1. BASIC DECLARATION :

In [1]:
```
message = "Hello Pythan"
```

In [2]:
```
print(message)
```

```
Hello Pythan
```

In [3]:
```
# Whenever we declare Single quote '', or Double quotes "", it is going to be printed as it is.
print("message")
```

```
message
```

2. DECLARING & COMPARING :

In [4]:
```
# DECLARING:
Name = "prasanth"
# COMPARING:
Name == "prasanth"
```

Out[4]:
```
True
```

In [5]:
```
# DECLARING:
Name = "pRasanth"  #CAUSE : WE DECLARED CAPITAL -'R' IN THE MIDDLE OF THE NAME, THAT'S WHY IT IS 'FALSE'.
# COMPARING:
Name == "prasanth"
```

Out[5]:
```
False
```

3. TITLE METHOD()

In [6]:
```
name = "jagadeesh prasanth"
```

In [7]:
```
print(name.title())
```

```
Jagadeesh Prasanth
```

4. UPPER METHOD()

In [8]:
```
print(name.upper())
```

```
JAGADEESH PRASANTH
```

5. LOWER METHOD()

In [9]:
```
print(name.lower())
```

```
jagadeesh prasanth
```

6. COMBINING / CONCATINATING : Str (String data):

In [10]:
```
first_name = "jagadeesh"
last_name = "prasanth"
full_name = first_name+ "     " +last_name
print(full_name)
# GIVING '+' --> FOR CONCATINATING,   DOUBLE QUATES "" --> FOR ADDING SPACE BETWEEN FIRST AND LAST NAME.
```

```
jagadeesh     prasanth
```

7. USING : (\t)TAB SPACE and (\n)NEXT LINE :

In [11]:
```
print("prasanth".title())         # USING .title()
print("***************")
print("\tprasanth")               # \t TAB SPACE
print("***************")
print("\nprasanth".upper())       # \n NEXT LINE, .upper()
```

```
Prasanth
***************
        prasanth
***************

PRASANTH
```

8. HANDLING NUMBERS :

In [12]:
```
num1 = 10
num2 = 30
```

In [13]:
```
# int data type :
type(num1)
```

Out[13]:
```
int
```

In [14]:
```
# DOING SUMMATION :
num1 + num2
```

Out[14]:
```
40
```

9. FLOAT :

In [15]:
```
num3 = 3.5
```

In [16]:
```
type(num3)
```

Out[16]:
```
float
```

10. CALCULATIONS :

In [17]:
```
5/4
```

Out[17]:
```
1.25
```

In [18]:
```
2*5
```

Out[18]:
```
10
```

In [19]:
```
# THE DIRECT CALCULATION WILL BE CALCULATED VERY FAST :
3+6-98*69/654-546+456
```

Out[19]:
```
-91.33944954128435
```

11. TRUE OR FALSE : BULLIAN PATTERN :

In [20]:
```
# THE OUTPUT WE ARE GETTING IN THE BULLIAN PATTERN :
4 > 5
```

Out[20]:
```
False
```

In [21]:
```
6 < 4
```

Out[21]:
```
False
```

```
In [22]: 6 > 4

Out[22]: True
```

12. CONVERT DATA from FLOAT to INTEGER :

```
In [23]: num4 = 3.6
```

```
In [24]: type(num4)

Out[24]: float
```

```
In [25]: # NOW I WANT TO CONVERT DATA FROM FLOAT TO INTEGER :
         int(num4)  # NUMBER AFTER POINT WILL NOT BE SHOWED.

Out[25]: 3
```

13. IDENTIFING THE MINIMUM VALUE IN BETWEEN :

```
In [26]: min(43,23,54,24,98,45,33,90,43)   # Here number '23' is the Minimum Value.

Out[26]: 23
```

14. IDENTIFING THE MAXIMUM VALUE IN BETWEEN :

```
In [27]: max(43,23,54,24,98,45,33,90,43)   # Here number '98' is the MAXIMUM Value.

Out[27]: 98
```

15. ABSOLUTE NUMBER :

```
In [28]: # FROM NEGATIVE TO POSITIVE :
         abs(-95.63)

Out[28]: 95.63
```

16. LOGICAL OPERATOR :

```
In [29]: # COMPARING :
         1 == 2

Out[29]: False
```

```
In [30]: # DECLARING :
         x = 2
```

```
In [31]: # or else this particular logical operator can also applu on the String :
         "_python_" == "_python"   # '==' COMPARING

Out[31]: False
```

```
In [32]: "PRASANTH" == "PRASANTH"

Out[32]: True
```

In [33]:
```
# IT WILL THROW ERROR, IF SINGLE OR DOUBLE QUOTES ARE NOT MENTIONED :
PRASANTH == PRASANTH
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_11260\2595558216.py in <module>
      1 # IT WILL THROW ERROR, IF SINGLE OR DOUBLE QUOTES ARE NOT MENTIONED :
----> 2 PRASANTH == PRASANTH

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

In [ ]:
```
'PRASANTH' == 'PRASANTH'
```

17. NUMERIC WITH STRING DATA : STRING METHOD()

In [ ]:
```
age = 22
```

In [ ]:
```
# IT THROWS ERROR BECAUSE, 'AGE' IS 'INT DATA', WE NEED TO CONVERT 'AGE' IN TO 'STRING DATA' :
msg = "Happy" + age + "nd Birthday"
```

In [ ]:
```
# CONVERTING 'AGE' TO 'STRING DATA' :
msg = "Happy   " + str(age) + "nd Birthday"  # GIVING SPACE INSIDE THE DOUBLE QUOTES " ".
print(msg)
```

COMPLEX DATA TYPES :

In [ ]:
```
#A . LIST,
#B . TUPLE,
#C . DICTIONARY - DICT,
#D . SET,
#E . FROZEN SET.
```

18. PROPERTIES / CHARECTERISTIC OF A LIST :

In [ ]:
```
#LIST SUPPORT :
# ADDING OBJECT TO LIST,
# DELETE OBJECT FROM LIST,
# REMOVE OBJECT FROM LIST,
# APPEND OBJECT TO LIST.
```

In [ ]:
```
lst = ["Hi",25,46,"Happy"]  # can have Heterogenious items and unmutable(Editable).
```

In [ ]:
```
type(lst)
```

In [ ]:
```
print(lst)
```

In [ ]:
```
lst
```

In [ ]:
```
# Whatever objects declaring in SQURE BRASSES[], Considered as an a LIST OF OBJECTS :
```

In [ ]:
```
x = [1,"Happy",456,"Jump",56]
```

In [ ]:
```
type(x)
```

```
In [ ]: states = ['hyderabad','chennai','mumbai','kolkata','pune','vizag']
```

```
In [ ]: states
```

```
In [ ]: # NOW PRINTING '0' POSITION OF STATES : IN LIST THE INDEX STARTS FROM '0' : LIKE 0 1 2 3 4 5
        print(states[0])
```

```
In [ ]: # NOW PRINTING '-1' POSITION OF STATES : IN LIST THE REVERSE INDEX STARTS FROM '-1' : LIKE -6 -5 -4 -3 -2 -1
        print(states[-1])
```

```
In [ ]: print(states[-6])
```

```
In [ ]: # TITLE METHOD( ) IN LIST :          # TITLE METHOD USED TO CONVERT FIRST CHARACTER IN THE STRING.
        print(states[1].title())
```

```
In [ ]: # UPPER METHOD( ) IN LIST :
        print(states[5].upper())
```

```
In [ ]: # APPEND ON LIST : # TO ADD ADDITIONAL ITEMS TO THE LIST:
        states.append("kia")
```

