CBCS SCHEME

USN | | | | | | | | | | BPLCK105B

First Semester B.E/B.Tech. Degree Examination, Dec.2024/Jan.2025
Introduction to Python Programming

Time: 3 hrs. Max. Marks:100

Note: *1. Answer any FIVE full questions, choosing ONE full question from each module.*
2. M : Marks , L: Bloom's level , C: Course outcomes.

		Module – 1	M	L	C
1	a.	Explain basic data types like int, float, double and string with an example.	6	L2	CO1
	b.	Differentiate between local scope and global scope.	6	L2	CO1
	c.	Develop a program to calculate factorial of a number. Program to compute binomial coefficient (Given N and R).	8	L3	CO1
		OR			
2	a.	Define functions. Explain how to pass parameters through the function with return statement.	6	L2	CO1
	b.	What is exception? How exception are handled in python? Write a program to solve divide by zero exception.	6	L2	CO1
	c.	Develop a program to generate Fibonacci sequence of length (N). Read N from the console.	8	L3	CO1
		Module – 2			
3	a.	Explain Augmented short hand assignment operators with an example.	7	L2	CO2
	b.	Explain different type of methods like append( ), Remove( ), sort( ), pop( ) in python programming list.	7	L2	CO2
	c.	Develop a program to find mean, variance and standard deviation.	6	L3	CO3
		OR			
4	a.	Explain set( ) and setdefault( ) method in dictionary.	7	L2	CO2
	b.	Develop a python to print area of rectangle.	6	L3	CO2
	c.	Define pretty printing. How does pretty print work in python with an example.	7	L2	CO2
		Module – 3			
5	a.	Explain useful string functions like : i) Capitalize ii) Count iii) Find iv) Lower v) Upper vi) Replace with an example.	8	L2	CO3
	b.	Develop a python code to determine whether given string is a palindrome or not a palindrome.	6	L3	CO3
	c.	Explain : i) isalpha ii) isalnum iii) isspace( ).	6	L2	CO3

		OR			
6	a.	Explain OS path module with an example.	6	L3	CO2
	b.	Explain the concept of file path. Also discuss absolute and relative file path.	8	L3	CO3
	c.	Program to print of multi clipboard with appropriate message.	6	L3	CO3
		Module – 4			
7	a.	Develop a program to backing up a given folder (folder in a current working directory) into a zip file by using relevant modules and suitable methods.	6	L3	CO4
	b.	List out the difference between shutil.copy( ) and shutil.copythree( ) method.	6	L1	CO4
	c.	Explain the following file operations in pythons with suitable example : i) Copying files and folders ii) Moving files and folders iii) Permanently deleting files and folders.	8	L2	CO4
		OR			
8	a.	Briefly explain assertion and raising a exception.	8	L2	CO4
	b.	List out the benefits of using logging module with an example.	6	L1	CO4
	c.	Write a function named DivExp which takes two parameters a, b and returns a value C(c= a/b). Write suitable assertion for a 70 in function DivExp and raise an exception for when b = 0. Develop a suitable program which reads two values from the console and calls a function DivExp.	6	L3	CO4
		Module – 5			
9	a.	Define a function which takes two objects representing complex numbers and returns a new complex number with a addition of two complex numbers. Define a suitable class 'complex' to represent the complex number. Develop a program to read N(N >= 2) complex numbers and compute the addition of 10 complex numbers.	8	L3	CO5
	b.	Explain the concept of inheritance with an example.	6	L2	CO5
	c.	Explain the _ str _ and the_init_ method with an example.	6	L2	CO5
		OR			
10	a.	Define a class and object, construct the class called rectangle and initialize it with height = 100, width = 200, starting point as (x = 0, y = 0). Write a program to display the centre pint co-ordinates of a rectangle.	8	L3	CO5
	b.	Briefly explain the printing of objects with an example.	6	L2	CO5
	c.	Differentiate operator over loading and operator overriding in python.	6	L2	CO5

* * * * *

VTU Question Paper Solution & Scheme
Introduction to Python Programming
Session Dec-24/Jan 2025

1.a Explain basic datatypes like int, float, double and string with an example [6 Marks]
Ans:

Explanation - [3 Marks]
Example & Code-[3 Marks]

Basic Data Types with Examples

1. Integer (int)
    - Represents whole numbers (positive, negative, or zero) without decimals.
    - Integers are used for counting and indexing

Example:
num = 10
print(type(num)) # Output: <class 'int'>

    -
2. Floating-Point (float)
    - Represents numbers with decimal points, including fractions.
    - Floats (and doubles) are used for precise calculations.

Example:
num = 10.5
print(type(num)) # Output: <class 'float'>

    -
3. **Double (`double`)**
    - Similar to `float` but with more precision.
    - **Note:** In **Python**, `float` acts as `double` since Python dynamically allocates precision.

Example:

```
num = 10.123456789012345
print(type(num))  # Output: <class 'float'>
```

    -
4. String (str)

- Represents a sequence of characters, enclosed in single or double quotes.
- **Strings** store and manipulate textual data.

