALUES WILL BE ON DISPLAY
```

[16]:

```
array([50, 31, 32, 21, 33, 30, 26, 29, 53, 54, 34, 57, 59, 51, 27, 41, 43,
       22, 38, 60, 28, 45, 35, 46, 56, 37, 48, 40, 25, 24, 58, 42, 44, 39,
       36, 23, 61, 69, 62, 55, 65, 47, 52, 66, 49, 63, 67, 72, 81, 64, 70,
       68])
```

isnull().sum()

*Counts the **missing values per column**.
* Returns boolean of the missing columns

```
data.isnull().sum()     # it will return a boolean value
                        # The number of null values of all classes will be shown
```

[18]:

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

drop()

* Removes the row or column name from a DataFrame

```
new_data = data.drop(['BMI','Pregnancies'],axis=1)
```

corr()

* Computes pairwise correlation between numeric columns.

```
corelation = new_data.corr()
```

MATPLOTLIB

What is **matplot?**

Matplot is a python library that helps us to plot data.
*The easiest and most basic plots are line, scatter and histogram plots.

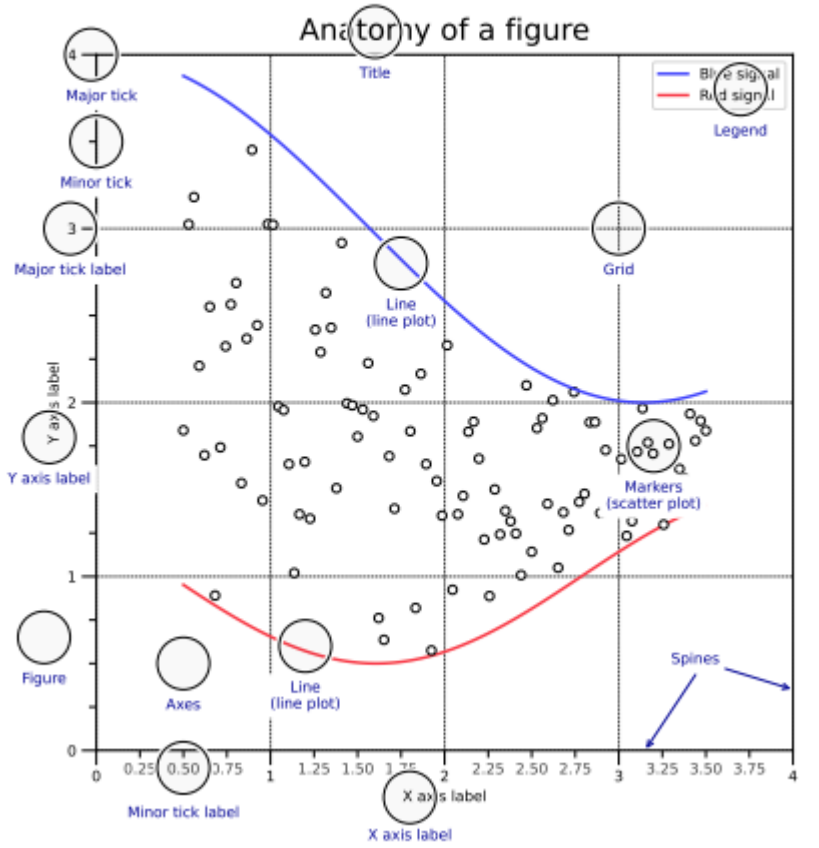
The most used and common types of plots are:

***Line Plot** is better when **x axis is time**.
***Scatter** is better when there is a **correlation between two variables**.
***HIstogram** is better when we need to **see distribution of numerical data**.
*Customization: i) Color labels,
ii) Thickness of line,
iii) the opacity,
iv) grid,
v) figsize,
vi) ticks of axes and
vii) linestyle.

A list of the types are given below:

Kind	Description
`'line'`	Line plot (default for time series or continuous data).
`'bar'`	Vertical bar plot.
`'barh'`	Horizontal bar plot.
`'hist'`	Histogram.
`'box'`	Box plot for statistical distribution.
`'kde'` or `'density'`	Kernel Density Estimate plot (smooth distribution curve).
`'area'`	Area plot (stacked by default).
`'pie'`	Pie chart.
`'scatter'`	Scatter plot (requires specifying `x` and `y`).
`'hexbin'`	Hexbin plot (for bivariate data density).

Anatomy of a figure



Plot Usage:

We can give the types of plotting we want within the parameter with after calling for the object:

kind = "line" / "hist" / "scatter"

Or we can also bring the name after the object name without using the plot function:

plt.**hist()** / plt.**line()**

Kind	Description
`'line'`	Line plot (default for time series or continuous data).
`'bar'`	Vertical bar plot.
`'barh'`	Horizontal bar plot.
`'hist'`	Histogram.
`'box'`	Box plot for statistical distribution.
`'kde'` or `'density'`	Kernel Density Estimate plot (smooth distribution curve).
`'area'`	Area plot (stacked by default).
`'pie'`	Pie chart.
`'scatter'`	Scatter plot (requires specifying `x` and `y`).
`'hexbin'`	Hexbin plot (for bivariate data density).

Point plotting when csv file is not present is also possible:

import matplotlib.pyplot as plt
object.plot(**abscisssa**, **ordinate**)

