Unit1- Introduction and Syntax of python Program

Winter-23

Q 1 (a) Enlist applications for python programming. (W-23) 2 Marks Ans:

Applications for python programming are -:

- Desktop GUI Applications
- Image Processing Applications
- Scientific and Numeric Applications
- Audio and Video Based Applications
- 3D CAD Applications
- Google's App Engine web development framework uses Python as an application language.
- Maya, a powerful integrated 3D modeling and animation system, provides a Python scripting API.
- Linux Weekly News, published by using a web application written in Python.
- Google makes extensive use of Python in its Web Search Systems.
- The popular YouTube video sharing service is largely written in Python programming.
- The NSA uses Python for cryptography and intelligence analysis.
- iRobot uses Python programming to develop commercial and military robotic devices.
- The Raspberry Pi single-board computer promotes Python programming as its educational language.

- Nextflix and Yelp have both documented the role of Python in their software infrastructures.
- Industrial Light and Magic, Pixar and others uses Python in the production of animated movies.

Q 1 (c) Describe the Role of indentation in python. (W-23) 2 Marks Ans:

- Indentation refers to the spaces at the beginning of a code line.
- Generally, four whitespaces are used for indentation and is preferred over tabs.
- Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.
- Indentation helps to convey a better structure of a program to the readers. It is used to clarify the link between control flow constructs such as conditions or loops, and code contained within and outside of them.
- Python uses indentation to indicate a block of code.

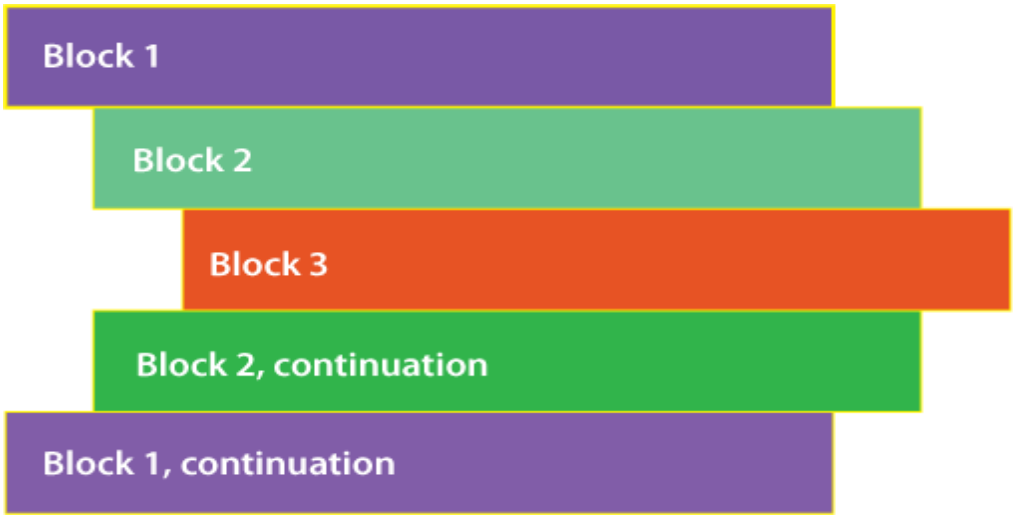


Figure of Indentation

- **Example:**

```
if 5 > 2:
        print("Five is greater than two!")
```

Q 3 a Explain building blocks of python. (W-23) 4 Marks Ans:

1) Python Identifiers:

- Variable name is known as identifier.

To name an identifier following are the rules:

- The first character of the variable must be an alphabet or underscore (_).
- All the characters except the first character may be an alphabet of lower-case(a-z), upper-case (A-Z), underscore or digit (0-9).
- Identifier name must not contain any white-space, or special character (!, @, #, %, ^, &, *).
- Identifier name must not be similar to any keyword defined in the language. o Identifier names are case sensitive for example my name, and MyName is not the same.
- **Example:**

  a,b,c=5,10,15

2) Reserved Words

- The following list shows the Python keywords.
- These are reserved words and cannot use them as constant or variable or any other identifier names.
- All the Python keywords contain lowercase letters only except True and False.

False	None	True	and	as
assert	break	class	continue	def
del	elif	else	except	finally
for	from	global	if	import
in	is	lambda	nonlocal	not
or	pass	raise	return	try
while	with	yield		

Figure of Reserved Words

3) Indentation:

- Python provides no braces to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation, which is compulsory.
- The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount.

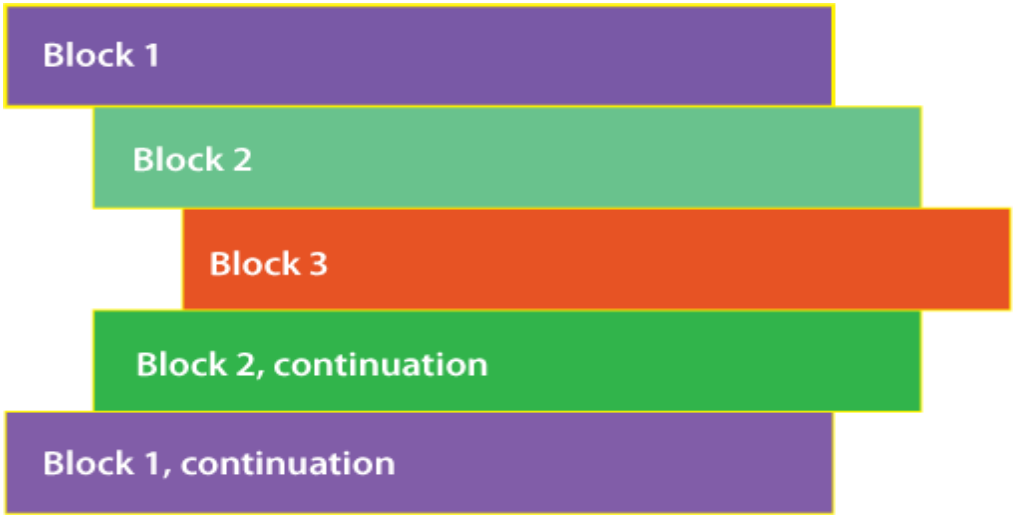


Figure of Indentation

- **Example:**

```
if True:
print "True" else:
print "False"
```

- Thus, in Python all the continuous lines indented with same number of spaces would form a block.

4) Python Data Types:

- The basic types in Python are String (str), Integer (int), Float (float), and Boolean (bool).
- There are also built in data structures to know when you learn Python.
- These data structures are made up of the basic types, you can think of them like Legos, the data structures are made out of these basic types.
- The core data structures to learn in Python are List (list), Dictionary (dict), Tuple (tuple), and Set (set).

Strings :

- Strings in Python are assigned with single or double quotations.
- As in many other programming languages, characters in strings may be accessed as if accessing an array.
- In the example below we'll assign a string to a variable, access the first element, check for a
- substring, and check the length of the string. x = 'abcd'

Numbers:

- Integers and Floats in Python are both Number types.
- They can interact with each other, they can be used in all four operations.
- In the example code we'll explore how these numbers can interact.

```
x = 2.5
y = 2
```

Boolean:

- Boolean variables in Python are either True or False.
- They will also return True for 1 and False for 0.
- The example shows how to assign either True or False to a variable in Python x = True

```
y = False
```

Lists:

- Lists in Python are represented with brackets.
- Like characters in a string, the elements in a list can be accessed with brackets.
- Lists can also be enumerated on to return both the index and the element.
- We'll go over enumerate when we cover for loops in Python.

- The example code shows how to declare lists, print elements in them, add to them, and remove from them.

```
x = [10, 25, 63, 104]
y = ['a', 'q', 'blah']
```

Dictionaries:

- Dictionaries in Python are a group of key-value pairs.
- Dictionaries are declared with curly braces and their entries can be accessed in two ways,
  a) with brackets, and
  b) with .get.
- The example code shows how we can access items in a dictionary. dict = { 'a': 'Sally sells sea shells', 'b': 'down by the seashore' }

Tuples:

- Tuples is an immutable sequence in Python. Unlike lists, you can't move objects out of order in a Tuple.
- Tuples are declared with parenthesis and must contain a comma (even if it is a tuple of 1).
- The example below shows how to add tuples, get a tuple from a list, and return information about it.

```
x = (a, b)
```

Sets:

- Sets in Python are the non-duplicative data structure.
- That means they can only store one of an element.
- Sets are declared with curly braces like dictionaries, but do not contain ':' in them.
- The example code shows how to turn a list into a set, access set elements by index, add to a set, and remove from a set.

```
# we can turn a list into a set x = ['a', 'a', 'b', 'c',
'c']
x = set(x)
```

5) Control structures:

- Control structures are used to determine the flow of execution of a Python program.
- Examples of control structures in Python include if-else statements, for and while loops, and try-except blocks.

6) Functions:

- Functions are reusable blocks of code that perform specific tasks.
- In Python, functions are defined using the def keyword.

7) Modules:

- Python modules are files that contain Python code and can be imported into other Python programs to reuse code and simplify development.

8) Packages:

- Packages are collections of related Python modules that can be installed and imported together.
- Packages are commonly used in Python for organizing and distributing libraries and tools.

Summer-23

Q 1 (a) List features of Python. 2 Marks Ans:

Features of Python are listed below:

- Easy to Learn and Use
- Interactive Mode
- Expressive Language
- Interpreted Language
- Cross-platform Language
- Portable
- Free and Open Source

- Object-Oriented Language
- Extensible
- Large Standard Library
- GUI Programming Support
- Integrated
- Databases
- Scalable

Q 1 (d) Describe any two data conversion function. (2 Marks) Ans:

- **int(x [,base]):**
  Converts x to an integer. base specifies the base if x is a string.
  Example:
  x=int('1100',base=2)=12
- **long(x [,base]):**
  Converts x to a long integer. base specifies the base if x is a string.
  Example:
  x=long('123'base=8)=83L
- **float(x):**
  Converts x to a floating point number.
  Example:
  x=float('123.45')=123.45
- **complex(real[,imag]) :**
  Creates a complex number.
  Example:
  x=complex(1,2) = (1+2j)
- **str(x):**
  Converts object x to a string representation.
  Example:
  x=str(10) = '10'
- **repr(x):**
  Converts object x to an expression string
  Example:
  x=repr(3) = 3
- **repr(x):**
  Evaluates a string and returns an object.
  Example:
  x=eval('1+2') = 3
- **tuple(s):**
  Converts s to a tuple
  Example:
  x=tuple('123') = ('1', '2', '3')
  x=tuple([123]) = (123,)
- **list(s):**
  Converts s to a list
  Example:
  x=list('123') = ['1', '2', '3']
  x=list(['12'] = ['12']
- **set(s):**
  Converts s to a set

Example:

x=set('Python')
{'y', 't', 'o', 'P', 'n', 'h'}

- **dict(d):**

Creates a dictionary. d must be a sequence of (key, value) tuples.

Example:

dict={'id':'11','name':'vijay'} print(dict)

Represents sequential data with a little difference from list.

- **Dictionary:**

Represents a collection of data that associate a unique key with each value.

- **Boolean:**

Represents truth-values (true or false).

1. Integers (int Data Type):

- An integer is a whole number that can be positive (+) or negative (−).
- Integers can be of any length, it is only limited by the memory available.
- **Example:** For number data types are integers.

