Day 9 – Python Session Day 2 – Accenture bootcamp

Name: Pothumulla Kankanamge Mewan Madhusha

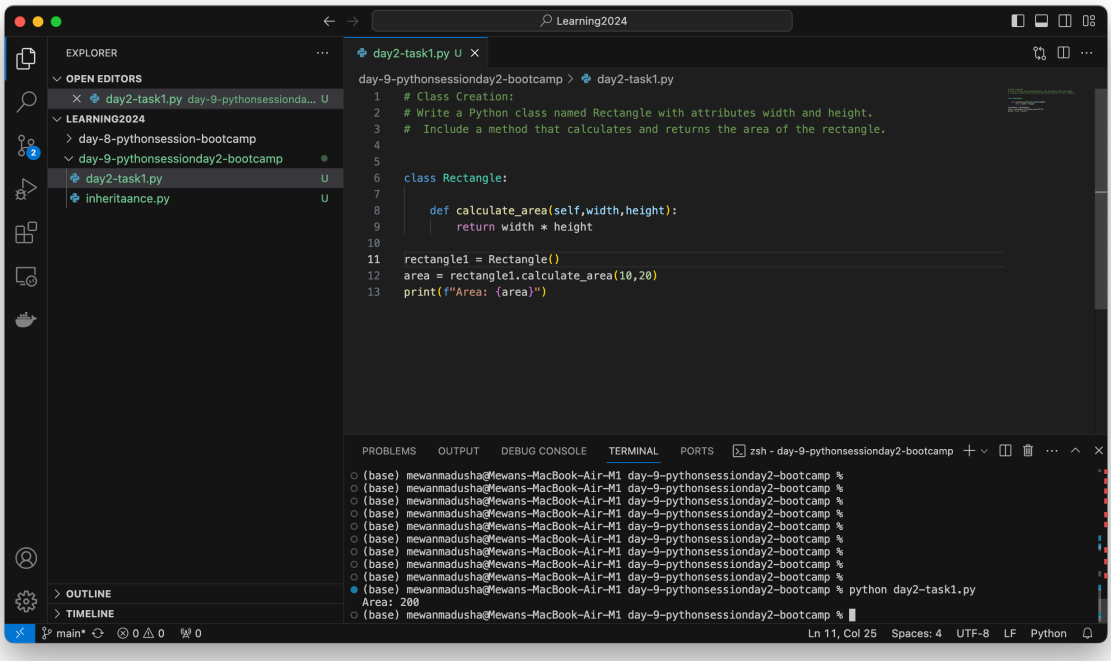
Table of Contents

Practice Related to Task 01

Class Creation:

Write a Python class named Rectangle with attributes width and height. Include a method that calculates and returns the area of the rectangle.

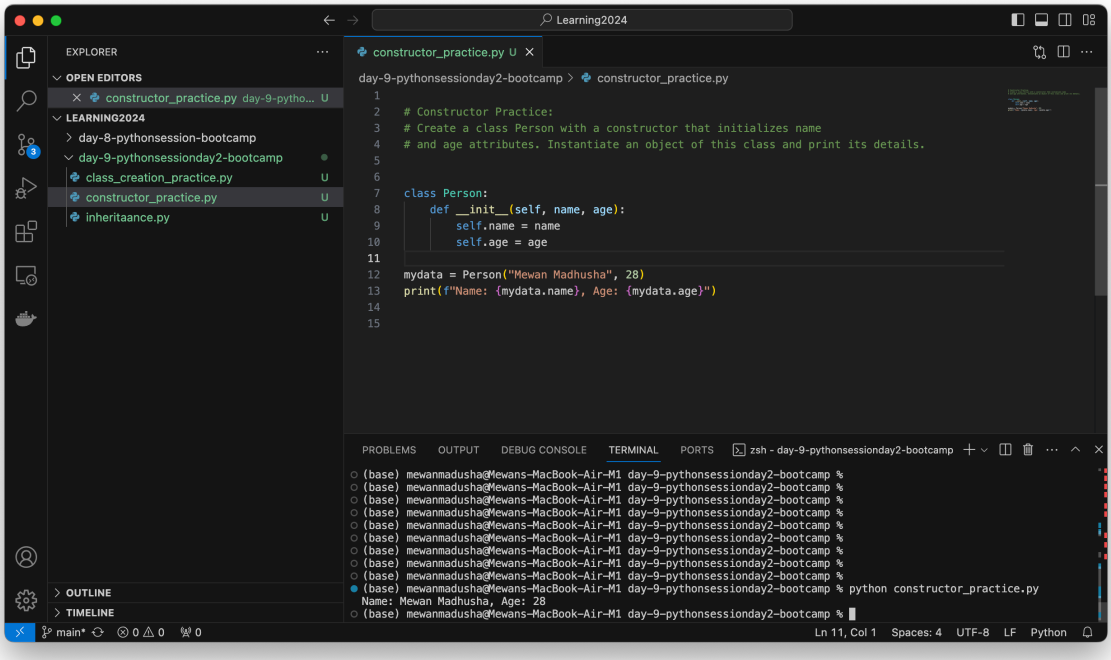
```
class Rectangle:

    def calculate_area(self,width,height):
        return width * height

rectangle1 = Rectangle()
area = rectangle1.calculate_area(10,20)
print(f"Area: {area}")
```



