AKTU B.Tech II-Year

ONE SHOT Revision

Python Programming

Common to all Branches

Unit-3 (Most Imp. Unit)

Python Complex Data Types

Pragya ma'am

PYTHON PROGRAMMING

AKTU UNIT-3 Syllabus

Python Complex data types: Using string data type and string operations, Defining list and list slicing, Use of Tuple data type. String, List and Dictionary, Manipulations Building blocks of python programs, string manipulation methods, List manipulation. Dictionary manipulation, Programming using string, list and dictionary in-built functions. Python Functions, Organizing python codes using functions.

Topic wise All PYQs Covered

LIST AKTU (2022-23)

- The list is a sequence data type which is used to store the collection of data
- Lists in Python can be created by just placing the sequence inside the square brackets[ ]. .
- list doesn't need a built-in function for its creation of a list. .
- the list may contain mutable element
- list may contain duplicate values

with their distinct positions

```
List = []
print("Blank List: ")
print(List)
List = [10, 20, 14]
print("\nList of numbers: ")
print(List)
List = ["GATEWAY", "CLASSES", "HELLO"]
print("\nList Items: ")
print(List[0])
print(List[2])
```

```
List = [1, 2, 4, 4, 3, 3, 3, 6, 5]
print("\nList with the use of Numbers: ")
print(List)
```

List Manipulation

LIST METHODS (AKTU 2022-23)

1 append()

- Used for adding elements to the end of the List.

```
1 l=[2,5,6,7]
2 l.append(8)
3 print(l)
```

```
[2, 5, 6, 7, 8]
```

2 max()

Calculates the maximum of all the elements of the List

```
1 l = [1.2, 1.3, 0.1]
2 max_value = max(l)
3 print(max_value)
4
```

```
1.3
```

3 min()

Calculates the minimum of all the elements of the List

```
1 numbers = [3, 2, 8, 5, 10, 6]
2 small = min(numbers);
3 print("The smallest number is:", small)
4
```

```
The smallest number is: 2
```

4 reverse() Reverses objects of the List in place

```
list1 = [1, 2, 3, 4, 1, 2, 6]
list1.reverse()
print(list1)
```

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

5 remove() Removes a given object from the List.

```
lis = ['a', 'b', 'c']
lis.remove("b")
print(lis)
```

```
['a', 'c']
```

6.Sort() Sort a List in ascending, descending, or user defined order

```
numbers = [1, 3, 4, 2]
numbers.sort(reverse=True)
print(numbers)
```

```
[4, 3, 2, 1]
```

```
numbers = [1, 3, 4, 2]
numbers.sort()
print(numbers)
```

```
[1, 2, 3, 4]
```

7 pop() Removes and returns the last value from the List or the given index value.

```
my_list = [1, 2, 3, 6,9, 8,12,5]
print("Popped element:", my_list.pop(5))
print("List after pop():", my_list)
```

```
Popped element: 8
List after pop(): [1, 2, 3, 6, 9, 12, 5]
```

```
my_list = [1, 2, 3, 6]
print("Popped element:", my_list.pop())
print("List after pop():", my_list)
```

```
Popped element: 6
List after pop(): [1, 2, 3]
```

8insert() method inserts an item at a specific index in a list.

```
fruit = ["banana","cherry","grape"]
fruit.insert(1,"apple")
print(fruit)
```

```
['banana', 'apple', 'cherry', 'grape']
```

9 index() method searches for a given element from the start of the list and returns the position of the first occurrence.

```
Animals= ["cat", "dog", "tiger","dog"]
print(Animals.index("dog"))
```

```
1
```

10 clear() removes all items from the List making it an empty/null list.

```
lis = [1, 2, 3]
lis.clear()
print(lis)
```

```
[]
```

11.Count() returns the count of the occurrences of a given element in a list.

main.py

```
fruits=["apple","cherry","apple"]
print(fruits.count("apple"))
```

```
2
```

12 extend()

It is method adds items of an iterable (list, tuple, dictionary, etc) at the end of a list.

```
cars = ["Audi","BMW","Jaguar"]
other_cars = ["Maruti", "Tata"]
cars.extend(other_cars)
print(cars)
```

```
['Audi', 'BMW', 'Jaguar', 'Maruti', 'Tata']
```

13 Copy()

This method creates a shallow copy of the list.

Changes to the copied_list do not affect the original_list, and vice versa.However, if the list contains mutable elements (like other lists), those elements are not copied deeply, meaning changes to nested elements will reflect in both lists.

```python
import copy
original_list = [[1, 2], [3, 4]]
shallow_copied_list = copy.copy(original_list)
deep_copied_list = copy.deepcopy(original_list)
original_list[0][0] = 99
print("Original List:", original_list)
print("Shallow Copied List:", shallow_copied_list)
print("Deep Copied List:", deep_copied_list)
```

input

```
Original List: [[99, 2], [3, 4]]
Shallow Copied List: [[99, 2], [3, 4]]
Deep Copied List: [[1, 2], [3, 4]]
```

main.py

```python
import copy
list=[1,2,3]
a=copy.copy(list)
print("shallow copy",a)
list[0]=5
print("original list",list)
print("shallow list",a)
```

```
shallow copy [1, 2, 3]
original list [5, 2, 3]
shallow list [1, 2, 3]
```

LIST ARE MUTUBLE (AKTU 2023-24 ODD)

- Yes, lists in Python are mutable, which means that their elements can be changed after the list has been created. This includes adding, removing, or modifying elements.

```
[1, 2, 99, 4, 5]
[1, 2, 3, 4]
[1, 99, 2, 3, 4]
[1, 99, 2, 3, 4, 5, 6]
```

```
#MODIFYING ELEMENT
L = [1, 2, 3, 4, 5]
L[2] = 99
print(L)
#ADDING ELEMENT
L1 = [1, 2, 3]
L1.append(4)
print(L1)
L1.insert(1, 99)
print(L1)
L1.extend([5, 6])
print(L1)
```

```
l3=[1,2,3,7,4,5]
del l3[0]
print (l3)
```

```
[2, 3, 7, 4, 5]
```

```
# deleting element
l2=[1,2,3,4,5]
l2.remove(3)
print(l2)
popped_element=l2.pop(1)
print(l2)
```

