CBCS SCHEME

USN | | | | | | | | | | |

18CS55

Fifth Semester B.E. Degree Examination, Jan./Feb. 2023
Application Development using Python

Time: 3 hrs. Max. Marks: 100

Note: Answer any FIVE full questions, choosing ONE full question from each module.

Module-1

1 a. List the salient features of python programming language. **(04 Marks)**

b. Explain the math operators in python from highest to lowest precedence with an example for each. Show steps to evaluate (5 – 1) * (7 + 1) (3 – 1) in python. **(08 Marks)**

c. Write a program to find the sum of all odd and even numbers of n elements. Here skip the numbers which are divisible by 3. **(08 Marks)**

OR

2 a. Give one example for each of the operation below in python language:
i) String concatenation and replication.
ii) Read input and display. **(10 Marks)**

b. Explain:
i) Def statements with parameters
ii) Return values and return statements with an example. **(10 Marks)**

Module-2

3 a. Discuss the different ways of traversing a list. Explain each with an example. **(10 Marks)**

b. Write a python program that allows a player to guess a secret number within 6 chances. The code that lets the player enter a guess and checks that guess is right or not by printing appropriate message. List of numbers are taken as an input from the user. **(07 Marks)**

c. Write a program to demonstrate the use of pretty function. **(03 Marks)**

OR

4 a. Compare list and dictionary data structures with respect to python language. **(04 Marks)**

b. Write a program in python that counts. The number of occurrences of each letter in a string. Display the results in column fashion. **(08 Marks)**

c. Write the string method syntax in python to perform below operations.
i) Removing white space characters from beginning, end or both sides of a string.
ii) To right-justify, left-justify and center a string. **(08 Marks)**

Module-3

5 a. List out the different character classes. Give representation, regular expression symbols, example and meanings for each character class. **(10 Marks)**

b. Describe the following with suitable python code snippet:
i) Greedy and Non Greedy pattern matching.
ii) Findall( ) method of RegeX object. **(10 Marks)**

OR

6 a. Write a python program to rename the filename contains American style dates (MM – DD – YYYY) to European style dates (DD – MM – YYYY) in the working directory. **(10 Marks)**

b. What are Assertions in python? Explain with an example. **(05 Marks)**

c. Explain the file reading and writing process with suitable python program. **(05 Marks)**

Module-4

7 a. How objects are mutable by nature justify with an example? **(04 Marks)**

b. Discuss the methods isinstance( ) and hasattr( ) with suitable example for each. **(06 Marks)**

c. What is operator over loading? Write a program to add two point objects by overloading + operator. Also, overload – str – ( ) to display point as an ordered pair. **(10 Marks)**

OR

8 a. Define Inheritance. Explain with an example. **(06 Marks)**

b. Briefly discuss the importance of - -init- -( ) and - -str- ( ) methods in python. **(04 Marks)**

c. Demonstrate the polymorphism to generate histogram to count the number of times each letter appears in word and in sentence. **(10 Marks)**

Module-5

9 a. Analyze the steps involved in downloading and saving web page on to local system along with program. **(06 Marks)**

b. List any 4 CSS selectors of bs4 module. Using Beautiful soup passel, retrieve all of the paragraph tags in the web page www.amazon.com. **(06 Marks)**

c. How selenium module is useful to deal with web pages. What methods do it uses to simulate mouse clicks and keyboard keys? **(08 Marks)**

OR

10 a. Write a program to read the census data from the excel spreadsheet, count the number of census tracts in each country, count the total population of each country and prints the results.

1	Census tract	State	County	POP2010
9841	06075010500	CA	SanFrancsico	2685
9842	06075010600	CA	SanFrancisco	3894
9843	06075010700	CA	SanFrancisco	5592

(08 Marks)

b. How to zip the files and folders. Demonstrate with one example. **(04 Marks)**

c. Write a script that will go through every PDF in a folder and encrypt the PDFS using a password provided on the command line. Save each encrypted PDF with an-encrypted pdf suffix added to the original filename. **(08 Marks)**

* * * * *

VTU Question Paper Scheme and Solution

Sub:	Application Development using Python	Sub Code:	18CS55	Branch:	CSE

1

1a List the salient features of the python programming language? 4M

1.Easy to Read and Write: Python's syntax is clear and easy to understand, making it an ideal choice for beginners. It uses indentation to define code blocks, which enhances code clarity and readability[1].