>>>a=10
>>>b -10
To determine the type of a variable type() function is used.
>>>type(a)
>>> <class 'int'>

2. Boolean (Bool Data Type):

- The simplest build-in type in Python is the bool type, it
- represents the truth-values False and True. Internally the true value is represented as 1 and false is 0.
- **Example:** For example

>>>a = 18 > 5
>>>print(a) True b=2>3
print(b) False

3. Floating-Point/Float Numbers (Float Data Type):

- Floating-point number or Float is a positive or negative number with a fractional part.
- A floating point number is accurate up to 15 decimal places. Integer and floating points are separated by decimal points.
- 1 is integer, 1.0 is floating point number.
- **Example:** Floating point number.

x=10.1
type(x)
<class 'float'>

4. Complex Numbers (Complex Data Type):

- Complex numbers are written in the form, x + yj, where x is the real part and y is the imaginary part.
- **Example:**

Complex number.
>>>x = 3+4j
>>>print(x.real) 3.0
>>>print(x.imag) 4

5. String Data Type:

- String is a collection of group of characters.
- Strings are identified as a contiguous set of characters enclosed in single quotes (' ') or double quotes (" ").

- Any letter, a number or a symbol could be a part of the string.
- Strings are unchangeable (immutable). Once a string is created, it cannot be modified.
- **Example:** For string data type.

```
>>> s1="Hello" #string in double quotes
>>> s2='Hi' #string in single quotes
>>> s3="Don't open the door" #single quote string in double quotes
>>> s4='I said "yipee"' #double quote string in single quotes
>>>type(s1)
<class 'str'>
```

6. List Data Type:

- List is an ordered sequence of items.
- It is one of the most used datatype in Python and is very flexible.
- List can contain heterogeneous values such as integers, floats, strings, tuples, lists and
- dictionaries but they are commonly used to store collections of homogeneous objects.
- The list datatype in Python programming is just like an array that can store a group of
- elements and we can refer to these elements using a single name. Declaring a list is
- pretty straight forward. Items separated by commas (,) are enclosed within brackets [ ].
- **Example:** For list.

```
>>> first=[10, 20, 30] # homogenous values in list
>>> second=["One","Two","Three"] # homogenous values in list
>>> first [10, 20, 30]
>>> second
['One', 'Two', 'Three']
>>> first + second # prints the concatenated lists [10, 20, 30, 'One',
'Two', 'Three']
```

7. Tuple Data Type:

- Tuple is an ordered sequence of items same as list.
- The only difference is that tuples are immutable.
- Tuples once created cannot be modified. It is defined within parentheses ( ) where items are separated by commas (,).
- A tuple data type in python programming is similar to a list data type, which also contains heterogeneous items/elements.
- **Example:** For tuple.

```
>>> a=(10,'abc',1+3j)
>>> a
(10, 'abc', (1+3j))
>>> a[0]
```

Q 3 (a) List data types used in Python. Explain any two with example. 4 Marks Ans:

Same as Winter 2022-5(a)

4(d) Explain building blocks of python. (4M) Ans:

Same as Winter 2023-3(a)

Unit 2- Python Operators and Control Flow statements

Winter-23

Q 1 (b) Write the use of elif keyword in python. 2 marks Ans:

elif:

- elif stands for 'else if' and is used in Python programming to test multiple conditions.
- The if statements are executed from the top down.
- As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the ladder is bypassed.
- If none of the conditions is true, then the final else statement will be executed.

Q2 a Explain membership and identity operators in Python.(W-23) 4 Marks Ans:

Membership Operators:

- The membership operators in Python are used to find the existence of a particular element in the sequence, and used only with sequences like string, tuple, list, dictionary etc.
- Membership operators are used to check an item or an element that is part of a string, a list or a tuple.
- A membership operator reduces the effort of searching an element in the list.
- Python provides **'in'** and **'not in'** operators which are called membership operators and used to test whether a value or variable is in a sequence.

Operator	Description	Example

in	True if value is found in list or in sequence, and false it item is not in list or in sequence	>>> x="Hello World"
		>>> print('H' in x)
		True

not in	True if value is not found in list or in sequence, and false it item is in list or in sequence.	>>> x="Hello World"
		>>> print("Hello" not in x)
		False

Identity Operators:

- Sometimes, in Python programming we need to compare the memory address of two objects.
- This is made possible with the help of the identity operator.
- Identity operators are used to check whether both operands are same or not.

- Python provides 'is' and 'is not' operators which are called identity operators and both are used to check if two values are located on the same part of the memory.
- Two variables that are equal does not imply that they are identical.

Operator	Description	Example

is	Return true, if the variables on either side of the operator point to the same object and false otherwise.	>>> a=3
		>>> b=3
		>>> print(a is b)
		True

is not	Return false, if the variables on either side of the operator point to the same object and true otherwise.	>>> a=3
		>>> b=3
		>>> print(a is not b)
		False

Q2 b Write python program to display output like.(W-23) 4 Marks

2
4 6 8
10 12 14 16 18

Ans:

```
a=2
for i in range(1,6,2):
    for j in range(i):
        print(a,end=' ')
        a+=2
    print()
```

Q3 b Explain use of Pass and Else keyword with for loops in python.(W-23) 4 Marks Ans:

Pass Statement:

- It is used when a statement is required syntactically but we do not want any command or code to execute.
- A pass statement in Python also refers to as a will statement.
- The pass statement is a null operation; nothing happens when it executes.
- The pass is also useful in places where your code will eventually go, but has not been written yet.
- **Syntax:** pass
- **Example:** For pass statement.

```
for i in range(1,11):
        if i%2==0: # check if the number is even pass # (No
                operation)
        else:
                print("Odd Numbers: ",i)
```

Output:

Odd Numbers: 1
Odd Numbers: 3

Odd Numbers: 5
Odd Numbers: 9
Odd Numbers: 7

Else Statement:

- The else block just after for/while is executed only when the loop is NOT terminated by a break statement.
- The else keyword in a for loop specifies a block of code to be executed when the loop is finished.
- **Example:**

```
for i in range(1, 4):
        print(i)
else: # Executed because no break in for
        print("Finally Exit")
```

- **Output:**

1
2
3
Finally Exit

Summer-23

Q 1 (b) Describe membership operators in python. (any four points) 2 Marks Ans:

Same as Winter 2023-2(a)

Q 2 (a) Describe Keyword "continue" with example. 4 Marks Ans:

Continue:

- The continue statement in Python returns the control to the beginning of the while loop.
- The continue statement rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop.
- **Syntax:**

```
continue
```

- **Example:** For continue statement. i=0

```
while i<10:
        i=i+1   if i==5:
                continue
        print("i= ",i)
```

- **Output:**

i=1
i=2
i=3
i=4
i=6
i=7
i=8

```
i=9
i=10
```

2(d) Write a Python program to find the factorial of a number provided by the user. (4M)

Ans:

```
num=int(input("Enter Number:")) n=num
fact=1
if num< 0:
   print("Sorry, factorial does not exist for negative numbers") elif num == 0 or
num == 1:
   print("The factorial of",n, "is 1") else:
   while num >=2: fact=fact*num
      num-=1
   print("The factorial of ",n," is ",fact)
```

Output:

```
Enter Number: 5
The factorial of 5 is 120
```

3(a) Explain Bitwise operator in Python with appropriate example. (4M) Ans:

Bitwise operators available in Python:

1) **Bitwise AND (&):**

Performs a bitwise AND operation on the corresponding bits of two numbers. Each bit of the output is 1 if the corresponding bits of both operands are 1; otherwise, it is 0.

Example:

```
a = 10 # binary: 1010
b = 6 # binary: 0110 result = a & b
print(result) # Output: 2 (binary: 0010)
```

2) **Bitwise OR (|):**

Performs a bitwise OR operation on the corresponding bits of two numbers. Each bit of the output is 0 if the corresponding bits of both operands are 0; otherwise, it is 1.

Example:

```
a = 10 # binary: 1010
b = 6 # binary: 0110 result = a | b
print(result) # Output: 14 (binary: 1110)
```

3) **Bitwise XOR (^):**

Performs a bitwise XOR (exclusive OR) operation on the corresponding bits of two numbers. Each bit of the output is 1 if the corresponding bits of the operands are different; otherwise, it is 0.

Example:

```
a = 10 # binary: 1010
b = 6 # binary: 0110 result = a ^ b
print(result) # Output: 12 (binary: 1100)
```

4) Bitwise NOT (~):

Performs a bitwise NOT operation on a single operand, which inverts all the bits. It returns the complement of the given number.

Example:

```
a = 10 # binary: 1010 result = ~a
print(result) # Output: -11 (binary: -1011)
```

5) Bitwise left shift (<<):

Shifts the bits of the left operand to the left by a specified number of positions. Zeros are shifted in from the right side.

Example:

```
a = 10 # binary: 1010 result = a << 2
print(result) # Output: 40 (binary: 101000)
```

6) Bitwise right shift (>>):

Shifts the bits of the left operand to the right by a specified number of positions. Zeros are shifted in from the left side.

Example:

```
a = 10 # binary: 1010 result = a >> 2
print(result) # Output: 2 (binary: 10)
```

5(a) Write a Python Program to check if a string is palindrome Or not. (6M) Ans:

```
def is_palindrome(string):
    # Remove whitespace and convert to lowercase string =
    string.replace(" ", "").lower()
    # Reverse the string reversed_string = string[::-1]
    # Check if the original and reversed strings are the same if string ==
    reversed_string:
        return True else:
        return False
# Test the function
input_string = input("Enter a string: ") if
is_palindrome(input_string):
    print("The string is a palindrome.") else:
    print("The string is not a palindrome.")
```

Output:

```
Enter a string:  Nitin
The string is a palindrome.
```

Winter-22

Q 1(b) List comparison operators in Python. 4 Marks Ans:

Operator	Meaning

==	Equal to
!=	Not Equal to
<	Less than
>	Greater than
<=	Less than and Equal to
>=	Greater than and Equal to

Q 2 (a) Describe bitwise operators in Python with example. 4 Marks Ans:

Same as Summer 2023-3(a)

Q 2 (d) Write python program to illustrate if else ladder. 4 Marks Ans:

```
i= 20
if (i == 10):
        print ("i is 10") elif (i == 15):
        print ("i is 15") elif (i == 20):
        print ("i is 20")
else:   print ("i is not present")

output:
i is 20
```

Note: Similar type of program can consider

Q 3 (b) Write Python code for finding greatest among four numbers. 4 Marks Ans:

```
list1 = [ ]
num = int(input("Enter number of elements in list: ")) for i in range(1,
num + 1):
        element = int(input("Enter elements: ")) list1.append(element)
print("Largest element is:", max(list1)) Output:
Enter number of elements in list: 4 Enter elements: 10
Enter elements: 20
Enter elements: 45
Enter elements: 20 Largest element is: 45
```

Summer-22

Q 1 (b) List identity operators in python. 2 Marks Ans:

Same as Winter 2023-2(a)

Q 2(a) Write a program to print following

1
1 2
1 2 3
1 2 3 4 **(4M)**

Ans:

```
for i in range(1,5):
 for j in range(1,i+1): print(j,end=' ')
 print()
```

Q 3 (b) Explain membership and assignment operators with example. 4 Marks

Ans:

Membership Operators:

Same as Winter 2023-2(a)

Assignment Operators (Augmented Assignment Operators):

- Assignment operators are used in Python programming to assign values to variables.
- The assignment operator is used to store the value on the right-hand side of the expression on the left-hand side variable in the expression.
- For example, a = 5 is a simple assignment operator that assigns the value 5 on the right to the variable a on the left.
- There are various compound operators in Python like a += 5 that adds to the variable and later assigns the same.
- It is equivalent to a = a + 5.
- **Following table shows assignment operators in Python programming:**

