Mega Python Assignment- 2

Q1. Why do we call Python as a general purpose and high-level programming language?

Sol)

General-purpose: Python is a versatile language that can be used for a wide range of applications. It supports various programming paradigms, including procedural, object-oriented, and functional programming. Python can be used for developing web applications, desktop applications, scientific computing, data analysis, artificial intelligence, machine learning, and more. Its flexibility and extensive libraries make it suitable for diverse tasks, making it a general-purpose language.

High-level: Python is considered a high-level language because it abstracts away many low-level details that are inherent in other programming languages. It provides constructs and syntax that are closer to human language, making it easier to read, write, and understand. Python allows programmers to focus more on solving problems rather than dealing with complex system-level details, memory management, or hardware interactions. This high-level nature enhances productivity and reduces the time required to develop software.

Q2. Why is Python called a dynamically typed language?

Sol)

Python is called a dynamically typed language because the type of a variable is determined and checked at runtime, rather than being explicitly declared or enforced at compile-time. In dynamically typed languages like Python, variables can hold values of different types during the execution of a program, and the type of a variable can be changed as needed.

Q3. List some pros and cons of Python programming language?

Sol)

Pros:

- Readability and simplicity
- Extensive standard library and third-party packages
- Cross-platform compatibility

Cons:

- Performance
- Mobile and browser limitations
- Memory consumption:

Q4. In what all domains can we use Python?

Sol)

- Web Development
- AI
- Data Analytics and Big Data
- Scientific Computing
- Internet of things
- Desktop applications
- Game Development

Q5. What are variable and how can we declare them?

Sol)

In Python, a variable is a named reference to a value stored in memory. It represents a location in memory that can hold data of different types, such as numbers, strings, lists, or objects. Variables allow us to store and manipulate data in our programs.

To declare or create a variable in Python, you simply assign a value to a name using the assignment operator (=). Here's the basic syntax for variable declaration

```
age = 25
```

Q6. How can we take an input from the user in Python?

Sol)

```
age = input("Enter your age: ")
print("Your age is " + age)
```

Q7. What is the default datatype of the value that has been taken as an input using input() function?

Sol)
The default data type of the value returned by the input() function in Python is a string.

```
age = input("Enter your age: ")
print(type(age))
```

```
<class 'str'>
```

Q8. What is type casting?

Sol)

Type casting, also known as type conversion, refers to the process of converting a value from one data type to another in a programming language. Type casting allows you to change the interpretation or representation of a value to suit the requirements of a specific operation or to store it in a different type of variable.

```
# Float to integer
a = 5.8
b = int(a)  # b is now 5 (integer)
```

Q9. Can we take more than one input from the user using single input() function? If yes, how? If no, why?

Sol)

No, the input() function in Python does not allow you to directly take multiple inputs using a single function call. The input() function is designed to prompt the user for a single value and return that value as a string.

Q10. What are keywords?

Sol)

In programming, keywords, also known as reserved words, are predefined words or identifiers that have special meanings and purposes within the programming language. These keywords are reserved by the language and cannot be used as variable names or other identifiers.

```
False      class      finally    is         return
None       continue   for        lambda     try
```

Q11. Can we use keywords as a variable? Support your answer with reason.

Sol)

No, we cannot use keywords as variables in Python or any programming language. Keywords are reserved by the language and have predefined meanings and purposes. They are specifically used to define the syntax and structure of the programming language.

```
if = 10  # SyntaxError: invalid syntax
```

In this case, you will encounter a SyntaxError because if is a keyword used for conditional statements, not for variable assignment.

Q12. What is indentation? What's the use of indentaion in Python?

Sol)

In Python, indentation refers to the whitespace at the beginning of a line of code. It is used to define the structure, grouping, and hierarchy of statements within the program. In most programming languages, indentation is primarily for readability purposes, but in Python, it serves a more significant role.

Q13. How can we throw some output in Python?

Sol)

To generate output in Python, you can use the print() function. The print() function is a built-in function that allows you to display text, variables, or other values on the console or output stream.

```
print("Hello, World!")
```

Q14. What are operators in Python?

Sol)

In Python, operators are special symbols or characters that perform specific operations on one or more operands (values or variables). Operators allow you to manipulate and perform computations on data, make comparisons, assign values, and more. They are an essential part of the Python language and are used extensively in programming.

Python supports a wide range of operators that can be classified into different categories based on their functionality. Here are the main categories of operators in Python:

Arithmetic Operators: Arithmetic operators perform mathematical operations on numeric operands and include addition (+), subtraction (-), multiplication (*), division (/), floor division (//), modulus (%), and exponentiation (**).

Comparison Operators: Comparison operators compare two values and return a Boolean result (True or False). They include equal to (==), not equal to (!=), greater than (>), less than (<), greater than or equal to (>=), and less than or equal to (<=).

Assignment Operators: Assignment operators are used to assign values to variables. The most common assignment operator is the equal sign (=), but there are also compound assignment operators like +=, -=, *=, /=, and more.

Logical Operators: Logical operators perform logical operations on Boolean values or expressions. They include and, or, and not. These operators are used to combine conditions or invert the result of a condition.

Bitwise Operators: Bitwise operators perform operations on binary representations of integers. They include bitwise AND (&), bitwise OR (|), bitwise XOR (^), bitwise left shift (<<), and bitwise right shift (>>).

Membership Operators: Membership operators test whether a value is a member of a sequence or collection. They include in and not in.

Identity Operators: Identity operators compare the identity of two objects, whether they refer to the same memory location or not. They include is and is not.

Q15. What is difference between / and // operators?

Sol)

The key difference between / and // is that / always produces a floating-point result, while // always produces an integer result by truncating any fractional part.

Q16. Write a code that gives following as an output.

Sol)

```
iNeuroniNeuroniNeuroniNeuron
```

```
output = "iNeuroni" * 3 + "Neuron"
print(output)
```

Q17. Write a code to take a number as an input from the user and check if the number is odd or even.

Sol)

```
number = int(input("Enter a number: "))  # Take input from the user
and convert it to an integer

if number % 2 == 0:
    print(number, "is even.")
else:
    print(number, "is odd.")
```

Q18. What are boolean operator?

Sol)

Boolean operators in Python are used to perform logical operations on Boolean values or expressions. They take one or more Boolean operands and return a Boolean result (True or False) based on the outcome of the logical operation.

Q19. What will the output of the following?

Sol)

```
1 or 0  -  1

0 and 0  - 0
```

True and False and True   -   False

1 or 0 or 0   -   1
```

Q20. What are conditional statements in Python?

Sol)

Conditional statements in Python allow you to control the flow of your program based on certain conditions. They help you make decisions and execute specific blocks of code selectively, depending on whether a condition is true or false. The main conditional statements in Python are:

if statement:
if-else
if-elif-else

Q21. What is use of 'if', 'elif' and 'else' keywords?

Sol)

'if' keyword:

- The 'if' keyword is used to specify a condition that is evaluated as either true or false.
- If the condition is true, the code block following the 'if' statement is executed.
- If the condition is false, the code block is skipped, and the program continues to the next statement after the 'if' block.
```

'elif' keyword:

- The 'elif' keyword (short for "else if") is used to specify an alternative condition to be checked if the previous 'if' or 'elif' conditions are false.
- 'elif' allows you to test multiple conditions one by one until a condition is found to be true or until all conditions have been evaluated.
- If an 'elif' condition is true, the corresponding code block is executed, and the program skips the remaining conditions and blocks.

'else' keyword:

- The 'else' keyword is used as the final part of a conditional statement.
- It specifies a block of code to be executed if all the preceding conditions ('if' and 'elif') are false.
- The 'else' block is optional, but it provides a fallback option for cases when none of the preceding conditions are true.
- If none of the conditions preceding the 'else' statement are true, the code block following the 'else' statement is executed.

Q22. Write a code to take the age of person as an input and if age >= 18 display "I can vote". If age is < 18 display "I can't vote".

Sol)

```
age = int(input("Enter your age: "))  # Take input from the user and
convert it to an integer

if age >= 18:
    print("I can vote.")
else:
    print("I can't vote.")
```

Q23. Write a code that displays the sum of all the even numbers from the given list.

```
numbers = [12, 75, 150, 180, 145, 525, 50]
```

```
numbers = [12, 75, 150, 180, 145, 525, 50]

sum_even = 0


for number in numbers:

    if number % 2 == 0:

        sum_even += number


print("Sum of even numbers:", sum_even)
```

Sol)

Q24. Write a code to take 3 numbers as an input from the user and display the greatest no as output.

Sol)

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

if num1 >= num2 and num1 >= num3:
    greatest = num1
elif num2 >= num1 and num2 >= num3:
    greatest = num2
else:
    greatest = num3

print("The greatest number is:", greatest)
```

Q25. Write a program to display only those numbers from a list that satisfy the following conditions

- The number must be divisible by five

- If the number is greater than 150, then skip it and move to the next number

- If the number is greater than 500, then stop the loop

```
numbers = [12, 75, 150, 180, 145, 525, 50]
```

Sol)

```
numbers = [12, 75, 150, 180, 145, 525, 50]

for number in numbers:
    if number > 500:
        break  # Stop the loop if the number is greater than 500

    if number > 150:
        continue  # Skip the current iteration if the number is
greater than 150

    if number % 5 == 0:
        print(number)
```

Q26. What is a string? How can we declare string in Python?

Sol)

In Python, a string is a sequence of characters enclosed within single quotes (' ') or double quotes (" "). It is a data type used to represent text or a sequence of characters.

```
my_string1 = 'Hello, World!'
```

Q27. How can we access the string using its index?

Sol)

In Python, you can access individual characters or a specific range of characters within a string using indexing. The index of a string starts from 0 for the first character and goes up to the length of the string minus one.

```
my_string = "Python"

# Accessing individual characters
print(my_string[0])  # Output: 'P'
```

Q28. Write a code to get the desired output of the following

```
string = "Big Data iNeuron"
desired_output = "iNeuron"

string = "Big Data iNeuron"
desired_output = string[9:]

print(desired_output)
```

Q29. Write a code to get the desired output of the following

```
string = "Big Data iNeuron"
desired_output = "norueNi"
```

Sol)

```
string = "Big Data iNeuron"
desired_output = string[13:6:-1]

print(desired_output)
```

Q30. Resverse the string given in the above question.

Sol)

```
string = "Big Data iNeuron"
reversed_string = string[::-1]

print(reversed_string)
```

Q31. How can you delete entire string at once?

Sol)

```
string = "Hello, World!"

# Deleting the variable holding the string
del string

# Trying to access the deleted string will raise an error
print(string)
```

Q32. What is escape sequence?

Sol)

In Python, an escape sequence is a combination of characters that begins with a backslash (\) and is used to represent special characters or to include characters that cannot be easily entered or displayed directly. When the escape sequence is encountered in a string, it is interpreted as a special character or a specific action instead of its literal representation.

Escape sequences are used to handle characters that have special meanings or cannot be easily typed or displayed directly, such as newline characters (\n), tab characters (\t), backslashes (\\), or quotes (\" or \')

Q33. How can you print the below string?

'iNeuron's Big Data Course'

```
print("iNeuron's Big Data Course")
```

Q34. What is a list in Python?

Sol)

In Python, a list is a data structure that allows you to store and organize a collection of items. It is one of the built-in data types in Python and is mutable, which means its elements can be modified after creation. Lists are ordered and can contain elements of different data types, such as integers, floats, strings, or even other lists.

Lists are defined by enclosing comma-separated elements within square brackets [ ]. Here's an example of creating a list:

```
fruits = ["apple", "banana", "orange"]
```

Q35. How can you create a list in Python?

Sol)

Lists are defined by enclosing comma-separated elements within square brackets [ ]. Here's an example of creating a list:

```
fruits = ["apple", "banana", "orange"]
```

Q36. How can we access the elements in a list?

Sol)

```
fruits = ["apple", "banana", "orange"]
print(fruits[0])   # Output: apple
print(fruits[2])   # Output: orange
```

Q37. Write a code to access the word "iNeuron" from the given list.

```
lst = [1,2,3,"Hi",[45,54, "iNeuron"], "Big Data"]
```

Q38. Take a list as an input from the user and find the length of the list.

Sol)

```
lst = [1, 2, 3, "Hi", [45, 54, "iNeuron"], "Big Data"]
word = lst[4][2]
print(word)  # Output: iNeuron
```

Q39. Add the word "Big" in the 3rd index of the given list.

```
lst = ["Welcome", "to", "Data", "course"]
```

Sol)

```
lst = ["Welcome", "to", "Data", "course"]
lst.insert(3, "Big")
print(lst)
```

Q40. What is a tuple? How is it different from list?

Sol)

In Python, a tuple is another built-in data type that allows you to store a collection of items. A tuple is similar to a list in many ways, but there are some key differences between the two.

Here are the characteristics of a tuple:

- Immutable: Unlike lists, tuples are immutable, which means their elements cannot be modified once they are assigned. Once a tuple is created, you cannot add, remove, or change elements in it.
- Ordered: Similar to lists, tuples maintain the order of their elements. You can access elements in a tuple using indexing or slicing, just like you would with a list.
- Heterogeneous: Tuples can contain elements of different data types, just like lists. You can have a tuple with a combination of integers, floats, strings, or even other tuples.
- Enclosed in parentheses: Tuples are defined by enclosing comma-separated elements within parentheses (). However, parentheses are optional and are usually omitted when creating a tuple.

Q41. How can you create a tuple in Python?

Sol)

```
my_tuple = (1, 2, "hello", 3.5)
```

Q42. Create a tuple and try to add your name in the tuple. Are you able to do it? Support your answer with reason.

Sol)

No, I cannot add my name to a tuple. Tuples are immutable, which means that once they are created, their elements cannot be modified or added. Therefore, it is not possible to directly add or modify elements in a tuple.

Q43. Can two tuple be appended. If yes, write a code for it. If not, why?

Sol)

No, two tuples cannot be appended directly. Tuples are immutable, which means their elements cannot be modified once they are assigned. Therefore, it is not possible to append or modify a tuple directly.

Q44. Take a tuple as an input and print the count of elements in it.

Sol)

```
my_tuple = (1,2,3,4,5,6)
count = len(my_tuple)
count
```

Q45. What are sets in Python?

Sol)

In Python, a set is an unordered collection of unique elements. It is a built-in data type that is used to store a collection of distinct items. The elements in a set are not stored in any particular order, and duplicates are not allowed.

Here are some key characteristics of sets in Python:

- Unordered: Sets do not maintain any specific order of elements. The elements are stored in an unordered manner.
- Unique Elements: Sets only contain unique elements. Duplicate elements are automatically removed.
- Mutable: Sets are mutable, which means you can add or remove elements from a set after it is created.
- Enclosed in Curly Braces: Sets are defined by enclosing comma-separated elements within curly braces {}.

Q46. How can you create a set?

Sol)

```
my_set = {1, 2, 3, 4, 5}
```

Q47. Create a set and add "iNeuron" in your set.

Sol)

```
my_set = {"Python", "Java", "C++"}  # Create a set

my_set.add("iNeuron")  # Add "iNeuron" to the set

print(my_set)  # Output: {'C++', 'Java', 'Python', 'iNeuron'}
```

Q48. Try to add multiple values using add() function.

Sol)

The **add()** function in Python's set data type allows you to add a single element at a time. If you want to add multiple values to a set, you can use the **update()** function or the union (|) operator.

Q49. How is update() different from add()?

Sol)

The add() function in Python's set data type allows you to add a single element at a time. If you want to add multiple values to a set, you can use the update() function or the union (|) operator.

```
my_set = {"Python", "Java", "C++"}  # Create a set

my_set.update(["iNeuron", "JavaScript", "R"])  # Add multiple values
using update()

print(my_set)
```

Q50. What is clear() in sets?

Sol)

In Python sets, the clear() method is used to remove all the elements from a set, making it an empty set. It modifies the set in-place and does not return any value.

Q51. What is frozen set?

Sol)

Q52. How is frozen set different from set?

Sol)

In Python, a frozen set is an immutable version of a set. It is a built-in data type that represents an unordered collection of unique elements, similar to a regular set. The key difference is that a frozen set is immutable, meaning its elements cannot be modified after creation.

Q53. What is union() in sets? Explain via code.

Sol)

In Python sets, the union() method is used to return a new set that contains all the unique elements from two or more sets. It combines the elements from multiple sets into a single set, excluding any duplicates.

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

union_set = set1.union(set2)

print(union_set)
```

Q54. What is intersection() in sets? Explain via code.

Sol)

In Python sets, the intersection() method is used to return a new set that contains the common elements present in two or more sets. It retrieves the elements that are common between the sets.

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

intersection_set = set1.intersection(set2)

print(intersection_set)
```

Q55. What is dictionary in Python?

Sol)

In Python, a dictionary is a built-in data type that represents an unordered collection of key-value pairs. It is also known as an associative array or a hash map in some programming languages. Dictionaries are used to store and retrieve data using keys rather than indexing based on numerical positions.

Here are some characteristics of dictionaries:

- Key-Value Pairs: A dictionary consists of key-value pairs. Each key is unique within the dictionary, and it is used to access its corresponding value. The key-value pairs are separated by colons (:) and enclosed in curly braces ({}).
- Mutable: Dictionaries are mutable, meaning you can add, remove, or modify key-value pairs after the dictionary is created.
- Unordered: The elements in a dictionary are not ordered by their insertion order. The order of key-value pairs may vary when iterating over the dictionary.
- Dynamic Size: Dictionaries can dynamically grow or shrink as key-value pairs are added or removed.

Q56. How is dictionary different from all other data structures.

Sol)

Dictionaries in Python have several distinct characteristics that differentiate them from other data structures:

- Key-Value Structure: Dictionaries are unique in that they store data as key-value pairs. Other data structures like lists, tuples, and sets primarily store values without any associated keys. The key-value structure of dictionaries allows for efficient lookup and retrieval of values based on keys.

- Unordered: Dictionaries are unordered data structures, meaning they do not maintain a specific order of key-value pairs. The order of items in a dictionary is not guaranteed and can change during operations. In contrast, lists and tuples preserve the order of elements, while sets are also unordered but store only unique values.

- Unique Keys: Dictionaries enforce unique keys. Each key in a dictionary must be unique, allowing for efficient retrieval of values based on keys. If a duplicate key is assigned, the previous value associated with that key will be overwritten.

- Mutable: Dictionaries are mutable, meaning you can add, remove, or modify key-value pairs after the dictionary is created. This flexibility allows for dynamic updates and changes to the dictionary's content.

- Efficient Key Lookup: Dictionaries provide efficient key lookup. Instead of iterating through all elements to find a specific value, dictionaries use a hash function to compute the memory location of a given key, allowing for fast retrieval of the associated value. This makes dictionaries suitable for scenarios where quick access to values based on specific keys is required.

- Dynamic Size: Dictionaries can dynamically grow or shrink as key-value pairs are added or removed. They do not require a predefined size like arrays and lists, providing flexibility in managing varying amounts of data.

The unique characteristics of dictionaries make them well-suited for scenarios where data needs to be organized and accessed based on specific keys. They are commonly used for tasks such as mapping, lookups, caching, and storing configurations or data with key-based relationships.

Q57. How can we delare a dictionary in Python?

Sol)

```
student = {"name": "John", "age": 20, "grade": "A"}

print(student["name"])  # Output: John
print(student["age"])   # Output: 20
print(student["grade"]) # Output: A
```

Q58. What will the output of the following?

```
var = {}
print(type(var))
```

Sol)

```
<class 'dict'>
```

Q59. How can we add an element in a dictionary?

Sol)

```
my_dict = {}
my_dict['key'] = 'value'

my_dict = {}
my_dict['name'] = 'John'
my_dict['age'] = 25
print(my_dict)
```

Q60. Create a dictionary and access all the values in that dictionary.

Sol)

```
my_dict = {'name': 'John', 'age': 25, 'city': 'New York'}

# Method 1: Using values() method
all_values = my_dict.values()
print(all_values)

# Method 2: Iterating over the dictionary
for value in my_dict.values():
    print(value)
```

Q61. Create a nested dictionary and access all the element in the inner dictionary.

Sol)

```
my_dict = {
    'person1': {'name': 'John', 'age': 25},
    'person2': {'name': 'Alice', 'age': 30}
}

# Accessing elements in the inner dictionary
print(my_dict['person1']['name'])
print(my_dict['person2']['age'])
```

Q62. What is the use of get() function?

Sol)

The get() function in Python is used to retrieve the value associated with a given key from a dictionary. It provides a safe way to access dictionary elements without raising a KeyError if the key does not exist.

```
my_dict = {'name': 'John', 'age': 25}

# Accessing dictionary values using get()
name = my_dict.get('name')
print(name)  # Output: John
```

Q63. What is the use of items() function?

Sol)

The items() function is useful in scenarios where you need to access both the keys and values of a dictionary simultaneously, such as iterating over the items, performing specific operations on each key-value pair, or converting the dictionary to another data structure

```
my_dict = {'name': 'John', 'age': 25, 'city': 'New York'}

# Accessing key-value pairs using items()
for key, value in my_dict.items():
    print(key, value)
```

Q64. What is the use of pop() function?

Sol)

The pop() function in Python is used to remove an element from a dictionary based on its key and retrieve its corresponding value. It allows you to remove a key-value pair from a dictionary and obtain the value of the removed element.

Q65. What is the use of popitems() function?

Sol)

The popitem() function in Python is used to remove and return an arbitrary key-value pair from a dictionary. It removes and returns the last key-value pair added to the dictionary. However, the specific key-value pair that is popped is implementation-dependent and may vary across different Python versions.

Q66. What is the use of keys() function?

Sol)

The keys() function in Python is used to retrieve a view object that contains all the keys present in a dictionary. It returns a view object that provides a dynamic view of the dictionary's keys.

```
my_dict = {'name': 'John', 'age': 25, 'city': 'New York'}

# Retrieving the keys using keys()
keys = my_dict.keys()
print(keys)  # Output: dict_keys(['name', 'age', 'city'])
```

Q67. What is the use of values() function?

Sol)

The values() function in Python is used to retrieve a view object that contains all the values present in a dictionary. It returns a view object that provides a dynamic view of the dictionary's values.

```
my_dict = {'name': 'John', 'age': 25, 'city': 'New York'}

# Retrieving the values using values()
values = my_dict.values()
print(values)  # Output: dict_values(['John', 25, 'New York'])
```

Q68. What are loops in Python?

Sol)

Loops in Python are control structures that allow you to execute a block of code repeatedly based on a specific condition or for a given number of iterations. They provide a way to automate repetitive tasks and iterate over collections of data.

Q69. How many type of loop are there in Python?

Sol)

In Python, there are two types of loops:

1) For Loop
2) While Loop

Q70. What is the difference between for and while loops?

Sol)

For Loop	While Loop
Used for iterating over a sequence or iterable object.	Used for executing a block of code repeatedly based on a condition.
Has a definite iteration count.	Continues iterating until a condition becomes false.
Iterates over each element in the sequence.	Condition is checked before each iteration.

Q71. What is the use of continue statement?

Sol)

The continue statement is used in Python to skip the rest of the code inside a loop for the current iteration and move on to the next iteration. When encountered, the continue statement causes the program to jump back to the top of the loop and evaluate the loop condition again.

Q72. What is the use of break statement?

Sol)

The break statement is used in Python to exit or terminate a loop prematurely, regardless of the loop condition. When encountered, the break statement immediately terminates the innermost loop (for loop or while loop) and the program execution resumes at the next statement after the loop.

Q73. What is the use of pass statement?

Sol)

In Python, the pass statement is a placeholder statement that does nothing. It is used when a statement is syntactically required but you don't want to perform any action or write any code at that point. The pass statement acts as a null operation, allowing you to have a valid code structure without any functional code inside it.

Q74. What is the use of range() function?

Sol)

The range() function in Python is used to generate a sequence of numbers. It is often used in looping constructs to iterate over a specific range of values.

Q75. How can you loop over a dictionary?

Sol)

There are different ways to accomplish this . Some techniques are:

1) Using the for loop over dictionary
2) Using Keys() function in for loop

3) Using the values() function in the loop
4) Using items() in the loop

Coding problems

Q76. Write a Python program to find the factorial of a given number.

Sol)

```python
def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)

# Take user input
num = int(input("Enter a number: "))

# Check if the number is negative
if num < 0:
    print("Factorial cannot be calculated for a negative number.")
else:
    result = factorial(num)
    print("The factorial of", num, "is", result)
```

Q77. Write a Python program to calculate the simple interest.
Formula to calculate simple interest is SI = (PRT)/100

Sol)

```python
def calculate_simple_interest(principal, rate, time):
    simple_interest = (principal * rate * time) / 100
    return simple_interest

# Take user input
principal = float(input("Enter the principal amount: "))
rate = float(input("Enter the rate of interest: "))
time = float(input("Enter the time period (in years): "))

# Calculate simple interest
interest = calculate_simple_interest(principal, rate, time)
```

```python
# Print the result
print("The simple interest is:", interest)
```

Q78. Write a Python program to calculate the compound interest. Formula of compound interest is A = P(1+ R/100)^t.

Sol)

```python
def calculate_compound_interest(principal, rate, time):
    amount = principal * (1 + rate / 100) ** time
    compound_interest = amount - principal
    return compound_interest

# Take user input
principal = float(input("Enter the principal amount: "))
rate = float(input("Enter the rate of interest: "))
time = float(input("Enter the time period (in years): "))

# Calculate compound interest
interest = calculate_compound_interest(principal, rate, time)

# Print the result
print("The compound interest is:", interest)
```

Q79. Write a Python program to check if a number is prime or not.

Sol)

```python
def is_prime(n):
    if n <= 1:
        return False
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return False
    return True

# Take user input
```

```python
num = int(input("Enter a number: "))

# Check if the number is prime
if is_prime(num):
    print(num, "is a prime number.")
else:
    print(num, "is not a prime number.")
```

Q80. Write a Python program to check Armstrong Number.

Sol)

```python
def is_armstrong(number):
    # Convert the number to a string to get the number of digits
    num_str = str(number)
    num_digits = len(num_str)

    # Calculate the sum of the cubes of the digits
    sum_of_cubes = 0
    for digit in num_str:
        sum_of_cubes += int(digit) ** num_digits

    # Check if the sum of cubes is equal to the original number
    if sum_of_cubes == number:
        return True
    else:
        return False

# Take user input
num = int(input("Enter a number: "))

# Check if the number is an Armstrong number
if is_armstrong(num):
    print(num, "is an Armstrong number.")
else:
    print(num, "is not an Armstrong number.")
```

Q81. Write a Python program to find the n-th Fibonacci Number.

Sol)

```python
def fibonacci(n):
    if n <= 0:
        return "Invalid input. Please enter a positive integer."
    elif n == 1:
        return 0
    elif n == 2:
        return 1
    else:
        # Initialize the first two numbers of the Fibonacci sequence
        fib_1 = 0
        fib_2 = 1

        # Calculate the n-th Fibonacci number
        for _ in range(3, n + 1):
            fib_1, fib_2 = fib_2, fib_1 + fib_2

        return fib_2

# Take user input
n = int(input("Enter a positive integer (n): "))

# Find the n-th Fibonacci number
result = fibonacci(n)

# Print the result
print(f"The {n}-th Fibonacci number is:", result)
```

Q82. Write a Python program to interchange the first and last element in a list.

Sol)

```python
def interchange_first_last(lst):
    if len(lst) < 2:
        return lst
    # Swap the first and last elements
    lst[0], lst[-1] = lst[-1], lst[0]

    return lst

# Take user input for the list elements
elements = input("Enter the elements of the list (space-separated):
").split()

# Convert the elements to integers
lst = [int(num) for num in elements]

# Interchange the first and last elements in the list
result = interchange_first_last(lst)

# Print the result
print("List after interchanging first and last elements:", result)
```

Q83. Write a Python program to swap two elements in a list.

Sol)

```
def swap_elements(lst, index1, index2):
    if index1 < 0 or index2 < 0 or index1 >= len(lst) or index2 >=
len(lst):
        return "Invalid indices. Please enter valid indices within
the range of the list."

    # Swap the elements at the specified indices
    lst[index1], lst[index2] = lst[index2], lst[index1]

    return lst

# Take user input for the list elements
elements = input("Enter the elements of the list (space-separated):
").split()
```

```
# Convert the elements to integers
lst = [int(num) for num in elements]

# Take user input for the indices to swap
index1 = int(input("Enter the index of the first element to swap: 
"))
index2 = int(input("Enter the index of the second element to swap: 
"))

# Swap the elements in the list
result = swap_elements(lst, index1, index2)

# Print the result
print("List after swapping the elements:", result)
```

Q84. Write a Python program to find N largest element from a list.

Sol)

```
def find_n_largest_elements(lst, n):
    if n <= 0:
        return []

    # Sort the list in descending order
    sorted_list = sorted(lst, reverse=True)

    # Return the first N elements
    return sorted_list[:n]

# Take user input for the list elements
elements = input("Enter the elements of the list (space-separated): 
").split()

# Convert the elements to integers
lst = [int(num) for num in elements]

# Take user input for the value of N
n = int(input("Enter the value of N: "))

# Find the N largest elements in the list
result = find_n_largest_elements(lst, n)
```

```python
# Print the result
print(f"The {n} largest elements in the list:", result)
```

Q85. Write a Python program to find cumulative sum of a list.

Sol)

```python
def cumulative_sum(lst):
    total = 0

    for num in lst:
        total += num
    return total

# Take user input for the list elements
elements = input("Enter the elements of the list (space-separated): 
").split()

# Convert the elements to integers
lst = [int(num) for num in elements]

# Find the cumulative sum of the list
result = cumulative_sum(lst)

# Print the result
print("Cumulative sum of the list:", result)
```

Q86. Write a Python program to check if a string is palindrome or not.

Sol)

```python
def is_palindrome(s):
    # Remove whitespace and convert to lowercase
    s = s.replace(" ", "").lower()

    # Compare the string with its reverse
    if s == s[::-1]:
        return True
    else:
        return False

# Take user input for the string
string = input("Enter a string: ")

# Check if the string is a palindrome
result = is_palindrome(string)

# Print the result
if result:
    print("The string is a palindrome.")
else:
    print("The string is not a palindrome.")
```

Q87. Write a Python program to remove i'th element from a string.

Sol)

```
def remove_element(s, i):
    if i < 0 or i >= len(s):
        return "Invalid index. Please enter a valid index within the
range of the string."

    # Create a new string without the i'th element
    new_string = s[:i] + s[i+1:]

    return new_string

# Take user input for the string
string = input("Enter a string: ")

# Take user input for the index of the element to remove
index = int(input("Enter the index of the element to remove: "))

# Remove the i'th element from the string
result = remove_element(string, index)
```

```python
# Print the result
print("String after removing the element:", result)
```

Q88. Write a Python program to check if a substring is present in a given string.

Sol)

```python
def check_substring(string, substring):
    if substring in string:
        return True
    else:
        return False

# Take user input for the string
string = input("Enter a string: ")

# Take user input for the substring to search
substring = input("Enter a substring: ")

# Check if the substring is present in the string
result = check_substring(string, substring)

# Print the result
if result:
    print("The substring is present in the string.")
else:
    print("The substring is not present in the string.")
```

Q89. Write a Python program to find words which are greater than given length k.

Sol)

```python
def find_long_words(string, k):
    # Split the string into words
    words = string.split()

