Page No.
Date

Assignment 2

1. Write a syntax of nested if-else statement.

```
if condition 1:
    # code to execute if condition 1 is true
    if condition 2:
        # code to execute if condition 2 is true
    else:
        # code to execute if condition 2 is false
else:
    # code to execute if condition 1 is false
```

2. Write syntax for for loop and while loop.

for loop syntax:

```
for item in iterable:
    # code block to be executed for each item in the iterable
```

while loop syntax:

```
while condition:
    # code block to be executed as long as the condition is true.
```

Page No.
Date

3. What do you mean by nesting of loop

A nested loop is a loop inside another loop. Although all kinds of loops can be nested, the most common nested loop involves for loops. These loops are particularly useful when displaying multidimensional data.

4. Define range function in python programming language.

The range function in Python is a built in function used to generate a sequence of numbers within a specified range. It is commonly used in loops for iteration.

5. Write a python program to check entered number is even or odd.

```
num = int(input("Enter any number
    to test whether it is odd or even"))
if (num % 2) == 0:
    print ("The number is even")
else:
    print ("The number is odd")
```

6. What are different conditional statements in Python programming language with example

In python, conditional statements are used to execute different blocks of code based on whether a certain condition evaluates to true or false.
The main conditional statements in python are:

if statement:

```
x = 10
if x > 5:
    print("x is greater than 5")
```

if-else statement:

```
x = 3
if x > 5:
    print("x is greater than 5")
else:
    print("x is not greater than 5")
```

if-elif-else statement:

```
x = 7
if x > 10:
    print("x is greater than 10")
elif x > 5:
    print("x is greater than 5 but
          not greater than 10")
```

```
else:
    print ("x is not greater than 5")
```

Nested if statements:

```
x = 10
if x > 5:
    print ("x is greater than 5")
    if x > 10:
        print ("x is also greater than 10")
    else:
        print ("x is not greater than 10")
else:
    print ("x is not greater than 5")
```

8. What do you mean by Continue, break and pass statement in detail with example.

Break Statement

The break statement in Python is used to terminate the loop or statement in which it is present. After that, the control will pass to the statements that are present after the break statement, if available.

Example of break statement

```
s = 'geeksforgeeks'
for letter in s:
    print(letter)

    if letter == 'e' or letter == 's':
        break
print("out of for loop")
```

Continue statement

continue is also a loop control statement just like the break statement. continue statement forces to execute the next iteration of the loop while skipping the part initialized to continue.

Example of continue statement

```
for i in range(1, 11):
    if i == 6:
        continue
    else:
        print(i, end = " ")
```

Pass statement

The "pass" statement in Python is used when a statement is required syntactically but you do not want any command or code to execute. It is like null operation, as nothing will happen if it is executed.

Page No.
Date

9 Write a Python program to find the percentage of student on the basis of 5 subject marks.

```
# Input marks for 5 subjects
subject 1 = int(input("Enter marks
                for subject 1: "))
subject 2 = int(input("Enter marks
                for subject 2: "))
subject 3 = int(input("Enter marks
                for subject 3: "))
subject 4 = int(input("Enter marks
                for subject 4: "))
subject 5 = int(input("Enter marks
                for subject 5: "))

# calculate total
total = subject1 + subject2 + subject 3 +
        subject 4 + subject 5

# calculate percentage
percentage = (total / 500) * 100

# display the percentage

print("Your percentage is: " + str
    (percentage) + "%").
```

10. Write a python program to find factorial of a entered number

```
num = int(input(" Enter a number to print Factorial :"))
def fact (num):
    x = 1
    for y in range(1, num+1):
        x = x*y
        print(x)
fact (num)
```

Page No.
Date

Assignment 3

1. What is function?

Function is a block of code designed to perform a specific task. It can take inputs, perform operations or calculations, and return an output. Functions help organize code, make it more modular, reusable, and easier to understand

2. What are keyword arguments?

keyword arguments, also known as named arguments are a feature in many programming languages that allow you to specify the name of the arguments explicitly when calling a function.