Sr. No.	Operator	Description	Example
1	=	Assigns values from right side operands to left side operand.	c = a + b assigns value of a + b into c
2	+=	It adds right operand to the left operand and assign the result to left operand.	c += a is equivalent to c = c + a
3	−=	It subtracts right operand from the left operand and assign the result to left operand.	c −= a is equivalent to c = c − a
4	*=	It multiplies right operand with the left operand and assign the result to left operand	c *= a is equivalent to c = c * a
5	/=	It divides left operand with the right operand and assign the result to left operand.	c /= a is equivalent to c = c / a
6	%=	It takes modulus using two operands and assign the result to left operand.	c %= a is equivalent to c = c % a
7	**=	Performs exponential (power) calculation on operators and assign value to the left operand.	c **= a is equivalent to c = c ** a
8	//=	Performs exponential (power) calculation on operators and assign value to the left operand.	c //= a is equivalent to c = c // a

Q 4 (b) Explain decision making statements If-else, if-elif-else with example. 4Marks

Ans:

If-else statement:

- if statements executes when the conditions following if is true and it does nothing when the condition is false.
- The if-else statement takes care of a true as well as false condition.
- **Syntax-1:**

```
If condition:
        Statement(s)
else:
        Statement(s)
```

- **Or Syntax-2:**

```
If condition:
        If_Block
else:   else_Block
```

- **Example:**

```
i=20
if(i<15):

        print(" less than 15")
else:
        print("greater than 15")
```

Output:

```
greater than 15
```

If-elif-else (ladder) statements:

- In If-elif-else (ladder), a user can decide among multiple options.
- The if statements are executed from the top down.
- As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the ladder is bypassed.
- If none of the conditions is true, then the final else statement will be executed.
- **Syntax:**

```
if (condition-1):
        statement elif (condition-2):
        statements
        .
        .
elif(condition-n):
        statements
else:   statements
```

- **Example:**

```
i = 20

if (i == 10):
        print ("i is 10")
```

```
        elif (i == 15):
                print ("i is 15") elif (i == 20):
                print ("i is 20")
        else:
```

- **Output:**

```
        i is 20
print ("i is not present")
```

Unit 3- Data Structures in Python

Winter-23

1(g) Explain two ways to add objects / elements to list. (2M) Ans:

1)append method:

The append() method adds an element to the end of a list. We can insert a single item in the list data time with the append().

Example: Program for append() method.

```
>>> list1=[10,20,30]
>>> list1 [10, 20, 30]
>>> list1.append(40) # add element at the end of list
>>> list1
[10, 20, 30, 40]
```

2. extend() Method:

The extend() method extends a list by appending items. We can add several items using extend() method.

Example: Program for extend() method.

```
>>>list1=[10, 20, 30, 40]
>>>list1
[10, 20, 30, 40]
>>> list1.extend([60,70]) #add elements at the end of list
>>> list1
[10, 20, 30, 40, 60, 70]
```

3. insert() Method:

We can insert one single item at a desired location by using the method insert() or insert multiple items by squeezing it into an empty slice of a list.

Example: Program for insert() method.

```
>>> list1=[10, 20]
>>>list1 [10,20]
>>> list1.insert(1,30)
>>> list1 [10, 30, 20]
```

2(c) Explain four built-in list functions.. (4M) Ans:

PWP – EPA – By Rajan Sir

Sr. No.	Function	Description	Example

1	len(list)	It returns the length of the list.	>>> list1
			[1, 2, 3, 4, 5]
			>>> len(list1)
			5

2	max(list)	It returns the item that has the maximum value in a list.	>>> list1
			[1, 2, 3, 4, 5]
			>>> max(list1)
			5

3	sum(list)	Calculates sum of all the elements of list.	>>>list1 [1, 2, 3, 4, 5] >>>sum(list1) 15

4	min(list)	It returns the item that has the minimum value in a list.	>>> list1
			[1, 2, 3, 4, 5]
			>>> min(list1)
			1

5	list(seq)	It converts a tuple into a list.	>>> list1
			[1, 2, 3, 4, 5]
			>>> list(list1)
			[1, 2, 3, 4, 5]

6	list.append(item)	It adds the item to the end of the list.	>>> list1
			[1, 2, 3, 4, 5]
			>>> list1.append(6)
			>>> list1
			[1, 2, 3, 4, 5, 6]

7	list.count(item)	It returns number of times the item occurs in the list.	>>> list1
			[1, 2, 3, 4, 5, 6, 3]
			>>> list1.count(3)
			2

8	list.extend(seq)	It adds the elements of the sequence at the end of the list.	>>> list1
			[1, 2, 3, 4, 5]
			>>> list2
			['A', 'B', 'C']
			>>> list1.extend(list2)
			>>> list1
			[1, 2, 3, 4, 5, 'A', 'B', 'C']

9	list.pop(item=list[-1])	It deletes and returns the last element of the list.	>>> list1
			[1, 2, 7, 3, 4, 5, 3]
			>>> list1.pop()
			3
			>>> list1.pop(2)
			7

10	list.remove(item)	It deletes the given item from the list.	>>> list1
			[1, 2, 3, 4, 5]
			>>> list1.remove(3)
			>>> list1
			[1, 2, 4, 5]

11	list.reverse()	It reverses the position (index number) of the items in the list.	>>> list1
			[1, 2, 3, 4, 5]
			>>> list1.reverse()
			>>> list1
			[5, 4, 3, 2, 1]

12	list.sort([func])	It sorts the elements inside the list and uses compare function if provided.	>>> list1
			[1, 3, 2, 5, 4]
			>>> list1.sort()
			>>> list1
			[1, 2, 3, 4, 5]

3(c) **(4M)**

T = ('spam, Spam', SPAM!', 'SaPm') print (T [2])
print (T[-2])
print (T[2:]) print (List (T)) Ans:

Python statement	Output
print (T [2])	SPAM!
print (T[-2])	SPAM!
print (T[2:])	['SPAM!', 'SaPm']
print (list (T))	['spam', 'Spam', 'SPAM!', 'SaPm']

4(a) Explain different functions or ways to remove key : value pair from Dictionary. (4M) Ans:

pop():

- We can remove a particular item in a dictionary by using the method pop(). This method removes as item with the provided key and returns the value.
- **Example:**

```
>>> squares
{1: 1, 2: 4, 3: 9, 4: 16}
>>> squares.pop(2) # remove a particular item 4
>>> squares
{1: 1, 3: 9, 4: 16}
```

Popitem():

- The method, popitem() can be used to remove and return an arbitrary item (key, value) form the dictionary.
- **Example:**

```
>>> squares={1:1,2:4,3:9,4:16,5:25}
>>> squares
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
>>> print(squares.popitem()) # remove an arbitrary item (5, 25)
>>> squares
{1: 1, 2: 4, 3: 9, 4: 16}
```

Clear():

- All the items can be removed at once using the clear() method.
- **Example:**

```
>>> squares
{1: 1, 4: 16}
>>> squares.clear() # removes all items
>>> squares
{}
```

Del():

- We can also use the del keyword to remove individual items or the entire dictionary itself.
- **Example:**

```
>>> squares
{1: 1, 3: 9, 4: 16}
>>> del squares[3] # delete a particular item
>>> squares
{1: 1, 4: 16}
```

5(a) Explain any four set operations with example. (6M) Ans:

Set Operations:

1. **Set Union:**
   - The union of two sets is the set of all the elements of both the sets without duplicates.
   - We can use the '|' operator to find the union of a Python set.

```
>>> first_set = {1, 2, 3}
>>> second_set = {3, 4, 5}
>>> first_set.union(second_set)
{1, 2, 3, 4, 5}
>>> first_set | second_set # using the '|' operator
{1, 2, 3, 4, 5}
```

2. **Set Intersection:**
   - The intersection of two sets is the set of all the common elements of both the sets.
   - We can use the '&' operator to find the intersection of a Python set.

```
>>> first_set = {1, 2, 3, 4, 5, 6}
>>> second_set = {4, 5, 6, 7, 8, 9}
>>> first_set.intersection(second_set)
{4, 5, 6}
>>> first_set & second_set # using the '&' operator
{4, 5, 6}
```

3. **Set Difference:**
   - The difference between two sets is the set of all the elements in first set that are not present in the second set.
   - We can use the '–' operator to achieve this in Python.

```
>>> first_set = {1, 2, 3, 4, 5, 6}
>>> second_set = {4, 5, 6, 7, 8, 9}
>>> first_set.difference(second_set)
{1, 2, 3}
>>> first_set - second_set # using the '-' operator
{1, 2, 3}
```

```
>>> second_set - first_set
{8, 9, 7}
```

4. **Set Symmetric Difference:**
   - The symmetric difference between two sets is the set of all the elements that are either in the first set or the second set but not in both.
   - We have the choice of using either the symmetric_difference() method or the ^ operator to do this in Python.

```
>>> first_set = {1, 2, 3, 4, 5, 6}
>>> second_set = {4, 5, 6, 7, 8, 9}
>>> first_set.symmetric_difference(second_set)
{1, 2, 3, 7, 8, 9}
>>> first_set ^ second_set # using the `^` operator
{1, 2, 3, 7, 8, 9}
```

6(c) List and explain any four built-in functions on set. (6M) Ans:

Built-in Functions with Set :

1) add()
2) discard()
3) copy()
4) remove()
5) clear()
6) union()
7) difference()
8) intersection()
9) discard()
10) issubset()
11) issuperset()
12) pop()
13) update()
14) symmetric_difference()

add(): Adds an element to the set. If an element is already exist in the set, then it does not add that element.

Example:

```
s = {'g', 'e', 'k', 's'}
# adding f into set s
s.add('f')
print('Set after updating:', s)
```

output:

```
Set after updating: {'s', 'f', 'e', 'g', 'k'}
```

discard():

Removes the element from the set

Example:

```
s = {'g', 'e', 'k', 's'}
print('Set before discard:', s) s.discard('g')
print('\nSet after discard g:', s)
```

Output:

Set before discard: {'s', 'e', 'k', 'g'}
Set after discard g: {'s', 'e', 'k'}

remove():

Removes the specified element from the set. If the specified element not found, raise an error.

Example:

```
s = {'g', 'e', 'k', 's'}
print('Set before remove:', s)
s.remove('e')
print('\nSet after remove e:', s)
```

Output:

Set before remove: {'s', 'k', 'e', 'g'}
Set after remove e: {'s', 'k', 'g'}

clear():

Removes all elements from the set

Example:

```
s = {'g', 'e', 'k', 's'}
print('Set before clear:', s)
s.clear()
print('\nSet after clear:', s)
```

Output:

Set before clear: {'g', 'k', 's', 'e'}
Set after clear: set()

copy():

Returns a shallow copy of the set

Example:

```
s = {'g', 'e', 'k', 's'}
p=s.copy()
print("original set:",s)
print("Copied set:",p)
```

Output:

original set: {'k', 's', 'g', 'e'}
Copied set: {'k', 's', 'g', 'e'}

Union():

The set.union() method returns a new set with distinct elements from all the given sets.

Example:

```
nums1 = {1, 2, 2, 3, 4, 5}
nums2 = {4, 5, 6, 7, 7, 8}
distinct_nums = nums1.union(nums2) print("The union of two
sets is: ", distinct_nums)
```

Output:

The union of two sets is: {1, 2, 3, 4, 5, 6, 7, 8}

Difference(): The set.difference() method returns the new set with the unique elements that are not in the other set passed as a parameter.