Example:
python
CopyEdit
text = "Hello, World!"

- print(type(text)) # Output: <class 'str'>

b. Differentiate between local scope and global scope [7 Marks]
Ans: [explanation : 1 point each]

Feature	Local Scope	Global Scope
1. Definition	Variables declared inside a function, accessible only within that function.	Variables declared outside any function, accessible throughout the program.
2. Lifetime	Created when the function starts and destroyed when it ends.	Exists as long as the program runs.
3. Accessibility	Accessible only within the function where it is defined.	Accessible from anywhere in the program, including functions (unless shadowed by a local variable).
4. Modification	Can be modified only inside the function where it is declared.	Can be modified inside a function using the `global` keyword.
5. Memory Usage	Stored in function-specific memory, freeing space after execution.	Stored in global memory, consuming space until the program ends.
6. Namespace	Part of the local function's namespace.	Part of the global namespace.
7. Example	`python def func(): x = 5 # Local variable print(x) func() print(x) # Error: x is not defined globally`	`python x = 10 # Global variable def func(): print(x) # Accessible inside function func() print(x) # Accessible globally`

c. Develop a program to calculate factorial of a number. Program to compute binomial coefficient (Given N and R) [8 MArks]
Ans:
Full Program- [5 Marks]
Logic-[3 Marks]

PROGRAM:

```
def factorial(n):

    if n == 0 or n==1:
        return 1

    return n * factorial(n-1)

N = int(input("Enter N"))
R=int(input("Enter R"))
if N>R:
  bin=factorial(N)/(factorial(R)*factorial(N-R))

print("Factorial of", R, "is", factorial(R))
print("Factorial of", N, "is", factorial(N))
print("Binomial of", N,R ,"is", bin)
```

OUTPUT:

```
Enter N 3
Enter R 5

Factorial of 5 is 120
Factorial of 3 is 6
Binomial of 3 5 is 10.0
```

2. a. Define Functions. Explain how to pass parameters through the function with return statement. [6 Marks]
Ans:
Explanation - [3 Marks]
Example & Code-[3 Marks]

A function is like a mini-program within a program.
def Statements with Parameters:

```
def hello(name):
print('Hello ' + name)
hello('Alice')
hello('Bob')
```

Output:

```
Hello Alice
Hello Bob
```

Return Values and return Statements:
When you call the len() function and pass it an argument such as 'Hello', the function call evaluates to the integer value 5, which is the length of the string you passed it. In general, the value that a function call evaluates to is called the return value of the function.
When creating a function using the def statement, you can specify what the return value should be with a return statement. A return statement con- sists of the following:
• The return keyword
• The value or expression that the function should return

```
# Type 3: no parameter with returntype

def add():
    a=int(input("Enter 1st number:"))
    b=int(input("Enter 2nd number:"))
    return a+b
c=add()
print("Addition is:",c )
```

```
Enter 1st number: 2
Enter 2nd number: 2
Addition is: 4
```

```
# Type 4: with parameter with returntype

def add(a,b):
    #c=a+b
    return a+b
a=int(input("Enter 1st number:"))
b=int(input("Enter 2nd number:"))
print("Addition is:",add(a,b))
```

```
Enter 1st number: 2
Enter 2nd number: 2
Addition is: 4
```

b. What is exception? How exception are handled in python? Write a program to solve divide by zero exception [6 Marks]
Ans:
Explanation - [3 Marks]
Example & Code-[3 Marks]

Getting an error, or exception, in your Python program means the entire program will crash. You don't want this to happen in real-world pro- grams. Instead, you want the program to detect errors, handle them, and then continue to run.
For example, consider the following program, which has a "divide-by-zero" error. Open a new file editor window and enter the following code, saving it as zeroDivide.py:

```
def spam(divideBy):
return 42 / divideBy
print(spam(2))

print(spam(12))
print(spam(0))
print(spam(1))
```

We've defined a function called spam, given it a parameter, and then printed the value of that function with various parameters to see what hap- pens. This is the output you get when you run the previous code:
21.0
3.5
A ZeroDivisionError happens whenever you try to divide a number by zero. From the line

number given in the error message, you know that the return statement in spam() is causing an error.
Errors can be handled with try and except statements. The code that could potentially have an error is put in a try clause. The program execu- tion moves to the start of a following except clause if an error happens.
You can put the previous divide-by-zero code in a try clause and have an except clause contain code to handle what happens when this error occurs.

```
def spam(divideBy):
try:
    return 42 / divideBy
except ZeroDivisionError:
    print('Error: Invalid argument.')
print(spam(2))
print(spam(12))
print(spam(0))
print(spam(1))
```

c. Develop a program to generate Fibonacci sequence of length (N). Read N from the console. [8 Marks]
Ans:
Full Program- [5 Marks]
Logic-[3 Marks]

Program:

```
n=int(input("enter the number"))
a=0
b=1
sum=0
i=1
print("fibonacci series")

while(i<=n):
   sum=a+b
   print(a)
   a=b
   b=sum
   i=i+1
```