3. What are Global scope of variables?

The global scope of variables refers to variables that are defined outside of any function, class, or block and are accessible from any part of the code. These variables have a global lifetime, meaning they exist for the duration of the program's execution.

4. What is Recursion?

Recursion is a programming technique where a function calls itself in order to solve a problem. Each recursive call should bring the problem closer to a base class which is a condition that stops the recursion.

5. What is System function in python?

In python, the 'system' function is provided by the 'os' module and is used to execute a command in the system's shell. It allows you to run shell commands from within a Python script.

6. Explain about scope of function and explain Local, Global and scope function under function in python

In python, the scope of a function refers to the region of the code where a particular variable is accessible. There are several scopes to be aware of: local, global and nonlocal.

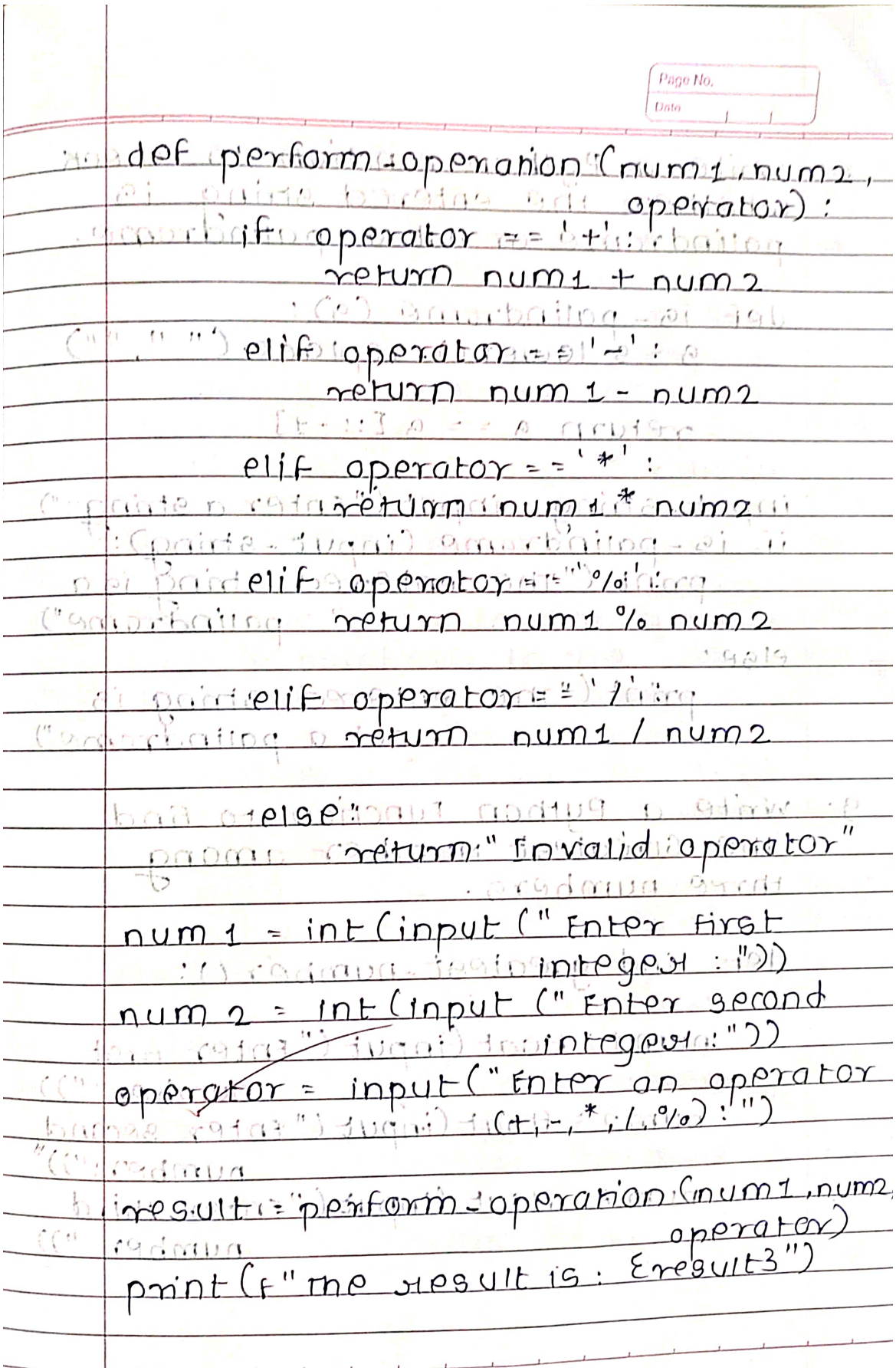
1. Local Scope
Variables defined inside a function have a local scope and are only accessible within the function.
Example:

```
def my-function():
    local-var=10
    print(local-var)
my-function()
```