```
x = [0, 1, 2, 3, 4]
y1 = [0, 1, 4, 9, 16]
plt. plot (x, y1)
```

LINE plotting

*When creating a line plot (e.g., kind='line'), the **x-axis** and **y-axis** depend on **the data structure and parameters** provided.

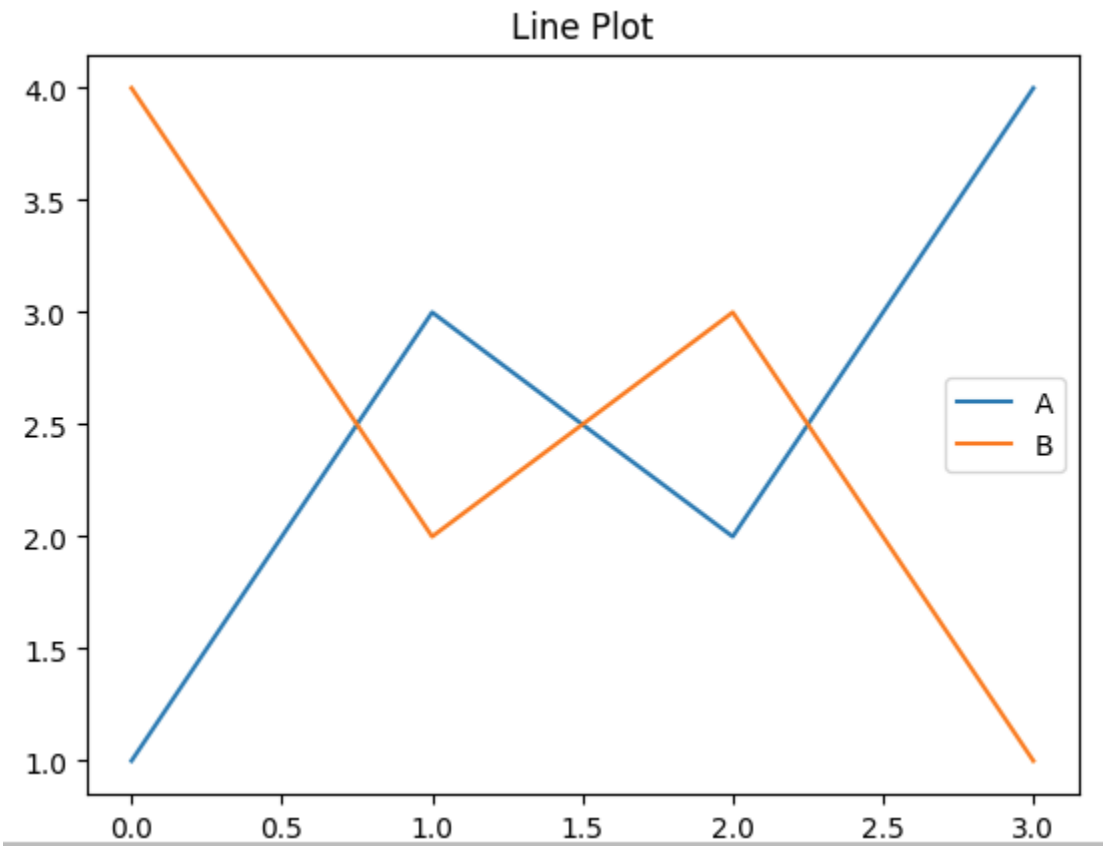
For example:
*Here A & B are plotted on the y-axis and x-axis is by default 0,1,2,3.....

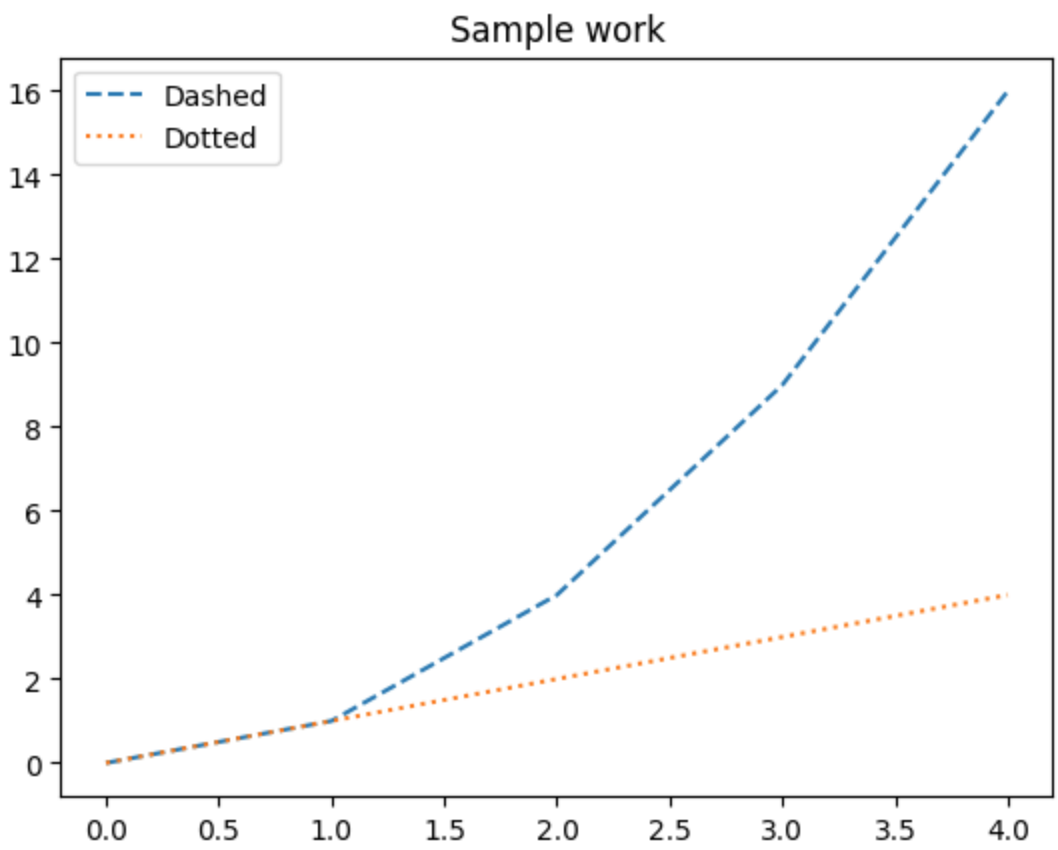
CODE:

```
import pandas as pd

import matplotlib.pyplot as plt              # Example DataFrame
data = pd.DataFrame({
        'A': [1, 3, 2, 4],
        'B': [4, 2, 3, 1]
})                                           # Default behavior
data.plot(kind='line', title='Line Plot')
plt.show()
```

OUTPUT:






```
x = [0, 1, 2, 3, 4]
y1 = [0, 1, 4, 9, 16]
y2 = [0, 1, 2, 3, 4]

# Plot with different linestyles
plt.plot(x, y1, linestyle='--', label='Dashed')
plt.plot(x, y2, linestyle=':', label='Dotted')

# Add legend and show the plot
plt.title("Sample work")
plt.legend()
plt.show()
```

SETTING UP PARAMETER:

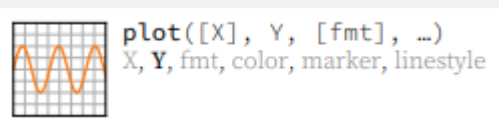
From a csv file after we read it, we will have to store it to make an object of the csv file.

housing= "Housing.csv"

Then, we will need to take a specific column & plot it writing:

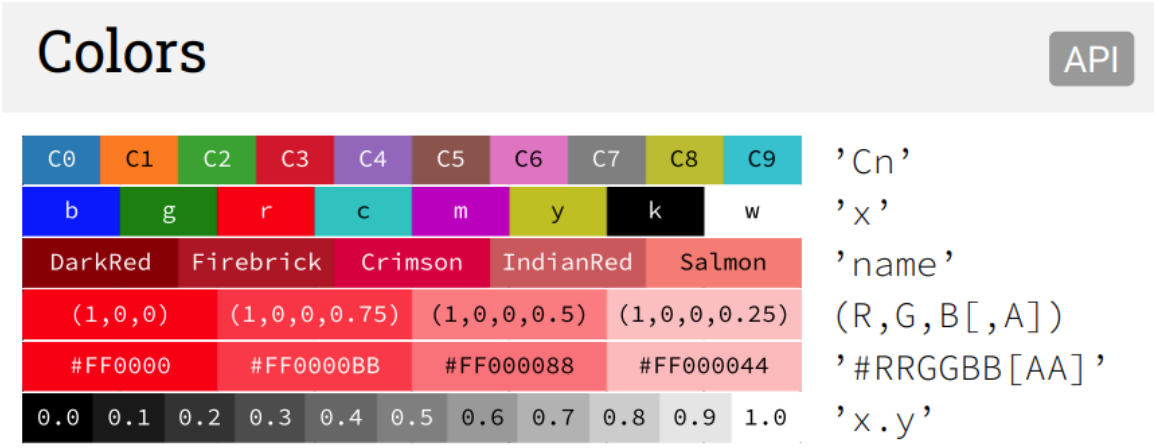
*Here we will take the area column and plot it.
csv_filename . column_name . plot()

Afterwards, we will need to bring in the parameters that we want to apply to the Line plotting.



Label is the name of the line that is plotted the name of which we can set:
label='label_name'

color is the color obviously of the line or something which we can set up.
color='g' # this means green



linewidth sets the width of the line.
linewidth=1

alpha sets the opacity of the line
aplha= 0-->1 (ex: .75)

grid sets if the the plot will be within a grid: [it only takes boolean entries]
grid = True / False

linestyle sets the style of the line:

linestyle = ':'

'-' or 'solid' are the same. [solid line **default**]
'--' or 'dashed' are the same. [dashed line]
'-.' or 'dashdot' are the same. [dash dot line]
':' or 'dotted' are the same. [dotted line]

figsize is the shaper of the figure of the plotting which has two parameters
Length X Breadth
We can write this like:

figsize(**length**, **Breadth**)

LEGEND()

What is **Legend?**

A legend in a plot is a small box or section that provides context about the visual elements of the graph. It explains what each line, marker, or color in the plot represents by associating them with a label.

The Label names will be displayed with the help of this function. By default it is shown in the left upper right corner.

Now, the localization of the legends can be set as we want:

WIth direction:

object. legend(loc="upper right")
object. legend(loc="upper left")
