Python-Module-2-Important-Topics

- 2. Write a Python code segment that opens a file for input and prints the number of four-letter words in the file.
- 3. Write a Python program to create a set of functions that compute the mean,median and mode of a set of numbers. Each function should expect a list of numbers as an argument and return a single number. Each function should return 0 if the list is empty. Include a main function that tests the three functions with a given list.
- 4. Write a Python program to check whether a list contains a sublist.
- 5. Assume that the variable data refers to the string "Python rules!". Use a string method to perform the following tasks:
- 6. Use higher order python function filter to extract a list of positive numbers from a given list of numbers. You should use a lambda to create the auxiliary function
- 7. Assume that there is a text file named “numbers.txt”. Write a python program to find the median of list of numbers in the file without using standard function for median
- 8. Write a Python program to convert a decimal number to its binary equivalent.
- 9. Write a Python program to read a text file and store the count of occurrences of each character in a dictionary
- 10. Write a Python code to create a function called listof frequency that takes a string and prints the letters in non-increasing order of the frequency of their occurrences. Use dictionaries.
- 11. Write a Python program to read a list of numbers and sort the list in a non-decreasing order without using any built in functions. Separate function should be written to sort the list wherein the name of the list is passed as the parameter.
- 12. Illustrate the following Set methods with an example.
- 13. Write a Python program to check the validity of a password given by the user.
- 14.Create a dictionary of names and birthdays. Write a Python program that asks the user to enter a name, and the program display the birthday of that person

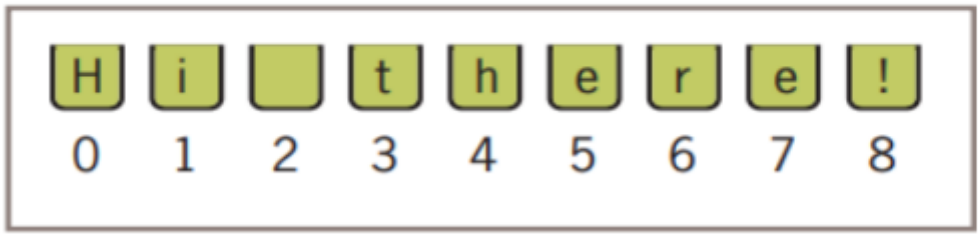
1. Strings

What is a string?

- A String is a sequence of 0 or more characters
- We can mention a python string using either single quote marks or double quote marks

- Example: `"Hello"`
- A string's length is the number of characters it contains. Python's len function returns this value
  - `len("Hi there!")`
  - This returns 9
- The positions of a string's characters are numbered from 0, on the left, to the length of the string minus 1, on the right.

The sequence of characters and their positions in the string "Hi there!"



- The string is an **immutable data structure.** This means that its internal data elements, the characters, can be accessed, but cannot be replaced, inserted, or removed.

Subscript operator

- `[]` is the subscript operator
- We can use this to access any element of the string

```
name = "Jack Dorsey"
name[0]  # This will return 'J', which is the first character

name[3] # This will return 'k', which is the 4th character
```

Slicing

```
name = "Jack Dorsey"
name[0:1] # This Returns 'Ja' which is the first and 2nd characters
name[0:2] # This Returns 'Jac' which is the first,2nd  and 3rd characters
name[3:]  # 'k Dorsey' Starts from the 4th character

name[-3:] # 'sey', Last 3 characters
```

String Methods

String Methods/ String function

String Method	What it Does
`s.center(width)`	Returns a copy of **s** centered within the given number of columns.
`s.count(sub [, start [, end]])`	Returns the number of non-overlapping occurrences of substring **sub** in **s**. Optional arguments **start** and **end** are interpreted as in slice notation.
`s.endswith(sub)`	Returns **True** if **s** ends with **sub** or **False** otherwise.
`s.find(sub [, start [, end]])`	Returns the lowest index in **s** where substring **sub** is found. Optional arguments **start** and **end** are interpreted as in slice notation.
`s.isalpha()`	Returns **True** if **s** contains only letters or **False** otherwise.
`s.isdigit()`	Returns **True** if **s** contains only digits or **False** otherwise.
`s.join(sequence)`	Returns a string that is the concatenation of the strings in the sequence. The separator between elements is **s**.
`s.lower()`	Returns a copy of **s** converted to lowercase.
`s.replace(old, new [, count])`	Returns a copy of **s** with all occurrences of substring **old** replaced by **new**. If the optional argument **count** is given, only the first **count** occurrences are replaced.
`s.split([sep])`	Returns a list of the words in **s**, using **sep** as the delimiter string. If **sep** is not specified, any whitespace string is a separator.
`s. startswith(sub)`	Returns **True** if **s** starts with **sub** or **False** otherwise.
`s.strip([aString])`	Returns a copy of **s** with leading and trailing whitespace (tabs, spaces, newlines) removed. If **aString** is given, remove characters in **aString** instead.
`s.upper()`	Returns a copy of **s** converted to uppercase.