Constructor Practice:

Create a class Person with a constructor that initializes name and age attributes. Instantiate an object of this class and print its details.

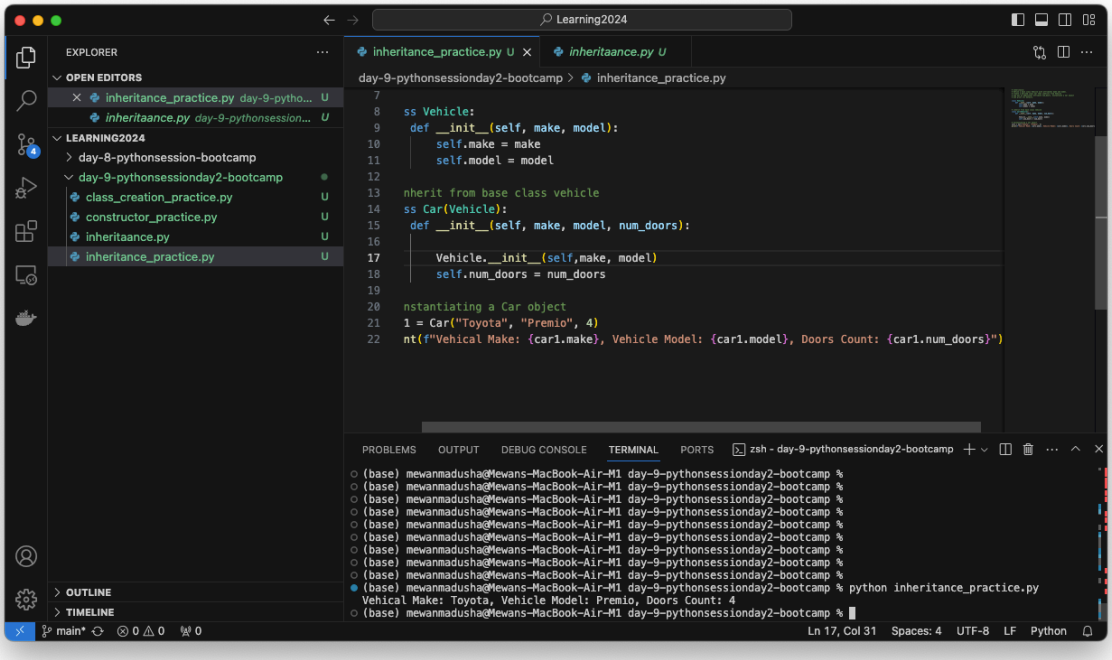
```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

mydata = Person("Mewan Madhusha", 28)
print(f"Name: {mydata.name}, Age: {mydata.age}")
```



Inheritance:

Define a base class Vehicle with attributes make and model. Create a derived class Car that inherits from Vehicle and has an additional attribute num_doors. Instantiate a Car object and print its details.

