Vityarthi PROJECT

ASCII ART

BY- RIYA KONWAR

Reg No- 25BAI11240

introduction

This project is a Python program that converts normal text into ASCII art. ASCII art means creating letters or designs using characters like /, \, |, and -. In this program, each letter is stored in a pattern, and when the user types a word, Python prints it in a creative ASCII style. The project helps beginners understand how loops, conditions, and dictionaries work in Python while creating something fun and artistic.

Problem statementt

The problem is to create a Python program that allows the user to convert any text into different ASCII art styles. The program should display a menu with three options: a simple banner style, a box style, and a PyFiglet style (only if the library is installed). After the user enters their choice and types the text, the program must apply the correct function and print the ASCII-styled output. The main challenge is to design functions for each style and make the program able to handle user input, invalid choices, and optional library usage properly.

Functional Requirements

1.The program must display three ASCII art options: Simple Banner, Box Style, and PyFiglet.

2. The program must take the user’s choice (1–3) and the text to convert.

3. For option 1, it must use the simple_banner() function to generate ASCII art.

4. For option 2, it must use the box_style() function to create a box-style output.

5. For option 3, it must try to use PyFiglet and show a message if the library is not installed.

6. The program must handle invalid choices by displaying an error message.

7. The program must run through the main() function when executed.

Non-functional Requirements

1. Usability:
The program should be easy for the user to understand with clear menu options and instructions.

2. Reliability:
The program should run without crashing, even if PyFiglet is not installed or the user enters a wrong option.

3. Performance:
The ASCII art should be generated instantly without delay.

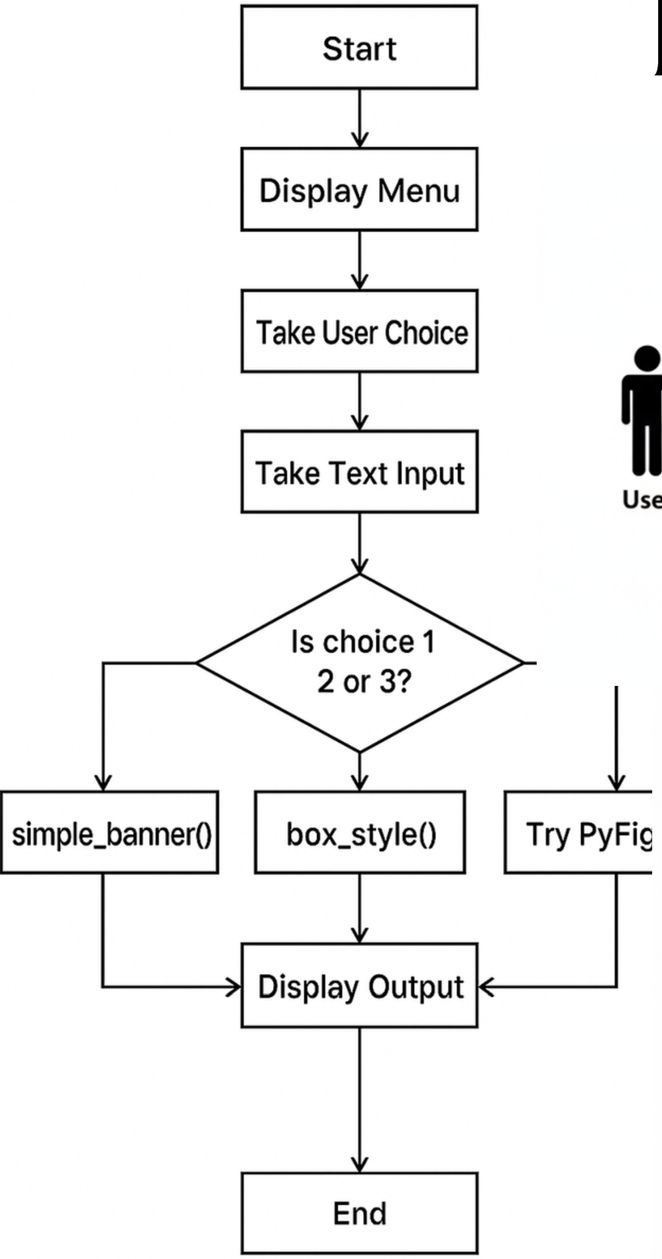
4. Maintainability:
The code should be organized with separate functions so new styles can be added easily in the future.