Example:

```
nums1 = {1, 2, 2, 3, 4, 5}
```

```
nums2 = {4, 5, 6, 7, 8, 8}
nums3 = nums1.difference(nums2) nums4 =
nums2.difference(nums1) print("nums1 - nums2: ", nums3)
print("nums2 - nums1: ", nums4)
```

Output:

```
nums1 - nums2: {1, 2, 3}
nums2 - nums1: {8, 6, 7}
```

Intersection():

The set.intersection() method returns a new set with the elements that are common in the given sets.

Example:

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}
z = x.intersection(y) print(z)
```

Output:

```
apple
```

Summer-23

1(c) Write down the output of the following Python code. (2M)

>>> indices=['zero','one','two',' three','four', 'five']

i) >>>indices[:4]

ii) >>>indices[:-2] Ans:

Output as follows:

i) >>>indices[:4]

```
[zero, one, two, three]
```

ii) >>>indices[:-2]

```
[zero, one, two, three]
```

2(b) Explain creating Dictionary and accessing Dictionary Elements with example. (6M) Ans:

Creating Dictionary

The simplest method to create dictionary is to simply assign the pair of keys: values to the dictionary using operator (=).

- **There are two ways for creation of dictionary in python.**

1. We can create a dictionary by placing a comma-separated list of key: value pairs in curly braces {}. Each key is separated from its associated value by a colon:

   Example: For creating a dictionary using { }.

```
>>> dict1={} #Empty dictionary
>>> dict1
{}
>>>  dict2={1:"Orange",  2:"Mango",  3:"Banana"}  #Dictionary  with integer keys
>>> dict2
```

```
{1: 'Orange', 2: 'Mango', 3: 'Banana'}
>>> dict3={"name":"vijay", 1:[10,20]} #Dictionary with mixed keys
>>> dict3
{'name': 'vijay', 1: [10, 20]}
```

2. Python provides a build-in function dict() for creating a dictionary

Example: Creating directory using dict().

```
>>> d1=dict({1:"Orange",2:"Mango",3:"Banana"})
>>> d2=dict([(1,"Red"),(2,"Yellow"),(3,"Green")])
>>> d3=dict(one=1, two=2, three=3)
>>> d1
{1: 'Orange', 2: 'Mango', 3: 'Banana'}
>>> d2
{1: 'Red', 2: 'Yellow', 3: 'Green'}
>>> d3
{'one': 1, 'two': 2, 'three': 3}
```

Accessing Values in a Dictionary

• We can access the items of a dictionary by following ways:

1. Referring to its key name, inside square brackets([]).

Example: For accessing dictionary items [ ] using.

```
>>> dict1={'name':'vijay','age':40}
>>> dict1['name'] 'vijay'
>>> dict1['adr']
Traceback (most recent call last):
File "<pyshell#79>", line 1, in <module> dict1['adr']
KeyError: 'adr'
>>> Here, if we refer to a key that is not in the dictionary, you'll get an
exception. This error can be avoided by using get() method.
```

2. Using get() method returns the value for key if key is in the dictionary, else None, so that this method never raises a KeyError.

Example: For accessing dictionary elements by get().

```
>>> dict1={'name':'vijay','age':40}
>>> dict1.get('name') 'vijay'
```

2(c) Explain any four Python's Built-in Function with example. (6M) Ans:

Python's Built-in Function:

1. **len(list)**

   It returns the length of the list.

Example:

```
>>> list1  [1, 2, 3, 4, 5]
>>> len(list1) 5
```

2. **max(list)**

It returns the item that has the maximum value in a list

Example:

>>> list1 [1, 2, 3, 4, 5]
>>> max(list1) 5

3. **sum(list)**

Calculates sum of all the elements of list.

Example:

>>>list1
[1, 2, 3, 4, 5]
>>>sum(list1) 15

4. **min(list)**

It returns the item that has the minimum value in a list.

Example:

>>> list1 [1, 2, 3, 4, 5]
>>> min(list1) 1

5. **list(seq)**

It converts a tuple into a list.

Example:

>>> list1 [1, 2, 3, 4, 5]
>>> list(list1) [1, 2, 3, 4, 5]

6. **abs(n)**

It returns the absolute value of a number.

Example:

>>> abs(10) 10

7. **all()**

The all() function returns True if all items in an iterable are true, otherwise it returns False.

Example:

>>> x=[True, True, True]
>>> all(x) True

8. **any()**

The any() function returns True if any item in an iterable are true, otherwise it returns False. If the iterable object is empty, the any() function will return False.

Example:

>>> x=[True, False, True]
>>> any(x) True

9. **bin()**

The bin() function returns the binary version of a specified integer. The result will always start with the prefix 0b

Example:

'0b1010' with the prefix 0b.

10. **bool()**

The bool() function returns the boolean value of a specified object.

Example:

>>> bool(1) True

11. **exp()**

The method exp() returns returns exponential of x: ex. x: This is a numeric expression.

Example:

>>> math.exp(1) 2.718281828459045

3(a) Write a python program to input any two tuples and interchange the tuple variables. (4M)

Ans:

```
def interchange_tuples(tup1, tup2): new_tup1 = tup2
        new_tup2 = tup1
        return new_tup1, new_tup2
# Input two tuples
tuple1 = tuple(input("Enter the elements of the first tuple (separated by commas): ").split(","))
tuple2 = tuple(input("Enter the elements of the second tuple (separated by commas): ").split(","))
# Interchange the tuples
result_tuple1, result_tuple2 = interchange_tuples(tuple1, tuple2)

# Display the result print("Interchanged tuples:")
print("Tuple 1:", result_tuple1)
print("Tuple 2:", result_tuple2)
```

output:

Enter the elements of the first tuple (separated by commas): 10,20 Enter the elements of the second tuple (separated by commas): 30,40 Interchanged tuples:
Tuple 1: ('30', '40')
Tuple 2: ('10', '20

4(a) Differentiate between list and Tuple. (4M) Ans:

Sr. No	List	Tuple
1	Lists are mutable.	Tuples are immutable.
2	Iteration in lists is time consuming.	Iteration in tuples is faster
3	Lists are better for insertion and deletion operations	Tuples are appropriate for accessing the elements
4	Lists consume more memory	Tuples consume lesser memory
5	Lists have several built-in methods	Tuples have comparatively lesser built-in methods.
6	Lists are more prone to unexpected errors	Tuples operations are safe and chances of error are very less
8	Lists are initialized using square brackets [].	Tuples are initialized using parentheses ().

5 (c) Write a Python Program to accept values from user in a list and find the largest number and smallest number in a list. (6M)

Ans:

```
list = []
num = int(input('Enter size of list you want: ')) for n in
range(num):
    numbers = int(input('Enter list element ')) list.append(numbers)
print("Gratest element in the list is : ", max(list)) print("Smallest element in the
list is : ", min(list))
```

output:

```
Enter size of list you want:  5 Enter list element  99
Enter list element  8 Enter list element
199 Enter list element  2 Enter list
element  56
Gratest element in the list is :  199 Smallest element
in the list is :  2
```

6 (a) Explain any six set function with example. (6M)

1) union():

Return a new set containing the union of two or more sets

Example:

```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
union_set = set1.union(set2) print(union_set)
```

Output:

{1, 2, 3, 4, 5}

2) Intersection:

Intersection operation performed on two sets returns all the elements which are common or in both the sets.

Example:

```
set1 = {1, 2, 3}
set2 = {2, 3, 4}
intersection_set = set1.intersection(set2) print(intersection_set)
```

Output:

{2, 3}

3) Difference:

Difference operation on two sets set1 and set2 returns all the elements which are present on set1 but not in set2.

Example:

```
set1 = {1, 2, 3, 4, 5}
set2 = {3, 4}
difference_set = set1.difference(set2) print(difference_set) #
Output: {1, 2, 5}
```

4) add(element):

This function adds an element to a set.

Example:

```
fruits = {"apple", "banana", "cherry"} fruits.add("orange")
print(fruits) # Output: {'apple', 'banana', 'cherry', 'orange'}
```

5) remove(element):

This function removes an element from a set.

Example:

```
numbers = {1, 2, 3, 4, 5} numbers.remove(3)
print(numbers)
# Output: {1, 2, 4, 5}
```

6) clear():

This function removes all elements from a set, making it an empty set.

Example:

```
numbers = {1, 2, 3, 4, 5} numbers.clear()
print(numbers)
# Output: set()
```

7) isdisjoint():

The isdisjoint() method in Python's set class is used to check whether two sets have any common elements. It returns True if the sets are disjoint (i.e., they have no common elements), and False otherwise.

Example:

```
# Example 1
set1 = {1, 2, 3, 4}
set2 = {5, 6, 7}
set3 = {3, 4, 5}
print(set1.isdisjoint(set2)) # True, no common elements print(set1.isdisjoint(set3)) #
False, both sets have elements 3 and 4
# Example 2
fruits = {"apple", "banana", "orange"} colors = {"red",
"green", "blue"}

print(fruits.isdisjoint(colors)) # True, no common elements # Example 3
setA = {1, 2, 3}
setB = {4, 5, 6}
print(setA.isdisjoint(setB)) # True, no common elements
```

8) pop():

Method in Python's set class is used to remove and return an arbitrary element from the set. Since sets are unordered collections, there is no guarantee on which element will be popped.

Example:

```
fruits = {"apple", "banana", "orange", "grape"}
# Remove and return an arbitrary element from the set popped_element = fruits.pop()
print(popped_element)
```

Output:

```
#an arbitrary element from the set apple
print(fruits)
#the modified set after popping an element
{'grape', 'banana', 'orange'}
```

9) update(): The update() method in Python's set class is used to update a set by adding elements from another iterable or set. It modifies the set in place by adding all the elements from the iterable or set specified.

Example:

```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
set1.update(set2) print(set1)
```

Output:

{1, 2, 3, 4, 5}

Winter-22

1(c) Describe Tuples in Python. (2M) Ans:

- A tuple is a collection of items which is ordered and unchangeable.
- Tuples are the sequence or series values of different types separated by commas (,).
- Example: tup1=(10,20,30)

Q 2 (b) Write any four methods of dictionary. 4 Marks Ans:

Sr. No.	Function	Description	Example

1	clear()	Removes all the elements from the dictionary	>>> car = {
			"brand": "Ford",
			"model": "Mustang",
			"year": 1964 }
			>>> car.clear()
			>>> print(car)
			{}

2	copy()	Returns a copy of the dictionary	>>> car = { "brand": "Ford",
			"model": "Mustang", "year": 1964 }
			>>> car.copy()
			>>> print(car)
			{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}

3	fromkeys()	Returns a dictionary with the specified keys and value	>>> x = ('key1', 'key2', 'key3')
			>>> y = 0
			>>> thisdict = dict.fromkeys(x, y)
			>>> print(thisdict)
			{'key1': 0, 'key2': 0, 'key3': 0}

4	get()	Returns the value of the specified key	>>> car = { "brand": "Ford", "model": "Mustang", "year": 1964 } >>>x=car.get("model") >>>print(x) Mustang
5	items()	Returns a list containing a tuple for each key value pair	>>> car = { "brand": "Ford", "model": "Mustang", "year": 1964 } >>>x = car.items() >>>print(x) dict_items([('brand', 'Ford'),('model', 'Mustang'), ('year', 1964)])
6	keys()	Returns a list containing the dictionary's keys	>>> car = { "brand": "Ford", "model": "Mustang", "year": 1964 } >>>x = car.keys() >>>print(x) dict_keys(['brand', 'model', 'year'])
7	pop()	Removes the element with the specified key	>>> car = { "brand": "Ford", "model": "Mustang", "year": 1964 } >>>car.pop("model") >>>print(car) {'brand': 'Ford', 'year': 1964}
8	popitem()	Removes the last inserted key- value pair	>>> car = { "brand": "Ford", "model": "Mustang", "year": 1964 } >>>car.popitem("model") >>>print(car)

			{'brand': 'Ford', 'model': 'Mustang'}
9	setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value	>>> car = { "brand": "Ford", "model": "Mustang", "year": 1964 } >>>x=car.setdefault("model", "Bronco") >>>print(x) Mustang
10	update()	Updates the dictionary with the specified key-value pairs	>>> car = { "brand": "Ford", "model": "Mustang", "year": 1964 } >>>car.update({"color":"White"}) >>>print(car) {'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'White'}
11	values()	Returns a list of all the values in the dictionary	>>> car = { "brand": "Ford", "model": "Mustang", "year": 1964 } >>>x = car.values() >>>print(x) dict_values(['Ford', 'Mustang', 1964])
12	all()	Returns true if all keys of the dictionary are true(or if the dictionary is empty)	>>> car = { "brand": "Ford", "model": "Mustang", "year": 1964 } >>>all(car) True
13	any()	Returns true if any keys if dictionary is not empty and false if dictionary is empty	>>> car = { } >>>any(car) False
14	Len()	Returns total number of element in the dictionary	>>> car = { "brand": "Ford", "model": "Mustang", "year": 1964 } >>>len(car) 3

3(a) Write basis operations of list. (4M) Ans:

1) Accessing values in list:

- Accessing elements liters from a list in Python is a method to get values that are stared in the list at a particular location or index.
- To access values in lists, use the square brackets for slicing along with the index or indices to obtain

value available at that index.

- Example: accessing list values.

```
>>> list1 = ["one","two",3,10,"six",20]
>>> list1[0] 'one'
>>> list1[-2] 'six'
>>> list1[1:3] ['two', 3]
>>> list1[3:] [10, 'six', 20]
>>> list1[:4] ['one', 'two', 3, 10]
>>>
```

2) Deleting Values in List:

- The pop() method in Python is used to remove a particular item/element from the given index in the list.
- The pop() method removes and returns the last item if index is not provided. This helps us implement lists as stacks (first in, last out data structure).

```
>>> list= [10, 20, 30, 40]
>>> list
[10, 20, 30, 40]
30
>>> list [10, 20, 40]
>>> list.pop() 40
>>> list [10, 30]
```

- We can delete one or more items from a list using the keyword del. It can even delete the list entirely. But it does not store the value for further use

```
>>> list= [10, 20, 30, 40]
>>> list
[10, 20, 30, 40]
>>> del (list[1]) # del() with index
>>> list [10, 30, 40]
>>> del list[2] # del with index
>>> list [10, 30]
```

- The remove() method in Python issued to remove a particular element from the list. We use the remove() method if we know the item that we want to remove or delete from the list (but not the index).

```
>>> list=[10,"one",20,"two"]
>>> list.remove(20)
>>> list
[10, 'one', 'two']
>>> list.remove("one")
>>> list [10, 'two']
>>>
```

3. Updating Lists:

- List are mutable, meaning their elements can be changed or updated unlike string or tuple.
- Mutability is the ability for certain types of data to be changed without entirely recreating it.
- Using mutable data types can allow programs to operate quickly and efficiently.
- Multiple values can be added into list. We can use assignment operator (=) to change an item or a range of items.
- We can update items of the list by simply assigning the value at the particular index position.
- We can also remove the items from the list using remove() or pop() or del statement.

```
>>> list1= [10, 20, 30, 40, 50]
>>> list1
```

```
[10, 20, 30, 40, 50]
>>> list1[0]=0 # change 0th index element
>>> list1
[0, 20, 30, 40, 50]
>>> list1[-1]=60 # change last index element
>>> list1
[0, 20, 30, 40, 60]
>>> list1[1]=[5,10] # change 1st index element as sublist
>>> list1
[0, [5, 10], 30, 40, 60]
>>> list1[1:1]=[3,4] # add elements to a list at the desired location
>>> list1
[0, 3, 4, [5, 10], 30, 40, 60]
```

Indexing:

- There are various ways in which we can access the elements of a list.
- List Index: We can use the index operator [] to access an item in a list. Index starts from 0. So, a list having 5 elements will have index from 0 to 4.
- **Example:**

```
>>> list1=[10,20,30,40,50]
>>> list1[0] 10
>>> list1[4] 50
>>> list1[1:3] [20, 30]
```

5. List Slicing:

- The slicing operator returns a subset of a list called slice by specifying two indices, i.e. start and end.
- **Syntax:**

  List_variable[start_index:end_index]
- **Example:**

```
>>> l1=([10,20,30,40,50])
>>> l1[1:4] [20, 30, 40]
```

4(a) Compare list and dictionary. (Any 4 points) (4M)

List	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 indices of the list are integers starting from 0.	The keys of the dictionary can be of any data type.
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.

4(b) Explain any four file modes in Python. (4M)

Same as Summer 2023-3C)

(a) How to create user defined package in java. Explain with suitable example. (6M)

Same as Summer 2023-3C)

5(c) Write python program to perform following operations on Set (Instead of Tuple)

i) Create set
ii) Access set Element Update set
iii) Delete set **(6M)**

Ans:

```
# To Create set
S={10,20,30,40,50}

# To Access Elements from set
print (S)
#To add element into set using add method
S.add(60)
print(S)
#To update set using update method
```

iv)

```
S.update(['
A','B'])
print(S)

#To Delete element from Set using discard() method
S.discard(30)
print(S)

#To delete element from set using remove() method
S.remove('A')
print(S)

#To delete element from set using pop()
method S.pop()
print(S)
```

Output:

```
{50, 20, 40, 10, 30}
{50, 20, 40, 10, 60, 30}
{'B', 50, 20, 'A', 40, 10, 60, 30}
{'B', 50, 20, 'A', 40, 10, 60}
{'B', 50, 20, 40, 10, 60}
 20, 40, 10, 60}

back (most recent call last):
C:\Users\Vijay Patil\AppData\Loc

back (most recent call last):
C:\Users\Vijay
AppData\Local\Programs\Python\Python310\temp.py", line 9, in
```

in Python are divided in two categories:

Immutable data types – Values cannot be changed. Immutable data types in Python are

1. Numbers
2. String
3. Tuple

Mutable data types – Values can be changed.

Mutable data types in Python are:

1. List
2. Dictionaries
3. Sets

1. Numbers:

- Python supports integers, floats and complex numbers.
- An integer is a number without decimal point for example 5, 6, 10 etc.
- A float is a number with decimal point for example 6.7, 6.0, 10.99 etc.
- A complex number has a real and imaginary part for example 7+8j, 8+11j etc.
- **Example:**

```
# int
num1 = 10
num
2 =
100
#
float
a = 10.5
b = 8.9
# complex
numbers x =
3 + 4j
y = 9 + 8j
```

2. String:

- A string is usually a bit of text (sequence of characters). In Python we use " (double quotes) or ' (single quotes) to represent a string.

There are several ways to create strings in Python:

1) We can use ' (single quotes), see the string str in the following code.
2) We can use " (double quotes), see the string str2 in the source code below.
3) Triple double quotes """ and triple single quotes "' are used for creating multi-line strings in Python.

Example:

```
str =
'beginnersboo
k' str2 =
"Chaitanya"
# multi-line
string str3 =
"""Welcome
to
Pythonsbook"
"""
```

s similar to List except that the objects in tuple are immutable which means we cannot change the elements of a tuple once assigned.

- On the other hand, we can change the elements of a list.
- To create a tuple in Python, place all the elements in a () parenthesis, separated by commas.
- A tuple can have heterogeneous data items, a tuple can have string and list as data items as well.

Example

```
# tuple of strings
my_data = ("hi", "hello", "bye") # tuple of int, float, string
my_data2 = (1, 2.8, "Hello World") # tuple of string and list  my_data3 = ("Book", [1, 2, 3])
# tuples inside another tuple # nested tuple
my_data4 = ((2, 3, 4), (1, 2, "hi"))
```

Tuple:

- In Python, a tuple i

4. List

- A list is a data type that allows you to store various types data in it. List is a compound
- data type which means you can have different-2 data types under a list, for example we can have integer, float and string items in a same list.
- To create a list all you have to do is to place the items inside a square bracket [] separated by comma ,.

Example:

```
# list of floats
num_list = [11.22, 9.9, 78.34, 12.0]
# list of int, float and strings
mix_list = [1.13, 2, 5, "beginnersbook", 100, "hi"] # an empty list
nodata_list = []
```

5. Dictionaries

- Dictionary is a mutable data type in Python. A python dictionary is a collection of key and value pairs separated by a colon (:), enclosed in curly braces {}.
- Left side of the colon(:) is the key and right side of the : is the value.

Example:

```
mydict = {'StuName': 'Ajeet', 'StuAge': 30, 'StuCity': 'Agra'}
```

6. Sets:

- Set is an unordered and unindexed collection of items in Python.
- Unordered means when we display the elements of a set, it will come out in a random order.
- Unindexed means, we cannot access the elements of a set using the indexes like we can do in list and tuples.
- The elements of a set are defined inside curly brackets and are separated by commas.

Example:

```
myset = {1, 2, 3, 4, "hello"}
```

Summer-22

Q 1 (c) Give two differences between list and tuple. 2 Marks Ans:

Q 2 (b) Explain four Buit-in tuple functions in python with example

.

4 Marks Ans:

Sr. No.	Function	Description	Example

1	cmp(tuple1, tuple2)	Compares elements of both tuples.	>>> tup1=(1,2,3)
			>>> tup2=(1,2,3)
			>>> cmp(tup1,tup2)
			0

2	len(tuple)	Gives the total length of the tuple.	>>> tup1
			(1, 2, 3)
			>>> len(tup1)
			3

3	max(tuple)	Returns item from the tuple with max value.	>>> tup1
			(1, 2, 3)
			>>> max(tup1)
			3

4	min(tuple)	Returns item from the tuple with min value.	>>> tup1
			(1, 2, 3)
			>>> min(tup1)
			1

5	count()	Returns the number of times a specified value occurs in a tuple	>>> tup1
			(1, 2, 3, 2, 4)
			>>> tup1.count(2)
			2

6	zip(tuple1,tuple2)	It zips elements from two tuples into a list of tuples.	>>> tup1=(1,2,3)
			>>> tup2=('A','B','C')
			>>> tup3=zip(tup1,tup2)
			>>> list(tup3)
			[(1, 'A'), (2, 'B'), (3, 'C')]

7	index()	Searches the tuple for a specified value and returns the position of where it was found	>>> tup1
			(1, 2, 3)
			>>> tup1.index(3)
			2

8	tuple(seq)	Converts a list into tuple.	>>>tuple1 = (1, 2, 3, 4, 5) >>>list1 = list (tuple1) >>>list1 [1,2,3,4,5]

3(c) Explain indexing and slicing in list with example. (4M) Ans:

Indexing:

- An individual item in the list can be referenced by using an index, which is an integer number that indicates the relative position of the item in the list.
- There are various ways in which we can access the elements of a list some as them are given below:

1. **List Index:**
   We can use the index operator [] to access an item in a list. Index starts from 0. So, a list having 5 elements will have index from 0 to 4.

   Example: For list index in list.