object. legend(loc="lower right/left")
object. legend(loc="right")
object. legend(loc="center")
object. legend(loc="center right/left")
object.legend(loc="lower/upper center")
object.legend(loc="best") **#automatically sets the best position for the legend.**

WIth pixel: #for fine tuning

object. legend(loc=10)

xlabel / ylabel / main [name setup]:

* To set up the **name of the label** in **x** or **y axis** we will need to write:

object.**xlabel**("x_axis_name")

object.**ylabel**("y_axis_name")

* We can also set up the **main label name** which will appear at the top of the graph plot:

object.**title**("Main_label_name")

*We can set the parameter within the plot() function too. Here:

object.plot(kind="Whatever",xlabel="x_axis_name", ylabel="y_axis_name", title="Main_label_name")

Scatter Plot

Using a csv file:

1) We will need to get the csv file to be read by the help of the pandas library:

import pandas as pd

2) We will then need to save the Csv file after reading it to a variable:

attribute_var = pd.**read_csv(** "File_name" **)**

3) Then we will need to plot the points as scatter using **plot** [might as well add more parameters]:

#WE WILL NEED TO SPECIFY WHICH AXIS HAS WHAT

attribute_var . plot (kind=scatter, **x="area"**, **y="price"**, xlabel="X-axis",ylabel="y-axis")

4) Using pyplot, we can also add the parameters individually: [**better practice**]

These should be mentioned after the plotting or .plot() function

import matplotlib.pyplot as pt

pt.title("Title_name")

IGNORING ERRORS

1) Import the warnings library:

import warnings

2) To ignore the warnings that are given, we will have to call the **filterwarnings()** function and pass in **"ignore"** string as command.

warnings**.filterwarnings("ignore")**

Plotting Histogram:

Bins [exclusive to hist]

Bins refer to **the intervals** into which **data points are grouped** when creating a histogram.

*Each bin **represents a range of values**,

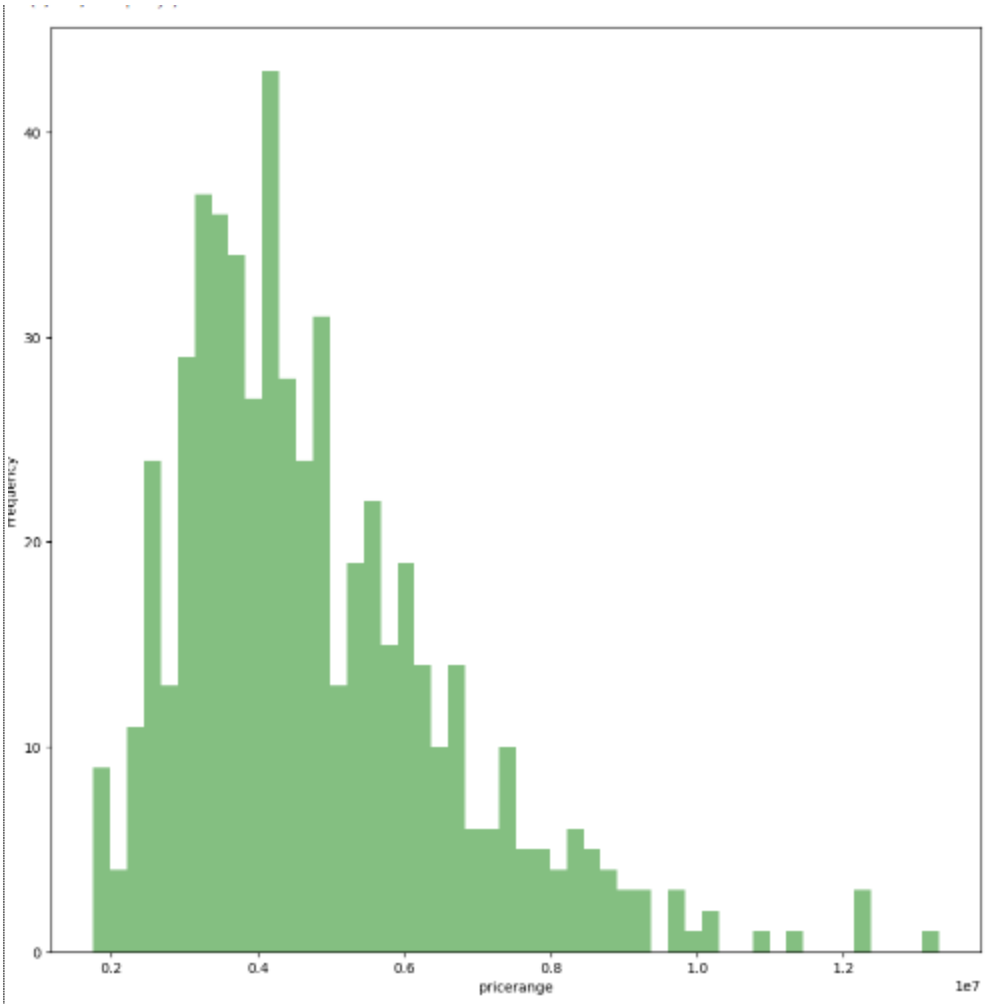
*and the **height of the bar** in the histogram corresponds to the **number of data points** that **fall into that range**.

Steps to plotting a histogram:

1) Definitely read the csv file at first and store it in an object.
2) Here, one of type of data having both the Frequency and the range need to be selected.
3) They need to be plotted and shown afterwards.

FULL INSTANCE

