CMR
INSTITUTE OF
TECHNOLOGY
USN

Internal Assessment Test 1 – December 2022

Sub:	**Data Analytics using Python**							**Sub Code:**	**20MCA31**
Date:	**26/12//2022**	**Duration:**	**90 min's**	**Max Marks:**	**50**	**Sem:**	**III**	**Branch:**	**MCA**

Note: Answer FIVE FULL Questions, choosing ONE full question from each Module

	PART I	MARKS	OBE CO	OBE RBT
1(a)	Write the features of Python. Give the advantages & disadvantages of it.	5	CO1	L2
(b)	Write a python program to check and print given list contains even numbers or not.	5	CO2	L3
	OR			
2(a)	Discuss the Looping Statements Syntax and example.	5	CO1	L2
(b)	Write any three differences between a list, tuples and dictionary ?	5	CO2	L4
	PART II			
3(a)	How to declare and call functions in python program? Illustrate with an example script.	5	CO1	L2
(b)	Write a Python program to check given string is palindrome or not	5	CO3	L3
	OR			
4	Develop a python program to calculate the area of square, rectangle, and circle using function **Solution:** ***Example***: **Input:** shape name = "Rectangle" length = 10 breadth = 15 **Output:** Area: 150 ***Example***: **Input:** shape name = "Square" side = 10 **Output:** Area: 100	[10]	CO3	L4
5	Explain in detail about python operators with examples. Discuss the precedence and associativity of the operators	10	CO2	L4
	OR			
6 a)	Illustrate args and kwargs parameters in python programming language with an example	[5]	CO2	L2
b)	Write python program to perform linear search	[5]	CO2	L4

	PART IV			
7 a)	How to create dictionary in python. Explain five methods with a brief description with example	[5]	CO2	L2
b)	Explain any five string functions with syntax and example.	[5]	CO2	L2
	OR			
8 a)	Write python program to find the Sum of tuple elements using looping through tuple.	[3]		
b)	Discuss the following tasks on tuple with syntax and example i) append ii) Add two tuples iii) remove item from tuple iv) multiply tuples by 3 v) Count method in tuple	[7]	CO2	L3 L4
	PART V			
9 a)	How to create Constructors, inheritance and operator overloading in Python? Demonstrate with an example.	[5]	CO2	L3
b)	Write a Python program to sort elements using Selection Sort.	[5]	CO2	L3
	OR			
10	Write Python program to count words and store in dictionary for the given input text **Input Text** : the clown ran after the car and the car ran into the tent and the tent fell down on the clown and the car **Output :** word count : {'and': 3, 'on': 1, 'ran': 2, 'car': 3, 'into': 1, 'after': 1, 'clown': 2, 'down': 1, 'fell': 1, 'the': 7, 'tent': 2}	[10]	CO2	L5

ANSWER KEY
Internal Assessment Test 1 – Dec 2022

Sub:	**Data Analytics using Python**							**Sub Code:**	**20MCA31**
Date:	**26/12//2022**	**Duration:**	**90 min's**	**Max Marks:**	**50**	**Sem:**	**III**	**Branch:**	**MCA**

1. **(a) Write the features of Python. Give the advantages & disadvantages of it. (5 Marks)**

Features

- **Object oriented language : .** Python supports object-oriented language and concepts of classes, objects encapsulation, etc.
- **Interpreted language** : Python code is executed line by line at a time.
- **Supports dynamic data type :** A variable is decided at run time not in advance. hence, we don't need to specify the type of variable. (for example- int, double, long, etc.)
- **Simple and easy to code :** Python is very easy code
- **High-level Language**
- **Automatic memory management**
- **open source:** Python language is freely available

advantages & disadvantages

advantages

- Free availability (like Perl, Python is open source).
- Stability (Python is in release 2.6 at this point and, as I noted earlier, is older than Java).
- Very easy to learn and use
- Good support for objects, modules, and other reusability mechanisms.
- Easy integration with and extensibility using C and Java.

disadvantages

- Smaller pool of Python developers compared to other languages, such as Java
- Lack of true multiprocessor support
- Absence of a commercial support point, even for an Open Source project (though this situation is changing)
- Software performance slow, not suitable for high performance applications