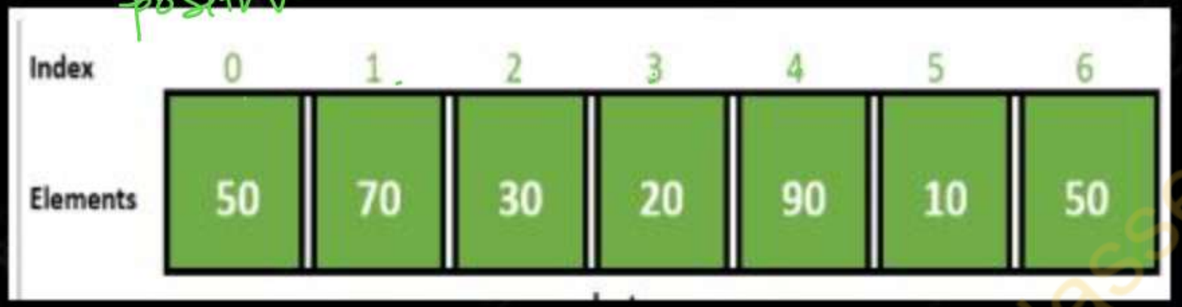
```
[1, 2, 4, 5]
[1, 4, 5]
```

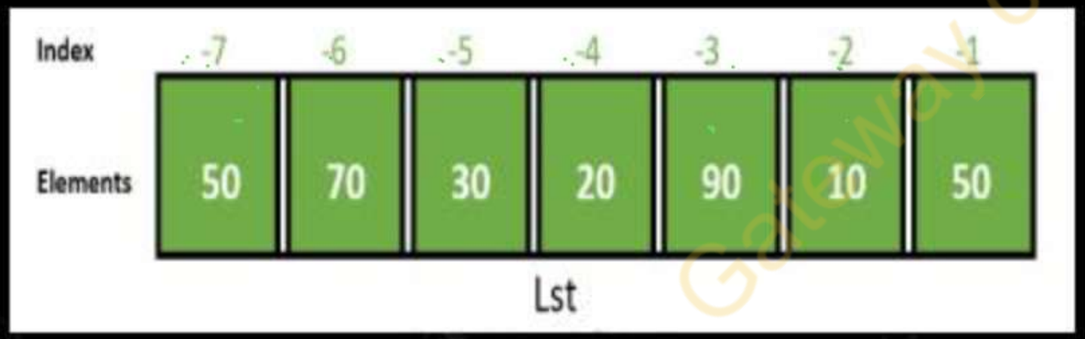
LIST SLICING

In order to access a range of elements in a list, you need to slice a list. One way to do this is to use the simple slicing operator i.e. colon(:).

Lst[Initial : End : IndexJump]

positive





TO PRINT THE COMPLETE LIST

```
1 Lst = [50, 70, 30, 20, 90, 10, 50]
2 print(Lst[::])
3
```

```
[50, 70, 30, 20, 90, 10, 50]
```

```
1 Lst = [50, 70, 30, 20, 90, 10, 50]
2 print(Lst[-7::1])
```

```
[50, 70, 30, 20, 90, 10, 50]
```

PRINT ONLY 2,3,4 ,5 ELEMENT

```
1 Lst = [50, 70, 30, 20, 90, 10, 50]
2 print(Lst[1:5])
```

```
[70, 30, 20, 90]
```

1 Print from 4 element of the list then take jump of 2 upto the end of the list.

2 Print every second list from the starting of the list.

```
1 List = [1, 2, 3, 4, 5, 6, 7, 8, 9]
2 print(List[3:9:2])
3 print(List[::2])
4 print(List[1::2])
```

[2, 4, 6, 8]

```
[4, 6, 8]
[1, 3, 5, 7, 9]
```

Write a function to find ASCII value of the character.

```
1 def ascii_value(a):
2     print(ord(a))
3 ascii_value("A")
```

```
65
```

Write a program to reverse a list

```
1 my_list = [1, 2, 3, 4, 5]
2 reversed_list = my_list[::-1]
3 print(reversed_list)
4
```

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

List comprehension (AKTU 2022-23)

It is used to create a new list from the existing list

Syntax: newList = **[** expression(element) **for** element **in** oldList **if** condition **]**

```
numbers = [12, 13, 14,]
doubled = [x *2 for x in numbers]
print(doubled)
```

```
[24, 26, 28]
```

```
list = [i+3 for i in range(11) if i % 2 == 0]
print(list)
```

```
[3, 5, 7, 9, 11, 13]
```

Question:- Write a function lessthan (lst,k) to return list of number less than k from the list lst. The function mnust use list comprehension Lessthan([1,-2,0,5,-3],0) returns[-2.-3] (AKTU 2020-21)

```
def lessthan(lst,k):
    return[i for i in lst if i<k]
print(lessthan([1,-2,0,5,-3],0))
```

```
[-2, -3]
```

Illustrate different list slicing constructs for the following operations on the following list:L= [1, 2, 3, 4, 5, 6, 7, 8, 9]

(AKTU 2023-24 ODD)

1. Return a list of numbers starting from the last to second item of the list
2. Return a list that start from 3rd item to second last item.
3. Return a list that has only even position elements of list L.
4. Return a list that starts from the middle of the list L.
5. Return a list that reverses all the elements starting from element at index 0 to middle index only and return the entire list.
6. Divide each element of the list by 2 and replace it with the remainder.

main.py

```
#1 solution
l = [1, 2, 3, 4, 5, 6, 7, 8, 9]
result = l[::-1][:-1]
print(result)
#1 alternanting solution
sliceL = l[-1:-3:-1]
print(sliceL)
#2 solution
result2 = l[2:-1]
print(result2)
#3 solution
result3 = l[0::2]
print(result3)
#4 solution
middle_index = len(l) // 2
result4 = l[middle_index:]
print(result4)
```

```
[9, 8, 7, 6, 5, 4, 3, 2]
[9, 8]
[3, 4, 5, 6, 7, 8]
[1, 3, 5, 7, 9]
[5, 6, 7, 8, 9]
```

lustrate different list slicing constructs for the following operations on the following list:L= [1, 2, 3, 4, 5, 6, 7, 8, 9]

1. Return a list of numbers starting from the last to second item of the list
2. Return a list that start from 3rd item to second last item.
3. Return a list that has only even position elements of list L to list M.
4. Return a list that starts from the middle of the list L.
5. Return a list that reverses all the elements starting from element at index 0 to middle index only and return the entire list.
6. Divide each element of the list by 2 and replace it with the remainder.

```
#5 solution
l=[1,2,3,4,5,6,7,8,9]
middle_index=len(l)//2
result5=l[:middle_index+1][::-1]+l[middle_index+1:]
print(result5)
#6 solution
result6=[x%2 for x in l]
print(result6)
```

input

```
[5, 4, 3, 2, 1, 6, 7, 8, 9]
[1, 0, 1, 0, 1, 0, 1, 0, 1]
```

Write a function make Pairs that takes as input two lists of equal length and return a single list of same length where kth element is the pair of k element from the input list

For example

Make Pairs([1,,3,4,5],[2,3,4,9])

Return[(1,2),(3,3),(4,4),(5,9)]

Make Pairs([],[])

Return[] (AKTU 2020-21)

```python
def makePairs(list1,list2):
    x=len(list1)
    y=len(list2)
    if x==y:
        pairlist=[(list1[i],list2[i]) for i  in range(0,x)]
    return pairlist

print(makePairs([1,3,5,7],[2,4,6,8]))
```

input

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

Write python function, alternating(lst) that takes as a argument a sequence list. The function return true if the elements in list are alternatively odd and even starting with even number otherwise it return false (AKTU 2020-21)

```python
def alternating(lst):
    if len(lst)==0:
        return True
    elif len(lst)==1:
        if lst[0]%2==0:
            return True
        return False
    elif lst[0]%2==0:
        for i in range(len(lst)-1):
            if lst[i]%2==lst[i+1]%2:
                return False
            return True
    else:
        return False
print(alternating([]))
print(alternating([10]))
print(alternating([7]))
print(alternating([10,7,8]))
print(alternating([10,8,8]))
```

```
True
True
False
True
False
```

String

the Updating or deletion of characters from a String is not allowed.

