

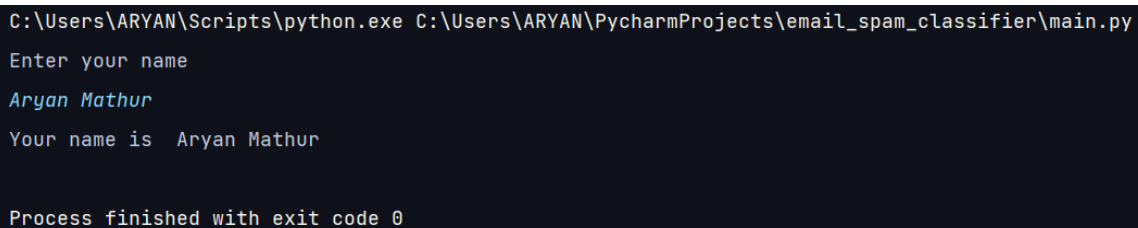
EXPERIMENT NO 1

Aim: Basic data types and operators: Create a program that prompts the user for their name and age and prints a personalized message.

Source Code:

```
name = input("Enter your name\n")  
  
print("Your name is ",name)
```

Output:



```
C:\Users\ARYAN\Scripts\python.exe C:\Users\ARYAN\PycharmProjects\email_spam_classifier\main.py  
Enter your name  
Aryan Mathur  
Your name is Aryan Mathur  
  
Process finished with exit code 0
```

Viva Questions:

Question: What is the purpose of the print() function in Python?

Answer: The print() function is used to display output on the console. It takes one or more arguments and prints them to the standard output (usually the console window)

Question: How can you print multiple values on the same line using the print() function?

Answer: You can print multiple values on the same line by separating them with commas within the print() function. For example, print("Hello", "world!") will output "Hello world!".

Question: What is the significance of the end parameter in the print() function?

Answer: The end parameter allows you to specify the character(s) that should be printed at the end of the output. By default, end is set to "\n" (newline), but you can change it to any other character(s) as needed.

Question: How can you prevent the print() function from automatically adding a newline character at the end of the output?

Answer: You can prevent print() from adding a newline character by setting the end parameter to an empty string "". For example, print("Hello", end="") will print "Hello" without a newline.

Question: How do you format output using the print() function in Python?

Answer: You can format output using f-strings or the format() method. For example, name = "Alice"; age = 30; print(f"My name is {name} and I am {age} years old.") will output "My name is Alice and I am 30 years old."

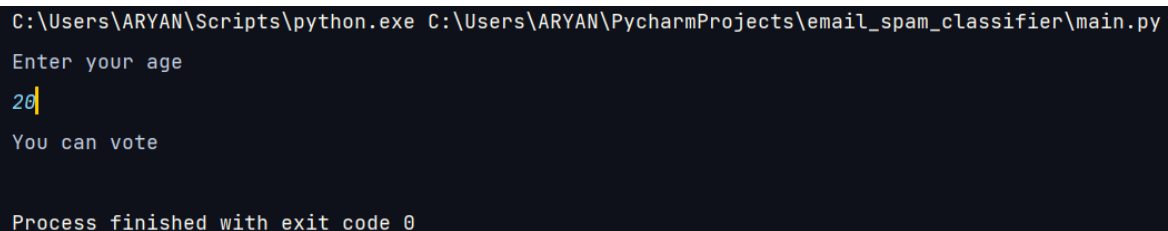
EXPERIMENT NO 2

Aim: Conditional statements: Create a program that prompts the user for their age and tells them if they can vote in the next election.

Source Code:

```
age = int(input("Enter your age\n"))  
if age >=18 :  
    print("You can vote")  
else :  
    print("You cannot vote")
```

Output:



```
C:\Users\ARYAN\Scripts\python.exe C:\Users\ARYAN\PycharmProjects\email_spam_classifier\main.py  
Enter your age  
20  
You can vote  
  
Process finished with exit code 0
```

Viva Questions:

Question: What is the purpose of the if-else statement in Python?

Answer: The if-else statement allows for conditional execution of code. It evaluates a condition and executes one set of statements if the condition is true, and another set if the condition is false.

