

Setting Up a Python Project with GitHub, Local Environment, Logging, and Testing

Author: Hari Prasad

Date: 12/09/2023

Introduction

This document provides an overview of the steps taken to create a Python project repository on GitHub and set up the project on a local machine.

The project involves generating a basic project structure and a setup.py file for a Python application.

Step 1: Creating a GitHub Repository

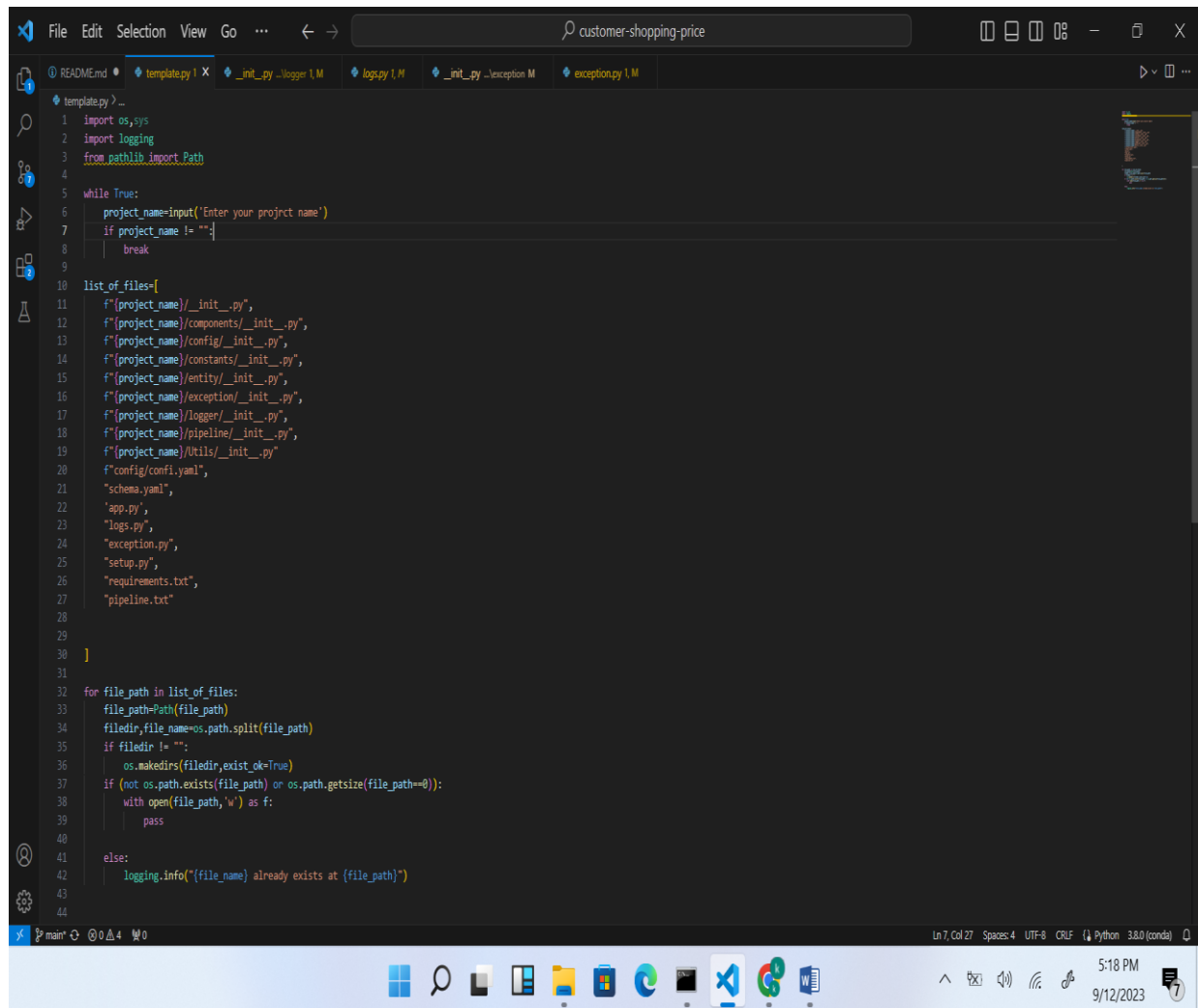
- 1.1. Visit the GitHub website (<https://github.com>) and log in to your account.
- 1.2. Click on the '+' icon in the top-right corner and select "New Repository" to create a new GitHub repository.
- 1.3. Fill in the repository name and provide an optional description and other settings as needed.
- 1.4. Click on the "Create repository" button to create the GitHub repository.

