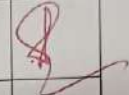
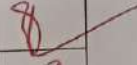



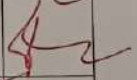



Table of Contents

Sl.no	Date of Execution	Exercise / Project Title	Program Write - up [10]	Execution & Output [40]	Total [50]	Faculty Sign
1	21/8/24	Simple Calculator	5	40	45	
2	28/8/24	Text File Word Counter	6	40	46	
3	7/9/24	Library Management System	5	40	45	
4	10/9/24	Data Analysis with Pandas	6	40	46	
5	17/9/24	To-Do List Application	5	40	45	
6	8/10/24	Blog Post Manager	6	40	46	
7	15/10/24	User Feedback Form	5	40	45	
8		Contact Manager				
9		User Registration and Login				
10		Basic Flask Blog				
11		To-Do List with Flask				
12		Deploying Django on PythonAnywhere				

Basics of Python

Lab Exercise 1: Basic Python Program

• Title: "Simple Calculator"

- Question: Jane is a math teacher who wants to create a simple calculator program for students. The program should take two numbers and an operator (+, -, *, /) as inputs and display the result of the operation.

• Additional Constraints:

- Ensure that the program handles division by zero by displaying an appropriate error message.
- Validate that the operator entered is one of the allowed operators, otherwise display an error message.

```
def calc(x: int, op: str, y: int):
```

```
    if op == "+":
        return x+y
```

```
    elif op == "-":
        return x-y
```

```
    elif op == "*":
        return x*y
```

```
    elif op == "/":
        return x/y
        if y==0:
            return "Invalid input"
    else:
        return x/y
```

```
    else:
        return "Invalid sign"
```

```
a = int(input())
op = input()
b = int(input())
```

```
answer = calc(a, op, b)
print(answer)
```

Viva Questions :

1. How do you take user input in Python and convert it to a numeric type?
2. What are conditional statements in Python and how do you use them to handle different operators?
3. Explain the use of try-except blocks in Python for handling exceptions.
4. How can you define and use functions in Python to perform arithmetic operations?
5. What is the significance of indentation in Python and how does it affect the program's flow?

1. user input:
use input() convert it with int()
⇒ num = int(input("Enter the number"))

2. Conditional statements:
- we use if, elif, else to handle conditions
eg if x > 0
print("x is positive")

3. Catching exception : we use try-except
- to catch and handle exception
eg ~~x = 1/0~~
try:
x = 1/0
except ZeroDivisionError:
print("Error")

4. Function: we define them with 'def' and perform actions inside
 →

```
def add(a,b):
    return a+b
```

5. indentation: Defines code blocks, critical for programming structure
 if x > 0:
 print (x is positive)
 ↙ indentation

Evaluator Remark (if Any):

Marks Secured: 30 out of 50

Signature of the Evaluator with Date

Lab Exercise 2: File Handling in Python

Title: "Text File Word Counter"

• Question: Mike is working on a project where he needs to analyze large text files. He wants a program that can read a text file, count the number of words in it, and print the count.

Additional Constraints:

- The program should ignore punctuation and case (e.g., "Hello" and "hello" should be counted as the same word).
- Provide the option to exclude common stop words (like "the", "and", "in") from the word count.

```
import string
stop_words = {'the', 'and', 'in', 'a', 'an', 'to', 'of', 'on', 'for', 'with', 'at', 'by', 'from', 'about'}
```

```
def count_words(file_path, exclude_stop_words=False):
```

```
    try:
```

```
        with open(file_path, 'r', encoding='utf-8') as file:
```

```
            text = file.read().lower()
```

```
            text = text.translate(str.maketrans('', '', string.punctuation))
```

```
            words = text.split()
```

```
            if exclude_stop_words:
```

```
                words = [word for word in words if word not in stop_words]
```

```
            word_count = len(words)
```

```
            print(f"Word count: {word_count}")
```

except FileNotFoundError:
print("File not found, check the path")

file_path = "sample.txt"
count_words(file_path, exclude_stop_words=True)

Viva Questions :

1. How do you open and read the contents of a text file in Python?
2. What are regular expressions, and how can they be used to remove punctuation from text?
3. How can you convert a string to lowercase in Python, and why is this useful for counting?
4. Explain the concept of lists and sets in Python and how you might use them to manage stop words.
5. How do you split a string into a list of words in Python?