Question: How do you write a basic if-else statement in Python?

Answer: A basic if-else statement in Python is written as follows:

"If a certain condition is true, perform one action; otherwise, perform another action."

Question: Can you have multiple conditions in an if-else statement?

Answer: Yes, you can have multiple conditions using the elif (else if) statement. It allows you to specify additional conditions to be checked if the initial if condition is false.

Question: How can you nest if-else statements in Python?

Answer: You can nest if-else statements by placing one if-else statement inside another. This allows for more complex conditional logic.

Question: What is the ternary conditional operator in Python?

Answer: The ternary conditional operator is a concise way to write an if-else statement in a single line. It has the syntax `value_if_true if condition else value_if_false`.

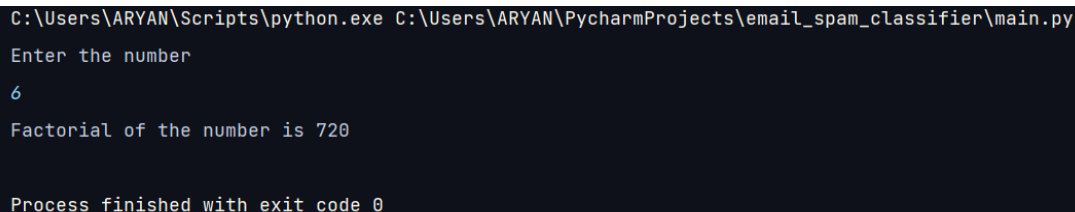
EXPERIMENT NO 3

Aim: Loops: Create a program that calculates the factorial of a number entered by the user using a loop

Source Code:

```
n = int(input("Enter the number\n"))
ans=1
while n:
    ans*=n
    n=n-1
print("Factorial of the number is",ans)
```

Output:



```
C:\Users\ARYAN\Scripts\python.exe C:\Users\ARYAN\PycharmProjects\email_spam_classifier\main.py
Enter the number
6
Factorial of the number is 720

Process finished with exit code 0
```

Viva Questions:

Question: What are loops used for in Python?

Answer: Loops are used to execute a block of code repeatedly as long as a specified condition is true, or for iterating over a sequence (such as lists, tuples, dictionaries, etc.).

Question: What is the difference between a for loop and a while loop?

Answer: A for loop is used for iterating over a sequence or iterable object, such as lists or strings, while a while loop is used to repeatedly execute a block of code as long as a specified condition is true.

Question: How do you exit a loop prematurely in Python?

Answer: You can exit a loop prematurely using the break statement. When the break statement is encountered inside a loop, the loop is terminated immediately, and the program control moves to the next statement after the loop.

Question: What is an infinite loop in Python?

Answer: An infinite loop is a loop that continues to execute indefinitely because its condition never evaluates to false. This can happen accidentally if the loop's condition is not properly defined or if there is no condition at all.

Question: How do you skip the current iteration of a loop in Python?

Answer: You can skip the current iteration of a loop and move to the next iteration using the continue statement. When the continue statement is encountered inside a loop, the current iteration is terminated, and the loop proceeds to the next iteration.

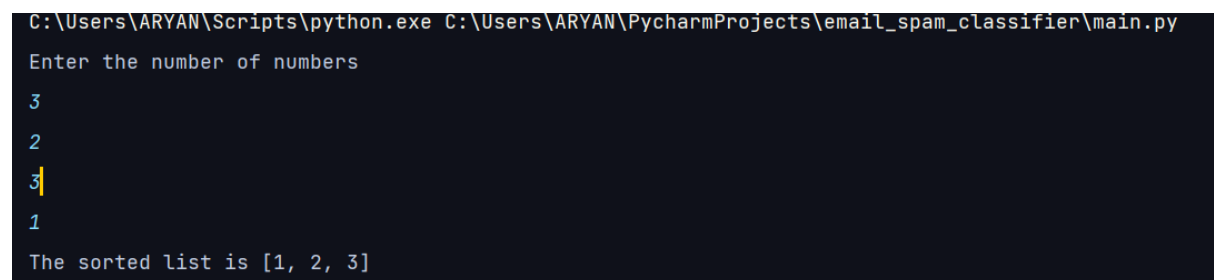
EXPERIMENT NO 4

Aim: Lists and arrays: Create a program that prompts the user for a list of numbers and then sorts them in ascending order.