Step 2: Cloning the Repository Locally

- 2.1. Open your terminal or command prompt on your local machine.
- 2.2. Use the git clone command to clone the newly created GitHub repository to your local machine.
For example:
`git clone https://github.com/kalehariprasad/customer-shopping-price`

Step 3: Creating the Python Project Structure

- 3.1. In your local project directory, you've created a Python script called template.py. This script is responsible for generating the initial project structure.
- 3.2. The template.py script prompts you to enter a project name and then generates the following project structure snippet:



```
1 import os,sys
2 import logging
3 from pathlib import Path
4
5 while True:
6     project_name=input('Enter your project name')
7     if project_name != "":
8         break
9
10 list_of_files=[
11     f'{project_name}/__init__.py',
12     f'{project_name}/components/__init__.py',
13     f'{project_name}/config/__init__.py',
14     f'{project_name}/constants/__init__.py',
15     f'{project_name}/entity/__init__.py',
16     f'{project_name}/exception/__init__.py',
17     f'{project_name}/logger/__init__.py',
18     f'{project_name}/pipeline/__init__.py',
19     f'{project_name}/utils/__init__.py',
20     f'config/config.yaml',
21     "schema.yaml",
22     "app.py",
23     "logs.py",
24     "exception.py",
25     "setup.py",
26     "requirements.txt",
27     "pipeline.txt"
28 ]
29
30
31
32 for file_path in list_of_files:
33     file_path=Path(file_path)
34     filedir,file_name=os.path.split(file_path)
35     if filedir != "":
36         os.makedirs(filedir,exist_ok=True)
37     if (not os.path.exists(file_path) or os.path.getsize(file_path)==0):
38         with open(file_path,'w') as f:
39             pass
40     else:
41         logging.info(f'{file_name} already exists at {file_path}')
42
43
44
```

3.3. The script creates directories for various project components and initializes empty `__init__.py` files to indicate Python packages

Step 4: Creating the setup.py File

4.1. The setup.py file is used to define project metadata and dependencies for packaging.

4.2. The script, setup.py, imports required modules and defines a function, `get_requirements_list()`, to read project dependencies from a requirements.txt file.