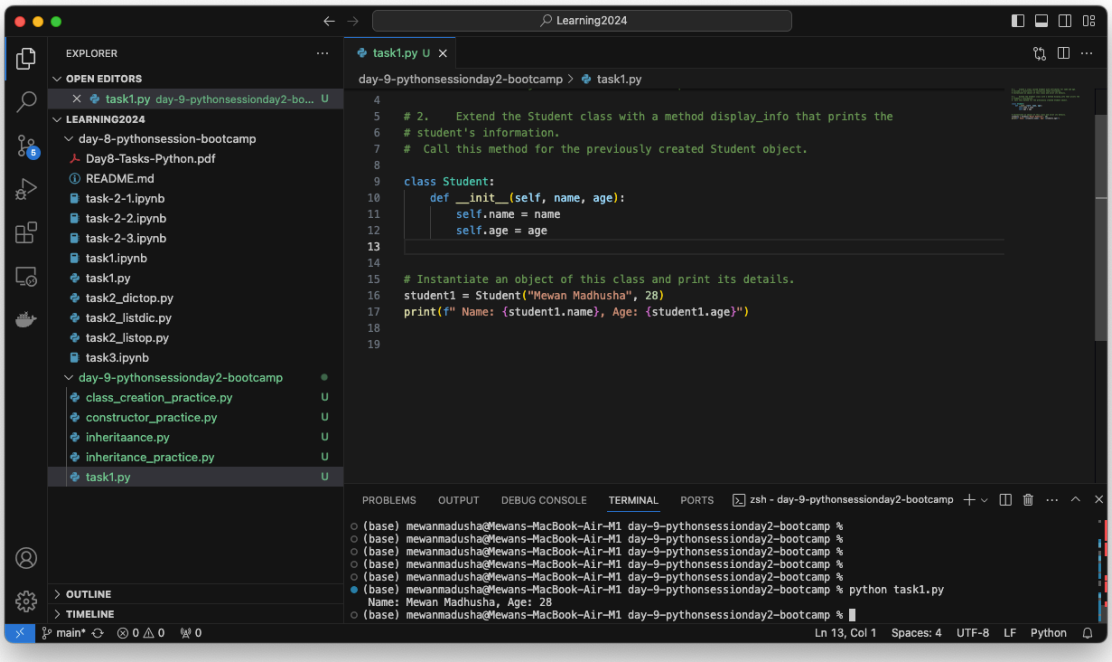
```
class Vehicle:
    def __init__(self, make, model):
        self.make = make
        self.model = model

# inherit from base class vehicle
class Car(Vehicle):
    def __init__(self, make, model, num_doors):

        Vehicle.__init__(self,make, model)
        self.num_doors = num_doors

# Instantiating a Car object
car1 = Car("Toyota", "Premio", 4)
print(f"Vehical Make: {car1.make}, Vehicle Model: {car1.model}, Doors Count:
{car1.num_doors}")
```


Learning2024
EXPLORER
OPEN EDITORS
inheritance_practice.py day-9-pytho... U
inheritaance.py day-9-pythonsession... U
LEARNING2024
day-8-pythonsession-bootcamp
day-9-pythonsessionday2-bootcamp
class_creation_practice.py U
constructor_practice.py U
inheritaance.py U
inheritance_practice.py U
OUTLINE
TIMELINE
inheritance_practice.py U
inheritaance.py U
day-9-pythonsessionday2-bootcamp > inheritance_practice.py
ss Vehicle:
def __init__(self, make, model):
self.make = make
self.model = model
nherit from base class vehicle
ss Car(Vehicle):
def __init__(self, make, model, num_doors):
Vehicle.__init__(self,make, model)
self.num_doors = num_doors
nstantiating a Car object
1 = Car("Toyota", "Premio", 4)
nt(f"Vehical Make: {car1.make}, Vehicle Model: {car1.model}, Doors Count: {car1.num_doors}")
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
zsh - day-9-pythonsessionday2-bootcamp
(base) mewanmadusha@Mewans-MacBook-Air-M1 day-9-pythonsessionday2-bootcamp %
(base) mewanmadusha@Mewans-MacBook-Air-M1 day-9-pythonsessionday2-bootcamp % python inheritance_practice.py
Vehical Make: Toyota, Vehicle Model: Premio, Doors Count: 4
(base) mewanmadusha@Mewans-MacBook-Air-M1 day-9-pythonsessionday2-bootcamp %
main*
Ln 17, Col 31
Spaces: 4
UTF-8
LF
Python