String are immutable

```
1  String1 = "gatewayclasses"
2  print("Initial String: ", String1)
3  #string slicing
4  print( String1[-3])# negative index
5  print(String1[3])#positive index
6  print(String1[1:3])
7  String1[0]="k"
8
```

```
Initial String:  gatewayclasses
s
e
at
Traceback (most recent call last):
  File "/home/main.py", line 7, in <module>
    String1[0]="k"
TypeError: 'str' object does not support item assignment
```

Len() It can be used to find the length of an object

```
1  list1 = ['hi','for','3444',2022]
2  print(len(list1))
3  tuple1 = (1,2,3,4)
4  print(len(tuple1))
5  string1 = "python"
6  print(len(string1))
7
8
```

```
4
4
6
```

Concatenation

.

```
1  str1="hello"
2  str2="students"
3  print(str1+str2)
4
```

```
hellostudents
```

Consider following filter (AKTU 2023-24 ODD)

1 filter all the numbers

```
def filter_number(text):
    return[word for word in text if word.isdigit()]
text=["apple","123","agra","banana", "Ramesh", "1234","patna","orange"]
print(filter_number(text))
```

input

```
['123', '1234']
```

2 filter all the string starting with vowels

```
def filter_number(text):
    vowels={"a","e","i","o","u","A","E","I","O","U"}
    return[word for word in text if word[0] in vowels]

text=["apple","123","agra","banana", "Ramesh", "1234","patna","orange"]
print(filter_number(text))
```

input

```
['apple', 'agra', 'orange']
```

filters all the strings that contains any of the following noun:
agra ,Ramesh , tomato, Patna

main.py

```
1  def filter_number(text):
2      noun={"agra","banana", "Ramesh","patna"}
3      return[word for word in text if any( noun in word for noun in noun)]
4
5  text=["apple","123","agra","banana", "Ramesh", "1234","patna","orange"]
6  print(filter_number(text))
```

input

```
['agra', 'banana', 'Ramesh', 'patna']
```

String manipulation methods/ function(AKTU 2022-23)

1 capitalize() **this** method returns a copy of the original string and converts the first character of the string to a capital **(uppercase)** letter while making all other characters in the string **lowercase** letters.

```
name = "draco MALFOy"
print(name.capitalize())
```

```
Draco malfoy
```

2. Lower() it converts all letters of a string to lowercase. If no uppercase characters exist, it returns the original string.

```
string = "ConvErT ALL tO LOWErCASe"
print(string.lower())
```

```
convert all to lowercase
```

3. Numeric() It is used to determine whether the string consists of numeric characters or not.

```
string = "123456789"
result = string.isnumeric()
print(result)
string1 = "1h2k3456k78"
result1 = string1.isnumeric()
print(result1)
```

```
True
False
```

4 Upper() method converts all lowercase characters in a string into uppercase characters and returns it.

```
original_text = "lisT Upper"
upper_text = original_text.upper()
print(upper_text)
```

```
LIST UPPER
```

5 **center() method** creates and returns a new string that is padded with the specified character.

```
1 string = "python"
2 new_string = string.center(12, "#" )
3 print("After padding String is: ", new_string)
4
```

input

```
After padding String is:  ###python###
```

6 **count()** function returns the number of occurrences of a substring within a String.

```
1 my_string = "Apple"
2 char_count = my_string.count('A')
3 print(char_count)
4
```

```
1
```

.7 endswith() method returns True if a string ends with the given suffix, otherwise returns False

```
1 string = "python"
2 print(string.endswith("on"))
3
```

```
True
```

8 find() method returns the lowest index or first occurrence of the substring if it is found in a given string.

```
1 word = 'me find me if you can me'
2 print(word.find('me'))
```

```
0
```

9. isalpha() method is used to check whether all characters in the String are an alphabet.

```
1  string = "gateway"
2  print(string.isalpha())
3  string = "gateway11"
4  print(string.isalpha())
5
```

```
True
False
```

10 decimal() function returns true if all characters in a string are decimal, else it returns False.

```
1  print("100".isdecimal())
```

```
True
```

11 isalnum() method checks whether all the characters in a given string are either alphabet or numeric (alphanumeric) characters

```
1  string = "abc 123"
2  print(string, "is alphanumeric?", string.isalnum())
3
```

```
abc 123 is alphanumeric? False
```

```
1  string1="abs1"
2  print(string1.isalnum())
```

```
True
```

12 **isupper()** method returns whether all characters in a _string are uppercase or not

```
1 print(("python").isupper())
2 print(("PYTH").isupper())
3
```

```
False
True
```

islower() method returns whether all characters in a _string are lower or not

```
1 print(("python").islower())
2 print(("PYTH").islower())
3
```

```
True
False
```

Write a program that accepts a sentence and calculate the number of digit, uppercase and lowercase (AKTU 2021-22)

```python
def string_test(s):
    d = {"UPPER_CASE": 0, "LOWER_CASE": 0,"DIGIT":0}
    for c in s:
        if c.isupper():
            d["UPPER_CASE"] += 1
        elif c.islower():
            d["LOWER_CASE"] += 1
        elif c in range(0,10):
            d["DIGIT"] += 1
        else:
            pass

    print("Original String: ", s)

    print("No. of Upper case characters: ", d["UPPER_CASE"])

    print("No. of Lower case Characters: ", d["LOWER_CASE"])
    print("No. of digit: ", d["DIGIT"])

string_test('The quick Brown Fox')

```

input

```
Original String:  The quick Brown Fox
No. of Upper case characters:  3
No. of Lower case Characters:  13
No. of digit:  0
```

Write a program to change a given string to a new string where the first and last chars have been changed(AKTU 2021-22)

main.py

```python
def change_string(str1):
    return str1[-1:] + str1[1:-1] + str1[:1]
print(change_string('abcd'))

```

```
dbca
```

Write a program to count the vowels present in given input . (AKTU 2022-23)

```python
def vowel_count(str):
    count = 0
    vowel = set("aeiouAEIOU")
    for alphabet in str:
        if alphabet in vowel:
            count = count + 1
    print("No. of vowels :", count)

str = "pythonprogramming"
vowel_count(str)
```

```
No. of vowels : 4
```

Write a program to check whether a string is a palindrome or not (AKTU 2022-23)

```python
def ispalindrome(s):
    return s==s[::-1]
k=input("enter the string\n")
ans=ispalindrome(k)
if ans:
    print("yes plaindrome")
else:
    print("no palindrome")
```

```
enter the string
malayalam
yes plaindrome
```

How can you randomize the items of a list in place in python ? (AKTU 2021-22)

```
import random
test_list = [1, 4, 5, 6, 3]
print("The original list is : " + str(test_list))
random.shuffle(test_list)
print("The shuffled list is : " + str(test_list))