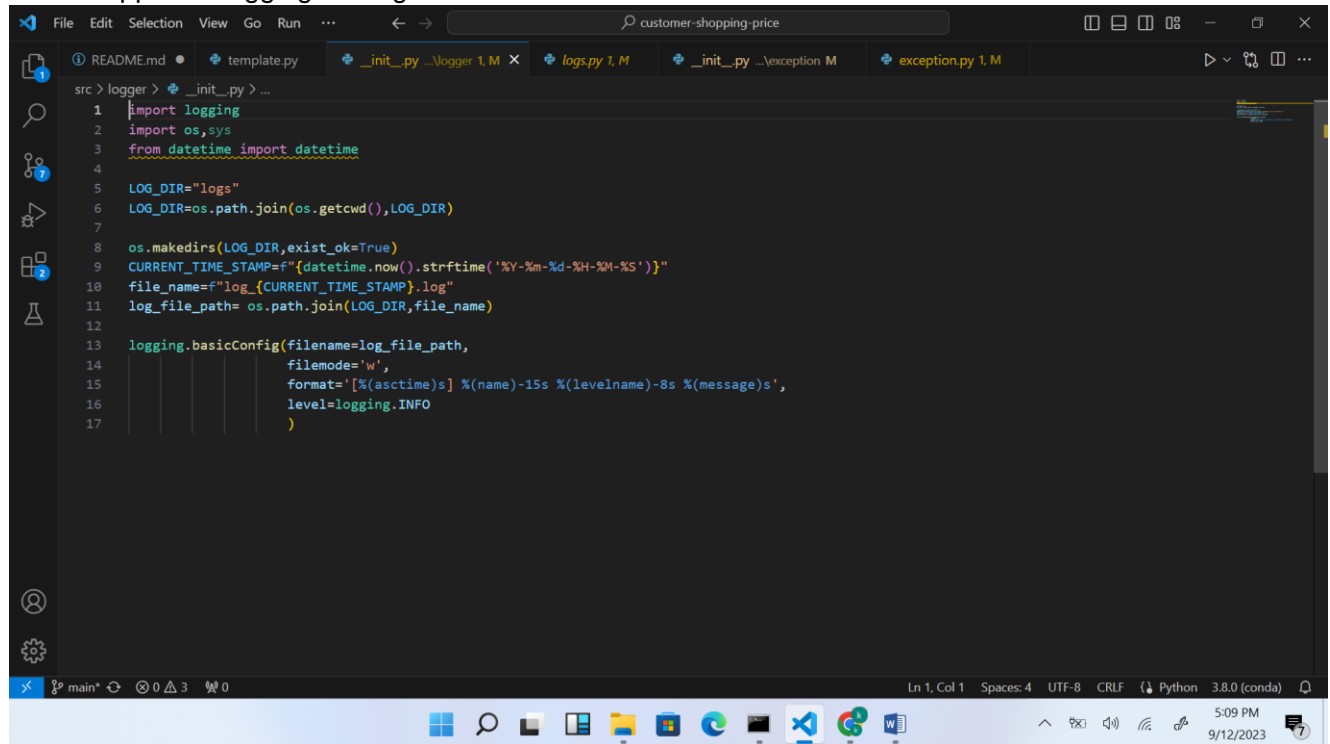
4.3. It then calls `setup()` from the `setuptools` library to configure the project with metadata such as name, version, author, and packages.

4.4. The script reads the project dependencies using the `get_requirements_list()` function and sets them as install requirements.