(b) Write a Python program to check and print given list contains even numbers or not **(5 Marks)**

```
def contains_even_number(l):
  for ele in l:
    if ele % 2 == 0:
      print ("list contains an even number")
      break

  # This else executes only if break is NEVER
  # reached and loop terminated after all iterations.
  else:
    print ("list does not contain an even number")
```

```
# Driver code
print ("For List 1:")
contains_even_number([1, 9, 8])
print (" \nFor List 2:")
contains_even_number([1, 3, 5])
```

output:

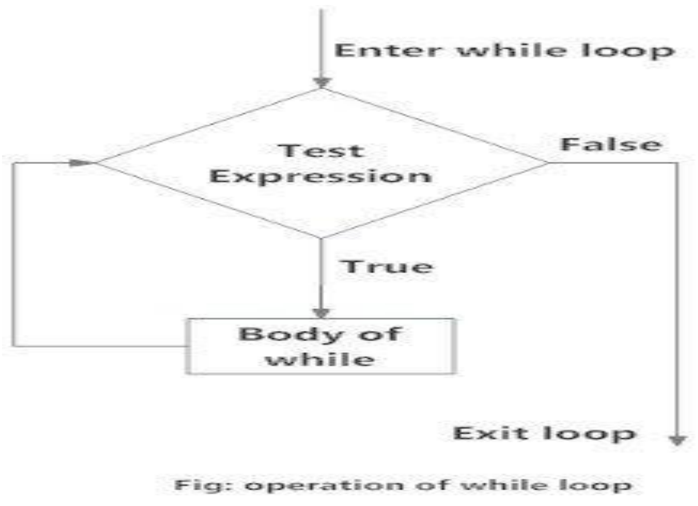
```
For List 1:
list contains an even number

For List 2:
list does not contain an even number
```

2. **Discuss the Looping Statements with syntax and example. (5 Marks)**
   (i) while (ii) for

The while loop in Python is used to iterate over a block of code as long as the test expression (condition) is true.
We generally use this loop when we don't know the number of times to iterate beforehand.
A "for" loop also involves a line ending in a colon, and an indented block:

```
x = 1
while x < 100:
    print(x)
    x = x + x ** 2
print('the loop ended when x hit', x)
output
1
2
6
42
the loop ended when x hit 1806
```

Flowchart of while Loop



for loop

The for loop in Python is used to iterate over a sequence (list, tuple, string) or other iterable objects. Iterating over a sequence is called traversal.
Here, val is the variable that takes the value of the item inside the sequence on each iteration.
Loop continues until we reach the last item in the sequence. The body of for loop is separated

from the rest of the code using indentation.