2. Global Scope
Variables defined outside any function or in the global scope are accessible throughout the module, including inside functions.
Example:

```
global-var = 20
def my-function():
    print(global-var)
my-function()
print(global-var)
```

7. Write a python program to accept two integers and a character from (+, -, *, /, %). Write a function which will perform respective operation based on character and display output.



```
def perform_operation(num1, num2, operator):
    if operator == '+':
        return num1 + num2
    elif operator == '-':
        return num1 - num2
    elif operator == '*':
        return num1 * num2
    elif operator == '%':
        return num1 % num2
    elif operator == '/':
        return num1 / num2
    else:
        return "Invalid operator"

num1 = int(input("Enter first integer: "))
num2 = int(input("Enter second integer: "))
operator = input("Enter an operator (+,-,*,/,%): ")
result = perform_operation(num1, num2, operator)
print(f"The result is: {result}")
```

Page No.
Date

8. Write a Python function to check whether the entered string is palindrome or not a palindrome.

```
def is_palindrome(s):
    s = s.lower().replace(" ", "")
    return s == s[::-1]

input_string = input("Enter a string:")
if is_palindrome(input_string):
    print("The entered string is a palindrome")
else:
    print("The entered string is not a palindrome")
```

9. Write a Python function to find the Greatest Number among three numbers.

```
def find_greatest_number():
    num1 = float(input("Enter first number:"))
    num2 = float(input("Enter second number:"))
    num3 = float(input("Enter third number:"))
```

```
if (num1 >= num2) and (num1 >= num3):
    largest = num1
elif (num2 >= num1) and (num2 >= num3):
    largest = num2
else:
    largest = num3
print("The largest number is:
      &largest3")
find_greatest_number()
```

10. Python program to display the first 'n' numbers in the fibonacci sequence using Recursion:

```
def fibonacci(n):
    if n <= 1:
        return n
    else:
        return fibonacci(n-1)+fibonacci(n-2)
n = int(input("Enter the no of terms:"))
if n <= 0:
    print("Please enter +'ve integer")
else:
    print("Fibonacci Sequence:")

for i in range(n):
    print(fibonacci(i))
```

Page No.
Date

Assignment 4:

1. How do we find a substring in Python programming?

In python, you can find a substring within a string using the 'find()' method or the 'index()' method

2. Explain the differences between upper() and isupper() string functions

1. 'upper()' function
   - This function is used to convert all lowercase letters in string to uppercase
   - It returns a new string with all uppercase letters.
2. 'isupper()' function
   - This function checks if all characters in a string are uppercase.
   - It returns a boolean value ('True' or 'False').

3. What is a list? Why do we use a list?

A list in Python is a data structure that can hold a collection of elements. Used to store multiple items in a single variable. Lists are ordered, mutable (changeable) and allow duplicates. elements.

why do we use a list?

1. Grouping data
2. Sequential Access
3. Mutability
4. Dynamic size
5. versatility.

4. What is a tuple? How is a tuple different than a list?

A tuple is similar to a list in that it is also a data structure used to store a collection of elements. Tuples are immutable and created using parentheses, whereas lists are mutable and created using square brackets. Tuples are used when you need to store fixed collection of elements, while lists are used for dynamic collections

Page No.
Date

5. What is a dictionary in Python programming?

A dictionary is a data structure that stores a collection of key-value pairs. It is a versatile and powerful data type used for mapping keys to corresponding values.
Dictionaries are unordered, mutable, and allow for duplicate values (though not duplicate keys).