```

input

```
The original list is : [1, 4, 5, 6, 3]
The shuffled list is : [6, 4, 1, 5, 3]
```

What will be the output

```
def cube(x):
    return x*x*x
x=cube(3)
print(x)
```

```
27
```

Write a program that accepts sequence of lines as input and print the lines after making all character in the sentence is capitalized(AKTU 2022-23)

```
lines = []
while True:
    l = input()
    if l:
        lines.append(l.upper())
    else:
        break;
for l in lines:
    print(l)
```

```
hello world
pratices makes perfect

HELLO WORLD
PRATICES MAKES PERFECT
```

Consider a function perfect_square(number) that returns a number if it is perfect square otherwise return-1 (AKTU 2023-24 ODD)

```python
import math
def perfect_square(number):
    z= math.sqrt(number)
    if z*z==number:
        return number
    else:
        return-1
print(perfect_square(4))
print(perfect_square(2))
```

```
4
-1
```

Split()

```python
text= 'Split this string'

# splits using space
print(text.split())

grocery = 'Milk, Chicken, Bread'

# splits using ,
print(grocery.split(', '))

grocery1 = 'Milk:Chic:ken, Bread'
print(grocery1.split(':'))
```

```
['Split', 'this', 'string']
['Milk', 'Chicken', 'Bread']
['Milk', 'Chic', 'ken, Bread']
```

Construct a program that accepts a comma separated sequence of word as an input and prints the word in a comma- separated sequence after sorting them alphabetically(AKTU 2023-24 ODD)

```
1 items = input("Input comma-separated sequence of words\n")
2 words = [word for word in items.split(",")]
3 print(",".join(sorted(list(set(words)))))
4
```

```
Input comma-separated sequence of words
without,hello,bag,world
bag,hello,without,world
```

What will be the output. (AKTU 2022-23)

```
1 list1=['M','N','K','Y']
2 print("@".join(list1))
3
```

```
M@N@K@Y
```

Final result is string

Write a program to find permutation of a given string.

main.py

```python
from itertools import permutations

def find_permutations(string):
    perm_list = list(permutations(string))
    perm_strings = [''.join(perm) for perm in perm_list]
    return perm_strings
input_string = "abc"
permutations_list = find_permutations(input_string)
print(permutations_list)
```

input

```
['abc', 'acb', 'bac', 'bca', 'cab', 'cba']
```

What will be the output ? (AKTU 2022-23)

main.py

```python
def count1(s):
    vowels="AEIOUaeiou"
    count=0
    for c in s:
        if c in vowels:
            count=count+1
    return count
print(count1("i love India"))
```

```
6
```

Question.1 Which of the following statement produce an error in python?

statement 1: x,y,z=1,2,3

statement 2: a,b=4,5,6

statement 3: u=7,8,9(AKTU 2020-21)

Statement 1 give no error Statement 2 gives error because 6 is not assigned to any variable

Statement 3 gives error as 8,9 is not declared to any variable

Q2 Explain why this program generate error

X=['12','hello',456]

X[0]*=3

X[1][1]='bye'

X[0]*=3 (AKTU 2020-21) [2023-24 Odd]

It does not give any error . X[0] refers to the string '12'. Multiplying a string by an integer in Python repeats the string that many times. This updates X[0] to '121212', and now the list X becomes:

X = ['121212', 'hello', 456]

X[1][1]='bye'

Here, X[1] refers to the string 'hello', and X[1][1] refers to the character at index 1 of 'hello', which is 'e'. The intention seems to be to replace the character 'e' with the string 'bye'. However, this line generates an error because strings in Python are immutable, meaning their individual characters cannot be changed.

Construct a function ret_smaller(l) that return smaller list from a nested list. if two list have same length then return the first list that is encountered (AKTU 2023-24 ODD)

```python
def ret_smaller(l):
    # If the list is empty, return an empty list
    if not l:
        return []

    # Start by assuming the first sublist is the smallest
    smallest_list = l[0]

    # Iterate over the rest of the sublists
    for sublist in l[1:]:
        # If a smaller sublist is found, update smallest_list
        if len(sublist) < len(smallest_list):
            smallest_list = sublist

    # Return the smallest sublist found
    return smallest_list

# Example usage
nested_list = [[3, 4, 5], [1, 2], [6, 7, 8, 9], [10]]
print(ret_smaller(nested_list))
```

input

```
[10]
```

TUPLES(AKTU 2022-23)

- Tuples are similar as list but only difference is that they are immutable

Creating Python Tuples

- Using round brackets

```
var = ("hello", "for", "bye")
print(var)
print(type(var))
```

```
('hello', 'for', 'bye')
<class 'tuple'>
```

- With one item

```
mytuple = ("hi",)
print(type(mytuple))
#NOT a tuple
mytuple = ("hi")
print(type(mytuple))
```

```
<class 'tuple'>
<class 'str'>
```

- Tuple Constructor

```
tuple_constructor = tuple(("dsa", "developement", "deep learning"))
print(tuple_constructor)
```

What is Immutable in Tuples?

- Tuples in Python are similar to Python list but not entirely. Tuples are immutable and ordered and allow duplicate values. Some Characteristics of Tuples in Python.
- We can find items in a tuple since finding any item does not make changes in the tuple.
- One cannot add items to a tuple once it is created.
- Tuples cannot be appended or extended.
- We cannot remove items from a tuple once it is created.

```python
mytuple = (1, 2, 3, 4, 5)

# tuples are indexed
print(mytuple[1])
print(mytuple[4])

# tuples contain duplicate elements
mytuple = (1, 2, 3, 4, 2, 3)
print(mytuple)

# adding an element
mytuple[1] = 100
print(mytuple)
```

input

```
2
5
(1, 2, 3, 4, 2, 3)
Traceback (most recent call last):
  File "/home/main.py", line 12, in <module>
    mytuple[1] = 100
TypeError: 'tuple' object does not support item assignment
```

Unpacking a Tuple (AKTU 2023-24 ODD)

In Python, there is a very powerful tuple assignment feature that assigns the right-hand side of values into the left-hand side. In another way, it is called unpacking of a tuple of values into a variable. In packing, we put values into a new tuple while in unpacking we extract those values into a single variable.