1. with open('file.txt', 'r') as file:
content = file.read()

2. patterns used for string matching

```
import re
text = re.sub('[\W\s]', '', text)
```

3. string → lowercase

```
text = text.lower()
```

4. Lists vs sets

Lists: ordered, allows duplicates

Sets: unordered, no duplicates

→ sets can be used to store stopwords for fast membership checks & no duplicates

Sr words = text.split() → split string into words

Lab Exercise 3: Object-Oriented Programming in Python

- Title: "Library Management System"
- Question: Sarah is managing a small community library and wants a simple system to keep track of the books. She needs a class 'Book' with attributes like title, author, and ISBN, and methods to display and update book details. Create this class and demonstrate its functionality.
- Additional Constraints:
 - Add a method to search for books by title or author.
 - Implement a way to mark books as borrowed and returned, and ensure borrowed books cannot be borrowed again until returned.

class Book:

def __init__(self, t, a, i):

self.t = t

self.a = a

self.i = i

self.borrowed = False

def disp(self):

print(f"Title: {self.t}, author: {self.a}, ISBN: {self.i}")

Borrowed: {self.borrowed}"

def upd(self, t=None, a=None, i=None)

if t: self.t = t

if a: self.a = a

if i: self.i = i

Evaluator Remark (if Any):

Marks Secured: 46 out of 50

Signature of the Evaluator with Date

[2024-25] - ODD SEM - 23CS05EF - PFSD WORKBOOK

```
def search(self, term):  
    return term.lower() in self.t.lower() or  
    term.lower in self.q.lower()
```

```
def borrow(self, term):  
    if not self.borrowed:  
        self.borrowed = True  
        print(f'{self.t} is now borrowed')  
    else:  
        print(f'{self.t} is already borrowed')
```

```
def return_book(self):  
    self.borrowed = False  
    print(f'{self.t} is returned')
```

```
b1 = Book(1984, 'George Orwell', '123456')  
b1.display()  
b1.borrow()  
b1.borrow()  
b1.return_book()  
b1.display()
```

[2024-25] - ODD SEM - 23CS05EF - PFSD WORKBOOK

Viva Questions:

1. What is a class in Python and how do you define one?
2. How do you create and initialize attributes within a class in Python?
3. Explain the purpose and use of methods in a class. How do you define and call a method in Python?
4. How can you implement a search functionality within a class to find objects based on certain attributes?
5. What are the principles of object-oriented programming (OOP) and how do they apply to this library management system?

1. A blueprint for creating objects

```
class Book:
    pass
```

2. Use `__init__()` method to set attributes

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

3. Functions inside a class to perform actions. Defined call like this

```
class Book:
```

```
    def read(self):
        print("Reading the Book")
```

b.

```
b = book()
```

```
b.read()
```

4. Implement a method that searches objects by attributes

```
def search_by_title(self, title):
    if self.title == title:
        return self
```

5. OOP principles

- encapsulation
- Abstraction
- Inheritance
- Polymorphism

Evaluator Remark (if Any):

Marks Secured: 40 out of 50

Signature of the Evaluator with Date

4. Lab Exercise 4: Working with Python Libraries

- Title: "Data Analysis with Pandas"
- Question: Alex is a data analyst who needs to perform some basic data analysis on a CSV file containing sales data. He needs to calculate the total sales, average sales, and filter the data for a specific month. Use Pandas to help Alex with his analysis.
- Additional Constraints:
 - Ensure the program handles missing or corrupt data gracefully.
 - Add functionality to group the data by product category and calculate total sales for each category.

Import pandas as pd

def analyze(fp, m=None):

try:

df = pd.read_csv(fp).dropna(subset=['sales'])

df['sales'] = pd.to_numeric(df['sales'], errors='coerce').fillna(0)

t = df['sales'].sum()

a = df['sales'].mean()

print(f"Total: {t} \n Average: {a}")

if m:

df_m = df[df['month'] == m]

print(f"Sales for {m}: {df_m['sales'].sum()}")

```
cat_sales = df.groupby('category')['sales'].sum()
print("\n By category :", cat_sales)
```

```
except FileNotFoundError:
    print("file not found")
```


Viva Questions :

1. What is the Pandas library in Python, and how is it useful for data analysis?
2. How do you read data from a CSV file into a Pandas DataFrame?
3. Explain how you can handle missing or corrupt data in a Pandas DataFrame.
4. How can you calculate summary statistics, such as total and average, using Pandas?
5. Describe how you can filter data by a specific condition and group data by a category in Pandas.