6. Write a python program to find if the reverse of entered string.

```
def check_palindrome():
    user_input = input("Enter a string: ")
    reversed_input = user_input[::-1]
    if user_input == reversed_input:
        print("The entered string is a palindrome")
    else:
        print("The entered string is not a palindrome")
check_palindrome()
```

Page No.
Date

7. Write a Python program to convert all the lower case vowels to upper case in the string accepted from the user

```
def convert_vowels_to_uppercase(input_string):
    vowels = 'aeiou'
    output_string = ''

    for char in input_string:
        if char.lower() in vowels:
            output_string += char.upper()
        else:
            output_string += char

    return output_string

user_input = input("Enter a string: ")
result = convert_vowels_to_uppercase(user_input)
print("Modified string:", result)
```

8. Write a Python program to accept marks of 10 students and display them in a list.

```
marks = []
for i in range(10):
    mark = float(input("Enter marks of student %i+1:"))
```

```
marks.append(mark)
print("Marks of 10 students:")
print(marks)
```

9. Write a Python program to accept a decimal number, create a function to convert this decimal number to binary and display the output.

```
def decimal_to_binary(decimal):
    return bin(decimal).replace("0b","")

decimal_number = int(input("Enter a decimal number:"))
binary_number = decimal_to_binary(decimal_number)
print("The binary representation of {decimal_number} is: {binary_number}")
```

10. Python program to accept a string from user and store it as a list.

```
user_input = input("Enter a string:")
string_list = list(user_input)
print("List from the input string:", string_list)
```

Page No.
Date

Assignment 5

1. What is Exception ?

Exception is an event that occurs during the execution of a program, disrupting its normal flow. Exceptions are typically errors such as trying to divide by zero or accessing a non-existent file.

result = 10/0
except ZeroDivisionError:
print("cannot divide by zero!")

2. List the different keywords to handle exception.

1. try: used to wrap the code that may raise an exception.
2. except: used to catch and handle exceptions that occur in try block
3. else: executed if no exceptions occur in try block
4. finally: executed regardless of whether an exception occurs typically used for cleanup actions

Page No.
Date

3. How many except statements can a try-except block have?

A try block in Python can have multiple except statements to handle different types of exceptions. There is no fixed limit to the number of except statements, you can include as many as needed to catch and handle different exceptions separately. This allows for more granular and specific error handling

4. When will the "else" part of try-except-else be executed?

The else part of a try-except-else block in python is executed only if no exceptions are raised in the try block. If an exception occurs, the else block is skipped, and the corresponding except block is executed instead.

5. When is the finally block executed?

The finally block in Python is executed always, regardless of whether an exception was raised or not in the try block. This ensures that any necessary cleanup actions or resources releases are performed.

Page No.
Date

6. Why use Exceptions in Python? Explain in detail.

Using exceptions in Python is crucial for several reasons, all contributing to robust and maintainable code.

Why exceptions are used:

1. Error Handling and Seperation of concerns:
   - Explicit Handling :- Exceptions allow for explicit handling of errors, making it clear where potential problems might occur and how they are addressed.
   - Seperation of Concerns - By isolating error handling code from the main program logic, developers can focus on the core functionality.
2. Robustness and Reliability:
   - Graceful Degradation: Using exceptions ensures that programs can handle unexpected situations gracefully without crashing.
   - Resource Management: The finally block ensures that resources (like files or network connections) are released properly regardless of whether an error occurred.

Page No.
Date

3. Improved debugging
• Stack trace : provided when exception occurs, detailed report of the error, including where it happened & state of program that time.

4. Code clarity and maintenance
• Readability : code that uses exceptions for error handling is often cleaner and more readable
• Maintainability : Adding new exception types and handlers is often easier than refactoring extensive conditional error checks.

5. Flow control
• Breaking out of Deep Logic : Exception provide a way to exit from deeply nested structures without having to pass error codes back through multiple layers.

6. Custom Exceptions :
• Tailored Error messages : Developers can define custom exceptions types for their specific application needs.
• Enhanced Handling : custom exceptions can carry additional information, making it easier to handle complex error scenarios in a structured way.