Flowchart of for Loop

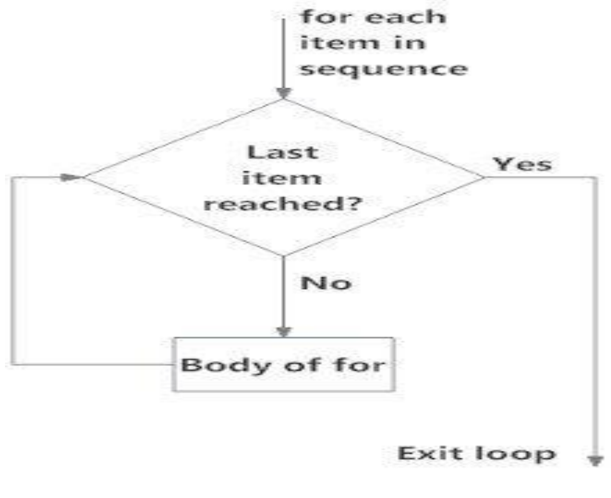


Fig: operation of for loop

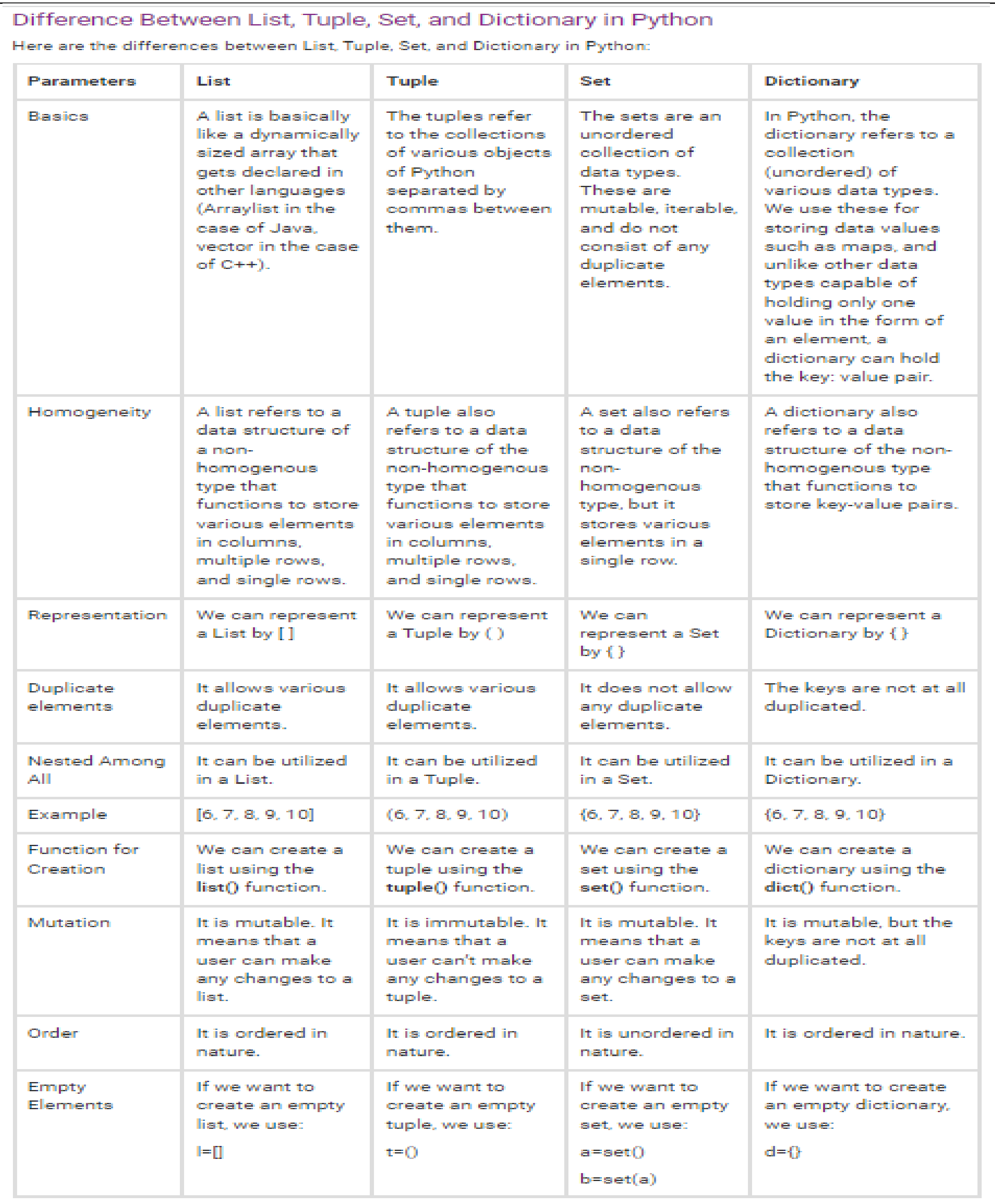
In [3]:

```
months = ["January", "February", "March", "April", "May"]
for m in months:
    print(m[:3])
Jan
Feb
Mar
Apr
May
```

In [4]:

```
for x in range(10, 50, 10):
    print("x is %d so x**2 is %d" % (x, x ** 2))

output
x is 10 so x**2 is 100
x is 20 so x**2 is 400
x is 30 so x**2 is 900
x is 40 so x**2 is 1600
```

3. (a) What is the difference between a list and a tuple? Give an example (5 Marks)