```
import matplotlib.pyplot as pt
import pandas as pd
import warnings
warnings.filterwarnings("ignore")
housing=pd.read_csv("housing.csv")
housing.price.plot(kind="hist",alpha=0.5,color="g",bins=50,figsize=(12,12))
pt.xlabel("pricerange")
pt.ylabel("Frequency")
```



SEABORN

* Used for statistical data visualization

sns.heatmap()

* Plots the correlation matrix as a coloured grid
* Relation of each with each

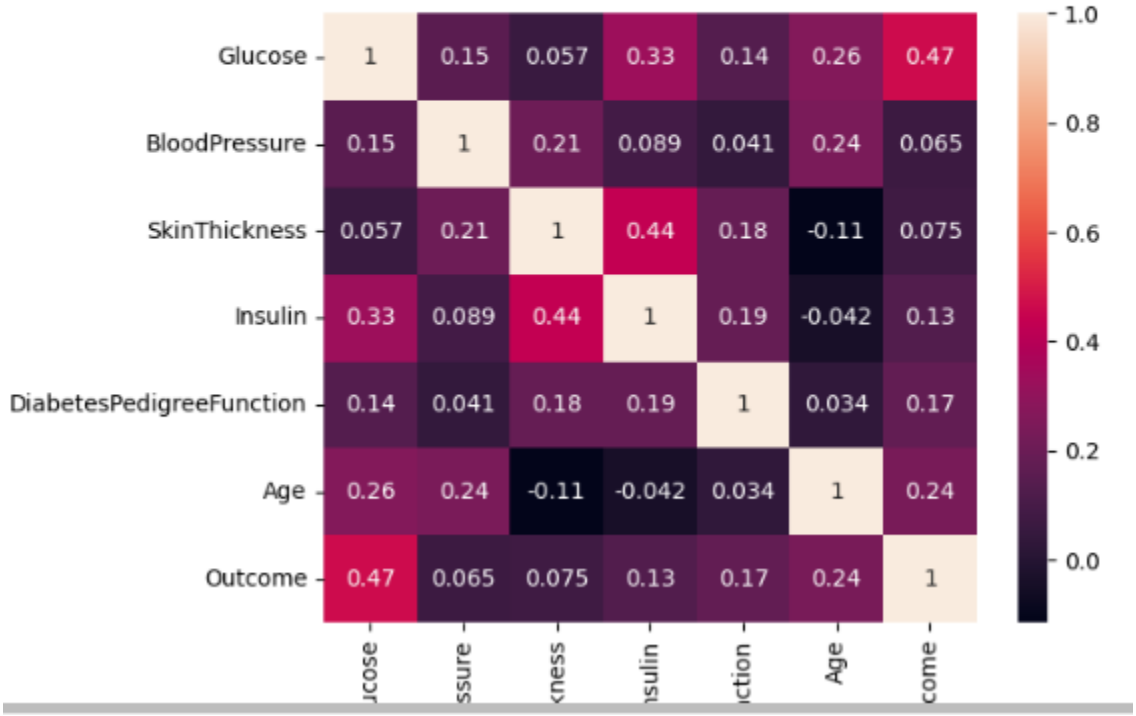
Parameters

* **xticklabel, yticklabel** are used to **customize axis label**. (True/False) (hide/show x/y label)
* **annot** = True/False for displaying value
* **fmt** = string **formatting** for **annotations**
* **cmap** = Color map ["Blues","coolworm","virdis"]
* **linecolor** = Color of lines between cells
* **linewidths** = widths of lines between cells
* **cbar** = show/hide color bars (True/False)
* **vmin/vmax** = min/max values for **color scaling [scaling of values]**