Task 01

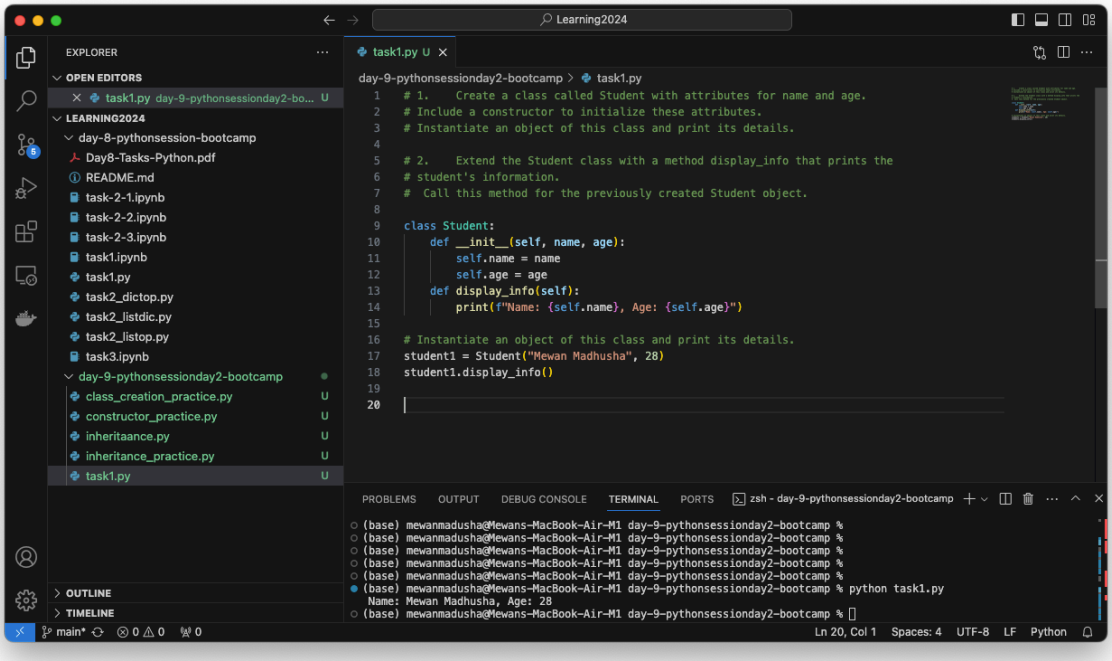
1. **Create a class called Student with attributes for name and age. Include a constructor to initialize these attributes. Instantiate an object of this class and print its details.**

```python
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age

# Instantiate an object of this class and print its details.
student1 = Student("Mewan Madhusha", 28)
print(f" Name: {student1.name}, Age: {student1.age}")
```



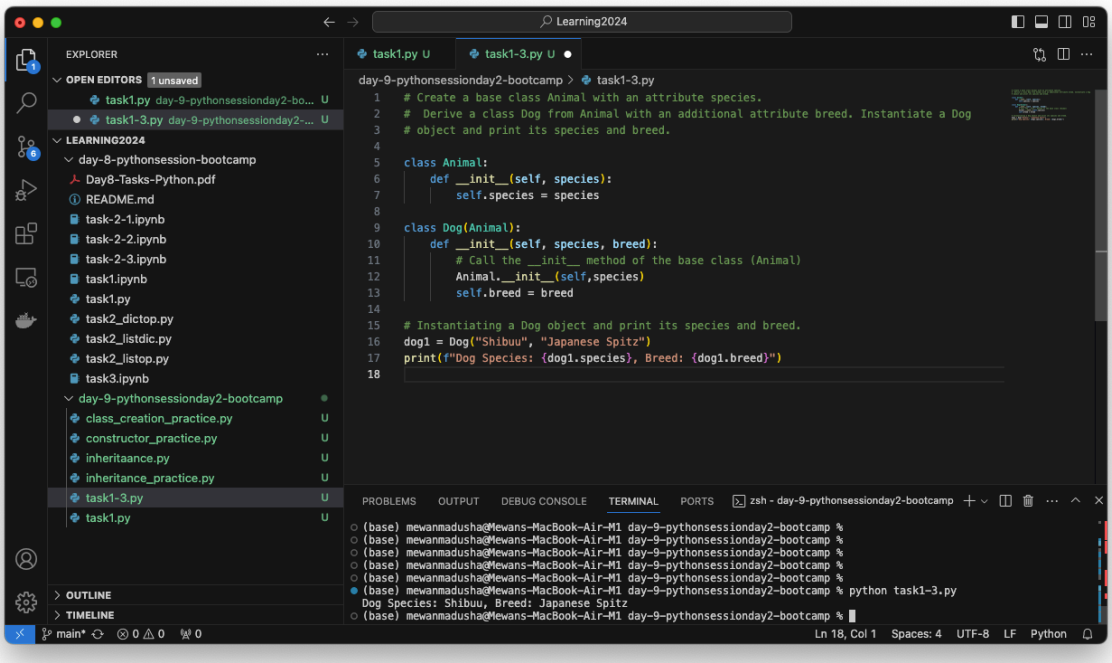
2. **Extend the Student class with a method display_info that prints the student's information. Call this method for the previously created Student object.**

```python
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def display_info(self):
        print(f"Name: {self.name}, Age: {self.age}")

# Instantiate an object of this class and print its details.
student1 = Student("Mewan Madhusha", 28)
student1.display_info()
```