OUTPUT:

enter the number 5

fibonacci series
0
1
1
2
3

3.a. Explain Augmented shorthand assignment operators with an example. [7 Marks]
Ans:
Explanation - [3 Marks]
Example & Code-[4 Marks]

- Augmented shorthand assignment operators are a concise way to update the value of a variable by applying an operation and assigning the result back to the same variable.
- These operators combine arithmetic or bitwise operations with assignment, reducing redundancy in code.
- Example:
  variable op= expression

  This is equivalent to:
  variable = variable op expression

Operator	Equivalent to	Example
+=	a=a+b	a+=b
-=	a=a-b	a-=b
*=	a=a*b	a*=b
/=	a=a/b	a/=b
//=	a=a//b	a//=b
%=	a=a%b	a%=b
&=	a=a&b	a&=b
^=	a=a^b	a^=b
<<=	a=a<<b	a<<=b
>>=	a=a>>b	a>>=b

Example with multiple operators:

```
a = 8
a *= 2  # Equivalent to a = a * 2
print(a)  # Output: 16

b = 20
b //= 3  # Equivalent to b = b // 3
print(b)  # Output: 6
```

b. Explain different type of methods like append(), remove(), sort(), pop() in python programming list. [7 Marks]
Ans:

append():

append() method call adds the argument to the end of

the list.

```
>>> spam = ['cat', 'dog', 'bat']

>>> spam.append('moose')

>>> spam
```

['cat', 'dog', 'bat', 'moose']

remove():

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']

>>> spam.remove('bat')

>>> spam
```

['cat', 'rat', 'elephant']

- Attempting to delete a value that does not exist in the list will result in a ValueError error.
- If the value appears multiple times in the list, only the first instance of the value will be removed.

sort(): The sort() method will sort the items inside the list in ascending order

```
>>> spam = [2, 5, 3.14, 1, -7]

>>> spam.sort()

>>> spam

[-7, 1, 2, 3.14, 5]

>>> spam = ['ants', 'cats', 'dogs', 'badgers', 'elephants']
```

```
>>> spam.sort()
>>> spam
['ants', 'badgers', 'cats', 'dogs', 'elephants']
```

pop():

The pop() method removes the element at the specified position.

example:

```
fruits = ['apple', 'banana', 'cherry']
fruits.pop(1)
```

c. Develop a program to find mean, variance and standard deviation. [8 Marks]
Ans:
Full Program- [5 Marks]
Logic-[3 Marks]

PROGRAM:

```
n = int(input("Enter the range of value to be read => "))
lst = []
for i in range(n):
  lst.append(int(input("Enter the value => ")))
mn = sum(lst)/n
print("The mean of all the list number = > %.2f"%mn)
vr = 0
for i in lst:
  vr += (i-mn)**2
print("The variance of all the list number => %.2f"%(vr/n))
print("The standard deviation of all the list number => %.2f"%((vr/n)**(1/2)))
```

OUTPUT:

```
Enter the range of value to be read =>  5
Enter the value =>  1
Enter the value =>  2
Enter the value =>  3
Enter the value =>  4
Enter the value =>  5

The mean of all the list number = > 3.00
The variance of all the list number => 2.00
```

The standard deviation of all the list number => 1.41

Ans 4a.

The get() Method

Dictionaries have a get() method that takes two arguments: the key of the value to retrieve and a fallback value to return if that key does not exist.

Enter the following into the interactive shell:

```
>>> picnicItems = {'apples': 5, 'cups': 2}
>>> 'I am bringing ' + str(picnicItems.get('cups', 0)) + ' cups.'
'I am bringing 2 cups.'
>>> 'I am bringing ' + str(picnicItems.get('eggs', 0)) + ' eggs.'
 'I am bringing 0 eggs.'
```

The setdefault() Method

You'll often have to set a value in a dictionary for a certain key only if that key does not already have a value. The code looks something like this:

```
spam = {'name': 'Pooka', 'age': 5}
 if 'color' not in spam:
 spam['color'] = 'black'
```

The setdefault() method offers a way to do this in one line of code. The first argument passed to the method is the key to check for, and the second argument is the value to set at that key if the key does not exist. If the key does exist, the setdefault() method returns the key's value.

```
>>> spam = {'name': 'Pooka', 'age': 5}
>>> spam.setdefault('color', 'black')
'black'
>>> spam
 {'color': 'black', 'age': 5, 'name': 'Pooka'}
 >>> spam.setdefault('color', 'white')
'black'
>>> spam
 {'color': 'black', 'age': 5, 'name': 'Pooka'}
```

The first time setdefault() is called, the dictionary in spam changes to {'color': 'black', 'age': 5, 'name': 'Pooka'}. The method returns the value 'black' because this is now the value set for the key 'color'. When spam.setdefault('color', 'white') is called next, the value for that key is *not* changed to 'white' because spam already has a key named 'color'.

The setdefault() method is a nice shortcut to ensure that a key exists.

program that counts the number of occurrences of each let- ter in a string.