```
In [ ]: states   # 'kia' ADDED TO THE LIST : APPEND OBJECT WILL BE ADDED IN THE 'LAST' :
```

19. LIST : INSERTING OBJECT ON LIST BASED ON INDEX POSITION :

In [ ]:
```
# AT '1' POSITION I AM GOING TO ADD '5 STAR' :
type(states)
```

In [ ]:
```
states.insert(7,"5 star")   # '5 star' added in '1' position and chennai will be removed. Because we added object,
                            #  exactly based on the position.
states
```

20. LIST : del (DELETE) OBJECT FROM THE LIST : BASED ON INDEX POSITION :

In [ ]:
```
states
```

In [ ]:
```
del states[1]     # Means '5 star' in the first position has been removed. (here we mentioned the position).
states            # del used with index number: to remove
```

21. LIST : 'REMOVE' OBJECT FROM THE LIST : BASED ON KEYWORD :

In [ ]:
```
states.remove("pune")  # Means 'kia' has been removed from the list. (here we mentioned the keyword 'kia').
states                 # we use it straight string:
```

22. LIST : ORGANISING THE LIST : ALPHABETICAL ARRANGEMENT :

In [ ]:
```
states
```

In [ ]:
```
states.sort()
print(states)
```

23. LIST : REVERSE SORTING :

In [ ]:
```
# WE CONSIDER THIS AS AN A DESCENDING ORDER :
```

In [ ]:
```
states.sort(reverse = True)  # while mentioning 'True', first letter 'T' must be in 'CAPITAL'.
states
```

24. LIST : HOW MANY OBJECTS IN THE LIST :

In [ ]:
```
len(states)
```

25. LIST : SLICING TECHNIQUE(TENQ) ON THE LIST :

In [ ]:
```
# Whenever we give ':' COLON inside, It will print all the objects :
print(states[:])
```

In [ ]:
```
print(states)
```

In [ ]:
```
# without ':' COLON inside, It will throw error :
print(states[])
```

26. LIST : SLICING TENQ : AFTER ' : ' COLON (SELECT FIRST ONLY OBJECTS) :

In [ ]:
```
states
```

In [ ]:
```
print(states[:3])   # AFTER [:3]  Only first '3' objects selected : # first colon: first objects
```

27. LIST : SLICING TENQ : BEFORE ' : ' COLON (SKIP OBJECTS) :

In [ ]:
```
states
```

In [ ]:
```
print(states[3:])  # BEFORE [3:]  Skip first '3' Objects : after colon : last objects
```

In [ ]:
```
# Some examples :
```

In [ ]:
```
states
```

In [ ]:
```
print(states[:2])  # Selected 1st 5 objects from the list 'states':
```

In [ ]:
```
print(states[2:])  # Skip 1st 5 objects from the list 'states':
```

In [ ]:
```
print(states[6:])  # Here all 6 objects have been skipped from the list 'states':
```

In [ ]:
```
# After : select '5'  and  Before : skip '2'
```

In [ ]:
```
print(states[2:5])   # Here [:5] selected first 5 objects and [2:] skipped first two objects :
```

In [ ]:
```
print(states[4:6])
```

In [ ]:
```
print(states[4:4])
```

In [ ]:
```
print(states[:])   # All objects printed :
```

In [ ]:
```
print(states[5:1])
```

In [ ]:
```
# COPY NUMBER OF OBJJECTS TO NEW LIST :
sts = states[:4]
sts
```

In [ ]:
```

```

LOOPS :

In [ ]:
```
# A. LOOPS are Important Objects or components in all Traditional and Python Languages.
# B. LOOPS Help in Building Logic very flexible and Compatible.
# c. We have an a different Kinds of LOOPS, Which includes CONDITIONAL LOOPS : 1. for loop, 2. if loop, 3. if else,
#    4. if elif else, 5. while loop, 6. while else.
# NESTED LOOPS : 1. for if else, 2. while for.
# These particular LOOPS Helps in Building all the Software Developing Programs or Logics with flexibility.
# The LOOP is going to be repeated based on your 'Index Length'.
```

In [ ]:
```
x = [1,"happy",34,85,"bye"]
for i in x:
    print(i)   # Here we will get INDENTATION SPACE(5 SPACES) before print().
```

In [ ]:
```
# SOME EXAMPLES :
```

In [ ]:
```
mx = [5,3,4,5,6,7,8,912,3,45,78,90]    # Index : 0 t0 11 (total 12 Objects) # 'i' index incremental for the list:
```

In [ ]:
```
for i in mx:
    print(i)      # At 1st object(prints '0' position), again goes to 'i'.
```

28. LOOPS : SORTED METHOD() in for loop: ASCENDING ORDER :

In [ ]:
```
for item in sorted(mx):
    print(item)
```

29. LOOPS : BASIC ADDITIONS AND APPEND OPERATIONS USING " for loop" :

In [ ]:
```
# Now using 'list' and 'range' method(): here khow we append the empty list
```

In [ ]:
```
lt = []
for x in range(5):   # range(0 to 5) means, 0 to 5 it is going to be added.
    lt.append(x)     # adding list item :
    print(lt)        # Here print() is in 'inside loop', it will print for each iterarion.
```

In [ ]:
```
lt = []
for x in range(5):
    lt.append(x)
print(lt)          # Here print() is in 'Outside loop', After completing all iterations, print() will print.
```

30. LOOPS : REVERSE THE LOOP :

In [ ]:
```
for i in reversed(mx):
    print(i)
```

31. LOOPS : APPENDING(adding) THE OBJECT :

In [ ]:
```
# Here this is a Lengthy Loop :
lt = []
for i in mx:
    x = i + 2
    lt.append(x)
    print(lt)
```

In [ ]:
```
# Here print() is in 'Outside Loop' and The 'iteration' we are appending to an a 'lt'(empty list) :
lt = []
print(mx)
print("**************************************************")
for i in mx:
    x = i + 2    # adding increament to the items
    lt.append(x)
print(lt)
```

32. zip method() using in ' for loop ' :

In [ ]:
```
names = ['alice','bob','trudy']
hobbies = ['singing','painting','hacking']
ages = [21,26,35]
for person,age,hobby in zip(names,ages,hobbies):
    print(person+ " is " + str(age)+"years old and hobby is "+hobby)
```

33. LOOP : GENERATE THE NUMBERS :

In [ ]:
```
# if i want to print up to '10', then i need to give '11' :
nbrs = list(range(1,9))          # Here range ( starting number + n -1 )
print(nbrs)
```

34. LOOPS : EVEN NUMBERS :

In [ ]:
```
even_numbers = list(range(2,11,2))
print(even_numbers)
```

35. LOOPS : GENERATE NUMBERS USING range() in list output (or) list pattern :

In [ ]:
```
even_numbers = list(range(2,21,2)) # range '2' to '21' with the step of '2'(2,21,2).
print(even_numbers)
```

36. LOOPS : for loop in depth :

In [ ]:
```
sq = []
for val in range(1,11):
    ddr = val * 2
    sq.append(ddr)
print(sq)
```

In [ ]:
```

```

STRING FORMATTING :

```
In [ ]: # DECLARING STRING OBJECTS BASED ON INDEX
        # INJECTING THE VARIABLE IN TO YOUR STATEMENT (or) STRING
        # STRING FORMAT CAN BE DECLARED AS :   .format() (or) f-string()
```

```
In [ ]: hello = "prasanth"
```

```
In [ ]: hello
```

```
In [ ]: print("Hi How are you "+hello)
```

37. UTILISING STRING FORMAT :

```
In [ ]: print("This is a string{}".format(' APPLE'))   # I gave 'space' before the APPLE to give 'space' between string and ap
```

EXAMPLES ON STRING FORMAT :

```
In [ ]: print('The {} {} {}'.format('fox','brown','quick'))  # Here the positions are 0,1,2. and gave 'space' between strings.
```