```
>>> list1=[10,20,30,40,50]
>>> list1[0] 10
>>> list1[3:] # list[m:] will return elements indexed from mth index to last index
[40, 50]
>>>list1[:4] # list[:n] will return elements indexed from first index to n-1th
index
[10, 20, 30, 40]
>>> list1[1:3] # list[m:n] will return elements indexed from m to n-1.
[20, 30]
>>> list1[5]
Traceback (most recent call last):
File "<pyshell#71>", line 1, in <module> list1[5]
IndexError: list index out of range
```

2. **Negative Indexing:**
   - Python allows negative indexing for its sequences.
   - The index of −1 refers to the last item, −2 to the second last item and so on.

   Example: For negative indexing in list.

```
>>> list2=['p','y','t','h','o','n']
>>> list2[-1] 'n'
>>> list2[-6] 'p'
>>> list2[-3:] ['h', 'o', 'n']
>>> list2[-7]
Traceback (most recent call last):
File "<pyshell#76>", line 1, in <module> list2[-7]
IndexError: list index out of range
```

3. **List Slicing:**
   - Slicing is an operation that allows us to extract elements from units.
   - The slicing feature used by Python to obtain a specific subset or element of the data structure using the colon (:) operator.
   - The slicing operator returns a subset of a list called slice by specifying two indices, i.e. start and end.
   - **Syntax:**

     list_variable[start_index:end_index]
   - This will return the subset of the list starting from start_index to one index less than that of the ending

- **Example:** For slicing list.

```
>>> l1=([10,20,30,40,50])
>>> l1[1:4] [20, 30, 40]
>>>l1[2:5] [30,40,50]
```

4(a) Write a program to create dictionary of student the includes their ROLL NO and NAME
i) Add three students in above dictionary
ii) Update name='Shreyas' of ROLL NO=2
iii) Delete information of ROLL NO=1 **(4M)**
Ans:

i)

```
>>> dict1={1:"Vijay",2:"Santosh",3:"Yogita"}
>>>print(dict1)
{1: 'Vijay', 2: 'Santosh', 3: 'Yogita'}
```

ii)

```
>>>dict1[2]="Shreyas"
>>>print(dict1)
{1: 'Vijay', 2: 'Shreyas', 3: 'Yogita'}
```

iii)

```
>>>dict1.pop(1)
'Vijay'
>>>print(dict1)
{2: 'Shreyas', 3: 'Yogita'}
```

5 (A) Write the output of the following. (6M)
Ans:

i)
```
>>> a=[2,5,1,3,6,9,7]
>>> a[2:6]=[2,4,9,0]
>>> print(a)
```
Output: [2, 5, 2, 4, 9, 0, 7]

ii)
```
>>> b=["Hello","Good"]
>>> b.append("python")
>>>print(b)
```
Output: ['Hello', 'Good', 'python']

iii)

	Output:
>>>t1=[3,5,6,7]	
>>>print(t1[2])	>>>6
>>>print(t1[-1])	>>>7
>>>print(t1[2:])	>>>[6, 7]
>>>print(t1[:])	>>>[3, 5, 6, 7]

Unit 4- Python Functions, modules and packages

Winter-23

1(e) State use of namespace in python. (2M) Ans:

- Namespaces bring you three advantages: they group names into logical containers, they prevent clashes between duplicate names, and third, they provide context to names.
- Namespaces prevent conflicts between classes, methods and objects with the same name that might have been written by different people.
- A namespace is a system to have a unique name for each and every object in Python. An object might be a variable or a method. Python itself maintains a namespace in the form of a Python dictionary.
- A namespace in python is a collection of names. So, a namespace is essentially a mapping of names to corresponding objects.

2(d) Write python program using module, show how to write and use module by importing it. (4M)

Ans:

For creating a module write following code and save it as p1.py #p1.py

```
def add(a, b):
   #This function adds two numbers and return the result result = a + b
   return result
def sub(a, b):
   #This function subtracts two numbers and return the result result = a - b
   return result
```

Import the definitions inside a module:

```
import p1 print(p1.add(10,20))
print(p1.sub(20,10))
```

Output:

```
30
10
```

4(b) Explain Numpy package in detail. (4M) Ans:

- NumPy is the fundamental package for scientific computing with Python. NumPy stands for "Numerical Python". It provides a high-performance multidimensional array object, and tools for working with these arrays.
- An array is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers and represented by a single variable. NumPy's array class is called ndarray. It is also known by the alias array.
- In NumPy arrays, the individual data items are called elements. All elements of an array should be of the same type. Arrays can be made up of any number of dimensions.
- In NumPy, dimensions are called axes. Each dimension of an array has a length which is the total number of elements in that direction.
- The size of an array is the total number of elements contained in an array in all the dimension. The size of NumPy arrays are fixed; once created it cannot be changed again.
- Numpy arrays are great alternatives to Python Lists. Some of the key advantages of Numpy arrays are that they are fast, easy to work with, and give users the opportunity to perform calculations across entire arrays.

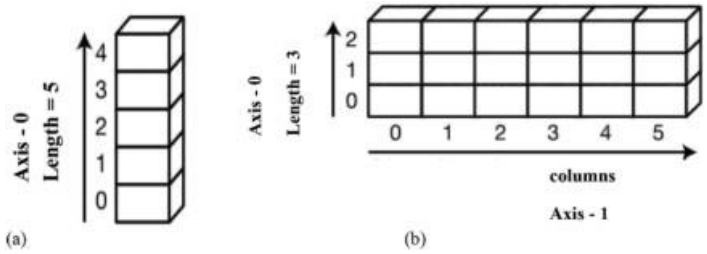


Figure shows the axes (or dimensions) and lengths of two example arrays; (a) is a one-dimensional array and (b) is a two-dimensional array.

- A one-dimensional array has one axis indicated by Axis-0. That axis has five elements in it, so we say it has length of five.
- A two-dimensional array is made up of rows and columns. All rows are indicated by Axis-0 and all columns are indicated by Axis-1. If Axis-0 in two-dimensional array has three elements, so its length it three and Axis-1 has six elements, so its length is six.

Commands to install numpy in window, Linux and MAC OS:

1) python -m pip install numpy
2) To use NumPy you need to import Numpy:
3) import numpy as np # alias np

Using NumPy, a developer can perform the following operations:

1) Mathematical and logical operations on arrays.
2) Fourier transforms and routines for shape manipulation.
3) Operations related to linear algebra.
4) NumPy has in-built functions for linear algebra and random number generation.

5 (c) Write a program illustrating use of user defined package in python. (6M)

Ans:

 # student.py

```
class Student:
        def __init__(self, student):
                self.name = student['name'] self.gender =
                student['gender'] self.year = student['year']
        def get_student_details(self):
                return f"Name: {self.name}\nGender: {self.gender}\nYear: {self.year}"
```

 # faculty.py

```
class Faculty:
        def __init__(self, faculty):
                self.name = faculty['name'] self.subject =
                faculty['subject'] def get_faculty_details(self):
                return f"Name: {self.name}\nSubject: {self.subject}"
```

 # testing.py

```
        # importing the Student and Faculty classes from respective files from student import
        Student
        from faculty import Faculty
        # creating dicts for student and faculty
        student_dict = {'name' : 'ABC', 'gender': 'Male', 'year': '3'} faculty_dict = {'name':
        'XYZ', 'subject': 'Programming'}
```

```
# creating instances of the Student and Faculty classes student =
Student(student_dict)
faculty = Faculty(faculty_dict)
# getting and printing the student and faculty details print(student.get_student_details())
print() print(faculty.get_faculty_details())
```

Output :

```
Name: ABC Gender: Male Year: 3
Name: XYZ
Subject: Programming
```

Summer-23

4(d) Explain Module and its use in Python. (4M) Ans:

Modules:

- Modules are primarily the (.py) files which contain Python programming code defining functions, class, variables, etc. with a suffix .py appended in its file name.
- A file containing .py python code is called a module.
- If we want to write a longer program, we can use file where we can do editing, correction. This is known as creating a script. As the program gets longer, we may want to split it into several files for easier maintenance.
- We may also want to use a function that you've written in several programs without copying its definition into each program.
- In Python we can put definitions in a file and use them in a script or in an interactive instance of the interpreter. Such a file is called a module.

Use of module in python : Code organization:

Modules allow you to organize related code into separate files, making it easier to navigate and maintain large projects.

Code reusability:

Modules can be imported and reused in multiple programs, enabling code reuse and reducing duplication.

Encapsulation:

Modules provide a way to encapsulate code and hide the implementation details, allowing users to focus on the functionality provided by the module.

Name spacing:

Modules help avoid naming conflicts by providing a separate namespace for the names defined within the module. 5(b) Write a Python program to calculate sum of digit of given number using function. (6M)

Ans:

```
def calculate_digit_sum(number):
    # Convert the number to a string num_str = str(number)
    # Initialize a variable to store the sum digit_sum = 0
    # Iterate over each character in the string for digit in num_str:
        # Convert the character back to an integer and add it to the sum digit_sum += int(digit)
        # Return the sum of the digits return digit_sum
# Test the function
input_number = int(input("Enter a number: ")) sum_of_digits =
calculate_digit_sum(input_number) print("Sum of digits:", sum_of_digits)
```

output:

```
Enter a number: 123 Sum of digits: 6
```

Winter-22

1(d) Write use of lambda function in python. (2M)

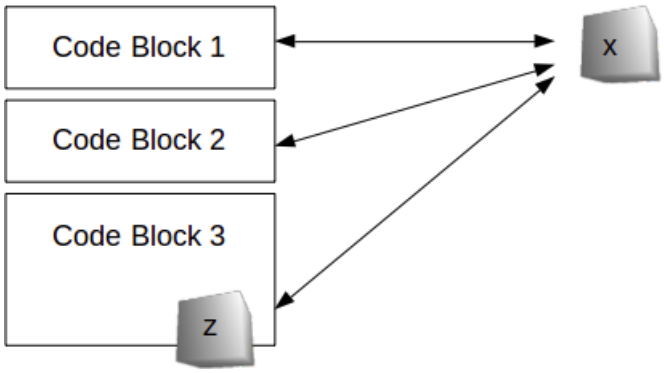
ns:

- The lambda function, which is also called anonymous function.
- A lambda function can take any number of arguments, but can only have one expression.
- **Syntax:**
  lambda arguments : expression
- **Example:**
  x= lambda a,b : a*b Print(x(10,5)
- **Output:**
  50

2(c) What is local and global variables? Explain with appropriate example. (2M) Ans:

- **Global variables:** global variables can be accessed throughout the program body by all functions.
- **Local variables:** local variables can be accessed only inside the function in which they are declared

Concept Diagram:



- A global variable (x) can be reached and modified anywhere in the code, local variable (z) exists only in block 3.
- **Example:**
  g=10 #global variable g def test():
  l=20 #local variable l print("local variable=",l)
  # accessing global variable print("Global variable=",g) test()
  print("global variable=",g)
- **output:**
  local variable= 20
  Global variable= 10
  global variable= 10

5(b) Example module. How to define module. (6M) Ans

- A module allows you to logically organize your Python code.
- Grouping related code into a module makes the code easier to understand and use.
- A module is a Python object with arbitrarily named attributes that you can bind and reference.
- Simply, a module is a file consisting of Python code.
- A module can define functions, classes and variables. A module can also include runnable code.

Example:The Python code for a module named aname normally resides in a file named

- aname.py. Here is an example of a simple module, support.py