Page No.
Date

7. Explain catching in python with example

Catching exceptions in Python is a way to handle errors gracefully during the execution of a program. When an error occurs, instead of the program crashing, you can catch the exception and decide what to do next, such as logging the error, retrying the operation, or providing a user-friendly message.

Example:

```
try:
    result = 10/0
except ZeroDivisionError:
    print("You can't divide by zero!")
else:
    print("The result is:", result)
finally:
    print("Execution complete")
```

- Try: This block contains code that may raise an exception.
- Except: This block catches the exception and allows you to handle it.
- Else: This block executes if no exception were raised in the try block.
- Finally: This block contains code that will run no matter what whether an exception is raised or not.

Page No.
Date

8. Explain regular expression in details in python

Regular expressions (regex) are sequences of characters that define search patterns. They are used for matching, searching and manipulating strings based on specific patterns. Python's built-in 're' module provides support for working with regular expressions.

Common Regex Functions in Python:

1. re.match()
- Tries to match a pattern at the beginning of a string.
- Returns a match object if successful

Example:

```
import re
pattern = r'^Hello'
result = re.match(pattern, "Hello world!")
print(result)
```

2. re.search()
- Searches the entire string for a pattern
- Returns a match object if the pattern is found

Example:

```
import re
pattern = r"world"
result = re.search(pattern, "Hello world!")
print(result)
```

3. re.findall()
- Returns a list of all non overlapping matches of a pattern in the string.

Example:
```
import re
pattern = r'\d+'
result = re.findall(pattern, 'there are 123 apples and 456 oranges')
print(result)
```

4. re.finditer()
- Returns an iterator yielding match objects for all non overlapping matches

Example:
```
import re
pattern = re.finditer(pattern, 'there are 123 apples and 456 oranges')
for match in results:
    print(match.group())
```

5. re.sub()
- Replaces all occurrences of a pattern with a replacement string.

Example
```
import re
pattern = r'apples'
replacement = 'bananas'
result = re.sub(pattern, replacement, 'I like apples')
print(result)
```

Page No.
Date

9. Write a function that accepts a string and searches it for a valid phone number

```
import re

def find-phone-number (input-string):
    phone-pattern = re.compile (r'(\+?
        \d{1,3}?[-.\s]?(\(?\d{1,43
        ?)[-.\s]?)?(\d{1,43[-.\s]?
        \d{1,43[-.\s]?\d{1,9}')
    matches = phone-pattern.findall
        (input-string)
    phone-numbers = ["".join(match)
        for match in matches]
    return phone-numbers

input-string = "contact us at +1-800-555
    -1234 or (123) 456-7890.
    Alternatively, call
    987-654-3210"
phone-numbers = find-phone-number
    (input-string)
print (phone-numbers)
```

Page No.
Date

10. Explain advantages and disadvantages of Regular Expression in Python

Regular expressions (regex) are powerful tools for pattern matching and text processing in Python, but they come with their own set of advantages and disadvantages.