Step 5: Configuring Logging and Testing

5.1. Created a `logging` file (e.g., `logging.py`) in your project directory to handle logging configurations.

Code snippet for logging file is given below

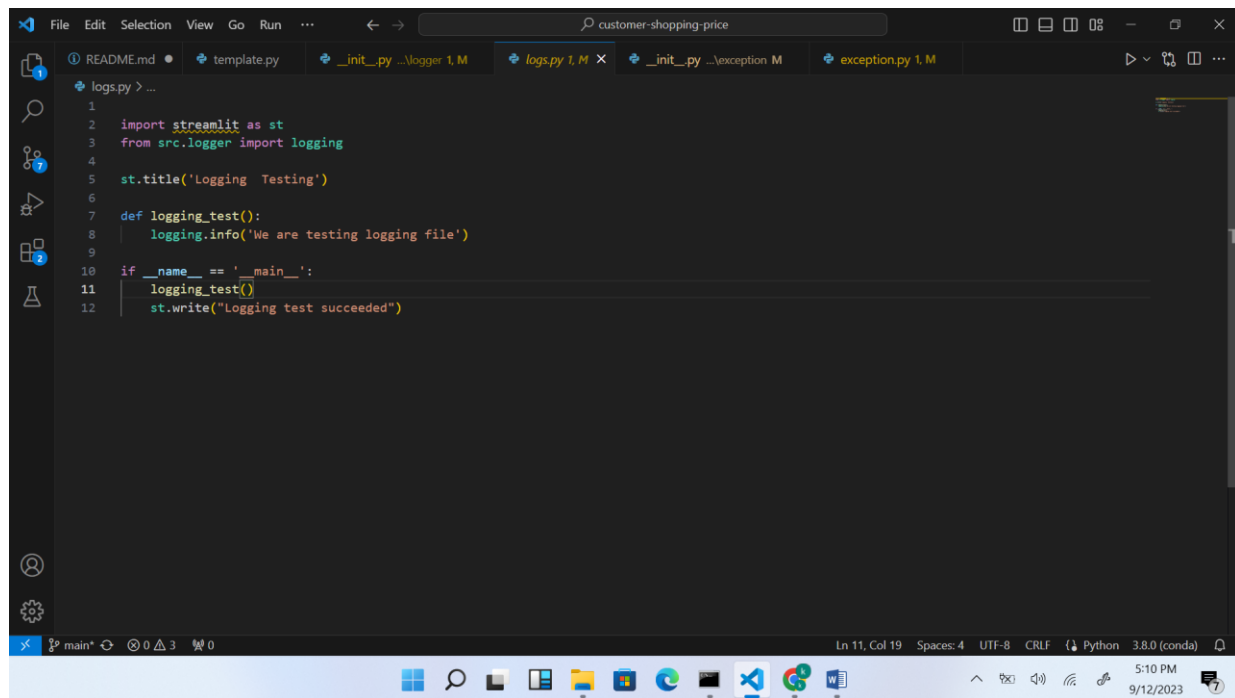
A screenshot of a code editor window with a dark theme. The title bar shows 'customer-shopping-price'. The editor has several tabs open: 'README.md', 'template.py', 'src > logger > _init_.py > ...', 'logger 1, M', 'logs.py 1, M', 'src > _init_.py > ...', 'exception M', and 'exception.py 1, M'. The active tab is 'logs.py 1, M'. The code in the editor is as follows:

```
1 import logging
2 import os, sys
3 from datetime import datetime
4
5 LOG_DIR="logs"
6 LOG_DIR=os.path.join(os.getcwd(), LOG_DIR)
7
8 os.makedirs(LOG_DIR, exist_ok=True)
9 CURRENT_TIME_STAMP=f"{datetime.now().strftime('%Y-%m-%d-%H-%M-%S')}"
10 file_name=f"log_{CURRENT_TIME_STAMP}.log"
11 log_file_path= os.path.join(LOG_DIR, file_name)
12
13 logging.basicConfig(filename=log_file_path,
14                     filemode='w',
15                     format='%(asctime)s] %(name)-15s %(levelname)-8s %(message)s',
16                     level=logging.INFO
17 )
```

The status bar at the bottom shows 'Ln 1, Col 1', 'Spaces: 4', 'UTF-8', 'CRLF', 'Python 3.8.0 (conda)', and the system clock '5:09 PM 9/12/2023'.