2. Text files

- A text file is a software object that stores data on a permanent medium such as a disk, CD, or flash memory.

Writing Text to a File

- Data can be output to a text file using a file object.
- Python‘s open function, which expects a file name and a mode string as arguments, opens a connection to the file on disk and returns a file object.
- For reading, we will use 'r' mode
- For writing, we will use 'w' mode

Example

```
f = open("myfile.txt", 'w')
```

- This opens myfile.txt in write mode

```
f.write("First line.\nSecond line.\n")
```

- This will write the string into myfile.txt
- When we are done writing to a file, we should close it
- We can close a file by `f.close()`

Example-Reading a file

```
file = open("myfile.txt", "r")
line = file.readline()
```

- This will open the textfile and read one line

3. Functions

What is a function?

- A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function. A function can return data as a result.
- In Python, a function is a group of related statements that performs a specific task

- Functions help break our program into smaller and modular chunks. As our program grows larger and larger, functions make it more organized and manageable.
- Furthermore, it avoids repetition and makes the code reusable.

Syntax of a function

```
def function_name(parameters):
        statement(s)
```

- Keyword def that marks the start of the function header.
- A function name to uniquely identify the function.
- Parameters (arguments) through which we pass values to a function. They are optional.
- A colon (:) to mark the end of the function header.
- One or more valid python statements that make up the function body.
- Statements must have the same indentation level (usually 4 spaces).
- An optional return statement to return a value from the function

Calling a function

- To call a function, use the function name followed by parenthesis:
  Example

```
def my_function():
        print("Hello from a function")
my_function() # calling a function
```

4. Higher order functions

What is a higher order function?

- A function that is having another function as an argument or a function that returns another function as a return in the output is called the high order function.

Properties of high order functions in python

1. In high order function, we can store a function inside a variable
2. We can return a function as a result of another function
3. We can pass a function as a parameter or argument inside another function
4. We can store python high order function in data structure format

Types of HIgh order functions

Mapping

- The first type of useful high order function is called mapping
- This process applies a function to each value in a sequence and returns a new sequence of results

Example

```
>>> words = ["231", "20", "-45", "99"]
>>> words = list(map(int, words)) # Convert all strings to int
>>> [231, 20, -45, 99]
```

Filtering

- A second type of higher-order function is called a filtering. In this process, a function called a predicate is applied to each value in a list.
- If the predicate returns True, the value passes the test and is added to a filter object (similar to a map object).
- The process is a bit like pouring hot water into a filter basket with coffee.
- The good stuff to drink comes into the cup with the water, and the coffee grounds left behind can be thrown on the garden.

Example

```
def odd(n):
        return n%2 == 1
print(list(filter(odd,range(10))))
```

Output

```
[1, 3, 5, 7, 9]
```

- This uses odd function as the predicate
- The predicate checks whether the number is odd
- So, only odd numbers will be there in the list

Reducing

- Our final example of a higher-order function is called a reducing. Here we take a list of values and repeatedly apply a function to accumulate a single data value.
- A summation is a good example of this process. The first value is added to the second value, then the sum is added to the third value, and so on, until the sum of all the values is produced.

Example

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

def multiply(x,y):
        return x * y
data = [1,2,3,4]
print(reduce(add,data))
print(reduce(multiply,data))
```

Output

```
10
24
```

- Here `print(reduce(add,data))` adds the elements in the list and returns 10
- `print(reduce(multiply,data))` multiplies the elements together and returns 24

5. Lambda Functions

What is Lambda function?

- A lambda is an anonymous function. It has no name of its own, but contains the names of its arguments as well as a single expression.

- When the lambda is applied to its arguments, its expression is evaluated, and its value is returned.

Syntax of lambda function

```
lambda <argname-1, ..., argname-n>: <expression>
```

- All code must appear on one line

Example

Example-1