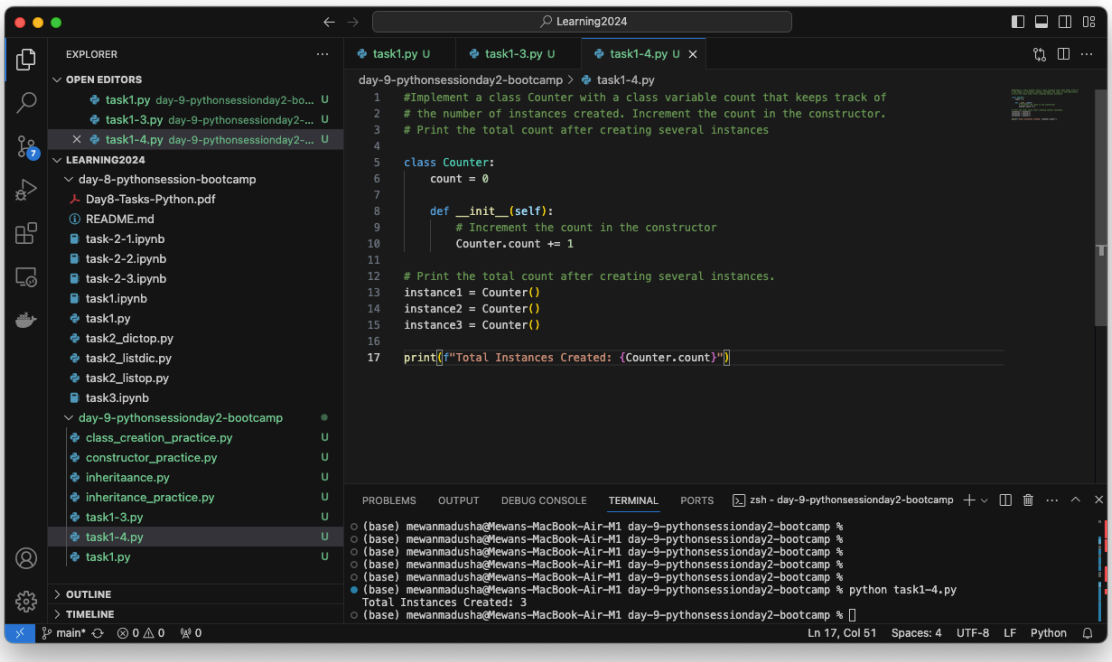
3. **Create a base class Animal with an attribute species. Derive a class Dog from Animal with an additional attribute breed. Instantiate a Dog object and print its species and breed.**

```python
class Animal:
    def __init__(self, species):
        self.species = species

class Dog(Animal):
    def __init__(self, species, breed):
        # Call the __init__ method of the base class (Animal)
        Animal.__init__(self,species)
        self.breed = breed

# Instantiating a Dog object and print its species and breed.
dog1 = Dog("Shibuu", "Japanese Spitz")
print(f"Dog Species: {dog1.species}, Breed: {dog1.breed}")
```