```
message = 'It was a bright cold day in April, and the clocks were striking thirteen.'
count = {}
for character in message:
 count.setdefault(character, 0)
 count[character] = count[character] + 1
print(count)
```

Output:

```
{' ': 13, ',': 1, '.': 1, 'A': 1, 'I': 1, 'a': 4, 'c': 3, 'b': 1, 'e': 5, 'd': 3, 'g': 2, 'i':
6, 'h': 3, 'k': 2, 'l': 3, 'o': 2, 'n': 4, 'p': 1, 's': 3, 'r': 5, 't': 6, 'w': 2, 'y': 1}
```

Ans 4b.Develop a python to print area of Rectangle?

```
import numpy as np

x = np.array(5) #  length
y = np.array(8) #  width

area = np.multiply(x, y)
print(area)
```

Output:

40

Ans 4C:**Define Pretty printing.How does pretty print work in python with an example?**

Ans:Pretty Printing

If you import the pprint module into your programs, you'll have access to the pprint() and pformat() functions that will "pretty print" a dictionary's values. This is helpful when you want a cleaner display of the items in a dictionary than what print() provides.

program that counts the number of occurrences of each let- ter in a string.

```
import pprint
message = 'It was a bright cold day in April, and the clocks were striking
thirteen.' count = {}
for character in message:
        count.setdefault(character, 0)
        count[character] = count[character] + 1
pprint.pprint(count)
```

This time, when the program is run, the output looks much cleaner, with the keys sorted.

```
 {' ': 13,
 ',': 1,
 '.': 1,
 'A': 1,
 'I': 1,
 'a': 4,
 'b': 1,
```

'c': 3,
'd': 3,
'e': 5,
'g': 2,
'h': 3,
'i': 6,
' k': 2,
'l': 3,
'n': 4,
'o': 2,
'p': 1,
'r': 5,
's': 3,
't': 6,
'w': 2,
'y': 1}

The pprint.pprint() function is especially helpful when the dictionary itself contains nested lists or dictionaries.

If you want to obtain the prettified text as a string value instead of dis- playing it on the screen, call pprint.pformat() instead. These two lines are equivalent to each other:

```
pprint.pprint(someDictionaryValue)
print(pprint.pformat(someDictionaryValue))
```

Module-3

5a.Explain useful string functions like:

i.capitalize

The capitalize() method returns a string where the first character is upper case, and the rest is lower case.

Syntax

string.capitalize()

ii.count

The count() method returns the number of elements with the specified value.

Syntax

```
list.count(value)
```

Iii.find

The find() method finds the first occurrence of the specified value.

The find() method returns -1 if the value is not found.

The find() method is almost the same as the index() method, the only difference is that the index() method raises an exception if the value is not found. (See example below)

Syntax

```
string.find(value, start, end)
```

```
Iv.lower
```

The lower() method returns a string where all characters are lower case.

Symbols and Numbers are ignored.

Syntax

```
string.lower()
```

```
V.upper
```

The upper() method returns a string where all characters are in upper case.

Symbols and Numbers are ignored.

Syntax

```
string.upper()
```

```
Vii.repalce
```

The replace() method replaces a specified phrase with another specified phrase.

Syntax

```
string.replace(oldvalue, newvalue, count)
```

5b.Develop python code to determine whether given string is palindrome or not palindrome?

Ans:

```
s = "malayalam" # string
```

```
if s == s[::-1]:
    print("Yes")
else:
    print("No")
```

OUTPUT:yes

5c.Explain: isapla,isalnum,isspace

isalpha:

The isalpha() method returns True if all the characters are alphabet letters (a-z).

```
txt = "Company10"
x = txt.isalpha()
print(x)
```

OUTPUT:

False

Isalnum

The isalnum() method returns True if all the characters are alphanumeric, meaning alphabet letter (a-z) and numbers (0-9).

```
txt = "Company 12"
x = txt.isalnum()
print(x)
```

OUTPUT:

Yes

isspace:

The isspace() method returns True if all the characters in a string are whitespaces, otherwise False.

```
txt = "    s    "

x = txt.isspace()

print(x)
```

OUTPUT:

Yes

6a.Explain OS path module with an example?

Ans:

The os.path module also has some useful functions related to absolute and relative paths:
Calling os.path.abspath(path) will return a string of the absolute path of the argument. This is an easy way to convert a relative path into an absolute one.
Calling os.path.isabs(path) will return True if the argument is an absolute path and False if it is a relative path.
Calling os.path.relpath(path, start) will return a string of a relative path from the start path to path. If start is not provided, the current working directory is used as the start path.
Try these functions in the interactive shell:

```
>>> os.path.abspath('.')