```
def print_func( par ):
        print "Hello : ", par return
```

- To create a module just save the code you want in a file with the file extension .py:

Example:

- Save this code in a file named mymodule.py

```
def greeting(name):
        print("Hello, " + name)
```

Now we can use the module we just created, by using the import statement:

Example

Import the module named mymodule, and call the greeting function: import mymodule
mymodule.greeting("ABC")

Summer-22

2(c) Explain how to use user defined function in python with example. (4M) Ans:

- In Python, def keyword is used to declare user defined functions.
- The function name with parentheses (), which may or may not include parameters and arguments and a colon:
- An indented block of statements follows the function name and arguments which contains the body of the function.

Syntax:

```
def function_name():
        statements
        .
        .
```

Example:

```
def fun():
        print("User defined function")
fun()
```

output:

User defined function:

Parameterized function:

The function may take arguments(s) also called parameters as input within the opening and closing parentheses, just after the function name followed by a colon.

Syntax:

```
def function_name(argument1, argument2, ...): statements
        .
        .
```

Example:

```
def square( x ):
        print("Square=",x*x)

# Driver code square(2)
```

Output:

Square= 4

3(d) Write a program for importing module for addition and subtraction of two numbers. (4M) Ans:

calculation.py:

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

return (x-y) operation.py:

import calculation print(calculation.add(1,2))
print(calculation.sub(4,2)) **Output:**
3
2

4(c) Explain use of format() method with example. (4M) Ans:

- The format() method formats the specified value(s) and insert them inside the string's placeholder.
- The placeholder is defined using curly brackets: {}.
- The format() method returns the formatted string.
- **Syntax :**

  string.format(value1, value2...)
- **Example:**

```
#named indexes:
>>>txt1 = ("My name is {fname}, I'm {age}".format(fname = "abc", age = 36))
>>>print(txt1)
My name is abc, I'm 36
#numbered indexes:
>>>txt2 =( "My name is {0}, I'm {1}".format("xyz",36))
>>>print(txt2)
My name is xyz, I'm 36

#empty placeholders:
>>>txt3 = ("My name is {}, I'm {}".format("pqr",36))
>>>print(txt3)
My name is pqr, I'm 36
```

4(e) Write a program illustrating use of user defined package in python. (4M) Ans:

- A package is a hierarchical file directory structure that defines a single Python application environment that consists of modules and subpackages and sub- subpackages, and so on.
- **Packages** allow for a hierarchical structuring of the module namespace using **dot notation**.
- Creating a package is quite straightforward, since it makes use of the operating system's inherent hierarchical file structure.

- **Consider the following arrangement:**
- Here, there is a directory named pkg that contains two modules, mod1.py and mod2.py.
- **The contents of the modules are:**

mod1.py

```
def m1():
        print("first module")
```

d2.py

def m2():

- print("second module")
- If the pkg directory resides in a location where it can be found, you can refer to the two modules with dot notation(pkg.mod1, pkg.mod2) and import them with the syntax:
- **Syntax-1:**

  import <module_name>[, <module_name> ...]
- **Example:**

  >>>import pkg.mod1, pkg.mod2

  >>> pkg.mod1.m1()

first module

- **Syntax-2:**

  from <module_name> import <name(s)>
- **Example:**

  >>> from pkg.mod1 import m1

  >>> m1()

  First module

  >>>

  Syntax-3:

  from <module_name> import <name> as <alt_name>

  Example:

  >>> from pkg.mod1 import m1 as module

  >>> module() first module

  You can import modules with these statements as well:

  from <package_name> import <modules_name>[, <module_name> ...] from <package_name> import <module_name> as <alt_name> Example:

  >>> from pkg import mod1

  >>> mod1.m1() First module

6(a) Explain package NumPy with example.(6M) Ans:

NumPy package:

Same as Winter 2023-4(b)

Using NumPy, a developer can perform the following operations:

1. Mathematical and logical operations on arrays.
2. Fourier transforms and routines for shape manipulation.
3. Operations related to linear algebra.
4. NumPy has in-built functions for linear algebra and random number generation

Example:

For NumPy with array object.

```
>>> import numpy as np
>>> a=np.array([1,2,3]) # one dimensional array
>>> print(a) [1 2 3]
>>> arr=np.array([[1,2,3],[4,5,6]]) # two dimensional array
>>> print(arr) [[1 2 3]
[4 5 6]]
>>> type(arr)
<class 'numpy.ndarray'>
>>> print("No. of dimension: ", arr.ndim) No. of dimension: 2
>>> print("Shape of array: ", arr.shape) Shape of array: (2, 3)
>> >print("size of array: ", arr.size) size of array: 6
>>> print("Type of elements in array: ", arr.dtype) Type of elements
in array: int32
>>> print("No of bytes:", arr.nbytes) No of bytes: 24
```

5- Object Oriented Programming in Python

Winter-23

1(d) Define Data Hiding concept? Write two advantages of Data Hiding. (2M) Ans:

Data hiding:

- Data hiding is a concept which underlines the hiding of data or information from the user.
- Data hiding is a software development technique specifically used in Object-Oriented Programming (OOP) to hide internal object details (data members).
- Data hiding includes a process of combining the data and functions into a single unit to conceal data within a class by restricting direct access to the data from outside the class.

Advantages of Data Hiding

- Data hiding ensures exclusive data access to class members and protects object integrity by preventing unintended or intended changes.
- Data hiding is also known as information hiding. An object's attributes may or may not be visible outside the class definition.
- Data hiding also minimizes system complexity for increase robustness by limiting interdependencies between software requirements.
- The objects within the class are disconnected from irrelevant data.
- It heightens the security against hackers that are unable to access confidential data.
- It helps to prevent damage to volatile data by hiding it from the public.

- A user outside from the organization cannot attain the access to the data.
- Within the organization/ system only specific users get the access. This allows better operation.

3(d) Explain method overloading and overriding in python. (4M) Ans:

Method Overloading:

- Method overloading is the ability to define the method with the same name but with a different number of arguments and data types.
- With this ability one method can perform different tasks, depending on the number of arguments or the types of the arguments given.
- Method overloading is a concept in which a method in a class performs operations according to the parameters passed to it.
- Python does not support method overloading, that is, it is not possible to define more than one method with the same name in a class in Python.
- This is because method arguments in python do not have a type.
- A method accepting one argument can be called with an integer value, a string or a double as shown in next example.
- **Example:**

```
class Demo:
    def method(self, a):
            print(a)   obj= Demo( )
obj.method(50) obj.method('Meenakshi')
obj.method(100.2)
```

Output:

```
50
Meenakshi 100.2
```

- It is clear that method overloading is not supported in python but that does not mean that we cannot call a method with different number of arguments.
- There are a couple of alternatives available in python that make it possible to call the same method but with different number of arguments.

Method Overriding:

- Overriding is the ability of a class to change the implementation of a method provided by one of its base class.
- Method overriding is thus a strict part of the inheritance mechanism.
- To override a method in the base class, we must define a new method with same name and same parameters in the derived class.
- Overriding is a very important part of OOP since it is the feature that makes inheritance exploit its full power.

- Through method overriding a class may "copy" another class, avoiding duplicated code, and at the same time enhance or customize part of it.
- **Example:** For method overriding.

```
class A: # parent class
        #Parent Class def display(self):
                print ('This is base class.') class B(A): #derived
class
        #Child/Derived class def display(self):
                print ('This is derived class.') obj = B() #
        instance of child
        obj.display() # child calls overridden method
```

Output:

This is derived class.

6 (a) Write a program to create class student with Roll no. and Name and display its contents. (6M)

Ans:

```
class Student:
        def __init__(self, name, rollno): self.name = name
                self.rollno = rollno
        def __str__(self):
                return f"{self.name}({self.rollno})"

s1 = Student("ABC", 32) print(s1)
```

Output:

ABC 32

6 (a) Write program to implement concept of inheritance in python. (6M)

Ans:

```
class Animal: #super class
# attribute and method of the parent class name = ""
        def eat(self):
                print("I can eat")
# inherit from Animal class Dog(Animal):
        # new method in subclass def display(self):
                # access name attribute of superclass using self print("My name is ",
                self.name)

# create an object of the subclass labrador = Dog()
# access superclass attribute and method
```

labrador.name = "Rohu" labrador.eat()
# call subclass method labrador.display()
Output:
I can eat
My name is Rohu

Summer-23

1(d) With neat example explain default constructor concept in Python. (2M) Ans:
Default constructor:

- The default constructor is simple constructor which does not accept any arguments.
- It's definition has only one argument which is a reference to the instance being constructed.

Example : Display Hello message using default constructor. class Student:

```
    def _ _init_ _(self):
            print("This is non parametrized constructor") def
    show(self,name):
            print("Hello",name)
```

s1 = Student() s1.show("Student1") **Output:**
This is non parametrized constructor Hello Student1

3(d) Describe 'Self Parameter with example.(4M) Ans:

- In Python, the self-parameter is a convention used in object-oriented programming (OOP) to refer to the instance of a class within the class itself.
- It allows you to access the attributes and methods of the class from within its own methods. The name self is not a keyword in Python, but it is widely adopted and recommended as a convention.

Example:

```
    class Car:
            def _init_(self, make, model, year): self.make = make
                    self.model = model  self.year = year

            def get_info(self):
                    info = f"Make: {self.make}, Model: {self.model}, Year: {self.year}" return info
```

```
    def start_engine(self):
        print("Engine started!") # Create an

instance of the Car class my_car = Car("Toyota", "Corolla",

2022)

# Access the attributes using the self parameter
print(my_car.make) # Output: Toyota print(my_car.model) #
Output: Corolla print(my_car.year) # Output: 2022

# Call the method using the self parameter
car_info = my_car.get_info()
print(car_info) # Output: Make: Toyota, Model: Corolla, Year: 2022 # Call the method

that does not require any additional parameters my_car.start_engine() # Output:

Engine started!
```

6(b) Design a class student with data members : name, roll no., department, mobile no. Create suitable methods for reading and printing student information.(6M)

Ans:

```
class Student:
    def __init__(self):
        self.name = "" self.roll_no = ""
        self.department = "" self.mobile_no = ""

    def read_student_info(self):
        self.name = input("Enter student name: ") self.roll_no = input("Enter
        roll number: ") self.department = input("Enter department: ")
        self.mobile_no = input("Enter mobile number: ")

    def print_student_info(self): print("Student Information:")
        print("Name:", self.name) print("Roll Number:",
        self.roll_no)
        print("Department:", self.department) print("Mobile
        Number:", self.mobile_no)

    # Create an instance of the Student class student = Student()
```

```
# Read and set student information

student.read_student_info() # Print student
information

student.print_student_info()
```

Output:

```
Enter student name: raj Enter roll number:
11
Enter department: computer Enter mobile
number: 123456 Student Information:
Name: raj
Roll Number: 11 Department: computer
Mobile Number: 123456
```

6(c) With suitable example explain inheritance in Python. (6M) Ans:

Inheritance:

- In inheritance objects of one class procure the properties of objects of another class.
- Inheritance provides code usability, which means that some of the new features can be added to the code while using the existing code.
- The mechanism of designing or constructing classes from other classes is called inheritance.
- The new class is called derived class or child class and the class from which this derived class has been inherited is the base class or parent class.
- In inheritance, the child class acquires the properties and can access all the data members and functions defined in the parent class. A child class can also provide its specific implementation to the functions of the parent class.
- **Syntax:**