Source Code:

```
n=int(input("Enter the number of numbers\n"))
array = []
while n :
    ele=int(input())
    array.append(ele)
    n=n-1
array.sort()
print("The sorted list is",array)
```

Output:



```
C:\Users\ARYAN\Scripts\python.exe C:\Users\ARYAN\PycharmProjects\email_spam_classifier\main.py
Enter the number of numbers
3
2
3
1
The sorted list is [1, 2, 3]
```

Viva Questions:

Question: What is a list in Python?

Answer: A list in Python is a collection of items or elements, ordered and mutable, which means you can change the content of a list after it has been created. Lists are represented by square brackets [].

Question: How do you access elements in a list?

Answer: You can access elements in a list using indexing. Indexing starts from 0 for the first element, 1 for the second element, and so on. Negative indexing is also supported, where -1 refers to the last element, -2 refers to the second last, and so forth.

Question: What is the difference between lists and arrays in Python?

Answer: In Python, lists and arrays are similar in that they both store collections of data. However, arrays are a part of the numpy module and are used for numerical computing, offering more functionality and efficiency compared to lists for mathematical operations.

Question: How do you append elements to a list in Python?

Answer: You can append elements to a list using the `append()` method. This method adds an item to the end of the list. Alternatively, you can use the `insert()` method to insert an element at a specified position.

Question: What are some common operations performed on lists in Python?

Answer: Some common operations on lists include adding elements, removing elements, accessing elements using indexing, slicing, iterating over elements using loops, checking if an element exists in a list, and sorting the elements. Lists offer a wide range of functionalities for manipulating data.

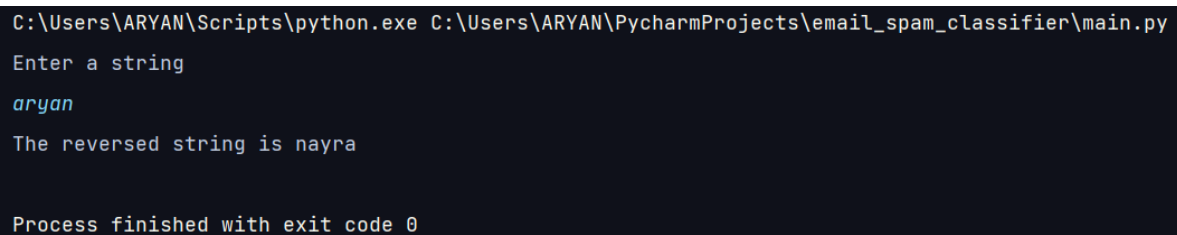
EXPERIMENT NO 5

Aim: Strings and string manipulation: Create a program that prompts the user for a string and then prints out the string reversed

Source Code:

```
str = input("Enter a string\n")
rev = str[::-1]
print("The reversed string is",rev)
```

Output:



```
C:\Users\ARYAN\Scripts\python.exe C:\Users\ARYAN\PycharmProjects\email_spam_classifier\main.py
Enter a string
aryan
The reversed string is nayra
Process finished with exit code 0
```

Viva Questions:

Question: What is string manipulation in Python?

Answer: String manipulation refers to the process of modifying or manipulating strings in various ways, such as concatenating strings, splitting strings, replacing characters or substrings, converting cases, and more.

Question: How do you concatenate strings in Python?

Answer: You can concatenate strings in Python using the + operator. For example, "Hello" + " " + "World" will result in the string "Hello World".

Question: How do you split a string into substrings in Python?

Answer: You can split a string into substrings using the split() method. This method splits a string based on a specified separator and returns a list of substrings.

Question: How do you replace characters or substrings in a string in Python?

Answer: You can replace characters or substrings in a string using the replace() method. This method replaces all occurrences of a specified substring with another substring.