    # Find words greater than length k
    long_words = [word for word in words if len(word) > k]

    return long_words

# Take user input for the string
string = input("Enter a string: ")

# Take user input for the length, k
k = int(input("Enter the value of k: "))

# Find words greater than length k in the string
result = find_long_words(string, k)

# Print the result
print("Words greater than length", k, "in the string:", result)
```

Q90. Write a Python program to extract unquire dictionary values.

Sol)

```python
def extract_unique_values(list_of_dicts):
    unique_values = set()

    for dictionary in list_of_dicts:
        for value in dictionary.values():
            unique_values.add(value)

    return unique_values
```

```python
# Example list of dictionaries
list_of_dicts = [
    {"name": "John", "age": 30},
    {"name": "Alice", "age": 25},
    {"name": "John", "age": 35},
    {"name": "Bob", "age": 30}
]

# Extract the unique values from the list of dictionaries
result = extract_unique_values(list_of_dicts)

# Print the unique values
print("Unique values:", result,type(result))
```

Q91. Write a Python program to merge two dictionary.

Sol)

```python
def merge_dicts(dict1, dict2):
    merged_dict = {**dict1, **dict2}
    return merged_dict

# Example dictionaries
dict1 = {"name": "John", "age": 30}
dict2 = {"country": "USA", "occupation": "Engineer"}

# Merge the dictionaries
result = merge_dicts(dict1, dict2)

# Print the merged dictionary
print("Merged dictionary:", result)
```

Q92. Write a Python program to convert a list of tuples into dictionary.

```
Input : [('Sachin', 10), ('MSD', 7), ('Kohli', 18), ('Rohit', 45)]
Output : {'Sachin': 10, 'MSD': 7, 'Kohli': 18, 'Rohit': 45}
```

Sol)

```python
def convert_to_dictionary(lst):
    dictionary = dict(lst)
    return dictionary

# Example list of tuples
lst = [('Sachin', 10), ('MSD', 7), ('Kohli', 18), ('Rohit', 45)]

# Convert the list of tuples to a dictionary
result = convert_to_dictionary(lst)

# Print the resulting dictionary
print("Converted dictionary:", result)
```

Q93. Write a Python program to create a list of tuples from given list having number and its cube in each tuple.

```
Input: list = [9, 5, 6]
Output: [(9, 729), (5, 125), (6, 216)]
```

Sol)

Q94. Write a Python program to get all combinations of 2 tuples.

Sol)

```
Input : test_tuple1 = (7, 2), test_tuple2 = (7, 8)
Output : [(7, 7), (7, 8), (2, 7), (2, 8), (7, 7), (7, 2), (8, 7),
(8, 2)]
```

Q95. Write a Python program to sort a list of tuples by second item.

```
Input : [('for', 24), ('Geeks', 8), ('Geeks', 30)]
Output : [('Geeks', 8), ('for', 24), ('Geeks', 30)]
```

Sol)

Q96. Write a python program to print below pattern.

```
*
* *
* * *
* * * *
* * * * *
```

Sol)

```python
def print_pattern(n):
    for i in range(1, n+1):
        for j in range(i):
            print("*", end=" ")
        print()

# Take user input for the number of rows
n = int(input("Enter the number of rows: "))

# Print the pattern
print_pattern(n)
```

Q97. Write a python program to print below pattern.

```
    *
   **
  ***
 ****
*****
```

Sol)

```python
def print_pattern(n):
    for i in range(1, n+1):
        spaces = " " * (n - i)
        asterisks = "*" * i
        print(spaces + asterisks)

# Take user input for the number of rows
n = int(input("Enter the number of rows: "))

# Print the pattern
print_pattern(n)
```

Q98. Write a python program to print below pattern.

```
    *
   * *
  * * *
 * * * *
* * * * *
```

Sol)

```python
def print_pattern(n):
    for i in range(1, n+1):
        spaces = " " * (n - i)
        asterisks = "* " * i
        print(spaces + asterisks)

# Take user input for the number of rows
n = int(input("Enter the number of rows: "))

# Print the pattern
print_pattern(n)
```

Q99. Write a python program to print below pattern.

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

Sol)

```
def print_pattern(n):
    for i in range(1, n+1):
        for j in range(i):
            print(j+1, end=" ")
        print()

# Take user input for the number of rows
n = int(input("Enter the number of rows: "))

# Print the pattern
print_pattern(n)
```

Q100. Write a python program to print below pattern.

```
A
B B
C C C
D D D D
E E E E E
```

Sol)

```python
def print_pattern(n):
    for i in range(n):
        letter = chr(ord('A') + i)
        pattern = (letter + ' ') * (i + 1)
        print(pattern)

# Take user input for the number of rows
n = int(input("Enter the number of rows: "))

# Print the pattern
print_pattern(n)
```