2.*Extensive Standard Library: Python comes with a vast standard library that includes modules and packages for various tasks. This extensive library eliminates the need to write code from scratch for common functionalities like file handling, web development, data manipulation, and more[1].

3.Cross-Platform Compatibility: Python is a cross-platform language, which means you can write code on one platform (e.g., Windows) and run it on another (e.g., macOS or Linux) with little to no modification[1].

4.Dynamically Typed: Python programming is a dynamically typed language. It means you don't need to declare variable types explicitly. The interpreter determines the data type during runtime[1].

5. Object-Oriented Language: Python supports object-oriented language and concepts of classes, object encapsulation, etc[2].

6. GUI Programming Support: Graphical User interfaces can be made using a module such as PyQt5, PyQt4, wxPython, or Tk in Python[2].

7. **High-Level Language**: When we write programs in Python, we do not need to remember the system architecture, nor do we need to manage the memory[2].

8. Easy to Debug: Excellent information for mistake tracing. You will be able to quickly identify and correct the majority of your program's issues once you understand how to interpret Python's error traces[2].

1b Explain the math operators in python from highest to lowest precedence with an example for each. Show steps to evaluate (5-1)*(7+1)(3-1) in python 8M

In Python, mathematical operators have the following precedence from highest (executed first) to lowest (executed last):

1. **Parentheses** ( ): They have the highest precedence and can be used to force an expression to evaluate in the order you want.
   - Example: `(2 + 3) * 4` evaluates to 20.
2. **Exponentiation** **: This operator raises the number to its left to the power of the number to its right.
   - Example: `2 ** 3` evaluates to 8.

3. **Multiplication `*`, Division `/`, Floor Division `//`, and Modulus `%`**: These operators all have the same precedence. If they appear together, they are evaluated from left to right.
    - Example: `2 * 3 / 2` evaluates to `3.0`.
4. **Addition `+` and Subtraction `-`**: These also have the same precedence, which is lower than multiplication, division, etc. If they appear together, they are evaluated from left to right.
    - Example: `2 + 3 - 1` evaluates to 4.

Now let's evaluate `(5-1)*(7+1)(3-1)`:

This expression is not valid in Python because it's missing an operator between `(7+1)` and `(3-1)`. If you meant for these to be multiplied together, you should write it as `(5-1)*(7+1)*(3-1)`. Let's evaluate this corrected expression:

```
result = (5-1)*(7+1)*(3-1)
print(result)  # Outputs: 64
```

Here's how it's evaluated:

- First, the expressions in parentheses are evaluated: `(5-1)` becomes 4, `(7+1)` becomes 8, and `(3-1)` becomes 2.
- Then, these results are multiplied together: `4 * 8 * 2` equals 64. So, `(5-1)*(7+1)*(3-1)` evaluates to 64.

1c Write a program to find the sum of all and even numbers of n elements.Here skip the numbers which are divisible by 3 8M

Program:

```
def calculate_sums(n):
        total_sum = 0
        even_sum = 0

        for i in range(1, n+1):
        if i % 3 == 0:  # Skip numbers divisible by 3
        continue
        total_sum += i
        if i % 2 == 0:  # Check if the number is even
        even_sum += i

        return total_sum, even_sum

n = int(input("Enter a number: "))
total, even = calculate_sums(n)
print(f"The sum of all numbers from 1 to {n} (excluding multiples of 3) is {total}")
print(f"The sum of all even numbers from 1 to {n} (excluding multiples of 3) is {even}")
```

2a Give one example for each of the operation below in python language:

i) string concatenation and replication.

ii) Read input and display

```
# String Concatenation
str1 = "Hello"
str2 = "World"
concatenated_str = str1 + " " + str2
print(concatenated_str)  # Outputs: Hello World

# String Replication
replicated_str = str1 * 3
print(replicated_str)  # Outputs: HelloHelloHello

# Read Input and Display
user_input = input("Please enter something: ")
print("You entered: " + user_input)
```

2b. Explain:
i) Def statements with parameters
ii) Return values and return statements with an example

i) Def Statements with Parameters:

In Python, `def` is a keyword used for defining functions. Functions are blocks of reusable code that perform a specific task. When defining a function, you can specify parameters that the function takes as input.