```
z = lambda x,y:x+y
print(z(2,3))
```

- The above lambda function accepts arguments x and y
- And it returns x+y as output
- So z(2,3) = 2+3 = 5

Example-2

```
def myfunc(n):
        return lambda a:a*n
mydoubler = myfunc(2)
mytripler = myfunc(3)
print(mydoubler(11))
print(mytripler(11))
```

- The above lambda function takes a as parameter
- And returns a * n

6. List

What is a List?

- A list is a sequence of data values called items or elements. An item can be of any type. Each of the items in a list is ordered by position.
- Each item in a list has a unique index that specifies its position.
- The index of the first item is 0, and the index of the last item is the length of the list minus 1.
- In Python, a list is written as a sequence of data values separated by commas. The entire sequence is enclosed in square brackets `[ ]`.

Example

```
[123,456,890] # list of integers
['apples','oranges'] # list of strings
[ ] # An empty list
```

List methods

LIST METHOD	WHAT IT DOES
`L.append(element)`	Adds **element** to the end of **L**.
`L.extend(aList)`	Adds the elements of **aList** to the end of **L**.
`L.insert(index, element)`	Inserts **element** at **index** if **index** is less than the length of **L**. Otherwise, inserts **element** at the end of **L**.
`L.pop()`	Removes and returns the element at the end of **L**.
`L.pop(index)`	Removes and returns the element at **index**.

7. Set

What is a set?

- A set is a collection which is unordered and un-indexed. In Python, sets are written with curly brackets `{ }`
- Sets are unordered, so you cannot be sure in which order the items will appear.

- **Accessing Items**: You cannot access items in a set by referring to an index or a key.

Example

```
thisset = {"apple","banana","cherry"}
print(thisset)
```

Set Methods

Add items

- To add one item to a set use the add() method. To add more than one item to a set use the update() method.

```
# add() method
thisset = {"apple","banana","cherry"}
thisset.add("orange")
print(thisset)

#Update() method
thisset = {"apple","banana","cherry"}
thisset.update(["orange","mango","grapes"])
print(thisset)
```

Output

```
{'apple', 'orange', 'banana', 'cherry'}
{'banana', 'orange', 'cherry', 'apple', 'mango', 'grapes'}
```

Length of a set

- Using len(), we can get length of a set

```
thisset = {"apple", "banana", "cherry"}
print(len(thisset)) # Output: 3
```

Remove item

We can use the following methods

```
thisset.remove("banana")
thisset.discard("banana")
x = thisset.pop()
```

Other methods

Python Sets Methods	
Method	**Description**
add()	Adds an element to the set
clear()	Removes all the elements from the set
copy()	Returns a copy of the set
difference()	Returns a set containing the difference between two or more sets
difference_update()	Removes the items in this set that are also included in another, specified set
discard()	Remove the specified item
intersection()	Returns a set, that is the intersection of two other sets
intersection_update()	Removes the items in this set that are not present in other, specified set(s)
isdisjoint()	Returns whether two sets have a intersection or not
issubset()	Returns whether another set contains this set or not
issuperset()	Returns whether this set contains another set or not
pop()	Removes an element from the set
remove()	Removes the specified element
symmetric_difference()	Returns a set with the symmetric differences of two sets
symmetric_difference_update()	inserts the symmetric differences from this set and another
union()	Return a set containing the union of sets
update()	Update the set with the union of this set and others

8. Tuples

What is a tuple?

- A tuple is a type of sequence that resembles a list, except that, unlike a list, a tuple is immutable.(It cannot be modified)
- You indicate a tuple in Python by enclosing its elements in parentheses `( )` instead of square brackets `[ ]`.

Example

```
fruits = ("apple","banana")
```

Benefits of tuple?

- Tuples are faster than lists
- Tuples can be used as keys in dictionaries, but list cant
- Protects data against accidental changes

9. Dictionaries

What is a dictionary?

- In Python, a dictionary associates a set of keys with data values.

Example

```
shop = {"pen":12,"paper":100}
```

- Here pen is key and 12 is value
- Similarly, paper is key and 100 is the value for paper

Adding keys and replacing values

```
info = {}
info["name"] = "Sandy"
info["occupation"] = "hacker"
print(info)
```

Output

