Features of Python:

1. **Easy to Learn and Read:** Python has a clean and readable syntax, making it easy for beginners to learn and write code quickly.
2. **Expressive Language:**Python allows developers to express concepts in fewer lines of code than languages like C++ or Java. This promotes code readability and reduces development time.
3. **Interpreted Language:** Python is an interpreted language, which means that the code can be executed line by line, facilitating rapid development and debugging.
4. **Dynamically Typed:**Python is dynamically typed, allowing developers to create variables without specifying their data types. This makes the code more flexible and adaptable..
5. **Cross-Platform:**Python is platform-independent, meaning that Python code can run on various operating systems without modification.
6. **Object-Oriented:**Python supports object-oriented programming principles, facilitating code organization and reuse.

Applications of Python:

1. **Web Development:**Python is used for building dynamic websites and web applications using frameworks like Django, Flask, and Pyramid.
2. **Data Science and Machine Learning:**Python is widely used in data science and machine learning for tasks like data analysis, visualization, and implementing machine learning models using libraries like NumPy, Pandas, Matplotlib, and TensorFlow.
3. **Scientific Computing:**Python is used in scientific research and engineering for numerical and scientific computing with libraries like SciPy and NumPy.
4. **Automation and Scripting:**Python is often used for automation and scripting tasks, allowing developers to automate repetitive tasks and system administration.
5. **Game Development:**Python is employed in game development using libraries like Pygame, enabling the creation of 2D games.
6. **Cybersecurity:**Python is utilized in cybersecurity for tasks such as penetration testing, analyzing security vulnerabilities, and developing security tools.

Module and Package in Python: In Python, modules serve as a way to organize code by grouping related functions, classes, or variables into a single file. Modules can be imported into other modules or scripts, promoting code reuse and organization. Each module is a self-contained unit, contributing to a modular and maintainable codebase.On the other hand, packages extend the concept of modules by organizing them into a directory hierarchy. A package includes an `__init__.py` file to indicate that the directory should be treated as a package. This hierarchical structure allows developers to organize code at a higher level, making it easier to manage large projects.For example, a package named `mypackage` may contain modules like `module1.py`, `module2.py`, and a subpackage (`subpackage`) with its own modules (`module3.py`). This modular approach enhances code structure and supports scalability in Python projects, fostering code readability and collaboration.

```
# Example of modules and packages in Python
def greet(name):
    return f"Hello, {name}!"
def add_numbers(a, b):
    return a + b
from mypackage import module1, module2
result1 = module1.greet("Alice")
result2 = module2.add_numbers(3, 7)
print(result1)
print(result2)
```

a) Write a Python program to check if a given number is Armstrong.

```
def is_armstrong(number):
    num_str = str(number)
    num_digits = len(num_str)
        armstrong_sum =
sum(int(digit) ** num_digits for
digit in num_str)
        return armstrong_sum ==
number
num_to_check = 153
if is_armstrong(num_to_check):
    print(f"{num_to_check} is an
Armstrong number.")
else:
    print(f"{num_to_check} is not an
Armstrong number.")
```

b) Write a Python program to find gcd of a number using recursion.

```
def gcd_recursive(a, b):
    if b == 0:
        return a
    else:
        return gcd_recursive(b, a % b)
num1 = 48
num2 = 18
result = gcd_recursive(num1, num2)
print(f"GCD of {num1} and {num2}
is {result}")
```

b) Write a program in python to accept 'n' integers in a list, compute & display addition of all squares of these integers.

```
def sum(integers):
 return sum(x ** 2 for x in integers)
n = int(input("Enter the number of
integers: "))
integer_list = [int(input(f"Enter
integer {i + 1}: ")) for i in range(n)]
result =sum(integer_list)
print(f"The addition of squares of
the integers is: {result}")
```

b) Write a Python program to display power of 2 using anonymous function

```
power_of_2 = lambda x: 2 ** x
for i in range(6):
    print(f"2^{i} = {power_of_2(i)}")
```

c) Write a Python program to print even length words in a string

```
def even(sentence):
    words = sentence.split()
even_length_words = [word for word in
words if len(word) % 2 == 0]
    print("Even length words:", ",
".join(even_length_words))
input_sentence = "Python is a
programming language"
even(input_sentence)
```

c) Write a Python program to check if a given key already exists in a dictionary

```
def key_exists(dictionary, key):
    return key in dictionary
sample_dict = {'a': 1, 'b': 2, 'c': 3}
search_key = 'b'

if key_exists(sample_dict, search_key):
    print(f"The key '{search_key}' exists in
the dictionary.")
else:
    print(f"The key '{search_key}' does
not exist in the dictionary.")
```