Difference Between List, Tuple, Set, and Dictionary in Python

Here are the differences between List, Tuple, Set, and Dictionary in Python:

Parameters	List	Tuple	Set	Dictionary
Basics	A list is basically like a dynamically sized array that gets declared in other languages (Arraylist in the case of Java, vector in the case of C++).	The tuples refer to the collections of various objects of Python separated by commas between them.	The sets are an unordered collection of data types. These are mutable, iterable, and do not consist of any duplicate elements.	In Python, the dictionary refers to a collection (unordered) of various data types. We use these for storing data values such as maps, and unlike other data types capable of holding only one value in the form of an element, a dictionary can hold the key: value pair.
Homogeneity	A list refers to a data structure of a non-homogenous type that functions to store various elements in columns, multiple rows, and single rows.	A tuple also refers to a data structure of the non-homogenous type that functions to store various elements in columns, multiple rows, and single rows.	A set also refers to a data structure of the non-homogenous type, but it stores various elements in a single row.	A dictionary also refers to a data structure of the non-homogenous type that functions to store key-value pairs.
Representation	We can represent a List by [ ]	We can represent a Tuple by ( )	We can represent a Set by { }	We can represent a Dictionary by { }
Duplicate elements	It allows various duplicate elements.	It allows various duplicate elements.	It does not allow any duplicate elements.	The keys are not at all duplicated.
Nested Among All	It can be utilized in a List.	It can be utilized in a Tuple.	It can be utilized in a Set.	It can be utilized in a Dictionary.
Example	[6, 7, 8, 9, 10]	(6, 7, 8, 9, 10)	{6, 7, 8, 9, 10}	{6, 7, 8, 9, 10}
Function for Creation	We can create a list using the **list()** function.	We can create a tuple using the **tuple()** function.	We can create a set using the **set()** function.	We can create a dictionary using the **dict()** function.
Mutation	It is mutable. It means that a user can make any changes to a list.	It is immutable. It means that a user can't make any changes to a tuple.	It is mutable. It means that a user can make any changes to a set.	It is mutable, but the keys are not at all duplicated.
Order	It is ordered in nature.	It is ordered in nature.	It is unordered in nature.	It is ordered in nature.

Empty Elements	If we want to create an empty list, we use:
	l=[]
		If we want to create an empty tuple, we use:
		t=()
			If we want to create an empty set, we use:
			a=set()
			b=set(a)
				If we want to create an empty dictionary, we use:
				d={}

(b) Write a Python program to check given string is palindrome or not (5 Marks)

```
string=input(("Enter a letter:"))
if(string==string[::-1]):
    print("The letter is a palindrome")
else:
    print("The letter is not a palindrome")
```

Output:

```
Enter a letter: CMRIT

The letter is not a palindrome
```

```
Enter a letter: MADAM

The letter is a palindrome
```

4. **Develop a python program to calculate the area of a square, rectangle, and circle using the function**

   Solution:

   Example:

   Input: shape name = "Rectangle"

   length = 10

   breadth = 15

   Output: Area: 150

   Example:

   Input: shape name = "Square"

   side = 10

   Output: Area: 100

Solution

```
    def calculate_area(name):

      # converting all characters
      # into lower cases
      name = name.lower()

      # check for the conditions
      if name == "rectangle":
        l = int(input("Enter rectangle's length: "))
        b = int(input("Enter rectangle's breadth: "))

        # calculate area of rectangle
        rect_area = l * b
        print(f"The area of rectangle is {rect_area}.")

      elif name == "square":
        s = int(input("Enter square's side length: "))

        # calculate area of square
        sqt_area = s * s
        print(f"The area of square is  {sqt_area}.")

      elif name == "triangle":
        h = int(input("Enter triangle's height length: "))
        b = int(input("Enter triangle's breadth length: "))

        # calculate area of triangle
        tri_area = 0.5 * b * h
        print(f"The area of triangle is {tri_area}.")

      elif name == "circle":
        r = int(input("Enter circle's radius length: "))
        pi = 3.14
        # calculate area of circle
        circ_area = pi * r * r
        print(f"The area of circle is {circ_area}.")

      elif name == 'parallelogram':
        b = int(input("Enter parallelogram's base length: "))
        h = int(input("Enter parallelogram's height length: "))

        # calculate area of parallelogram
        para_area = b * h
        print(f"The area of parallelogram is {para_area}.")

      else:
        print("Sorry! This shape is not available")

    # main code
    if __name__ == "__main__" :

      print("Calculate Shape Area")
      shape_name = input("Enter the name of shape whose area you want to
    find: ")

      # function calling
      calculate_area(shape_name)
```