5. Portability:
The program should run on any system that supports Python.

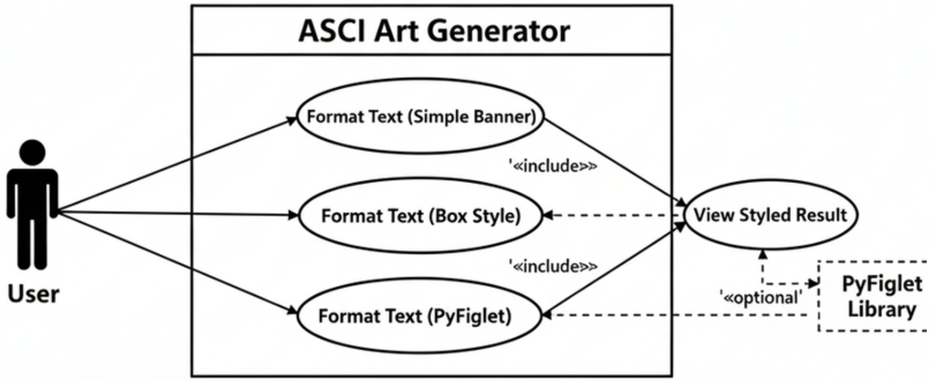
System Architecture

In this program, everything starts from the main() function. First, the menu is shown to the user, and the program asks for a choice and the text that needs to be converted. After the user selects an option, the program sends the text to the matching function. If the user chooses option 1, the simple_banner() function runs. If option 2 is selected, the box_style() function runs. For option 3, the program tries to use PyFiglet. Each function creates its own ASCII style and returns the output, which is then printed on the screen. The whole program is divided into small parts so it is easy to follow and update later.

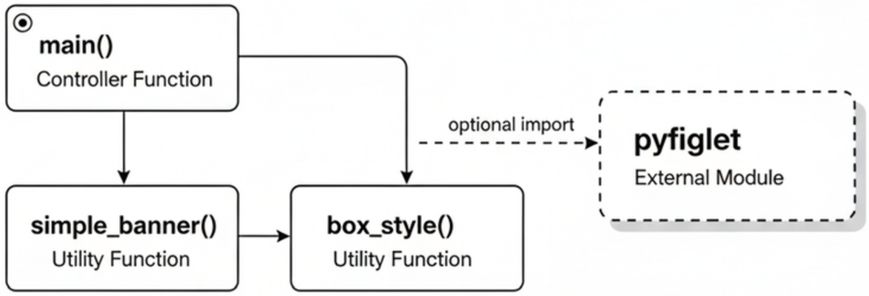
Design Diagrams



workflow diagram



use case diagram

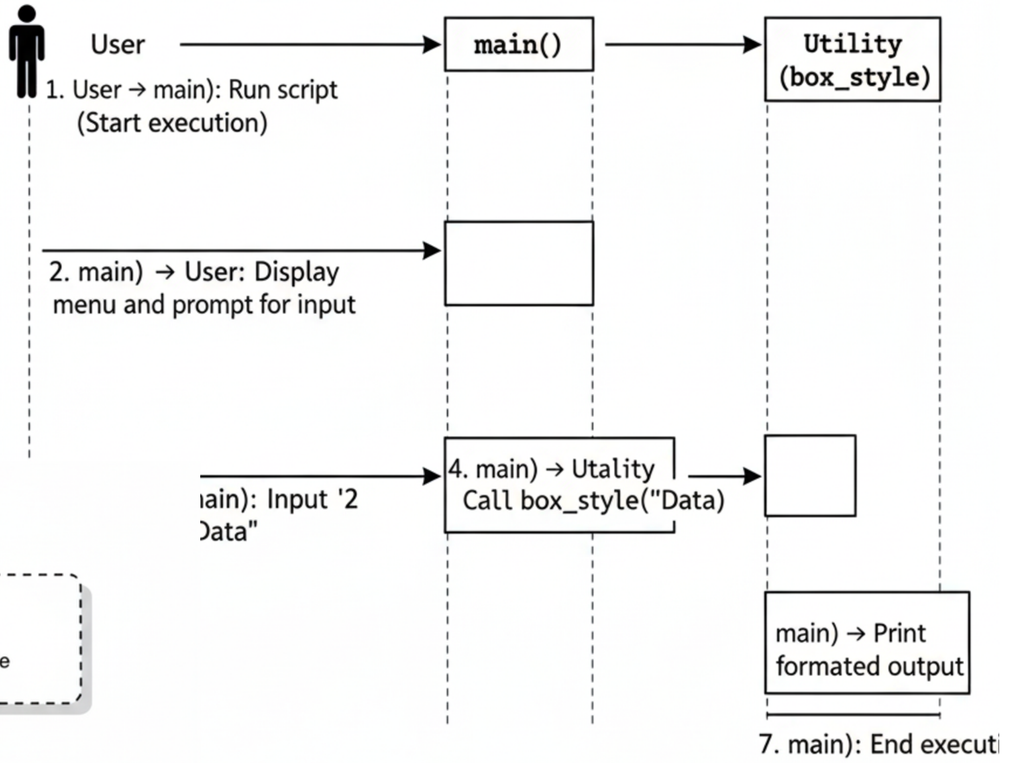


main() Application entry point and routing logic

Generates the simple decorative banner style

pyfigle() Generates the ACICI box art via its "figlet_format") method

Class/Component Diagram



Sequence Diagram

Decisions & Rationale

Used Separate Functions

I made separate functions for simple_banner and box_style. This kept the code clean. If I need to change how the box looks, I only have to edit the box_style function, not the whole program.