Tuple unpacking in Python allows you to assign elements of a tuple to multiple variables in a single statement. This is useful when you have a tuple and want to extract its values directly into separate variables

```python
# this lines PACKS values
# into variable a
a = ("MNNIT Allahabad", 5000, "Engineering")

# this lines UNPACKS values
# of variable a
(college, student, type_ofcollege) = a

# print college name
print(college)

# print no of student
print(student)

# print type of college
print(type_ofcollege)
```

```
MNNIT Allahabad
5000
Engineering
```

Operation on tuples

```
# 1concatenation
tuple1 = (0, 1, 2, 3)
tuple2 = ('python', 'unit2')
print(tuple1 + tuple2)
#nesting
tuple1 = (0, 1, 2, 3)
tuple2 = ('python', 'hi')
tuple3 = (tuple1, tuple2)
print(tuple3)
#3 repetition
tuple3 = ('python',)*3
print(tuple3)
# deletion is not allowed

```

```
(0, 1, 2, 3, 'python', 'unit2')
((0, 1, 2, 3), ('python', 'hi'))
('python', 'python', 'python')
```

main.py

```
# slicing
tuple1 = (0 ,1, 2, 3)
print(tuple1[1:])
print(tuple1[::-1])
print(tuple1[2:4])


```

```
1, 2, 3)
3, 2, 1, 0)
2, 3)
```

Accessing Values in Python Tuples

- **Positive index**

```
var = ("hi", "for", "bye")
print("Value in Var[0] = ", var[0])
print("Value in Var[1] = ", var[1])
print("Value in Var[2] = ", var[2])
```

- **Negative index**

```
var = (1, 2, 3)
print("Value in Var[-1] = ", var[-1])
print("Value in Var[-2] = ", var[-2])
```

DICTIONARY

- Dictionaries in Python is a data structure, used to store values in key : value format.
- A dictionary can be created by placing a sequence of elements within curly {} braces, separated by a 'comma'

```
1 Dict = {1: 'PYTHON', 2: 'LIST', 3: 'TUPLE'}
2 print(Dict)
```

```
{1: 'PYTHON', 2: 'LIST', 3: 'TUPLE'}
```

```python
Dict = {}
print("Empty Dictionary: ")
print(Dict)
Dict[0] = 'HI'
Dict[2] = 'For'
Dict[3] = 1
print("\nDictionary after adding 3 elements: ")
print(Dict)
Dict[2] = 'Welcome'
print("\nUpdated key value: ")
print(Dict)
Dict['name']='python'
Dict['unit']=3
Dict['topic']='dictionary'
print ("agaian updated")
print(Dict)
print(Dict['name'])
```

```
Empty Dictionary:
{}

Dictionary after adding 3 elements:
{0: 'HI', 2: 'For', 3: 1}

Updated key value:
{0: 'HI', 2: 'Welcome', 3: 1}
```

```
agaian updated
{0: 'HI', 2: 'Welcome', 3: 1, 'name': 'python', 'unit': 3, 'topic': 'dictionary'}
python
```

Python Dictionary Method (AKTU 2022-23)

Functions Name	Descriptions
clear()	Removes all items from the dictionary
copy()	Returns a shallow copy of the dictionary
fromkeys()	Creates a dictionary from the given sequence
get()	Returns the value for the given key

Functions Name	Descriptions
items()	Return the list with all dictionary keys with values
keys()	Returns a view object that displays a list of all the keys in the dictionary in order of insertion
pop()	Returns and removes the element with the given key
popitem()	Returns and removes the key-value pair from the dictionary
setdefault()	Returns the value of a key if the key is in the dictionary else inserts the key with a value to the dictionary
values()	Updates the dictionary with the elements from another dictionary
update()	Returns a list of all the values available in a given dictionary

clear()

```
text = {1: "g", 2: "for"}
text.clear()
print('text =', text)
```

```
text = {}
```

copy()

main.py

```
original = {1: 'g', 2: 'for'}
new = original.copy()
new.clear()
print('new: ', new)
print('original: ', original)
```

```
new:  {}
original:  {1: 'g', 2: 'for'}
```

```
seq = {'a', 'b', 'c', 'd', 'e'}

# creating dict with default values as None
res_dict = dict.fromkeys(seq)

print("The newly created dict with None values : " + str(res_dict))

# creating dict with default values as 1
res_dict2 = dict.fromkeys(seq, 1)

print("The newly created dict with 1 as value : " + str(res_dict2))
```

input

```
The newly created dict with None values : {'c': None, 'b': None, 'd': None, 'a': None, 'e': None}
The newly created dict with 1 as value : {'c': 1, 'b': 1, 'd': 1, 'a': 1, 'e': 1}
```

```python
d = {'coding': 'good', 'thinking': 'better'}
print(d.get('coding'))

```

```
good
```

```python
Dictionary1 = {'A': 'python', 'B': 'For', 'C': 'python'}
print(Dictionary1.keys())
```

```
dict_keys(['A', 'B', 'C'])
```

```python
student = {"rahul":7, "Aditya":1, "Shubham":4}
print(student)
suspended = student.pop("rahul")
print("suspended student roll no. = ",suspended)
print("remaining student", student)

```

```
{'rahul': 7, 'Aditya': 1, 'Shubham': 4}
suspended student roll no. =  7
remaining student {'Aditya': 1, 'Shubham': 4}
```

```python
d = {1: '001', 2: '010', 3: '011'}
print(d.popitem())
```

```
(3, '011')
```

```python
d = {'a': 97, 'b': 98}
print("setdefault() returned:", d.setdefault('b', 99))
print("After using setdefault():", d)
```

```
setdefault() returned: 98
After using setdefault(): {'a': 97, 'b': 98}
```

```python
d = {'a': 97, 'b': 98, 'c': 99, 'd': 100}
# space key added using setdefault() method
d.setdefault(' ', 32)
print(d)
```

```
{'a': 97, 'b': 98, 'c': 99, 'd': 100, ' ': 32}
```

```python
dictionary = {"raj": 2, "striver": 3, "vikram": 4}
print(dictionary.values())
```

```
dict_values([2, 3, 4])
```

Append() AKTU (2022-23)	Extend()
Append single element at the end of the list	Append multiple elements fro, an iterable at the end of the list
Accepts single element as an argument	Accept an iterable (e.g. list,tuple, string) as an argument
List AKTU(2022-23)	**tuples**
List are mutable	Tuples are immutable
List are usually slower than tuples	They are faster than list
list1=[10,20,30]	Tup1=(10,20,30)

main.py

```
1
2  Dictionary1 = {'A': 'hello', 'B': 'For', }
3  Dictionary2 = {'B': 'bye'}
4
5  # Dictionary before Updation
6  print("Original Dictionary:")
7  print(Dictionary1)
8
9  # update the value of key 'B'
10 Dictionary1.update(Dictionary2)
11 print("Dictionary after updation:")
12 print(Dictionary1)
13
```

```
Original Dictionary:
{'A': 'hello', 'B': 'For'}
Dictionary after updation:
{'A': 'hello', 'B': 'bye'}
```

main.py

```
1  # Dictionary with three items
2  Dictionary1 = { 'A': 'hello', 'B': 4, 'C': 'bye' }
3
4  print("Dictionary items:")
5
6  # Printing all the items of the Dictionary
7  print(Dictionary1.items())
8
```

inpu

```
Dictionary items:
dict_items([('A', 'hello'), ('B', 4), ('C', 'bye')])
```

LIST (AKTU 2022-23)	DICTIONARY
The list is a collection of index value pairs like that of the array in C++.	The dictionary is a hashed structure of **the key and value** pairs.
The list is created by placing elements in **[ ]** separated by commas “,”	The dictionary is created by placing elements in **{ }** as “key”:”value”, each key-value pair is separated by commas “, “
The elements are accessed via indices.	The elements are accessed via key-value pairs.
The order of the elements entered is maintained.	There is no guarantee for maintaining order.
Lists are orders, mutable, and can contain duplicate values.	Dictionaries are unordered and mutable but they cannot contain duplicate keys.