```
{'name':Sandy,'occupation':'hacker'}
```

Accessing Values

```
>>> info["name"]
'Sandy'
```

Removing keys

```
info.pop("job")
```

DICTIONARY OPERATION	WHAT IT DOES
`len(d)`	Returns the number of entries in **d**.
`aDict[key]`	Used for inserting a new key, replacing a value, or obtaining a value at an existing key.
`d.get(key [, default])`	Returns the value if the key exists or returns the default if the key does not exist. Raises an error if the default is omitted and the key does not exist.
`d.pop(key [, default])`	Removes the key and returns the value if the key exists or returns the default if the key does not exist. Raises an error if the default is omitted and the key does not exist.
`list(d.keys())`	Returns a list of the keys.
`list(d.values())`	Returns a list of the values.
`list(d.items())`	Returns a list of tuples containing the keys and values for each entry.
`d.clear()`	Removes all the keys.
`for key in d:`	**key** is bound to each key in **d** in an unspecified order.

Python-Module-2-University-Questions-Part-A

1. Write Python code for the following statements

i)writes the text ”PROGRAMMING IN PYTHON” to a file with name code.txt

ii) then reads the text again and prints it to the screen

```
def write_and_read_text():
  """Writes text to a file and then reads it back, printing it to the
screen."""

  # Text to write
  text = "PROGRAMMING IN PYTHON"

  # Open the file in write mode ("w")
  with open("code.txt", "w") as file:
    file.write(text)

  # Open the file again in read mode ("r")
  with open("code.txt", "r") as file:
    # Read the entire content
    read_text = file.read()

  # Print the read text
  print(read_text)

# Call the function
write_and_read_text()
```

2. What are mutable and immutable properties in the case of Python datastructures?

In Python, data structures can be mutable (changeable) or immutable (unchangeable).

- **Mutable:** Lists, dictionaries, sets (can modify elements after creation).
- **Immutable:** Strings, tuples, numbers (content remains fixed).

3. Write the output of following python code :

S = “Computer”

print(S[::2])

print(S[::-1])

print(S[:])

```
>>> S="Computer"
>>> print(S[::2])
Cmue
>>> print(S[::-1])
retupmoC
>>> print(S[:])
Computer
```

*4. Write a recursive function in python to find GCD of two numbers.**

-

```
def gcd(a, b):
  """
  This function calculates the GCD (Greatest Common Divisor) of two non-
negative integers using recursion.

  Args:
      a: The first non-negative integer.
      b: The second non-negative integer.

  Returns:
      The GCD of a and b.
  """

  # Base case: If b is 0, then GCD is a
  if b == 0:
    return a

  # Recursive case: GCD(a, b) = GCD(b, a % b)
  else:
    return gcd(b, a % b)

# Example usage
result = gcd(12, 18)
print(f"The GCD of 12 and 18 is: {result}")
```

5. Differentiate between lists and tuples with the help of examples

- Lists and tuples are both fundamental data structures in Python used to store collections of elements, but they differ in their mutability:
- **Mutability:**
  - **Lists:** Mutable - elements can be added, removed, or changed after creation.
  - **Tuples:** Immutable - elements cannot be modified after creation.

  Example

```
# List - mutable (can be changed)
fruits = ["apple", "banana", "cherry"]
fruits.append("orange")  # Add element
fruits[1] = "mango"    # Modify element

# Tuple - immutable (cannot be changed)
colors = ("red", "green", "blue")
# colors[0] = "purple"  # This will cause an error
```

6. Write a Python program to print all palindromes in a line of text.

```
def is_palindrome(text):
  """
  Checks if a given text is a palindrome (reads the same backward as
forward).

  Args:
      text: The text to be checked.

  Returns:
      True if the text is a palindrome, False otherwise.
  """
  # Convert to lowercase and remove non-alphanumeric characters
```

```
  return text == text[::-1]

def find_palindromes(text):
  """
  Finds all palindromes in a line of text and prints them.

  Args:
      text: The line of text to search for palindromes.
  """
  words = text.split()
  for word in words:
    if is_palindrome(word):
      print(word)

# Example usage
text = "level madam, i am adam."
find_palindromes(text)
```

7. Illustrate format specifiers and escape sequences with examples.

Format Specifiers:

- Insert variables or expressions into strings at designated points.
- Common examples:
  - `{:d}` - Integers (decimal)
  - `{:f}` - Floating-point numbers
  - `{:s}` - Strings
  - `{:c}` - Single characters