Output

```
Calculate Shape Area

Enter the name of shape whose area you want to find: rectangle

Enter rectangle's length: 10

Enter rectangle's breadth: 15

The area of rectangle is 150.
```

5. **Explain in detail about python operators (10 Marks)**

Python Operators in general are used to perform operations on values and variables. These are standard symbols used for the purpose of logical and arithmetic operations. In this article, we will look into different types of Python operators.

Arithmetic Operators

Arithmetic operators are used to performing mathematical operations like addition, subtraction, multiplication, and division.

Operator	Description	Syntax
+	Addition: adds two operands	x + y
–	Subtraction: subtracts two operands	x – y
*	Multiplication: multiplies two operands	x * y

/	Division (float): divides the first operand by the second	x / y
//	Division (floor): divides the first operand by the second	x // y
%	Modulus: returns the remainder when the first operand is divided by the second	x % y
**	Power: Returns first raised to power second	x ** y

Example: Arithmetic operators in Python

```
# Examples of Arithmetic Operator
a = 9
b = 4

# Addition of numbers
add = a + b

# Subtraction of numbers
sub = a - b

# Multiplication of number
mul = a * b

# Division(float) of number
div1 = a / b

# Division(floor) of number
div2 = a // b

# Modulo of both number
mod = a % b

# Power
p = a ** b
```

Comparison Operators
Comparison of Relational operators compares the values. It either returns **True** or **False** according to the condition.

Operator	Description	Syntax
>	Greater than: True if the left operand is greater than the right	x > y
<	Less than: True if the left operand is less than the right	x < y

==	Equal to: True if both operands are equal	x == y
!=	Not equal to – True if operands are not equal	x != y
>=	Greater than or equal to True if the left operand is greater than or equal to the right	x >= y
<=	Less than or equal to True if the left operand is less than or equal to the right	x <= y

Example: Comparison Operators in Python

```
# Examples of Relational Operators
a = 13
b = 33

# a > b is False
print(a > b)

# a < b is True
print(a < b)

# a == b is False
print(a == b)

# a != b is True
print(a != b)

# a >= b is False
print(a >= b)

# a <= b is True
print(a <= b)
```

Logical operators perform **Logical AND**, **Logical OR**, and **Logical NOT** operations. It is used to combine conditional statements.

Operator	Description	Syntax
and	Logical AND: True if both the operands are true	x and y
or	Logical OR: True if either of the operands is true	x or y
not	Logical NOT: True if the operand is false	not x

Example: Logical Operators in Python

```
# Examples of Logical Operator
```

```
a = True
b = False

# Print a and b is False
print(a and b)

# Print a or b is True
print(a or b)

# Print not a is False
print(not a)
```

Output

```
False
True
False
```

Bitwise Operators

Bitwise operators act on bits and perform the bit-by-bit operations. These are used to operate on binary numbers.

Operator	Description	Syntax
&	Bitwise AND	x & y
|	Bitwise OR	x | y
~	Bitwise NOT	~x
^	Bitwise XOR	x ^ y
>>	Bitwise right shift	x>>
<<	Bitwise left shift	x<<

Example: Bitwise Operators in Python

```
# Examples of Bitwise operators
a = 10
b = 4

# Print bitwise AND operation
print(a & b)

# Print bitwise OR operation
print(a | b)

# Print bitwise NOT operation
print(~a)
```

```
# print bitwise XOR operation
print(a ^ b)

# print bitwise right shift operation
print(a >> 2)

# print bitwise left shift operation
print(a << 2)
```

Output

```
0
14
-11
14
2
40
```

Assignment Operators

Assignment operators are used to assigning values to the variables.