'C:\\Users\\Al\\AppData\\Local\\Programs\\Python\\Python37'
>>> os.path.abspath('.\\Scripts')
'C:\\Users\\Al\\AppData\\Local\\Programs\\Python\\Python37\\Scripts'
>>> os.path.isabs('.')
False
>>> os.path.isabs(os.path.abspath('.'))
True
```

6b**.**Explain the concept of file path. And also explain the Difference between absolute and relative path.

Files and File Paths

A file has two key properties: a filename and a path. The path specifies the location of a file on the computer. For example, there is a file on my Windows laptop with the filename project.docx in the path C:\Users\Al\Documents. The part of the filename after the last period is called the file's extension and a file's type. The filename project.docx is a Word document, and Users, Al, and Documents all refer to folders(also called directories). Folders can contain files and other folders. For example, project.docx is in the Documents folder, which is inside the Al folder, which is inside the Users folder

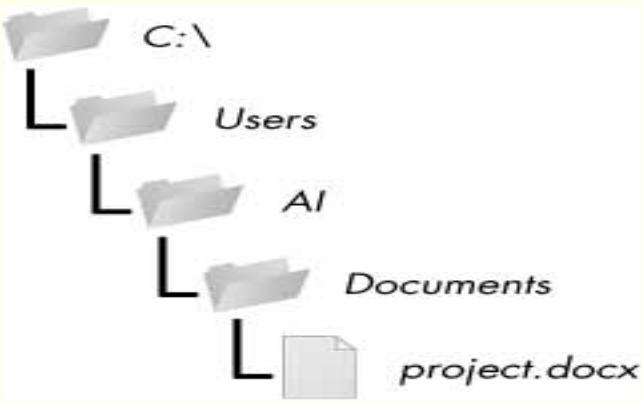

Figure 9-1: A file in a hierarchy of folders

The C:\ part of the path is the root folder, which contains all other folders. On Windows, the root folder is named C:\ and is also called the C: drive.

Absolute vs. Relative Paths

There are two ways to specify a file path:

An absolute path, which always begins with the root folder

A relative path, which is relative to the program's current working directory

There are also the dot (.) and dot-dot (..) folders. These are not real folders

but special names that can be used in a path. A single period ("dot")

for a folder name is shorthand for "this directory." Two periods ("dot-dot")

means "the parent folder."

Below is an example of some folders and files. When the current working

directory is set to C:\bacon, the relative paths for the other folders and files are

set as they are in the figure.

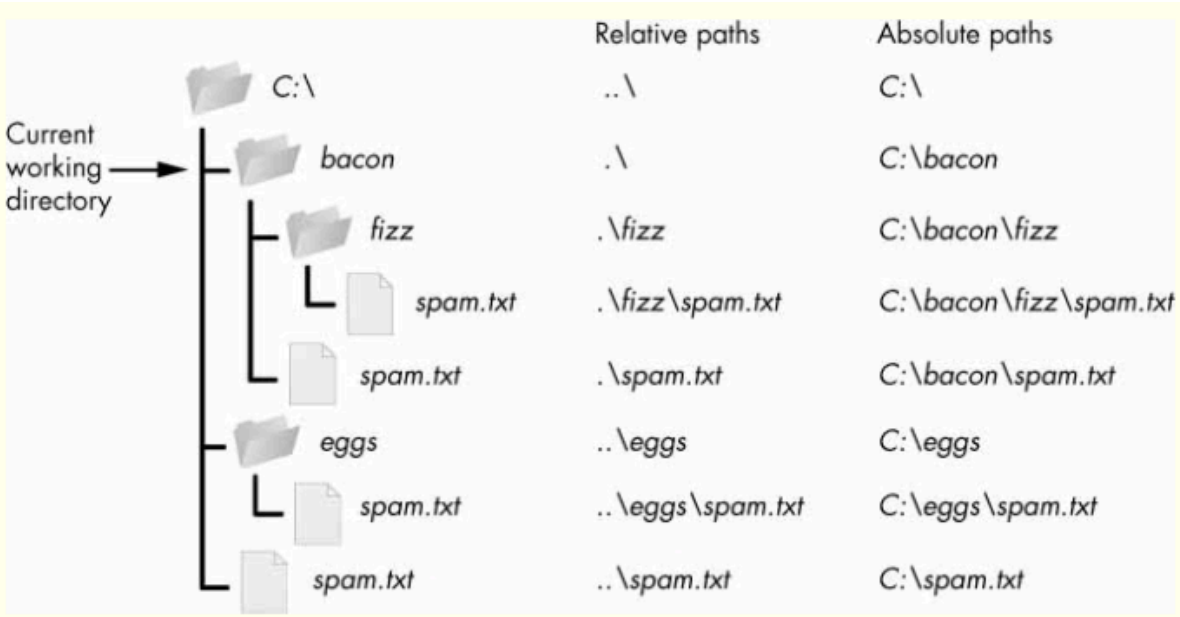

Figure 9-2: The relative paths for folders and files in the working directory C:\bacon

6c.Program to print of multi clipboard with appropriate message.

Ans:*Step 1: Comments and Shelf Setup*

```
❶ # Usage: py.exe mcb.pyw save <keyword> - Saves clipboard to keyword.
  #        py.exe mcb.pyw<keyword> - Loads keyword to clipboard.
  #        py.exe mcb.pyw list - Loads all keywords to clipboard.