Question: How do you convert the case of a string in Python?

Answer: You can convert the case of a string in Python using methods such as upper() to convert to uppercase, lower() to convert to lowercase, capitalize() to capitalize the first character, and title() to capitalize the first character of each word.

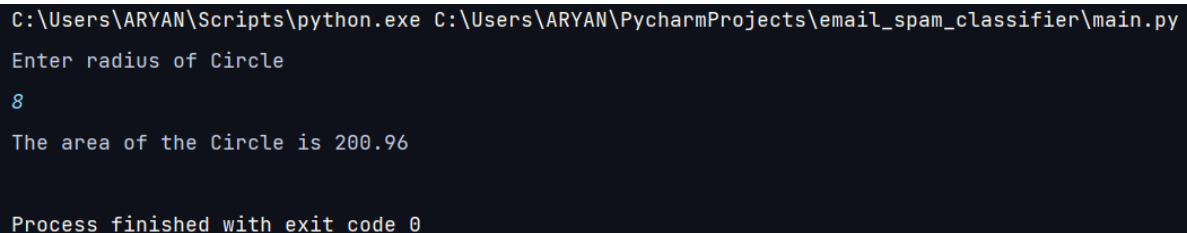
EXPERIMENT NO 6

Aim: Functions: Create a program that defines a function to calculate the area of a circle based on the radius entered by the user

Source Code:

```
def area(r):  
    return 3.14*r*r  
  
r = int(input("Enter radius of Circle\n"))  
ans = area(r)  
print("The area of the Circle is",ans)
```

Output:



```
C:\Users\ARYAN\Scripts\python.exe C:\Users\ARYAN\PycharmProjects\email_spam_classifier\main.py  
Enter radius of Circle  
8  
The area of the Circle is 200.96  
  
Process finished with exit code 0
```

Viva Questions:

Question: What is a function in Python?

Answer: A function in Python is a block of reusable code that performs a specific task. It takes inputs, called parameters or arguments, and returns outputs. Functions help organize code, improve reusability, and facilitate modular programming.

Question: How do you define a function in Python?

Answer: You define a function in Python using the `def` keyword followed by the function name and parameters enclosed in parentheses. The function body is indented below the `def` statement.

Question: What is the difference between parameters and arguments in Python functions?

Answer: Parameters are variables listed in the function definition, while arguments are the values passed to the function when it is called. Parameters receive the values of arguments, allowing functions to work with different inputs.

Question: How do you call a function in Python?

Answer: You call a function in Python by using its name followed by parentheses () containing the arguments, if any, to be passed to the function. For example, `my_function(arg1, arg2)`.

Question: Can a Python function return multiple values?

Answer: Yes, a Python function can return multiple values by returning them as a tuple. You can then unpack the tuple when calling the function to access individual values. For example, `return value1, value2`.

EXPERIMENT NO 7

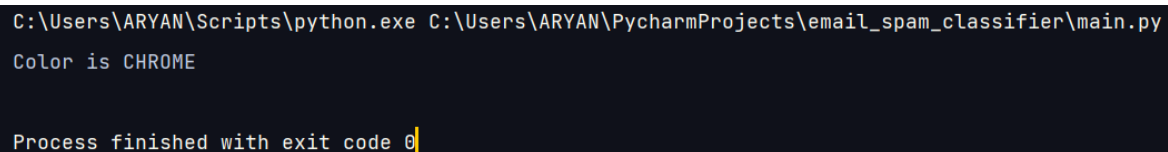
Aim: Classes and objects: Create a program that defines a class to represent a car and then creates an object of that class with specific attributes

Source Code:

```
class Car:
    def __init__(self,company, color):
        self.color=color
        self.company=company
    def getColor(self):
        return self.color
```

```
c = Car("Audi","CHROME")
color = c.getColor()
print("Color is",color)
```

Output:



```
C:\Users\ARYAN\Scripts\python.exe C:\Users\ARYAN\PycharmProjects\email_spam_classifier\main.py
Color is CHROME

Process finished with exit code 0
```

Viva Questions:

Question: What is the difference between a class and an object in Python?