```
name = "Alice"
age = 30
print(f"Hello, my name is {name} and I am {age} years old.")
```

Escape Sequences:

- Special sequences of characters that represent non-printable characters or modify how a string is displayed.
- Start with a backslash (`\`) followed by another character.
- Common examples:
  - `\n` - Newline character (inserts a line break)
  - `\t` - Horizontal tab
  - `\"` - Double quote (to print a double quote within a string)
  - `\'` - Single quote (to print a single quote within a string)

Example

```
print("This string has a \"double quote\" and a \'single quote\'.")
```

Python-Module-2-University-Question-Part-B

1. Write a Python program to implement Caesar cipher encryption and decryption on a string of lowercase letters. Take distance value and the string as input.

(Hint: Caesar cipher encryption strategy replaces each character in the plaintext with the character that occurs a given distance away in the sequence.
Encryption:Eg. input: 3, “invade”, Eg. output: “lqydgh”
Decryption: Eg. input: 3, “lqydgh”, Eg. output: “invade”)

```
# Encrypt
plainText = input("Enter word")
distance = int(input("Enter distance value"))

code = ""
for ch in plainText:
        ordvalue = ord(ch)
        cipherValue = ordvalue + distance
        if cipherValue > ord('z'):
                cipherValue = ord('a') + distance - (ord('z') - ordvalue +
```

```
1)
        code+=chr(cipherValue)
print(code)

#Decrypt
code = input("Enter text")
distance = int(input("Enter distance"))
plainText = ""

for ch in code:
        ordvalue = ord(ch)
        cipherValue = ordvalue - distance
        if cipherValue < ord('a'):
                cipherValue = ord('z') - (distance - ord('a')-ordvalue - 1)
        plainText += chr(cipherValue)
print(plainText)
```

2. Write a Python code segment that opens a file for input and prints the number of four-letter words in the file.

```
def count_four_letter_words(filename):
  """
  Counts the number of four-letter words in a given file.

  Args:
      filename: The name of the file to read.
  """
  count = 0
  try:
    with open(filename, 'r') as file:
      for line in file:
        words = line.split()
        for word in words:
          # Check if word has 4 letters and is alphabetic (only letters)
          if len(word) == 4 and word.isalpha():
            count += 1
  except FileNotFoundError:
    print(f"Error: File '{filename}' not found.")
```

```
# Example usage
filename = "your_file.txt"  # Replace with your actual filename
count_four_letter_words(filename)
```

3. Write a Python program to create a set of functions that compute the mean,median and mode of a set of numbers. Each function should expect a list of numbers as an argument and return a single number. Each function should return 0 if the list is empty. Include a main function that tests the three functions with a given list.

(Hint: Mean: Mean is the average value of a list of numbers

Median: If the number of values in a list is odd, the median of thelist is the value at the midpoint when the set of numbers is sorted; otherwise, themedian is the average of the two values surrounding the midpoint.

Mode: The mode of a list of values is the value that occurs most frequently)*

```
def calculate_mean(numbers):
  """
  Calculates the mean (average) of a list of numbers.

  Args:
      numbers: A list of numbers.

  Returns:
      The mean of the list, or 0 if the list is empty.
  """

  if not numbers:
    return 0

  return sum(numbers) / len(numbers)

def calculate_median(numbers):
  """
  Calculates the median of a list of numbers.
```

```
  Args:
      numbers: A list of numbers (assumed to be sortable).

  Returns:
      The median of the list, or 0 if the list is empty.
  """

  if not numbers:
    return 0

  sorted_numbers = sorted(numbers)  # Sort a copy to avoid modifying the
original list
  midpoint = len(sorted_numbers) // 2

  if len(sorted_numbers) % 2 == 0:
    # Even number of elements, calculate average of middle two
    return (sorted_numbers[midpoint - 1] + sorted_numbers[midpoint]) / 2
  else:
    # Odd number of elements, return middle element
    return sorted_numbers[midpoint]

def calculate_mode(numbers):
  """
  Calculates the mode (most frequent value) of a list of numbers.

  Args:
      numbers: A list of numbers.