❷ import shelve, pyperclip, sys

❸ mcbShelf = shelve.open('mcb')

  # TODO: Save clipboard content.

  # TODO: List keywords and load content.

  mcbShelf.close()
```

Step 2: Save Clipboard Content with a Keyword

```
❶ if len(sys.argv) == 3 and sys.argv[1].lower() == 'save':
       ❷ mcbShelf[sys.argv[2]] = pyperclip.paste()
  eliflen(sys.argv) == 2:
  ❸ # TODO: List keywords and load content.

mcbShelf.close()
```

Step 3: List Keywords and Load a Keyword's Content

```
# Save clipboard content.
if len(sys.argv) == 3 and sys.argv[1].lower() == 'save':
        mcbShelf[sys.argv[2]] = pyperclip.paste()
eliflen(sys.argv) == 2:
    # List keywords and load content.
  ❶ if sys.argv[1].lower() == 'list':
    ❷ pyperclip.copy(str(list(mcbShelf.keys())))
    elifsys.argv[1] in mcbShelf:
    ❸ pyperclip.copy(mcbShelf[sys.argv[1]])

mcbShelf.close()
```

Ans **7(a)** Python program for backing up a folder into a zip file

```
import os

import shutil

from datetime import datetime



def backup_folder(source_folder, backup_location):

    if not os.path.exists(source_folder):

        print("Source folder does not exist!")

        return

    # Generate a unique backup file name with timestamp

    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")

    backup_filename = f"backup_{timestamp}.zip"

    backup_path = os.path.join(backup_location, backup_filename)

    try:

        # Create a ZIP archive
```

```
        shutil.make_archive(backup_path.replace('.zip', ''), 'zip', source_folder)
        print(f"Backup created successfully: {backup_path}")
    except Exception as e:
        print(f"Error during backup: {e}")

# Example usage
source_folder = "path/to/source/folder"  # Change this to the folder you want to back up
backup_location = "path/to/backup/location"  # Change this to where you want to store the backup
backup_folder(source_folder, backup_location)
```

Ans 7(b) In Python's `shutil` module, `shutil.copy()` and `shutil.copytree()` serve distinct purposes for copying files and directories:

- `shutil.copy()`:
  - Purpose: Copies a single file from a source to a destination.
  - Usage: `shutil.copy(src, dst)`[stackoverflow.com+2blog.csdn.net+2realpython.com+2](#)
    - `src`: Path to the source file.
    - `dst`: Destination path or directory.
  - Behavior:
    - If `dst` is a directory, the source file is copied into this directory with its original name.
    - If `dst` is a filename, the source file is copied and renamed to the specified filename.
  - Metadata: Copies the file's content and permissions but does not preserve other metadata like timestamps.[sopriza.com+2realpython.com+2designgurus.io+2](#)
- `shutil.copytree()`:
  - Purpose: Recursively copies an entire directory tree (a directory and all its subdirectories and files) to a new location.[tracedynamics.com+1realpython.com+1tracedynamics.com+1](#)
  - Usage: `shutil.copytree(src, dst, dirs_exist_ok=False)`[designgurus.io+5blog.csdn.net+5techgeekbuzz.com+5](#)

        - `src`: Path to the source directory.
        - `dst`: Destination directory path.
        - `dirs_exist_ok`: If set to `True`, allows copying into an existing directory; defaults to `False`, which raises an error if the destination exists.
    - Behavior:
        - Creates the destination directory and recursively copies all files and subdirectories from the source to the destination.
        - Raises an error if the destination directory already exists and `dirs_exist_ok` is `False`.
    - Metadata: Attempts to preserve the metadata of files and directories, such as timestamps and permissions.

Key Differences:

- Scope:
    - `shutil.copy()`: Designed for copying individual files.scribd.com+1coderslegacy.com+1scribd.com+1
    - `shutil.copytree()`: Intended for copying entire directories, including all nested files and subdirectories.
- Destination Requirements:
    - `shutil.copy()`: The destination can be an existing directory or a new filename.
    - `shutil.copytree()`: The destination directory must not already exist unless `dirs_exist_ok=True` is specified.designgurus.io+1coderslegacy.com+1designgurus.io+1
- Metadata Handling:
    - `shutil.copy()`: Does not preserve all metadata; for full metadata preservation, use `shutil.copy2()`.
    - `shutil.copytree()`: Preserves metadata by default, using `shutil.copy2()` for copying files.

Ans 7(c) 1. Copying Files and Folders

- Python provides the `shutil` module to copy files and folders.
- `shutil.copy()` is used to copy a file, and `shutil.copytree()` is used to copy an entire directory.

Example;

```
import shutil

# Copy a single file
```

```
shutil.copy("source.txt", "destination.txt")

# Copy an entire folder

shutil.copytree("source_folder", "destination_folder")
```

2. Moving Files and Folders

- The `shutil.move()` function moves files or directories from one location to another
- Example:

```
import shutil

# Move a file

shutil.move("file.txt", "new_directory/file.txt")

# Move a folder

shutil.move("source_folder", "destination_folder")
```

3. Permanently Deleting Files and Folders

- The `os` and `shutil` modules provide methods to delete files and folders permanently.
- `os.remove()` is used to delete a file.
- `shutil.rmtree()` is used to delete a folder along with its contents

Example:

```
import os

import shutil

# Delete a file

os.remove("file.txt")

# Delete an empty folder

os.rmdir("empty_folder")

# Delete a folder with all its contents

shutil.rmtree("folder_to_delete")
```

Ans- 8(a) Assertion (`assert` statement)

- Assertions are used to check if a condition is `True` during runtime.
- If the condition is `False`, an `AssertionError` is raised.
- Used mainly for debugging and testing.

Example: x = 10

assert x > 5 # Passes, as 10 > 5

assert x < 5, "x should be less than 5" # Fails and raises AssertionError

Raise Exception (`raise` statement)

- The `raise` statement is used to manually trigger an exception.
- It is useful for handling unexpected conditions in a controlled manner.

Example:

```
def check_age(age):
    if age < 18:
        raise ValueError("Age must be 18 or above.")
    print("Access granted.")


check_age(15)  # Raises ValueError
```

ANs- 8(b)

The logging module in Python provides a flexible framework for emitting log messages from applications. Here are some benefits of using it:

1. Better Debugging & Error Tracking
    - Helps track errors and understand the flow of execution.
    - Can log exceptions and other critical issues.
2. Different Logging Levels
    - Supports various levels (`DEBUG`, `INFO`, `WARNING`, `ERROR`, `CRITICAL`), making it easy to filter messages.
3. Configurable Output
    - Logs can be written to files, consoles, or even external services.

4. Thread-Safe Logging
    - Can be safely used in multi-threaded applications without causing conflicts.
5. Custom Formatting
    - Provides structured and readable logs by defining custom formats.
6. Performance Optimization
    - More efficient than `print()` as it allows selective logging.
7. Persistent Logging
    - Unlike `print()`, logs can be stored persistently in files or databases.
8. Flexibility with Handlers
    - Supports multiple handlers to send logs to different destinations (console, file, network, etc.).

Example:

```
import logging

# Configure logging

logging.basicConfig(

    filename="app.log",  # Log output to a file

    level=logging.DEBUG,  # Set the minimum logging level

    format="%(asctime)s - %(levelname)s - %(message)s",  # Define log format

)

# Sample log messages

logging.debug("This is a debug message.")

logging.info("This is an info message.")

logging.warning("This is a warning message.")

logging.error("This is an error message.")

logging.critical("This is a critical message.")

# Example with exception handling

try:

    result = 10 / 0
```

```
except ZeroDivisionError:
    logging.exception("An exception occurred: Division by zero")
print("Logging completed! Check app.log for details.")
```

Ans 8 (c)

```
 def DivExp(a, b):
    """Function to divide two numbers with assertions and exception handling."""
    assert isinstance(a, (int, float)), "a must be a number"
    assert isinstance(b, (int, float)), "b must be a number"
    if b == 0:
        raise ZeroDivisionError("Division by zero is not allowed")
    return a / b
# Read values from the console
try:
    a = float(input("Enter value for a: "))
    b = float(input("Enter value for b: "))
    result = DivExp(a, b)
    print(f"Result: {result}")
except ZeroDivisionError as e:
    print(f"Error: {e}")
except AssertionError as e:
    print(f"Assertion Error: {e}")
except ValueError:
    print("Invalid input! Please enter numerical values.")
```

Ans- 9(a) Python program that defines a `Complex` class, a function to add two complex numbers, and a program to read and compute the sum of 10 complex numbers.

```
class Complex:
    """Class to represent a complex number."""
    def __init__(self, real, imag):
        self.real = real
        self.imag = imag
    def __add__(self, other):
        """Overloading + operator to add two complex numbers."""
        return Complex(self.real + other.real, self.imag + other.imag)
    def __str__(self):
        """String representation of the complex number."""
        return f"{self.real} + {self.imag}i"
def add_complex_numbers(complex_list):
    """Function to add a list of complex numbers."""
    result = Complex(0, 0)
    for c in complex_list:
        result += c
    return result
# Read N complex numbers (N ≥ 2)
N = 10  # We need to sum 10 complex numbers
complex_numbers = []
for i in range(N):
    real = float(input(f"Enter real part of complex number {i+1}: "))
```

```
    imag = float(input(f"Enter imaginary part of complex number {i+1}: "))

    complex_numbers.append(Complex(real, imag))