Array AKTU 2019-20)	list
Arrays have a fixed size set during creation.	Lists are dynamic and can change in size during runtime.
All elements in an array must be of the same data type.	Lists can accommodate elements of different data types.
Memory for the entire array is allocated at once during initialization.	Lists dynamically allocate memory as elements are added or removed.
Arrays are less flexible as their size cannot be easily changed.	Lists are more flexible, allowing easy addition or removal of elements.

What will bet the output ? AKTU(2021-22)

main.py

```
def printalpha(abc_list, num_list):
    for char in abc_list:
        for num in num_list:
            print(char,num)
printalpha(["a","b","c"],[1,2,3])
```

input

```
a 1
a 2
a 3
b 1
b 2
b 3
c 1
c 2
c 3
```

Write a python function to convert a decimal number its binary ,octal and hexadecimal.

main.py

```
def bin_oct_hex(a):
    print(bin(a))
    print(oct(a))
    print(hex(a))
bin_oct_hex(10)
```

```
0b1010
0o12
0xa
```

Write a python function removekth (s,k) that takes as input a string and integer k>=0 and remove the character at index k. if k is beyond the length of s, the whole of s is returned

For example

removekth("python",2) return pyhon

Removekth("mango",1)return mngo

AKTU (2023-24 ODD)

AKTU (2020-21)

main.py

```
def removekth(s,k):
    str=""
    for i in range(len(s)):
        if i!=k:
            str=str+s[i]
    return str
print(removekth("mango",1))
print(removekth("mango",4))
print(removekth("mango",16))
```

```
mngo
mang
mango
```

Write a program factor(N) that return a list of all positive divisor of N (N>=1). AKTU (2020-21)

```python
def factor(num):
    flist=[]
    if num>=1:
        for i in range(1,num+1):
            if num%i==0:
                flist.append(i)
    return flist
print(factor(6))
```

```
[1, 2, 3, 6]
```

Write a python program to add item in a tuple. AKTU (2019-20)

```python
#first method
t=(10,20,40,70,90)
print(t)
t=t+(20,)
print(t)
#second method
l=list(t)
l.append(3)
t=tuple(l)
print(t)
```

```
(10, 20, 40, 70, 90)
(10, 20, 40, 70, 90, 20)
(10, 20, 40, 70, 90, 20, 3)
```

Write a python code to find out occurrence of an elements in a list.

```
1  vowels=['a','a','e','i','a','o','u','u']
2  print("count of a",vowels.count('a'))
3  print("count of e",vowels.count('e'))
4  print("count of i",vowels.count('i'))
5  print("count of o",vowels.count('o'))
6  print("count of u",vowels.count('u'))
```

input

```
count of a 3
count of e 1
count of i 1
count of o 1
count of u 2
```

Write a python program ,count_square(N) that returns the count of perfect square less than or equal to N(N>=1) for example count_square(1) return 1 only prefect square less than equal to 1

count_square(5) return 2 (1, 4 are the prefect square less than equal to 2

count_square(55) return 7 (1,4,9,16,25,36,49)<=55 AKTU (2020-21)

```python
import math
def count_square(N):
    if N <= 1:
        return 0
    # Find the largest integer whose square is less than or equal to N
    largest_integer = int(math.sqrt(N))
    # The number of perfect squares less than or equal to N is the same
    # as the largest integer whose square is less than or equal to N
    return largest_integer
N = 55
print("count of perfect square less than or eqaul to",count_square(N))
```

input

```
count of perfect square less than or eqaul to 7
```

Ques. Write a python function, searchMany(s,x,k) that takes an argument a sequence s and integersx,k(k>0) the function returns true if there are k occurrences of x in s .otherwise it returns false searchMany ([10,17,15,12],15,1) return true, searchMany([10,12,12,12],12,2) return falseAKTU (2020-21)

```
1  def searchMany(s,x,k):
2      count=0
3      for i in s:
4          if i==x:
5              count=count+1
6      if count==k;
7          return True
8      else:
9          return False
10
11 print(searchMany([10,17,15,12],15,1))
12 print(searchMany([10,17,15,12],17,3))
```

```
True
False
```

A website requires the users to input username and password to register. Construct a program to check the validity of password input by users. Following are the criteria for checking the password

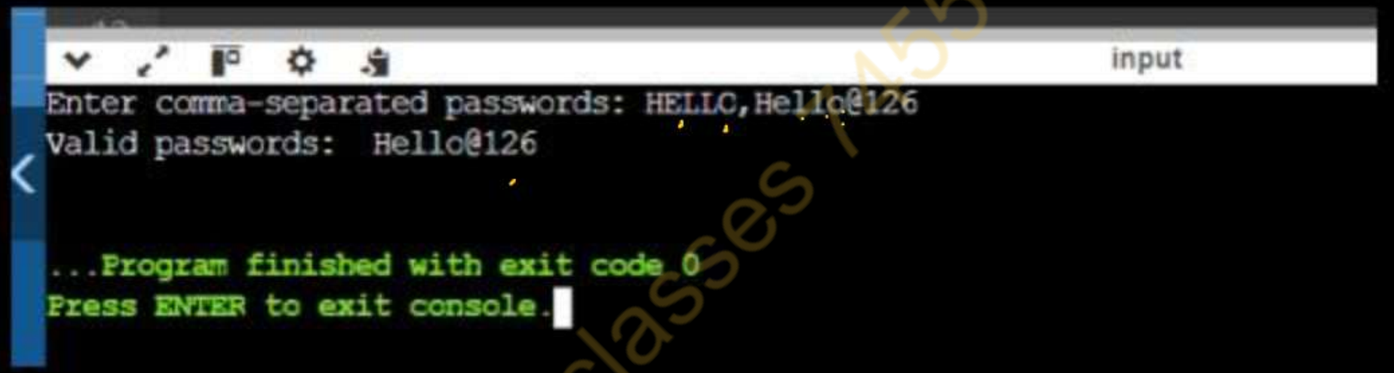
(Aktu 2023-24 ODD

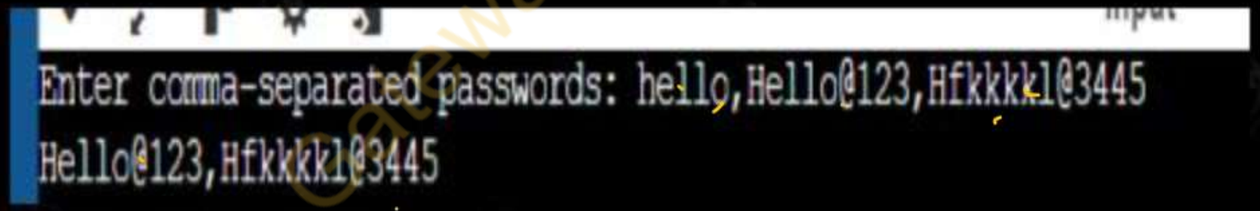
1. At least 1 letter between [a-z]
2. At least 1 number between [0-9]
3. At least 1 letter between [A-Z]
4. At least 1 character from [$#@]
5. Minimum length of transaction password: 6
6. Maximum length of transaction password: 12
7. Your program should accept a sequence of comma separated passwords and will check them according to the above criteria. Passwords that match the criteria are to be printed, each separated by a comma