Operator	Description	Syntax
=	Assign value of right side of expression to left side operand	x = y + z
+=	Add AND: Add right-side operand with left side operand and then assign to left operand	a+=b a=a+b
-=	Subtract AND: Subtract right operand from left operand and then assign to left operand	a-=b a=a-b
*=	Multiply AND: Multiply right operand with left operand and then assign to left operand	a*=b a=a*b
/=	Divide AND: Divide left operand with right operand and then assign to left operand	a/=b a=a/b
%=	Modulus AND: Takes modulus using left and right operands and assign the result to left operand	a%=b a=a%b
//=	Divide(floor) AND: Divide left operand with right operand and then assign the value(floor) to left operand	a//=b a=a//b
**=	Exponent AND: Calculate exponent(raise power) value using operands and assign value to left operand	a**=b a=a**b
&=	Performs Bitwise AND on operands and assign value to left	a&=b

	operand	a=a&b
|=	Performs Bitwise OR on operands and assign value to left operand	a|=b a=a|b
^=	Performs Bitwise xOR on operands and assign value to left operand	a^=b a=a^b
>>=	Performs Bitwise right shift on operands and assign value to left operand	a>>=b a=a>>b
<<=	Performs Bitwise left shift on operands and assign value to left operand	a <<= b a= a << b

Example: Assignment Operators in Python

```
# Examples of Assignment Operators
a = 10

# Assign value
b = a
print(b)

# Add and assign value
b += a
print(b)

# Subtract and assign value
b -= a
print(b)

# multiply and assign
b *= a
print(b)

# bitwise lishift operator
b <<= a
print(b)
```

Output

```
10
20
10
100
102400
```

Identity Operators

is and **is not** are the identity operators both are used to check if two values are located on the same part of the memory. Two variables that are equal do not imply that they are identical.

is True if the operands are identical
is not True if the operands are not identical
Example: Identity Operator

```
a = 10
b = 20
c = a

print(a is not b)
print(a is c)
```

Output
```
True
True
```
Membership Operators
in and **not in** are the membership operators; used to test whether a value or variable is in a sequence.

in	True if value is found in the sequence
not in	True if value is not found in the sequence

Example: **Membership** Operator

```
# Python program to illustrate
# not 'in' operator
x = 24
y = 20
list = [10, 20, 30, 40, 50]

if (x not in list):
print("x is NOT present in given list")
else:
print("x is present in given list")

if (y in list):
print("y is present in given list")
else:
print("y is NOT present in given list")
```

Output
```
x is NOT present in given list
y is present in given list
```
Precedence and Associativity of Operators

6 (a) Write python program to illustrate variable length keyword arguments (5 Marks)
*args and **kwargs are mostly used in function definitions. *args and **kwargs allow you to pass an unspecified number of arguments to a function, so when writing the function definition, you do not need to know how many arguments will be passed to your function. *args is used to send a **non-keyworded** variable length argument list to the function. Here's an example to help you get a clear idea:

```
def test_var_args(f_arg, *argv):
    print("first normal arg:", f_arg)
    for arg in argv:
        print("another arg through *argv:", arg)

test_var_args('yasoob', 'python', 'eggs', 'test')
```

This produces the following result:

```
first normal arg: yasoob
another arg through *argv: python
another arg through *argv: eggs
another arg through *argv: test
```

I hope this cleared away any confusion that you had. So now let's talk about **kwargs

1.2. Usage of **kwargs

kwargs allows you to pass **keyworded variable length of arguments to a function. You should use **kwargs if you want to handle **named arguments** in a function. Here is an example to get you going with it:

```
def greet_me(**kwargs):
    for key, value in kwargs.items():
        print("{0} = {1}".format(key, value))

>>> greet_me(name="yasoob")
name = yasoob
```

So you can see how we handled a keyworded argument list in our function. This is just the basics of **kwargs and you can see how useful it is. Now let's talk about how you can use *args and **kwargs to call a function with a list or dictionary of arguments.