  Returns:
      The mode of the list, or 0 if the list is empty or all elements are
unique.
  """

  if not numbers:
    return 0

  # Create a dictionary to store the count of each number
  counts = {}
  for number in numbers:
    if number in counts:
      # If the number exists in the dictionary, increment its count
```

```
            counts[number] += 1
        else:
            # If the number is new, add it to the dictionary with a count of 1
            counts[number] = 1

    # Find the highest count
    highest_count = 0
    for count in counts.values():
        if count > highest_count:
            highest_count = count

    # Check for multiple modes or no mode
    if highest_count == 1:
        return 0  # All elements are unique, no mode
    else:
        # Find any element with the highest count (there can be multiple modes)
        for number, count in counts.items():
            if count == highest_count:
                return number  # Return the first mode found
def main():
    """
    Tests the mean, median, and mode functions with a sample list.
    """

    numbers = [1, 2, 3, 4, 5, 2, 3, 3]  # Sample list
    print(f"Mean: {calculate_mean(numbers)}")
    print(f"Median: {calculate_median(numbers)}")
    print(f"Mode: {calculate_mode(numbers)}")

if __name__ == "__main__":
    main()
```

4. Write a Python program to check whether a list contains a sublist.

Eg. Input 1: my_list = [3,4,5,2,7,8] , sub_list = [2,7]

output 1: True

input 2: my_list = [3,4,5,2,7,8] , sub_list = [5,7]

output 2: False

```
def is_sublist(my_list, sub_list):
  """
  Checks if a sub_list is present within the my_list.

  Args:
      my_list: The main list to search.
      sub_list: The sublist to be found.

  Returns:
      True if the sub_list is found within my_list, False otherwise.
  """

  # Iterate through each element in my_list
  for i in range(len(my_list)):
    # Check if the current element matches the beginning of the sub_list
    if my_list[i] == sub_list[0]:
      # If it matches, compare the remaining elements of sub_list with a
slice of my_list
      if my_list[i:i + len(sub_list)] == sub_list:
        return True  # Sublist found, return True

  # If the loop completes without finding a match, return False
  return False

# Example usage (True)
my_list = [3, 4, 5, 2, 7, 8]
sub_list = [2, 7]
print(is_sublist(my_list, sub_list))  # Output: True

# Example usage (False)
my_list = [3, 4, 5, 2, 7, 8]
sub_list = [5, 7]
print(is_sublist(my_list, sub_list))  # Output: False
```

5. Assume that the variable data refers to the string "Python rules!". Use a string method to perform the following tasks:

a. Obtain a list of the words in the string.

b. Convert the string to uppercase.

c. Locate the position of the string "rules" .

d. Replace the exclamation point with a question mark.

```
data = "Python rules!"

# a. Obtain a list of the words in the string
word_list = data.split()
print(word_list)  # Output: ['Python', 'rules!']

# b. Convert the string to uppercase
uppercase_data = data.upper()
print(uppercase_data)  # Output: PYTHON RULES!

# c. Locate the position of the string "rules"
position = data.find("rules")
print(position)  # Output: 7

# d. Replace the exclamation point with a question mark
replaced_data = data.replace("!", "?")
print(replaced_data)  # Output: Python rules?
```

6. Use higher order python function filter to extract a list of positive numbers from a given list of numbers. You should use a lambda to create the auxiliary function

```
numbers = [1, -2, 3, -4, 5]

# Use filter with a lambda to check for positive numbers
positive_numbers = list(filter(lambda x: x > 0, numbers))
```

```
print(positive_numbers)  # Output: [1, 3, 5]
```

7. Assume that there is a text file named “numbers.txt”. Write a python program to find the median of list of numbers in the file without using standard function for median

```
def calculate_median(numbers):
  """
  Calculates the median of a list of numbers without using standard
functions.

  Args:
      numbers: A list of numbers.

  Returns:
      The median of the list.
  """

  sorted_numbers = sorted(numbers)  # Sort the list in ascending order
  length = len(sorted_numbers)

  if length % 2 == 0:
    # Even number of elements, calculate average of middle two
    midpoint = length // 2
    return (sorted_numbers[midpoint - 1] + sorted_numbers[midpoint]) / 2
  else:
    # Odd number of elements, return middle element
    midpoint = length // 2
    return sorted_numbers[midpoint]

def read_numbers_from_file(filename):
  """
  Reads a list of numbers from a text file.

  Args:
      filename: The name of the text file.
```

```
  Returns:
      A list of numbers from the file, or an empty list if the file cannot
be read.
  """

  numbers = []
  try:
    with open(filename, 'r') as file:
      for line in file:
        # Convert each line to a float (assuming numerical data)
        number = float(line.strip())
        numbers.append(number)
  except FileNotFoundError:
    print(f"Error: File '{filename}' not found.")

  return numbers

# Example usage
filename = "numbers.txt"
numbers = read_numbers_from_file(filename)

if numbers:
  median = calculate_median(numbers)
  print(f"Median of numbers in '{filename}': {median}")
else:
  print(f"Error: No numbers found in '{filename}'.")
```

8. Write a Python program to convert a decimal number to its binary equivalent.