```
In [ ]: print('The {0} {0} {0}'.format('fox','brown','quick'))
```

```
In [ ]: print('The {1} {0} {1}'.format('fox','brown','quick'))
```

```
In [ ]: # Here we are 'injecting the words', based on the working :
        print('The {c} {b} {a}'.format(a='fox',b='brown',c='quick'))
```

38. CONDITIONAL LOOPS :

In [ ]:
```
# IF LOOP ENVIRONMENT : if, if else, if elif else, and another - while loop.
# 'if loop' is nothing but - 1. When the Condition satisfies (ENTER THE LOOP),
                            #2. if condition fails (it exit)
# Based on Condition only we are going to enter the loop.
```

Control Flow (or) Control Structure :

In [ ]:
```
# SYNTAX : if condition and statements :
```

In [ ]:
```
# All Data Accepted:
if True:
    print("It's True...!")
```

In [ ]:
```
if 3 > 2:
    print("It's True...!")
```

In [ ]:
```
if 3 < 2:    # Here we don't get any Output, because 3 is not less than 2.
    print("It's True...!")
else:
    print("It's False")
```

39. SINGLE CONDITION :

In [ ]:
```
x = 6
if x > 5:
     print("x is Greater than 5")
```

In [ ]:
```
x = 7
if x > 5:
     print("x is Greater than 5")
print(x * 2)
```

40. DUAL CONDITION :

In [ ]:
```
x = 5
if x > 5:
     print("x is Greater than 5")
     print(x * 2)
else:
    print("x is less than equal to 5")
```

41. elif statement : More than TWO Conditions :

In [ ]:
```
age = 15
if age < 4:
    print("Your age is under 4")
elif age < 18:
    print("Under 18")
else:
    print("valid")
```

In [ ]:
```
x = 2
if x > 5:
    print("x is Greater")
elif x == 5:
    print("x is = 5")
elif x < 5:
    print("x is less than 5")
else:
    print(" Not valid")
```

In [ ]:
```
x = 56.2
if x > 5:
    print("x is Greater")
elif x == 5:
    print("x is = 5")
elif x < 5:
    print("x is less than 5")
else:
    print(" Not valid")
```

42. DUAL CONDITION IN SINGLE Stmt(STATEMENT) : Using 'and' Operator : called as Logical Operators :

In [ ]:
```
x = 5
if x == 5 and type(x) is int:
    print("x is equal to 5 and is integer")
    print(x)
elif x == 10 and type(x) is int:
    print("x is equal to 10 and integer")
    print(x)
```

In [ ]:
```
x = 10
if x == 5 and type(x) is int:
    print("x is equal to 5 and is integer")
    print(x)
elif x == 10 and type(x) is int:
    print("x is equal to 10 and integer")
    print(x)
```

In [ ]:
```
x = 105
if x == 5 and type(x) is int:
    print("x is equal to 5 and is integer")
    print(x)
elif x == 10 and type(x) is int:
    print("x is equal to 10 and integer")
    print(x)
# Here no output, Because 'no condition added' All the given conditions has been failed.
```

43. List data in Conditional Loops :

page 13 blue notebook

In [11]:
```
cars = ["kia","bmw","mercedes","mg","toyota"]
for car in cars:
    if car=="bmw":
        print(car.upper())
else:
    print(car.title())
```

```
BMW
Toyota
```

In [14]:
```
cars
```

Out[14]:
```
['kia', 'bmw', 'mercedes', 'mg', 'toyota']
```

```
In [36]: print(car.title())
```

```
Toyota
```

```
In [16]: cars
         for jelly in cars:
             print(jelly)
```

```
kia
bmw
mercedes
mg
toyota
```

44.WHILE LOOP in CONDITIONAL LOOPS:

page 15 blue notebook

```
In [ ]: #Specially designed for the 'Infinity Declaration'
        #for every 'iteration' Mandatorily we need to declare 'Increment' in the While loop
        #Specially designed to 'Iterate' over a 'Block Code', as long as 'Condition True'
```

In [37]:
```
x = 1
while x <= 10:
    print(x)
    x = x+1
```

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

In [ ]:
```
# if we won't declare particular 'Increment' like 'x = x+1'
# then it will be in the 'loop' of an a 'Infinity loop'
# Sum explanation in page 16 blue notebook
# if --> [*]: Then we have to 'Stop' the loop, otherwise it will run continuesly:
```

In [ ]:

```
x = 1
while x <= 10:
    print(x)
```

```
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
```

In [48]:
```
# cnt in (Infinite loop):
# if --> [*]: Then we have to 'Stop' the loop, otherwise it will run continuesly:

cnt=2
while cnt < 6:
    print(cnt)
    print("This is inside loop")
```

```
2
This is inside loop
2
This is inside loop
2
This is inside loop
2
This is inside loop
2
This is inside loop
2
This is inside loop
2
This is inside loop
2
This is inside loop
2
This is inside loop
2
```

In [49]:

```
# Here we are using 'cnt += 1'

cnt=2
while cnt < 6:
    print(cnt)
    print("This is inside loop")
    cnt +=1
```

```
2
This is inside loop
3
This is inside loop
4
This is inside loop
5
This is inside loop
```

In [52]:

```
cnt=2
while cnt < 6:
    print(cnt)
    print("This is inside loop")
    cnt += 1
else:
        print(cnt)
        print("This is outside loop")
```

```
2
This is outside loop
```

In [53]:
```
cnt=2
while cnt > 6:
    print(cnt)
    print("This is inside loop")
    cnt += 1
else:
        print(cnt)
        print("This is outside loop")
```

```
2
This is outside loop
```

45 WHILE LOOP : Read Data from 'list' by using While loop:

page 20 blue notebook

In [6]:
```
pets = ['cat','dog','cat','goldfish','cat','rabbit','cat']
print(pets)
print("************************************************************")
while 'cat' in pets:
    pets.remove('cat')
    print(pets)
```

```
['cat', 'dog', 'cat', 'goldfish', 'cat', 'rabbit', 'cat']
************************************************************
['dog', 'cat', 'goldfish', 'cat', 'rabbit', 'cat']
['dog', 'goldfish', 'cat', 'rabbit', 'cat']
['dog', 'goldfish', 'rabbit', 'cat']
['dog', 'goldfish', 'rabbit']
```

In [7]:
```
mylist = [1,2,3,4,5,6,7,8,9,10]
for jelly in mylist:
    print(jelly)
```

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

In [20]:
```
for jelly in mylist:
    print(jelly)
    print("Hello.....")
    print("Python")
```

```
1
Hello.....
Python
2
Hello.....
Python
3
Hello.....
Python
4
Hello.....
Python
5
Hello.....
Python
6
Hello.....
Python
7
Hello.....
Python
8
Hello.....
Python
9
Hello.....
Python
10
Hello.....
Python
```

46 Print even number from given list:

In [23]:
```
mylist=[1,2,3,4,5,6,7,8,9,10]
for num in mylist:
    if num % 2 == 0:
        print(num)
```

```
2
4
6
8
10
```

47 Print letter by letter :

In [24]:
```
mystring = "Hello Python...."
for letter in mystring:
    print(letter)
```

```
H
e
l
l
o
 
P
y
t
h
o
n
.
.
.
.
```

In [25]:
```
mystring = "Hello Python...."
for jelly in mystring:
    print(jelly)
```

```
H
e
l
l
o
 
P
y
t
h
o
n
.
.
.
.
```

48 SETS :

In [27]:
```
# More than one tuple in a list is a 'SET':
# [()] LIST OF TUPLE:
```

In [28]:
```
mylist = [(1,2),(3,4),(5,6),(7,8)]
```

In [29]:
```
len(mylist) # Returns number of tuples in a list
```

Out[29]:
```
4
```

49 sets: print all the objects:

In [30]:
```
for item in mylist:
    print(item)
```

```
(1, 2)
(3, 4)
(5, 6)
(7, 8)
```

50 sets: print only oneside objects: split the data:

In [31]:
```
for (a,b)in mylist:
    print(a)
```

```
1
3
5
7
```

51 sets: print both the sides in a Single line column:

In [33]:
```
for (a,b) in mylist:
    print(a)
    print(b)
# Means, like ab ab ab ab ab
```

```
1
2
3
4
5
6
7
8
```

52 sets: Calling middle objects

In [34]:
```
mylist2 = [(1,2,3),(4,5,6),(7,8,9)]
for (a,b,c) in mylist2:
    print(b)
```

```
2
5
8
```

53 sets: Set of keys: like dictionary keys

In [35]:
```
d = {'k1':1, 'k2':2, 'k3':3}   # eg: Here, k1 is 'key' and '1' is value
```

In [36]:
```
type(d)
```

Out[36]:
```
dict
```

In [39]:
```
# All the items not printed:
for item in d:
    print(item)
```

```
k1
k2
k3
```

54 sets: " d.items( ) "

In [40]:
```
# d.items --> Means, 'd' is the data
# in that 'd' --> i want to print all the items:
```

In [51]:
```
d = {'k1':1 , 'k2':2 , 'k3':3}
for item in d.items():
    print(item)
```

```
('k1', 1)
('k2', 2)
('k3', 3)
```

In [52]:
```
for key,value in d.items():
    print(key)
    print("***********")
    print(value)
```

