Notes Click

Question Bank for ETE(Programming In Python)

1. Identify how python language is more simple other than Java and C. Give proper example.

    - Variable Declaration and Type Inference:
    - Automatic Memory Management:
    - Built-in Data Structures and Libraries:

    Overall, Python's simplicity stems from its clean syntax, dynamic typing, automatic memory management, and the availability of built-in data structures and libraries, which make it easier to write and read code compared to Java and C

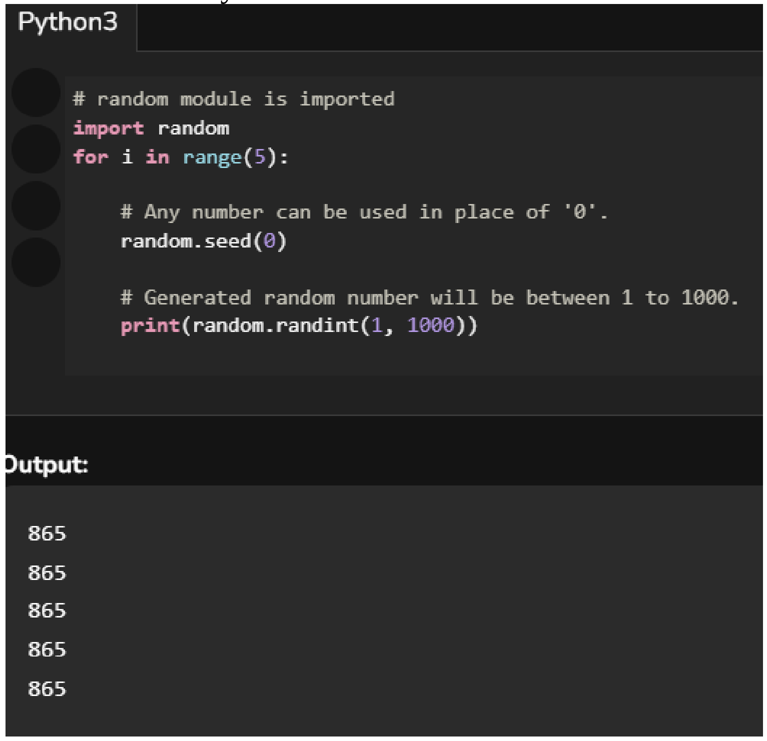
2. Explain the working of random.seed() function.

Syntax : random.seed(l, version)

Parameter :
l : Any seed value used to produce a random number.
version : A integer used to specify how to convert l in a integer.
Returns: A random value.

How Seed Function Works ?

Seed function is used to save the state of a random function, so that it can generate same random numbers on multiple executions of the code on the same machine or on different machines (for a specific seed value). The seed value is the previous value number generated by the generator. For the first time when there is no previous value, it uses current system time.



Python3

```python
# random module is imported
import random
for i in range(5):

    # Any number can be used in place of '0'.
    random.seed(0)

    # Generated random number will be between 1 to 1000.
    print(random.randint(1, 1000))
```

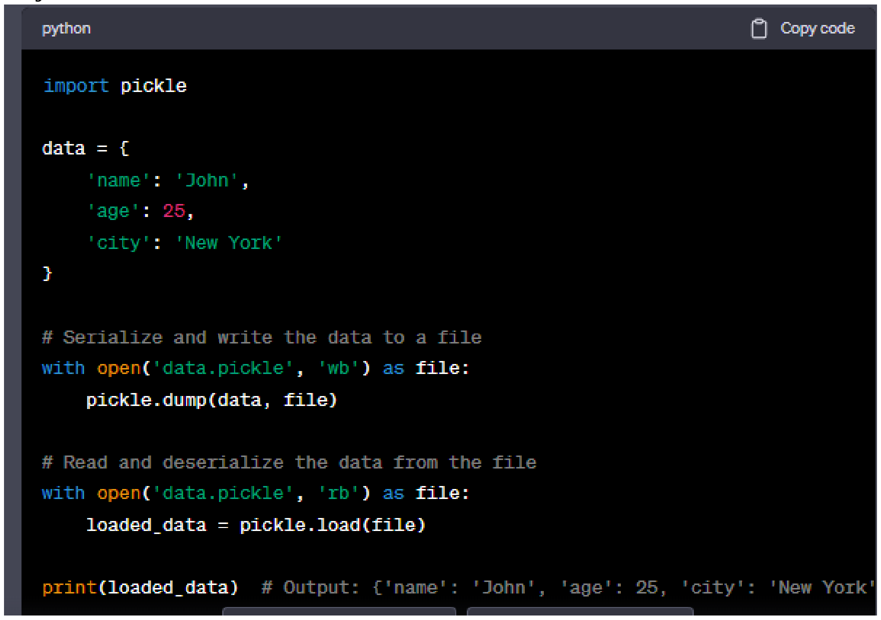
Output:

```
865
865
865
865
865
```

3. (a) Apply a comprehensive explanation of pickle module in Python and list out advantages and
limitations of using pickle to write binary files in Python.
(b) Explain what is range() function and how it is used in lists?

(a) The pickle module in Python provides a way to serialize and deserialize Python objects into a binary representation, allowing you to store them in a file or send them over a network. It is a convenient way to save and load complex data structures without manually converting them to a specific file format.

The pickle module offers two main functions: pickle.dump(obj, file) and pickle.load(file). The pickle.dump() function serializes the object obj and writes it to the file, while pickle.load() reads the serialized object from the file and deserializes it back into a Python object.



```python
import pickle

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

# Serialize and write the data to a file
with open('data.pickle', 'wb') as file:
    pickle.dump(data, file)

# Read and deserialize the data from the file
with open('data.pickle', 'rb') as file:
    loaded_data = pickle.load(file)

print(loaded_data)  # Output: {'name': 'John', 'age': 25, 'city': 'New York'
```

Advantages of using pickle to write binary files in Python:

Simplicity: The pickle module makes it easy to serialize and deserialize complex Python objects with just a few lines of code.
Flexibility: Almost any Python object can be pickled, including custom classes and instances.
Limitations of using pickle:

Security risks: Untrusted pickle files can execute arbitrary code during deserialization, which poses a security risk. It is recommended to only unpickle data from trusted sources.
Version compatibility: Pickle files may not be compatible between different versions of Python.

(b) The range() function in Python is used to generate a sequence of numbers. It returns an immutable sequence of numbers starting from a specified start value (default is 0) and incrementing by a specified step value (default is 1), up to but not including a specified stop value.

The syntax of the range() function is as follows:

```
range(stop)
range(start, stop)
range(start, stop, step)

# Using range() to generate a list of numbers
numbers = list(range(1, 10, 2))
print(numbers)  # Output: [1, 3, 5, 7, 9]
```

4. List the operators that python supports. Explain the relational and logical operators along with their precedence while evaluating an expression.

Python supports several operators, which can be categorized into different types based on their functionality. The operators in Python include arithmetic operators, assignment operators, comparison (relational) operators, logical operators, identity operators, membership operators, and bitwise operators.

Arithmetic Operators:

Addition (+): Adds two operands.
Subtraction (-): Subtracts the second operand from the first.
Multiplication (*): Multiplies two operands.
Division (/): Divides the first operand by the second operand (result is always a float).
Floor Division (//): Divides the first operand by the second operand, discarding the remainder (result is an integer).
Modulus (%): Returns the remainder of the division of the first operand by the second.
Exponentiation (**): Raises the first operand to the power of the second operand.

Assignment Operators:

Assignment (=): Assigns the value on the right side to the variable on the left side.
Compound Assignment (e.g., +=, -=, *=): Performs an operation and assigns the result to the variable.

Comparison (Relational) Operators:

Equal to (==): Returns True if the operands are equal.
Not equal to (!=): Returns True if the operands are not equal.
Greater than (>): Returns True if the left operand is greater than the right operand.
Less than (<): Returns True if the left operand is less than the right operand.
Greater than or equal to (>=): Returns True if the left operand is greater than or equal to the right operand.
Less than or equal to (<=): Returns True if the left operand is less than or equal to the right operand.

Logical Operators:

Logical AND (and): Returns True if both operands are True.
Logical OR (or): Returns True if either of the operands is True.
Logical NOT (not): Returns the opposite Boolean value of the operand.

When evaluating an expression involving both relational and logical operators, the precedence rules determine the order of evaluation. The precedence order for these operators is as follows (from highest to lowest):

Parentheses: Expressions within parentheses are evaluated first.
Logical NOT: Evaluates the logical NOT operation.
Multiplicative and Exponentiation operators: *, /, //, %, and ** are evaluated from left to right.
Additive operators: + and - are evaluated from left to right.
Relational operators: ==, !=, >, <, >=, and <= are evaluated from left to right.
Logical AND: Evaluates the logical AND operation from left to right.
Logical OR: Evaluates the logical OR operation from left to right.

5. (a) Show the value of L after you run the code below?

```
 L = ["life", "answer", 42, 0]
for thing in L:
if thing == 0:
L[thing] = "universe"

  elif thing == 42:

L[1] = "everything"
```

[‘universe’, ‘everything’, 42, 0]

(b) Show the value of L3 after you execute all the operations in the code below?

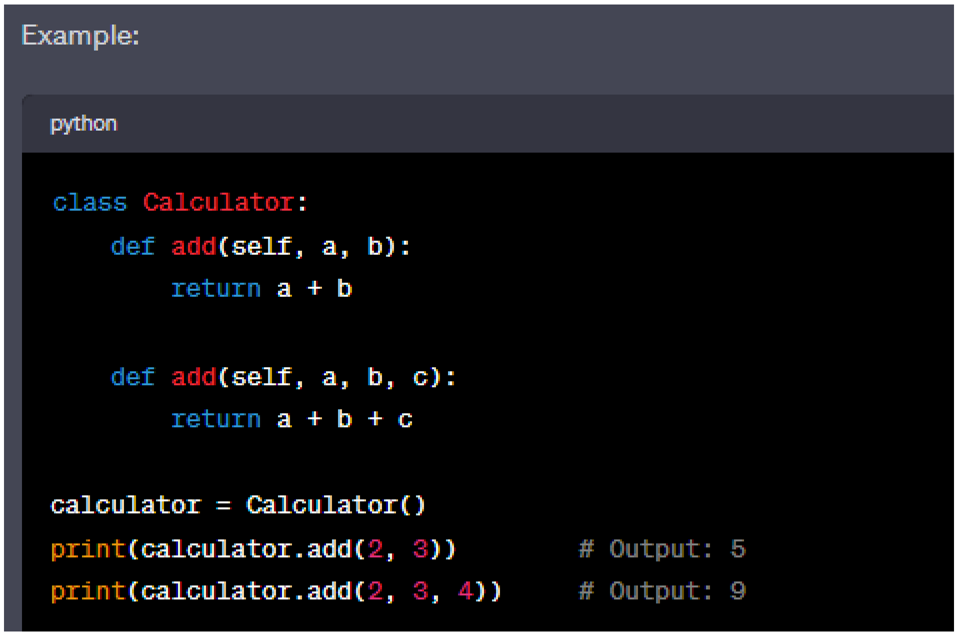
```
 L1 = ['re']
 L2 = ['mi']
 L3  = ['do']
 L4 = L1 + L2
 L3.extend(L4)
 L3.sort()
del(L3[0])
L3.append(['fa','la'])
```

['mi', 're', ['fa', 'la']]

6. Categorize and discuss the types of Polymorphism in details with proper example.

Polymorphism is a concept in object-oriented programming that allows objects of different classes to be treated as objects of a common superclass. It provides a way to perform a single action in different ways, depending on the type of object it is being performed upon. In Python, there are two main types of polymorphism: static (compile-time) polymorphism and dynamic (runtime) polymorphism.

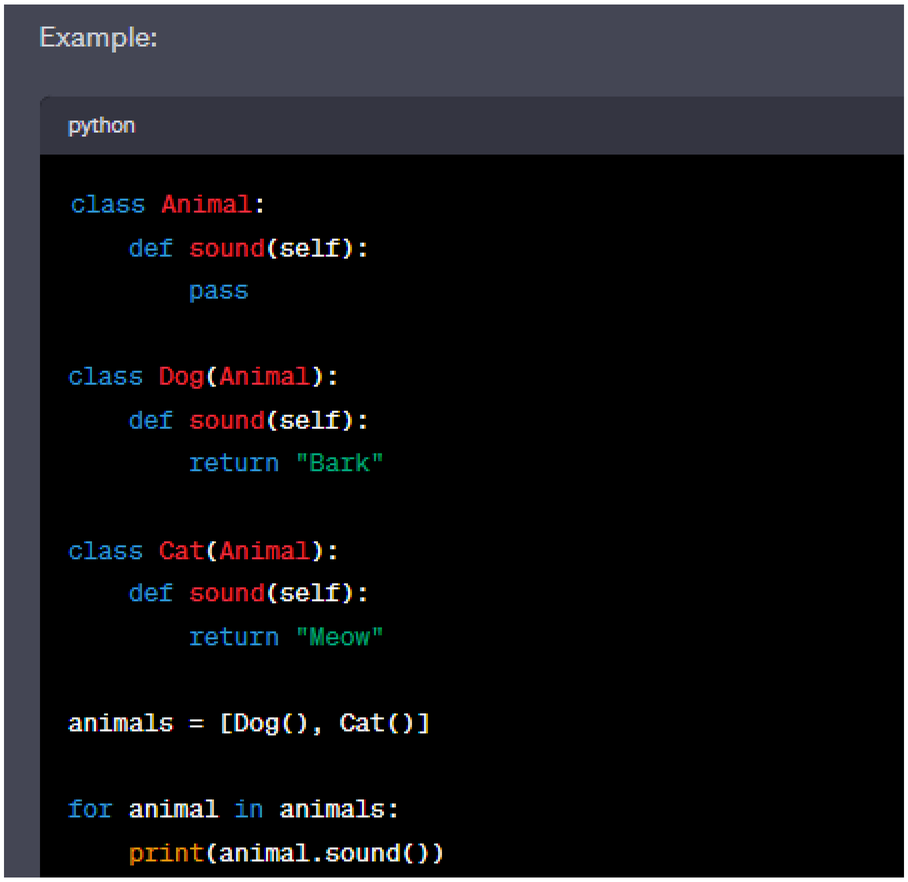
Static (Compile-Time) Polymorphism:
Static polymorphism is achieved through method overloading and operator overloading. In method overloading, multiple methods with the same name but different parameters are defined in a class. The appropriate method to be called is determined at compile-time based on the arguments provided.


Example:

python

```python
class Calculator:
    def add(self, a, b):
        return a + b

    def add(self, a, b, c):
        return a + b + c

calculator = Calculator()
print(calculator.add(2, 3))          # Output: 5
print(calculator.add(2, 3, 4))       # Output: 9
```

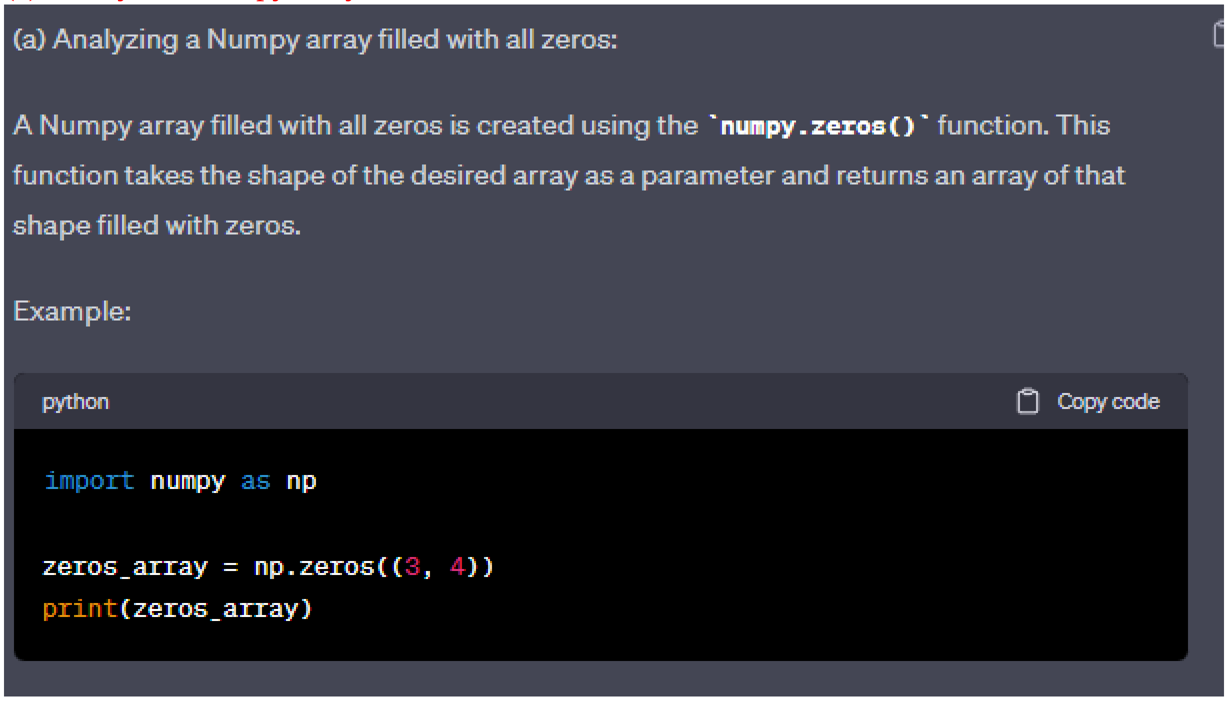
Dynamic (Runtime) Polymorphism:
Dynamic polymorphism is achieved through method overriding and inheritance. Method overriding occurs when a subclass defines a method with the same name and signature as a method in its superclass. The subclass method is then called instead of the superclass method when the method is invoked on an object of the subclass.


Example:

python

```python
class Animal:
    def sound(self):
        pass

class Dog(Animal):
    def sound(self):
        return "Bark"

class Cat(Animal):
    def sound(self):
        return "Meow"

animals = [Dog(), Cat()]

for animal in animals:
    print(animal.sound())
```

7. (a) Analyze a Numpy array filled with all zeros.



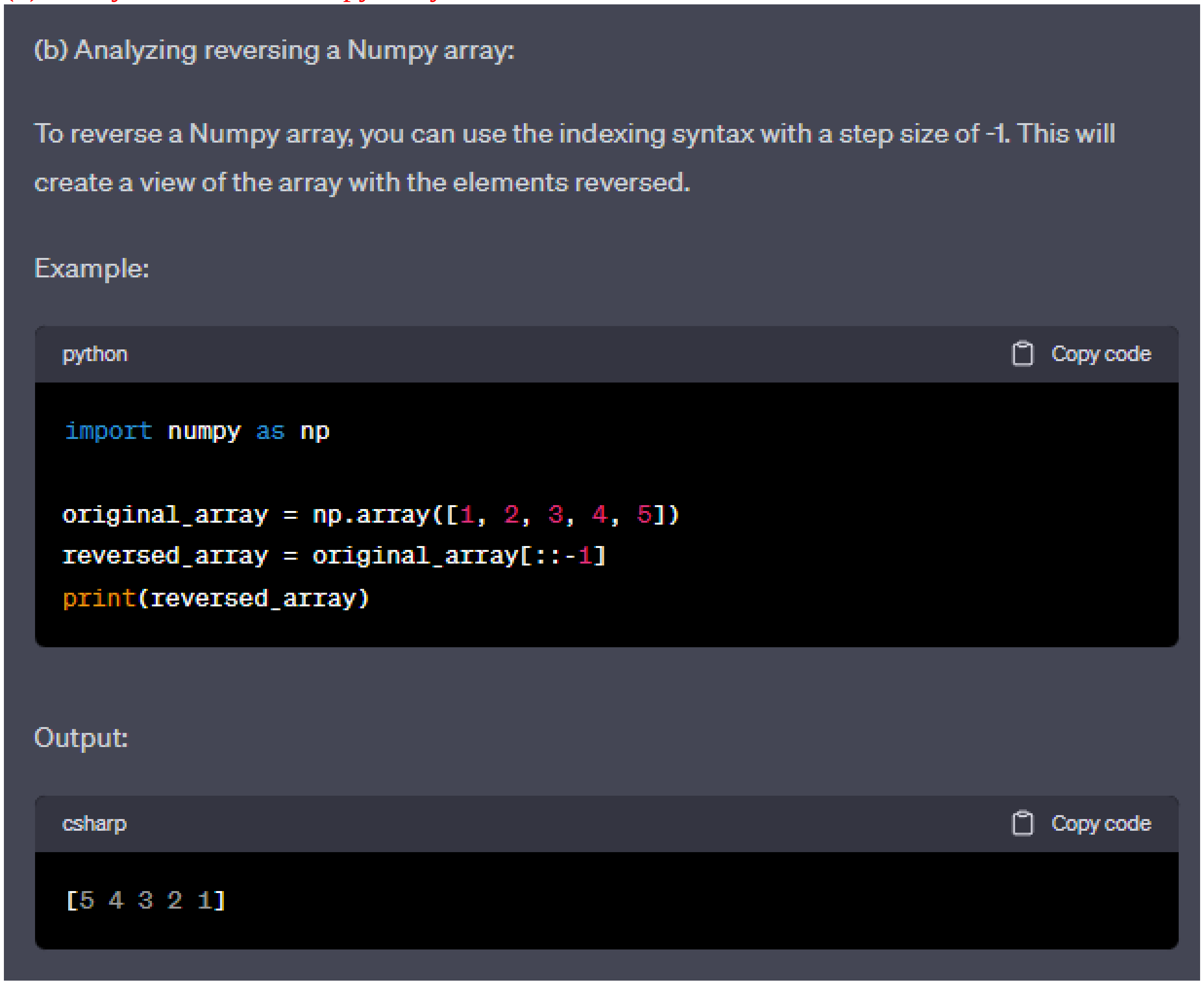
(a) Analyzing a Numpy array filled with all zeros:

A Numpy array filled with all zeros is created using the `numpy.zeros()` function. This function takes the shape of the desired array as a parameter and returns an array of that shape filled with zeros.

Example:

```python
import numpy as np

zeros_array = np.zeros((3, 4))
print(zeros_array)
```

(b) Analyze reverse a Numpy array.



(b) Analyzing reversing a Numpy array:

To reverse a Numpy array, you can use the indexing syntax with a step size of -1. This will create a view of the array with the elements reversed.

Example:

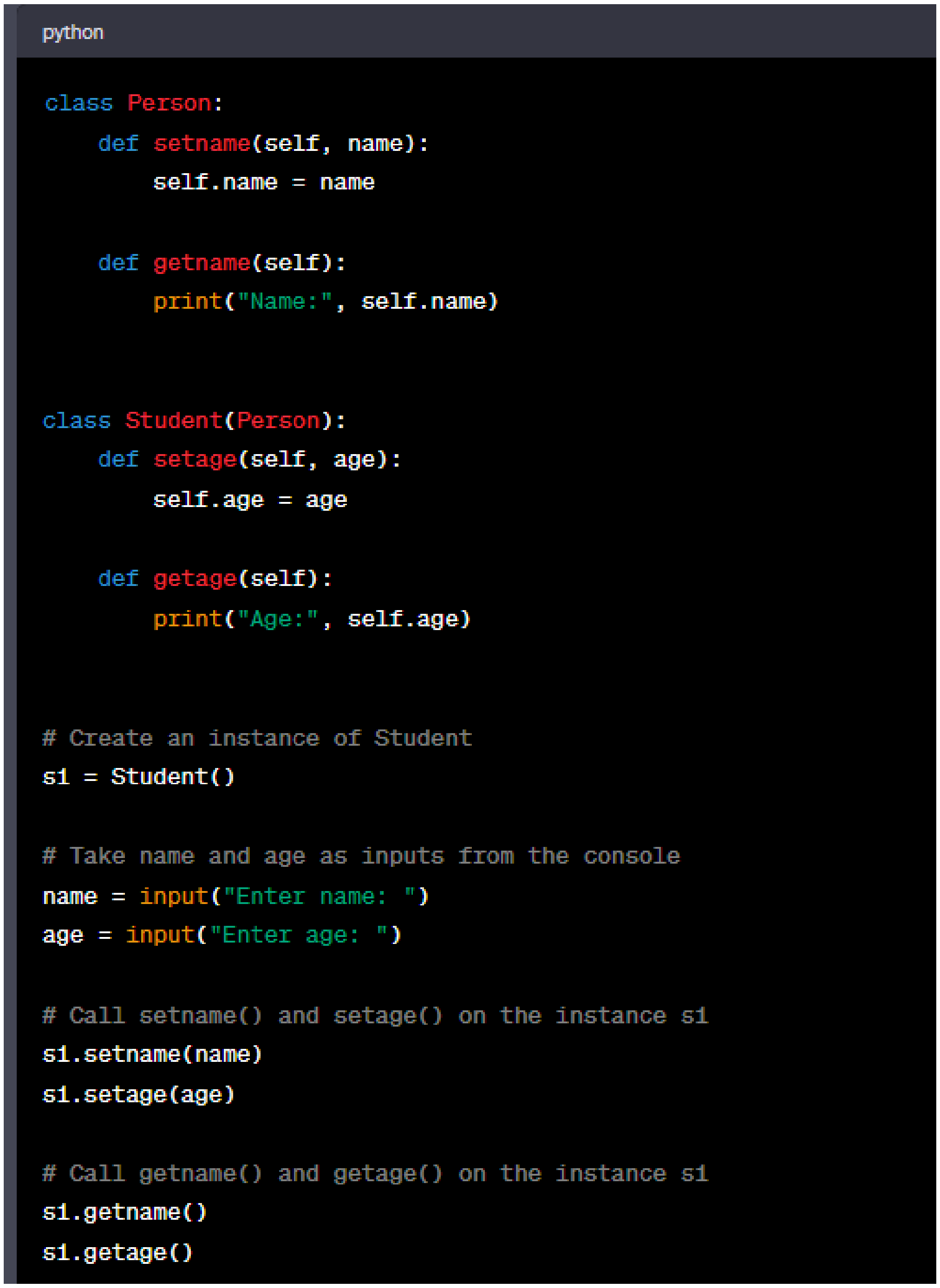
```python
import numpy as np

original_array = np.array([1, 2, 3, 4, 5])
reversed_array = original_array[::-1]
print(reversed_array)
```

Output:

```csharp
[5 4 3 2 1]
```

8. Assuming the instructions given below, write a simple program using a class()

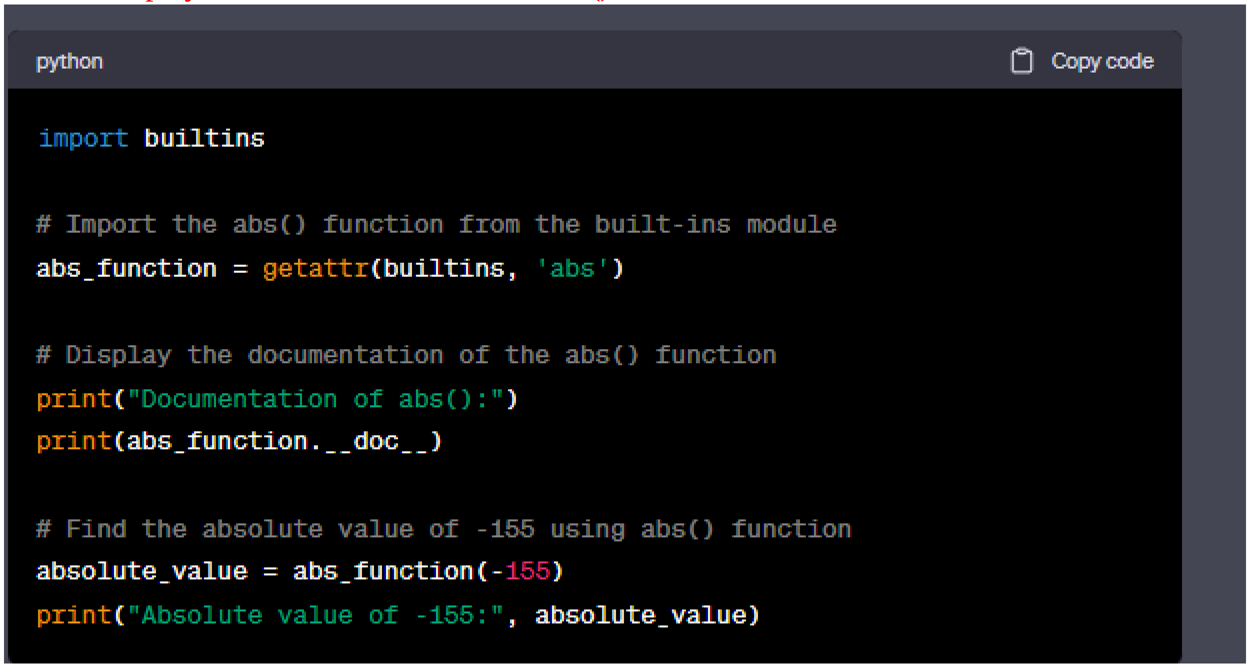
- A base class Person and a derived class Student with Person as its base class.
- Add two methods **setname()** (which takes the parameter self and name)and **getname()** which prints the name in the base class.
- Add two methods in the derived class: **setage()** (which takes the parameters self and age) which sets the age and **getage()** which prints the age.
- Create an instance of **Student** and name it as **s1**.
- Take **name** and **age** as inputs from the console.
- Call the **setname()** and **setage()** on this instance by passing the **name** and **age** parameters.
- Call the **getname()** and **getage()** on this class, which prints the passed parameters



python

```python
class Person:
    def setname(self, name):
        self.name = name

    def getname(self):
        print("Name:", self.name)


class Student(Person):
    def setage(self, age):
        self.age = age

    def getage(self):
        print("Age:", self.age)


# Create an instance of Student
s1 = Student()

# Take name and age as inputs from the console
name = input("Enter name: ")
age = input("Enter age: ")

# Call setname() and setage() on the instance s1
s1.setname(name)
s1.setage(age)

# Call getname() and getage() on the instance s1
s1.getname()
s1.getage()
```

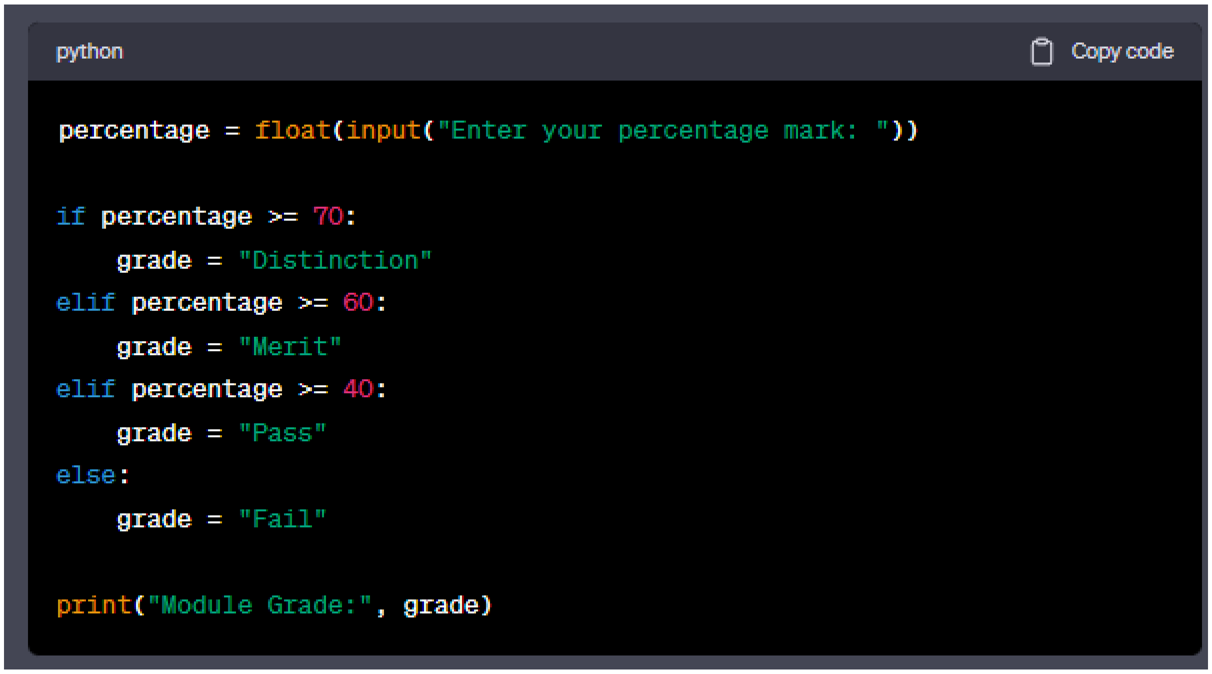
9. Apply a Python program that imports the abs() function using the built-ins module, displays the documentation of the abs() function and finds the absolute value of -155.



```python
import builtins

# Import the abs() function from the built-ins module
abs_function = getattr(builtins, 'abs')

# Display the documentation of the abs() function
print("Documentation of abs():")
print(abs_function.__doc__)

# Find the absolute value of -155 using abs() function
absolute_value = abs_function(-155)
print("Absolute value of -155:", absolute_value)
```

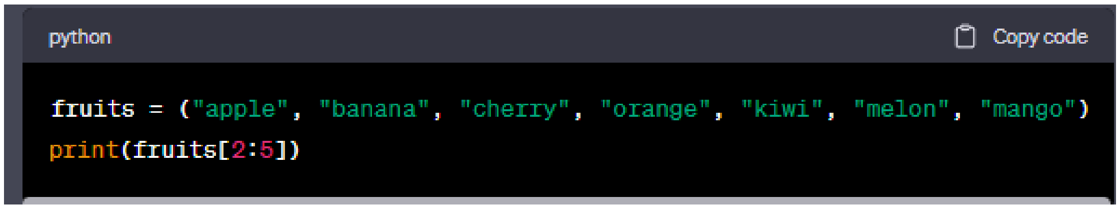
10. Develop a python program that asks users to enter their percentage mark for a module of study. The program prints the module grade as either distinction, merit, pass or fail depending on the percentage mark entered.

I. A mark of 70% and above is awarded a distinction.
II. A mark in the range of 60% through to 69% is awarded a merit.
III. A mark in the range of 40% through to 59% is awarded a pass.
IV. Marks less than 40% are awarded a fail



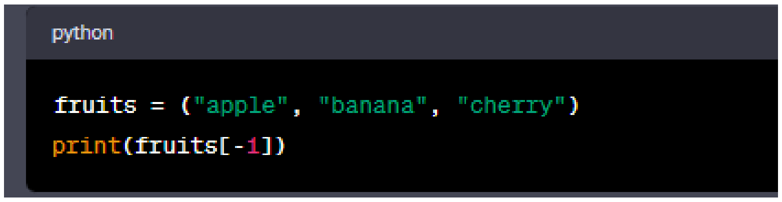
```python
percentage = float(input("Enter your percentage mark: "))

if percentage >= 70:
    grade = "Distinction"
elif percentage >= 60:
    grade = "Merit"
elif percentage >= 40:
    grade = "Pass"
else:
    grade = "Fail"

print("Module Grade:", grade)
```

11. (a) Assume fruits = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango") write the print statement using a range of indexes to print the third, fourth, and fifth item in the tuple.



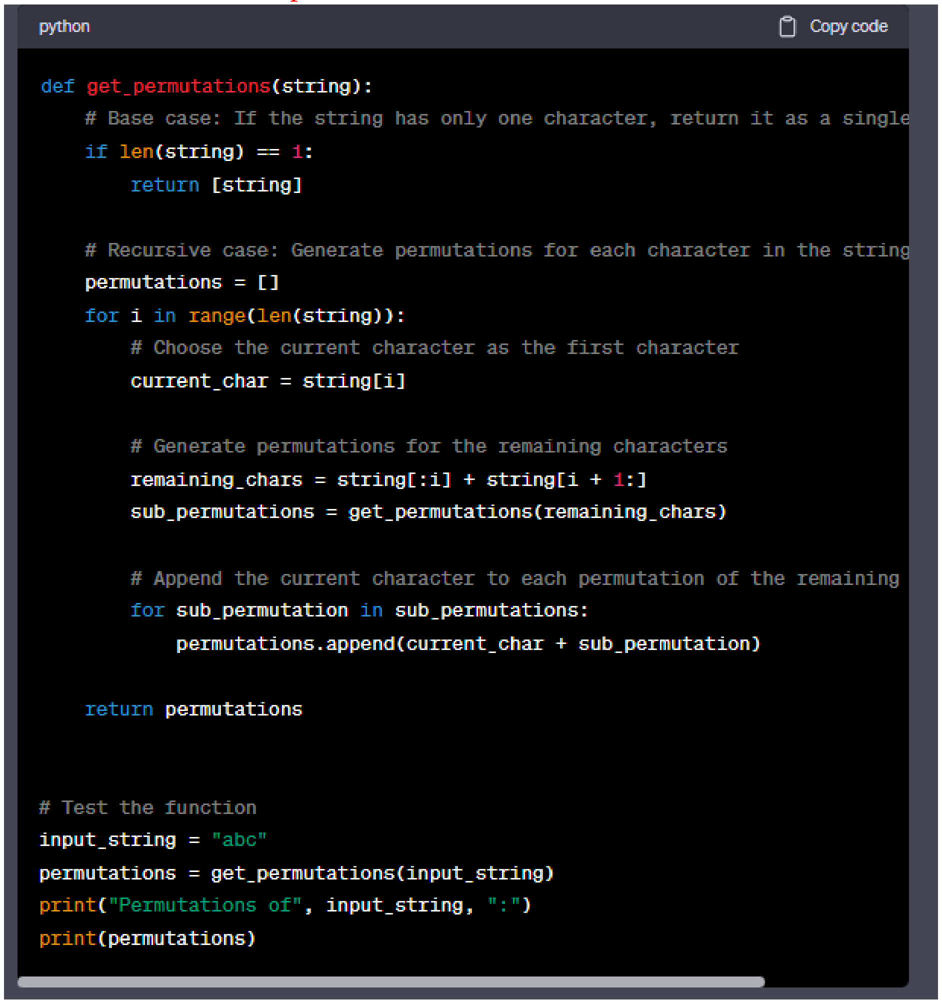
```python
fruits = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(fruits[2:5])
```

(b) Assume fruits = ("apple", "banana", "cherry") write the python code using negative indexing to print the last item in the tuple.

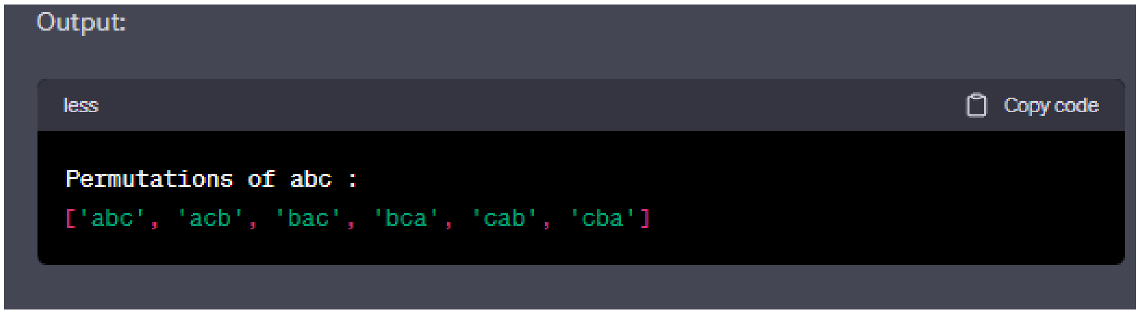


```python
fruits = ("apple", "banana", "cherry")
print(fruits[-1])
```

12. A permutation is simply a name for a reordering. So the permutations of the string „abc" are „abc", „acb", „bac", „bca", „cab", and „cba". Note that a sequence is a permutation of itself (the trivial permutation). Take the permutation of the string in the list and write python program for a recursive function (get_permutations) that takes a string and returns a list of all its permutations.



```python
def get_permutations(string):
    # Base case: If the string has only one character, return it as a single
    if len(string) == 1:
        return [string]

    # Recursive case: Generate permutations for each character in the string
    permutations = []
    for i in range(len(string)):
        # Choose the current character as the first character
        current_char = string[i]

        # Generate permutations for the remaining characters
        remaining_chars = string[:i] + string[i + 1:]
        sub_permutations = get_permutations(remaining_chars)

        # Append the current character to each permutation of the remaining
        for sub_permutation in sub_permutations:
            permutations.append(current_char + sub_permutation)

    return permutations


# Test the function
input_string = "abc"
permutations = get_permutations(input_string)
print("Permutations of", input_string, ":")
print(permutations)
```



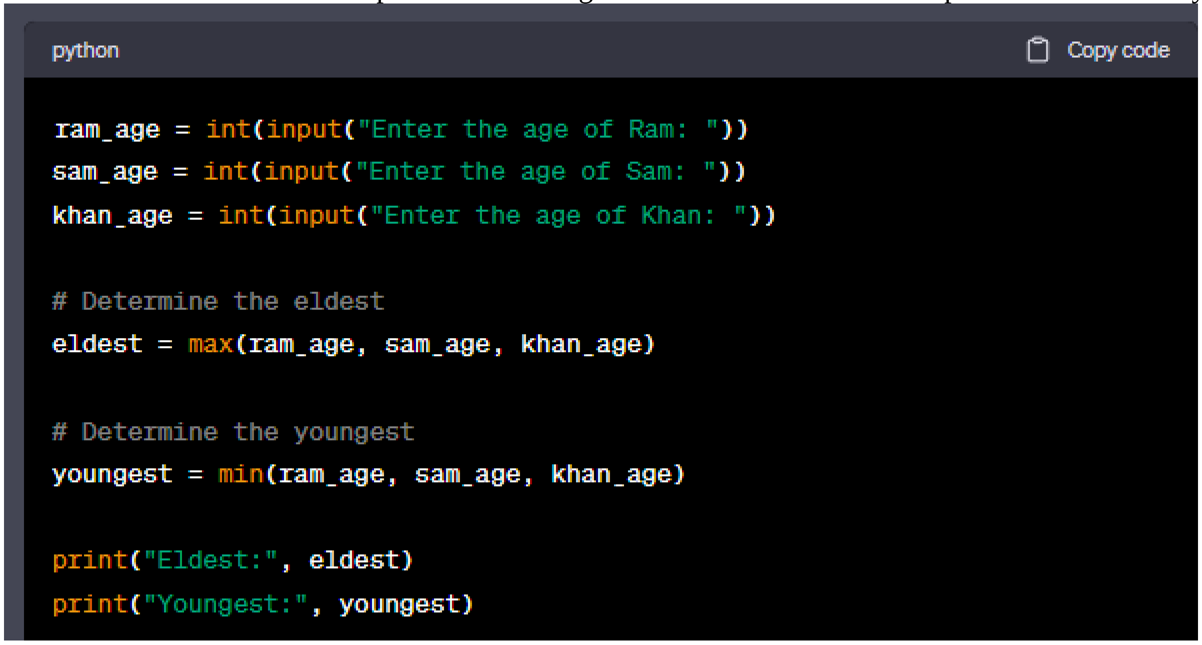
```
Permutations of abc :
['abc', 'acb', 'bac', 'bca', 'cab', 'cba']
```

13. Explain different rules about to define an identifier in python. If the age of Ram, Sam, and Khan are input through the keyboard, write a python program to determine the eldest and youngest of the three.

In Python, identifiers are used to name variables, functions, classes, modules, and other entities. Here are the rules for defining identifiers in Python:

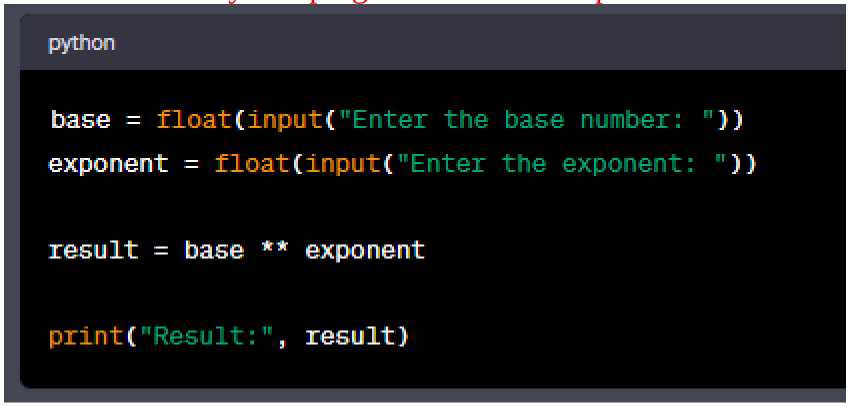
- The first character of an identifier must be a letter (a-z, A-Z) or an underscore (_). It cannot start with a digit.
- The rest of the identifier can contain letters, digits (0-9), and underscores.
- Identifiers are case-sensitive, meaning myVar and myvar are considered different identifiers.
- Python keywords cannot be used as identifiers. Keywords are reserved words that have special meanings in Python, such as if, else, for, while, class, etc.
- Identifiers should not contain any special characters like !, @, #, $, %, etc.

It is recommended to use descriptive and meaningful names for identifiers to improve code readability.



```python
ram_age = int(input("Enter the age of Ram: "))
sam_age = int(input("Enter the age of Sam: "))
khan_age = int(input("Enter the age of Khan: "))

# Determine the eldest
eldest = max(ram_age, sam_age, khan_age)

# Determine the youngest
youngest = min(ram_age, sam_age, khan_age)

print("Eldest:", eldest)
print("Youngest:", youngest)
```

14. Write a Python program to find the exponentiation of a number.

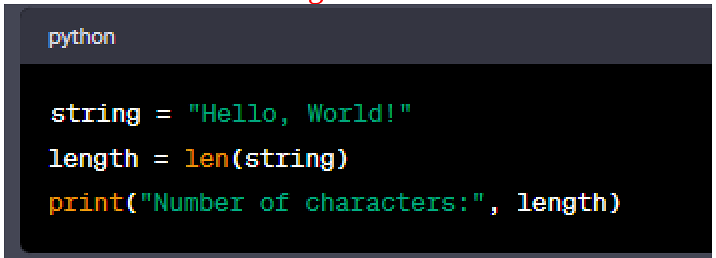


```python
base = float(input("Enter the base number: "))
exponent = float(input("Enter the exponent: "))

result = base ** exponent

print("Result:", result)
```

15. Identify the string method used to implement the following.

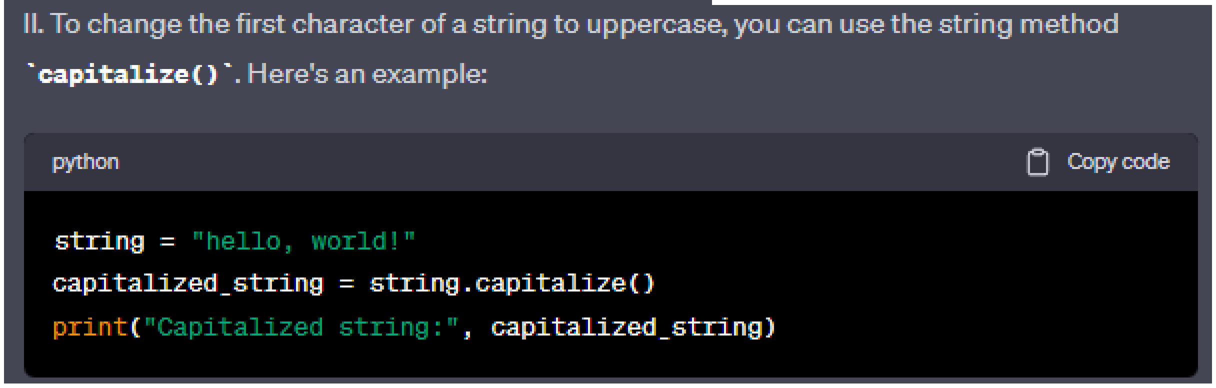
I. To count the number of characters in the string.

II. To change the first character of the string in capital letter.



python

```python
string = "Hello, World!"
length = len(string)
print("Number of characters:", length)
```



II. To change the first character of a string to uppercase, you can use the string method `capitalize()`. Here's an example:

```python
string = "hello, world!"
capitalized_string = string.capitalize()
print("Capitalized string:", capitalized_string)
```

16. Define SciPy, Scrapy, Scikit-learn, PyGame, PyTorch, PyBrain and Keras.

SciPy: SciPy is a scientific computing library for Python. It provides functionality for mathematical and scientific computations, including numerical integration, optimization, linear algebra, signal and image processing, statistics, and more. It is built on top of NumPy, another popular numerical computing library in Python.

Scrapy: Scrapy is a powerful web scraping framework used for extracting data from websites. It provides a high-level API and tools to facilitate the crawling and scraping of web pages. Scrapy allows you to define how to navigate websites, extract data from HTML/XML responses, and save the extracted data.

Scikit-learn: Scikit-learn is a machine learning library in Python that provides a wide range of algorithms and tools for tasks such as classification, regression, clustering, dimensionality reduction, and model selection. It is built on top of NumPy, SciPy, and Matplotlib and offers a simple and consistent interface for applying machine learning techniques.

PyGame: PyGame is a cross-platform set of Python modules specifically designed for game development. It provides functionality for handling graphics, sound, input, and other aspects of game development. PyGame is widely used for creating 2D games and multimedia applications.

PyTorch: PyTorch is a popular open-source deep learning framework developed by Facebook's AI research lab. It provides a dynamic computational graph-based approach to building and training neural networks. PyTorch offers extensive support for GPU acceleration and is widely used for research and development in the field of deep learning.

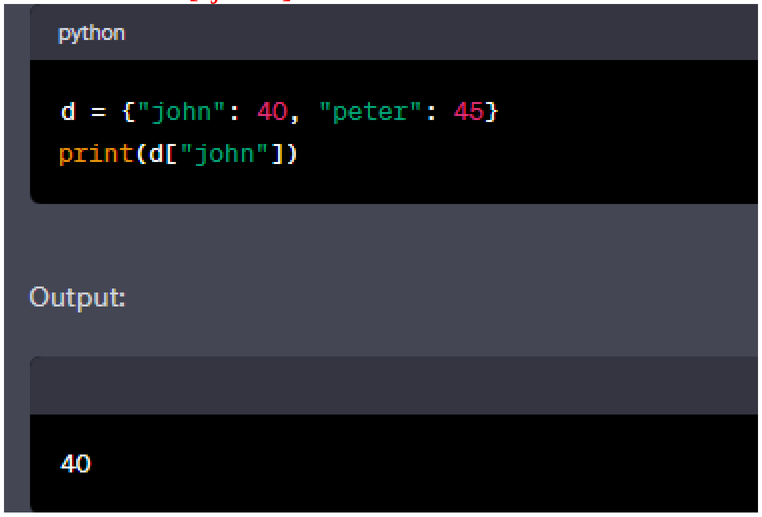
PyBrain: PyBrain is a modular machine learning library that focuses on neural networks and reinforcement learning algorithms. It provides tools for building, training, and evaluating various types of neural networks, including feedforward and recurrent networks. PyBrain also offers reinforcement learning algorithms for tasks such as Q-learning and policy gradients.

Keras: Keras is a high-level neural networks library written in Python. It provides a user-friendly API for building and training deep learning models. Keras supports multiple backends, including TensorFlow and Theano, allowing users to take advantage of their computational capabilities. Keras simplifies the process of defining and training neural networks, making it accessible to both beginners and experienced practitioners.

17. (a) Show the output of the following Python code?

d = {"john":40, "peter":45}

d["john"]



python

```python
d = {"john": 40, "peter": 45}
print(d["john"])
```

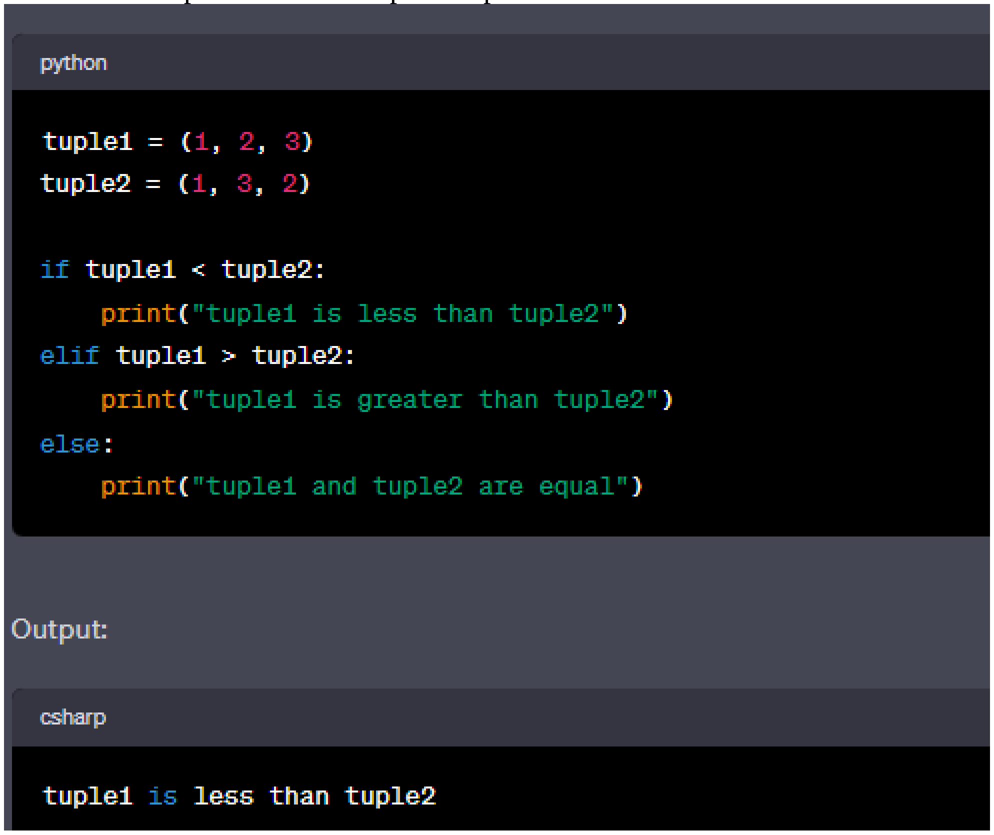
Output:

```
40
```

b) Is tuple comparison possible? Explain how with example.

(b) Tuple comparison is possible in Python. Tuples are compared lexicographically, meaning they are compared element by element in a pair-wise manner from left to right.

Here's an example to illustrate tuple comparison:



python

```python
tuple1 = (1, 2, 3)
tuple2 = (1, 3, 2)

if tuple1 < tuple2:
    print("tuple1 is less than tuple2")
elif tuple1 > tuple2:
    print("tuple1 is greater than tuple2")
else:
    print("tuple1 and tuple2 are equal")
```

Output:

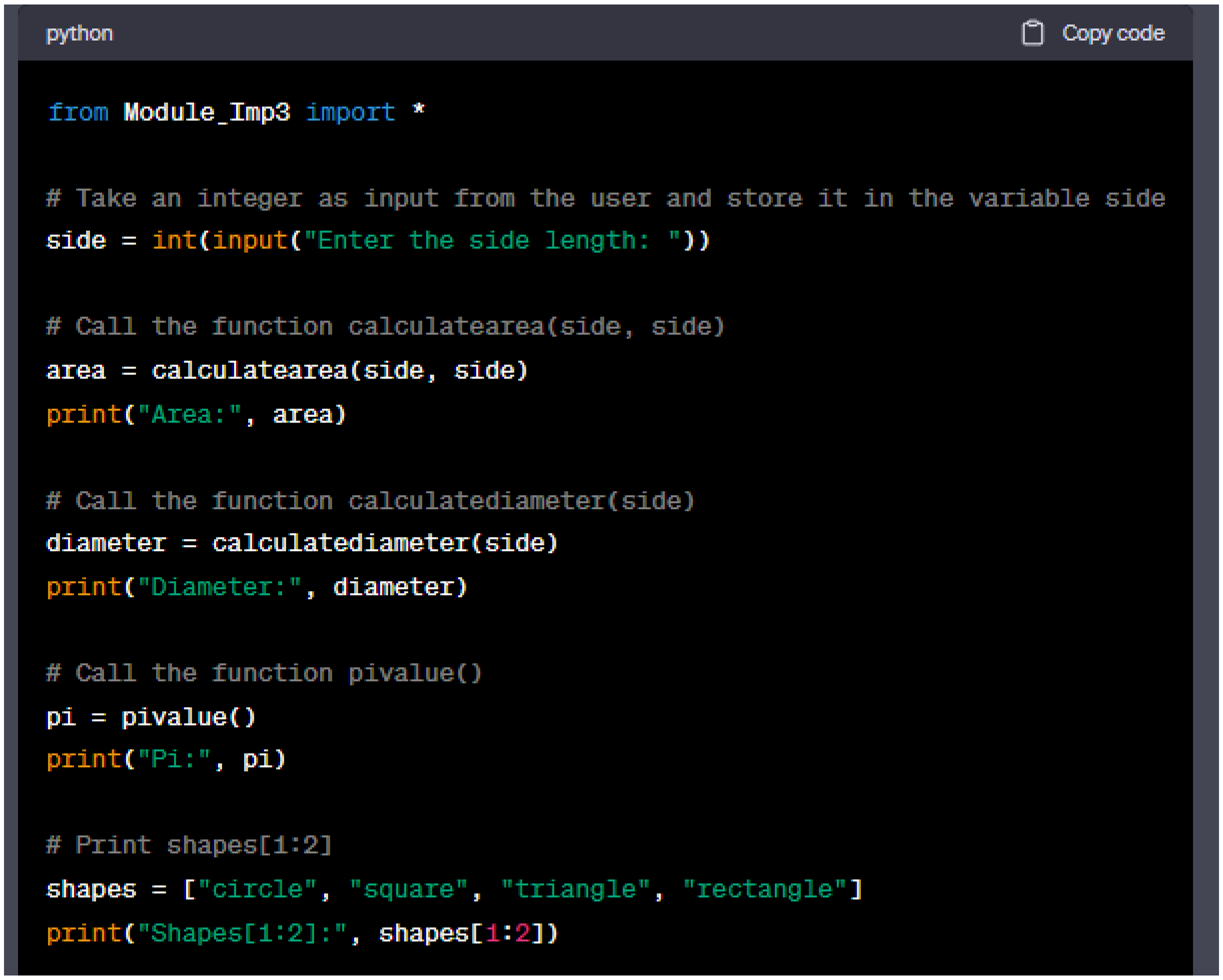
csharp

```
tuple1 is less than tuple2
```

18. Assume the given instructions while writing the program

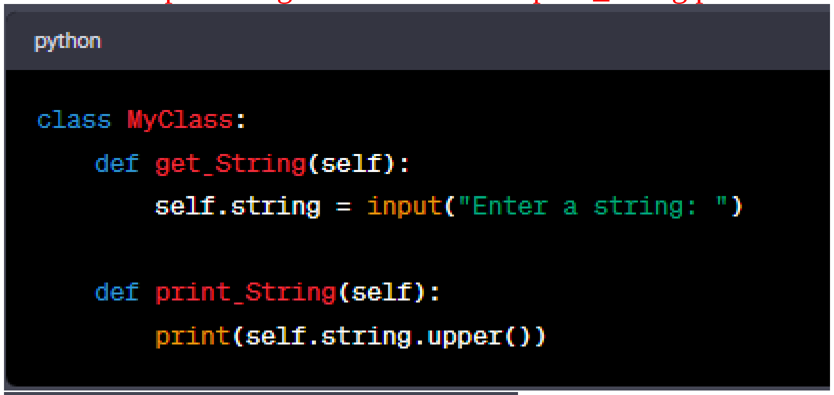
- Use the **Module_Imp3** which contains functions that can be imported.
- Use from Module_Imp3 import *
- Take an integer as **input** from user and store it in the variable **side.**
- Call the function calculatearea(side,side)
- Call the function calculatediameter(side)

- Call the function pivalue()
- print shapes[1:2]



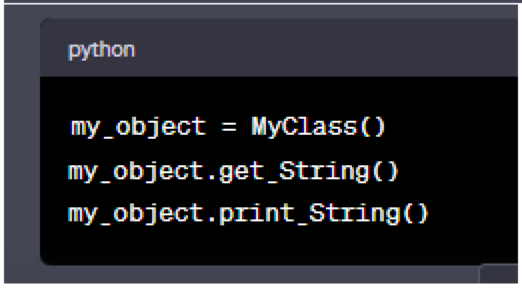
```python
from Module_Imp3 import *

# Take an integer as input from the user and store it in the variable side
side = int(input("Enter the side length: "))

# Call the function calculatearea(side, side)
area = calculatearea(side, side)
print("Area:", area)

# Call the function calculatediameter(side)
diameter = calculatediameter(side)
print("Diameter:", diameter)

# Call the function pivalue()
pi = pivalue()
print("Pi:", pi)

# Print shapes[1:2]
shapes = ["circle", "square", "triangle", "rectangle"]
print("Shapes[1:2]:", shapes[1:2])
```

19. Analyze a Python class that has two methods: get_String and print_String , get_String accept a string from the user and print_String prints the string in upper case.



python

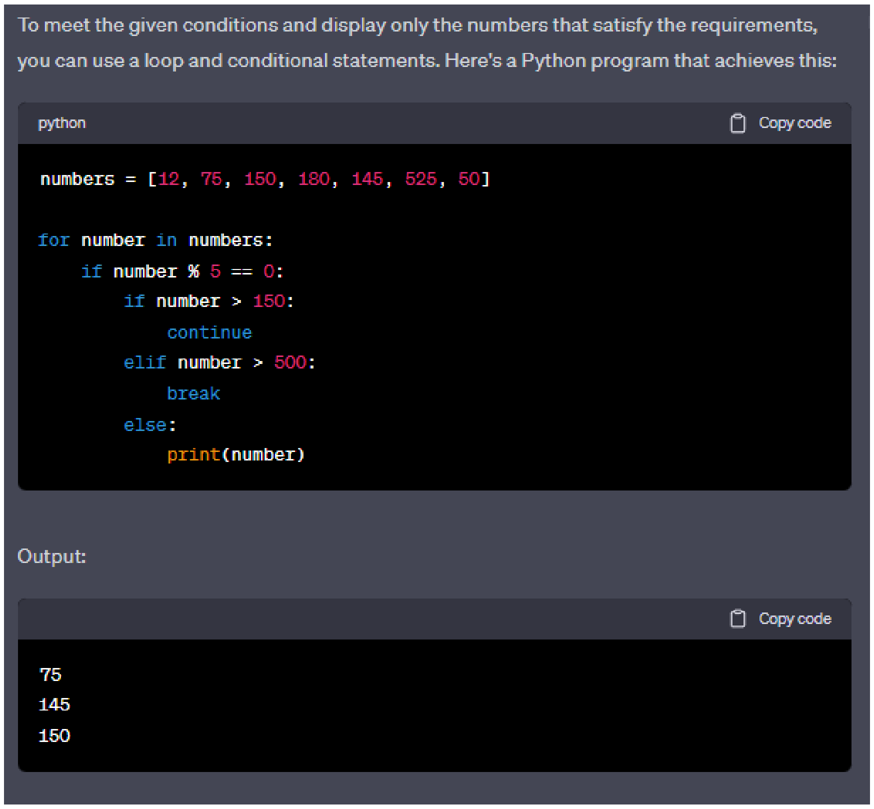
```python
class MyClass:
    def get_String(self):
        self.string = input("Enter a string: ")

    def print_String(self):
        print(self.string.upper())
```



python

```python
my_object = MyClass()
my_object.get_String()
my_object.print_String()
```

20. List below are the following conditions to write a program to display only those numbers
    (a) The number must be divisible by five
    (b) If the number is greater than 150, then skip it and move to the next number
    (c) If the number is greater than 500, then stop the loop
    (d) Input: numbers = [12, 75, 150, 180, 145, 525, 50]
    (e) Output:
        75
        145
        150



To meet the given conditions and display only the numbers that satisfy the requirements, you can use a loop and conditional statements. Here's a Python program that achieves this:

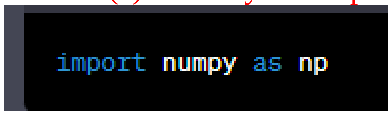
python

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

for number in numbers:
    if number % 5 == 0:
        if number > 150:
            continue
        elif number > 500:
            break
        else:
            print(number)
```

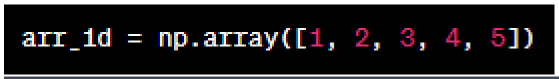
Output:

```
75
145
150
```

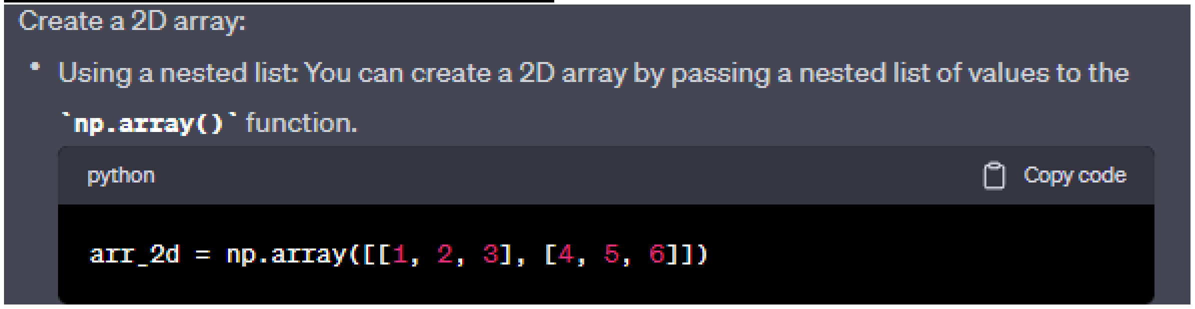
21. (a) Identify the steps to create a 1D array and 2D array.



```python
import numpy as np
```



```python
arr_1d = np.array([1, 2, 3, 4, 5])
```

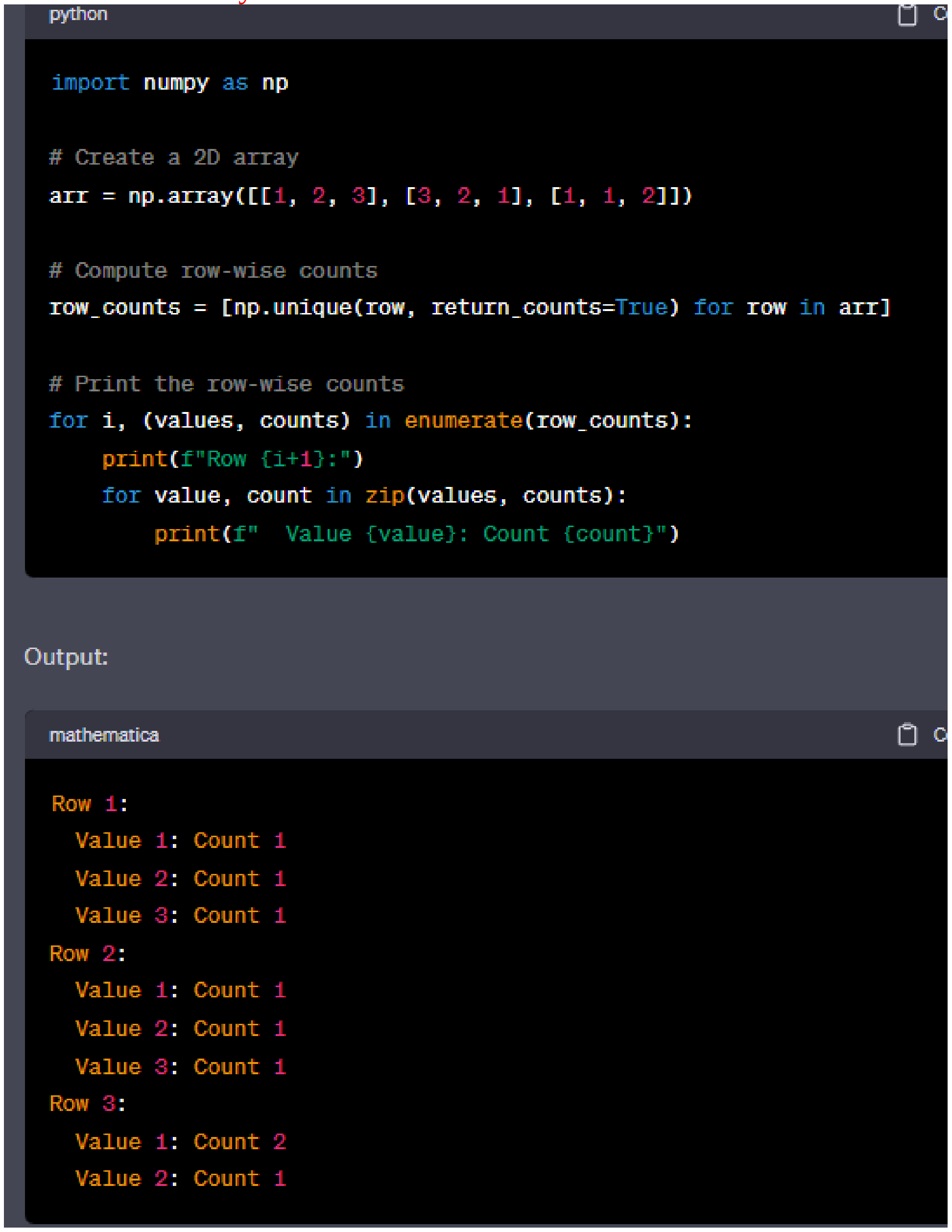


Create a 2D array:

- Using a nested list: You can create a 2D array by passing a nested list of values to the `np.array()` function.

```python
arr_2d = np.array([[1, 2, 3], [4, 5, 6]])
```

(b) Apply Python programming to compute the row wise counts of all possible values in an array.

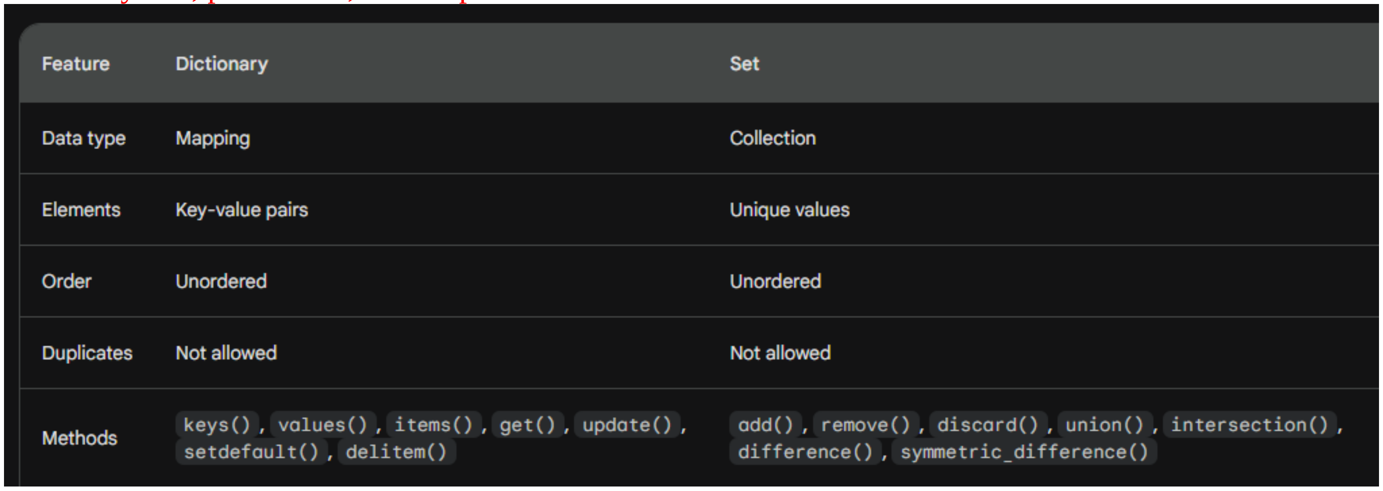


```python
import numpy as np

# Create a 2D array
arr = np.array([[1, 2, 3], [3, 2, 1], [1, 1, 2]])

# Compute row-wise counts
row_counts = [np.unique(row, return_counts=True) for row in arr]

# Print the row-wise counts
for i, (values, counts) in enumerate(row_counts):
    print(f"Row {i+1}:")
    for value, count in zip(values, counts):
        print(f"  Value {value}: Count {count}")
```

Output:

```
Row 1:
  Value 1: Count 1
  Value 2: Count 1
  Value 3: Count 1
Row 2:
  Value 1: Count 1
  Value 2: Count 1
  Value 3: Count 1
Row 3:
  Value 1: Count 2
  Value 2: Count 1
```

22. (a) Compare the differences between a python dictionary and a python set, including the syntax, parameters, and output of each.



Feature	Dictionary	Set
Data type	Mapping	Collection
Elements	Key-value pairs	Unique values
Order	Unordered	Unordered
Duplicates	Not allowed	Not allowed
Methods	`keys()`, `values()`, `items()`, `get()`, `update()`, `setdefault()`, `delitem()`	`add()`, `remove()`, `discard()`, `union()`, `intersection()`, `difference()`, `symmetric_difference()`

Syntax

Dictionaries are created using curly braces, while sets are created using the `set()` function. For example:

Code snippet
```
my_dict = {"key1": "value1", "key2": "value2"}
my_set = set([1, 2, 3, 4])
```
content_copy

Parameters

Dictionaries take key-value pairs as parameters, while sets take unique values as parameters. For example:

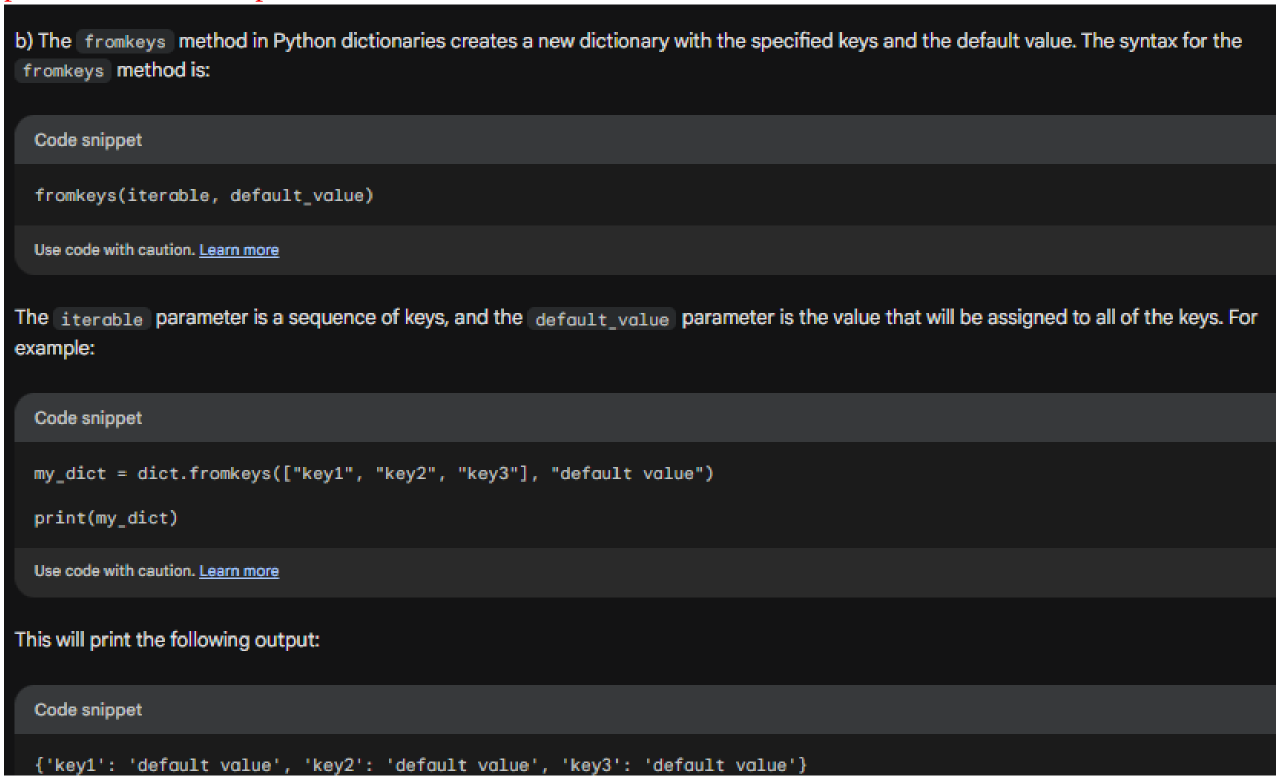
Code snippet
```
my_dict = {"key1": "value1", "key2": "value2"}
my_set = set([1, 2, 3, 4])
```

Output

Dictionaries and sets are both iterable, so they can be printed out using a for loop. For example:

Code snippet
```
for key, value in my_dict.items():
    print(key, value)

for item in my_set:
    print(item)
```

(b) Analyze the "fromkeys" method in python dictionaries, including its syntax, parameters, and output.



b) The `fromkeys` method in Python dictionaries creates a new dictionary with the specified keys and the default value. The syntax for the `fromkeys` method is:

Code snippet

```
fromkeys(iterable, default_value)
```

Use code with caution. Learn more

The `iterable` parameter is a sequence of keys, and the `default_value` parameter is the value that will be assigned to all of the keys. For example:

Code snippet

```
my_dict = dict.fromkeys(["key1", "key2", "key3"], "default value")

print(my_dict)
```

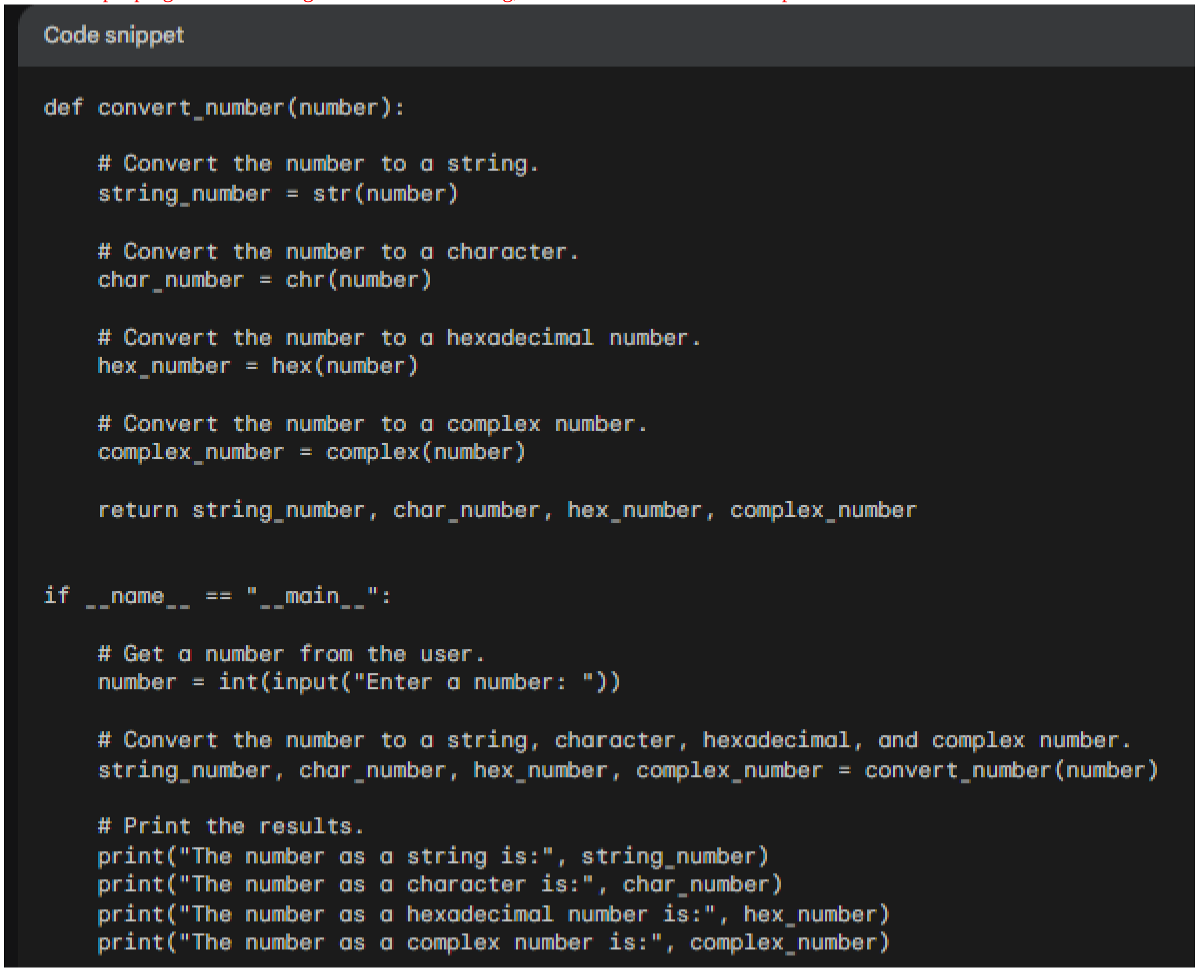
Use code with caution. Learn more

This will print the following output:

Code snippet

```
{'key1': 'default value', 'key2': 'default value', 'key3': 'default value'}
```

(c) Write a simple program to convert given number into string, char and hexadecimaland complex number.

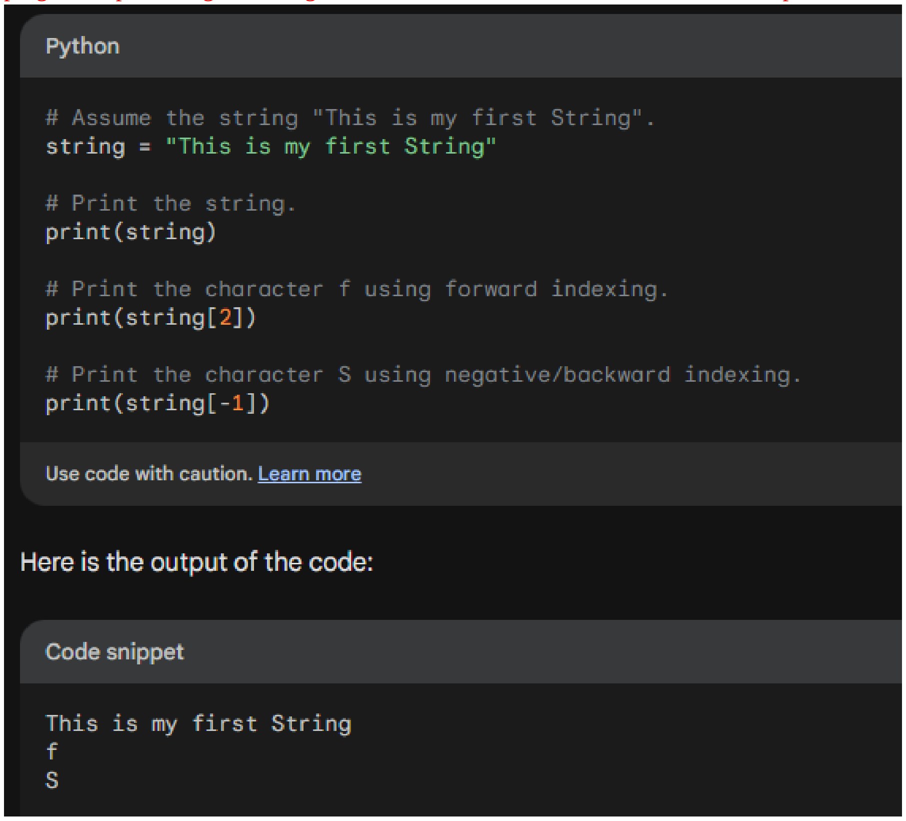


Code snippet

```
def convert_number(number):

    # Convert the number to a string.
    string_number = str(number)

    # Convert the number to a character.
    char_number = chr(number)

    # Convert the number to a hexadecimal number.
    hex_number = hex(number)

    # Convert the number to a complex number.
    complex_number = complex(number)

    return string_number, char_number, hex_number, complex_number


if __name__ == "__main__":

    # Get a number from the user.
    number = int(input("Enter a number: "))

    # Convert the number to a string, character, hexadecimal, and complex number.
    string_number, char_number, hex_number, complex_number = convert_number(number)

    # Print the results.
    print("The number as a string is:", string_number)
    print("The number as a character is:", char_number)
    print("The number as a hexadecimal number is:", hex_number)
    print("The number as a complex number is:", complex_number)
```

23. (a) Assume the string "This is my first String". Write a program to print the following:print the string, print the character f using forward indexing, and print the character S using negative/backward indexing.

(b) Test for two inputs from the user using input() function, one is string str and another one is integer n. Write a program to print the given string str n times. Print the resultas shown in the example.



Python

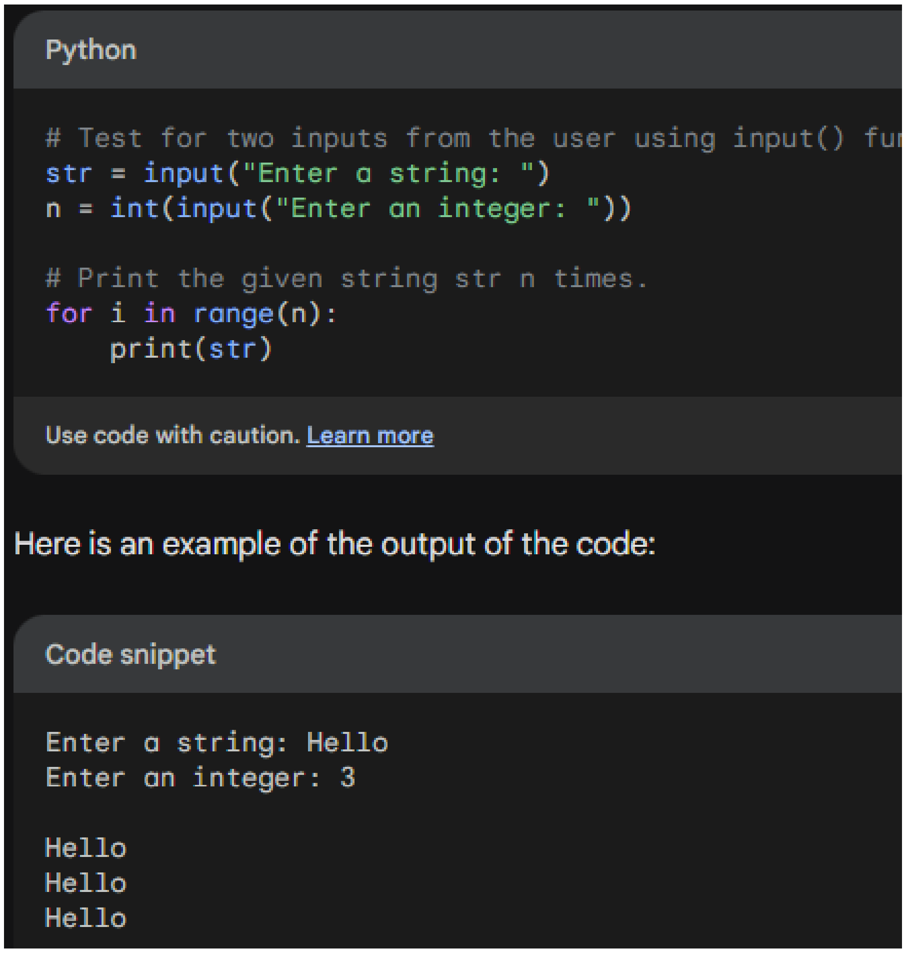
```python
# Assume the string "This is my first String".
string = "This is my first String"

# Print the string.
print(string)

# Print the character f using forward indexing.
print(string[2])

# Print the character S using negative/backward indexing.
print(string[-1])
```

Use code with caution. Learn more

Here is the output of the code:

Code snippet

```
This is my first String
f
S
```



Python

```python
# Test for two inputs from the user using input() fu
str = input("Enter a string: ")
n = int(input("Enter an integer: "))

# Print the given string str n times.
for i in range(n):
    print(str)
```

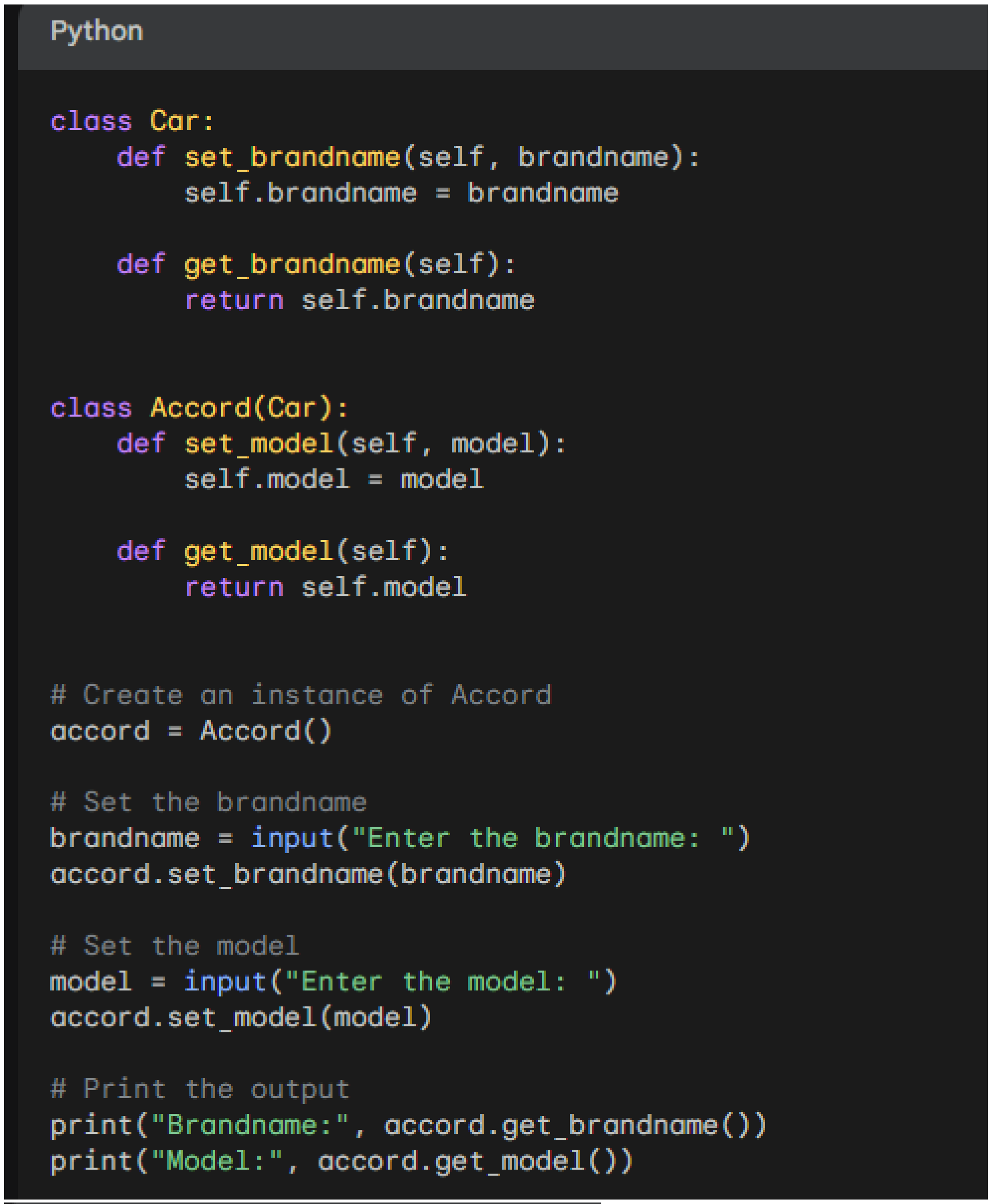
Use code with caution. Learn more

Here is an example of the output of the code:

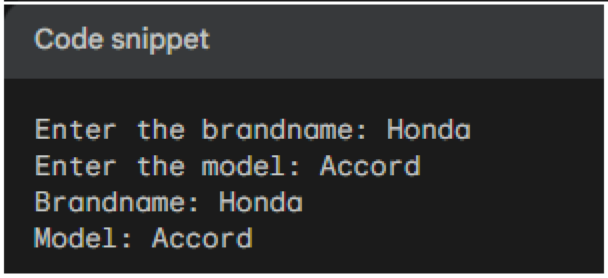
Code snippet

```
Enter a string: Hello
Enter an integer: 3

Hello
Hello
Hello
```

24. Assume the given below instructions to define a base class Car and a derived class Accord.
    a) Car class has two methods which sets and gets the method brandname.
    b) Accord is a derived class of the base class Car, which has two methods which sets and gets the brandname.
    c) Accord also has the model set and get methods.
    d) Now create an instance of Accord and set the brandname to the user given input.
    e) Set the input model for the same instance.
    f) Now print the output by calling the methods getbrandname() and getmodel() on Accord instance

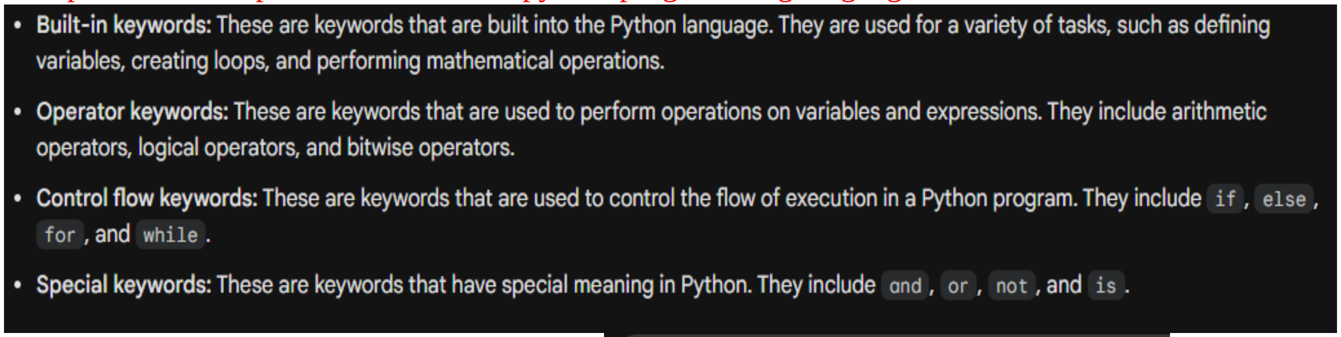


```python
class Car:
    def set_brandname(self, brandname):
        self.brandname = brandname

    def get_brandname(self):
        return self.brandname


class Accord(Car):
    def set_model(self, model):
        self.model = model

    def get_model(self):
        return self.model


# Create an instance of Accord
accord = Accord()

# Set the brandname
brandname = input("Enter the brandname: ")
accord.set_brandname(brandname)

# Set the model
model = input("Enter the model: ")
accord.set_model(model)

# Print the output
print("Brandname:", accord.get_brandname())
print("Model:", accord.get_model())
```

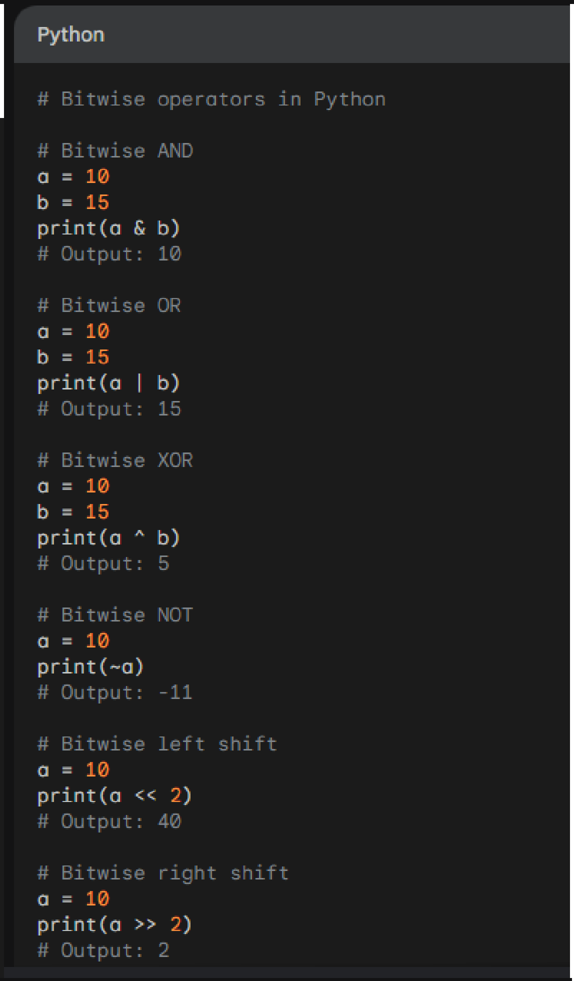


```
Enter the brandname: Honda
Enter the model: Accord
Brandname: Honda
Model: Accord
```

25. Recall and discuss different kind of keywords in python. Write a source code in python to explain bitwise operators available in python programming language.

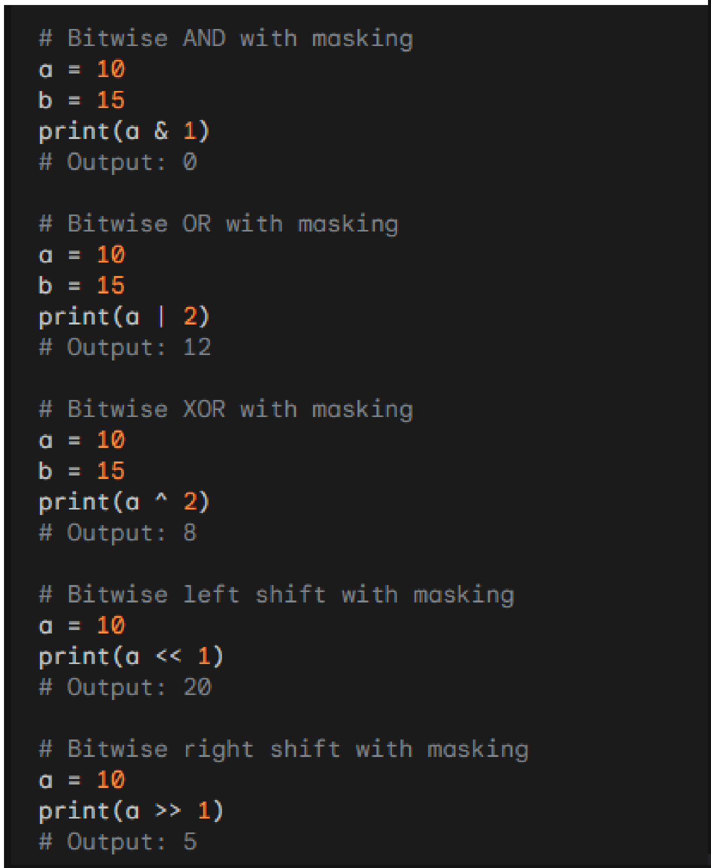


- **Built-in keywords:** These are keywords that are built into the Python language. They are used for a variety of tasks, such as defining variables, creating loops, and performing mathematical operations.
- **Operator keywords:** These are keywords that are used to perform operations on variables and expressions. They include arithmetic operators, logical operators, and bitwise operators.
- **Control flow keywords:** These are keywords that are used to control the flow of execution in a Python program. They include `if`, `else`, `for`, and `while`.
- **Special keywords:** These are keywords that have special meaning in Python. They include `and`, `or`, `not`, and `is`.

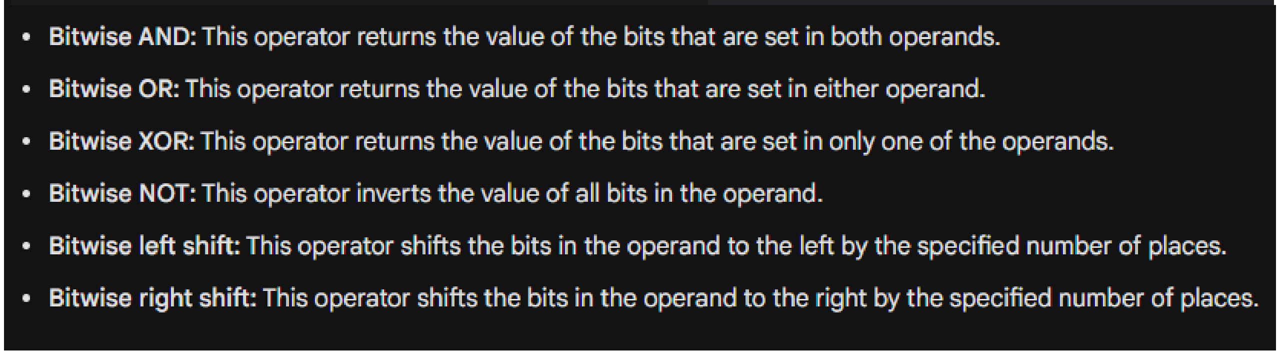


Python

```python
# Bitwise operators in Python

# Bitwise AND
a = 10
b = 15
print(a & b)
# Output: 10

# Bitwise OR
a = 10
b = 15
print(a | b)
# Output: 15

# Bitwise XOR
a = 10
b = 15
print(a ^ b)
# Output: 5

# Bitwise NOT
a = 10
print(~a)
# Output: -11

# Bitwise left shift
a = 10
print(a << 2)
# Output: 40

# Bitwise right shift
a = 10
print(a >> 2)
# Output: 2
```

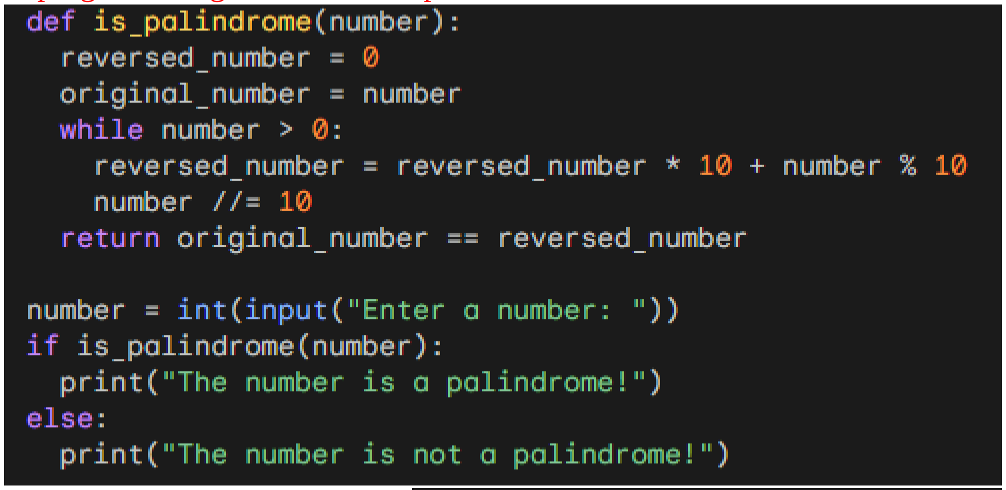


```python
# Bitwise AND with masking
a = 10
b = 15
print(a & 1)
# Output: 0

# Bitwise OR with masking
a = 10
b = 15
print(a | 2)
# Output: 12

# Bitwise XOR with masking
a = 10
b = 15
print(a ^ 2)
# Output: 8

# Bitwise left shift with masking
a = 10
print(a << 1)
# Output: 20

# Bitwise right shift with masking
a = 10
print(a >> 1)
# Output: 5
```

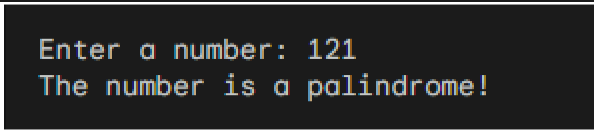


- **Bitwise AND:** This operator returns the value of the bits that are set in both operands.
- **Bitwise OR:** This operator returns the value of the bits that are set in either operand.
- **Bitwise XOR:** This operator returns the value of the bits that are set in only one of the operands.
- **Bitwise NOT:** This operator inverts the value of all bits in the operand.
- **Bitwise left shift:** This operator shifts the bits in the operand to the left by the specified number of places.
- **Bitwise right shift:** This operator shifts the bits in the operand to the right by the specified number of places.

26. Write a Python program that given number is palindrome or not.

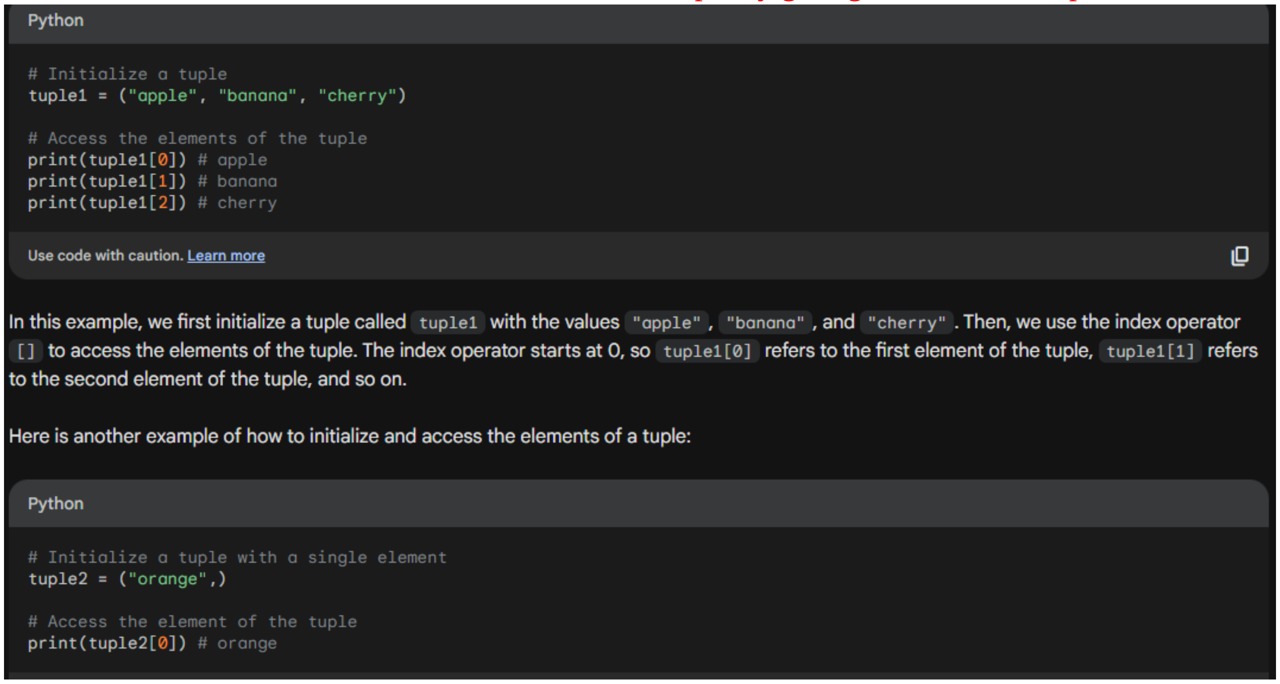


```python
def is_palindrome(number):
  reversed_number = 0
  original_number = number
  while number > 0:
    reversed_number = reversed_number * 10 + number % 10
    number //= 10
  return original_number == reversed_number

number = int(input("Enter a number: "))
if is_palindrome(number):
  print("The number is a palindrome!")
else:
  print("The number is not a palindrome!")
```



```
Enter a number: 121
The number is a palindrome!
```

27. Show how to initialize and access the elements of a tuple by giving a suitable example.



Python

```python
# Initialize a tuple
tuple1 = ("apple", "banana", "cherry")

# Access the elements of the tuple
print(tuple1[0]) # apple
print(tuple1[1]) # banana
print(tuple1[2]) # cherry
```

Use code with caution. Learn more

In this example, we first initialize a tuple called `tuple1` with the values `"apple"`, `"banana"`, and `"cherry"`. Then, we use the index operator `[]` to access the elements of the tuple. The index operator starts at 0, so `tuple1[0]` refers to the first element of the tuple, `tuple1[1]` refers to the second element of the tuple, and so on.

Here is another example of how to initialize and access the elements of a tuple:

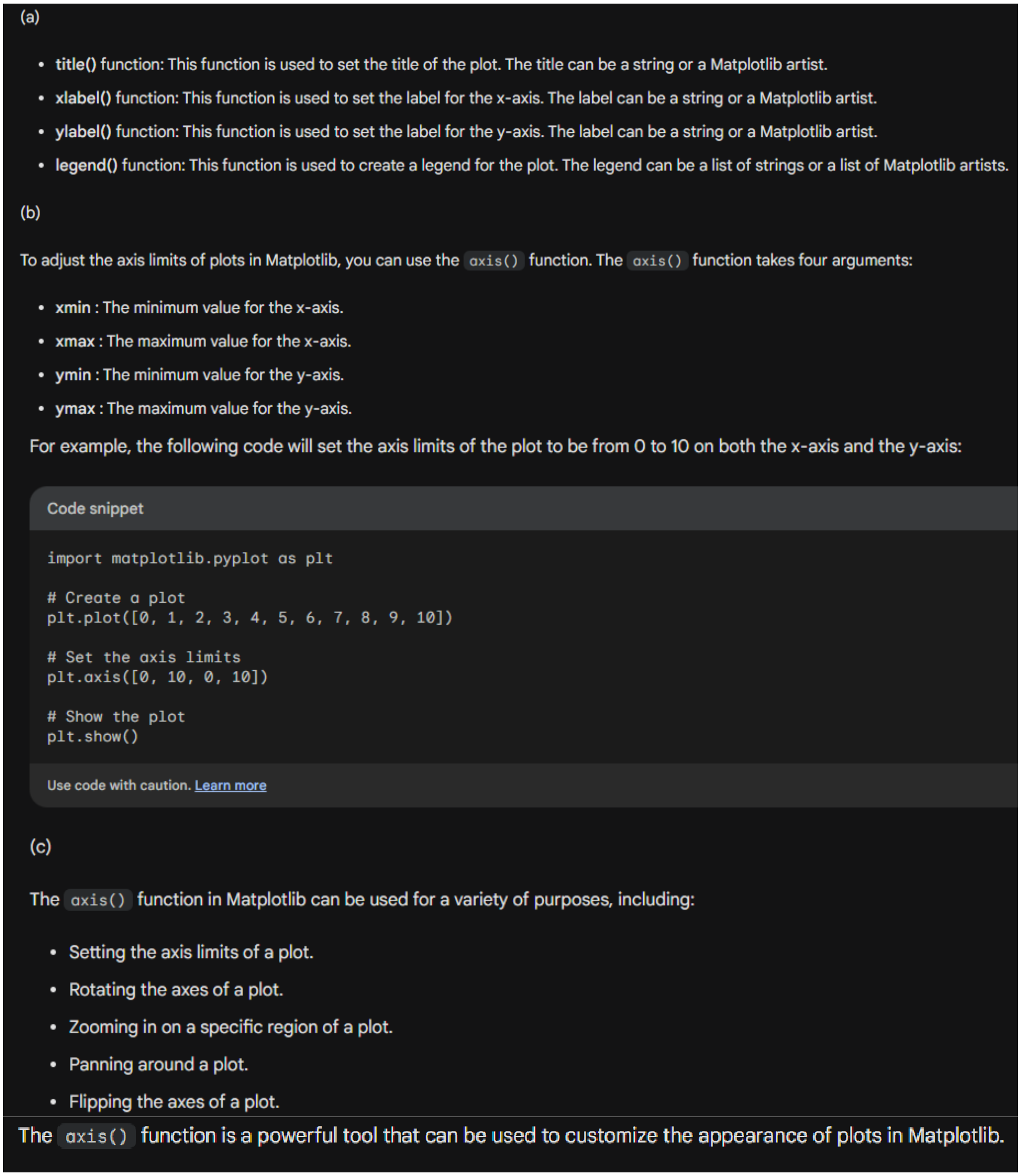
Python

```python
# Initialize a tuple with a single element
tuple2 = ("orange",)

# Access the element of the tuple
print(tuple2[0]) # orange
```

28. (a)Define the functions in Matplotlib: (i) title() (ii) xlabel() (iii) ylabel() (iv) legend()

(b) How can you adjust the axis limits of plots in matplotlib?

(c) What are the various uses of the axis() function in matplotlib?


(a)

- **title()** function: This function is used to set the title of the plot. The title can be a string or a Matplotlib artist.
- **xlabel()** function: This function is used to set the label for the x-axis. The label can be a string or a Matplotlib artist.
- **ylabel()** function: This function is used to set the label for the y-axis. The label can be a string or a Matplotlib artist.
- **legend()** function: This function is used to create a legend for the plot. The legend can be a list of strings or a list of Matplotlib artists.

(b)

To adjust the axis limits of plots in Matplotlib, you can use the `axis()` function. The `axis()` function takes four arguments:

- **xmin** : The minimum value for the x-axis.
- **xmax** : The maximum value for the x-axis.
- **ymin** : The minimum value for the y-axis.
- **ymax** : The maximum value for the y-axis.

For example, the following code will set the axis limits of the plot to be from 0 to 10 on both the x-axis and the y-axis:

Code snippet

```
import matplotlib.pyplot as plt

# Create a plot
plt.plot([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

# Set the axis limits
plt.axis([0, 10, 0, 10])

# Show the plot
plt.show()
```

Use code with caution. Learn more

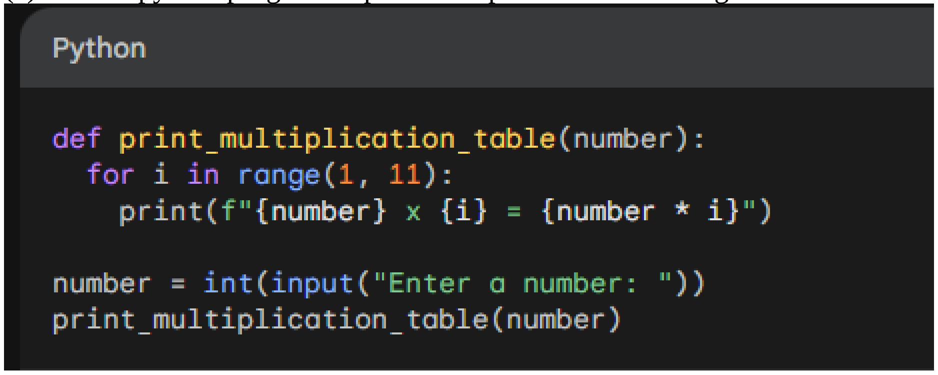
(c)

The `axis()` function in Matplotlib can be used for a variety of purposes, including:

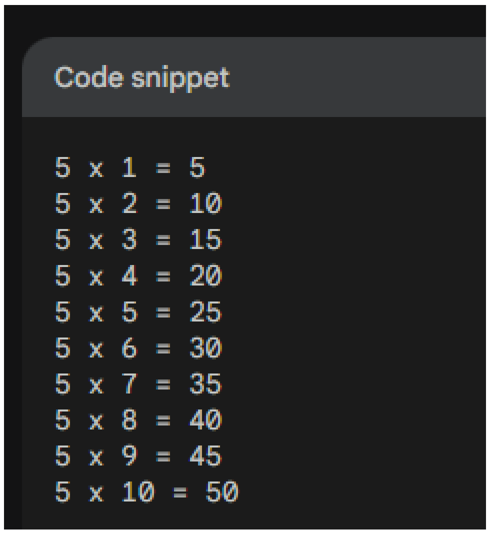
- Setting the axis limits of a plot.
- Rotating the axes of a plot.
- Zooming in on a specific region of a plot.
- Panning around a plot.
- Flipping the axes of a plot.

The `axis()` function is a powerful tool that can be used to customize the appearance of plots in Matplotlib.

(a) Show a python program to print multiplication table of a given number.

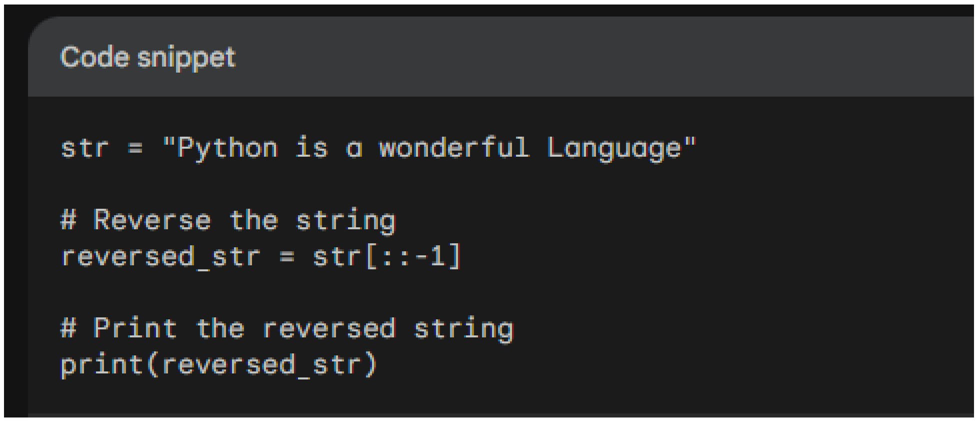

Python

```python
def print_multiplication_table(number):
  for i in range(1, 11):
    print(f"{number} x {i} = {number * i}")

number = int(input("Enter a number: "))
print_multiplication_table(number)
```


Code snippet

```
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
```

(b) Explain the syntax to read a string from the right-hand side. Consider str = "Python is a wonderful Language". Reverse and print the str from the right-hand side.

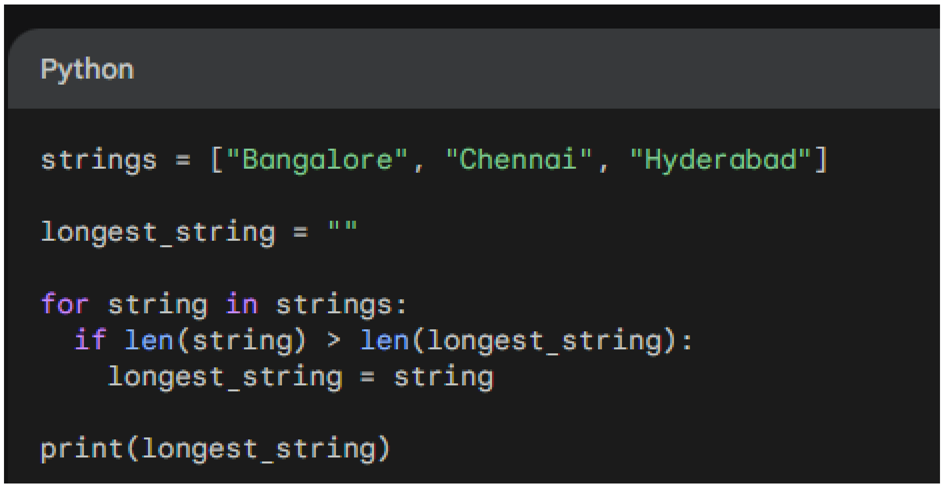


Code snippet

```
str = "Python is a wonderful Language"

# Reverse the string
reversed_str = str[::-1]

# Print the reversed string
print(reversed_str)
```

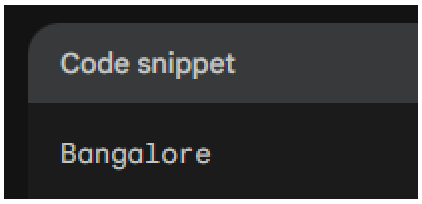
29. a) Assume three strings to compare. Using else-if compare then and print the longest string.

(b) Assume the string str = "Bangalore", Iterate it using for loop, and print characters vertically.



Python

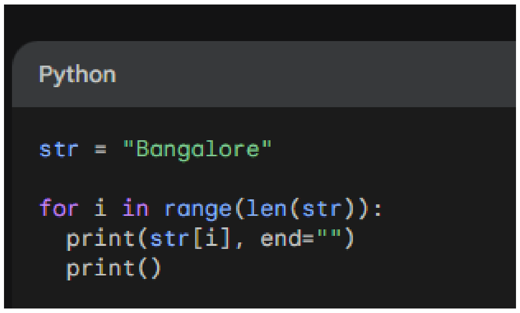
```
strings = ["Bangalore", "Chennai", "Hyderabad"]

longest_string = ""

for string in strings:
  if len(string) > len(longest_string):
    longest_string = string

print(longest_string)
```



Code snippet

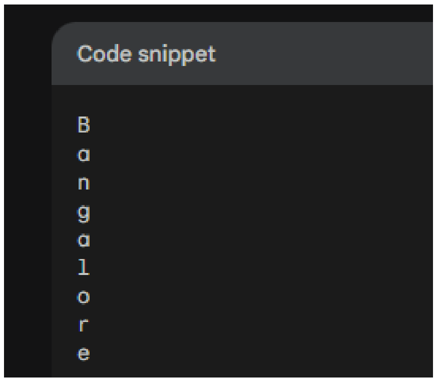
```
Bangalore
```

b



Python

```
str = "Bangalore"

for i in range(len(str)):
  print(str[i], end="")
  print()
```

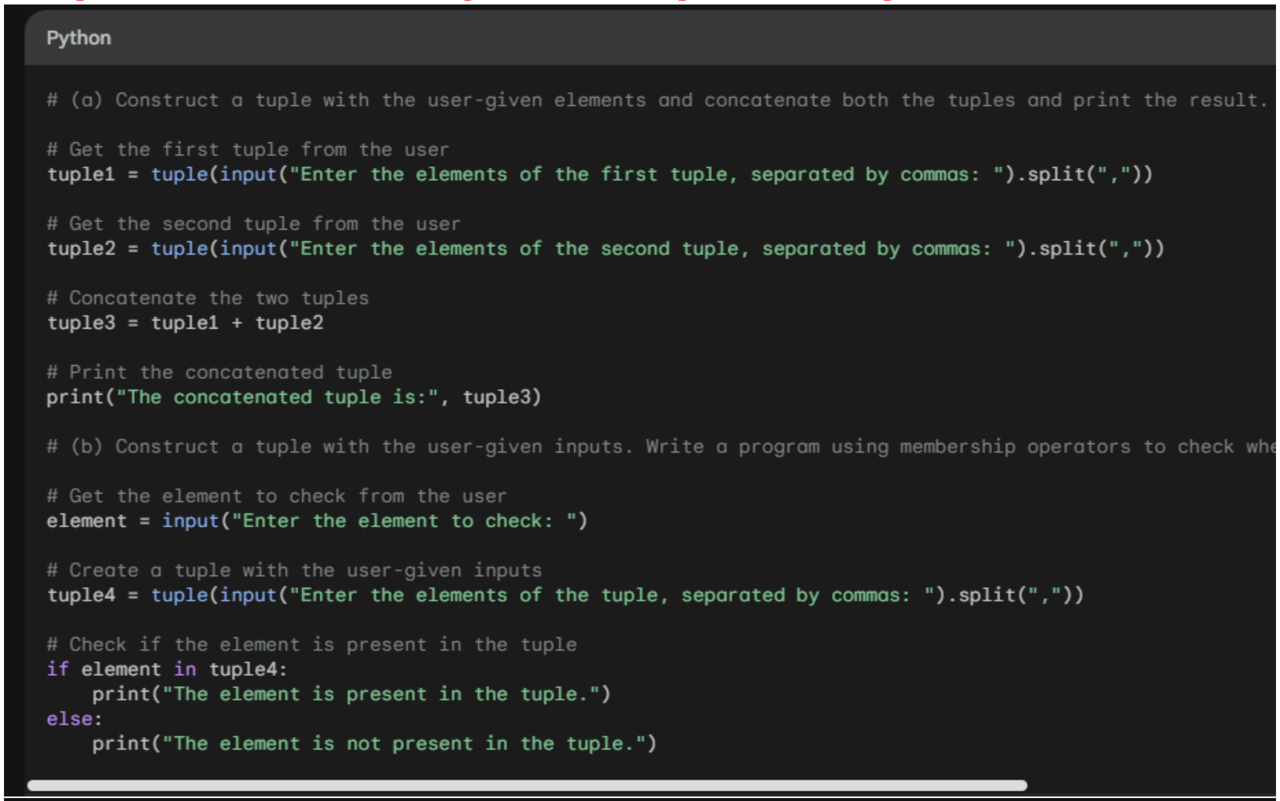


Code snippet

```
B
a
n
g
a
l
o
r
e
```

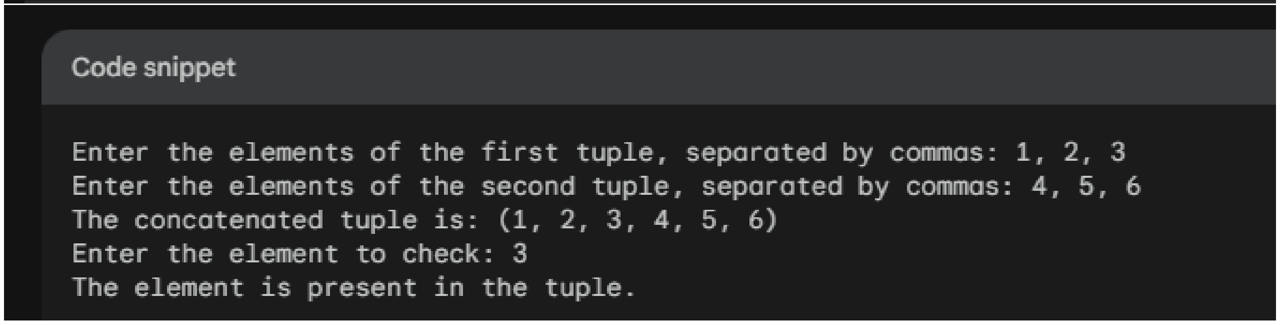
30. a) Construct a tuple with the user-given elements and concatenate both the tuples and print the result.

(b) Construct a tuple with the user-given inputs. Write a program using membership operators to check whether the given element is present in the tuple or not. Print the result.



Python

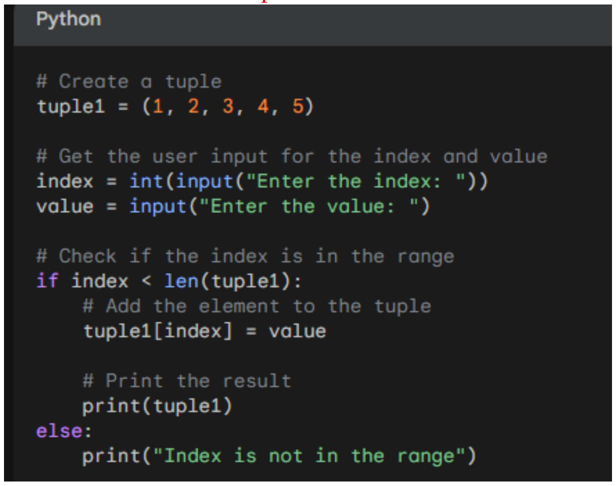
```python
# (a) Construct a tuple with the user-given elements and concatenate both the tuples and print the result.

# Get the first tuple from the user
tuple1 = tuple(input("Enter the elements of the first tuple, separated by commas: ").split(","))

# Get the second tuple from the user
tuple2 = tuple(input("Enter the elements of the second tuple, separated by commas: ").split(","))

# Concatenate the two tuples
tuple3 = tuple1 + tuple2

# Print the concatenated tuple
print("The concatenated tuple is:", tuple3)

# (b) Construct a tuple with the user-given inputs. Write a program using membership operators to check whe

# Get the element to check from the user
element = input("Enter the element to check: ")

# Create a tuple with the user-given inputs
tuple4 = tuple(input("Enter the elements of the tuple, separated by commas: ").split(","))

# Check if the element is present in the tuple
if element in tuple4:
    print("The element is present in the tuple.")
else:
    print("The element is not present in the tuple.")
```



Code snippet

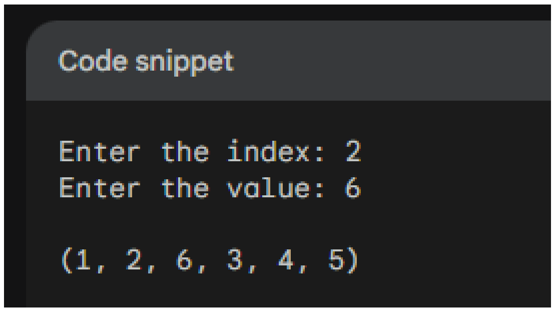
```
Enter the elements of the first tuple, separated by commas: 1, 2, 3
Enter the elements of the second tuple, separated by commas: 4, 5, 6
The concatenated tuple is: (1, 2, 3, 4, 5, 6)
Enter the element to check: 3
The element is present in the tuple.
```

(c) Apply a python program to add an element to a tuple based on the user-given value in a specific index, and print the result as shown in the example. If the index is not in the range print the error message as shown in the example.



Python

```python
# Create a tuple
tuple1 = (1, 2, 3, 4, 5)

# Get the user input for the index and value
index = int(input("Enter the index: "))
value = input("Enter the value: ")

# Check if the index is in the range
if index < len(tuple1):
    # Add the element to the tuple
    tuple1[index] = value

    # Print the result
    print(tuple1)
else:
    print("Index is not in the range")
```

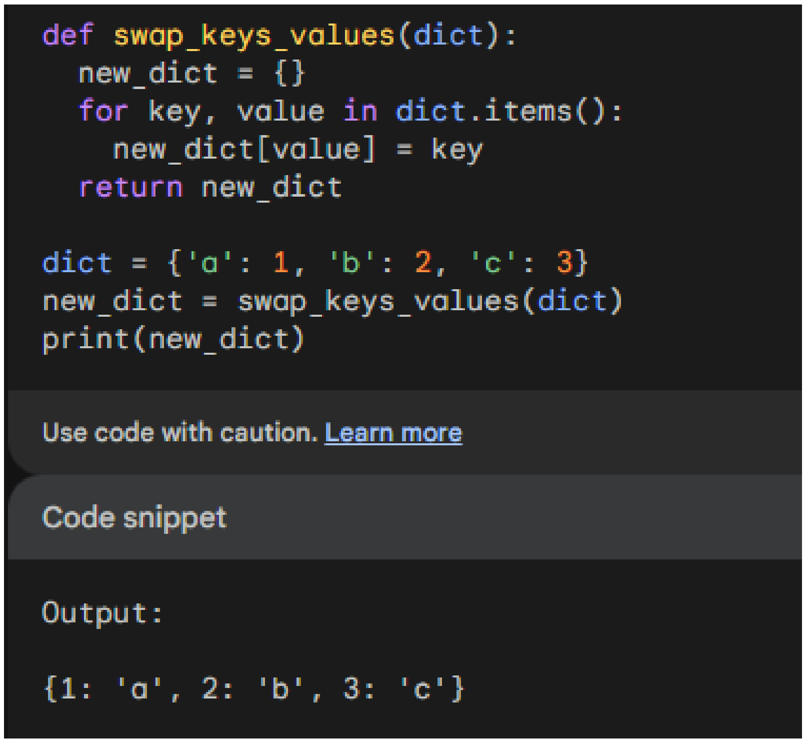


Code snippet

```
Enter the index: 2
Enter the value: 6

(1, 2, 6, 3, 4, 5)
```

31. a) Construct a python program to change Dictionary keys into values and values into keys, and print the result.

(b)Choose an integer sequence from the user. Write a program to print a dictionary from the given sequence. Consider the element in the sequence as a key, and the number of times the element occurs in the sequence as a value. Print the result.

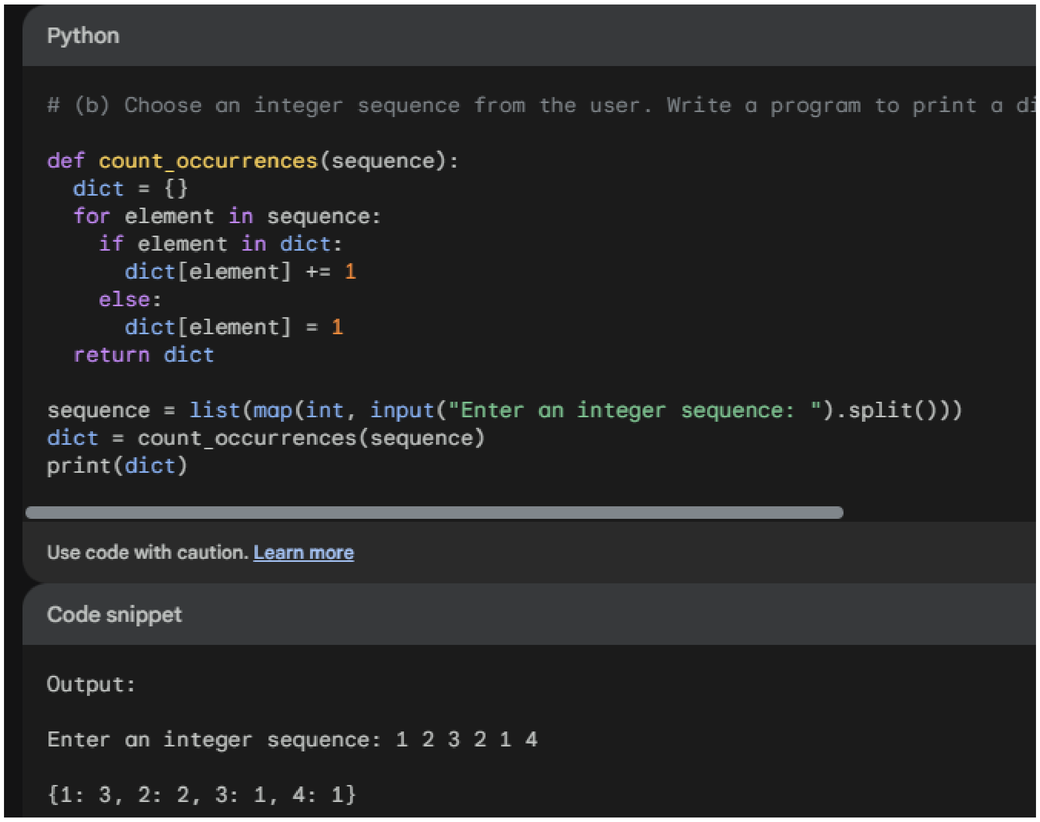


```python
def swap_keys_values(dict):
  new_dict = {}
  for key, value in dict.items():
    new_dict[value] = key
  return new_dict

dict = {'a': 1, 'b': 2, 'c': 3}
new_dict = swap_keys_values(dict)
print(new_dict)
```

Use code with caution. Learn more

Code snippet

```
Output:

{1: 'a', 2: 'b', 3: 'c'}
```



Python

```python
# (b) Choose an integer sequence from the user. Write a program to print a di

def count_occurrences(sequence):
  dict = {}
  for element in sequence:
    if element in dict:
      dict[element] += 1
    else:
      dict[element] = 1
  return dict

sequence = list(map(int, input("Enter an integer sequence: ").split()))
dict = count_occurrences(sequence)
print(dict)
```

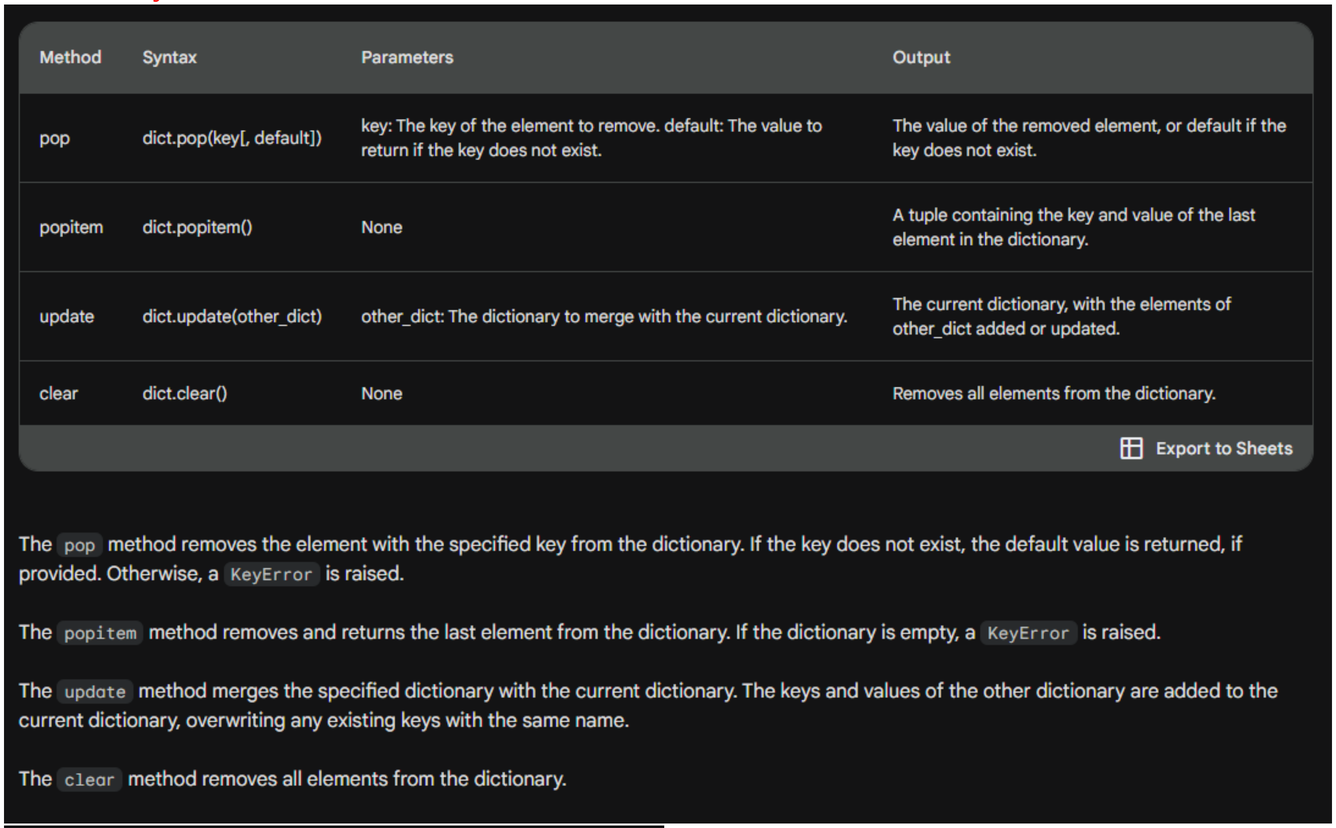
Use code with caution. Learn more

Code snippet

```
Output:

Enter an integer sequence: 1 2 3 2 1 4

{1: 3, 2: 2, 3: 1, 4: 1}
```

32. a. Identify different methods available in python dictionaries to add, remove, or change elements within a dictionary, including their syntax, parameters, and output.

b. Make Use of the "pop" method present in python dictionaries, including its syntax, parameters and output. Identify how it is differ from the "popitem" method, and when would you use one over the other?



Method	Syntax	Parameters	Output
pop	dict.pop(key[, default])	key: The key of the element to remove. default: The value to return if the key does not exist.	The value of the removed element, or default if the key does not exist.
popitem	dict.popitem()	None	A tuple containing the key and value of the last element in the dictionary.
update	dict.update(other_dict)	other_dict: The dictionary to merge with the current dictionary.	The current dictionary, with the elements of other_dict added or updated.
clear	dict.clear()	None	Removes all elements from the dictionary.

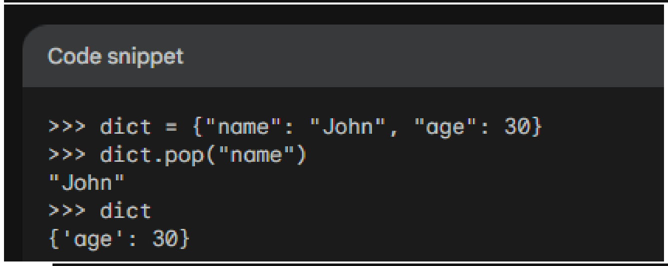
Export to Sheets

The `pop` method removes the element with the specified key from the dictionary. If the key does not exist, the default value is returned, if provided. Otherwise, a `KeyError` is raised.

The `popitem` method removes and returns the last element from the dictionary. If the dictionary is empty, a `KeyError` is raised.

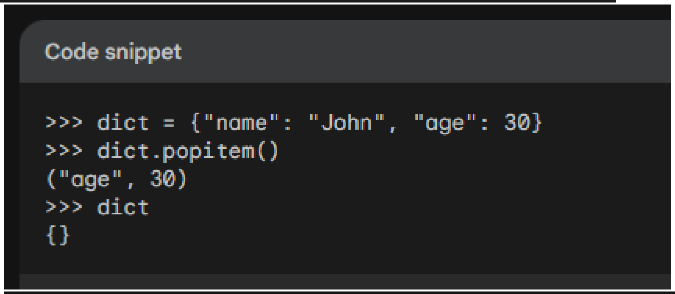
The `update` method merges the specified dictionary with the current dictionary. The keys and values of the other dictionary are added to the current dictionary, overwriting any existing keys with the same name.

The `clear` method removes all elements from the dictionary.



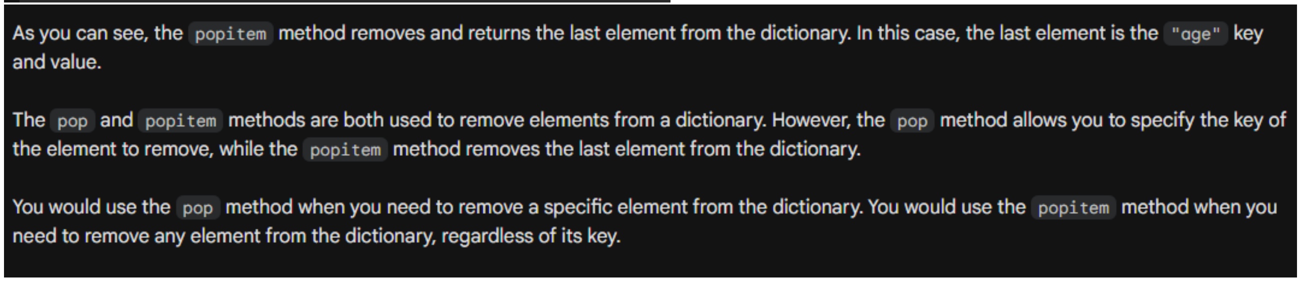
Code snippet

```
>>> dict = {"name": "John", "age": 30}
>>> dict.pop("name")
"John"
>>> dict
{'age': 30}
```



Code snippet

```
>>> dict = {"name": "John", "age": 30}
>>> dict.popitem()
("age", 30)
>>> dict
{}
```

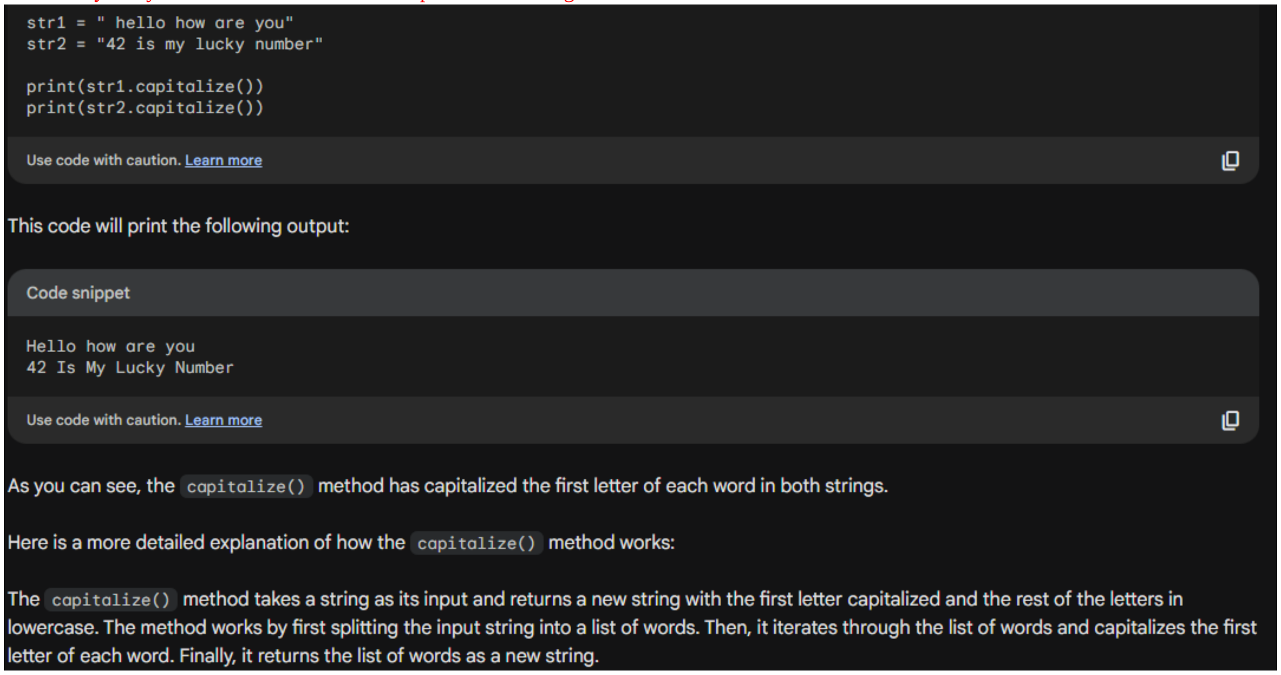


As you can see, the `popitem` method removes and returns the last element from the dictionary. In this case, the last element is the `"age"` key and value.

The `pop` and `popitem` methods are both used to remove elements from a dictionary. However, the `pop` method allows you to specify the key of the element to remove, while the `popitem` method removes the last element from the dictionary.

You would use the `pop` method when you need to remove a specific element from the dictionary. You would use the `popitem` method when you need to remove any element from the dictionary, regardless of its key.

33. Choose the capitalize() method works in python. Take str1 = " hello how are you" & str2 = "42 is my lucky number". Demonstrate the output for both strings.



```
str1 = " hello how are you"
str2 = "42 is my lucky number"

print(str1.capitalize())
print(str2.capitalize())
```

Use code with caution. Learn more

This code will print the following output:

Code snippet

```
Hello how are you
42 Is My Lucky Number
```

Use code with caution. Learn more

As you can see, the `capitalize()` method has capitalized the first letter of each word in both strings.

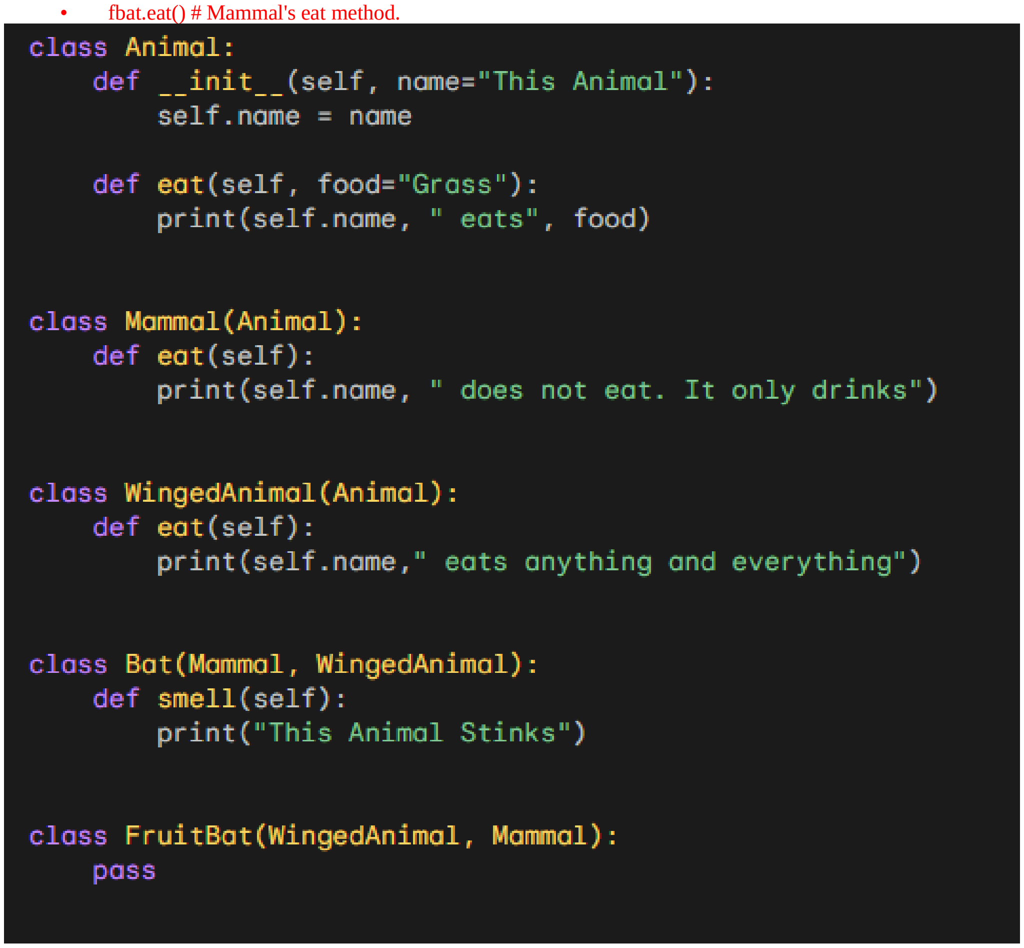
Here is a more detailed explanation of how the `capitalize()` method works:

The `capitalize()` method takes a string as its input and returns a new string with the first letter capitalized and the rest of the letters in lowercase. The method works by first splitting the input string into a list of words. Then, it iterates through the list of words and capitalizes the first letter of each word. Finally, it returns the list of words as a new string.

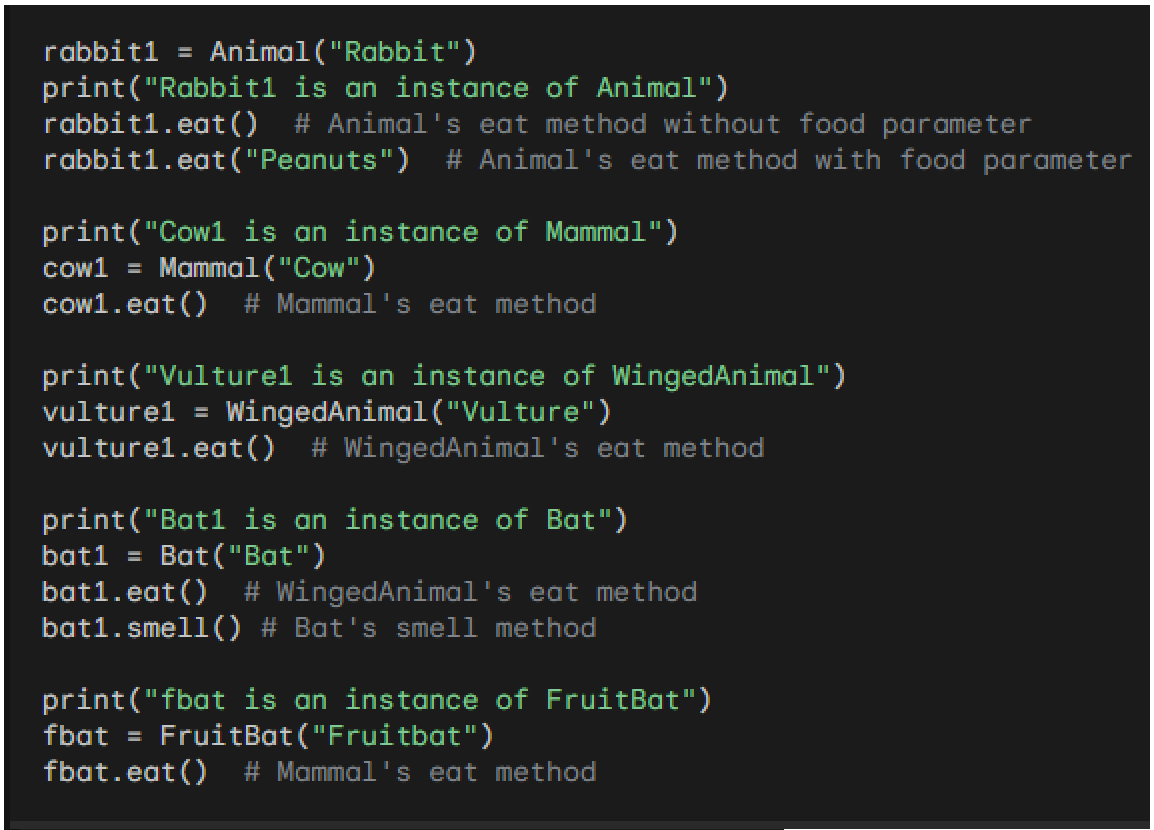
34. Assume the given below instructions to implements the method overriding method:

- Define class Animal
- Use constructor to set the name with a default value = "This Animal"
- Define a method eat with a parameter food with a default value = "Grass"
- Inside the method print (self.name, " eats", food)
- Define a class Mammal, inherit from Animal
- Inside the class, override eat method to print(self.name, " does not eat. It only drinks")
- Define class WingedAnimal, inherit from Animal

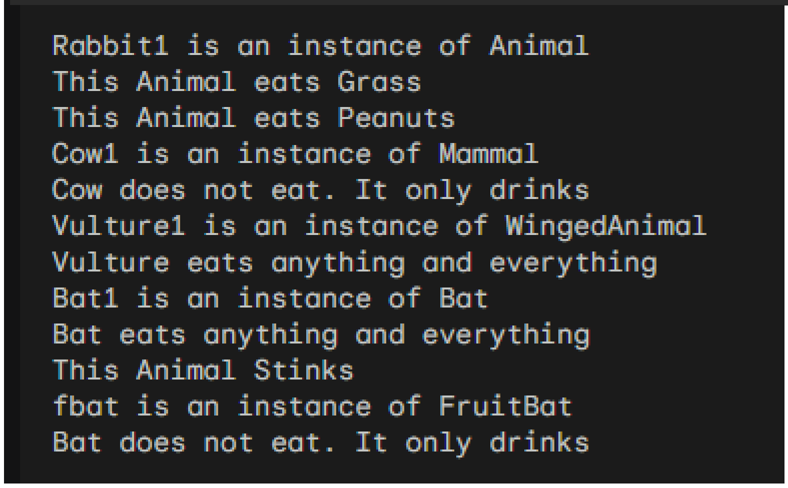
- Override eat method to print(self.name," eats anything and everything")
- Define a class called Bat, inherit from WingedAnimal, Mammal
- Define method smell, which prints "This Animal Stinks"
- Define a class called FruitBat, inherit from Mammal, WingedAnimal (Notice the Order)
- rabbit1 = Animal("Rabbit")
- print("Rabbit1 is an instance of Animal")
- rabbit1.eat() # Animal's eat method without food parameter
- rabbit1.eat("Peanuts") # Animal's eat method with food parameter
- print("Cow1 is an instance of Mammal")
- cow1 = Mammal("Cow")
- cow1.eat() # Mammal's eat method
- print("Vulture1 is an instance of WingedAnimal")
- vulture1 = WingedAnimal("Vulture")
- vulture1.eat() # WingedAnimal's eat method
- print("Bat1 is an instance of Bat")
- bat1 = Bat("Bat")
- bat1.eat() # WingedAnimal's eat method
- print("fbat is an instance of FruitBat")
- fbat = FruitBat("Fruitbat")
- fbat.eat() # Mammal's eat method.



```python
class Animal:
    def __init__(self, name="This Animal"):
        self.name = name

    def eat(self, food="Grass"):
        print(self.name, " eats", food)


class Mammal(Animal):
    def eat(self):
        print(self.name, " does not eat. It only drinks")


class WingedAnimal(Animal):
    def eat(self):
        print(self.name," eats anything and everything")


class Bat(Mammal, WingedAnimal):
    def smell(self):
        print("This Animal Stinks")


class FruitBat(WingedAnimal, Mammal):
    pass
```



```python
rabbit1 = Animal("Rabbit")
print("Rabbit1 is an instance of Animal")
rabbit1.eat()  # Animal's eat method without food parameter
rabbit1.eat("Peanuts")  # Animal's eat method with food parameter

print("Cow1 is an instance of Mammal")
cow1 = Mammal("Cow")
cow1.eat()  # Mammal's eat method

print("Vulture1 is an instance of WingedAnimal")
vulture1 = WingedAnimal("Vulture")
vulture1.eat()  # WingedAnimal's eat method

print("Bat1 is an instance of Bat")
bat1 = Bat("Bat")
bat1.eat()  # WingedAnimal's eat method
bat1.smell() # Bat's smell method

print("fbat is an instance of FruitBat")
fbat = FruitBat("Fruitbat")
fbat.eat()  # Mammal's eat method
```



```
Rabbit1 is an instance of Animal
This Animal eats Grass
This Animal eats Peanuts
Cow1 is an instance of Mammal
Cow does not eat. It only drinks
Vulture1 is an instance of WingedAnimal
Vulture eats anything and everything
Bat1 is an instance of Bat
Bat eats anything and everything
This Animal Stinks
fbat is an instance of FruitBat
Bat does not eat. It only drinks
```

35. a) Assume the given instructions below to write the program

i.Take the input values a, b from the user.

ii. Define the function add().

iii. It takes two arguments a and b.

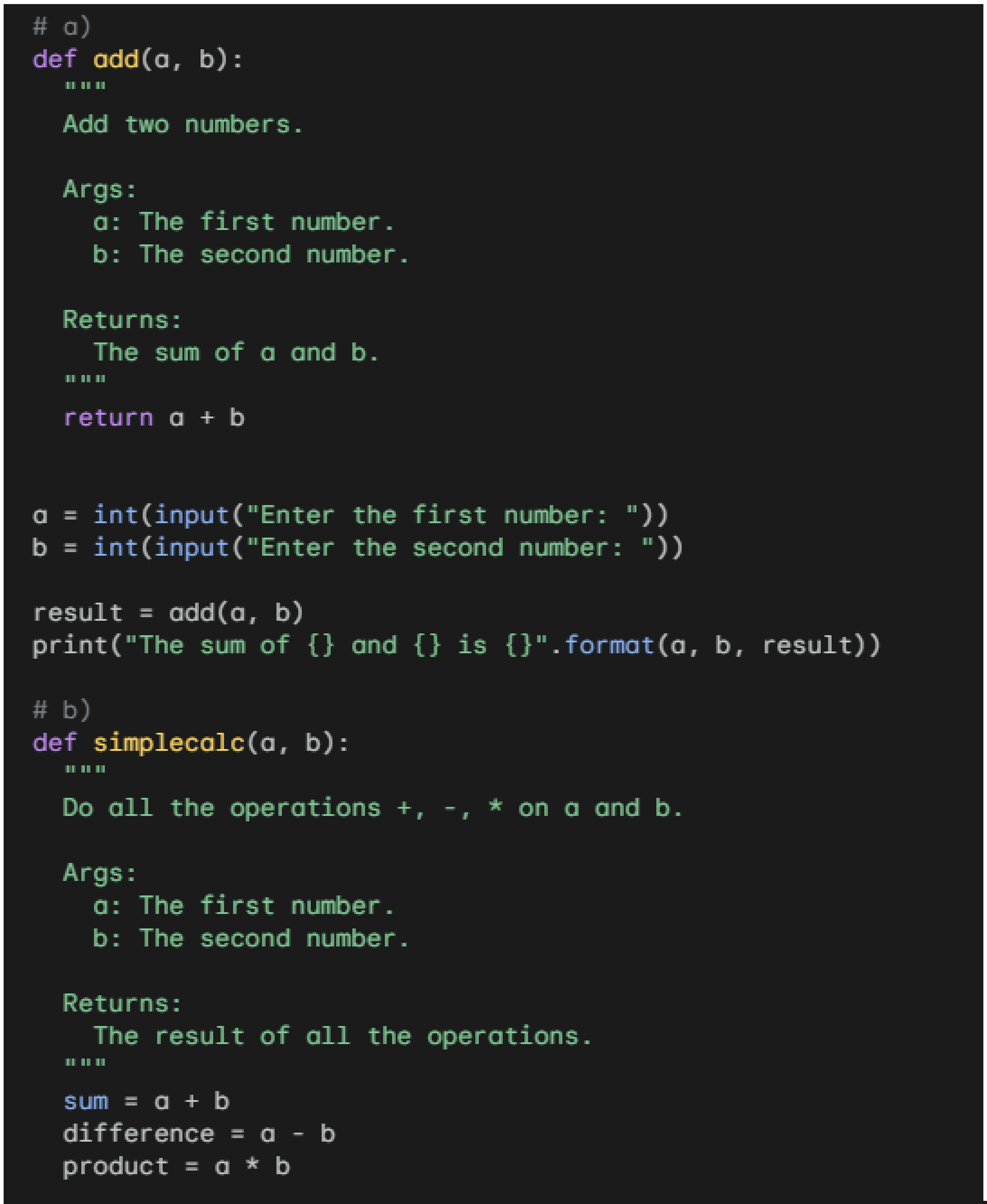
iv. Add a and b and return the result.

v. Call this function by passing the argument values a and b and print the result.

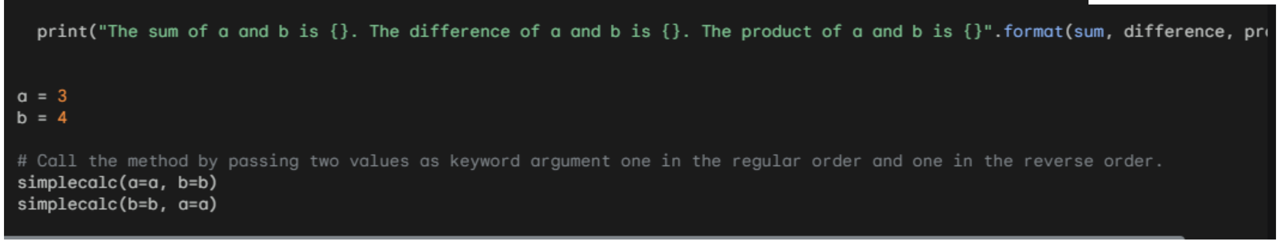
b) Assume the given instructions and write the program to undertand Key arguments

• Define a function simplecalc()

- Takes two parameters a and b simplecalc(a, b)
- Inside the function do all the operations +, -, * on a and b.
- Print the result of all the operations
- set values a = 3 and b = 4.
- Call the method by passing two values as keyword argument one in the regular order and one in the reverse order.



```
# a)
def add(a, b):
  """
  Add two numbers.

  Args:
    a: The first number.
    b: The second number.

  Returns:
    The sum of a and b.
  """
  return a + b


a = int(input("Enter the first number: "))
b = int(input("Enter the second number: "))

result = add(a, b)
print("The sum of {} and {} is {}".format(a, b, result))

# b)
def simplecalc(a, b):
  """
  Do all the operations +, -, * on a and b.

  Args:
    a: The first number.
    b: The second number.

  Returns:
    The result of all the operations.
  """
  sum = a + b
  difference = a - b
  product = a * b
```

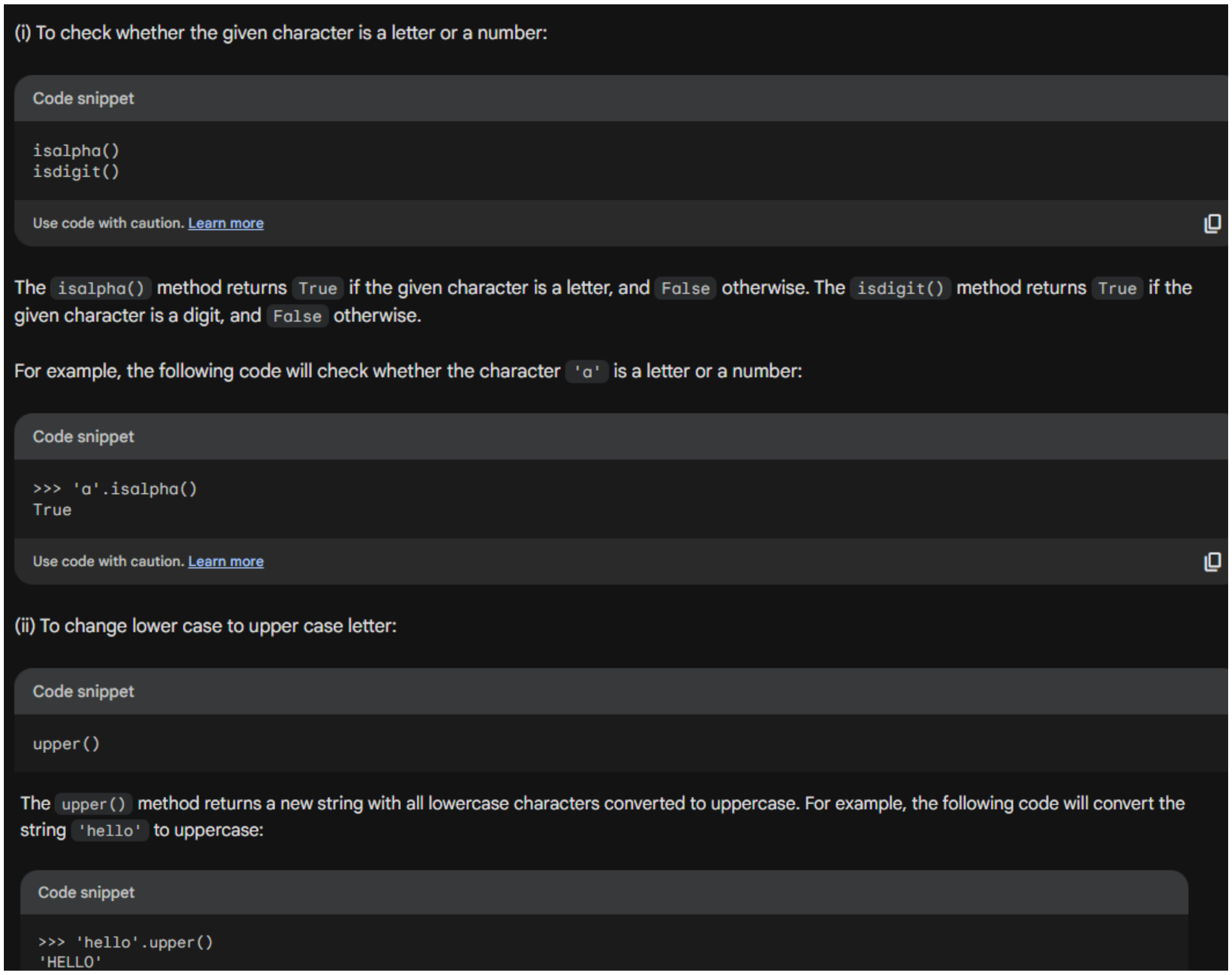


```
  print("The sum of a and b is {}. The difference of a and b is {}. The product of a and b is {}".format(sum, difference, pr


a = 3
b = 4

# Call the method by passing two values as keyword argument one in the regular order and one in the reverse order.
simplecalc(a=a, b=b)
simplecalc(b=b, a=a)
```

36. Which string method is used to implement the following?

(i) To check whether the given character is a letter or a number.

(ii) To change lower case to upper case letter.



(i) To check whether the given character is a letter or a number:

Code snippet

```
isalpha()
isdigit()
```

Use code with caution. Learn more

The `isalpha()` method returns `True` if the given character is a letter, and `False` otherwise. The `isdigit()` method returns `True` if the given character is a digit, and `False` otherwise.

For example, the following code will check whether the character `'a'` is a letter or a number:

Code snippet

```
>>> 'a'.isalpha()
True
```

Use code with caution. Learn more

(ii) To change lower case to upper case letter:

Code snippet

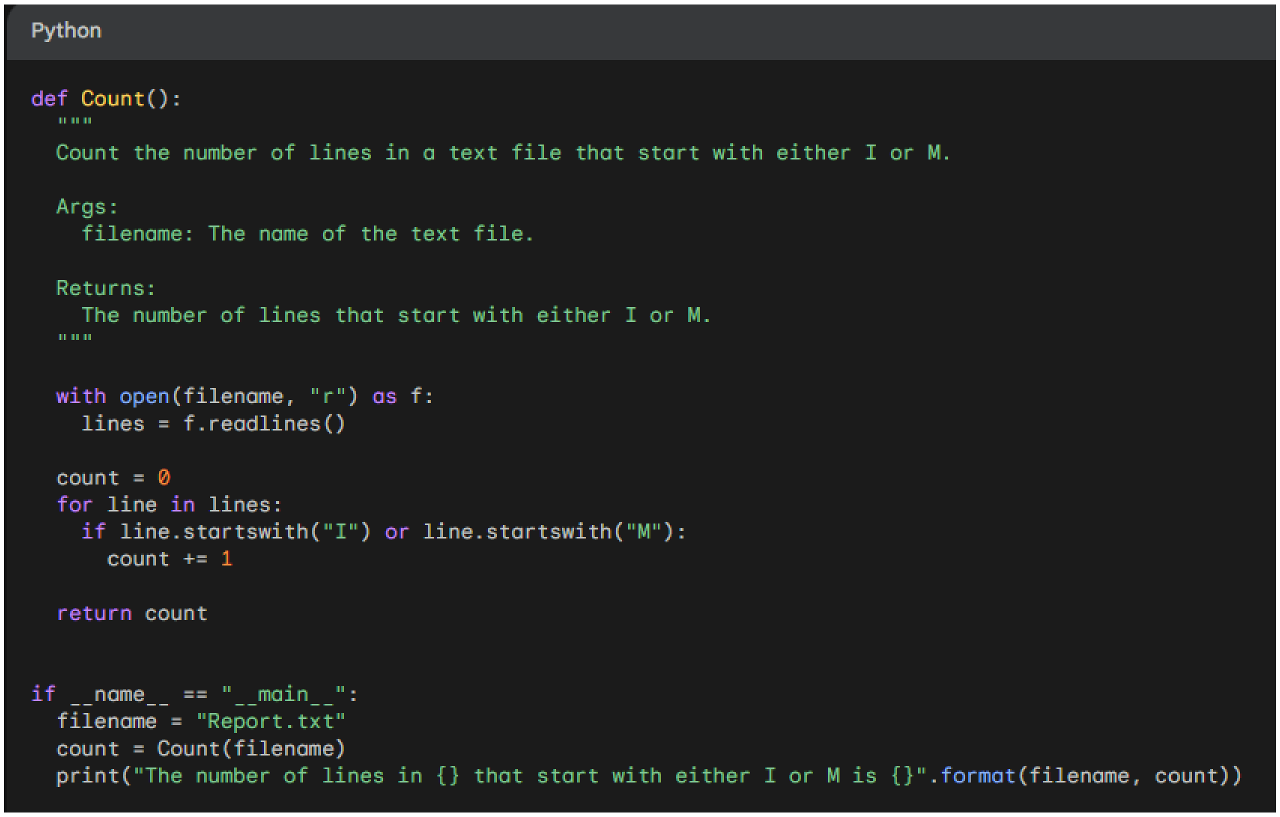
```
upper()
```

The `upper()` method returns a new string with all lowercase characters converted to uppercase. For example, the following code will convert the string `'hello'` to uppercase:

Code snippet

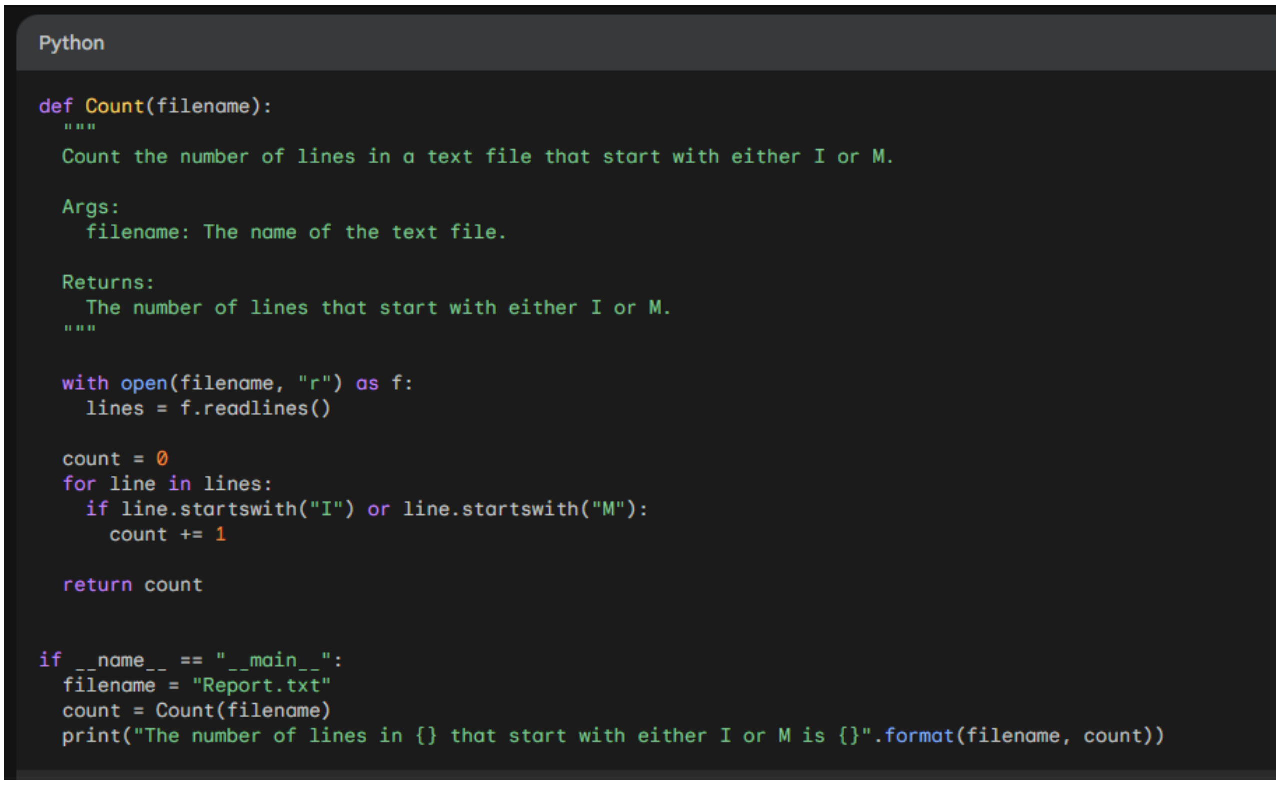
```
>>> 'hello'.upper()
'HELLO'
```

37. Write a user-defined function named Count() that will read the contents of text file named "Report.txt" and count the number of lines which starts with either "I" or "M" and displays the count.

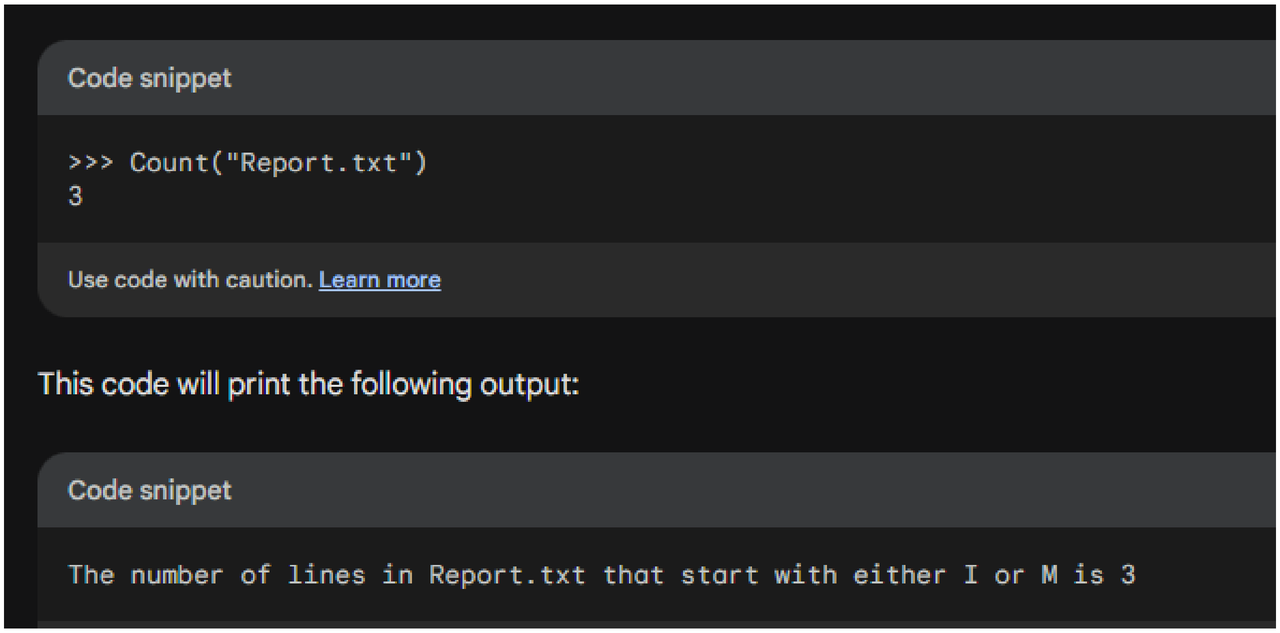

Python

```python
def Count():
  """
  Count the number of lines in a text file that start with either I or M.

  Args:
    filename: The name of the text file.

  Returns:
    The number of lines that start with either I or M.
  """

  with open(filename, "r") as f:
    lines = f.readlines()

  count = 0
  for line in lines:
    if line.startswith("I") or line.startswith("M"):
      count += 1

  return count


if __name__ == "__main__":
  filename = "Report.txt"
  count = Count(filename)
  print("The number of lines in {} that start with either I or M is {}".format(filename, count))
```

38. Write a user-defined function named Count() that will read the contents of text file named “Report.txt” and count the number of lines which starts with either “I‟ or “M‟ and displays the count.


Python

```python
def Count(filename):
  """
  Count the number of lines in a text file that start with either I or M.

  Args:
    filename: The name of the text file.

  Returns:
    The number of lines that start with either I or M.
  """

  with open(filename, "r") as f:
    lines = f.readlines()

  count = 0
  for line in lines:
    if line.startswith("I") or line.startswith("M"):
      count += 1

  return count


if __name__ == "__main__":
  filename = "Report.txt"
  count = Count(filename)
  print("The number of lines in {} that start with either I or M is {}".format(filename, count))
```



Code snippet

```
>>> Count("Report.txt")
3
```

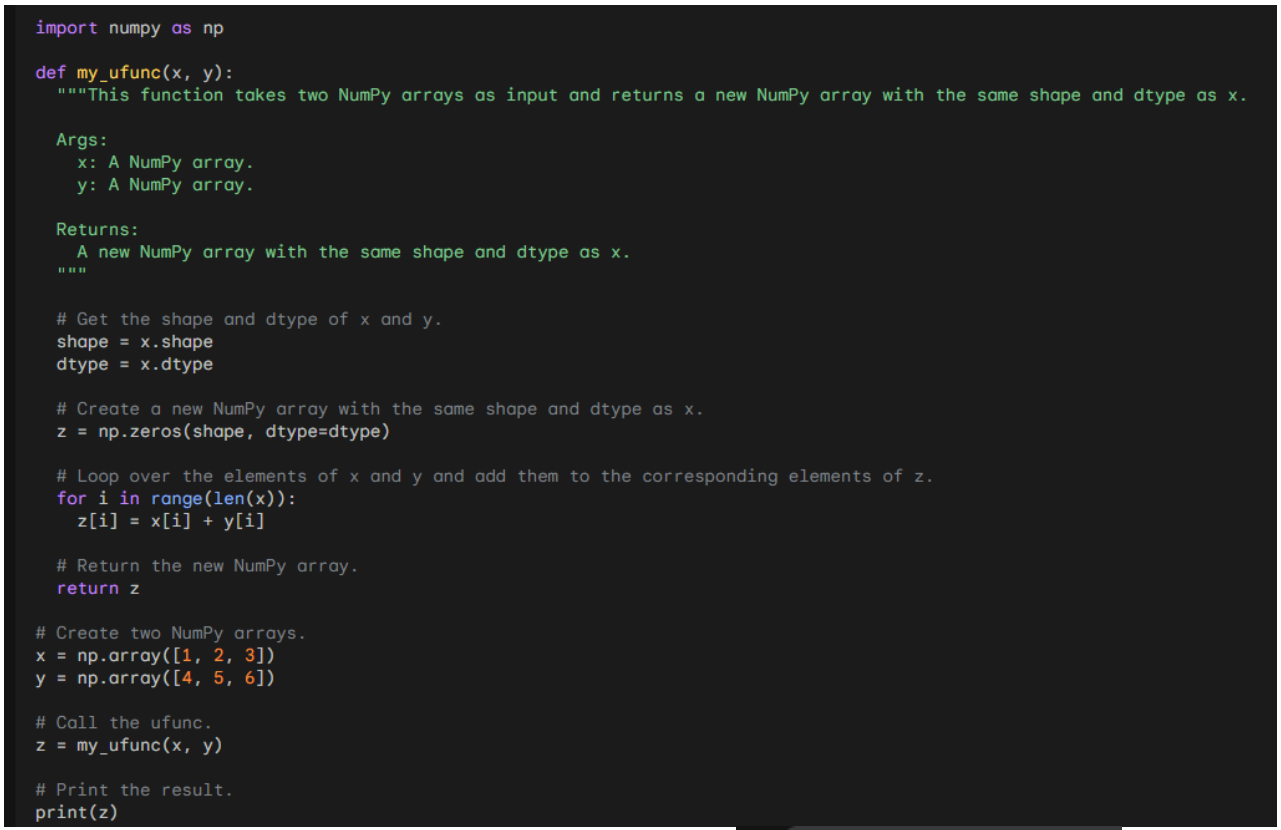
Use code with caution. Learn more

This code will print the following output:

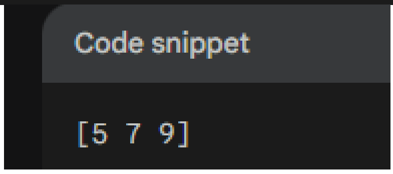
Code snippet

```
The number of lines in Report.txt that start with either I or M is 3
```

39. Implement in Python to create your own universal function in NumPy.



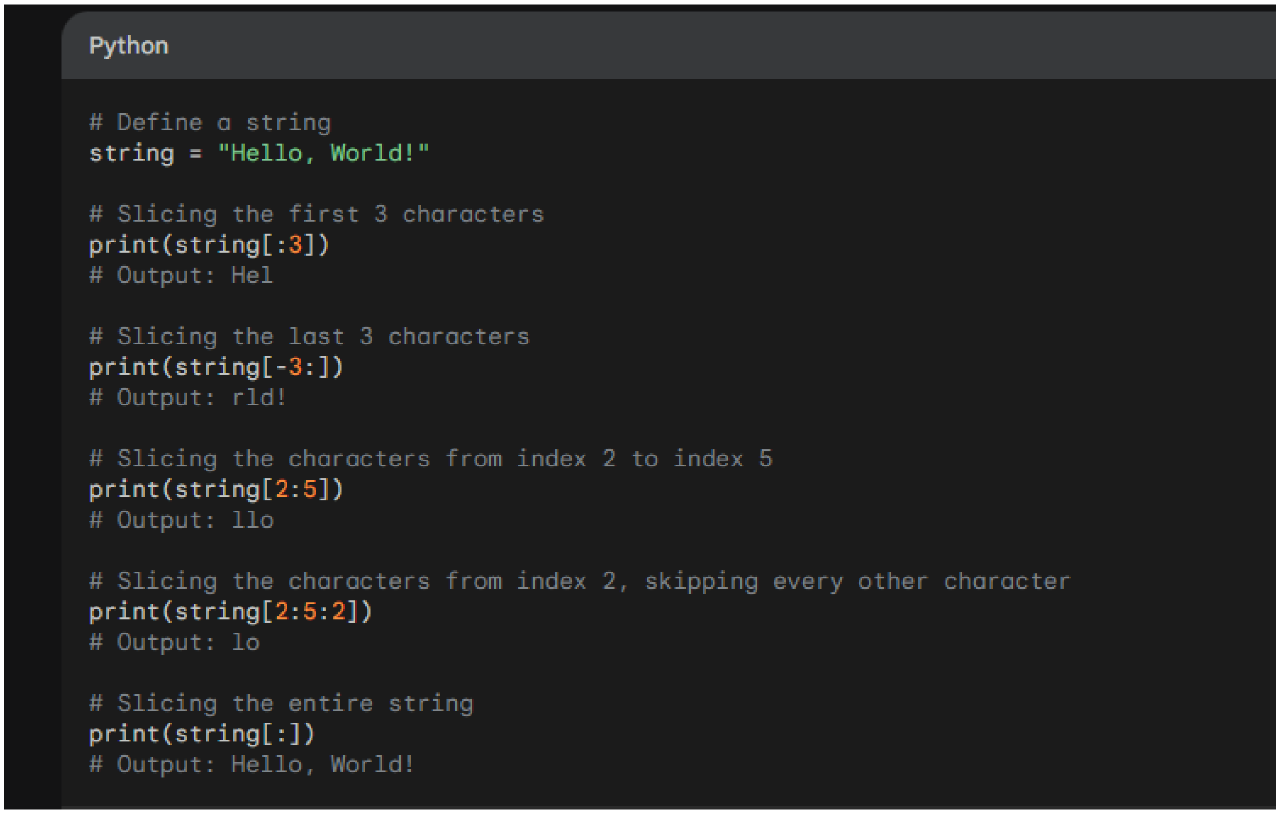
```
import numpy as np

def my_ufunc(x, y):
  """This function takes two NumPy arrays as input and returns a new NumPy array with the same shape and dtype as x.

  Args:
    x: A NumPy array.
    y: A NumPy array.

  Returns:
    A new NumPy array with the same shape and dtype as x.
  """

  # Get the shape and dtype of x and y.
  shape = x.shape
  dtype = x.dtype

  # Create a new NumPy array with the same shape and dtype as x.
  z = np.zeros(shape, dtype=dtype)

  # Loop over the elements of x and y and add them to the corresponding elements of z.
  for i in range(len(x)):
    z[i] = x[i] + y[i]

  # Return the new NumPy array.
  return z

# Create two NumPy arrays.
x = np.array([1, 2, 3])
y = np.array([4, 5, 6])

# Call the ufunc.
z = my_ufunc(x, y)

# Print the result.
print(z)
```



Code snippet

```
[5 7 9]
```

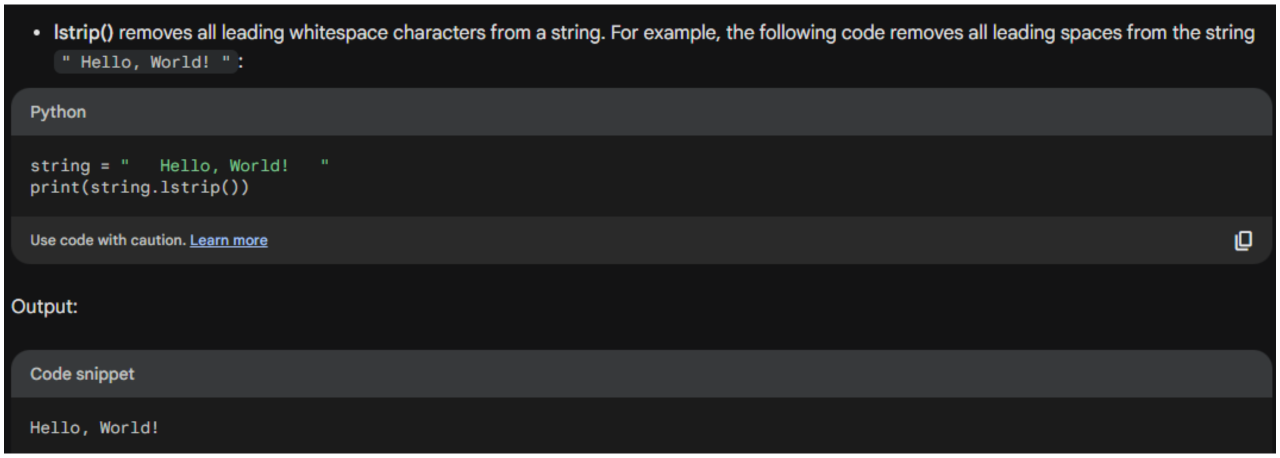
40. Devise string slicing in python with syntax. Take an example of string and demonstrate different types of slicing with code.



Python

```python
# Define a string
string = "Hello, World!"

# Slicing the first 3 characters
print(string[:3])
# Output: Hel

# Slicing the last 3 characters
print(string[-3:])
# Output: rld!

# Slicing the characters from index 2 to index 5
print(string[2:5])
# Output: llo

# Slicing the characters from index 2, skipping every other character
print(string[2:5:2])
# Output: lo

# Slicing the entire string
print(string[:])
# Output: Hello, World!
```

41. Discuss the use of the following methods with a sample string: lstrip(), rstrip(), strip()



- **lstrip()** removes all leading whitespace characters from a string. For example, the following code removes all leading spaces from the string `" Hello, World! "`:

Python

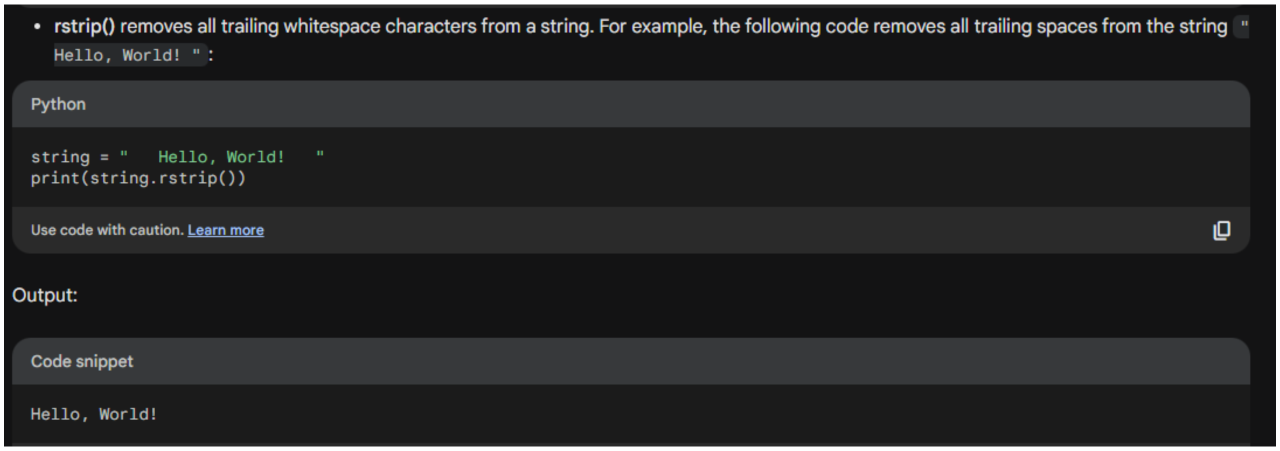
```python
string = "   Hello, World!   "
print(string.lstrip())
```

Use code with caution. Learn more

Output:

Code snippet

```
Hello, World!
```



- **rstrip()** removes all trailing whitespace characters from a string. For example, the following code removes all trailing spaces from the string `" Hello, World! "`:

Python

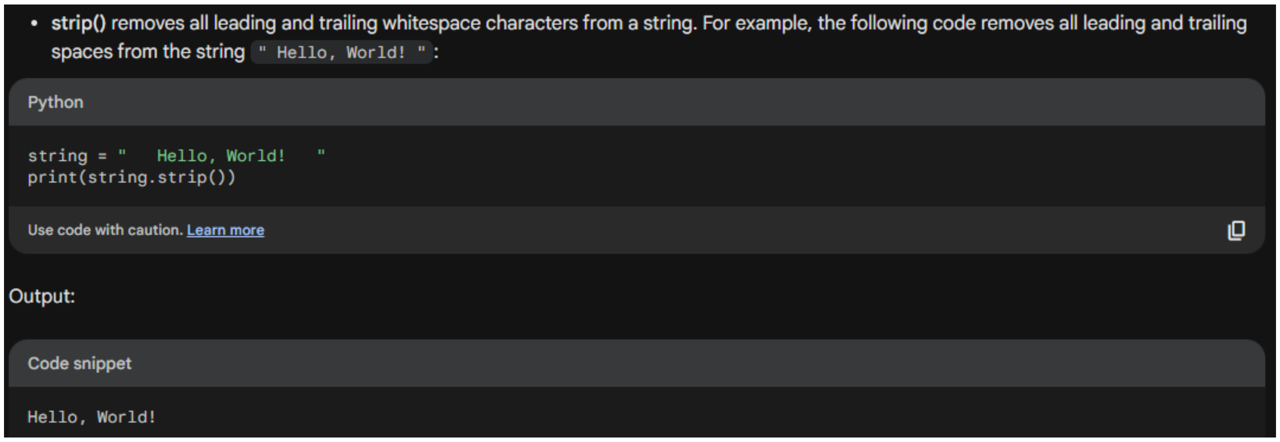
```python
string = "   Hello, World!   "
print(string.rstrip())
```

Use code with caution. Learn more

Output:

Code snippet

```
Hello, World!
```



- strip() removes all leading and trailing whitespace characters from a string. For example, the following code removes all leading and trailing spaces from the string `" Hello, World! "`:

Python

```python
string = "   Hello, World!   "
print(string.strip())
```

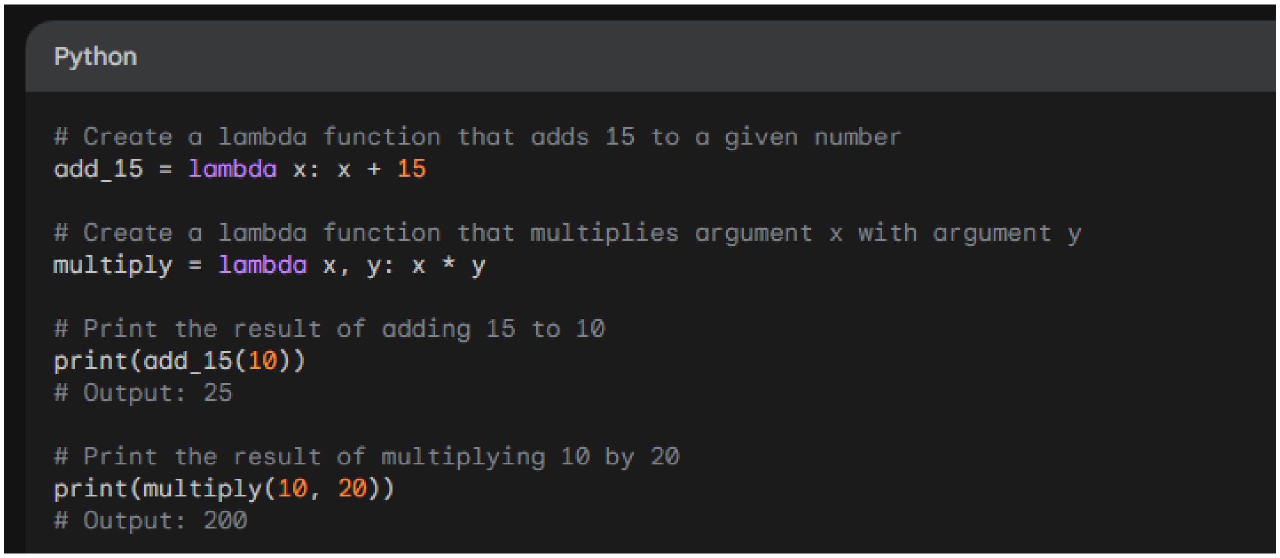
Use code with caution. Learn more

Output:

Code snippet

```
Hello, World!
```

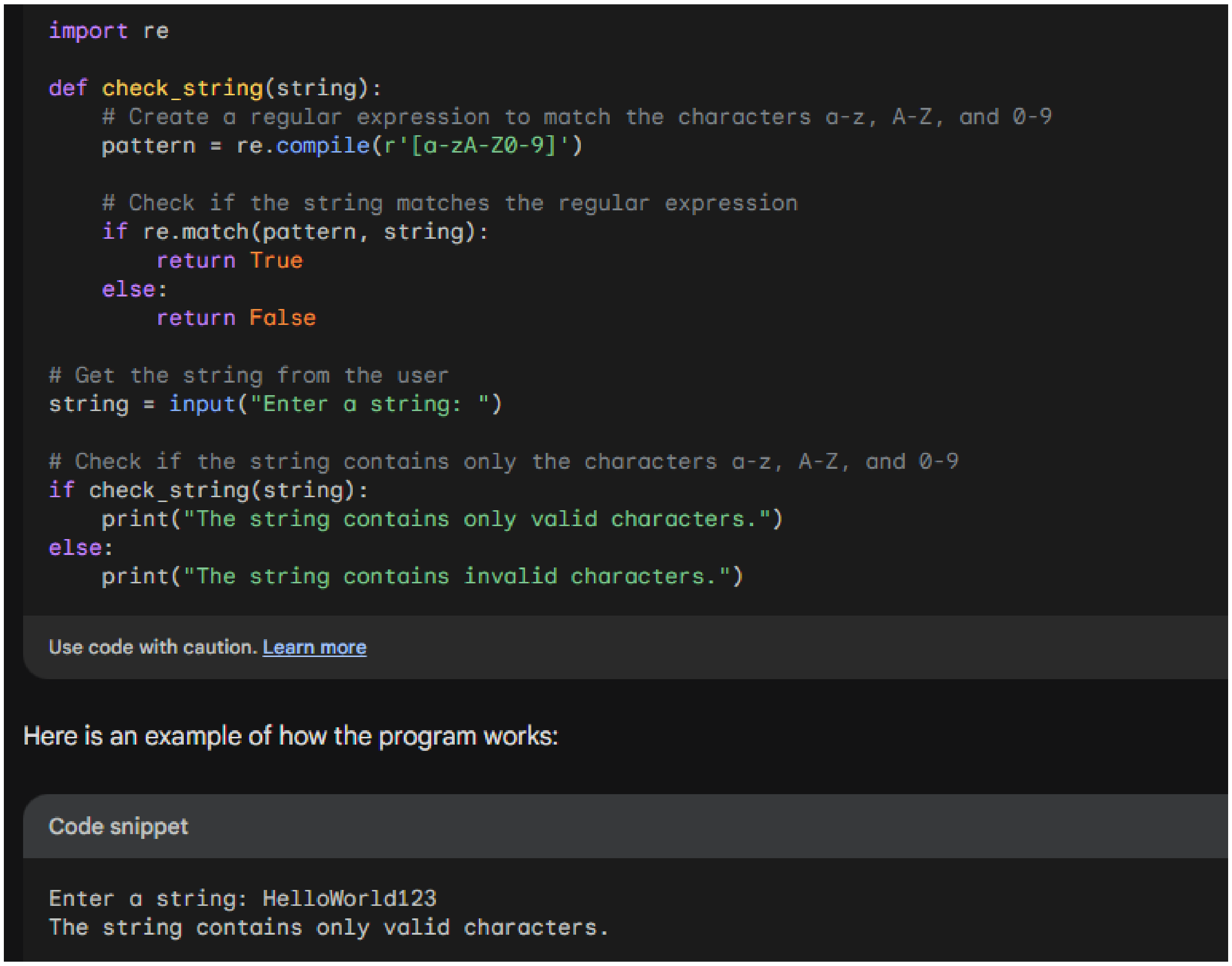
42. Write a Python program to create a lambda function that adds 15 to a given number passed in as an argument, also create a lambda function that multiplies argument x with argument y and prints the result.



Python

```python
# Create a lambda function that adds 15 to a given number
add_15 = lambda x: x + 15

# Create a lambda function that multiplies argument x with argument y
multiply = lambda x, y: x * y

# Print the result of adding 15 to 10
print(add_15(10))
# Output: 25

# Print the result of multiplying 10 by 20
print(multiply(10, 20))
# Output: 200
```

43. Using regular expression, write a python program to check that a string contains only a certain set of characters (in this case a-z, A-Z and 0-9).



```python
import re

def check_string(string):
    # Create a regular expression to match the characters a-z, A-Z, and 0-9
    pattern = re.compile(r'[a-zA-Z0-9]')

    # Check if the string matches the regular expression
    if re.match(pattern, string):
        return True
    else:
        return False

# Get the string from the user
string = input("Enter a string: ")

# Check if the string contains only the characters a-z, A-Z, and 0-9
if check_string(string):
    print("The string contains only valid characters.")
else:
    print("The string contains invalid characters.")
```

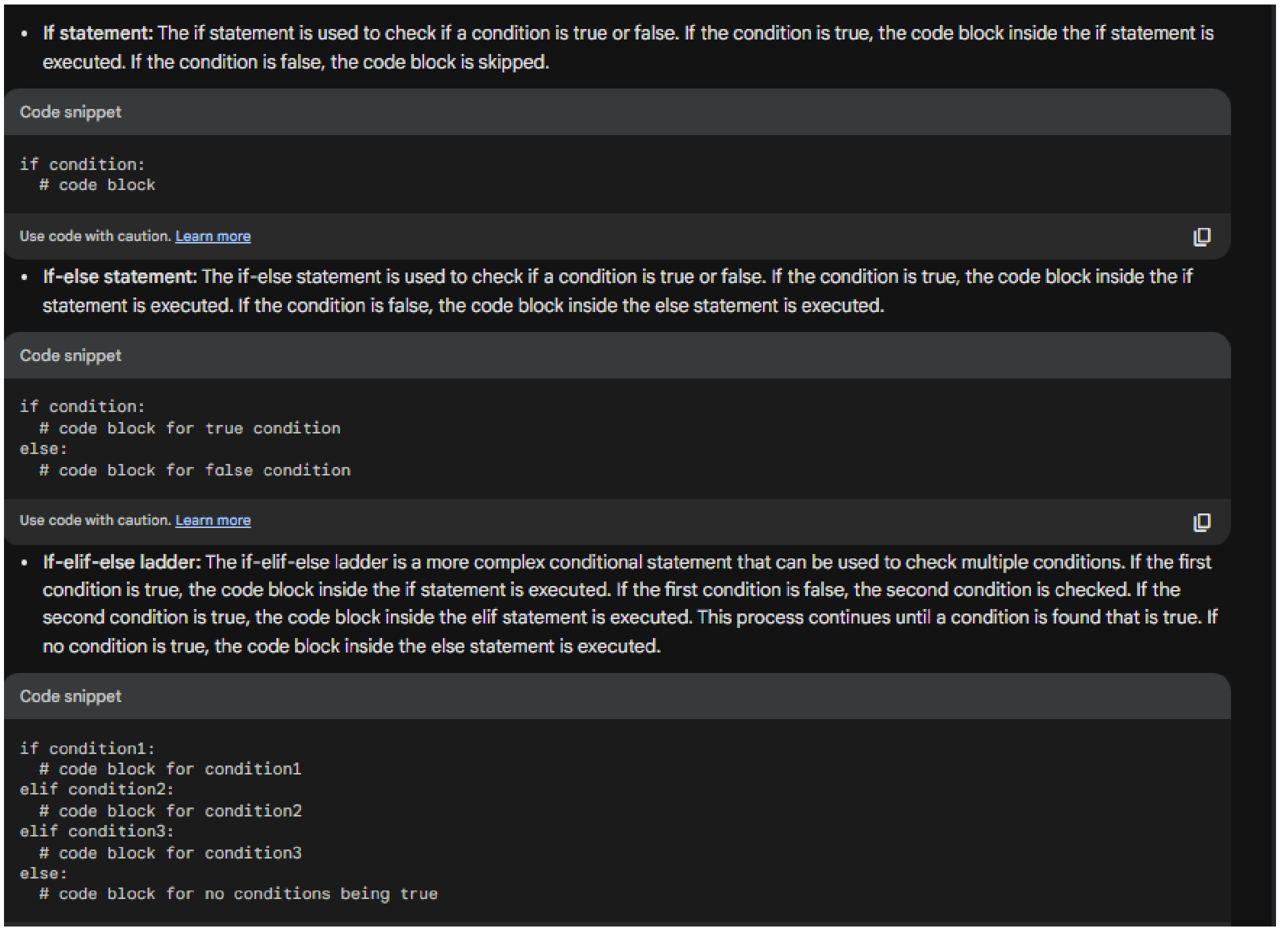
Use code with caution. Learn more

Here is an example of how the program works:

Code snippet

```
Enter a string: HelloWorld123
The string contains only valid characters.
```

44. List the three types of conditional statements and explain them.



- **If statement:** The if statement is used to check if a condition is true or false. If the condition is true, the code block inside the if statement is executed. If the condition is false, the code block is skipped.

Code snippet

```
if condition:
  # code block
```

Use code with caution. Learn more

- **If-else statement:** The if-else statement is used to check if a condition is true or false. If the condition is true, the code block inside the if statement is executed. If the condition is false, the code block inside the else statement is executed.

Code snippet

```
if condition:
  # code block for true condition
else:
  # code block for false condition
```

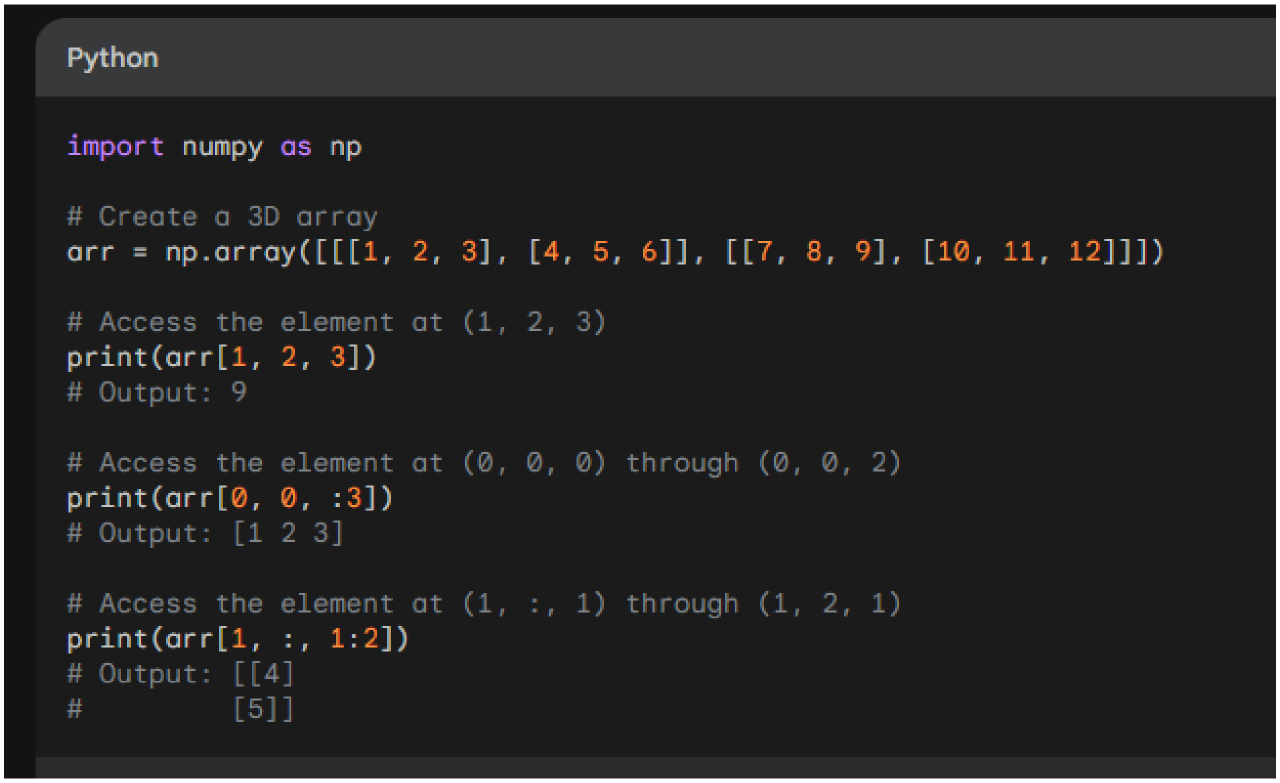
Use code with caution. Learn more

- **If-elif-else ladder:** The if-elif-else ladder is a more complex conditional statement that can be used to check multiple conditions. If the first condition is true, the code block inside the if statement is executed. If the first condition is false, the second condition is checked. If the second condition is true, the code block inside the elif statement is executed. This process continues until a condition is found that is true. If no condition is true, the code block inside the else statement is executed.

Code snippet

```
if condition1:
  # code block for condition1
elif condition2:
  # code block for condition2
elif condition3:
  # code block for condition3
else:
  # code block for no conditions being true
```

45. Implement in Python to accessing Data Along Multiple Dimensions Arrays in Python Numpy.

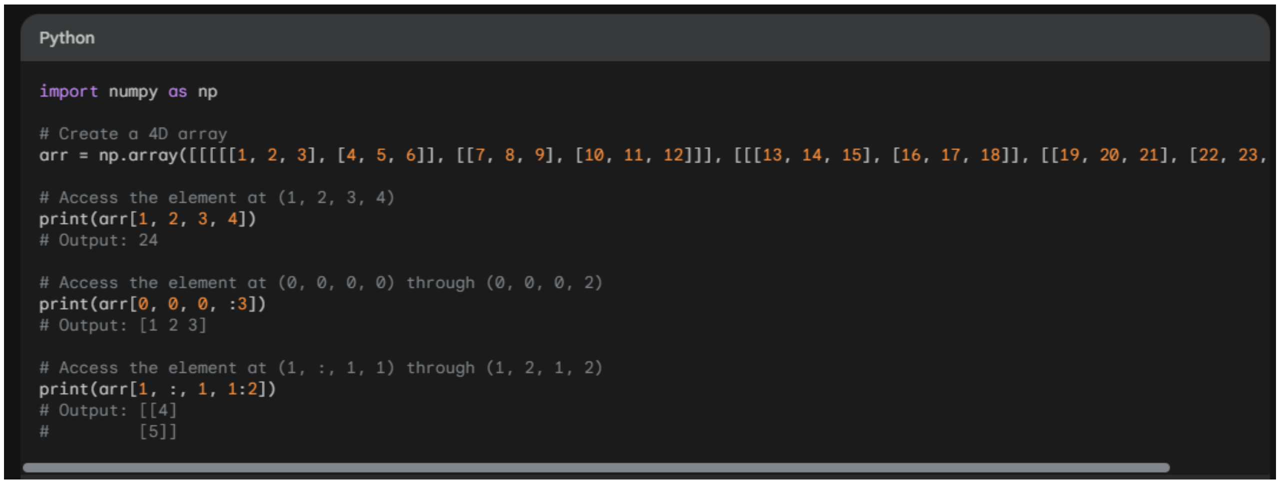


Python

```python
import numpy as np

# Create a 3D array
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])

# Access the element at (1, 2, 3)
print(arr[1, 2, 3])
# Output: 9

# Access the element at (0, 0, 0) through (0, 0, 2)
print(arr[0, 0, :3])
# Output: [1 2 3]

# Access the element at (1, :, 1) through (1, 2, 1)
print(arr[1, :, 1:2])
# Output: [[4]
#          [5]]
```

46. Implement in Python to accessing Data Along Multiple Dimensions Arrays in Python

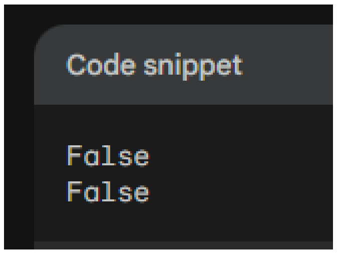
Numpy.



```
import numpy as np

# Create a 4D array
arr = np.array([[[[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]], [[[13, 14, 15], [16, 17, 18]], [[19, 20, 21], [22, 23,

# Access the element at (1, 2, 3, 4)
print(arr[1, 2, 3, 4])
# Output: 24

# Access the element at (0, 0, 0, 0) through (0, 0, 0, 2)
print(arr[0, 0, 0, :3])
# Output: [1 2 3]

# Access the element at (1, :, 1, 1) through (1, 2, 1, 2)
print(arr[1, :, 1, 1:2])
# Output: [[4]
#          [5]]
```

47. (a) Write the output of the below code?

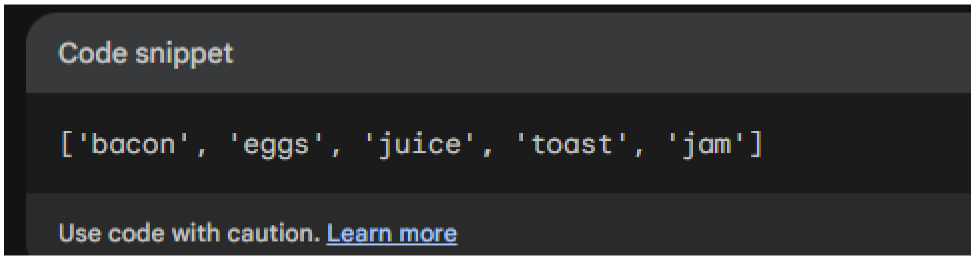
pset_time = 15
sleep_time = 8
print(sleep_time > pset_time)
derive = True
drink = False
both = drink and derive
print(both)



```
False
False
```

(b) Find the value of brunch after you execute all the operations in the code below?

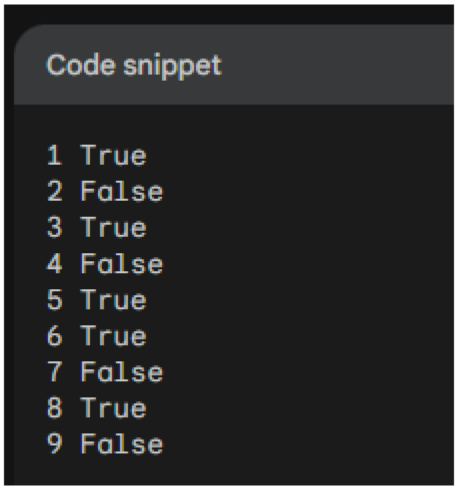
L1 = ["bacon", "eggs"]
L2 = ["toast", "jam"]
brunch = L1
L1.append("juice")
brunch.extend(L2)



```
['bacon', 'eggs', 'juice', 'toast', 'jam']
```

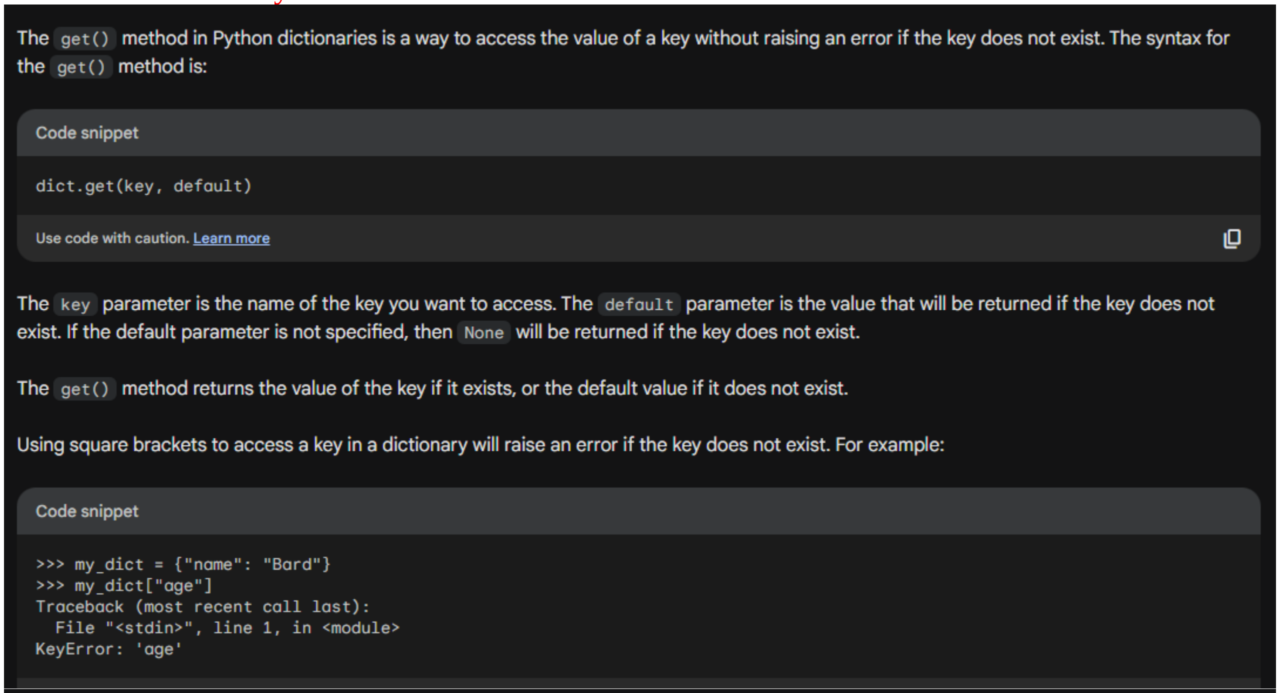
48. Evaluate the following python code and identify the output of each statement

a=6
b=7
c=42
print 1, a==6
print 2, a==7
print 3, a==6 and b==7
print 4, a==7 and b==7
print 5, not a==7 and b==7
print 6, a==7 or b==7
print 7, a==7 or b==6
print 8, not(a==7 and b==6)
print 9, not a==7 and b==6



```
True
False
True
False
True
True
False
True
False
```

49. Discuss the "get" method in python dictionaries, including its syntax, parameters, and output. How does it differ from using square brackets to access a key in a dictionary and when would you use one over the other?



The `get()` method in Python dictionaries is a way to access the value of a key without raising an error if the key does not exist. The syntax for the `get()` method is:

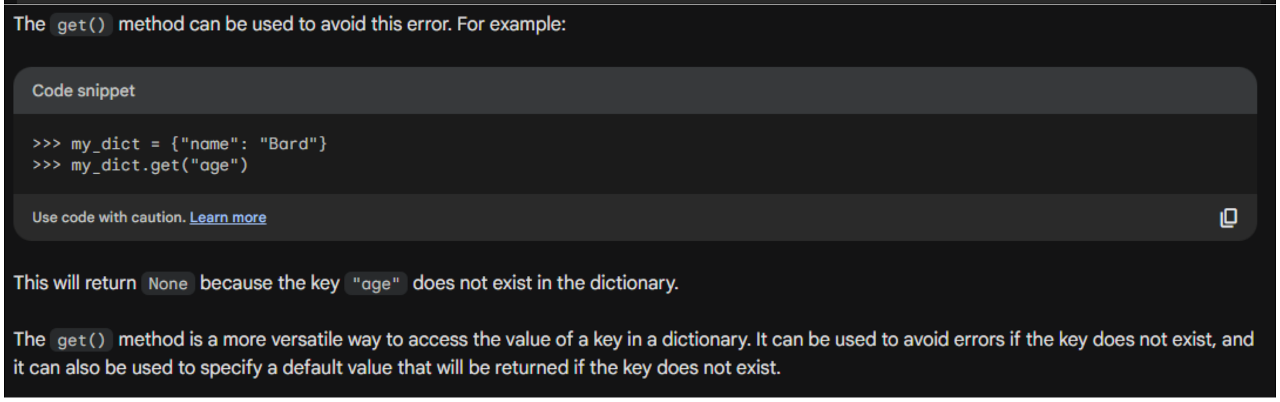
```
dict.get(key, default)
```

The `key` parameter is the name of the key you want to access. The `default` parameter is the value that will be returned if the key does not exist. If the default parameter is not specified, then `None` will be returned if the key does not exist.

The `get()` method returns the value of the key if it exists, or the default value if it does not exist.

Using square brackets to access a key in a dictionary will raise an error if the key does not exist. For example:

```
>>> my_dict = {"name": "Bard"}
>>> my_dict["age"]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'age'
```



The `get()` method can be used to avoid this error. For example:

```
>>> my_dict = {"name": "Bard"}
>>> my_dict.get("age")
```

This will return `None` because the key `"age"` does not exist in the dictionary.

The `get()` method is a more versatile way to access the value of a key in a dictionary. It can be used to avoid errors if the key does not exist, and it can also be used to specify a default value that will be returned if the key does not exist.