Here's an example:

```
def greet(name):
    print(f"Hello, {name}!")
```

In this example, `greet` is the function name, and `name` is a parameter. When you call the function, you provide an argument for this parameter:

ii) Return Values and Return Statements:

A return statement is used to end the execution of the function call and sends the result back to the caller. The statements after the return statement are not executed.

Here's an example:

```
def add_numbers(num1, num2):
    return num1 + num2
```

In this example, `add_numbers` is a function that takes two parameters: `num1` and `num2`. The function adds these numbers together and then uses a return statement to send the result back:

```
result = add_numbers(3, 4)
print(result)

 # Outputs: 7
```

In this case, 7 is the return value of the function. It's important to note that a function in Python returns None if it doesn't have a return statement or if it ends without hitting a return statement.

Module 2

3a Discuss the different ways of traversing a list .Explain each with an example 10M

1. Using a for loop:

A `for` loop is the most common way to traverse a list. It iterates over each element in the list.

```
fruits = ['apple', 'banana', 'cherry']
for fruit in fruits:
    print(fruit)
```

2. Using list comprehension:

List comprehension is a concise way to create lists. It can also be used to traverse a list and perform some operation on each element.

```
fruits = ['apple', 'banana', 'cherry']
[print(fruit) for fruit in fruits]
```

3. Using the enumerate() function:

The `enumerate()` function adds a counter to the list and returns it as an enumerate object. This can be used to also get the index of each element while traversing.

```
fruits = ['apple', 'banana', 'cherry']
for i, fruit in enumerate(fruits):
    print(f"Element {i} is {fruit}")
```

4. Using while loop:

A `while` loop can also be used to traverse a list by using an index-based approach.

```
fruits = ['apple', 'banana', 'cherry']
i = 0
while i < len(fruits):
    print(fruits[i])
    i += 1
```

3b Write a python program that allows a player to guess a secret number within 6 chances. The code that lets the player enter a guess and checks that guess is

right or not by printing the appropriate message. List of numbers are taken as an input from the user 7M

```
import random

def guess_the_number():
        numbers = input("Enter numbers separated by space: ")
        numbers = list(map(int, numbers.split()))
        secret_number = random.choice(numbers)

        chances = 6
        while chances > 0:
        guess = int(input("Enter your guess: "))
        if guess == secret_number:
        print("Congratulations! You've guessed the number correctly.")
        return
        else:
        print("Sorry, that's not correct.")
        chances -= 1

        print(f"You've run out of chances. The secret number was {secret_number}.")

guess_the_number()
```

3c Write a program to demonstrate the use of pretty function 3M

The `pprint` function in the `pprint` module provides a capability to print Python data structures in a format that can be used as input to the interpreter. If you have nested structures, `pprint` can print them in a more readable way.

```
from pprint import pprint

data = {
        "name": "John Doe",
        "age": 30,
        "cities_visited": ["Paris", "Berlin", "London"],
        "bio": {
        "dob": "1990-01-01",
        "hobbies": ["Reading", "Traveling", "Swimming"]
```

```
        }
}
pprint(data)
```

it will print the `data` dictionary in a pretty and more readable way. The `pprint` function is especially useful when dealing with complex data structures.

4a Compare List and Dictionary data structures with respect to python language 4M

List:

- Lists are ordered collections of items. The order in which you insert elements into a list is the order in which they are stored.
- Lists are mutable, meaning you can change their content without changing their identity. You can modify a list by adding, removing, or changing elements.
- Elements in a list are accessed by their index, which is an integer value that represents the position of an element in the list.
- Lists are great to use when you want to work with many related values. They enable you to keep data together that belongs together, condense your code, and perform the same methods and operations on multiple values at once.