```
k1
***********
1
k2
***********
2
k3
***********
3
```

55 CONCATINATING: " += "

In [53]:
```
hello = "python"
```

In [54]:
```
hello = "java"
```

```
In [55]: hello

Out[55]: 'java'
```

56 Here i want to print both 'python' and 'java'

```
In [77]: hello = "python"

In [78]: hello += " java"

In [79]: hello

Out[79]: 'python java'

In [91]: # one more example:
         name = "PRASANTH"

In [93]: name += " P J"

In [96]: name

Out[96]: 'PRASANTH P J'
```

57 PROMPT :

page 29 blue notebook

In [17]:
```
# Here, we are adding all this, Instead of writing at once.
# We are adding here Step by Step for the same variable.
# if i call one variable, the entire data is printing.

prompt = "Tell something, i will repeat"
prompt += " something."
prompt += "  Enter 'Quit', if logic Done"
```

In [19]:
```
prompt
```

Out[19]:
```
"Tell something, i will repeat something.  Enter 'Quit', if logic Done"
```

58 TUPLE : Complex Data Type:

In [20]:
```
# Whatever the data represents in this particular ( )parentheses is called--> 'COLLECTION OF OBJECTS IN TUPLE'
```

In [22]:
```
# PROPERITIES OF TUPLE:
# 1.Tuple accept all types of Data,
# 2. We Can access objects in Tuple, Based on Index,
# 3. We Can't 'Add', 'Delete', 'Remove', (or) 'Append' Objects from Tuple.
# In 'LIST' we Can 'Add' or 'Remove'. But when comes to 'TUPLE' We Can't...
```

In [28]:
```
# 1.Tuple accept all types of Data: both Numerical and Characters:

tup = (25,36,963,"king","lucky")
```

In [29]:
```
# Tuple of Objects have been Printed:

tup
```

Out[29]:
```
(25, 36, 963, 'king', 'lucky')
```

In [30]:
```
type(tup)
```

Out[30]:
```
tuple
```

In [32]:
```
# 2. We Can access objects in Tuple, Based on Index:

print(tup[1])
```

```
36
```

In [34]:
```
print(tup[3])
```

```
king
```

In [35]:
```
print(tup[4])
```

```
lucky
```

In [36]:
```
print(tup[2])
```

```
963
```

In [37]:
```
print(tup[5])
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_26356\4133959973.py in <module>
----> 1 print(tup[5])

IndexError: tuple index out of range
```

In [40]:
```
# Here it is printing 'Zero(0) index position in the Tuple:

print(tup[0])
```
```
25
```

59 LIST & TUPLE COMBINATION :

In [41]:
```
lt = [("sony",3,"f"),("tony",7,"m")]
```

In [42]:
```
lt
```
Out[42]:
```
[('sony', 3, 'f'), ('tony', 7, 'm')]
```

In [43]:
```
len(lt)
```
Out[43]:
```
2
```

In [44]:
```
# Because, List of Objects:

type(lt)
```
Out[44]:
```
list
```

In [45]:
```
print(lt[0])
```
```
('sony', 3, 'f')
```

In [46]:
```
print(lt[1])
```
```
('tony', 7, 'm')
```

In [48]:
```
# Because, we didn't defined 3rd indexing object in the list of tuple:

print(lt[3])
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_26356\1093007240.py in <module>
      1 # Because, we didn't defined 3rd indexing object in the list of tuple:
      2 
----> 3 print(lt[3])

IndexError: list index out of range
```

60 TUPLE OF LIST :

In [65]:
```
# A LIST OF QULIFICATION IN A TUPLE OF A PERSON :

tup = ('ravi',25,['BTech','MTech'])
```

In [66]:
```
type(tup)
```

Out[66]:
```
tuple
```

In [67]:
```
print(tup)
```

```
('ravi', 25, ['BTech', 'MTech'])
```

In [68]:
```
print(tup[1])
```

```
25
```

In [69]:
```
print(tup[2])
```

```
['BTech', 'MTech']
```

61 DICTIONARY :(dict) VVIMP "COLLECTION OF {KEY : VALUE} OBJECTS"

PAGE 33 BLUE NOTEBOOK

In [70]:
```
# DICTIONARY is nothing but "COLLECTION OF KEY : VALUE OBJECTS"
# The Objects are represented in Flower Brasses {key:value}
# Any data type, it may be take into the consideration. eg: {2015:562} Means, Year:Marks
# Means, We can't take Directly Data Type.

# We Can take Any Data Type --> eg: {TS:HYD} Means, STATE:CITY.
# We Can change the Object via 'Value'.
# We Can "Append" Objects to Dictionary.
```

In [77]:
```
# Dictionary Defined:

another = {"man":"bob",
           "women":"alice",
           "other":"tridy"}
```

In [78]:
```
another
```

Out[78]:
```
{'man': 'bob', 'women': 'alice', 'other': 'tridy'}
```

In [79]:
```
another = {"man":"bob","women":"alice","other":"tridy"}
```

```
In [80]: another

Out[80]: {'man': 'bob', 'women': 'alice', 'other': 'tridy'}
```

```
In [82]: # Now cross checking the 'type':

         type(another)

Out[82]: dict
```

```
In [83]: # Now taking 'empty' Dictionary:

         my_dict = {}
```

```
In [87]: # Adding Object to 'dict' Based on 'key':

         my_dict["Day"] = "Friday"   # Here, we added object which previously we don't have.
         another["Day"] = "Friday"   # Like this way we can 'add' Object to the Dictionary.
```

```
In [88]: my_dict

Out[88]: {'Day': 'Friday'}
```

```
In [89]: another

Out[89]: {'man': 'bob', 'women': 'alice', 'other': 'tridy', 'Day': 'Friday'}
```

62 dict, tuple, list :

page 36 blue notebook

In [90]:
```
# Now let's see this Complex Data Type and How it works Combinationally :
```

In [93]:
```
# Here, the 'key' are in 'tuple': and the units in 'list' are 'values':

sales = {
    ("branch","p1"):[100,200,500],
    ("branch1","p2"):[600,500,750],
    ("branch2","p3"):[500,600,750]
}
```

In [94]:
```
sales
```

Out[94]:
```
{('branch', 'p1'): [100, 200, 500],
 ('branch1', 'p2'): [600, 500, 750],
 ('branch2', 'p3'): [500, 600, 750]}
```

In [96]:
```
type(sales)
```

Out[96]:
```
dict
```

In [98]:
```
# In this particular way, we are declaring dictionary(dict) in a 'List' of 'Tuple' of Data:

print(sales)
```

```
{('branch', 'p1'): [100, 200, 500], ('branch1', 'p2'): [600, 500, 750], ('branch2', 'p3'): [500, 600, 750]}
```

In [99]:
```
# Here, i am declaring 'x' value equal to '25' :

x = 25
```

In [100]:
```
x
```

Out[100]:
```
25
```

In [101]:
```
# But Whenever i want to build a 'LOGIC' :
# There is 'input()' to declare the 'Runtime' declaration :
```

63 RUN TIME DECLARATION :

page 37

In [104]:
```
# Here, We start utilizing 'input()',
# 'input()' method helps you to declare values (or) constants (or) syntax (or) anything on "Runtime".

input()
```

```
23
```

Out[104]:
```
'23'
```

In [2]:
```
x = input()
```

```
786
```

In [3]:
```
# Now whenever we call 'x'

x
```

Out[3]:
```
'786'
```

In [4]:
```
# Now i am re-running the x = input()

x = input()
```

```
899
```

In [6]:
```
# Now whenever we call 'x' again, i am getting newly entered value. Means, In the Runtime

x
```

Out[6]:
```
'899'
```

In [11]:
```
# or else, we can declare like this also
# Means, This is function of an a 'Input Method'. whatever you want, you can declare like this

y = input(" pls enter 'y' value --> ")
```

```
 pls enter 'y' value --> 566
```

In [12]:
```
y
```

Out[12]:
```
'566'
```

In [15]:
```
# Now something different example:
# STATIC INSERTING VALUES ALLOCATION:
# CONSTANTS We are declaring, Whenever we want to change the values, we need to change Manually.