5.2. Created a testing script (e.g., `tests.py`) in your project directory to test the logging setup.

Code snippet for testing logging file

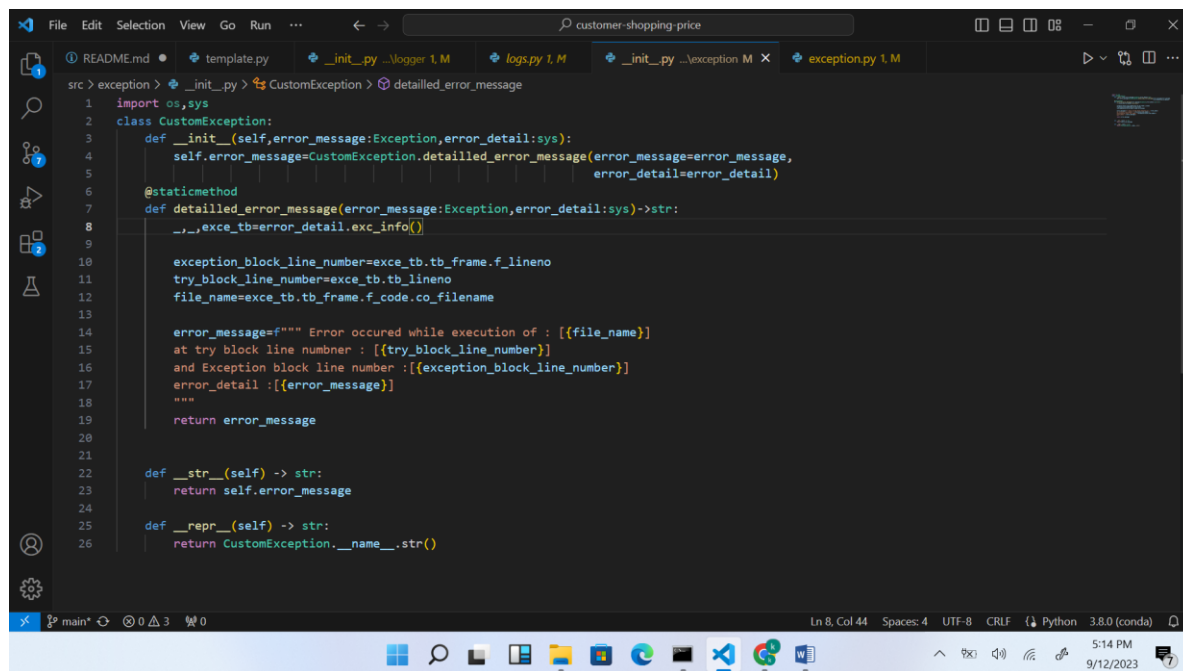
A screenshot of the Visual Studio Code editor interface. The top menu bar includes File, Edit, Selection, View, Go, Run, and a search icon. The search bar contains 'customer-shopping-price'. The file explorer on the left shows a project structure with files like README.md, template.py, __init__.py, logs.py, __init__.py, and exception.py. The main editor window displays the content of logs.py, which includes imports for streamlit and logging, a title 'Logging Testing', a logging_test() function, and a main block that calls logging_test() and writes a success message. The status bar at the bottom indicates 'main' is selected, with 0 errors and 3 warnings. The system tray shows the date and time as 5:10 PM on 9/12/2023.

```
1
2 import streamlit as st
3 from src.logger import logging
4
5 st.title('Logging Testing')
6
7 def logging_test():
8     logging.info('We are testing logging file')
9
10 if __name__ == '__main__':
11     logging_test()
12     st.write("Logging test succeeded")
```