Example:

```
fruits = ['apple', 'banana', 'cherry']
print(fruits[0])  # Outputs: apple
```

Dictionary:

- Dictionaries are unordered collections of key-value pairs. Unlike lists, which are indexed by a range of numbers, dictionaries are indexed by keys, which can be any type (immutable).
- Like lists, dictionaries are mutable. You can add, remove, or modify elements in a dictionary.
- Dictionaries are known as associative arrays or hash maps in other programming languages.
- Dictionaries are great to use when you want to associate values with keys, so you can look them up efficiently (by key) later.

Example:

```
person = {'name': 'John', 'age': 30}
print(person['name'])  # Outputs: John
```

4b.Write a program in python that counts the number of occurrences of each letter in a string. Display the results in column fashion 8M

```
def count_letters(text):
        count_dict = {}
        for letter in text:
        if letter.isalpha():  # Check if character is a letter
        if letter in count_dict:
```

```
        count_dict[letter] += 1
        else:
        count_dict[letter] = 1

        # Display the results in column fashion
        print("Letter | Count")
        print("-------|------")
        for letter, count in sorted(count_dict.items()):
        print(f"  {letter}   |   {count}")

# Test the function
count_letters("Hello, World!")
```

In this program, `count_letters` is a function that takes a string as input. It creates a dictionary `count_dict` where each key is a letter and each value is the number of occurrences of that letter in the string. The function then prints the contents of this dictionary in column fashion. The `sorted` function is used to sort the dictionary items by key (i.e., alphabetically by letter) before printing. The `isalpha` method is used to check if a character is a letter.

4 C Write the string method syntax in python to perform below operations.i)Removing white space characters from beginning,end or both sides of a stringii) To right-justify,left-justify and center a string **8M**

i) Removing white space characters from beginning, end or both sides of a string:

```
# For removing leading spaces
string.lstrip()

# For removing trailing spaces
string.rstrip()

# For removing spaces from both sides
string.strip()
```

ii) To right-justify, left-justify and center a string:

```
# To right-justify a string
string.rjust(width)

# To left-justify a string
string.ljust(width)
```

To center a string

string.center(width)

5a List out the different character classes. Give representation,Regular expression symbols, examples and meanings for each character class 10M

some of the different character classes in Regular Expressions:

1. **\d**: This matches any decimal digit. It's equivalent to [0-9].
   - Example: "\d" would match "2" in "Python 2".
2. **\D**: This matches any non-digit character. It's equivalent to [^0-9].
   - Example: "\D" would match "P" in "Python 2".
3. **\s**: This matches any whitespace character (like: [\t\n\r\f\v]).
   - Example: "\s" would match the space in "Python 2".
4. **\S**: This matches any non-whitespace character.
   - Example: "\S" would match "P" in "Python 2".
5. **\w**: This matches any alphanumeric character (like: [a-zA-Z0-9_]).
   - Example: "\w" would match "P" in "Python 2".
6. **\W**: This matches any non-alphanumeric character.
   - Example: "\W" would match the space in "Python 2".
7. **[abc]**: This matches any of the characters a, b, or c.
   - Example: "[abc]" would match "a" in "Jack".
8. **[^abc]**: This matches any character except a, b, or c.
   - Example: "[^abc]" would match "J" in "Jack".

5b Describe the following with suitable python code snippet

i)Greedy and NonGreedy pattern matching.

ii) Findall() method of Regex object 10M

Sure, here are the explanations and Python code snippets for the topics you mentioned:

i) **Greedy and Non-Greedy Pattern Matching**: In regular expressions, greedy and non-greedy matching refer to the quantity of text a special character matches.

- **Greedy matching** (default behavior) matches as much text as possible.
- **Non-greedy matching** (also called lazy matching) matches as little text as possible.

Here's an example with Python's `re` module:

```
import re

text = "<html><head><title>Title</title>"

# Greedy pattern

greedy_pattern = re.compile("<.*>")

greedy_match = greedy_pattern.search(text)
```

print("Greedy match: ", greedy_match.group())

Outputs: <html><head><title>Title</title>

Non-greedy pattern

non_greedy_pattern = re.compile("<.*?>")

non_greedy_match = non_greedy_pattern.search(text)

print("Non-greedy match: ", non_greedy_match.group()) # Outputs: <html>

ii) **Findall() Method of Regex Object**: The `findall()` method returns all non-overlapping matches of pattern in string, as a list of strings. If one or more groups are present in the pattern, return a list of groups.