a = 45
b = 65
c = a+b
print(c)
```

```
110
```

In [18]:
```
# int --> Only allows NUMERICALS:
# Now, RUNTIME ALLOCATION (or) DYNAMIC ALLOCATION:

a = int(input("Enter first value \n"))       # Means, Declare value in Runtime:
b = int(input("Enter second value \n"))
print("*********************************")
c = a + b
print(c)
```

```
Enter first value
555
Enter second value
666
*********************************
1221
```

In [23]:
```
# Another good example :

name = input("Enter Name plz --> \n")
age = int(input("Enter Age plz --> \n"))
sal = int(float(input("Enter Salary plz --> \n")))
print("******************************************")
info = (name,age,sal)
print(info)
```

```
Enter Name plz -->
SREEKANTH
Enter Age plz -->
30
Enter Salary plz -->
150000.0000
******************************************
('SREEKANTH', 30, 150000)
```

64 CONTROL FOLW :

PAGE 41 BLUE NOTE BOOK

In [24]:
```
# IF LOOP
# IF ELSE
# IF ELIF ELSE
# These are particular CONDITIONAL LOOPS, and also -->
# WHILE
# WHILE ELSE
# FOR:
```

In [28]:
```
# This is common thing, we know that from 1 to 9 is printing:
for number in range(1,10):
    print(number)
```

```
1
2
3
4
5
6
7
8
9
```

In [33]:
```
# But here, i'm declaring some condition:

# Using 'BREAK()' in CONDITIONAL LOOPS:

for number in range(1,10):
    if number==7:   # Means, if 1 == 7 , The condition failed. same as 2,3,4,5,6
        break        # Here 7==7, The condition satisfied. so, it has been stopped here.
    print(number)
```

```
1
2
3
4
5
6
```

In [35]:
```
# Using 'CONTINUE' in CONDITIONAL LOOPS :

for number in range(1,10):
    if number == 7:
        continue        # Means, 7==7 'continue', Here it will skip the '7'. that's why we don't have '7' in the OL
    print(number)        # And it will continue the Next Number.
```

```
1
2
3
4
5
6
8
9
```

65 NESTED LOOPS :

PAGE 44 BLUE NOTE BOOK :

In [19]:
```
# Nested loops are interesting and very important also:
# Here, Once the particular '4' numbers in 'i' has been completed multiplication with first object '10' in 'j',
# then Second time j == 20(20=20) it will become, Now the condition has been satisfied.
# then the loop has been closed in Break().
# This is called 'Nested Working Environment'.


list1 = [4,5,6,7]
list2 = [10,20,30,40]

for i in list1:
    for j in list2:
        if j == 20:
            break
        print(i * j)
    print("Outside the Loop")
```

```
40
Outside the Loop
50
Outside the Loop
60
Outside the Loop
70
Outside the Loop
```

In [20]:
```
# eg 2

list1 = [4,5,6,7]
list2 = [10,20,30,40]

for i in list1:
    for j in list2:
        if j == 10:
            break
        print(i * j)
    print("Outside the Loop")
```

```
Outside the Loop
Outside the Loop
Outside the Loop
Outside the Loop
```

In [21]:
```
# eg 3

list1 = [4,5,6,7]
list2 = [10,20,30,40]

for i in list1:
    for j in list2:
        if j == 30:
            break
        print(i * j)
    print("Outside the Loop")
```

```
40
80
Outside the Loop
50
100
Outside the Loop
60
120
Outside the Loop
70
140
Outside the Loop
```

In [22]:
```
# eg 4

list1 = [4,5,6,7]
list2 = [10,20,30,40]

for i in list1:
    for j in list2:
        if j == 40:
            break
        print(i * j)
    print("Outside the Loop")
```

```
40
80
120
Outside the Loop
50
100
150
Outside the Loop
60
120
180
Outside the Loop
70
140
210
Outside the Loop
```

In [25]:
```
# eg 5
# if i give j == 0, then all the objects in list 2 have been multiplied with list 1.

list1 = [4,5,6,7]
list2 = [10,20,30,40]

for i in list1:
    for j in list2:
        if j == 0:
            break
        print(i * j)
    print("Outside the Loop")
```

```
40
80
120
160
Outside the Loop
50
100
150
200
Outside the Loop
60
120
180
240
Outside the Loop
70
140
210
280
Outside the Loop
```

66 CONTINUES METHOD in NESTED LOOP :

PAGE 49 BLUE NOTE BOOK:

In [27]:
```
# Here, The multiplication with '20' is missing,
# Because, It has given 'Continue' at this stage.

# This is a 'Nested Loop' of an a 'Condition' with 'Break' and 'Continue'.

list1 = [4,5,6,7]
list2 = [10,20,30,40]

for i in list1:
    for j in list2:
        if j == 20:
            continue
        print(i * j)
    print("Outside the Loop")
```

```
40
120
160
Outside the Loop
50
150
200
Outside the Loop
60
180
240
Outside the Loop
70
210
280
Outside the Loop
```

67 Even Numbers :

page 50

In [32]:
```
# Here, i'm giving, The List of range of '0' to '11' with the step function of '2'.
# Here it is printing --> 0,1, '2',  2,3, '4',  4,5, '6',  6,7, '8',  8,9, '10'.

list(range(0,11,2))
```

Out[32]:
```
[0, 2, 4, 6, 8, 10]
```

68 index_count :

In [38]:
```
# Here, we didn't get index positions, All printed 'Zero(0)' itself.
# Because we didn't added += 1 to the index_count.
# Means, Index_Count Starts from 'Zero' and letters will settle down at String{}.

index_count = 0
for letter in 'abcde':
    print('At index{} the letter is {}'.format(index_count,letter))
```

```
At index0 the letter is a
At index0 the letter is b
At index0 the letter is c
At index0 the letter is d
At index0 the letter is e
```

In [39]:
```
# Here, we got all index positions successfully.
# Because, we gave index_count += 1.

index_count = 0
for letter in 'abcde':
    print('At index{} the letter is {}'.format(index_count,letter))
    index_count += 1
```

```
At index0 the letter is a
At index1 the letter is b
At index2 the letter is c
At index3 the letter is d
At index4 the letter is e
```

69 ENUMERATE METHOD() :

PAGE 51 BLUE NOTE BOOK :

In [40]:
```
# Enumerate method() helps to count the Iterations:
# The difference here is Manually we are declaring the 'count' in index_count() eg: index_count += 1
# But in 'Enumerate()' it will help to you declare the index positions(0,1,2,3,4,5.....) automatically.
# By using this enumerate method only, we are getting this count automatically.
```

In [41]:
```
word = 'abcde'
for item in enumerate(word):   # enumerate method() of an a word:
    print(item)
```

```
(0, 'a')
(1, 'b')
(2, 'c')
(3, 'd')
(4, 'e')
```

70 INDEPENDENT METHOD :

PAGE 52 BLUE NOTE BOOK:

In [43]:
```
# Here, i want to print 'INDEPENDENTLY':

word = 'abcde'
for index,letter in enumerate(word):
    print(index)
    print(letter)
    print('\n')
```

```
0
a


1
b


2
c


3
d


4
e


```

70 'ZIP' METHOD :

PAGE 53 BLUE NOTE BOOK:

In [44]:
```
# ZIP METHOD(): "Whenever we want to mix the two particular list data - we are using this 'zip' method ".
```

In [45]:
```
list1 = [4,5,6,7]
list2 = [10,20,30,40]

for item in zip(list1,list2):
    print(item)
```

```
(4, 10)
(5, 20)
(6, 30)
(7, 40)
```

In [53]:
```
# Here, How many variables you want, you can 'zip' it:
# by changing list1 values [4,5,6,7] to [4,5,6] or in list2 values [10,20,30,40] to [10,20,30]....