```
def decimal_to_binary(decimal_number):
  """
  Converts a decimal number to its binary equivalent.

  Args:
      decimal_number: An integer representing the decimal number.

  Returns:
```

```
      The binary equivalent of the decimal number as a string,
      or "0" if the input is 0.
  """

  if decimal_number == 0:
    return "0"

  binary_string = ""
  while decimal_number > 0:
    remainder = decimal_number % 2  # Get the remainder after division by 2
(binary digit)
    binary_string = str(remainder) + binary_string  # Prepend the remainder
to the string
    decimal_number //= 2  # Divide the decimal number by 2 for the next
iteration

  return binary_string

# Example usage
decimal_number = 10
binary_equivalent = decimal_to_binary(decimal_number)
print(f"{decimal_number} in binary: {binary_equivalent}")  # Output: 10 in
binary: 1010
```

9. Write a Python program to read a text file and store the count of occurrences of each character in a dictionary

```
def count_char_occurrences(filename):
  """
  Counts the occurrences of each character in a text file and stores them in
a dictionary.

  Args:
      filename: The name of the text file to read.

  Returns:
      A dictionary where the keys are characters and the values are their
counts.
```

```
    """

    char_counts = {}
    with open(filename, 'r') as file:
        for line in file:
            for char in line:
                if char in char_counts:
                    char_counts[char] += 1
                else:
                    char_counts[char] = 1

    return char_counts

# Example usage (replace with your actual filename)
filename = "your_file.txt"
char_counts = count_char_occurrences(filename)

print("Character counts:")
for char, count in char_counts.items():
    print(f"{char}: {count}")
```

10. Write a Python code to create a function called list_of frequency that takes a string and prints the letters in non-increasing order of the frequency of their occurrences. Use dictionaries.

```
def list_of_frequency(text):
    """
    Prints the letters in a string from most frequent to least frequent.

    Args:
        text: The string to analyze.
    """

    # Create a dictionary to store how many times each letter appears
    letter_counts = {}
    for char in text:
```

```
    if char in letter_counts:
      # If the letter is already in the dictionary, add 1 to its count
      letter_counts[char] += 1
    else:
      # If the letter is new, add it to the dictionary with a count of 1
      letter_counts[char] = 1

  # Create a list to store (letter, count) pairs
  letter_and_counts = []
  for letter, count in letter_counts.items():
    # Add a tuple with the letter and its count to the list
    letter_and_counts.append((letter, count))

  # Sort the list by the count (second element in each tuple), highest first
  letter_and_counts.sort(key=lambda item: item[1], reverse=True)

  # Print the letters in order from most frequent to least frequent
  print("Letters from most frequent to least frequent:")
  for letter, _ in letter_and_counts:  # We only care about the letter here
(not the count)
    print(letter, end="")

# Example usage
text = "Mississippi"
list_of_frequency(text)
```

11. Write a Python program to read a list of numbers and sort the list in a non- decreasing order without using any built in functions. Separate function should be written to sort the list wherein the name of the list is passed as the parameter.

```
def sort_list(numbers):
  """
  Sorts a list of numbers in increasing order.

  Args:
      numbers: The list of numbers to sort.
```

```
  Returns:
      The sorted list.
  """

  # Loop for the number of passes (worst case)
  n = len(numbers)
  for i in range(n - 1):
    # Inner loop to compare adjacent elements
    for j in range(n - 1 - i):
      if numbers[j] > numbers[j + 1]:
        # Swap elements if they are in the wrong order
        numbers[j], numbers[j + 1] = numbers[j + 1], numbers[j]

  return numbers

def main():
  """
  Gets numbers from the user, sorts them, and prints the result.
  """

  numbers = []
  while True:
    # Get a single number as input (assumes valid input)
    num_str = input("Enter a number (or press Enter to finish): ")
    # Check if user pressed Enter only
    if not num_str:
      break
    numbers.append(float(num_str))

  if not numbers:
    print("No numbers entered.")
  else:
    # Sort the list using the sort_list function
    sorted_numbers = sort_list(numbers.copy())  # Avoid modifying the
original list
    print("Sorted list:", sorted_numbers)

if __name__ == "__main__":
  main()
```

12. Illustrate the following Set methods with an example.

i. intersection( ) ii. Union( ) iii. Issubset( ) iv. Difference( ) v. update( ) vi. discard()

1. intersection()

- Returns a new set with elements common to both sets.