Write a program to read an entire text file

```
file_path = "example.txt"
try:
    with open(file_path, 'r') as file:
        content = file.read()
        print("File Content:\n", content)
except FileNotFoundError:
    print(f"Error: File '{file_path}' not
found.")
except Exception as e:
    print(f"Error: {e}")
```

a) Write a Python program to check for Zero Division Error Exception.

```
try:
    numerator =
int(input("Enter the
numerator: "))
    denominator =
int(input("Enter the
denominator: "))

    result = numerator /
denominator
    print(f"Result: {result}")

except ZeroDivisionError:
    print("Error: Division by
zero is not allowed.")
except ValueError:
    print("Error: Please enter
valid numeric values.")
```

a) Write a recursive function in Python to display addition of digits in single digit

```
def
sum_of_digits_recursive(num)
:
   if num < 10:
       return num
    else:
       return num % 10 +
sum_of_digits_recursive(num
// 10)
number = 9875
result =
sum_of_digits_recursive(num
ber)
print(f"The sum of digits in
{number} is {result} (a single
digit).")
```

Write a Python script to print a dictionary where the keys are numbers between 1 and 15 (both included) and the values are the square of the keys.

```
square_dict = {i: i**2 for i in
range(1, 16)}
print("Sample Dictionary:")
print(square_dict)
```

Write a Python function that accepts a string and counts the number of upper and lower case letters

```
def
count_upper_lower_letters(input_str
ing):
    upper_count = 0
    lower_count = 0
 for char in input_string:
        if char.isalpha():
            if char.isupper():
                upper_count += 1
            elif char.islower():
                lower_count += 1
 return upper_count, lower_count
user_input = input("Enter a string: ")
upper_letters, lower_letters =
count_upper_lower_letters(user_inp
ut)
print(f"Number of uppercase
letters: {upper_letters}")
print(f"Number of lowercase letters:
{lower_letters}")
```

a) Write a python program to Count Vowels and Consonants in a String

```
def
count_vowels_and_consonants(input_str
ing):
    vowels = "aeiouAEIOU"
    vowel_count = 0
    consonant_count = 0

    for char in input_string:
        if char.isalpha():
            if char in vowels:
                vowel_count += 1
            else:
                consonant_count += 1

    return vowel_count, consonant_count
user_input = input("Enter a string: ")
vowels, consonants =
count_vowels_and_consonants(user_inp
ut)
print(f"Number of vowels: {vowels}")
print(f"Number of consonants:
{consonants}")
```

Write a program to get a single string from two given strings, separated by space and swap the first two characters of each string Sample input: ‘abc’, ‘pqr’ Output: pqc abr

```
def swap_first_two_chars(str1,
str2):
    new_str1 = str2[:2] +
str1[2:]
    new_str2 = str1[:2] +
str2[2:]
        result = new_str1 + ' ' +
new_str2
    return result
string1 = input("Enter the
first string: ")
string2 = input("Enter the
second string: ")
output =
swap_first_two_chars(string1,
string2)
print("Output:", output)
```

Program to print the reciprocal of even numbers using exception

```
try:
    number = int(input("Enter an
even number: "))
    if number % 2 == 0:
        reciprocal = 1 / number
        print(f"The reciprocal of
{number} is: {reciprocal}")
    else:
        raise ValueError("Input is
not an even number.")
except ValueError as ve:
    print(f"Error: {ve}")
except ZeroDivisionError:
    print("Error: Cannot calculate
reciprocal of 0.")
except Exception as e:
    print(f"Error: {e}")
```

Program to demonstrate file opening exception

```
try:
    with open('nonexistent_file.txt',
'w') as file:
        file.write("Trying to write to
a file.")
except FileNotFoundError:
    print("Error: The specified file
does not exist.")
except PermissionError:
    print("Error: Permission denied.
Cannot write to the file.")
except Exception as e:
    print(f"Error: {e}")
```

Write a Python program to find sequences of lowercase letters joined with an underscore.

```
import re

def find_lowercase_sequences(text):
    pattern = re.compile(r'[a-z]+(_[a-
z]+)*')
    matches = pattern.findall(text)
    return matches

# Test the function
sample_text = "hello_world and
python_programming"
result =
find_lowercase_sequences(sample_text)
print("Sequences of lowercase letters
joined with an underscore:", result)
```

Exception Handling in Python: Exception handling is a critical aspect of writing robust and fault-tolerant Python code. The `try`, `except`, `else`, and `finally` blocks provide a structured way to handle runtime errors. The `try` block encloses code that might raise an exception. If an exception occurs, the corresponding `except` block is executed, allowing developers to handle the error gracefully. The `else` block contains code that runs if no exceptions occur, providing an opportunity to execute additional logic. The `finally` block ensures that certain code, such as cleanup operations, is executed whether an exception is raised or not.

Write a Python program to count the number of lines in a text file

```
file_path = "sample.txt"  # Replace with the path to your file

try:
    with open(file_path, 'r') as file:
        line_count = sum(1 for line in file)
        print(f"Number of lines in the file: {line_count}")
except FileNotFoundError:
    print(f"Error: File '{file_path}' not found.")
except Exception as e:
    print(f"Error: {e}")
```

Write a Python program to read an entire text file

```
file_path = "sample.txt"  # Replace with the path to your file

try:
    with open(file_path, 'r') as file:
        file_content = file.read()
        print("Entire file content:")
        print(file_content)
except FileNotFoundError:
    print(f"Error: File '{file_path}' not found.")
except Exception as e:
    print(f"Error: {e}")
```

Write a Python program to read first n lines of a file.

```
file_path = "sample.txt"  # Replace with the path to your file
n = 5  # Replace with the desired number of lines

try:
    with open(file_path, 'r') as file:
        first_n_lines = [next(file) for _ in range(n)]
        print(f"First {n} lines of the file:")
        print(first_n_lines)
except FileNotFoundError:
    print(f"Error: File '{file_path}' not found.")
except Exception as e:
    print(f"Error: {e}")
```

Which methods are used to read from a file? Explain any two with example.

Using `read()` method: The `read()` method is used to read the entire contents of a file as a single string. It reads the specified number of bytes from the file or the entire file if no size is specified.

```
# Opening a file in read mode
file_path = "example.txt"
with open(file_path, "r") as file:
    content = file.read()
    print(content)
```

Using `readline()` method: The `readline()` method is used to read a single line from the file. It reads characters from the current position until it encounters a newline character ('\n').

```
# Opening a file in read mode
file_path = "example.txt"
with open(file_path, "r") as file:
    line = file.readline()
    while line:
        print(line, end="")
        line = file.readline()
```

List -A list is a container which holds elements separated by comma between square brackets.A list is an object that contains multiple data items (elements). We can update the list such as Adding and removing elements during execution.

List Concatenation-Concatenation is done by + operator. Concatenation is supported by sequence data types(string. List, tuple). Concatenation is done between the same data types only.

List Repetition -The repetition operator will make multiple copies of that particular object and combines them 55,23) together. When is used with an integer, it performs multiplication but with list, tuple or strings it performs a repetition.

The 'in' Operator in Python -Python has two membership operators – “in” and “not in”. They are used to check if an element present in a sequence or not. The Python in operator is used to check if the element is present sequence. This operator can be used with loops and conditions and even just to ensure that a specific value is present when taking user input. Python “not in” Operator -The not

in operator is the opposite of the in operator. It checks for a particular element in a sequence and if it is not found returns true otherwise returns false

. **Tuple Assignment** -Tuples are basically a data type in python. These tuples are an ordered collection of elements of different data types. The process of assigning values to a tuple is known as packing. In python, we can perform tuple assignment. We can initialise or create a tuple in various ways.The tuple assignment Is also called as unpacking of tuple.The unpacking or tuple assignment is the process of assigning the values on the right-hand side to the left-hand side variables.

Dictionary -Dictionary stores data in the form of key-value pairs. The keys defined for a dictionary need to be unique, only immutable objects are allowed for keys.The values in a dictionary can be mutable or immutable objects.

Operations in Dictionary -Dictionaries are known as hash tables in other programming languages, these provide us a table associative array type, through two elements, a key and a value. In Python dictionaries, the keys must be an Immutable data type so that the dictionary must be consistent.1)**Definition operations**: These operations allow us to define or create a dictionary.2)Mutable operation: This operation allows us to work with dictionary such as altering or updating.3)Immutable operations: These operations allow us to work with dictionaries without altering or modifying their previous definition.

Properties of Dictionary Keys -There are two important points to remember about dictionary keys:1)Key must be unique: Dictionary keys must be unique because we use them for accessing values. If keys are duplicated then there will be ambiguity to access the value as mon than one values are having similar key which is not possible in dictionary).2)Keys must be immutable: Dictionary keys must be of an immutable type. Strings and numbers are the two most commonly used data types as dictionary keys. We can also use tuples as keys but they must contain only strings, integers, or other tuples. Stack Diagram -The stack diagram is used to represent the state of a program during a function call. The same kind of diagram can help interpret a recursive function.

Stack diagram -The stack diagram is used to represent the state of a program during a function call. The same kind of diagram can help interpret a recursive function.Stack diagram provides detail information about variable value and function call.

Regular Expression -Python Regular Expression or Python RegEx is pattern that permits us to ‘match’ various string values in a variety of ways. A pattern is simply one or more characters that represents a set of possible match characters. Regular expressions are a powerful language for matching text patterns.

Exception Handling -An exception can be defined as an unusual condition in a program resulting in the interruption in the flow of the program. Whenever an exception occurs, the program stops the execution, and thus the further code is not executed.

Raising exceptions: An exception can be raided by using the raise clause in Python. The syntax to use the raise statement is given below.Syntax: raise Exception.To raise an exception, raise statement is used followed by the exception class name.Once a user defined exception is created then we can write a tryexcept block to catch the user-defined exception in Python. For testing, inside the try block we are raising exception using raise keyword.

User Defined Exception- Run time error (Exception) situation can be handled by the developer by creating user defined exception. Defining your own exception helps us to handle such errors which can be generated during execution of program and hence the software will be developed without any erroneous statements. Python allows us to create our exceptions that can be raised from the program and caught using the except clause.

1. **Regular Expression (Regex):**
    - **Definition:** A regular expression is a powerful tool for pattern matching and searching in strings. It provides a concise and flexible means for matching strings based on certain patterns.
2. **Python as a Case-Sensitive Language:**
    - **Comment:** Python is case-sensitive, meaning that it distinguishes between uppercase and lowercase letters. For example, variables `variable` and `Variable` would be considered different in Python.
3. **Seek and Tell Functions:**
    - **Use of `seek` and `tell`:**
        - `seek(offset, whence)`: Moves the file pointer to a specified position.
        - `tell()`: Returns the current position of the file pointer.
4. **Enumerate Function:**
    - **Explanation:** The `enumerate()` function in Python is used to add a counter to an iterable (e.g., a list) and return it as an enumerate object, which contains pairs of index and element.
5. **Extend Method of List:**
    - **Explanation:** The `extend()` method is used to extend a list by appending elements from an iterable (e.g., another list) to the end of the original list.
6. **Use of `pass` Statement:**
    - **Explanation:** The `pass` statement in Python is a null operation. It is a no-op placeholder, and it is often used as a syntactic placeholder where some code is required syntactically but no action is desired or necessary.
7. **Filter, Map, Reduce:**
    - `filter()`: Filters elements of an iterable based on a function.
    - `map()`: Applies a function to all items in an input list and returns an iterator of the results.
    - `reduce()`: Applies a rolling computation to sequential pairs of values in an iterable.
8. **Recursion:**
    - **Explanation:** Recursion is a programming concept where a function calls itself in its own definition. It is particularly useful for solving problems that can be broken down into smaller instances of the same problem.
9. **Lambda Form, Union, Intersection, Package, Raw String:**
    - The terms are mentioned without specific details. If you have specific questions about any of these terms, please provide more details.

1. **Dry Run:**
    - **Definition:** A dry run is a step-by-step process of running a program or algorithm on paper before executing it on a computer. It helps in understanding the flow of the program, identifying errors, and analyzing the expected output.
2. **Lambda Function:**
    - **Definition:** A lambda function in Python is an anonymous function created using the `lambda` keyword. It is a concise way to define small, one-time-use functions without the need for a formal function definition.

Python as a Scripting Language:

- **Comment:** Python is often referred to as a scripting language because it allows developers to write and execute scripts quickly and interactively. It is an interpreted language, which means that code can be executed line by line without the need for compilation.