1. A powerful python library for data manipulation & analysis, providing DataFrames for handling structured data efficiently

2. `import pandas as pd`
`df = pd.read_csv('file.csv')`

3. `→ Drop missing values: df.dropna()`
`→ Fill missing values: df.fillna(value)`

4. Total: `df['column'].sum()`
 Average: `df['column'].mean()`

5. Filter: `df[df['column'] > value]`
 group by: `df.groupby('category').mean()`

Evaluator Remark (if Any):

Marks Secured: 46 out of 50

Signature of the Evaluator with Date

Applications Using Django Framework

Lab Exercise 5: Basic To-Do List Application

- Title: "To-Do List Application"
- Question: Emma often forgets her daily tasks and wants a simple application to keep track of them. Develop a Django app where she can add, view, and delete to-do items using Django's model, form, and template system to create this application.
- Additional Constraints:
 - Allow users to mark tasks as completed and filter the list to show completed or pending tasks.
 - Implement user authentication so that each user can only see and manage their own to-do items.

views.py

```
from django.shortcuts import render, redirect
from models import Task
from forms import TaskForm
from django.contrib.auth.decorators import login_required
```

```
def task_list(request)
```

```
    tasks = Task.objects.filter(user=request.user)
    return render(request, 'todoapp/task_list.html',
                  {'form': TaskForm()})
```

```
def add_task(request):
```

```
    if request.method == 'POST':
```

```
        form = TaskForm(request.POST)
```

```
        if form.is_valid():
```

```
            task = form.save(commit=False)
```

```
            task.user = request.user
```

```
            task.save()
```

```
            return redirect('task_list')
```

```
    else:
```

```
        form = TaskForm()
```

```
    return render(request, 'todoapp/add_task.html',
                  {'form': form})
```

```
def delete_task(request, pk):
```

```
    task = Task.objects.get(pk=pk, user=request.user)
```

```
    task.delete()
```

```
    return redirect('task_list')
```


In urls.py (app)

```
path('add/', add_task, name='add-task')
path('delete/<int:pk>/', delete_task, name='delete-task')
```

In urls.py (project)

```
path('', include('todo_app.urls'));
```

Viva Questions :

1. What is Django, and how does its model-view-template (MVT) architecture work?
2. How do you create a model in Django to represent a to-do item?
3. Explain how forms are used in Django to handle user input and create new to-do items.
4. How can you implement user authentication in Django to ensure each user has their own to-do list?
5. Describe how you can filter querysets in Django to display only completed or pending tasks in a to-do list.

1. A high level python web framework that follows the Model-view-Template (MVT) architecture

- model:
 - Represents data & database schema
 - handles business logic & interact with models
 - Renders the user interface

```
2. from django.db import models
class TodoItem(models.Model):
    title = models.CharField(max_length=200)
    completed = models.BooleanField(default=False)
```

```
3. from django import forms
from .models import TodoItem
class TodoForm(forms.ModelForm):
    class Meta:
        model = TodoItem
        fields = ['title', 'completed']
```


4. Add user model in TodoItem model to a user.
 → Add user model in TodoItem model to a user.
 → Use Django contrib.auth for login/logout & @login_required decorator to secure views

5. completed tasks = TodoItem.objects.filter(completed=True)
 pending tasks = TodoItem.objects.filter(completed=False)

Lab Exercise 6: Django Models and Admin Interface

- Title: "Blog Post Manager"
- Question: John is starting a blog and wants to manage his blog posts efficiently. He needs a Django model for a blog post with fields for title and content. Additionally, he wants to use the Django admin interface to add and view posts. Create this system for John.
- Additional Constraints:
 - Add a field for the publication date and ensure posts are ordered by publication date in the admin interface.
 - Implement a slug field for each post to create user-friendly URLs.

views.py

from django.shortcuts import render
 from .models import Post

```
def post_list(request):
    posts = Post.objects.orderby('-pub_date')
    return render(request, 'blog/post_list.html',
                  {'post': posts})
```

Evaluator Remark (if Any):

Marks Secured: 10 out of 50

Signature of the Evaluator with Date

urls.py

```
urlpatterns = [
    path("", post_list, name="post-list"),
```

]

models.py

```
from slugify import slugify
```

```
class Post(models.Model):
```

```
    title = models.CharField(max_length=200)
```

```
    content = models.TextField()
```

```
    pub_date = models.DateTimeField(default=timezone.now)
```

```
    slug = models.SlugField(unique=True, blank=True)
```

```
    def save(self, *args, **kwargs):
```

```
        if not self.slug:
```

```
            self.slug = slugify(self.title)
```

```
        super().save(*args, **kwargs)
```

class Meta:

```
    ordering = ['-pub_date']
```

```
def __str__(self):
```

```
    return self.title
```

admin.py

```
from django.contrib import admin
```

```
from models import Post
```

```
admin.site.register(Post)
```


Viva Questions :

1. How do you define a Django model and what is its purpose?
2. Explain the steps to create and apply migrations for a new model in Django.
3. How do you customize the Django admin interface to add, view, and manage blog posts?
4. What is a slug field in Django and how do you generate it for creating user-friendly URLs?
5. How can you order querysets by a specific field, such as the publication date in Django?