Answer: A class is a blueprint or template for creating objects, while an object is an instance of a class. Classes define the properties and behaviors of objects, whereas objects are specific instances of those classes with unique data.

Question: How do you achieve encapsulation in Python classes?

Answer: Encapsulation is achieved in Python classes by using private attributes and methods, denoted by a single underscore `_` or double underscore `__` prefix. These attributes and methods are only accessible within the class itself, providing data protection and hiding implementation details.

Question: What is inheritance and how does it work in Python classes?

Answer: Inheritance is a mechanism in which a new class (subclass) inherits properties and behaviors from an existing class (superclass). The subclass can extend or override the functionality of the superclass while inheriting its attributes and methods, promoting code reuse and modularity.

Question: Can you explain the concept of polymorphism in Python?

Answer: Polymorphism allows objects of different classes to be treated as objects of a common superclass. It enables different classes to implement the same interface or method in different ways. Polymorphism in Python is achieved through method overriding and method overloading.

Question: How do you implement composition in Python classes?

Answer: Composition is a design principle where objects contain other objects as part of their state. In Python, composition is implemented by creating objects of other classes within a class and using them to provide functionality. This promotes code reuse and modularity without relying on inheritance.

EXPERIMENT NO 8

Aim: File input/output: Create a program that reads data from a file and writes it to another file in a different format

Source Code:

```
import csv
import json

def convert_csv_to_json(input_file, output_file):
    with open(input_file, 'r') as csv_file:
        csv_reader = csv.DictReader(csv_file)
        data = []
        for row in csv_reader:
            data.append(row)
    with open(output_file, 'w') as json_file:
        json.dump(data, json_file, indent=4)

input_file_path = 'spam.csv'
output_file_path = 'output.json'

convert_csv_to_json(input_file_path, output_file_path)

print("Conversion complete. Data has been written to", output_file_path)
```

Output:



```
{
  "v1": "ham",
  "v2": "Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got
  "": ""
},
{
  "v1": "ham",
  "v2": "Ok lar... Joking wif u oni...",
  "": ""
},
{
  "v1": "spam",
  "v2": "Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive ent
  "": ""
},
```

```
output.json x plot.py x spam.csv x
v1,v2,,,
ham,"Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...
ham,Ok lar... Joking wif u oni,,,,
spam,Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question
ham,U dun say so early hor... U c already then say,,,,
ham,"Nah I don't think he goes to usf, he lives around here though",,,
spam,"FreeMsg Hey there darling it's been 3 week's now and no word back! I'd like some fun you up for it still? Tb
ham,Even my brother is not like to speak with me. They treat me like aids patent.,,,
ham,As per your request 'Melle Melle (Oru Minnaminunginte Nurungu Vettam)' has been set as your callertune for all
spam,WINNER!! As a valued network customer you have been selected to receive a 900 prize reward! To claim call 090
spam,Had your mobile 11 months or more? U R entitled to Update to the latest colour mobiles with camera for Free!
ham,"I'm gonna be home soon and i don't want to talk about this stuff anymore tonight, k? I've cried enough today."
spam,"SIX chances to win CASH! From 100 to 20,000 pounds txt> CSH11 and send to 87575. Cost 150p/day, 6days, 16+ T
spam,"URGENT! You have won a 1 week FREE membership in our 100,000 Prize Jackpot! Txt the word: CLAIM to No: 8101
ham,I've been searching for the right words to thank you for this breather. I promise i wont take your help for gra
```

```
C:\Users\ARYAN\Scripts\python.exe C:\Users\ARYAN\PycharmProjects\email_spam_classifier\main.py
Conversion complete. Data has been written to output.json

Process finished with exit code 0
```

Viva Questions:

Question: What is the difference between reading a file in text mode and binary mode in Python?

Answer: Text mode is used for reading and writing text files, where data is encoded as text using a specified encoding (e.g., UTF-8). Binary mode, on the other hand, is used for reading and writing binary files, such as images or executables, where data is represented as raw bytes.

Question: How do you handle large files efficiently in Python?