1.3. Using *args and **kwargs to call a function

So here we will see how to call a function using *args and **kwargs. Just consider that you have this little function:

```
def test_args_kwargs(arg1, arg2, arg3):
    print("arg1:", arg1)
    print("arg2:", arg2)
    print("arg3:", arg3)
```

Now you can use *args or **kwargs to pass arguments to this little function. Here's how to do it:

```
# first with *args
>>> args = ("two", 3, 5)
>>> test_args_kwargs(*args)
arg1: two
arg2: 3
arg3: 5

# now with **kwargs:
>>> kwargs = {"arg3": 3, "arg2": "two", "arg1": 5}
>>> test_args_kwargs(**kwargs)
arg1: 5
arg2: two
arg3: 3
```

Order of using *args **kwargs and formal args

So if you want to use all three of these in functions then the order is

```
some_func(fargs, *args, **kwargs)
```

(b) Write python program to perform linear search (5 Marks)

```
def search(arr, n, x):

   for i in range(0, n):
      if (arr[i] == x):
         return i
   return -1

# Driver Code
arr = [10,50,30,70, 80, 60, 20, 90,40]
x = 20
n = len(arr)

# Function call
result = search(arr, n, x)
if(result == -1):
   print("Element is not present in array")
else:
   print("Element is present at index", result)
```

7. A) How to create dictionary in python. Explain five methods with a brief description with example

Dictionaries are used to store data values in key:value pairs, can be referred to by using the key name.
A dictionary is a collection which is ordered*, changeable and do not allow duplicates.
Dictionary items are ordered, changeable, and does not allow duplicates.

Method#1

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

Method #2
# dict() Constructor

```
thisdict = dict(name = "John", age = 36, country = "Norway")
print(thisdict)
```

Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value

get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

b) Explain any 5 string functions with examples (5 Marks)

- **Upper() Method - Converts a string into lower case**

```
a = "Hello, World!"
print(a.upper())
```

- **Lower() method - Converts a string into upper case**

```
a = "Hello, World!"
print(a.lower())
```

- The strip() method removes any whitespace from the beginning or the **end**

```
a = " Hello, World! "
a.strip() # returns "Hello, World!“
```

- The replace() method replaces a string with another string

```
a = "Hello, World!"
a.replace("H", "J")
```

- The split() method splits the string into substrings if it finds instances of the separator

```
a = "Hello, World!"
print(a.split(",")) # returns ['Hello', ' World!']
```

- startswith(...)

```
|    S.startswith(prefix[, start[, end]]) -> bool
|
|    Return True if S starts with the specified prefix, False otherwise.
|    With optional start, test S beginning at that position.
|    With optional end, stop comparing S at that position.
|    prefix can also be a tuple of strings to try.
```
.

8. **a) Write python program to find the Sum of tuple elements using looping through tuple.**
(5 Marks)

```
## Sum of tuple elements
def summation(t):
  # converting into list
    test = list(t)

    # initializing count
    count = 0

    # for loop
    for i in test:
        count += i
    return count


# initializing test_tup
x = (5, 20, 3, 7, 6, 8)
print("The original tuple is :", x)
print("Summation of tuple is :", summation(x))
```

Output:

```
The original tuple is : (5, 20, 3, 7, 6, 8)
Summation of tuple is : 49
```

b) b) Discuss the following tasks on tuple with syntax and example
i) append ii) Add two tuples iii) remove item from tuple iv) multiply tuples by 3 v) Count method in tuple

i) ## *append*

```
thistuple = ("apple","banana","cherry")
y=list(thistuple)
y.append("orange")
thistuple=tuple(y)
print(thistuple)
```

output:

```
('apple', 'banana', 'cherry', 'orange')
```

ii) ***Add two tuples***

```
thistuple = ("apple","banana","cherry")
y = ("orange",)
thistuple += y
print(thistuple)
```

Output:

```
('apple', 'banana', 'cherry', 'orange')
```

iii) ***Remove item from tuples***

```
thistuple=("apple","banana","cherry")
y=list(thistuple)
y.remove("apple")
thistuple=tuple(y)
print(thistuple)
```