Here's an example:

```
import re

text = "Hello World! Hello Python!"

pattern = re.compile("Hello")

matches = pattern.findall(text)

print(matches)  # Outputs: ['Hello', 'Hello']
```

In this example, `findall()` finds all occurrences of "Hello" in the text. The result is a list of all matches.

6a Write a python program to rename the filename contains American style dates (MM-DD-YYYY) to European style dates (DD-MM-YYYY) in the working directory

```
import shutil, os, re

# Create a regex that matches files with the American date format.
datePattern = re.compile(r"""^(.*?) # all text before the date
    ((0|1)?\d)- # one or two digits for the month
    ((0|1|2|3)?\d)- # one or two digits for the day
    ((19|20)\d\d) # four digits for the year (must start with 19 or 20)
    (.*?)$ # all text after the date
    """, re.VERBOSE)

# Loop over the files in the working directory.
for amerFilename in os.listdir('.'):
    mo = datePattern.search(amerFilename)

    # Skip files without a date.
    if mo == None:
        continue
```

```
    # Get the different parts of the filename.
    beforePart = mo.group(1)
    monthPart  = mo.group(2)
    dayPart    = mo.group(4)
    yearPart   = mo.group(6)
    afterPart  = mo.group(8)

    # Form the European-style filename.
    euroFilename = beforePart + dayPart + '-' + monthPart + '-' + yearPart +
afterPart

    # Get the full, absolute file paths.
    absWorkingDir = os.path.abspath('.')
    amerFilename = os.path.join(absWorkingDir, amerFilename)
    euroFilename = os.path.join(absWorkingDir, euroFilename)

    # Rename the files.
    print('Renaming "%s" to "%s"...' % (amerFilename, euroFilename))
    #shutil.move(amerFilename, euroFilename) # uncomment after testing
```

6b What are assertions in python?Explain with an example

Assertions in Python are a debugging tool that lets you test if a condition in your code returns true, and if not, the program will raise an AssertionError with an optional error message.

Assertions are carried out by the `assert` statement in Python. The `assert` statement is used to continue the execute if the given condition evaluates to True. If the `assert` condition evaluates to False, then it raises the AssertionError exception with the specified error message.

Here's an example:

```
def apply_discount(product, discount):
        price = int(product['price'] * (1.0 - discount))
        assert 0 <= price <= product['price'], "The discounted price can't be
negative or more than original price"
        return price

product = {'name': 'iPhone', 'price': 70000}

print(apply_discount(product, 0.25))  # Outputs: 52500
print(apply_discount(product, 1.25))  # Raises AssertionError: The discounted
price can't be negative or more than original price
```

In this example, the `apply_discount` function calculates a discount on a product's price. The `assert` statement ensures that the discounted price is never less than zero or more than the original price. If this assertion fails, it raises an AssertionError with a custom error message. This helps catch bugs or incorrect values early in development.

6 c Explain the file reading and writing process with suitable python program 5M

Reading from a file:

You can read from a file using the `open()` function with mode `'r'` (read). Once the file is opened, you can read its contents with the `read()` method.

```
# Open the file in read mode ('r')
file = open('example.txt', 'r')
# Read the contents of the file
contents = file.read()
# Always close the file after you're done
file.close()
# Print the contents
print(contents)
```

Writing to a file:

You can write to a file using the `open()` function with mode `'w'` (write). Once the file is opened, you can write to it with the `write()` method.

```
# Open the file in write mode ('w')
file = open('example.txt', 'w')
# Write to the file
file.write("Hello, World!")
# Always close the file after you're done
file.close()
```

Please note that opening a file in write mode will erase its existing contents. If you want to append to the file without erasing its contents, use mode `'a'` (append).