```
class A:
# properties of class A

class B(A):
# class B inheriting property of class A # more properties
of class B
```

Example:

```
# Base class class Animal:
def __init__(self, name):
        self.name = name def speak(self):
        print("Animal speaks")
```

```
# Derived class inheriting from Animal class Dog(Animal):
        def speak(self):
                print("Dog barks")
# Derived class inheriting from Animal class Cat(Animal):
        def speak(self):
                print("Cat meows")

# Create instances of derived classes dog =
Dog("Buddy")
cat = Cat("Whiskers")

# Call the speak method of the derived classes dog.speak() #
Output: Dog barks
cat.speak() # Output: Cat meows
```

Winter-22

1(d) Write syntax of defining class in Python. (2M) Ans:

```
class <ClassName>:
        <statement1>
        <statement2>
        .
        .
        <statementN>
```

3(c) Explain method overloading and overriding in python. (4M) Ans:

Same as Winter 2023-3(d)

4(b) What is command line argument? Write python code to add b) two numbers given as input from command line arguments and print its sum. (4M)

Ans:

- Python Command line arguments are input parameters passed to the script when executing them. Almost all programming language provide support for command line arguments.
- Then we also have command line options to set some specific options for the program.
- There are many options to read python command line arguments.
- The three most common ones are:

  Python sys.argv Python getopt module
  Python argparse module

  Program:

```
import sys
x=int(sys.argv[1]) y=int(sys.argv[2])
sum=x+y
```

```
print("The addition is :",sum)
```

Output:

C:\Python34\python sum.py 6 4 The addition is : 10

6(b) Design a class student with data members : name, roll no., department, mobile no. Create suitable methods for reading and printing student information.(6M)

Ans:

Same as Summer 2023-6(b)

6(c) Create a parent class named Animals and a child class Herbivorous which will extend the class Animal. In the child class Herbivorous over side the method feed ( ). Create a object. (6M) Ans:

```
# parent class class Animal:
 # properties
        multicellular = True
        # Eukaryotic means Cells with Nucleus eukaryotic = True
        # function breath def breathe(self):
                print("I breathe oxygen.")
 # function feed
        def feed(self):
                print("I eat food.")

# child class
class Herbivorous(Animal):
# function feed
        def feed(self):
                 print("I eat only plants. I am vegetarian.") herbi =

Herbivorous()

herbi.feed()
# calling some other function herbi.breathe()
```

Output:

I eat only plants. I am vegetarian. I breathe oxygen.

Summer-22

1(e) Define class and object in python. (2M) Ans:

Class:

A class is a user-defined blueprint or prototype from which objects are created. Classes provide a means of bundling data and functionality together.

Object:
An object is an instance of a class that has some attributes and behavior. Objects can be used to access the attributes of the class.

3(d) Write a program to create class EMPLOYEE with ID and NAME and display its contents.(4M) Ans:

```
class employee :
    id=0 name=""
     def getdata(self,id,name): self.id=id
            self.name=name
    def showdata(self):
            print("ID :", self.id) print("Name :", self.name)

e = employee() e.getdata(11,"Vijay")
e.showdata()
```

Output:

```
ID : 11
Name : Vijay
```

5(b) Explain method overloading in python with example. (6M) Ans:

Same as Winter 2023-3(d)

6(b) Write a program to implement the concept of inheritance in python.(6M) Ans:

Same as Summer 2023-6(c)

Unit 6- File I/O Handling and Exception Handling

Winter-23

1(f) State the use of read() and readline () functions in python file handling. (2M) Ans:

1. read([n]) Method:
The read method reads the entire contents of a file and returns it as a string, if number of bytes are not given in the argument. If we execute read(3), we will get back the first three characters of the file.

Example: for read( ) method.

```
f=open("sample.txt","r")
print(f.read(5)) # read first 5 data print(f.read()) # read
rest of the file
```

2. readline([n]) Method:

The readline() method just output the entire line whereas readline(n) outputs at most n bytes of a single line of a file. It does not read more than one line. Once, the end of file is reached, we get empty string on further reading.

Example: For readline ( ) method.

```
f=open("sample.txt","r")
print(f.readline()) # read first line followed by\n print(f.readline(3))
print(f.readline())
```

3(a) Describe various modes of file object? Explain any two in detail. (4M) Ans:

Like, C, C++, and Java, a file in Python programming can be opened in various modes depending upon the purpose. For that, the programmer needs to specify the mode whether read 'r', write 'w', or append 'a' mode. Apart from this, two other modes exist, which specify to open the file in text mode or binary mode.

1. The text mode returns strings while reading from the file. The default is reading in text mode.
2. The binary mode returns bytes and this is the mode to be used when dealing with non-text files like image or executable files.

The text and binary modes are used in conjunction with the r, w, and a modes. The list of all the modes used in Python are given in following table:

Sr. No.	Mode	Description
1	r	Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
2	rb	Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.
3	r+	Opens a file for both reading and writing. The file pointer placed at the beginning of the file.
4	rb+	Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.
5	w	Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
6	wb	Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
7	w+	Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
8	wb+	Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
9	a	Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
10	ab	Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.

11	a+	Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
12	ab+	Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
13	t	Opens in text mode (default).
14	b	Opens in binary mode
15	+	Opens a file for updating (reading and writing).It returns the length of the list.

4(c) Explain seek ( ) and tell ( ) function for file pointer manipulation in python with example. (4M)
Ans:
seek():

- seek() function is used to shift/change the position of file object to required position.
- By file object we mean a cursor. And it's cursor, who decides from where data has to be read or write in a file.
- **Syntax:**
  f.seek(offset, fromwhere)
- where offset represents how many bytes to move fromwhere, represents the position from where the bytes are moving.
- **Example:**
  f = open("demofile.txt", "r")
  f.seek(4) #sets Reference point to fourth index position from the beginning print(f.readline())

tell():

- tell() returns the current position of the file pointer from the beginning of the file.
- **Syntax:** file.tell()
- **Example:**
  f = open("demofile.txt", "r") # points at the
  start print(f.tell())

4(c) WAP to read contents of first.txt file and write same content in second.txt file. (4M) Ans:

```
with open('first.txt', 'r') as f: # Open the first file for reading contents = f.read() #
    Read the contents of the file
with open('second.txt', 'w') as f: # Open the second file for writing f.write(contents) # Write the
    contents of the first file to the second file
```

Summer-23
1(f) Describe mkdir() function. (2M) Ans:

- We can make a new directory using the mkdir() method.
- This method takes in the path of the new directory. If the full path is not specified, the new directory is created in the current working directory.

- **Syntax:**
  os.mkdir("newdir")
- **Example:**
  >>> import os
  >>> os.mkdir("testdir")

2(c) With neat example differentiate between readline () and readlines ( ) functions in file-handling. (4M)

Ans:

readline():

- This method reads a single line from the file and returns it as a string.
- It moves the file pointer to the next line after reading.
- If called again, it will read the subsequent line.

Example:

```
# Open the file in read mode
file = open("example.txt", "r")
# Read the first line
line1 = file.readline()
print(line1)
# Read the second line
line2 = file.readline()
print(line2)
# Close the file
file.close()
```

readlines():

- This method reads all lines from the file and returns them as a list of strings.
- Each line is an individual element in the list.
- It reads the entire file content and stores it in memory.

Example:

```
# Open the file in read mode file =
open("example.txt", "r") # Read all lines
lines = file.readlines() # Close the file
file.close()
# Print each line for line in lines:
 print(line)
```

4(c) Write a program to show user defined exception in Python. (4M) Ans:

```
class MyException(Exception):  def _init_(self,
        message):
                self.message = message

# Function that raises the custom exception def

divide_numbers(a, b):
```

```
    if b == 0:
        raise MyException("Division by zero is not allowed!") return a / b
# Main program try:
    num1 = int(input("Enter the numerator: ")) num2 =
    int(input("Enter the denominator: ")) result =
    divide_numbers(num1, num2) print("Result:", result)
except MyException as e: print("Exception:", e.message)
```

Note: Any correct program of user defined exception can be considered. Output:

Enter the numerator: 10 Enter the
denominator: 0
Exception: Division by zero is not allowed! Enter the
numerator: 10
Enter the denominator: 5 Result: 2.0

Winter-22

1(f) List file operations in Python. (2M) Ans:

File operations are:

- Opening file (using open() function)
- Reading file (using read() function)
- Writing file (using write() function)
- Copy files
- Delete files (using remove() function)
- Closing file (Using close() function)

3(d) Explain how try-catch block is used for exception handling in python. (4M)

Ans:

- In Python, exceptions can be handled using a try statement.
- A try block consisting of one or more statements is used by programmers to partition code that might be affected by an exception.
- A critical operation which can raise exception is placed inside the try clause and the code that handles exception is written in except clause.
- The associated except blocks are used to handle any resulting exceptions thrown in the try block. That is we want the try block to succeed and if it does not succeed, we want to control to pass to the catch block.
- If any statement within the try block throws an exception, control immediately shifts to the catch block. If no exception is thrown in the try block, the catch block is skipped.

- There can be one or more except blocks. Multiple except blocks with different exception names can be chained together.
- The except blocks are evaluated from top to bottom in the code, but only one except block is executed for each exception that is thrown.
- The first except block that specifies the exact exception name of the thrown exception is executed. If no except block specifies a matching exception name then an except block that does not have an exception name is selected, if one is present in the code.
- For handling exception in Python, the exception handler block needs to be written which consists of set of statements that need to be executed according to raised exception. There are three blocks that are used in the exception handling process, namely, try, except and finally.

1. **try Block**:
   A set of statements that may cause error during runtime are to be written in the try block.
2. **except Block:**
   It is written to display the execution details to the user when certain exception occurs in the program. The except block executed only when a certain type as exception occurs in the execution of statements written in the try block.

Syntax:

```
try:
    D the operations here
    .....
Exception1:
    If there is Exception1, then execute this block. except Exception2:
    If there is Exception2, then execute this block.
    .....
else:
    If there is no exception then execute this block.
```

Example: For try-except clause/statement. n=10

```
m=0 try:
    n/m
except ZeroDivisionError: print("Divide by zero error")
else:
    print (n/m)
```

Output:

Divide by zero error

4(c) Write python code to count frequency of each characters in a given file.(4M)

Ans:

```
import collections import pprint
```

```
file_input = input('File Name: ') with open(file_input,
'r') as info:
        count = collections.Counter(info.read().upper()) value =
    pprint.pformat(count)
print(value)
```

4(d) Write python program to read contents of abc.txt and write same content to pqr.txt.4M)

Ans:

```
with open('abs.txt','r') as firstfile, open('prq.txt','w') as secondfile: # read content
    from first file
    for line in firstfile:
        # write content to second file secondfile.write(line)
```

Summer-22

1(f) List different modes of opening file in Python. (2M) Ans:

Same as Winter 2023-3(a)

5(c) Write a program to open a file in write mode and append some content at the end of file.(6M)

Ans:

```
file1 = open("myfile.txt", "w")
L = ["This is Delhi \n", "This is Paris \n", "This is London"] file1.writelines(L)
file1.close()
# Append-adds at last # append mode
file1 = open("myfile.txt", "a")
# writing newline character file1.write("\n")
file1.write("Today")

# without newline character file1.write("Tomorrow")
file1 = open("myfile.txt", "r")
print("Output of Readlines after appending") print(file1.read())
print() file1.close()
```

Output: Output of Readlines after appending This is Delhi
This is Paris This is London
TodayTomorrow

6(b) Explain Try-except block used in exception handling in python with example.(6M) Ans: