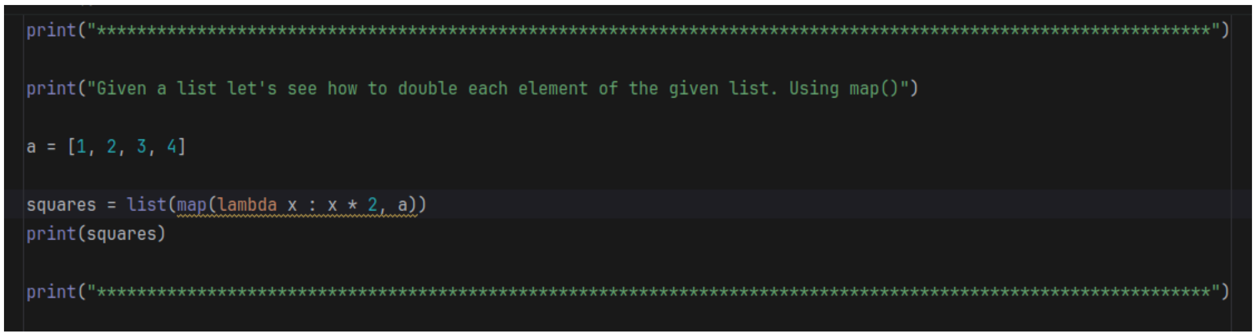
LAMBDA, MAP, FILTER,REDUCE

LAMBDA, MAP, FILTER,REDUCE

- Given a list let's see how to double each element of the given list. Using map()

a = [1, 2, 3, 4]

#Expected Output: [2, 4, 6, 8]

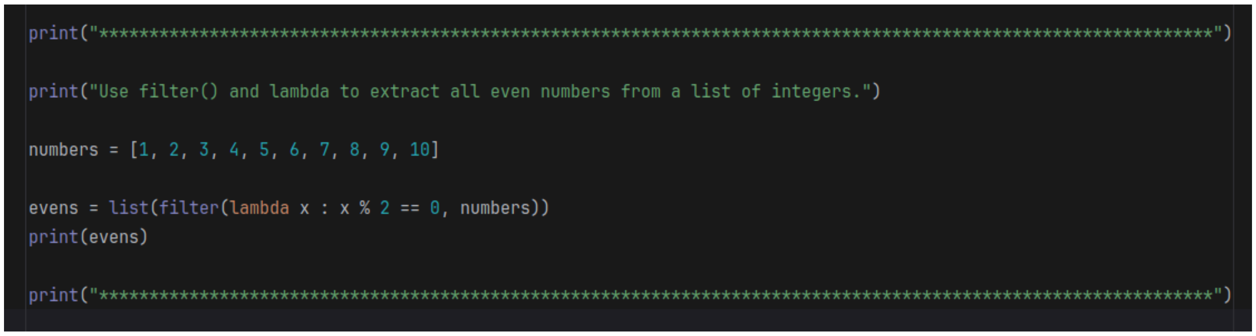


```
print("****************************************************************************************************************")

print("Given a list let's see how to double each element of the given list. Using map()")

a = [1, 2, 3, 4]

squares = list(map(lambda x : x * 2, a))
print(squares)

print("****************************************************************************************************************")
```

- Use `filter()` and `lambda` to extract all even numbers from a list of integers.

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

#Expected Output: [2, 4, 6, 8, 10]



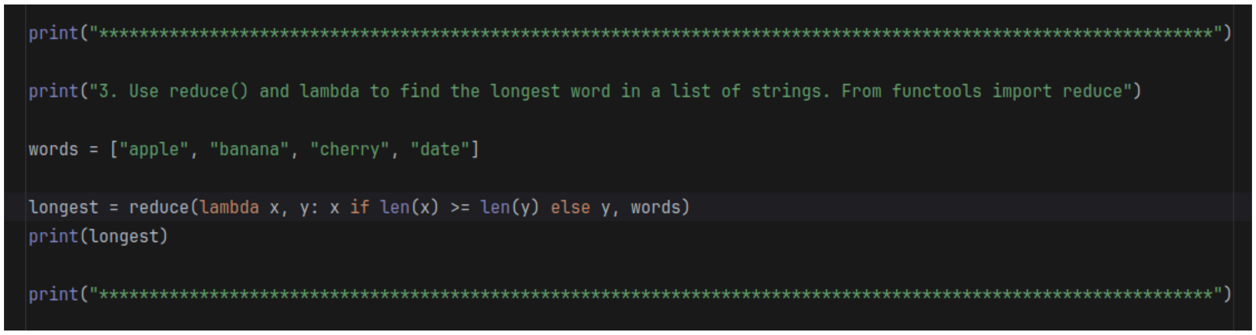
```
print("****************************************************************************************************************")

print("Use filter() and lambda to extract all even numbers from a list of integers.")

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

evens = list(filter(lambda x : x % 2 == 0, numbers))
print(evens)

print("****************************************************************************************************************")
```

- Use `reduce()` and `lambda` to find the longest word in a list of strings.

from functools import reduce

words = ["apple", "banana", "cherry", "date"]

#Expected Output: 'banana'

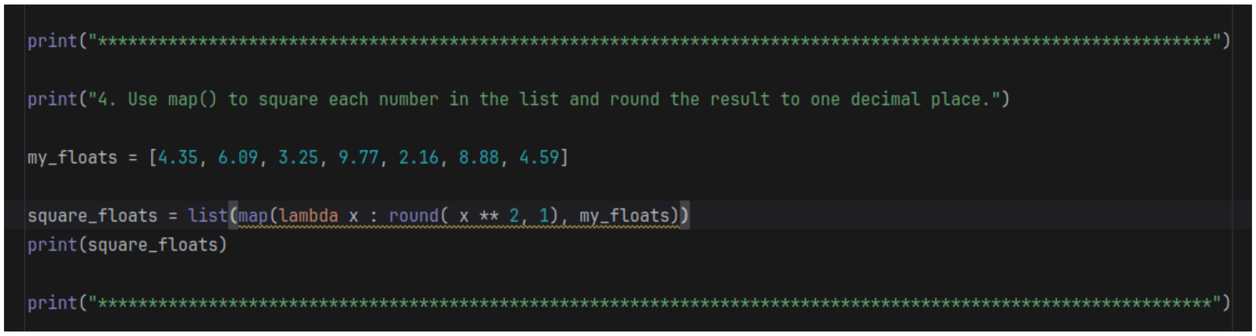


```python
print("***************************************************************************************************************")

print("3. Use reduce() and lambda to find the longest word in a list of strings. From functools import reduce")

words = ["apple", "banana", "cherry", "date"]

longest = reduce(lambda x, y: x if len(x) >= len(y) else y, words)
print(longest)

print("***************************************************************************************************************")
```

- Use `map()` to square each number in the list and round the result to one decimal place.

my_floats = [4.35, 6.09, 3.25, 9.77, 2.16, 8.88, 4.59]

#Expected Output: [18.9, 37.1, 10.6, 95.5, 4.7, 78.9, 21.1]

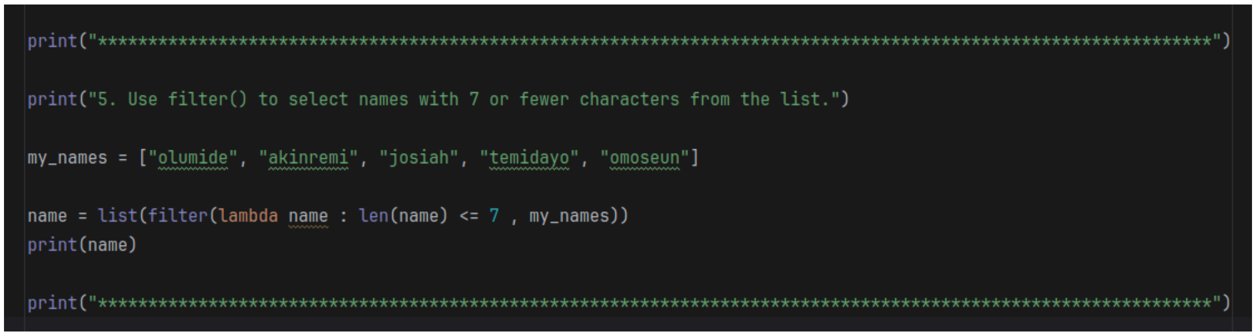


```python
print("***************************************************************************************************************")

print("4. Use map() to square each number in the list and round the result to one decimal place.")

my_floats = [4.35, 6.09, 3.25, 9.77, 2.16, 8.88, 4.59]

square_floats = list(map(lambda x : round( x ** 2, 1), my_floats))
print(square_floats)

print("***************************************************************************************************************")
```

- Use `filter()` to select names with 7 or fewer characters from the list.

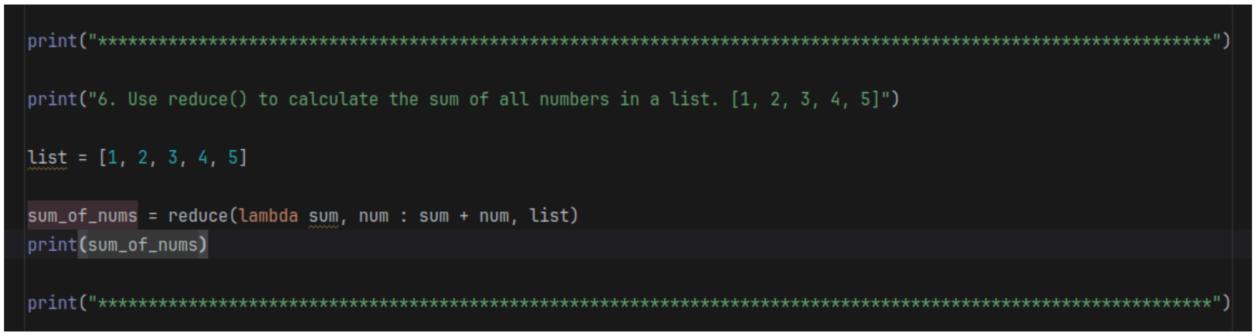
my_names = ["olumide", "akinremi", "josiah", "temidayo", "omoseun"]

#Expected Output: ['olumide', 'josiah', 'omoseun']



```python
print("***************************************************************************************************************")

print("5. Use filter() to select names with 7 or fewer characters from the list.")

my_names = ["olumide", "akinremi", "josiah", "temidayo", "omoseun"]

name = list(filter(lambda name : len(name) <= 7 , my_names))
print(name)

print("***************************************************************************************************************")
```

- Use `reduce()` to calculate the sum of all numbers in a list. [1, 2, 3, 4, 5]



```python
print("***************************************************************************************************************")

print("6. Use reduce() to calculate the sum of all numbers in a list. [1, 2, 3, 4, 5]")

list = [1, 2, 3, 4, 5]

sum_of_nums = reduce(lambda sum, num : sum + num, list)
print(sum_of_nums)

print("***************************************************************************************************************")
```

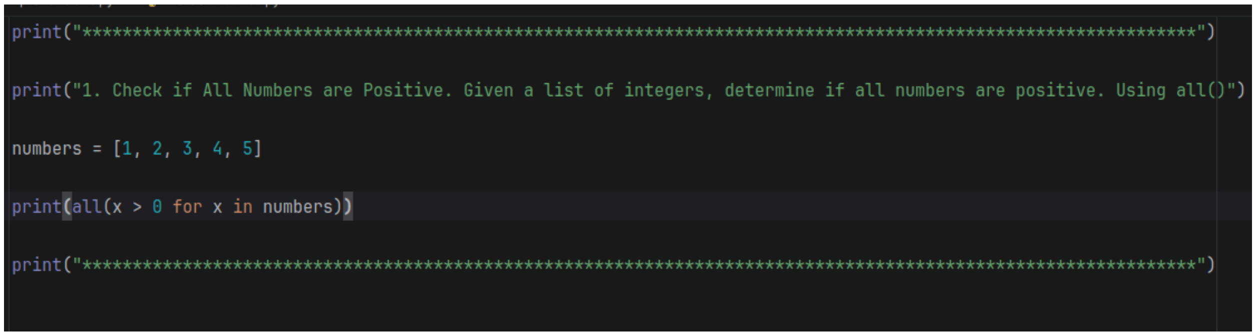
ALL AND ANY

ALL AND ANY

1. Check if All Numbers are Positive. Given a list of integers, determine if all numbers are positive. Using all()

   Input : numbers = [1, 2, 3, 4, 5]

   #Expected Output : True

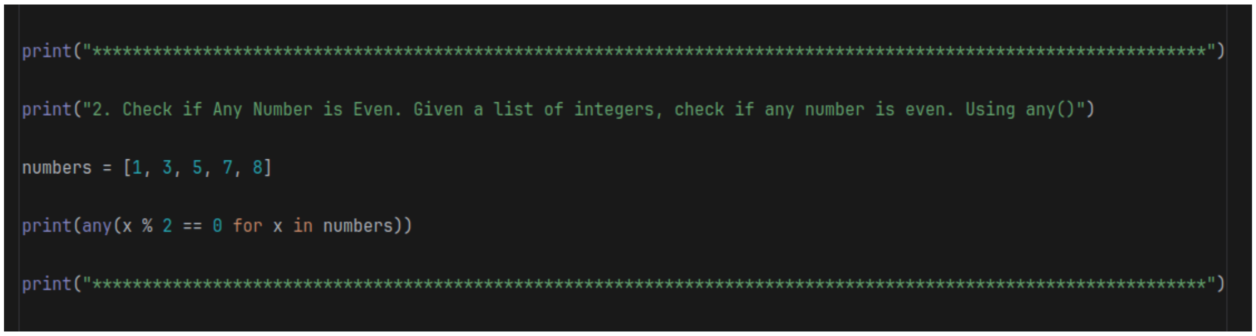


```python
print("*****************************************************************************************************")

print("1. Check if All Numbers are Positive. Given a list of integers, determine if all numbers are positive. Using all()")

numbers = [1, 2, 3, 4, 5]

print(all(x > 0 for x in numbers))

print("*****************************************************************************************************")
```

2. Check if Any Number is Even. Given a list of integers, check if any number is even. Using any()

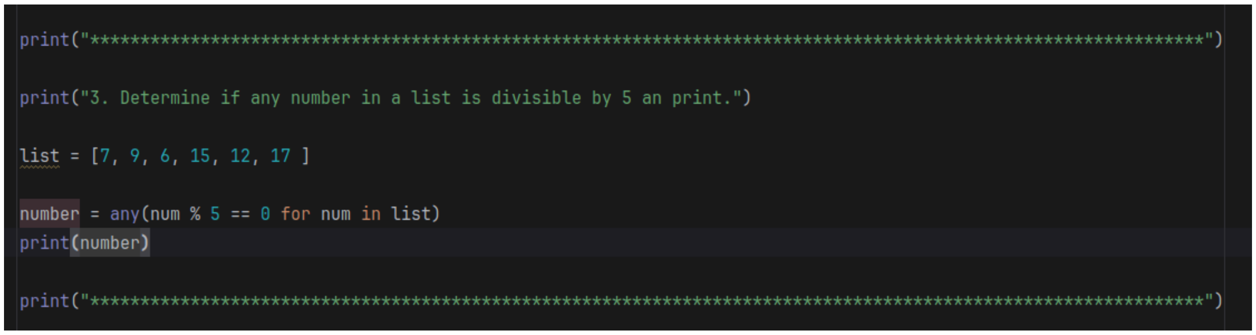
   Input: numbers = [1, 3, 5, 7, 8]

   #Expected Output: True



```python
print("*****************************************************************************************************")

print("2. Check if Any Number is Even. Given a list of integers, check if any number is even. Using any()")

numbers = [1, 3, 5, 7, 8]

print(any(x % 2 == 0 for x in numbers))

print("*****************************************************************************************************")
```

3. Determine if any number in a list is divisible by 5 an print.



```python
print("***************************************************************************************************************")

print("3. Determine if any number in a list is divisible by 5 an print.")

list = [7, 9, 6, 15, 12, 17 ]

number = any(num % 5 == 0 for num in list)
print(number)

print("***************************************************************************************************************")
```

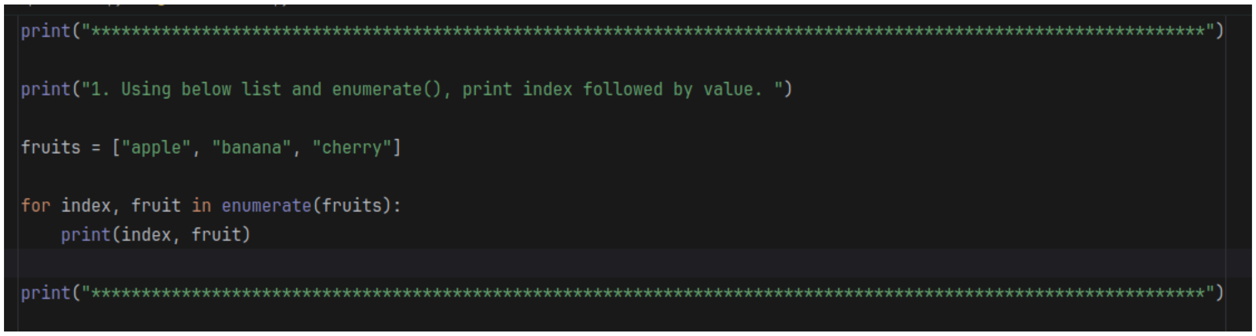
ENUMERATE

ENUMERATE

1. Using below list and enumerate(), print index followed by value.

   Input: fruits = ["apple", "banana", "cherry"]
   Output:

   ```
   0 apple
   1 banana
   2 cherry
   ```



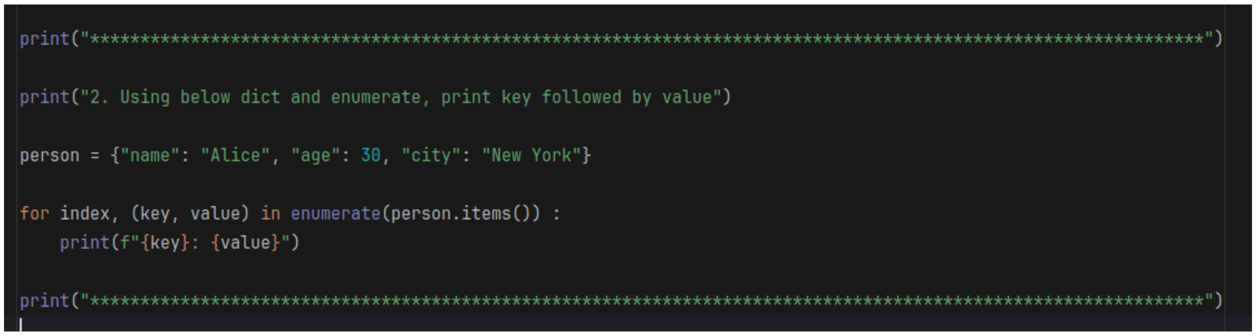
```python
print("****************************************************************************************************")

print("1. Using below list and enumerate(), print index followed by value. ")

fruits = ["apple", "banana", "cherry"]

for index, fruit in enumerate(fruits):
    print(index, fruit)

print("****************************************************************************************************")
```

2. Using below dict and enumerate, print key followed by value

   Input: person = {"name": "Alice", "age": 30, "city": "New York"}

   Output:

   ```
   name: Alice
   age: 30
   city: New York
   ```

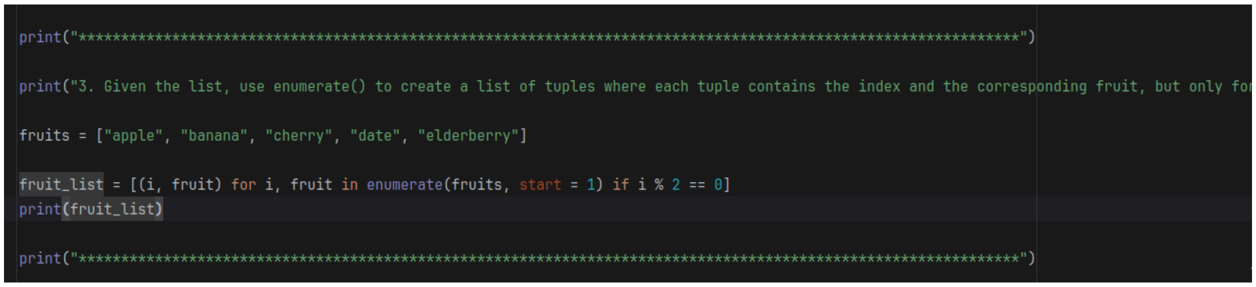


```python
print("****************************************************************************************************")

print("2. Using below dict and enumerate, print key followed by value")

person = {"name": "Alice", "age": 30, "city": "New York"}

for index, (key, value) in enumerate(person.items()) :
    print(f"{key}: {value}")

print("****************************************************************************************************")
```

3. Given the list `fruits = ["apple", "banana", "cherry", "date", "elderberry"]`, use `enumerate()` to create a list of tuples where each tuple contains the index and the corresponding fruit, but only for even indices.

   Output:

   ```
   [(2, 'banana'), (4, 'date')]
   ```

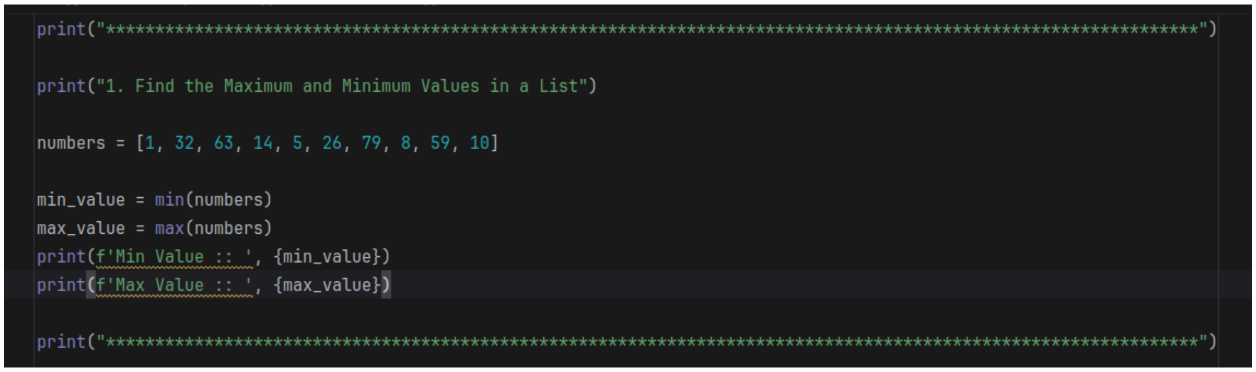


```python
print("***************************************************************************************************************")

print("3. Given the list, use enumerate() to create a list of tuples where each tuple contains the index and the corresponding fruit, but only fo

fruits = ["apple", "banana", "cherry", "date", "elderberry"]

fruit_list = [(i, fruit) for i, fruit in enumerate(fruits, start = 1) if i % 2 == 0]
print(fruit_list)

print("***************************************************************************************************************")
```

MAX AND MIN

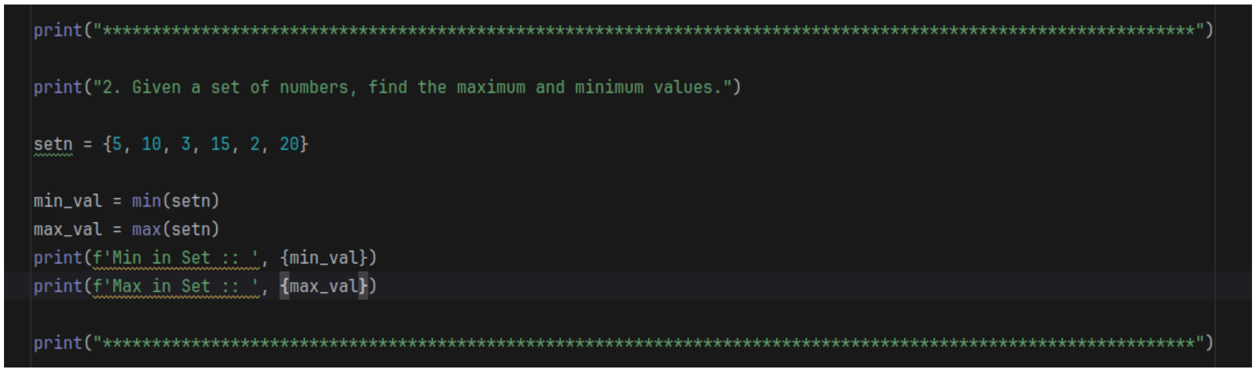
MAX AND MIN

1. Find the Maximum and Minimum Values in a List
   numbers = [1, 32, 63, 14, 5, 26, 79, 8, 59, 10]



```python
print("***************************************************************************************************************")

print("1. Find the Maximum and Minimum Values in a List")

numbers = [1, 32, 63, 14, 5, 26, 79, 8, 59, 10]

min_value = min(numbers)
max_value = max(numbers)
print(f'Min Value :: ', {min_value})
print(f'Max Value :: ', {max_value})

print("***************************************************************************************************************")
```

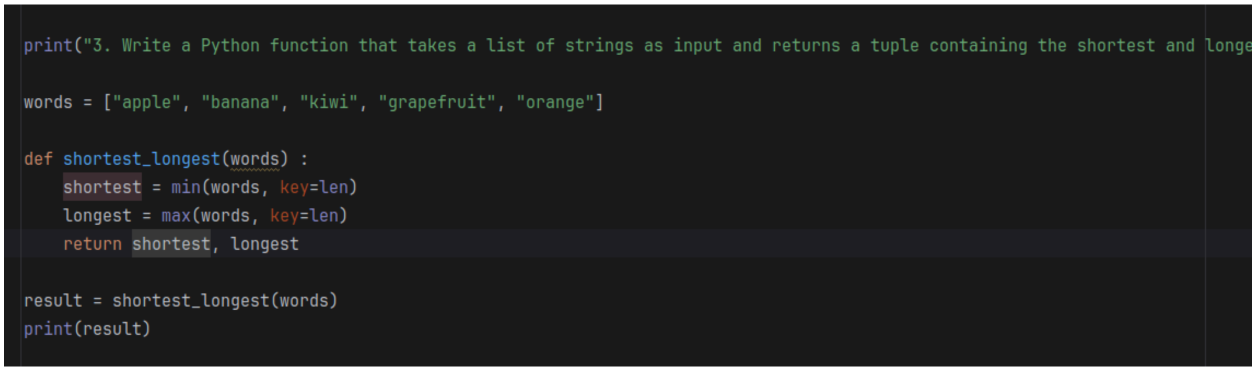
2. Given a set of numbers, find the maximum and minimum values.
   setn = {5, 10, 3, 15, 2, 20}



```python
print("***************************************************************************************************************")

print("2. Given a set of numbers, find the maximum and minimum values.")

setn = {5, 10, 3, 15, 2, 20}

min_val = min(setn)
max_val = max(setn)
print(f'Min in Set :: ', {min_val})
print(f'Max in Set :: ', {max_val})

print("***************************************************************************************************************")
```

3. Write a Python function that takes a list of strings as input and returns a tuple containing the shortest and longest word from the list, in that order. If there are multiple words of the same shortest or longest length, return the first shortest/longest word found.

   Input: words = ["apple", "banana", "kiwi", "grapefruit", "orange"]
   Output: ('kiwi', 'grapefruit')

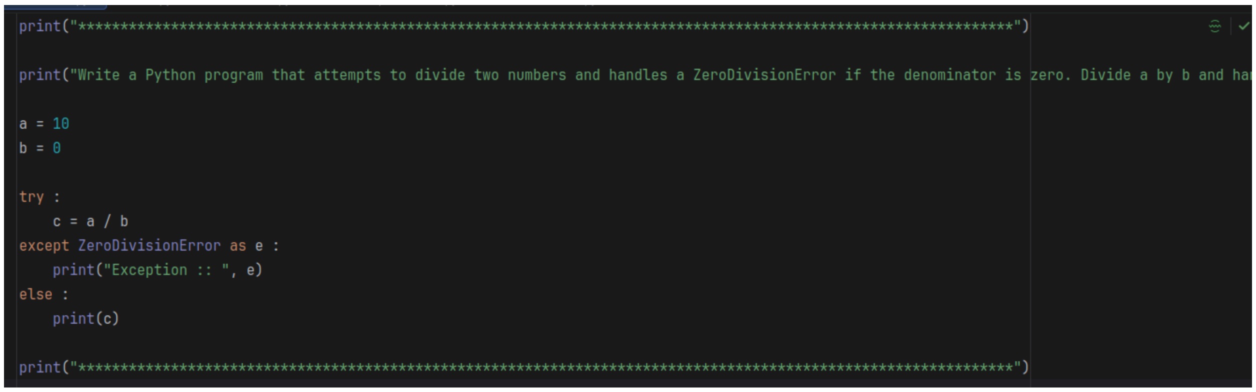


```python
print("3. Write a Python function that takes a list of strings as input and returns a tuple containing the shortest and longe

words = ["apple", "banana", "kiwi", "grapefruit", "orange"]

def shortest_longest(words) :
    shortest = min(words, key=len)
    longest = max(words, key=len)
    return shortest, longest

result = shortest_longest(words)
print(result)
```

EXCEPTION HANDLING

EXCEPTION HANDLING

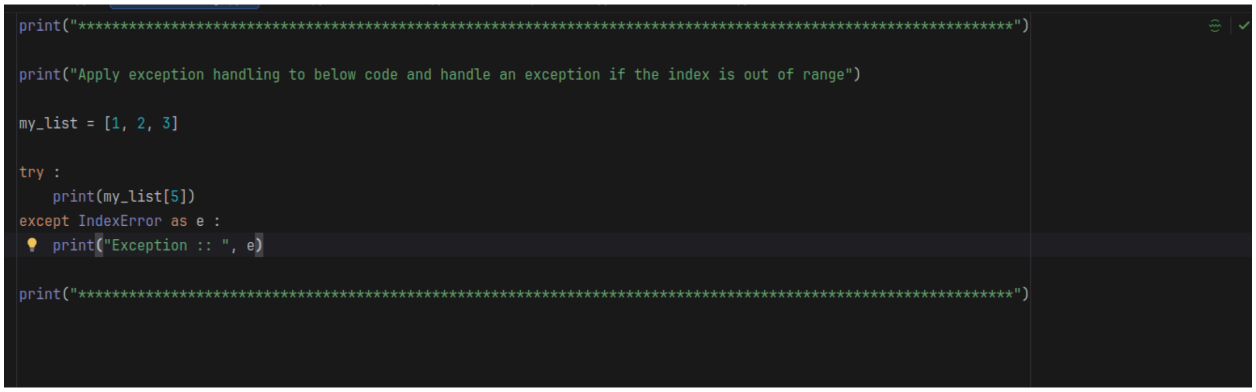
1. Write a Python program that attempts to divide two numbers a = 10 b = 0 and handles a `ZeroDivisionError` if the denominator is zero. Divide a by b and handle the exception and print the error



```
print("**********************************************************************************************************")

print("Write a Python program that attempts to divide two numbers and handles a ZeroDivisionError if the denominator is zero. Divide a by b and ha

a = 10
b = 0

try :
    c = a / b
except ZeroDivisionError as e :
    print("Exception :: ", e)
else :
    print(c)

print("**********************************************************************************************************")
```

2. Apply exception handling to below code and handle an exception if the index is out of range.

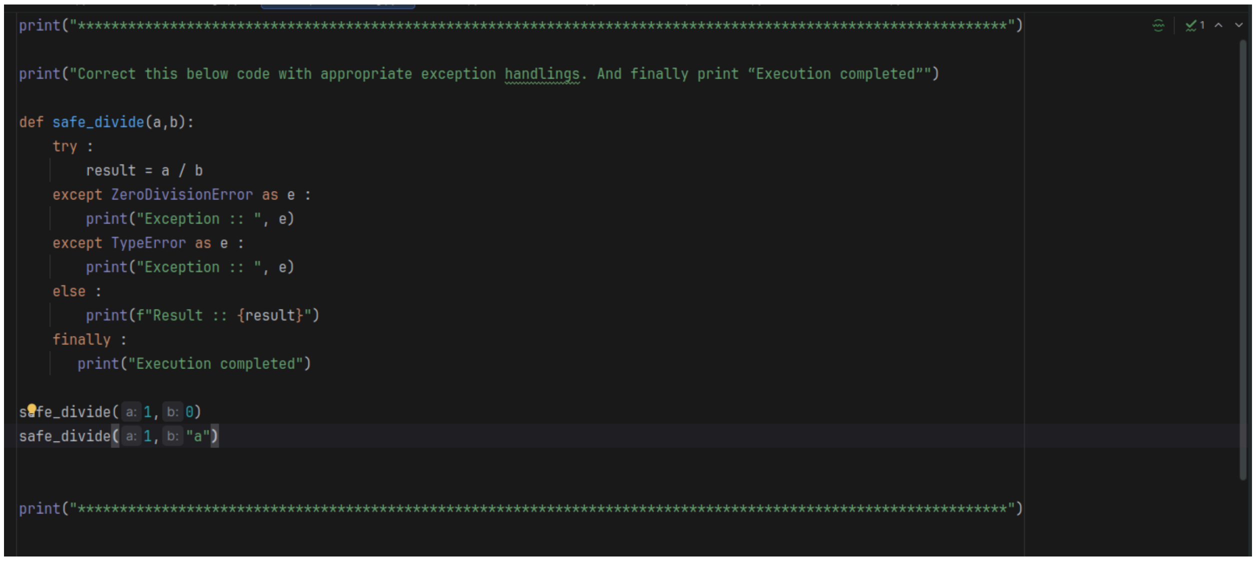
```
my_list = [1, 2, 3]
print(my_list[5])
```



```
print("**********************************************************************************************************")

print("Apply exception handling to below code and handle an exception if the index is out of range")

my_list = [1, 2, 3]

try :
    print(my_list[5])
except IndexError as e :
    print("Exception :: ", e)

print("**********************************************************************************************************")
```

3. Correct this below code with appropriate exception handlings. And finally print "Execution completed"

```
def safe_divide(a,b):
    result = a / b
    print(f"Result: {result}")

safe_divide(1,0)
safe_divide(1,"a")
```

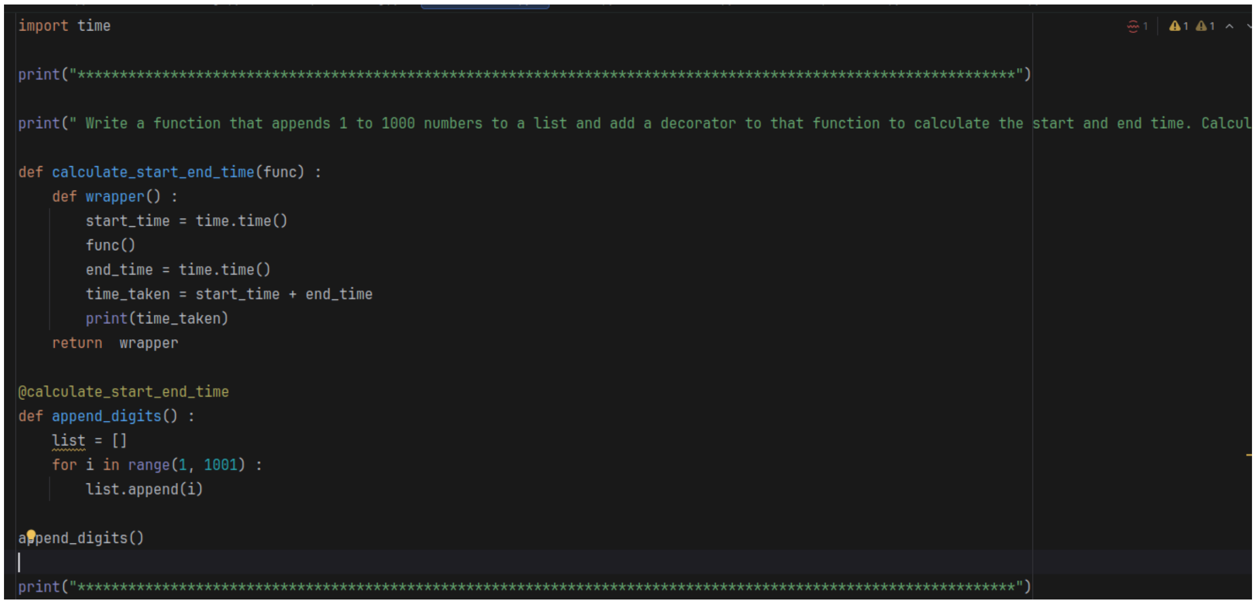


```
print("*****************************************************************************************************************")

print("Correct this below code with appropriate exception handlings. And finally print “Execution completed”")

def safe_divide(a,b):
    try :
        result = a / b
    except ZeroDivisionError as e :
        print("Exception :: ", e)
    except TypeError as e :
        print("Exception :: ", e)
    else :
        print(f"Result :: {result}")
    finally :
       print("Execution completed")

safe_divide( a: 1, b: 0)
safe_divide( a: 1, b: "a")


print("*****************************************************************************************************************")
```

DECORATOR

DECORATOR

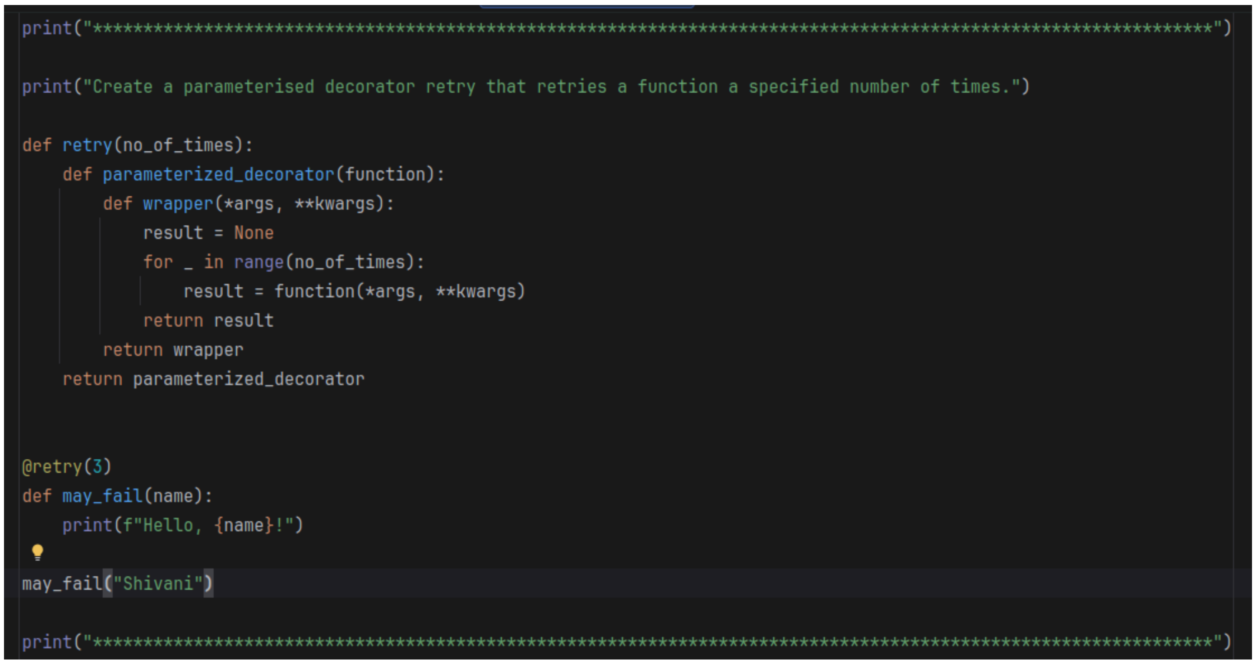
1. Write a function that appends 1 to 1000 numbers to a list and add a decorator to that function to calculate the start and end time. Calculate the total time taken and print.



```python
import time

print("**************************************************************************************************")

print(" Write a function that appends 1 to 1000 numbers to a list and add a decorator to that function to calculate the start and end time. Calcul

def calculate_start_end_time(func) :
    def wrapper() :
        start_time = time.time()
        func()
        end_time = time.time()
        time_taken = start_time + end_time
        print(time_taken)
    return  wrapper

@calculate_start_end_time
def append_digits() :
    list = []
    for i in range(1, 1001) :
        list.append(i)

append_digits()

print("**************************************************************************************************")
```

2. Create a parameterised decorator `retry` that retries a function a specified number of times.

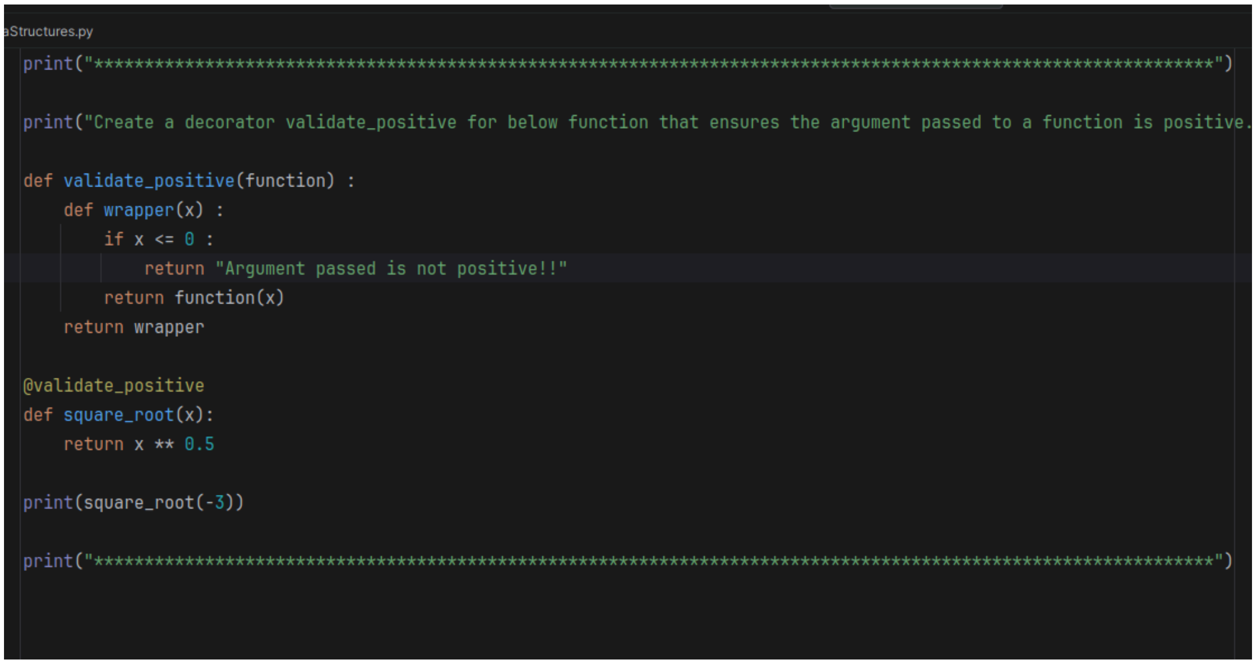
@retry(3)
def may_fail(name):
    print(f"Hello, {name}!")



```python
print("**************************************************************************************************")

print("Create a parameterised decorator retry that retries a function a specified number of times.")

def retry(no_of_times):
    def parameterized_decorator(function):
        def wrapper(*args, **kwargs):
            result = None
            for _ in range(no_of_times):
                result = function(*args, **kwargs)
            return result
        return wrapper
    return parameterized_decorator


@retry(3)
def may_fail(name):
    print(f"Hello, {name}!")

may_fail("Shivani")

print("**************************************************************************************************")
```

3. Create a decorator `validate_positive` for below function that ensures the argument passed to a function is positive.

```
def square_root(x):
    return x ** 0.5
```



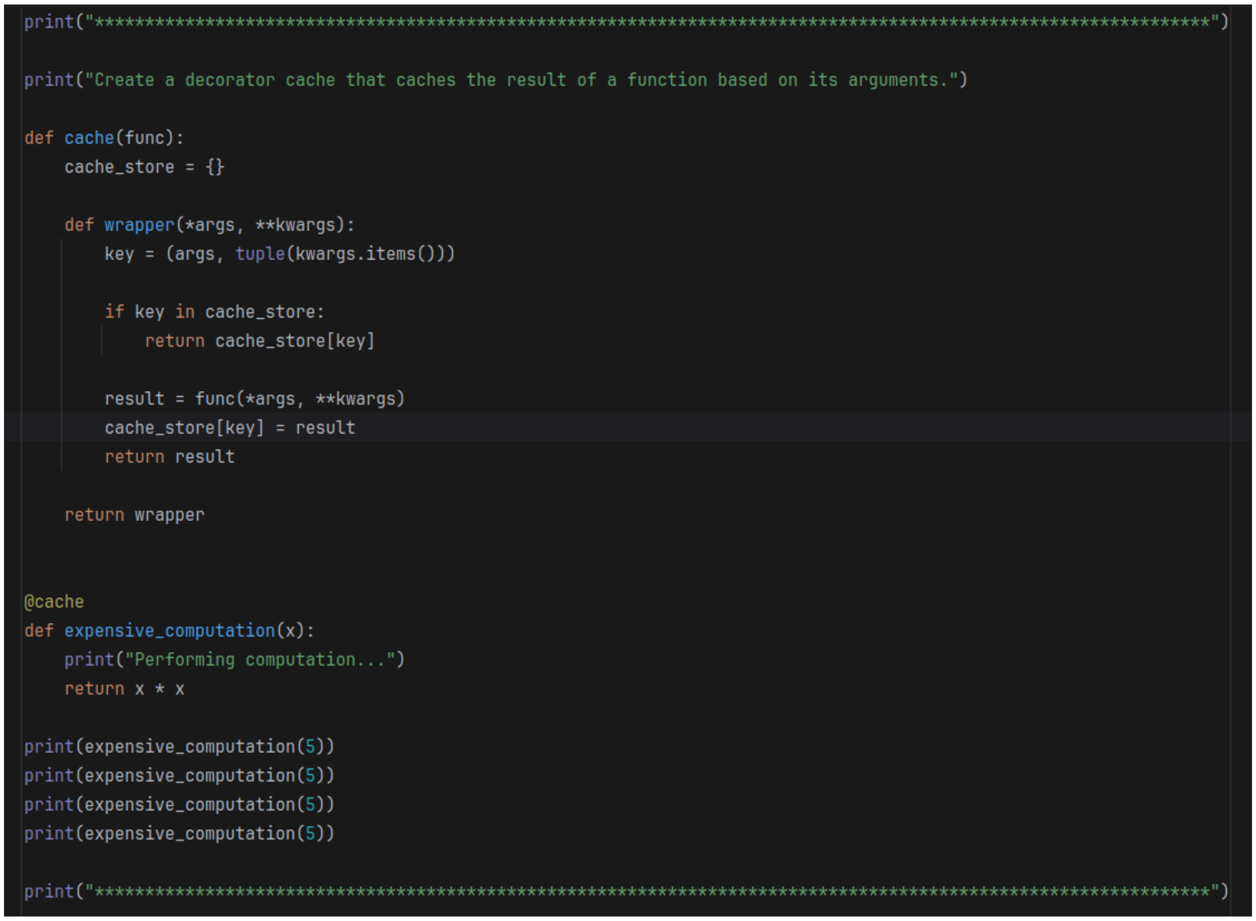
aStructures.py

```
print("***************************************************************************************************************")

print("Create a decorator validate_positive for below function that ensures the argument passed to a function is positive.

def validate_positive(function) :
    def wrapper(x) :
        if x <= 0 :
            return "Argument passed is not positive!!"
        return function(x)
    return wrapper

@validate_positive
def square_root(x):
    return x ** 0.5

print(square_root(-3))

print("***************************************************************************************************************")
```

4. Create a decorator `cache` that caches the result of a function based on its arguments.

```
@cache
def expensive_computation(x):
    print("Performing computation...")
    return x * x

expensive_computation(5)
expensive_computation(5)
```

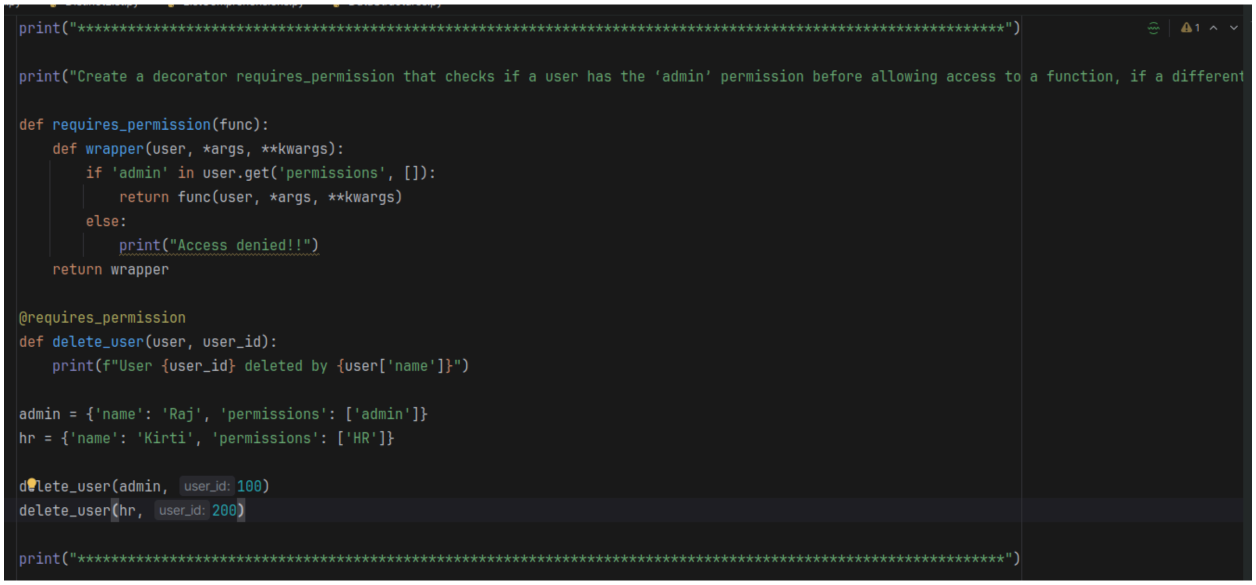
Write a cache decorator for it to check if the calculation is already performed then return the result.



```
print("**************************************************************************************************************")

print("Create a decorator cache that caches the result of a function based on its arguments.")

def cache(func):
    cache_store = {}

    def wrapper(*args, **kwargs):
        key = (args, tuple(kwargs.items()))

        if key in cache_store:
            return cache_store[key]

        result = func(*args, **kwargs)
        cache_store[key] = result
        return result

    return wrapper


@cache
def expensive_computation(x):
    print("Performing computation...")
    return x * x

print(expensive_computation(5))
print(expensive_computation(5))
print(expensive_computation(5))
print(expensive_computation(5))

print("**************************************************************************************************************")
```

5. Create a decorator `requires_permission` that checks if a user has the ‘admin’ permission before allowing access to a function, if a different user then responds “Access denied”.

```
 def delete_user(user, user_id):
     print(f"User {user_id} deleted by {user['name']}")

user1 = {'name': 'Alice', 'permissions': ['admin']}
user2 = {'name': John, 'permissions': ['dev']}
user3 = {'name': 'Kurt', 'permissions': ['test'']}
```



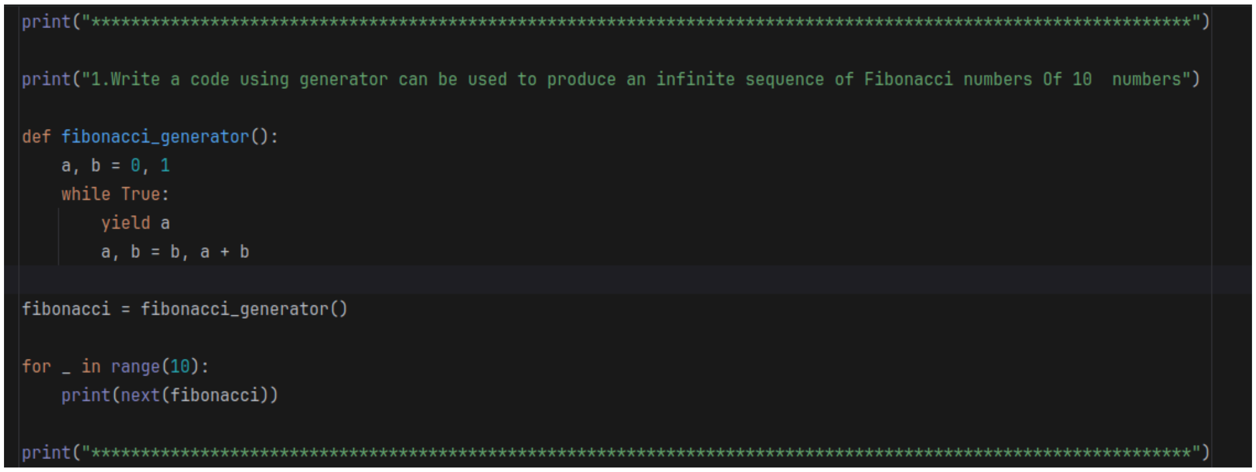
```
print("*****************************************************************************************************************")

print("Create a decorator requires_permission that checks if a user has the 'admin' permission before allowing access to a function, if a different

def requires_permission(func):
    def wrapper(user, *args, **kwargs):
        if 'admin' in user.get('permissions', []):
            return func(user, *args, **kwargs)
        else:
            print("Access denied!!")
    return wrapper

@requires_permission
def delete_user(user, user_id):
    print(f"User {user_id} deleted by {user['name']}")

admin = {'name': 'Raj', 'permissions': ['admin']}
hr = {'name': 'Kirti', 'permissions': ['HR']}

delete_user(admin, user_id: 100)
delete_user(hr, user_id: 200)

print("*****************************************************************************************************************")
```

GENERATOR

GENERATOR

1. Write a code using generator can be used to produce an infinite sequence of Fibonacci numbers Of 10 numbers

Output:
```
0
1
1
2
3
5
8
13
21
34
```

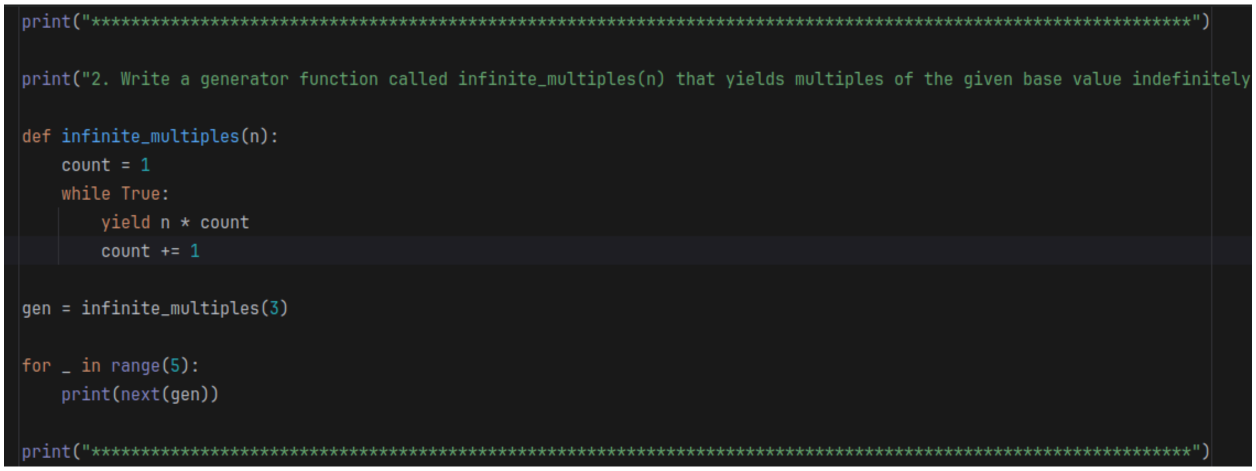


```python
print("***********************************************************************************************************")

print("1.Write a code using generator can be used to produce an infinite sequence of Fibonacci numbers Of 10  numbers")

def fibonacci_generator():
    a, b = 0, 1
    while True:
        yield a
        a, b = b, a + b

fibonacci = fibonacci_generator()

for _ in range(10):
    print(next(fibonacci))

print("***********************************************************************************************************")
```

2. Write a generator function called `infinite_multiples(n)` that yields multiples of the given base value indefinitely.

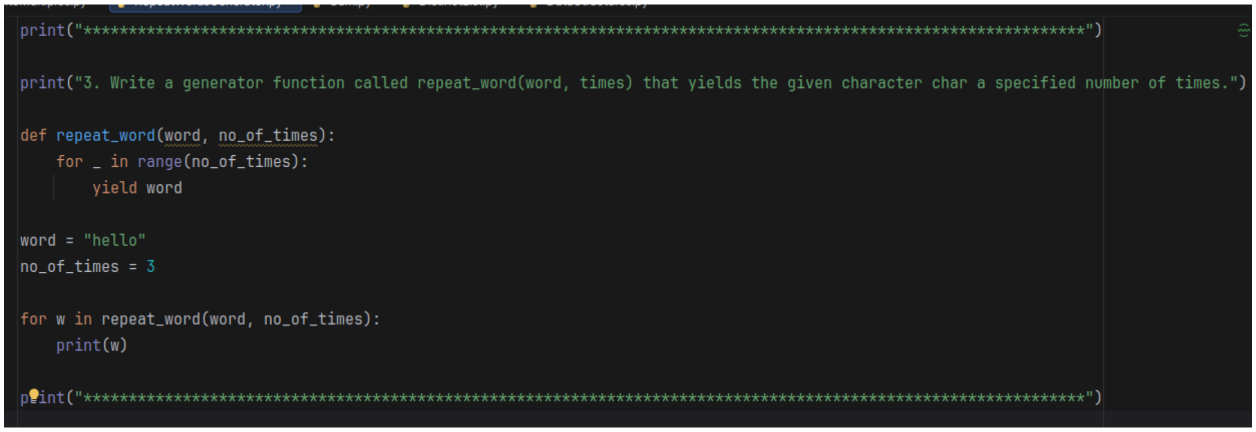
Input n=3

Output:
```
3
6
9
12
15
…
```


```python
print("***************************************************************************************************************")

print("2. Write a generator function called infinite_multiples(n) that yields multiples of the given base value indefinitely

def infinite_multiples(n):
    count = 1
    while True:
        yield n * count
        count += 1

gen = infinite_multiples(3)

for _ in range(5):
    print(next(gen))

print("***************************************************************************************************************")
```

3. Write a generator function called repeat_word(word, times) that yields the given character char a specified number of times.

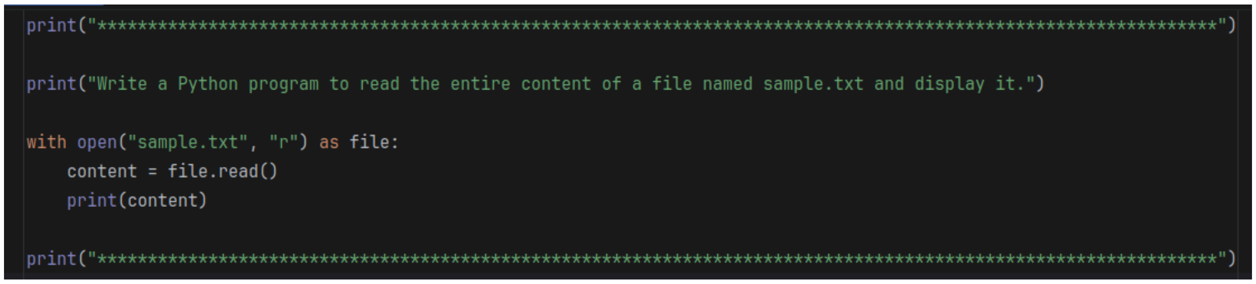
word = “hello”
times = 5


```python
print("***************************************************************************************************************")

print("3. Write a generator function called repeat_word(word, times) that yields the given character char a specified number of times.")

def repeat_word(word, no_of_times):
    for _ in range(no_of_times):
        yield word

word = "hello"
no_of_times = 3

for w in repeat_word(word, no_of_times):
    print(w)

print("***************************************************************************************************************")
```

FILE HANDLING

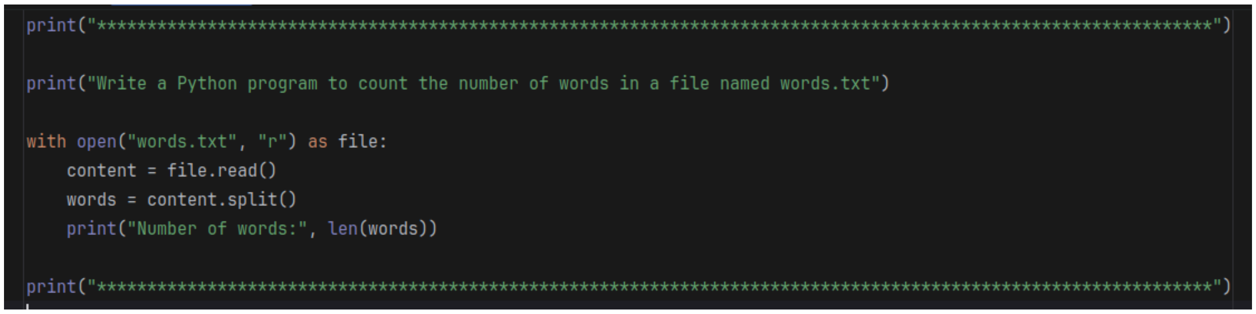
FILE HANDLING

1 . Write a Python program to read the entire content of a file named `sample.txt` and display it.



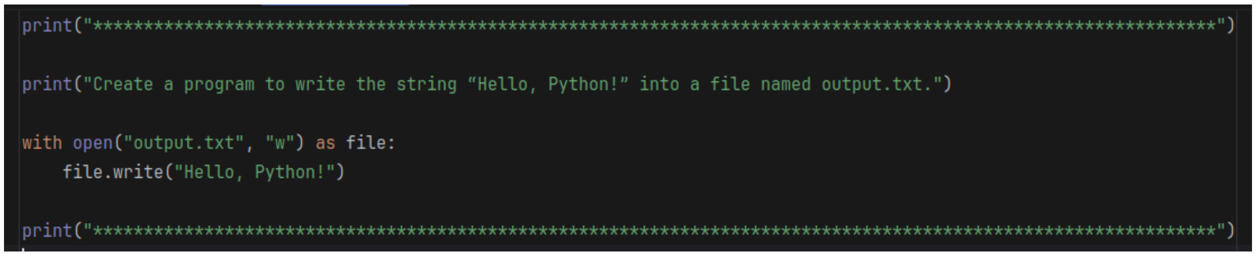
```python
print("****************************************************************************************************")

print("Write a Python program to read the entire content of a file named sample.txt and display it.")

with open("sample.txt", "r") as file:
    content = file.read()
    print(content)

print("****************************************************************************************************")
```

2. Write a Python program to count the number of words in a file named `words.txt`



```python
print("****************************************************************************************************")

print("Write a Python program to count the number of words in a file named words.txt")

with open("words.txt", "r") as file:
    content = file.read()
    words = content.split()
    print("Number of words:", len(words))

print("****************************************************************************************************")
```

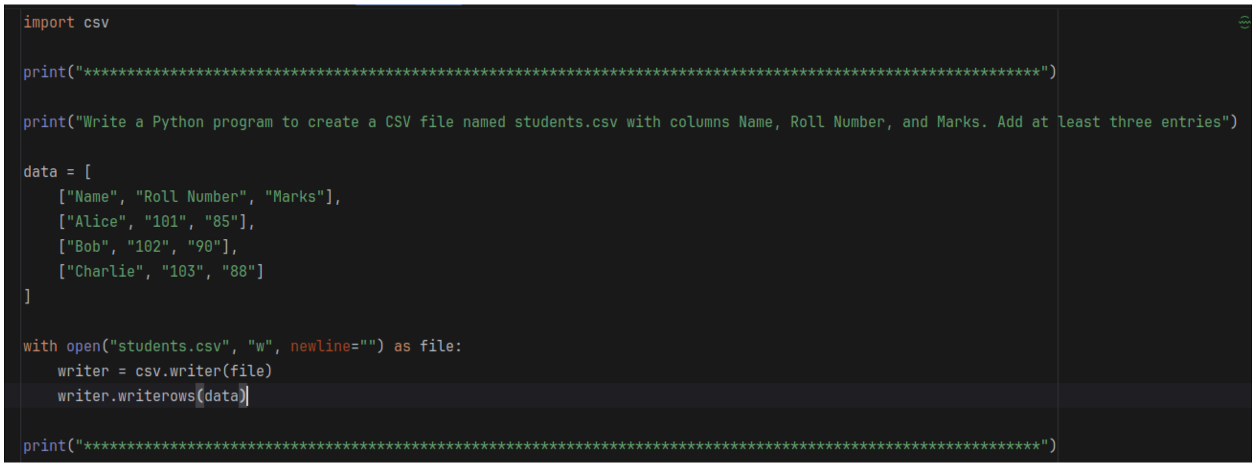
3 . Create a program to write the string “Hello, Python!” into a file named output.txt.



```python
print("****************************************************************************************************")

print("Create a program to write the string “Hello, Python!” into a file named output.txt.")

with open("output.txt", "w") as file:
    file.write("Hello, Python!")

print("****************************************************************************************************")
```

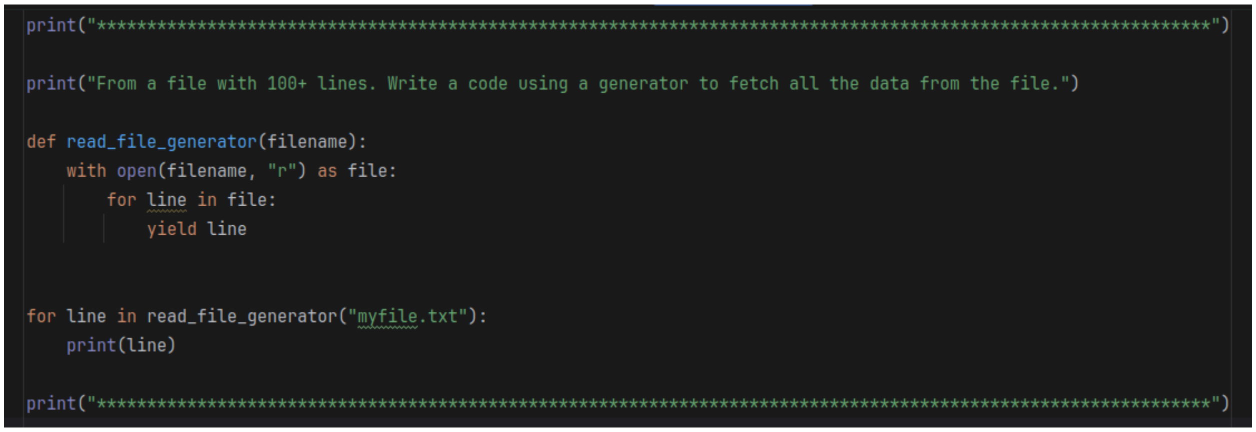
4. Write a Python program to create a CSV file named `students.csv` with columns `Name`, `Roll Number`, and `Marks`. Add at least three entries

```
    data = [
  ["Name", "Roll Number", "Marks"],
  ["Alice", "101", "85"],
  ["Bob", "102", "90"],
  ["Charlie", "103", "88"]
]
```



```python
import csv

print("****************************************************************************************************************")

print("Write a Python program to create a CSV file named students.csv with columns Name, Roll Number, and Marks. Add at least three entries")

data = [
    ["Name", "Roll Number", "Marks"],
    ["Alice", "101", "85"],
    ["Bob", "102", "90"],
    ["Charlie", "103", "88"]
]

with open("students.csv", "w", newline="") as file:
    writer = csv.writer(file)
    writer.writerows(data)

print("****************************************************************************************************************")
```

5. From a file with 100+ lines. Write a code using a generator to fetch all the data from the file.

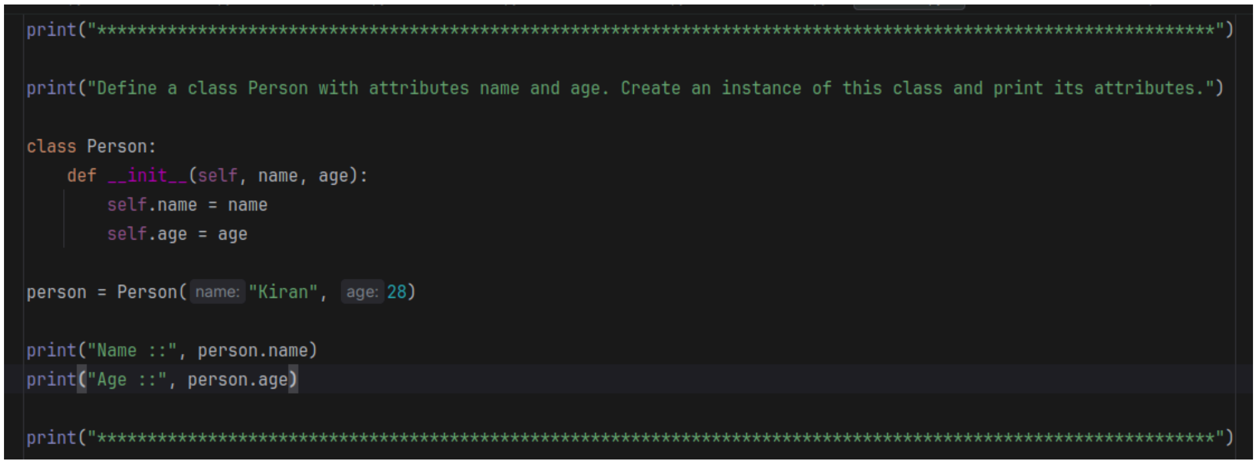


```python
print("****************************************************************************************************************")

print("From a file with 100+ lines. Write a code using a generator to fetch all the data from the file.")

def read_file_generator(filename):
    with open(filename, "r") as file:
        for line in file:
            yield line


for line in read_file_generator("myfile.txt"):
    print(line)

print("****************************************************************************************************************")
```

CLASS

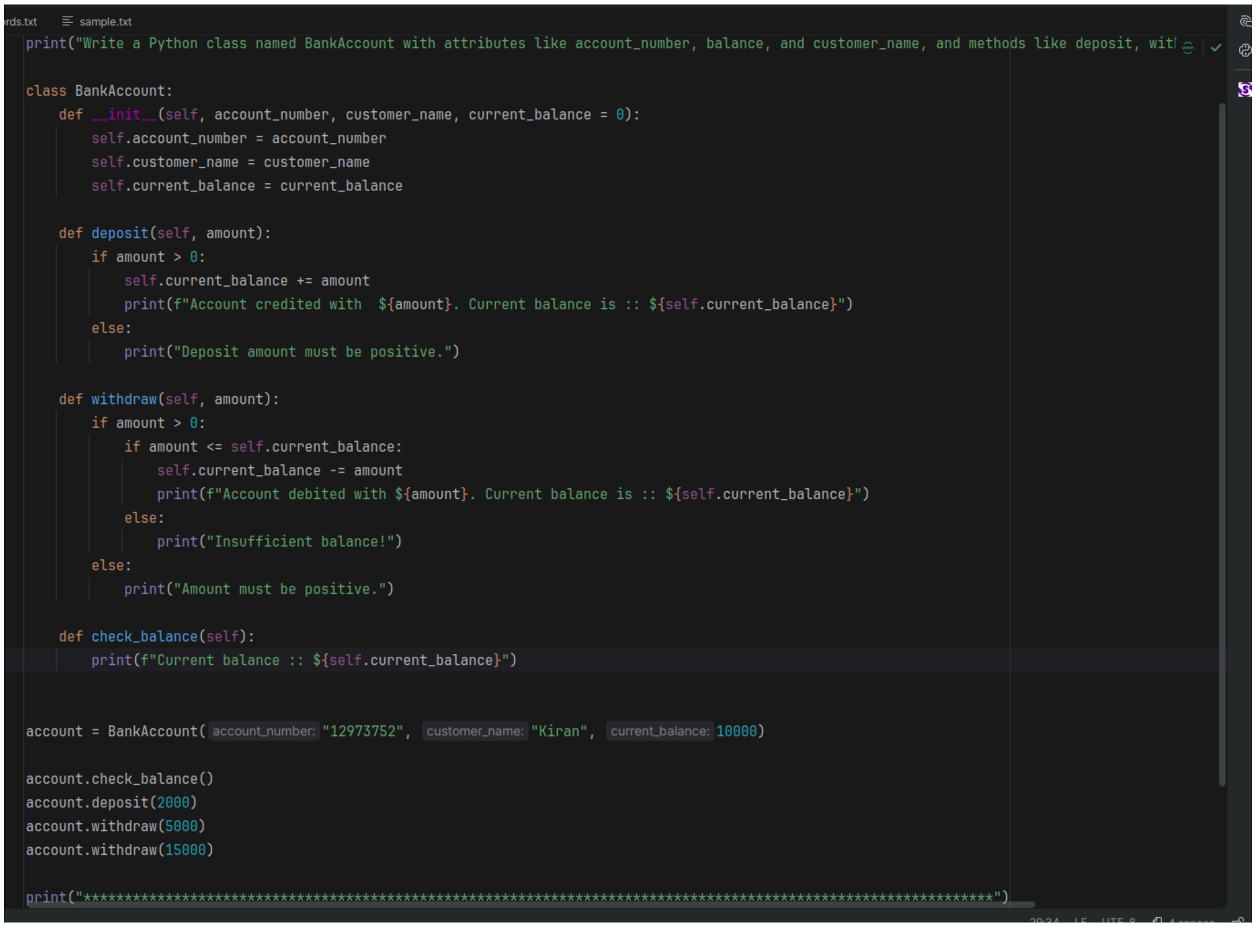
CLASS

1. Define a class `Person` with attributes `name` and `age`. Create an instance of this class and print its attributes.



```
print("*****************************************************************************************************************")

print("Define a class Person with attributes name and age. Create an instance of this class and print its attributes.")

class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

person = Person( name: "Kiran",  age: 28)

print("Name ::", person.name)
print("Age ::", person.age)

print("*****************************************************************************************************************")
```

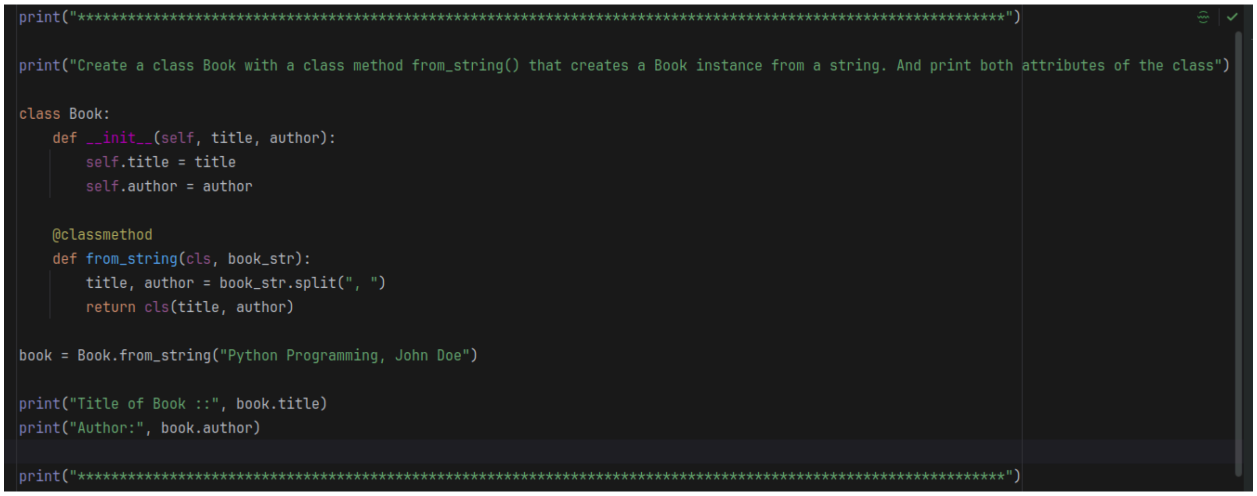
2. **Problem:** Write a Python class named `BankAccount` with attributes like `account_number`, `balance`, and `customer_name`, and methods like `deposit`, `withdraw`, and `check_balance`.



```
print("Write a Python class named BankAccount with attributes like account_number, balance, and customer_name, and methods like deposit, wit

class BankAccount:
    def __init__(self, account_number, customer_name, current_balance = 0):
        self.account_number = account_number
        self.customer_name = customer_name
        self.current_balance = current_balance

    def deposit(self, amount):
        if amount > 0:
            self.current_balance += amount
            print(f"Account credited with  ${amount}. Current balance is :: ${self.current_balance}")
        else:
            print("Deposit amount must be positive.")

    def withdraw(self, amount):
        if amount > 0:
            if amount <= self.current_balance:
                self.current_balance -= amount
                print(f"Account debited with ${amount}. Current balance is :: ${self.current_balance}")
            else:
                print("Insufficient balance!")
        else:
            print("Amount must be positive.")

    def check_balance(self):
        print(f"Current balance :: ${self.current_balance}")


account = BankAccount( account_number: "12973752", customer_name: "Kiran", current_balance: 10000)

account.check_balance()
account.deposit(2000)
account.withdraw(5000)
account.withdraw(15000)

print("*****************************************************************************************************************")
```

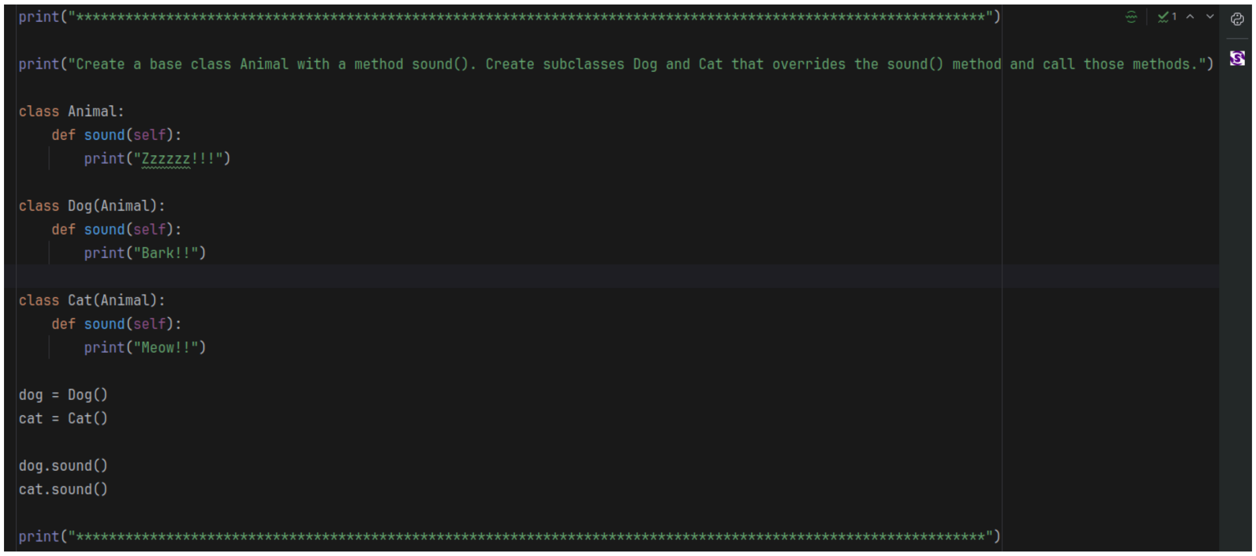
3. Create a class Book with a class method from_string() that creates a Book instance from a string. And print both attributes of the class

book = Book.from_string("Python Programming, John Doe")



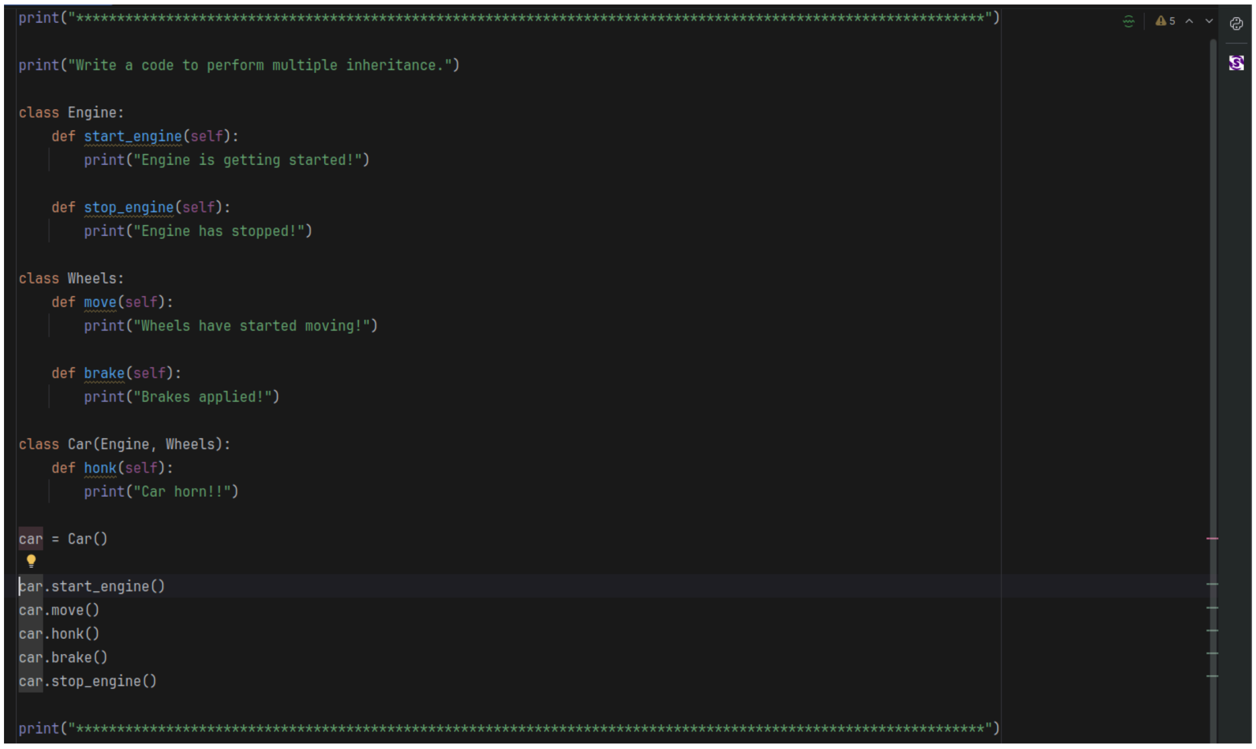
```
print("*****************************************************************************************************************")

print("Create a class Book with a class method from_string() that creates a Book instance from a string. And print both attributes of the class")

class Book:
    def __init__(self, title, author):
        self.title = title
        self.author = author

    @classmethod
    def from_string(cls, book_str):
        title, author = book_str.split(", ")
        return cls(title, author)

book = Book.from_string("Python Programming, John Doe")

print("Title of Book ::", book.title)
print("Author:", book.author)

print("*****************************************************************************************************************")
```

4. Create a base class `Animal` with a method `sound()`. Create subclasses `Dog and Cat` that overrides the `sound()` method and call those methods.



```python
print("****************************************************************************************************")

print("Create a base class Animal with a method sound(). Create subclasses Dog and Cat that overrides the sound() method and call those methods.")

class Animal:
    def sound(self):
        print("Zzzzzz!!!")

class Dog(Animal):
    def sound(self):
        print("Bark!!")

class Cat(Animal):
    def sound(self):
        print("Meow!!")

dog = Dog()
cat = Cat()

dog.sound()
cat.sound()

print("****************************************************************************************************")
```

5. Write a code to perform multiple inheritance.

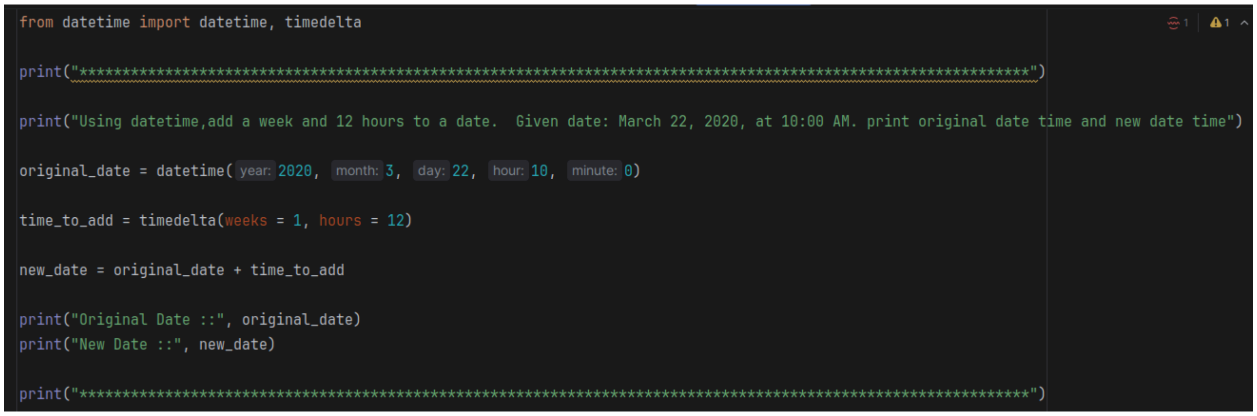


```python
print("****************************************************************************************************")

print("Write a code to perform multiple inheritance.")

class Engine:
    def start_engine(self):
        print("Engine is getting started!")

    def stop_engine(self):
        print("Engine has stopped!")

class Wheels:
    def move(self):
        print("Wheels have started moving!")

    def brake(self):
        print("Brakes applied!")

class Car(Engine, Wheels):
    def honk(self):
        print("Car horn!!")

car = Car()

car.start_engine()
car.move()
car.honk()
car.brake()
car.stop_engine()

print("****************************************************************************************************")
```

MODULE OS AND SYS DATA

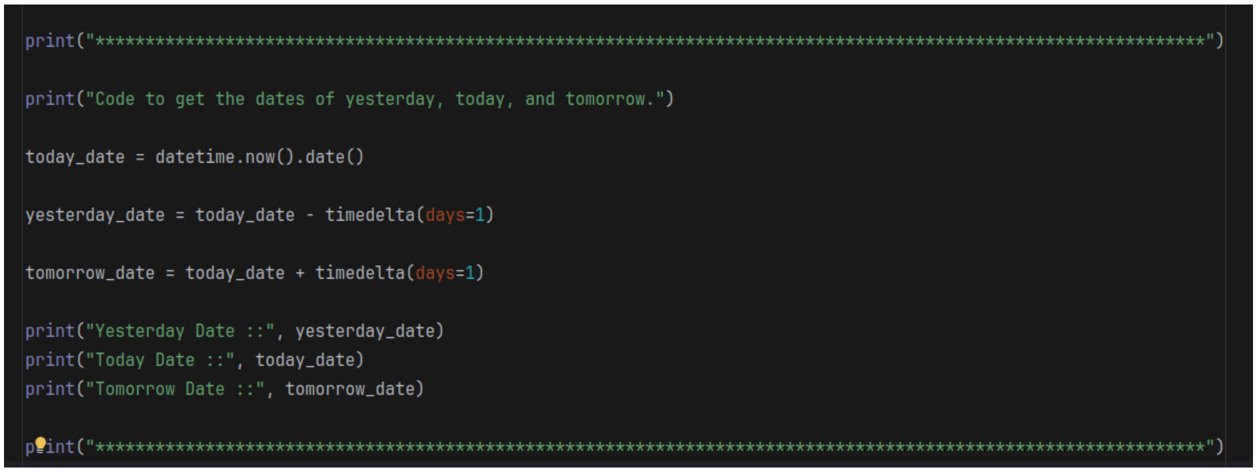
MODULE OS AND SYS DATA

1. Using datetime, add a week and 12 hours to a date. Given date: March 22, 2020, at 10:00 AM. print original date time and new date time.