Also, it's a good practice to use `with` statement when working with files. This ensures that the file is properly closed after it is no longer needed.

```
# Using 'with' for better practice
with open('example.txt', 'r') as file:
    print(file.read())
```

In this example, `file` is automatically closed outside the `with` block. This is more concise and avoids leaving the file open by accident

7a How objects are mutable by nature justify with an example? 4M

In Python, mutability is the ability of an object to change its state or contents after it has been created. Lists, sets, and dictionaries are examples of mutable objects in Python. This means you can change their content without changing their identity.

Here's an example with a list:

```
# Create a list

numbers = [1, 2, 3]

print(numbers)  # Outputs: [1, 2, 3]

# Modify the list

numbers.append(4)

print(numbers)  # Outputs: [1, 2, 3, 4]
```

In this example, we created a list `numbers` and then added an element to it with `append()`. The list `numbers` is still the same object as before (it has the same identity), but its content has changed. Therefore, we say that the list is mutable.

On the other hand, some objects like integers, floats, strings and tuples are immutable. This means once an object is created, it cannot be changed. Here's an example with a string:

```
# Create a string

greeting = "Hello"

# Try to change the string

greeting[0] = "J"  # Raises TypeError: 'str' object does not support item assignment
```

In this example, we tried to change the first letter of the string `greeting`, but Python raised a TypeError. This is because strings are immutable in Python - once a string is created, it cannot be changed.

7b Discuss the methods isinstance() and hasattr() with suitable example for each 6M

isinstance(): This method checks if an object is an instance or subclass of a class. It returns True if the object is an instance or subclass of a class, False otherwise.

Python

This code is AI-generated. Review and use carefully. Visit our FAQ for more information.

```
# Example for isinstance()

# Define a class

class MyClass:

    pass


# Create an instance of MyClass

my_instance = MyClass()
```

```
# Check if my_instance is an instance of MyClass

print(isinstance(my_instance, MyClass))  # Outputs: True

# Check if my_instance is an instance of another class (e.g., list)

print(isinstance(my_instance, list))  # Outputs: False
```

hasattr(): This method checks if an object has a given attribute. It returns True if the object has the attribute, False otherwise.

```
# Example for hasattr()

# Define a class with an attribute

class MyClass:

        my_attribute = "Hello"

# Create an instance of MyClass

my_instance = MyClass()

# Check if my_instance has the attribute 'my_attribute'

print(hasattr(my_instance, 'my_attribute'))  # Outputs: True

# Check if my_instance has another attribute (e.g., 'another_attribute')

print(hasattr(my_instance, 'another_attribute'))  # Outputs: False
```

7cWhat is operator overloading? Write program to add two point objects by overloading + operator. Also,overload -str-() to display point as an ordered pair

Operator overloading allows you to redefine the meaning of an operator (like `+`, `-`, `*`, etc.) for custom classes in Python. This is done by defining a special method in the class definition.

Here's an example of operator overloading with the `+` operator and the `__str__()` method for a `Point` class:

Python

This code is AI-generated. Review and use carefully. Visit our FAQ for more information.

Copy

```
class Point:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y
    # Overload the + operator
    def __add__(self, other):
        x = self.x + other.x
        y = self.y + other.y
        return Point(x, y)
```

```
    # Overload the __str__() method to display point as an ordered pair
    def __str__(self):
        return "({0}, {1})".format(self.x, self.y)
p1 = Point(1, 2)
p2 = Point(3, 4)
# Use the overloaded + operator
p3 = p1 + p2
# Use the overloaded __str__() method
print(p3)  # Outputs: (4, 6)
```

In this example, we've defined a `Point` class that represents a point in 2D space. We've overloaded the `+` operator with the `__add__()` method to allow us to add two `Point` objects together. We've also overloaded the `__str__()` method to print the `Point` object as an ordered pair. This makes it easier to work with and display `Point` objects in a way that makes sense in their context.