list1 = [4,5,6]
list2 = [10,20,30,40]

for item in zip(list1,list2):
    print(item)
```

```
(4, 10)
(5, 20)
(6, 30)
```

In [54]:
```
list1 = [4,5]
list2 = [10,20,30,40]

for item in zip(list1,list2):
    print(item)
```

```
(4, 10)
(5, 20)
```

In [60]:
```
list1 = [4,5,6,7]
list2 = [10]

for item in zip(list1,list2):
    print(item)
```

```
(4, 10)
```

71 SET : Complex data type:

PAGE 55 BLUE NOTE BOOK :

In [61]:
```
# Set is an a most important property:
# Set is also another 'Complex data type'.
# In this particular complex data type(SET), Whatever the Objects we are representing in {} flower Brasses
# is called as an a list objects.
```

In [62]:
```
# Properties of 'Set':
# 1. Set won't accept duplicates
# 2. Set won't allow adding, deleting, removing (or) appending objects.
# 3. We can't call objects based on index.
```

In [64]:
```
dt = {1,2,1,2,4,5,7,8,1,2,4,5}
```

In [65]:
```
type(dt)
```

Out[65]:
```
set
```

In [67]:
```
print(dt)   # Automatically it will remove the Duplicates from the 'Set'.
```

```
{1, 2, 4, 5, 7, 8}
```

'Update' object based on Set only :

In [69]:
```
# Direct, Independent, Objects we can't 'add'.
# But, We can 'update' object - Based on Set Only.

dt.update([45,55])
```

In [71]:
```
# Now, if we call 'dt' --> it will add the objects ([45,55]) to the list in the 'Set':

dt
```

Out[71]:
```
{1, 2, 4, 5, 7, 8, 45, 55}
```

72 FLAG : TRUE OR FALSE :

PAGE 57 BLUE NOTE BOOK :

In [83]:
```
# 'Flag' means, "The Declaration True or False"
# Means, Whatever we say, it is getting repeated.
# if we declare 'Quit' it will --> Stop.
# This is known as - "Flag declaration in Conditional loops".

prompt = "\n Tell me something, will"
prompt += " repeat it back to you."
prompt += " Enter 'Quit' to end the program\n"
active = True
while active:
    message=input(prompt)
    if message == 'Quit':
        active = False
    else:
        print(message)
```

```
 Tell me something, will repeat it back to you. Enter 'Quit' to end the program
Hi i am PRASANTH
Hi i am PRASANTH

 Tell me something, will repeat it back to you. Enter 'Quit' to end the program
How are you
How are you

 Tell me something, will repeat it back to you. Enter 'Quit' to end the program
i'm fine
i'm fine

 Tell me something, will repeat it back to you. Enter 'Quit' to end the program
Quit
```

In [84]:
```python
# eg 2
# Directly Declaring 'While True:' here;

prompt = "\n What Cities have been Visited"
prompt += " Enter 'Quit' When you have done \n"

while True:
    City=input(prompt)
    if City == 'Quit':
        break
    else:
        print(" I Have been to "+City+"......!")
```

```
 What Cities have been Visited Enter 'Quit' When you have done 
Mumbai
 I Have been to Mumbai......!

 What Cities have been Visited Enter 'Quit' When you have done 
Chennai
 I Have been to Chennai......!

 What Cities have been Visited Enter 'Quit' When you have done 
Andhra Pradesh
 I Have been to Andhra Pradesh......!

 What Cities have been Visited Enter 'Quit' When you have done 
Next Telangana
 I Have been to Next Telangana......!

 What Cities have been Visited Enter 'Quit' When you have done 
Quit
```

73 Print Variables by "String Format" :

page 60 BLue NoteBook :

In [5]:
```
lang = 'python'
version = '3.8'
print(f'I am learning {lang} version {version}')
```

```
I am learning python version 3.8
```

In [8]:
```
# Now i want to print some print statements:

print('=' * 40)
print('Author : Jagadeesh')
print( 'date : 23-01-2023')
print( '=' * 40)
```

```
========================================
Author : Jagadeesh
date : 23-01-2023
========================================
```

74 AREA CALICULATION :

In [24]:
```
# Area Formula --> area = pi * radius ** 2
# This is how we are getting an a particular Area Caliculation.

pi = 3.14
radius = 5
area = pi * radius ** 2
print(f'Area: {area : 1f}')   # Now, String of colon: one 'f'
```

```
Area:  78.500000
```

75 Find the File with proper Extension/Ending with colon':'

page 61

In [36]:
```
# I want to identify the 'File' in the 'Folder name'.

filename = 'emp12.csv'
if filename.endswith('csv'):
    print('YES')
else:
    print('NO')
```

```
YES
```

In [37]:
```
filename = 'civil.csv'
if filename.endswith('csv'):
    print('YES')
else:
    print('NO')
```

```
YES
```

In [41]:
```
filename = 'civil.txt'
if filename.endswith('txt'):
    print('YES')
else:
    print('NO')
```

```
YES
```

In [42]:
```
filename = 'civil.txt'
if filename.endswith('csv'):
    print('YES')
else:
    print('NO')
```

```
NO
```

In [43]:
```
filename = 'civil.txt'
if filename.endswith('txtghh'):
    print('YES')
else:
    print('NO')
```

```
NO
```

In [51]:
```
# 76 IDENTIFY (or) COUNT NUMBER OF PASSWORD CHARACTERS :

password = 'london12345'
if len(password)>=11:
    print('password correct')
```

```
password correct
```

In [52]:
```
password = 'london12345'
if len(password)>=15:
    print('password correct')
else:
    print('password Incorrect')
```

```
password Incorrect
```

77 CHECK 'PASSWORD' HAS '11' CHARACTERS & SPECIAL CHARACTERS AT '!'(NOT EQUAL TO) AVAILABLE OR NOT :

In [58]:
```
# Here, we are taking 'and' Operator, and special character '!' in password:
# also identifying --> Length as well as this '!' special character is available in password or not :

password = 'london12345!'
if len(password)>=11 and '!' in password:
    print('password correct')
else:
    print('password incorrect')
```

```
password correct
```

In [61]:
```
# Here, we are identifying particular project_id is there in existing project_ids and
# we are appending(adding) particular project_id to existing project_ids by using --> 'if not' and '.append()'

project_ids = ['01234','242526']
project_id = '01235'
if not project_id in project_ids:
    project_ids.append(project_id)
print(project_ids)
```

```
['01234', '242526', '01235']
```

78 LOGIC THAT FIND ALL '2' DIGIT NUMBERS, WHICH IS 'DIVISABLE' BY AND 'INDIVISABLE' BY SOME PARTICULAR VALUE :

In [67]:
```
# By using an a 'loop', we are going to work it out.
# Here, the logic that finds all '2' digit numbers, which divided by '11' and also not by'!' '3'.

result = []
for i in range(10,100):
    if i % 11 == 0 and i % 3!= 0:
        result.append(str(i))
print(' , '.join(result))
```

```
11 , 22 , 44 , 55 , 77 , 88
```

79 PYTHON FUNCTIONS :

PAGE 66 BLUE NOTE BOOK :

In [1]:
```
# Functions is a group of related statements that perform particular task :
# "To overcome the 'Clumsiness' and 'to reduce the code' and 'to utilize the re-usable code',
# we declare the logic in the function" :
# Function helps - Break our program into Similar and Smaller Modular Chunks:
# Means, De-bugging 45000 lines is pretty difficult, By using Function we can Debug 5000 to 4000 lines effectively
# It avoids repetation and makes 'Code' re-usable.