Output:

```
('banana', 'cherry')
```

iv) ***# Multiply the fruits tuple by 3:***

```
fruits = ("apple", "banana", "cherry")
mytuple = fruits * 3

print(mytuple)
```

Output:

```
('apple', 'banana', 'cherry', 'apple', 'banana', 'cherry', 'apple', 'banana', 'cherry')
```

v) *Sytac -- tuple.count(value)*
vi)

```
thistuple = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)

x = thistuple.count(5)

print(x)
```

Output:

```
2
```

9 a) Write a python program using object oriented programming to demonstrate encapsulation, overloading and inheritance (10 Marks)

```
class Base:
   def __init__(self):
     self.a = 10
     self._b = 20

   def display(self):
      print(" the values are :")
      print(f"a={self.a} b={self._b}")

class Derived(Base):                          # Creating a derived class
   def __init__(self):
     Base.__init__(self)                   # Calling constructor of Base class
     self.d = 30

   def display(self):
     Base.display(self)
```

```
        print(f"d={self.d}")

    def __add__(self, ob):
        return self.a + ob.a+self.d + ob.d
        #return self.a + ob.a+self.d + ob.d+self.b + ob.b

obj1 = Base()
obj2 = Derived()
obj3 = Derived()

obj2.display()
obj3.display()

print("\n Sum of two objects :",obj2 + obj3)
```

b) Write a Python program to sort elements using Selection Sort.

```
###selection sort

import numpy as np

def selection_sort(x):
   for i in range(len(x)):
       swap = i + np.argmin(x[i:])
       (x[i], x[swap]) = (x[swap], x[i])
   return x

## mian code
x = np.array([2, 1, 4, 3, 5])
selection_sort(x)
```

Output:

```
                                                                        Out[1]:
array([1, 2, 3, 4, 5])
```

10. Write Python program to count words and store in dictionary for the given input text
Input Text : the clown ran after the car and the car ran into the tent and the tent fell down on the clown and the car
Output : word count : {'and': 3, 'on': 1, 'ran': 2, 'car': 3, 'into': 1, 'after': 1, 'clown': 2, 'down': 1, 'fell': 1, 'the': 7, 'tent': 2}

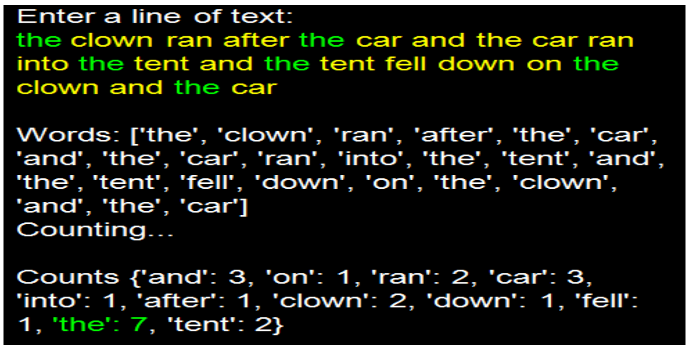
Method 1

```
counts = dict()
line = input('Enter a line of text:')
words = line.split()
print('Words:', words)
print('Counting...’)

for word in words:
    counts[word] = counts.get(word,0) + 1
print('Counts', counts)
```

Method 2

```
def word_count(str):
  counts = dict()
  words = str.split()
  for word in words:
     if word in counts:
        counts[word] += 1
     else:
        counts[word] = 1
  return counts

#Driver Code
print( word_count(' the clown ran after the car and the car ran into the tent and the tent fell down on the clown and the car’))
```

Output:



```
Enter a line of text:
the clown ran after the car and the car ran
into the tent and the tent fell down on the
clown and the car

Words: ['the', 'clown', 'ran', 'after', 'the', 'car',
'and', 'the', 'car', 'ran', 'into', 'the', 'tent', 'and',
'the', 'tent', 'fell', 'down', 'on', 'the', 'clown',
'and', 'the', 'car']
Counting...

Counts {'and': 3, 'on': 1, 'ran': 2, 'car': 3,
'into': 1, 'after': 1, 'clown': 2, 'down': 1, 'fell':
1, 'the': 7, 'tent': 2}
```