main.py

```python
import re

def is_valid_password(password):
    # Check if the password meets all the criteria
    if (6 <= len(password) <= 12 and
        re.search(r'[a-z]', password) and
        re.search(r'[0-9]', password) and
        re.search(r'[A-Z]', password) and
        re.search(r'[$#@]', password)):
        return True
    return False

def check_passwords(passwords):
    # Split the input string into individual passwords
    password_list = passwords.split(',')
    # Filter valid passwords
    valid_passwords = [password for password in password_list if is_valid_password(password)]
    # Join the valid passwords with a comma and return
    return ','.join(valid_passwords)

# Example usage
passwords = input("Enter comma-separated passwords: ")
valid_passwords = check_passwords(passwords)
print("Valid passwords:", valid_passwords)
```

The import re statement in Python is used to import the re module, which provides support for working with regular expressions. Regular expressions (regex) are patterns used to match character combinations in strings. The re module allows you to perform tasks such as searching, matching, and replacing text using these patterns.





FUNCTION

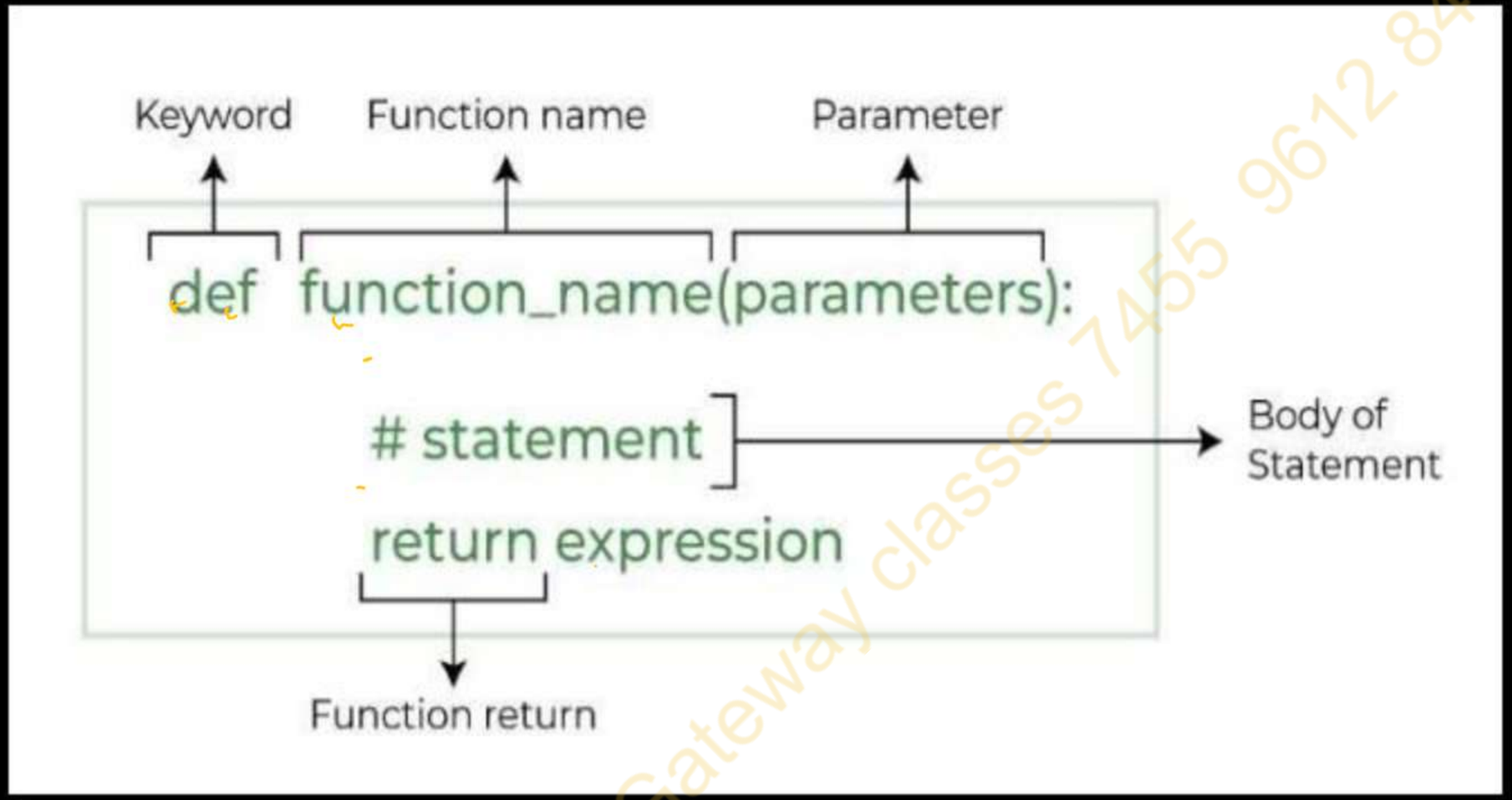
- Python Functions is a block of statements that return the specific task. The idea is to put some commonly or repeatedly done tasks together and make a function so that instead of writing the same code again and again for different inputs, we can do the function calls to reuse code contained in it over and over again.
- **Some Benefits of Using Functions**
- Increase Code Readability
- Increase Code Reusability

Types of Functions in Python

There are mainly two types of functions in Python

- Built-in library function: These are Standard functions in Python that are available to use.
- User-defined function: We can create our own functions based on our requirements.

Python Function Declaration



Python Function

- Syntax of function definition :
- def function name(arg 1, arg 2):

{

.....

}

Syntax of function call

Function_name(arg1,arg2)

Python Function

- **Keyword:** the keyword 'def' is used to define function header.
- **Function name:** we define the function name for identification or to uniquely identify the function.
- **(:)** a colon to mark the end of function header.
- **Arguments** these are the values passed to the function between parentheses
- **Body of the function:** the body process the argument to do something use ful
- An optional **return statement** to return the value from the function.
- **function call** To execute a function , we have to call it

Creating a Function in Python

- We can define a function in Python, using the def keyword. We can add any type of functionalities and properties to it as we require.

```
# A simple Python function
def fun():
    print("Welcome ")


# Driver code to call a function
fun()
```

```
Welcome

...Program finished with exit code 0
Press ENTER to exit console.
```

Python Function with Parameters

- If you have experience in C/C++ or Java then you must be thinking about the return type of the function and data type of arguments. That is possible in Python as well (specifically for Python 3.5 and above).
- Defining and calling a function with parameters
- def function_name(parameter: data_type) -> return_type:
- """Docstring"""
- # body of the function
- return expression.

Python Function with Parameters

```python
def add(num1: int, num2: int) -> int:
    """Add two numbers"""
    num3 = num1 + num2

    return num3

# Driver code
num1, num2 = 5, 15
ans = add(num1, num2)
print(f"The addition of {num1} and {num2} results {ans}.")