8a Define inheritance Explain with an example

Inheritance is a fundamental concept in Object-Oriented Programming (OOP) where a class (child class or subclass) can inherit properties and methods from another class (parent class or superclass). The main advantage of inheritance is reusability of code.

```
# Parent class
class Animal:
        def __init__(self, name):
        self.name = name
        def speak(self):
        return "I don't know what sound I make!"
# Child class
class Dog(Animal):
        def speak(self):
        return "Woof!"
# Child class
class Cat(Animal):
        def speak(self):
        return "Meow!"
# Create instances
```

```
dog = Dog("Rex")
cat = Cat("Whiskers")
print(dog.name)  # Outputs: Rex
print(dog.speak())  # Outputs: Woof!
print(cat.name)  # Outputs: Whiskers
print(cat.speak())  # Outputs: Meow!
```

8b Briefly discuss about __init__() and __str__() methods in python

1. `__init__()`: This is a special method in Python classes, known as a constructor. It's automatically called when an object of the class is instantiated. It's typically used to initialize the attributes of a class.

```
class Example:
    def __init__(self, value):
        self.value = value

e = Example(5)  # The __init__() method is called here
```

In this example, `__init__()` takes two arguments: `self` (which is a reference to the instance being created) and `value`, which is passed in when the `Example` class is instantiated.

2. `__str__()`: This is another special method that should return a string. It's used as a human-readable representation of the object, and is called by built-in functions like `print()` and `str()`.

```
class Example:
    def __init__(self, value):
        self.value = value
    def __str__(self):
        return "Example object with value " + str(self.value)

e = Example(5)
print(e)  # The __str__() method is called here
```

8c Demonstrate the polymorphism to generate histogram to count the number of times each letter in word and in sentence

Polymorphism in object-oriented programming allows us to use a single interface with different underlying forms. In Python, polymorphism allows us to define methods in the child class with the same name as defined in their parent class.

Here's an example of polymorphism where we create a histogram to count the number of times each letter appears in a word and a sentence:

Python

Copy

```
class Text:
    def __init__(self, text):
        self.text = text

    def histogram(self):
        hist = {}
        for char in self.text:
            hist[char] = hist.get(char, 0) + 1
        return hist

class Word(Text):
    def histogram(self):
        hist = super().histogram()
        return {k: v for k, v in sorted(hist.items(), key=lambda item: item[1], reverse=True)}

class Sentence(Text):
    def histogram(self):
        hist = super().histogram()
        return {k: v for k, v in sorted(hist.items(), key=lambda item: item[1], reverse=True) if k != '
'}

# Create a Word object
word = Word("hello")
print(word.histogram())  # Outputs: {'l': 2, 'h': 1, 'e': 1, 'o': 1}

# Create a Sentence object
sentence = Sentence("hello world")
print(sentence.histogram())  # Outputs: {'l': 3, 'o': 2, 'h': 1, 'e': 1, 'w': 1, 'r': 1, 'd': 1}
```

In this example, Word and Sentence are subclasses of Text. They both inherit the histogram method from Text, but override it to provide their own implementation. This is an example of polymorphism - the same method name (histogram) is used for different types (Word and Sentence), and the correct method is called based on the type of the object.

9a Analyze the steps involved in downloading and saving webpage on to local system along with program

1. Send a HTTP request to the URL of the webpage you want to access. The server responds to the request by returning the HTML content of the webpage.
2. Once you have accessed the HTML content, you can save it as a local HTML file on your system.

```
import os

import requests


# Make a request to the website

response = requests.get("http://www.example.com")

response.raise_for_status()  # Raise exception if invalid response


# Save the webpage content to a .html file

with open(os.path.join('path_to_directory', 'webpage.html'), 'wb') as file:

    for chunk in response.iter_content(100000):

        file.write(chunk)
```

In this program, replace 'path_to_directory' with the path where you want to save the HTML file.

Please note that this program only downloads the HTML content of the page, not any other resources like CSS, images, JavaScript, etc. If you want to download a full website for offline viewing (including all resources), you might want to look into web scraping tools or website downloaders that can recursively download all linked resources.

Also, be aware that not all websites allow their content to be downloaded or scraped. Always check the website's robots.txt file (e.g., http://www.example.com/robots.txt) and terms of service to ensure you are allowed to download or scrape their content. Be respectful and avoid overloading the website with too many requests in a short amount of time.