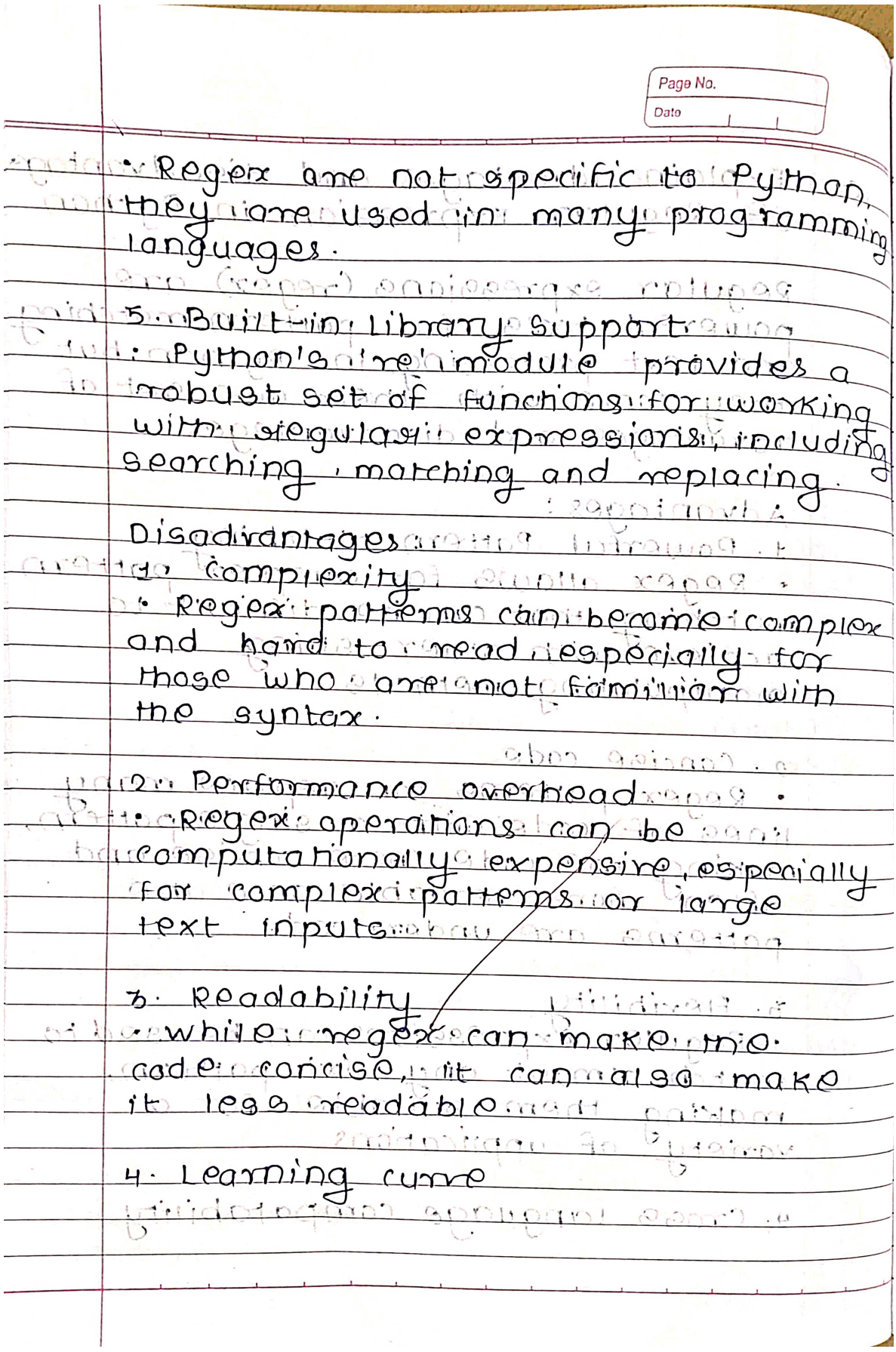
Advantages:

1. Powerful Pattern Matching
- Regex allows for complex pattern matching that can be difficult to achieve with other string manipulating methods.

2. Concise code
- Regex patterns can replace many lines of code with a single pattern, making the code more concise and often more readable once the patterns are understood.

3. Flexibility
- Regular expressions can be used to match almost any text pattern, making them suitable for a variety of applications.

4. Cross Language compatability



- Regex are not specific to Python. They are used in many programming languages.

5. Built-in Library Support
- Python's `re` module provides a robust set of functions for working with regular expressions, including searching, matching and replacing.

Disadvantages:

1. Complexity
- Regex patterns can become complex and hard to read, especially for those who are not familiar with the syntax.

2. Performance overhead
- Regex operations can be computationally expensive, especially for complex patterns or large text inputs.

3. Readability
- While regex can make the code concise, it can also make it less readable.

4. Learning curve

Page No.
Date

- There is a steep learning curve associated with regular expressions. Understanding and mastering the syntax requires practice and time.

5. Limited by Syntax
- While powerful, regex is not suitable for all parsing tasks. It can struggle with nested structures or patterns that require recursive matching, where more advanced parsing techniques might be necessary.