```

input

```
The addition of 5 and 15 results 20.


...Program finished with exit code 0
Press ENTER to exit console.
```

Types of Python Function Arguments AKTU (2022-23)

- Python supports various types of arguments that can be passed at the time of the function call. In Python, we have the following 4 types of function arguments.
- Default argument
- Keyword arguments (named arguments)
- Positional arguments
- Arbitrary arguments (variable-length arguments *args and **kwargs)

Default Arguments

- A default argument is a parameter that assumes a default value if a value is not provided in the function call for that argument

```
1  # Python program to demonstrate
2  # default arguments
3  def myFun(x, y=50):
4      print("x: ", x)
5      print("y: ", y)
6
7
8  # Driver code
9  myFun(10)
10 myFun(11,20)
11
12
```

input

```
: 10
: 50
: 11
: 20
```

Keyword Arguments

- The idea is to allow the caller to specify the argument name with values so that the caller does not need to remember the order of parameters.

```python
# Python program to demonstrate Keyword Arguments
def student(firstname, lastname):
    print(firstname, lastname)


# Keyword arguments
student(firstname='python', lastname='Practice')
student(lastname='Practice', firstname='python')
```

input

```
python Practice
python Practice


...Program finished with exit code 0
Press ENTER to exit console.
```

Keyword Arguments

- Keyword-only arguments mean whenever we pass the arguments(or value) by their parameter names at the time of calling the function in Python in which if you change the position of arguments then there will be no change in the output.
- Benefits of using Keyword arguments over positional arguments
- On using keyword arguments you will get the correct output because the order of argument doesn't matter provided the logic of your code is correct. But in the case of positional arguments, you will get more than one output on changing the order of the arguments..

Positional Arguments

- Position-only arguments mean whenever we pass the arguments in the order we have defined function parameters in which if you change the argument position then you may get the unexpected output. We should use positional Arguments whenever we know the order of argument to be passed.
- So now, we will call the function by using the position-only arguments in two ways, and In both cases, we will be getting different outputs from which one will be correct and another one will be incorrect

- We used the Position argument during the function call so that the first argument (or value) is assigned to name and the second argument (or value) is assigned to age. By changing the position, or if you forget the order of the positions, the values can be used in the wrong places, as shown in the Case-2 example below, where 27 is assigned to the name and Suraj is assigned to the age

Positional Arguments

```python
def nameAge(name, age):
    print("Hi, I am", name)
    print("My age is ", age)


# You will get correct output because
# argument is given in order
print("Case-1:")
nameAge("Suraj", 27)
# You will get incorrect output because
# argument is not in order
print("\nCase-2:")
nameAge(27, "Suraj")
```

input

```
Hi, I am Suraj
My age is  27

Case-2:
Hi, I am 27
My age is  Suraj
```

Arbitrary Keyword Arguments

(Variable length of arguments

- In Python Arbitrary Keyword Arguments, *args, and **kwargs can pass a variable number of arguments to a function using special symbols. There are two special symbols:
- *args in Python (Non-Keyword Arguments)
- **kwargs in Python (Keyword Arguments)

*args in Python (Non-Keyword Arguments)

- We can declare a variable length length argument with the*symbol
- The *args allows a function to accept any number of positional arguments
- The arguments are collected as a tuple within the function

*args in Python (Non-Keyword Arguments)

main.py

```
# beginning of the method
def sum(*args):
    resultfinal = 0
#beginning of for loop
    for num in args:
        resultfinal = resultfinal + num

    return resultfinal
#printing the values
print(sum(10, 20))              # 30
print(sum(10, 20, 30))          # 60
print(sum(10, 20, 2))           # 32
```

```
30
60
32
```

***kwargs in Python (Keyword Arguments)

- The *kwargs parameter allows a function to accept any number of keyword arguments.
- The arguments are collected as a dictionary within the function

```python
# Let's write a Python program
# *kwargs for a variable number of keyword arguments

def myPrg(**kwargs):
    for key, value in kwargs.items():
        print (key, "==", value)

# Driver code for kwargs in python
myPrg(first ='Hello', mid ='Welcome', last='Hello')
```

input

```
first == Hello
mid == Welcome
last == Hello

...Program finished with exit code 0
Press ENTER to exit console.
```

Calculator using function AKTU (2019-20)

main.py

```
# Python program for simple calculator

# Function to add two numbers
def add(num1, num2):
    return num1 + num2

# Function to subtract two numbers
def subtract(num1, num2):
    return num1 - num2

# Function to multiply two numbers
def multiply(num1, num2):
    return num1 * num2

# Function to divide two numbers
def divide(num1, num2):
    return num1 / num2

print("Please select operation -\n" \
      "1. Add\n" \
      "2. Subtract\n" \
      "3. Multiply\n" \
      "4. Divide\n")
```

Calculator using function

```
# Take input from the user
select = int(input("Select operations form 1, 2, 3, 4 :"))

number_1 = int(input("Enter first number: "))
number_2 = int(input("Enter second number: "))

if select == 1:
    print(number_1, "+", number_2, "=",
                    add(number_1, number_2))

elif select == 2:
    print(number_1, "-", number_2, "=",
                    subtract(number_1, number_2))

elif select == 3:
    print(number_1, "*", number_2, "=",
                    multiply(number_1, number_2))

elif select == 4:
    print(number_1, "/", number_2, "=",
                    divide(number_1, number_2))
else:
    print("Invalid input")
```

Calculator using function

```
Please select operation -
1. Add
2. Subtract
3. Multiply
4. Divide

Select operations form 1, 2, 3, 4 :2
Enter first number: 7
Enter second number: 8
7 - 8 = -1


...Program finished with exit code 0
Press ENTER to exit console.
```

Python Scope variable AKTU (2019-20)

- The location where we can find a variable and also access it if required is called the scope of a variable
- **Python Local variable**
- Local variables are those that are initialized within a function and are unique to that function. It cannot be accessed outside of the function

```
1
2  def f():
3
4      # local variable
5      s = "I love India"
6      print(s)
7
8
9  # Driver code
10 f()
11
```

```
I love India
```

Python Scope variable

- Python Local variable
- If we will try to use this local variable outside the function

```
1  def f():
2
3      # local variable
4      s = "I love python"
5      print("Inside Function:", s)
6
7  # Driver code
8  f()
9  print(s)
10
```

input

```
Inside Function: I love python
Traceback (most recent call last):
  File "/home/main.py", line 9, in <module>
    print(s)
NameError: name 's' is not defined


...Program finished with exit code 1
Press ENTER to exit console.
```

Python Scope variable

- **Python Global variables**
- **Global variables are the ones that are defined and declared outside any function and are not specified to any function. They can be used by any part of the program.**

```
1  # This function uses global variable s
2  def f():
3      print(s)
4
5
6  # Global scope
7  s = "I love python"
8  f()
9
10
```

input

```
I love python

...Program finished with exit code 0
Press ENTER to exit console.
```

Thank You