# Compute the sum of 10 complex numbers

sum_result = add_complex_numbers(complex_numbers)

print(f"\nSum of the {N} complex numbers: {sum_result}")
```

Ans -9 (b) Inheritance is an Object-Oriented Programming (OOP) concept where a class (child/derived class) can acquire the properties and behaviors (methods) of another class (parent/base class). This allows code reusability and modularity.

Types of Inheritance

1. **Single Inheritance** – One child class inherits from a single parent class.
2. **Multiple Inheritance** – A child class inherits from multiple parent classes.
3. **Multilevel Inheritance** – A child class inherits from another child class.
4. **Hierarchical Inheritance** – Multiple child classes inherit from a single parent class.
5. **Hybrid Inheritance** – A combination of two or more types of inheritance.

Example: SIngle inheritance

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

    def speak(self):
        return "I make a sound"

# Child Class inheriting from Animal
class Dog(Animal):
    def speak(self):
        return "Bark! Bark!"

# Creating objects
animal = Animal("Generic Animal")
dog = Dog("Buddy")

# Accessing methods
print(f"{animal.name}: {animal.speak()}")  # Generic Animal: I make a sound
print(f"{dog.name}: {dog.speak()}")        # Buddy: Bark! Bark!
```

Explanation

- Parent Class (Animal): Defines a name attribute and a speak() method.
- Child Class (Dog): Inherits Animal but overrides the speak() method.
- Objects Created: Animal prints a generic sound, while Dog prints "Bark! Bark!".

Ans **9 c**

Explanation **of `__init__` and `__str__` Methods in Python**

1. `__init__` Method (Constructor)

- The `__init__` method is a special method in Python classes.
- It is called automatically when an object is created.
- It initializes the object's attributes.

2. `__str__` Method (String Representation)

- The `__str__` method returns a user-friendly string representation of an object.
- It is automatically called when `print(object)` is used.

```
class Person:
    def __init__(self, name, age):
        """Constructor to initialize name and age"""
        self.name = name
        self.age = age

    def __str__(self):
        """Returns a readable string representation of the object"""
        return f"Person(Name: {self.name}, Age: {self.age})"

# Creating an object
person1 = Person("Alice", 25)

# Printing the object
print(person1)  # Calls __str__() method automatically

Output:
Person(Name: Alice, Age: 25)
```

Ans 10 (a)

```
class Rectangle:
    """Class to represent a rectangle."""

    def __init__(self, height=100, width=200, x=0, y=0):
        self.height = height
        self.width = width
        self.x = x
        self.y = y

    def get_center(self):
        """Method to calculate the center coordinates of the rectangle."""
        center_x = self.x + (self.width / 2)
        center_y = self.y + (self.height / 2)
        return (center_x, center_y)

# Creating an object of Rectangle
rect = Rectangle()  # Default values: height=100, width=200, x=0, y=0

# Getting center point coordinates
center = rect.get_center()

# Displaying the center coordinates
print(f"Center Point Coordinates: {center}")
```

Ans 10 (b) When an object is printed using the `print()` function, Python looks for a `__str__()` method in the class. If it is defined, it returns a user-friendly string representation of the object. If `__str__()` is not defined, Python falls back to `__repr__()` or prints the default memory address.

```
class Person:
    def __init__(self, name, age):
        """Constructor to initialize name and age"""
        self.name = name
        self.age = age

    def __str__(self):
        """Returns a user-friendly string representation"""
        return f"Person(Name: {self.name}, Age: {self.age})"

# Creating an object
person1 = Person("Alice", 25)
```

```
# Printing the object
print(person1)  # Calls __str__() method automatically

OUTPUT:
Person(Name: Alice, Age: 25)
```

Ans **10 (c)**

Feature	**Operator Overloading**	**Operator Overriding**
Definition	Redefining how built-in operators work for user-defined classes.	Modifying the behavior of an inherited method in a subclass.
Purpose	Allows operators like `+`, `-`, `*`, etc., to work with user-defined objects.	Changes the behavior of a method inherited from a parent class.
Concept	Uses special methods (magic methods) like `__add__`, `__sub__`, etc.	Uses method overriding in inheritance.
Inheritance Required?	No, it works within a single class.	Yes, it requires inheritance.
Example Operators	`+`, `-`, `*`, `/`, `==`, `>`, `<`	Methods like `__str__()`, `__eq__()`, etc.

Example of operator overloading

```
class ComplexNumber:
    def __init__(self, real, imag):
        self.real = real
        self.imag = imag

    def __add__(self, other):
        return ComplexNumber(self.real + other.real, self.imag + other.imag)

    def __str__(self):
        return f"{self.real} + {self.imag}i"

# Creating objects
c1 = ComplexNumber(2, 3)
c2 = ComplexNumber(1, 4)
```

```
# Using overloaded '+' operator
c3 = c1 + c2  # Calls __add__()
print(c3)  # Output: 3 + 7i
```

Here, **+** is overloaded to work with complex numbers.

Operator riding

```
class Parent:
    def show(self):
        print("Parent class method")

class Child(Parent):
    def show(self):
        print("Child class method (Overriding Parent)")

# Creating objects
p = Parent()
c = Child()

p.show()  # Output: Parent class method
c.show()  # Output: Child class method
```

Key Takeaways

- **Overloading:** Same operator behaves differently for user-defined objects.
- **Overriding:** Child class modifies inherited behavior.