1. A class that defines the structure of your database table. Its purpose is to represent and manage data using fields & relationships.

- 2.
1. Define the model in models.py
 2. Run: `python manage.py makemigrations` to create migration
 3. Apply the migration with: `python manage.py migrate`

3. Register the model in admin.py

from model's import BlogPost
admin.site.register(BlogPost)

Title	Python Full Stack Development with Django	Academic Year: 2024-25
Code(s)	23CS05EF	Page 34 of 68

4. → A slug is a URL-friendly string based on model's field, usually the title

→ from django.utils.text import slugify
post.slug = slugify(post.title)

5. posts = BlogPost.objects.all().order_by('publication_date')

Evaluator Remark (if Any):	Marks Secured: <u>46</u> out of 50
	Signature of the Evaluator with Date

Course Title	Python Full Stack Development with Django	Academic Year: 2024-25
Course Code(s)	23CS05EF	

Lab Exercise 7: Handling Forms in Django

- Title: "User Feedback Form"
- Question: Linda runs a website and wants to collect feedback from her users. She needs a Django form that collects user feedback (name, email, comments), validates the input, and displays a confirmation message upon submission. Build this form for Linda.
- Additional Constraints:
 - Ensure that the email field is validated to contain a valid email address.
 - Save the feedback to the database and display the collected feedback in the admin interface.

```
from django.shortcuts import render, redirect
from forms import feedbackform
from django.contrib import messages
```

```
def feedback_view(request):
```

```
    if request.method == 'POST':
```

```
        form = feedbackform(request.POST)
```

```
        if form.is_valid():
```

```
            form.save()
```

```
            messages.success(request, "Thank you")
```

```
    else
```

```
        form = feedbackform()
```

```
    return render(request, 'feedback.html', {'form': form})
```

urls.py

```
from django.urls import path
from views import feedback_view
```

```
urlpatterns = [
    path('feedback/', feedback_view, name='feedback'),
]
```

models.py

```
from django.db import models
```

```
class Feedback(models.Model):
```

```
    name = models.CharField(max_length=100)
```

```
    email = models.EmailField()
```

```
    comments = models.TextField()
```

```
    def __str__(self):
```

```
        return self.name
```

```

forms.py
from django import forms
from .models import Feedback

class FeedbackForm(forms.ModelForm):

    class Meta:
        model = Feedback
        fields = ['name', 'email', 'comments']

```

```

admin.py
from django.contrib import admin
from .models import Feedback
admin.site.register(Feedback)

```

Viva Questions :

1. How do you create a form in Django, and what is the purpose of using Django forms?
2. Explain the concept of form validation in Django. How can you validate that an email field contains a valid email address?
3. What are Django model forms, and how are they different from regular forms?
4. Describe the process of saving form data to a database using Django models.
5. How can you customize the Django admin interface to display and manage feedback submitted through the user feedback form?

```

1. from django import forms

class FeedbackForms(forms.Form):
    email = forms.EmailField()
    message = forms.CharField(widget=forms.Textarea)

```

2. ensures the input data is correct

```

email = forms.EmailField()

```

3.

```

from django import forms
from .models import Feedback

```

```

class FeedbackForm(forms.ModelForm):
    class Meta:
        model = Feedback
        fields = ['email', 'message']

```


4. Use a Model Form & call save()

```
if form.is_valid():
    form.save()
```

5. in admin.py

```
from django.contrib import admin
from .models import Feedback
admin.site.register(Feedback)
```

```
class FeedbackAdmin(admin.ModelAdmin):
    list_display = ('email', 'message', 'submitted_at')
admin.site.register(Feedback, FeedbackAdmin)
```

Lab Exercise 8: Business Contact Email Notification

- Title: "Contact Manager"
- Question: David needs an application to manage his business contacts. Develop a Django application where he can add, view, and delete contacts. Use Django's model, form, and template system to implement this functionality. Additionally, after creating a contact, provide an option to send the contact details to a user-defined email address.
- Additional Constraints:
 - Include fields for name, email, phone number, and address for each contact.
 - Implement search functionality to find contacts by name or email.

Evaluator Remark (if Any):

Marks Secured: 43 out of 50

Signature of the Evaluator with Date