Answer: You can handle large files efficiently in Python by using techniques like buffered reading and writing, reading and writing files in chunks, and using memory-mapped files. These techniques help reduce memory usage and improve performance when working with large files.

Question: What are context managers in Python file I/O, and how do you use them?

Answer: Context managers are objects that define `__enter__()` and `__exit__()` methods, allowing them to be used with the `with` statement. They provide resource management and ensure that resources are properly acquired and released, such as file handles in file I/O operations.

Question: How do you handle errors and exceptions in file I/O operations in Python?

Answer: You can handle errors and exceptions in file I/O operations by using try and except blocks to catch and handle specific exceptions, such as FileNotFoundError, PermissionError, or IOError. Additionally, you can use the finally block to perform cleanup operations.

Question: What are some best practices for working with files in Python?

Answer: Some best practices for working with files in Python include using context managers (with statement) to ensure proper resource management, handling exceptions gracefully, closing files explicitly using the close() method or by using the with statement, and using file modes (r, w, a, b, +) appropriately based on the intended operation.

EXPERIMENT NO 9

Aim: Regular expressions: Create a program that uses regular expressions to find all instances of a specific pattern in a text file

Source Code:

```
import re

def find_pattern_in_file(pattern, file_path):

    with open(file_path, 'r') as file:

        content = file.read()

        matches = re.findall(pattern, content)

        return matches

pattern_to_find = r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b'

file_path = 'emails.txt'

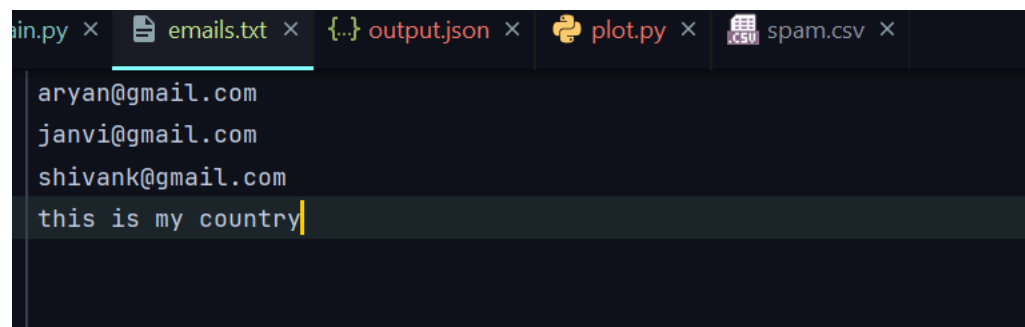
matches = find_pattern_in_file(pattern_to_find, file_path)

print("Matches found:")

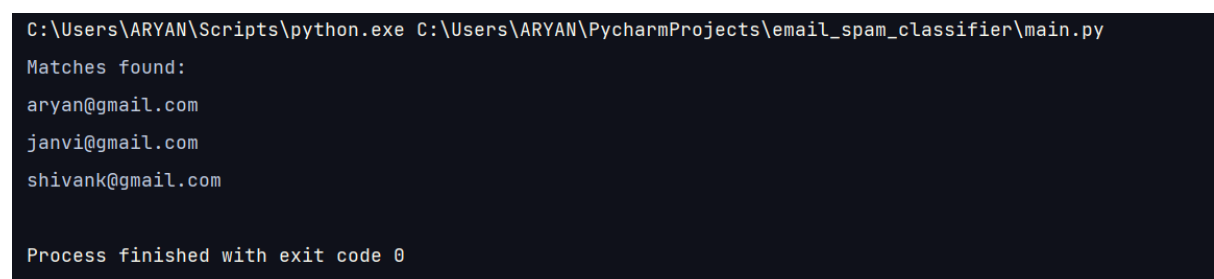
for match in matches:

    print(match)
```

Output:



```
main.py × emails.txt × {...} output.json × plot.py × spam.csv ×
aryan@gmail.com
janvi@gmail.com
shivank@gmail.com
this is my country
```



```
C:\Users\ARYAN\Scripts\python.exe C:\Users\ARYAN\PycharmProjects\email_spam_classifier\main.py
Matches found:
aryan@gmail.com
janvi@gmail.com
shivank@gmail.com

Process finished with exit code 0
```

Viva Questions:

Question: What is a regular expression and how is it used in Python?