# TYPES OF FUNCTIONS :
# UDF - User Defined Function,
# PDF - Pre Defined Function.

# UDF - "Based on the requirement", Means, Based on the user requirement, Building the Logic.
# PDF - "Defaultly This will be given by Python Environment", Means, Already Defined by Python --> We are 'Utilizir
# This is called as Pre-Defined Function(PDF).
```

In [2]:
```
# SYNTAX FOR FUNCTION :
# Function always starts from -->
# "def function_name(parameters/passing values):"
# Function always ends with Colon':'
# Here, we are going declare an a """Doc String""" . This is a particular Syntax.
# And this is an a Statement --> stmt 1, stmt 2, stmt 3 .....
# Sometimes we are going to utilize 'Return'. But 'Return' is a 'Optional'.
# Comparing to PYTHON and OTHER LANGUAGES --> The most important thing is something which is called 'Return Stateme
# In OTHER LANGUAGES --> 90% We require Mandatorily with an a 'Return Statement',
# But Whenever we come to PYTHON --> WE DON'T REQUIRE 'RETURN STATEMENT' --> IT'S AN A 'OPTIONAL PART' ONLY.
```

In [3]:
```
# DOC STRING :
# "It describe/ Comment / Explanation about that particular Function"
# DOC STRING --> Is a 'OPTIONAL' in Function :
```

In [4]:
```
# THE POINTS : HOW TO DECLARE THE FUNCTION :
# 1. Key Word 'def' marks to start Function Header.
# 2. 'Parameter' or Which we call 'Argument' - through which we can pass Value to Function.
# 3. 'Argument' - Means, 'PASSING VALUES'.
# 4. A 'COLON:' - To mark the end of Function Header.
# 5. 'RETURN STATEMENT(stmt)' - To return a value from the Function. This is 'OPTIONAL'.
# 6. 'DOC STRING' - Describes about Function, But 'OPTIONAL'.
```

In [5]:
```
def print_name(name):
    """This Function just prints names"""  # This is doc-string.
    print("Hello....!"+str(name))
```

In [8]:
```
print_name(" PRASANTH")  # Here, 'print_name' is an a Function Name. Function Name only we called here.
```

```
Hello....! PRASANTH
```

In [11]:
```
print_name(" KING")
```

```
Hello....! KING
```

NO ARGUMENT FUNCTION :

In [13]:
```
def print_2():
    """This Function without arg ::: """
    print("This is No Argument Function")
```

In [14]:
```
print_2()
```

```
This is No Argument Function
```

READING DOC_STRING OF PARTICULAR FUNCTION :

In [16]:
```
print(print_name.__doc__)  # doc_string --> Which we mentioned in the """ """ .
```

```
This Function just prints names
```

In [17]:
```
print(print_2.__doc__)
```

```
This Function without arg :::
```

80 SET : COMPLEX DATA TYPES :

PAGE 76 BLUE NOTE BOOK :

In [18]:
```
# 'SET' is going to be represented in Flower Brakets {}
# We Call this one --> {} As an a 'Set of OBJECTS'. Whatever we declare here, are called 'Set of Objects'.

# PROPERTIES OF 'SET' :
# 1. A Set is an UNORDERED COLLECTION OF OBJECTS.
# 2. "We Can't Call Objects from 'Set' --> Based on Index".
# 3. 'Set' --> Won't Allow DUPLICATES.
# 4. We Can ADD OBJECTS TO SET (SINGLE OR MULTI).
# 5. 'SET' IS USED TO PERFORM --> FOR MATHEMATICAL OPERATIONS such as (UNION, INTERSECTION, DIFFERENCE e.t.c) utili
```

In [19]:
```
dt = {1,1,21,2,12,3,4,5,3,2,1}
```

```
In [20]: type(dt)

Out[20]: set

In [21]: dt   # Here, 'SET' has Removed Duplicates.

Out[21]: {1, 2, 3, 4, 5, 12, 21}

In [22]: str2 = {'ram','king','horse','ram','king'}

In [23]: str

Out[23]: str

In [25]: str2   # Removed Duplicates --> ALso Applicable on the Alpha(Alphabeticle Data).

Out[25]: {'horse', 'king', 'ram'}

In [26]: str3 = {'ram','king',3,4,5,7,3,2,1,1,1,'horse','ram','king'}

In [27]: str3   # 'NUMERIC' Will be Applicable First :

Out[27]: {1, 2, 3, 4, 5, 7, 'horse', 'king', 'ram'}

In [28]: # The Main Property means, --> We Can't Call Objects, Based on Index in the 'Set'.
         # Because, It is a UNORDERED ELEMENT : It is NOT A STABLE.
```

'ADD OBJECTS' TO SET :

In [34]:
```
# Taking (dt) instead of (str):

dt.add(556)   # Adding to 19th sum values.
```

In [35]:
```
dt
```

Out[35]:
```
{1, 2, 3, 4, 5, 12, 21, 556}
```

UPDATE OBJECTS TO LIST :

In [38]:
```
# Here, In ADD --> We Can "ADD" Only 'SINGLE OBJECT'.
# But In UPDATE --> We Can "ADD" 'MULTIPLE OBJECTS'.

dt.update([55,66,77])
```

In [62]:
```
dt
```

Out[62]:
```
{1, 2, 3, 4, 5, 12, 21, 25, 55, 65, 66, 85, 95, 556}
```

UPDATE 'LIST' AND ALSO 'SET' at a Time :

In [63]:
```
dt.update([85,95,65],{25,1575,95})
```

In [64]:
```
type(dt)
```

Out[64]:
```
set
```

```
In [65]: # Means, We are Updating 'LIST' and Directly 'SET' also Combinedly :
         # This is The ADVANTAGE OF THE 'SET' :

         dt

Out[65]: {1, 2, 3, 4, 5, 12, 21, 25, 55, 65, 66, 85, 95, 556, 1575}
```

DISCARD in SET : 'Deleting Particular Object' :

```
In [74]: # 'DISCARD' Means, 'Deleting particular Object':
         # 'REMOVE' also Work in the Same Way like 'DISCARD'. But, There is no much difference between 'DISCARD' & 'REMOVE':
         # 'REMOVE' delete single object.
         # 'DISCARD' delete multiple objects.

         dt.discard(1575)
```

```
In [75]: dt    # The Number (1575) have been deleted in the 'SET':

Out[75]: {1, 2, 3, 4, 5, 12, 21, 25, 55, 65, 66, 85, 95, 556}
```

```
In [68]: dt.remove(77)

---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_1100\841475941.py in <module>
----> 1 dt.remove(77)

KeyError: 77
```

```
In [71]: dt

Out[71]: {1, 2, 3, 4, 5, 12, 21, 25, 55, 65, 66, 85, 95, 556}
```

```
In [69]: dt.remove(85,95)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_1100\2489827418.py in <module>
----> 1 dt.remove(85,95)

TypeError: set.remove() takes exactly one argument (2 given)
```

```
In [72]: dt.discard(12,21)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_1100\1803533850.py in <module>
----> 1 dt.discard(12,21)

TypeError: set.discard() takes exactly one argument (2 given)
```

```
In [76]: dt
```

```
Out[76]: {1, 2, 3, 4, 5, 12, 21, 25, 55, 65, 66, 85, 95, 556}
```

81 THE OPERATIONS :

PAGE 80 BLUE NOTEBOOK :

```
In [78]: # The Operations --> UNION, INTERSECTIONS e.t.c
```

UNION OPERATION :

In [87]:
```
# The 'INTERSECTION' Represents The 'COMMON ELEMENTS'.
# The 'UNION' Represents The 'OVER ALL ELEMENTS':

set1 = {1,2,3,4,5,6,7}
set2 = {3,4,5,6,7,8,9}
# UNION :
print(set1|set2)        # Here, We are using 'PIPE '|' SYMBOL' --> Which We Call as an a 'UNION'.
print(set1.union(set2))  # or We Can Call like this also.
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9}
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

In [91]:
```
# UNION : Here, All the Objects are printed. 'NO DUPLICATES'.

set1 = {1,2,3,4,5,6,7}
set2 = {3,4,5,6,7,8,9}
print(set1|set2)
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

In [92]:
```
set1 = {1,2,3,4,5,6,7}
set2 = {3,4,5,6,7,8,9}
print(set1.union(set2))
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

INTERSECTION : SET

In [93]:
```
# The 'INTERSECTION' Represents The 'COMMON ELEMENTS'.
# The 'UNION' Represents The 'OVER ALL ELEMENTS':

set1 = {1,2,3,4,5,6,7}
set2 = {3,4,5,6,7,8,9}
# INTERSECTION
print(set1 & set2)
print(set1.intersection(set2))
```

```
{3, 4, 5, 6, 7}
{3, 4, 5, 6, 7}
```

THE DIFFERENCE : SET

In [94]:
```
# The DIFFERENCE : Means, (a - b)
```

In [95]:
```
set1 = {1,2,3,4,5,6,7}
set2 = {3,4,5,6,7,8,9}
# DIFFERENCE :
print(set1 - set2)
print(set1.difference(set2))
```

```
{1, 2}
{1, 2}
```

In [96]:
```
# Here, if we give 'Reverse', 'Vise-Versa' --> set2 - set1 --> It Will get 8,9.

set1 = {1,2,3,4,5,6,7}
set2 = {3,4,5,6,7,8,9}
# REVERSE :
print(set2 - set1)
print(set2.difference(set1))
```

```
{8, 9}
{8, 9}
```