- Step 6: Configuring Exception and Testing

6.1 Created a `exception` file (e.g., `exception.py`) in your project directory to handle exception configurations.

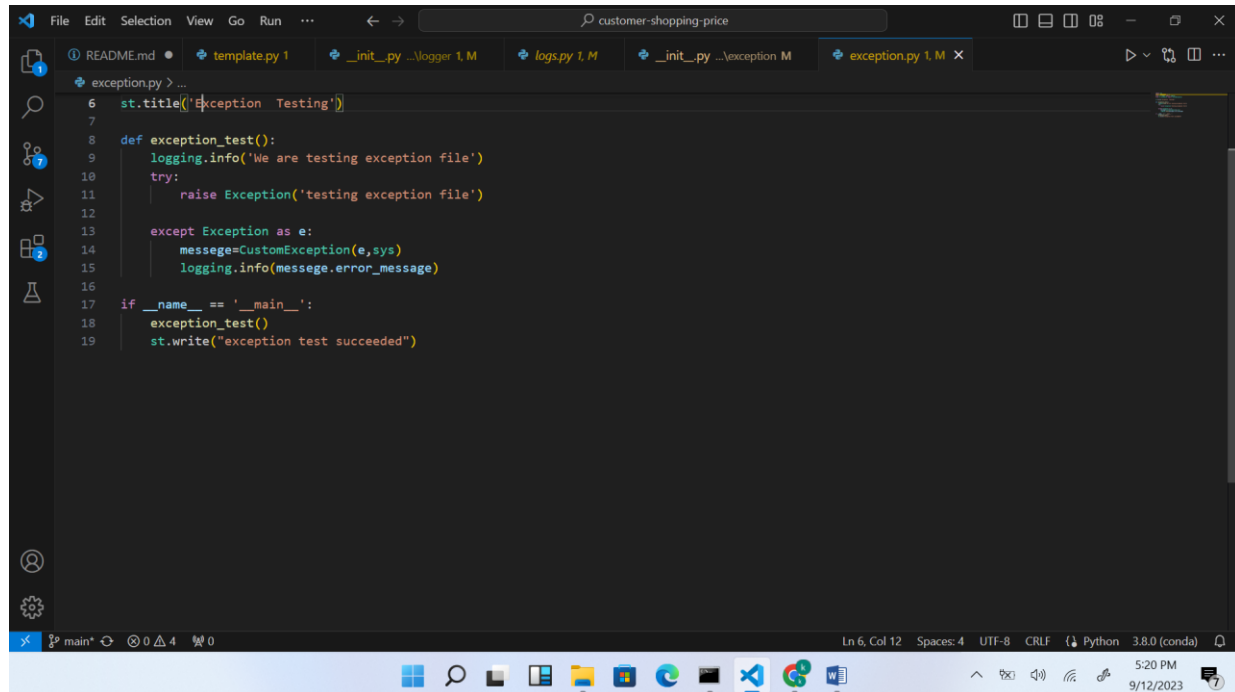
Code snippet for exception handling

A screenshot of the Visual Studio Code editor interface showing the exception.py file. The top menu bar and search bar are the same as in the previous screenshot. The file explorer shows the same project structure. The main editor window displays the content of exception.py, which defines a CustomException class with an __init__ method, a staticmethod detailed_error_message, and __str__ and __repr__ methods. The status bar at the bottom indicates 'main' is selected, with 0 errors and 3 warnings. The system tray shows the date and time as 5:14 PM on 9/12/2023.

```
src > exception > __init__.py > CustomException > detailed_error_message
1 import os,sys
2 class CustomException:
3     def __init__(self,error_message:Exception,error_detail:sys):
4         self.error_message=CustomException.detailed_error_message(error_message=error_message,
5                                                                     error_detail=error_detail)
6
7     @staticmethod
8     def detailed_error_message(error_message:Exception,error_detail:sys)->str:
9         exc_tb=error_detail.exc_info()
10
11         exception_block_line_number=exc_tb.tb_frame.f_lineno
12         try_block_line_number=exc_tb.tb_lineno
13         file_name=exc_tb.tb_frame.f_code.co_filename
14
15         error_message=f""" Error occured while execution of : [{file_name}]
16         at try block line number : [{try_block_line_number}]
17         and Exception block line number : [{exception_block_line_number}]
18         error_detail :[{error_message}]
19         """
20         return error_message
21
22     def __str__(self) -> str:
23         return self.error_message
24
25     def __repr__(self) -> str:
26         return CustomException.__name__.str()
```

6.2. Created a testing script (e.g., `exception.py`) in your project directory to test the exception setup.

Code Snippet :



```
6 st.title('Exception Testing')
7
8 def exception_test():
9     logging.info('We are testing exception file')
10    try:
11        raise Exception('testing exception file')
12    except Exception as e:
13        messege=CustomException(e,sys)
14        logging.info(messege.error_message)
15
16
17 if __name__ == '__main__':
18     exception_test()
19     st.write("exception test succeeded")
```