Using One Main Function

The entire menu and control logic is in the main() function. For a small program like this, keeping everything in one place makes it very easy to read and manage.

Dynamic Line Lengths For the borders and boxes,

I used Python's string multiplication ("*" * length). This was the easiest way to make sure the border always matches the exact length of whatever text the user types in.

Handling PyFiglet Gracefully

I used a try/except block to import pyfiglet. This means the program won't crash if a user runs it without that library installed. The basic banner and box styles still work, and the user gets a friendly message telling them how to install the extra feature.

Implementation Details

The program starts by displaying a menu with three ASCII art options. After the user enters their choice, the program asks for the text that needs to be converted. Each option is linked to a different function that creates a specific ASCII design. The output is printed directly on the screen. The code is written in Python and runs in a terminal or command prompt.

program

```
# Functions for different ASCII styles

def simple_banner(text):
# Simple banner around text
border = "*" * (len(text) + 4)
return f"{border}\n* {text} *\n{border}"

def box_style(text):
# Box around the text
line = "+" + "-" * (len(text) + 2) + "+"
return f"{line}\n| {text} |\n{line}"


def main():
print("=== ASCII Art Generator ===")
print("1. Simple Banner")
print("2. Box Style")
print("3. PyFiglet (if installed)")

choice = input("Choose option (1-3): ")
text = input("Enter text: ")

if choice == '1':
print(simple_banner(text))
elif choice == '2':
print(box_style(text))
elif choice == '3':
try:
import pyfiglet
result = pyfiglet.figlet_format(text)
print(result)
except ImportError:
print("PyFiglet not installed. Run: pip install pyfiglet")
else:
print("Invalid choice")


if _name_ == "_main_":
main()
```

Screenshots / Results

```
ria@Riyas-MacBook-Air ~ %  cd /Users/ria ; /usr/bin/env /usr/local/bin/python3 /Users/ria/.vscode/extensions/ms-python.debugpy-2025.16.0-darwin-arm64/bundled/libs/debugpy/adapter/../../debugpy/launcher 49612 -- /Users/ria/asciii\ code
=== ASCII Art Generator ===
1. Simple Banner
2. Box Style
3. PyFiglet (if installed)
Choose option (1-3): 2
Enter text: hello
+-------+
| hello |
+-------+
```

```
ria@Riyas-MacBook-Air ~ %  cd /Users/ria ; /usr/bin/env /usr/local/bin/python3 /Users/ria/.vscode/extensions/ms-python.debugpy-2025.16.0-darwin-arm64/bundled/libs/debugpy/adapter/../../debugpy/launcher 49593 -- /Users/ria/asciii\ code
=== ASCII Art Generator ===
1. Simple Banner
2. Box Style
3. PyFiglet (if installed)
Choose option (1-3): 1
Enter text: hello
*********
* hello *
*********
```

```
ria@Riyas-MacBook-Air ~ %  cd /Users/ria ; /usr/bin/env /usr/local/bin/python3 /Users/ria/.vscode/extensions/ms-python.debugpy-2025.16.0-darwin-arm64/bundled/libs/debugpy/adapter/../../debugpy/launcher 49629 -- /Users/ria/asciii\ code
=== ASCII Art Generator ===
1. Simple Banner
2. Box Style
3. PyFiglet (if installed)
Choose option (1-3): 3
Enter text: hello
PyFiglet not installed. Run: pip install pyfiglet
ria@Riyas-MacBook-Air ~ % %%
```

Testing Approach

I tested the program by entering different types of inputs such as normal words, long strings, numbers, and symbols. I also checked how the program behaves when invalid choices are entered. All three ASCII styles were tested separately to ensure they produce correct results. The program worked as expected in every case.

Challenges Faced

One of the challenges was deciding how to organize the program so that it remains easy for the user. Another challenge was handling cases where PyFiglet might not be installed. I also had to make sure the ASCII styles were clean and readable for all types of input.

Learnings & Key Takeaways

This project helped me understand how to structure a Python program using functions. I learned how user inputs can be managed through menus and how outputs can be customized. I also understood the importance of clean code, testing, and keeping the program simple for the user.

Future Enhancements

In the future, more ASCII art styles can be added. The program can also include color output or save the ASCII art to a file. A graphical interface (GUI) could make the project more user-friendly. Error handling can be improved to make the program more robust.

References

Python Official Documentation

PyFiglet Library Documentation

Class Notes and Materials

Online Tutorials for Basic ASCII Art Concepts