```
set1 = {1, 2, 3, 4, 5}
set2 = {2, 4, 6, 8}

intersection = set1.intersection(set2)
print("Intersection:", intersection)  # Output: Intersection: {2, 4}
```

2. union()

- Returns a new set with elements from both sets (without duplicates).

```
set1 = {1, 2, 3, 4, 5}
set2 = {2, 4, 6, 8}

union = set1.union(set2)
print("Union:", union)  # Output: Union: {1, 2, 3, 4, 5, 6, 8}
```

3. issubset()

- Returns True if all elements in set1 are also in set2.

```
set1 = {1, 2, 3}
set2 = {1, 2, 3, 4, 5}

is_subset = set1.issubset(set2)
print("Is set1 subset of set2:", is_subset)  # Output: Is set1 subset of
set2: True
```

4. difference()

- Returns a new set with elements in set1 but not in set2.

```
set1 = {1, 2, 3, 4, 5}
set2 = {2, 4, 6, 8}

difference = set1.difference(set2)
print("Difference:", difference)  # Output: Difference: {1, 3, 5}
```

5. update()

```
set1 = {1, 2, 3}
set2 = {4, 5, 6}

set1.update(set2)
print("set1 after update:", set1)  # Output: set1 after update: {1, 2, 3, 4,
5, 6}
```

6. discard()

```
set1 = {1, 2, 3, 4}

set1.discard(2)
print("set1 after discard:", set1)  # Output: set1 after discard: {1, 3, 4}
```

13. Write a Python program to check the validity of a password given by the user.

The Password should satisfy the following criteria:

1. **Contains at least one letter between a and z**
2. **Contains at least one number between 0 and 9**
3. **Contains at least one letter between A and Z**
4. **Contains at least one special character from $, #, @**
5. **Minimum length of password: 6**

```
def is_valid_password(password):
  """
```

```
  Checks if a password meets the following criteria:

  - Contains at least one lowercase letter (a-z)
  - Contains at least one uppercase letter (A-Z)
  - Contains at least one digit (0-9)
  - Contains at least one special character from $, #, @
  - Minimum length of 6 characters

  Args:
      password: The password string to validate.

  Returns:
      True if the password is valid, False otherwise.
  """

  has_lowercase = False
  has_uppercase = False
  has_digit = False
  has_special_char = False

  # Check for each character type individually
  for char in password:
    if char.islower():
      has_lowercase = True
    elif char.isupper():
      has_uppercase = True
    elif char.isdigit():
      has_digit = True
    elif char in "$#@":
      has_special_char = True

  # Check length and all criteria are met
  return len(password) >= 6 and has_lowercase and has_uppercase and
has_digit and has_special_char

def main():
  """
  Prompts the user for a password and checks its validity.
  """

  password = input("Enter your password: ")
```

```
  if is_valid_password(password):
    print("Password is valid.")
  else:
    print("Password is invalid. Please ensure it meets the following
criteria:")
    print("- Contains at least one lowercase letter (a-z)")
    print("- Contains at least one uppercase letter (A-Z)")
    print("- Contains at least one digit (0-9)")
    print("- Contains at least one special character ($, #, @)")
    print("- Minimum length of 6 characters")

if __name__ == "__main__":
  main()
```

14.Create a dictionary of names and birthdays. Write a Python program that asks the user to enter a name, and the program display the birthday of that person

```
birthdays = {
  "Alice": "Jan 1, 2000",
  "Bob": "Feb 14, 1990",
  "Charlie": "March 3, 1985"
}

def find_birthday(name):
  """
  Finds the birthday of a person in the dictionary.

  Args:
      name: The name of the person to search for.

  Returns:
      The birthday of the person if found, otherwise None.
  """

  if name in birthdays:
    return birthdays[name]
```

```
  else:
    return None

while True:
  # Get user input for a name
  name = input("Enter a name (or 'quit' to exit): ")

  # Check if user wants to quit
  if name.lower() == "quit":
    break

  # Find the birthday using the function
  birthday = find_birthday(name)

  # Display the result
  if birthday:
    print(f"{name}'s birthday is {birthday}.")
  else:
    print(f"Sorry, no birthday information found for {name}.")

print("Thank you for using the birthday finder!")
```