```
corelation = new_data.corr()
sns.heatmap(corelation, xticklabels=corelation.columns, yticklabels=corelation.columns
```

[51]:

<Axes: >



sns.pairplot()

* Scatter plots **for all numeric columns** (with histograms on diagonal).
* Creates a grid of scatter plots (for numeric columns) and histograms/KDEs (diagonal) to visualize pairwise relationships.

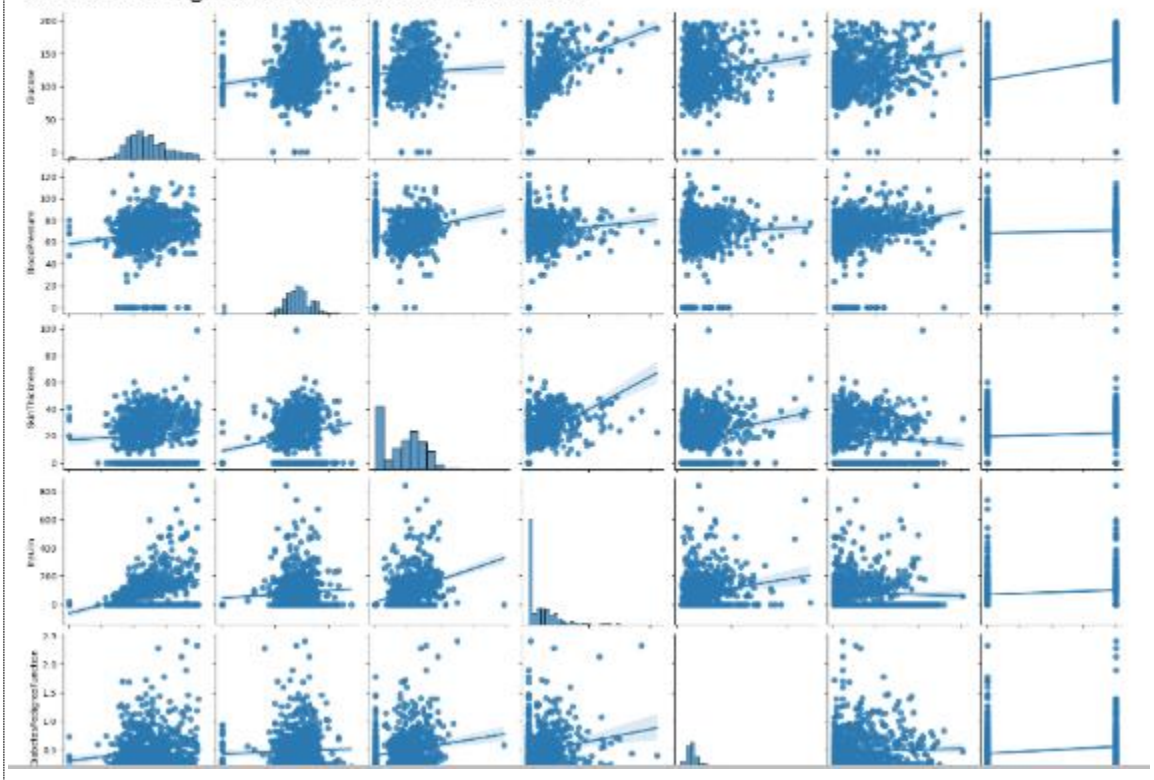
Parameters:

* data [**dataframe**]
* hue [Group by a **catagorical column** (adds color encoding)]
* vars [List of **column names to plot** (default: all numeric columns)]
* x_vars / y_vars [specify **columns** for **grid rows/columns seperately**]
* kind [plot type for **non-diagonal elements**] [default: 'scatter' or 'reg' regression]
* diag_kind [Plot type for diagonal 'hist'(default) or 'kde']
* palette [color scheme]
* markers [symbols for different **hue groups**]
* plot_kws [additional argument for non-diagonal plots e.g:{'alpha':0.5}]
* diag_kws [additional argument for diagonal plots e.g:{'edgecolor':'black'}]

```
sns.pairplot(new_data,kind='reg')   #pair wise plotting will be done
                                    # scatter plot with each info with each
```

[55]:

```
<seaborn.axisgrid.PairGrid at 0x284e5c54d60>
```



sns.relplot()

* Flexible interface relational plot.

parameter:

* data (dataframe)
* x,y (Column names for x/y axes)
* hue (Group by a catagorical column (color encoding))
* size (Vary point sizes by a numeric column)
* style (Vary point markers by a categorical column)
* col/row (Create faceted subplots based on a categorical column)
* kind ('scatter' / 'line')
* palette (color scheme)
* height / aspect (control figure size e.g height=4, aspect=1.5)
* facet_kws (Additional arguments for facetting e.g {'sharley' : False})