4. **Implement a class Counter with a class variable count that keeps track of the number of instances created. Increment the count in the constructor. Print the total count after creating several instances**

```python
class Counter:
    count = 0

    def __init__(self):
        # Increment the count in the constructor
        Counter.count += 1

# Print the total count after creating several instances.
instance1 = Counter()
instance2 = Counter()
instance3 = Counter()

print(f"Total Instances Created: {Counter.count}")
```



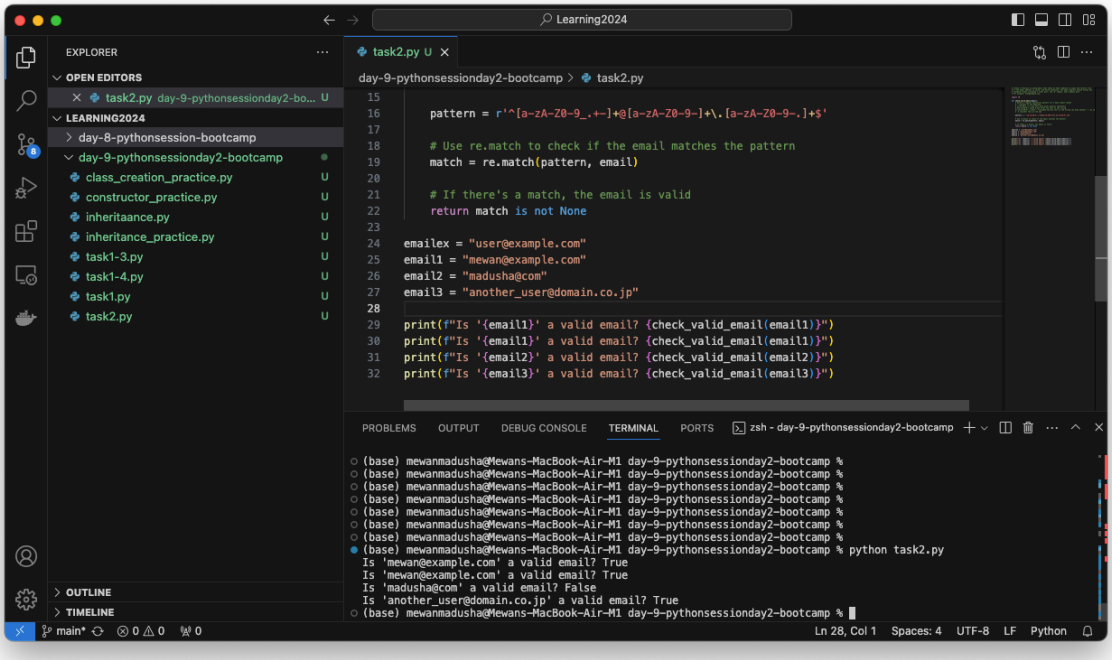
Task 02

Regex Assignment Question:
Problem Statement:
You are tasked with validating and extracting information from a set of email addresses. Write a Python program that uses regular expressions to accomplish the following tasks:

Email Validation:

Create a function is_valid_email that takes an email address as input and returns True if the email address is valid, and False otherwise. The email address should follow the standard format, including a local part, the '@' symbol, and a domain part. For example: user@example.com.