Answer: A regular expression, or regex, is a sequence of characters that defines a search pattern. In Python, the re module provides support for working with regular expressions. Regular expressions are used for searching, matching, and manipulating text based on patterns.

Question: What are some common metacharacters used in regular expressions and what do they represent?

Answer: Some common metacharacters used in regular expressions include . (dot), which matches any single character except newline, * (asterisk), which matches zero or more occurrences of the preceding character, + (plus), which matches one or more occurrences of the preceding character, ^ (caret), which matches the start of the string, and \$ (dollar), which matches the end of the string.

Question: What is the difference between greedy and non-greedy matching in regular expressions?

Answer: Greedy matching in regular expressions attempts to match as much of the string as possible, while non-greedy or lazy matching attempts to match as little as possible. Greedy matching is denoted by * and + quantifiers, while non-greedy matching is denoted by adding a ? after the quantifier.

Question: How do you capture groups in regular expressions and what are they used for?

Answer: Capture groups in regular expressions are used to extract and isolate specific parts of a matched string. They are denoted by enclosing a part of the pattern in parentheses (). Captured groups can be referenced later in the regex or used to extract substrings from the matched text.

Question: What are lookahead and lookbehind assertions in regular expressions?

Answer: Lookahead and lookbehind assertions in regular expressions are used to assert whether a certain pattern is present or absent ahead of or behind the current position in the string, without consuming characters in the match. Lookahead assertions are denoted by (?=pattern) or (!pattern), while lookbehind assertions are denoted by (?<=pattern) or (?<!pattern).

EXPERIMENT NO 10

Aim: Exception handling: Create a program that prompts the user for two numbers and then divides them, handling any exceptions that may arise

Source Code:

```
def divide_numbers():
    while True:
        try:
            num1 = float(input("Enter the first number: "))
            break
        except ValueError:
            print("Invalid input. Please enter a valid number.")
    while True:
        try:
            num2 = float(input("Enter the second number: "))
            break
        except ValueError:
            print("Invalid input. Please enter a valid number.")
    try:
        result = num1 / num2
        print("The result of dividing", num1, "by", num2, "is:", result)
    except ZeroDivisionError:
        print("Error: Division by zero. Please enter a non-zero divisor.")

divide_numbers()
```

Output:

```
C:\Users\ARYAN\Scripts\python.exe C:\Users\ARYAN\PycharmProjects\email_spam_classifier\main.py
Enter the first number: 6
Enter the second number: 0
Error: Division by zero. Please enter a non-zero divisor.

Process finished with exit code 0
```

Viva Questions:

Question: What is the purpose of exception handling in Python?

Answer: Exception handling in Python is used to gracefully manage errors and exceptions that occur during program execution. It allows you to detect and handle exceptional conditions, preventing your program from crashing and providing a way to recover from errors.

Question: What are some common built-in exceptions in Python and when do they occur?

Answer: Some common built-in exceptions in Python include `SyntaxError`, which occurs when there is a syntax error in the code, `TypeError`, which occurs when an operation or function is applied to an object of inappropriate type, and `ZeroDivisionError`, which occurs when division or modulo by zero is attempted.

Question: How do you handle multiple exceptions in Python?

Answer: You can handle multiple exceptions in Python by using multiple `except` blocks, each handling a specific type of exception. Alternatively, you can use a single `except` block followed by a tuple of exception types to handle multiple exceptions with the same block of code.

Question: What is the purpose of the finally block in Python exception handling?

Answer: The `finally` block in Python exception handling is used to define cleanup code that should be executed regardless of whether an exception occurs or not. It ensures that certain operations, such as closing files or releasing resources, are always performed, even if an exception is raised.

Question: How do you raise custom exceptions in Python?

Answer: You can raise custom exceptions in Python by using the `raise` statement followed by an instance of the desired exception class. Custom exceptions are typically defined by subclassing the built-in `Exception` class or one of its subclasses, allowing you to create specialized exceptions for your application.