**Day 9 – Python Session Day 2 – Accenture bootcamp**

**Name: Pothumulla Kankanamge Mewan Madhusha**

Table of Contents

[Practice Related to Task 01 3](#_Toc156486134)

[Task 01 6](#_Toc156486135)

[Task 02 10](#_Toc156486136)

# 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}")

A screenshot of a computer

Description automatically generated  
**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}")

A screenshot of a computer

Description automatically generated  
**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}")

A screenshot of a computer

Description automatically generated

# 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.**

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}")

**A screenshot of a computer

Description automatically generated**

1. **Extend the Student class with a method display\_info that prints the student's information. Call this method for the previously created Student object.**

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()

**A screenshot of a computer program

Description automatically generated**

1. **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.**

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}")

A screenshot of a computer program

Description automatically generated

1. **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**

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}")

A screenshot of a computer program

Description automatically generated

# **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](mailto: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)}")

A screenshot of a computer program

Description automatically generated  
**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)}")

A screenshot of a computer

Description automatically generated

**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('@')

# 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)}")