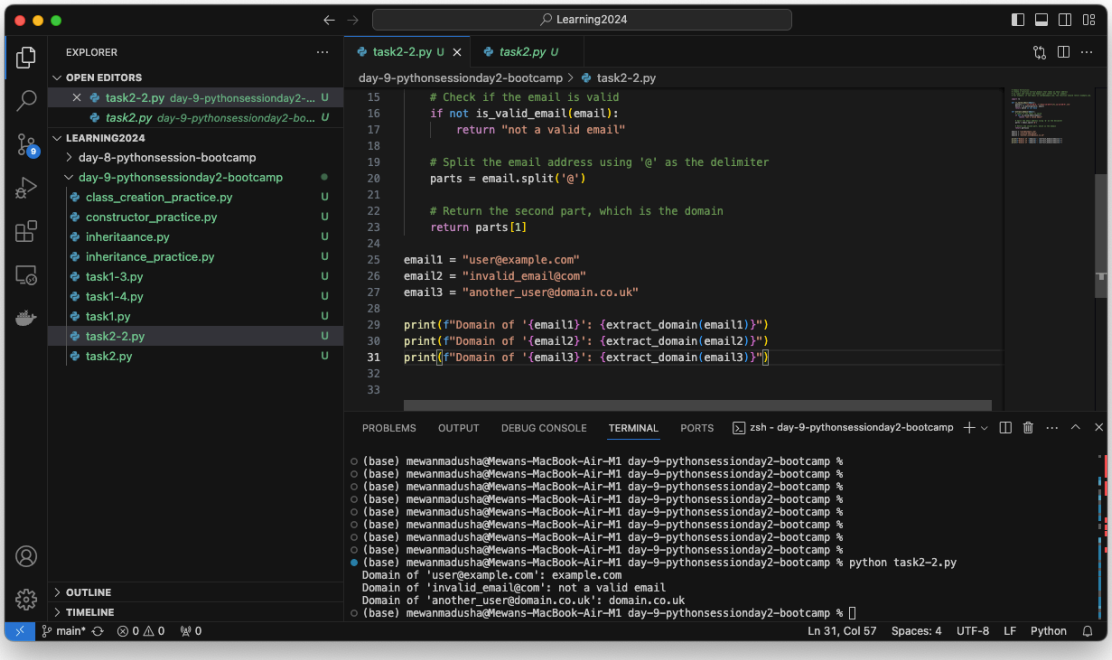
```
def check_valid_email(email):
    # Define a regular expression pattern for a basic email format
    # ^ consider as the begining
    # [a-zA-Z0-9_.+-] any text with given special characters
    # @[a-zA-Z0-9-] check @ and after any text in the string
    #\.[a-zA-Z0-9-.] check "."included and any text in the string can have another
"." as well
    #  $ consider this as the end

    pattern = r'^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$'

    # Use re.match to check if the email matches the pattern
    match = re.match(pattern, email)

    # If there's a match, the email is valid
    return match is not None

emailex = "user@example.com"
email1 = "mewan@example.com"
email2 = "madusha@com"
email3 = "another_user@domain.co.jp"

print(f"Is '{email1}' a valid email? {check_valid_email(email1)}")
print(f"Is '{email1}' a valid email? {check_valid_email(email1)}")
print(f"Is '{email2}' a valid email? {check_valid_email(email2)}")
print(f"Is '{email3}' a valid email? {check_valid_email(email3)}")
```



Domain Extraction:

Create a function extract_domain that takes an email address as input and returns the domain part (after the '@' symbol). For example, if the email is user@example.com, the function should return example.com.

```
import re

def is_valid_email(email):
    pattern = r'^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$'
    match = re.match(pattern, email)
    return match is not None

def extract_domain(email):
    # Check if the email is valid
    if not is_valid_email(email):
        return "not a valid email"

    # Split the email address using '@' as the delimiter
    parts = email.split('@')

    # Return the second part, which is the domain
    return parts[1]

email1 = "user@example.com"
email2 = "invalid_email@com"
email3 = "another_user@domain.co.uk"

print(f"Domain of '{email1}': {extract_domain(email1)}")
print(f"Domain of '{email2}': {extract_domain(email2)}")
print(f"Domain of '{email3}': {extract_domain(email3)}")
```



Username Extraction:

Create a function extract_username that takes an email address as input and returns the local part (before the '@' symbol). For example, if the email is user@example.com, the function should return user.

```
import re

def is_valid_email(email):
    pattern = r'^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$'
    match = re.match(pattern, email)
    return match is not None

def extract_domain(email):
    if not is_valid_email(email):
        return "not a valid email"

    # Split the email address using '@' as the delimiter
    parts = email.split('@')

    # Return the second part, which is the domain
    return parts[1]

def extract_username(email):
    if not is_valid_email(email):
        return "not a valid email"
    parts = email.split('@')
```

```python
    # Return the first part(username)
    return parts[0]

email1 = "user@example.com"
email2 = "invalid_email@com"
email3 = "another_user@domain.co.uk"

# print(f"Domain of '{email1}': {extract_domain(email1)}")
# print(f"Domain of '{email2}': {extract_domain(email2)}")
# print(f"Domain of '{email3}': {extract_domain(email3)}")


print(f"username of '{email1}': {extract_username(email1)}")
print(f"username of '{email2}': {extract_username(email2)}")
print(f"username of '{email3}': {extract_username(email3)}")
```