List any 4 CSS selectors for bs4 module using Beautifulsoup passel, retrieve all of the paragraph tags in the web page www.amazon.com

1. **Element Selector**: Selects elements based on the element name. For example, `soup.select('p')` selects all <p> elements.
2. **ID Selector**: Selects elements based on their id. For example, `soup.select('#some_id')` selects the element with the id `some_id`.
3. **Class Selector**: Selects elements based on their class. For example, `soup.select('.some_class')` selects all elements with the class `some_class`.
4. **Attribute Selector**: Selects elements based on an attribute and its value. For example, `soup.select('a[href="http://www.example.com"]')` selects all <a> elements with a href attribute of `http://www.example.com`.

To retrieve all of the paragraph tags in a webpage using BeautifulSoup, you can use the `find_all` method with the 'p' tag. Here's an example:

```
import requests
```

```
from bs4 import BeautifulSoup
# Send a GET request to the webpage
response = requests.get('http://www.amazon.com')
# Parse the content with BeautifulSoup
soup = BeautifulSoup(response.content, 'html.parser')
# Find all paragraph tags
paragraphs = soup.find_all('p')
# Print each paragraph's text
for p in paragraphs:
        print(p.get_text())
```

10 a How to Zip the files and folders. Demonstrate with one example

```
import os
import zipfile

def zipdir(path, ziph):
    # Iterate over all the directories and files in the path
    for root, dirs, files in os.walk(path):
        for file in files:
            # Write each file to the zip file
            ziph.write(os.path.join(root, file),
                       os.path.relpath(os.path.join(root, file),
                                       os.path.join(path, '..')))

# Create a ZipFile object
zipf = zipfile.ZipFile('example.zip', 'w', zipfile.ZIP_DEFLATED)

# Call the function with the path to folder you want to zip and the ZipFile object
zipdir('/path/to/folder', zipf)

# Close the ZipFile object
zipf.close()
```

In this example, `zipdir()` is a function that takes a path and a `ZipFile` object. It uses `os.walk()` to iterate over all directories and files in the path. For each file, it writes it to the zip file with its relative path (so that directories are preserved in the zip file). The `'..'` in

`os.path.join(path, '..')` is used to get the parent directory of `path`, so that `os.path.relpath()` gives us paths relative to the directory we want to zip.

Please replace '/path/to/folder' with your actual folder path that you want to compress. Also, make sure that you have read permissions for all files and folders that you're trying to compress. If not, this might raise a PermissionError.

10b Write a script that will go through every PDF in a folder and encrypt the PDFS using a password provided on the command line. Save each encrypted PDF with an encrypted pdf suffix added to the original filename

```
import os
import sys
import PyPDF2

# The password is provided as a command line argument
password = sys.argv[1]

# Go through every file in the current working directory
for filename in os.listdir('.'):
    if filename.endswith('.pdf'):
    pdf_file = open(filename, 'rb')
    pdf_reader = PyPDF2.PdfFileReader(pdf_file)

    # Check if PDF is already encrypted
    if pdf_reader.isEncrypted:
    print(f'{filename} is already encrypted.')
    else:
    # Create a PDF writer object
    pdf_writer = PyPDF2.PdfFileWriter()

    # Copy all pages from the reader to the writer object
    for page_num in range(pdf_reader.numPages):
        page_obj = pdf_reader.getPage(page_num)
        pdf_writer.addPage(page_obj)

    # Encrypt the writer object with the password
    pdf_writer.encrypt(password)
```

```
 # Create an encrypted version of the PDF
result_pdf = open(f'{filename[:-4]}_encrypted.pdf', 'wb')
pdf_writer.write(result_pdf)
result_pdf.close()
pdf_file.close()
```

This script takes a password as a command line argument, goes through every PDF in the current working directory, and creates an encrypted copy of each PDF. The encrypted copy has `_encrypted.pdf` added to the original filename.

Please note that this script doesn't delete the original, unencrypted PDFs. If you want to delete the originals after encryption, you can do so with `os.unlink(filename)`, but be careful with this as it's not reversible.

Also, please make sure to install the `PyPDF2` library using pip:

pip install PyPDF2

And always remember to respect copyright laws and only encrypt PDFs that you have permission to modify.

This script should be run from the command line like so:

python encrypt_pdfs.py yourpassword

Replace "yourpassword" with the password you want to use for encryption. Make sure your command line is in the directory where your PDFs are located.

[7]
CO1
L3