82 CONDITIONAL LOOPS :

PAGE 82 BLUE NOTEBOOK :

In [97]:
```
# The Most Important in 'CONDITION LOOPS' is :
# 1. 'if' loop --> We Utilize, Whenever We have 'Single Condition'.
# 2. 'if else' loop --> Whenever We have '2' Conditions.
# 3. 'if elif else' loop --> Wherever We have 'More than '2' Conditions'. Means, That many "elif's" We are going tc
# in our WORKING ENVIRONMENT.
# And also we have -->
# 4. 'while' loop :
# 5. 'while else' :
# And also we have -->
# 6. 'NESTED LOOPS' (if, while, for)
# NESTED LOOPS Include --> 'Multiple Conditional loops' --> like, (if, while, for).
# In NESTED LOOPS --> We can see, The BLOCK of 1st Condition, The BLOCK of 2nd Condition, The BLOCK of 3rd Conditic
```

In [98]:
```
# An 'if' condition is a programming.
# the conditional loops also called as 'control flows',
# In conditional loops --> we are going to work with --> if, if else, if elif else.
# Here, we are trying to execute a 'Bulk of logic or Code', if the Condition only Satisfies.
# The Conditional Statements or Loops helps to build --> 'HIGHEND PROGRAMMING LANGUAGE' :
```

In [100]:
```
# Here, 'True' Means, Accepts all data to 'Enter Loop' :
# True --> Any Data Condition, Whatever it may be, it will 'TAKE IT'.

if True:
    print("All Accepted...!")
```

```
All Accepted...!
```

83 If Condition True only, then Enter Loop :

page 88 BLUE NOTE BOOK :

In [3]:
```
x = 5
if x < 6:
    print("Entered Loop")
else:
    print("NOT SATISFIED")
```

```
Entered Loop
```

In [6]:
```
x = 5
if x > 6:
    print("Entered Loop")
else:
    print("NOT SATISFIED")
```

```
NOT SATISFIED
```

In [8]:
```
# Nothing is getting, The Main reason is '9' is not less than '6'. So, The above Condition is Failed.

x = 9
if x < 6:
    print("Entered Loop")
```

In [9]:
```
x = 6
if x > 5:
    print(" 'x' is Greater than 5 ")
```

```
 'x' is Greater than 5
```

In [13]:
```
x = 6
if x > 5:
    print("'x' is Greater than 5 ")
    print(  x * 2 )       # Means, 6 * 2
```

```
'x' is Greater than 5
12
```

In [16]:
```
# Else :

x = 5
if x > 5:
    print("'x' is Greater than 5 ")
    print(  x * 2 )        # Means, 6 * 2
else:
    print("x is not greater than 5")
```

```
x is not greater than 5
```

In [17]:
```
# More than '2' conditions :

age = 22
if age < 4:
    print("Your age is Under 4:")
elif age < 18:
    print("Under 18")
else:
    print("Valid")
```

```
Valid
```

In [21]:
```
x = 5
if x > 5:
    print("x is greater than 5")
    print(x * 2)
elif x == 5:
    print("x is Equal to 5:")
else:
    print("x is greater than 5:")
```

```
x is Equal to 5:
```

84 DUAL CONDITION IN A SINGLE STATEMENT, USING 'AND' OPERATOR :

PAGE 91

In [4]:
```
# Means, Single statement, multiple conditions :
# 'and' operator always declare '2' conditions need to be 'True'.

x = 15
if x == 5 and type(x) is int:
    print("x is equal to 5, x is a integer")
    print(x)
elif x > 10 and type(x) is int:
    print("x is an integer, But it is x value")
    print(x)
```

```
x is an integer, But it is x value
15
```

85 NESTED LOOPS ('for' and 'if' loop) :

page 92

In [6]:
```
# Here, We are Reading car one by one and Cross Checking the Condition :
# for loop --> We are Using 'for loop' --> "To Read Object One by One".
# This is Called as an a 'NESTED PATTERN'.

cars = ['audi','bmw','maruti','kia','toyota']
for car in cars:
    if car == 'bmw':
        print(car.upper())
    else:
        print(car.title())
```

```
Audi
BMW
Maruti
Kia
Toyota
```

86 WHILE LOOP :

PAGE 93

In [7]:
```
# 'While loop' is utilized to get an a unlimited of 'Iterations'.
# or We can Call as an a 'INFINITY LOOP'
# In 'While loop' there will be 'Increment' of an a 'Manual Increment'.
```

In [9]:
```
# This is an a 'Infinity loop'.
# if we won't stop, it will go on printing the loop.

x = 1
while x <= 5:
    print(x)
```

```
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
```

In [10]:
```
# So, To Overcome this 'Infinity loop', we need to take an a 'Increment' to stop Unlimited of Iterations:

x = 1
while x <=5:
    print(x)
    x = x + 1    # This is an 'Increment'.
```

```
1
2
3
4
5
```

In [12]:
```
# eg 2
# It is infinity loop :

cnt = 2
while cnt < 6:
    print(cnt)
    print("This is inside the loop :")
```

```
2
This is inside the loop :
2
This is inside the loop :
2
This is inside the loop :
2
This is inside the loop :
2
This is inside the loop :
2
This is inside the loop :
2
This is inside the loop :
2
This is inside the loop :
2
This is inside the loop :
2
```

In [17]:
```
cnt = 2
while cnt < 6:
    print(cnt)
    cnt += 1
    print("This is inside the loop :")
```

```
2
This is inside the loop :
3
This is inside the loop :
4
This is inside the loop :
5
This is inside the loop :
```

In [18]:
```
cnt = 2
while cnt < 6:
    print(cnt)
    cnt += 2
    print("This is inside the loop :")
```

```
2
This is inside the loop :
4
This is inside the loop :
```

87 WHILE ELSE :

PAGE 97

In [20]:
```
# 'else' Statement in a 'while Loop' :
```

In [21]:
```
cnt = 2
while cnt < 6:
    print(cnt)
    cnt += 1
    print("This is inside loop :")
else:
    print("This is outside loop :")
    print(cnt)
```

```
2
This is inside loop :
3
This is inside loop :
4
This is inside loop :
5
This is inside loop :
This is outside loop :
6
```

In [22]:
```
cnt = 2
while cnt < 6:
    print(cnt)
    cnt += 2
    print("This is inside loop :")
else:
    print("This is outside loop :")
    print(cnt)
```

```
2
This is inside loop :
4
This is inside loop :
This is outside loop :
6
```

88 "BREAK" and "CONTINUE" in 'CONDITIONAL LOOPS' :

PAGE 98

BREAK :

In [23]:
```
# BREAK --> STOP THE LOOP :
```

In [25]:
```
for number in range(1,10):
    if number == 7:
        break
    print(number)
```

```
1
2
3
4
5
6
```

CONTINUE :

In [26]:
```
# CONTINUE : It Skip Current Recurrence and Will Continue :
```

In [28]:
```
for number in range(1,10):
    if number == 7:
        continue
    print(number)    # '7' missing from the below recurrence.
```

```
1
2
3
4
5
6
8
9
```

88 "BREAK" and "CONTINUE" in 'NESTED LOOPS' :

PAGE 99

In [30]:
```
# It's an a pure NESTED LOOP : very important :
```

In [47]:
```
# It is trying to Iterate '10' itself :

list1 = [4,5,6,7]
list2 = [10,20,30,40]
for i in list1:
    for j in list2:
        if j == 20:
            break
        print(i * j)
    print("Outside the Nested Loop")
```

```
40
Outside the Nested Loop
50
Outside the Nested Loop
60
Outside the Nested Loop
70
Outside the Nested Loop
```

In [48]:
```python
# It is trying to Iterate '10' itself :

list1 = [4,5,6,7]
list2 = [10,20,30,40]
for i in list1:
    for j in list2:
        if j == 20:
            continue
        print(i * j)
    print("Outside the Nested Loop")
```

```
40
120
160
Outside the Nested Loop
50
150
200
Outside the Nested Loop
60
180
240
Outside the Nested Loop
70
210
280
Outside the Nested Loop
```

89 RANGE METHOD( ) :

PAGE 101 BLUE NOTE BOOK :

In [ ]:
```
```