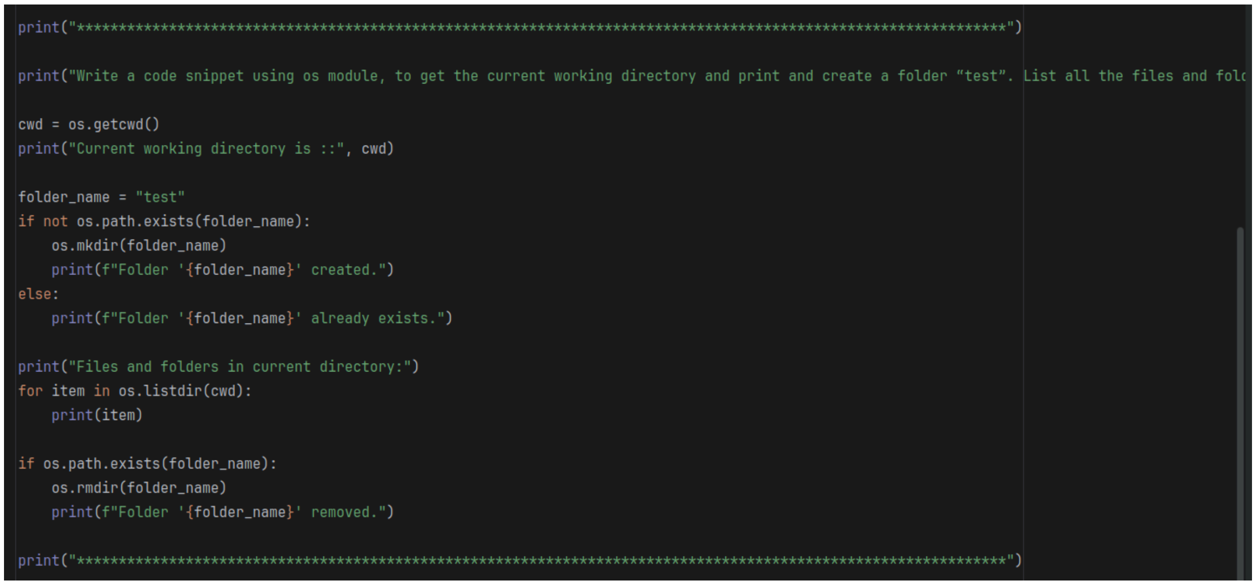
```python
from datetime import datetime, timedelta

print("****************************************************************************************************")

print("Using datetime,add a week and 12 hours to a date.  Given date: March 22, 2020, at 10:00 AM. print original date time and new date time")

original_date = datetime( year: 2020,  month: 3,  day: 22,  hour: 10,  minute: 0)

time_to_add = timedelta(weeks = 1, hours = 12)

new_date = original_date + time_to_add

print("Original Date ::", original_date)
print("New Date ::", new_date)

print("****************************************************************************************************")
```

2. Code to get the dates of yesterday, today, and tomorrow.



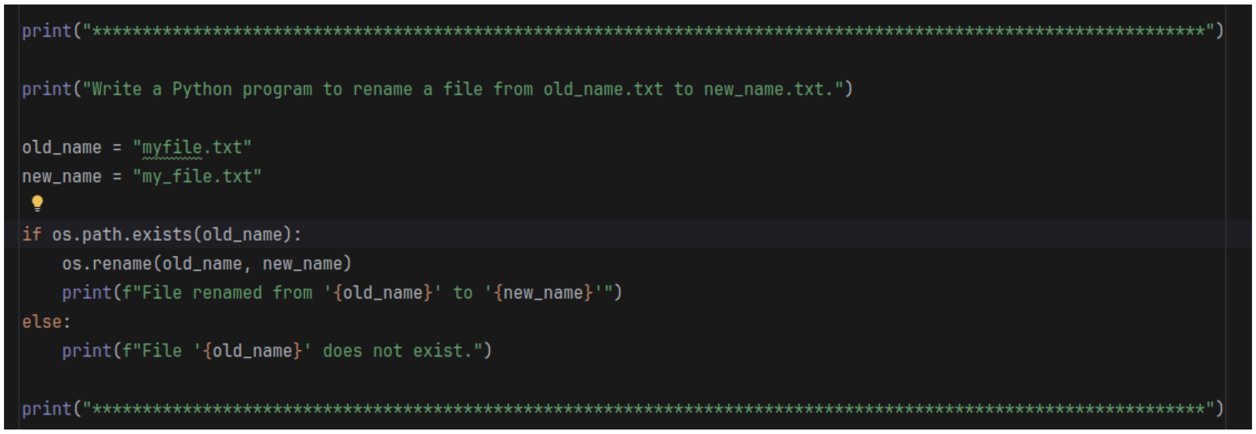
```python
print("****************************************************************************************************")

print("Code to get the dates of yesterday, today, and tomorrow.")

today_date = datetime.now().date()

yesterday_date = today_date - timedelta(days=1)

tomorrow_date = today_date + timedelta(days=1)

print("Yesterday Date ::", yesterday_date)
print("Today Date ::", today_date)
print("Tomorrow Date ::", tomorrow_date)

print("****************************************************************************************************")
```

3. Write a code snippet using os module, to get the current working directory and print and create a folder “test”. List all the files and folders in the current working directory and remove the directory “test” that was created.



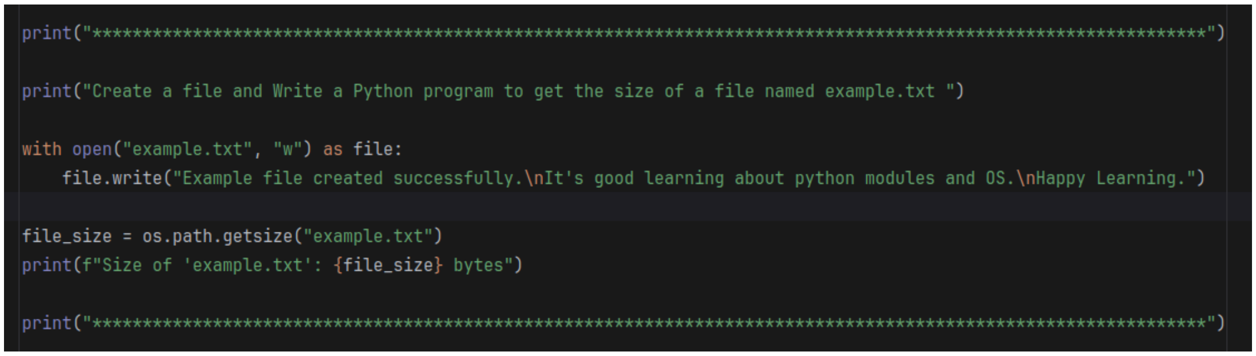
```python
print("***********************************************************************************************")

print("Write a code snippet using os module, to get the current working directory and print and create a folder “test”. List all the files and folc

cwd = os.getcwd()
print("Current working directory is ::", cwd)

folder_name = "test"
if not os.path.exists(folder_name):
    os.mkdir(folder_name)
    print(f"Folder '{folder_name}' created.")
else:
    print(f"Folder '{folder_name}' already exists.")

print("Files and folders in current directory:")
for item in os.listdir(cwd):
    print(item)

if os.path.exists(folder_name):
    os.rmdir(folder_name)
    print(f"Folder '{folder_name}' removed.")

print("***********************************************************************************************")
```

4. Write a Python program to rename a file from `old_name.txt` to `new_name.txt`.



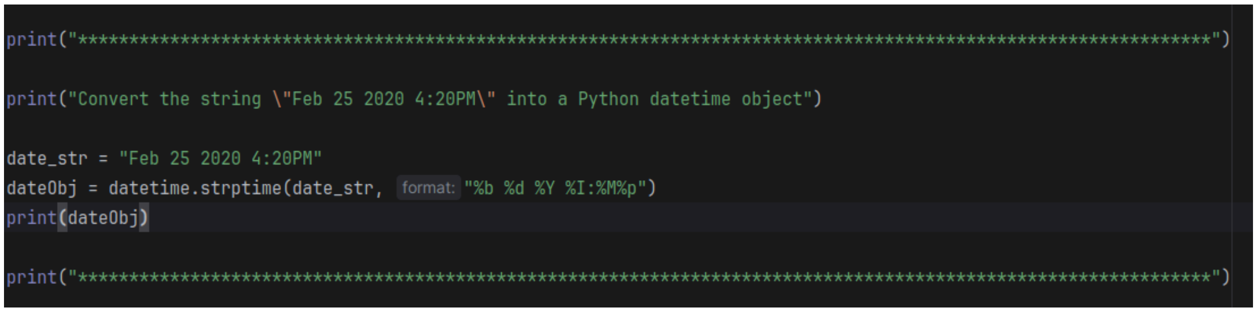
```python
print("***********************************************************************************************")

print("Write a Python program to rename a file from old_name.txt to new_name.txt.")

old_name = "myfile.txt"
new_name = "my_file.txt"

if os.path.exists(old_name):
    os.rename(old_name, new_name)
    print(f"File renamed from '{old_name}' to '{new_name}'")
else:
    print(f"File '{old_name}' does not exist.")

print("***********************************************************************************************")
```

5. Create a file and Write a Python program to get the size of a file named `example.txt`



```python
print("***********************************************************************************************")

print("Create a file and Write a Python program to get the size of a file named example.txt ")

with open("example.txt", "w") as file:
    file.write("Example file created successfully.\nIt's good learning about python modules and OS.\nHappy Learning.")

file_size = os.path.getsize("example.txt")
print(f"Size of 'example.txt': {file_size} bytes")

print("***********************************************************************************************")
```

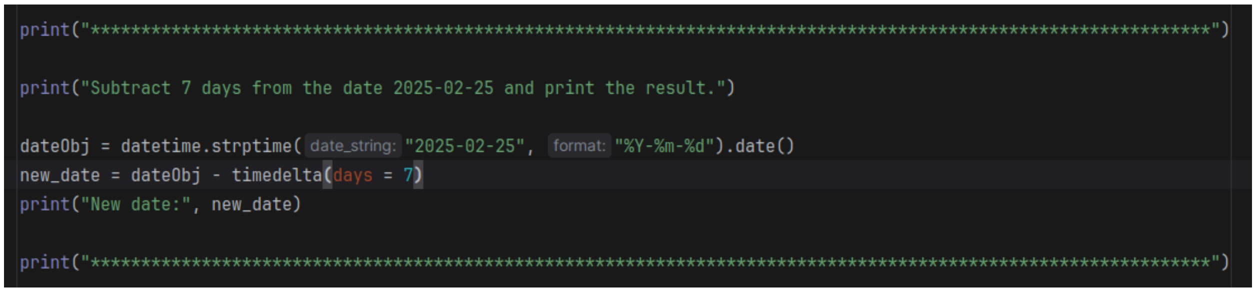
6. Convert the string "Feb 25 2020 4:20PM" into a Python datetime object
   O/P: 2020-02-25 16:20:00



```python
print("****************************************************************************************************************")

print("Convert the string \"Feb 25 2020 4:20PM\" into a Python datetime object")

date_str = "Feb 25 2020 4:20PM"
dateObj = datetime.strptime(date_str, format: "%b %d %Y %I:%M%p")
print(dateObj)

print("****************************************************************************************************************")
```

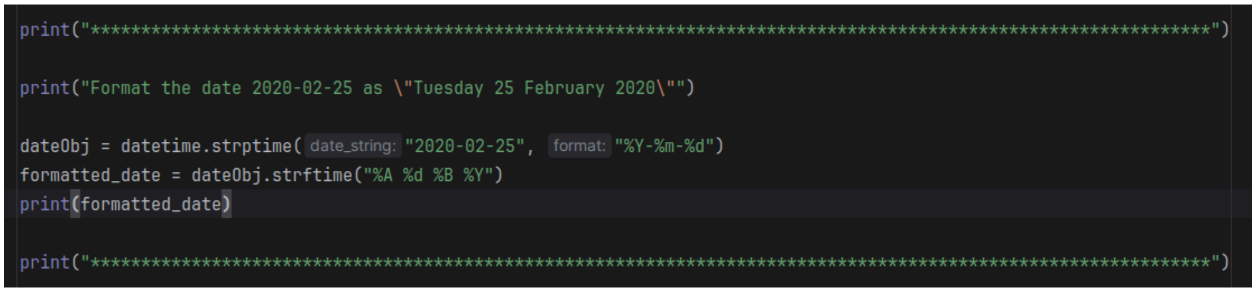
7. Subtract 7 days from the date `2025-02-25` and print the result.

   O/P: New date: 2025-02-18



```python
print("****************************************************************************************************************")

print("Subtract 7 days from the date 2025-02-25 and print the result.")

dateObj = datetime.strptime( date_string: "2025-02-25",  format: "%Y-%m-%d").date()
new_date = dateObj - timedelta(days = 7)
print("New date:", new_date)

print("****************************************************************************************************************")
```

8. Format the date `2020-02-25` as `"Tuesday 25 February 2020"`



```python
print("****************************************************************************************************************")

print("Format the date 2020-02-25 as \"Tuesday 25 February 2020\"")

dateObj = datetime.strptime( date_string: "2020-02-25",  format: "%Y-%m-%d")
formatted_date = dateObj.strftime("%A %d %B %Y")
print(formatted_date)

print("****************************************************************************************************************")
```