**PATTERN PRINTING**

-INDHUMATHI R

**TABLE OF CONTENT**

**DESCRIPTION PAGE NO**

**1.Aim of the project………………………………………………………… 3**

**2.Problem Statement……………………………………………………….. 3**

**3.Project Description……………………………………………………….. 3**

**4.Functionalities…………………………………………………………….. 3**

**5.Input Versatility with Error Handling and Exception Handling…….... 4**

**6.Code Implementation…………………………………………………… 5-6**

**7.Results and outcomes…………………………………………………… 7-8**

**8.Conclusion………………………………………………………………. 9**

**Aim of the Project:**

To print the name in a pattern (\*) asterisks where each letter of the name is replaced by an asterisk.

**Problem Statement:**

Python program that prints patterns representing each letter of the English alphabet using asterisks (\*). The program should allow the user to input a letter and then print the corresponding pattern.

**Project Description:**

### **Explanation**

1. **Pattern Dictionary**: The **patterns** dictionary stores the patterns for each letter. Extend this dictionary to include more letters as needed.
2. **Input Handling**: The program prompts the user for input and checks if it's a valid single letter from 'A' to 'Z' (case insensitive).
3. **Error Handling**: Using a **try-except** block, the program catches invalid inputs and prompts the user again until a valid input is provided.
4. **Pattern Printing**: The **print pattern** function prints the corresponding pattern for the given letter. If the pattern is not found, it notifies the user.

**Functionalities:**

### **Constraints**

* The program should handle invalid inputs gracefully, prompting the user to enter a valid uppercase letter if the input is not within 'A' to 'Z'.

**Pattern Specifications:**

* + Each letter's pattern should be represented within a fixed grid, for example, a 5x5 or 7x7 grid.
  + Each pattern should visually resemble the corresponding letter as closely as possible within the constraints of the grid size.

### **Implementation Hints**

* Use a dictionary to store the patterns for each letter, where the keys are the letters and the values are lists of strings representing the pattern.
* Loop through each line of the pattern and print it.
* Consider using functions to organize the code for better readability and maintainability.

**Input Versatility with Error Handling and Exception Handling:**

Adding versatility to the input and incorporating error handling and exception handling is essential to make the program robust and user-friendly. Let's enhance the previous problem statement with these features.

### **1. Input Versatility**

Input versatility means your code can handle different types of input, and unexpected input formats without crashing. Here are some techniques to achieve this:

#### **A. Accepting Multiple Input Types**

Ensure your code can handle inputs of different types (e.g., strings, integers, floats).

### **2. Error Handling**

Error handling ensures that your program can manage runtime errors gracefully.

#### **A. Using Try-Except Blocks**

Use **try-except** blocks to catch and handle exceptions.

**Code Implementation:**

letters = {

'A': [' \*\*\* ', '\* \*', '\*\*\*\*\*', '\* \*', '\* \*'],

'B': ['\*\*\*\* ', '\* \*', '\*\*\*\* ', '\* \*', '\*\*\*\* '],

'C': [' \*\*\*\*', '\* ', '\* ', '\* ', ' \*\*\*\*'],

'D': ['\*\*\*\* ', '\* \*', '\* \*', '\* \*', '\*\*\*\* '],

'E': ['\*\*\*\*\*', '\* ', '\*\*\*\*\*', '\* ', '\*\*\*\*\*'],

'F': ['\*\*\*\*\*', '\* ', '\*\*\* ', '\* ', '\* '],

'G': [' \*\*\*\*', '\* ', '\* \*\*', '\* \*', ' \*\*\*\*'],

'H': ['\* \*', '\* \*', '\*\*\*\*\*', '\* \*', '\* \*'],

'I': ['\*\*\*\*\*', ' \* ', ' \* ', ' \* ', '\*\*\*\*\*'],

'J': ['\*\*\*\*\*', ' \*', ' \*', '\* \*', ' \*\*\* '],

'K': ['\* \*', '\* \* ', '\*\* ', '\* \* ', '\* \*'],

'L': ['\* ', '\* ', '\* ', '\* ', '\*\*\*\*\*'],

'M': ['\* \*', '\*\* \*\*', '\* \* \*', '\* \*', '\* \*'],

'N': ['\* \*', '\*\* \*', '\* \* \*', '\* \*\*', '\* \*'],

'O': [' \*\*\* ', '\* \*', '\* \*', '\* \*', ' \*\*\* '],

'P': ['\*\*\*\* ', '\* \*', '\*\*\*\* ', '\* ', '\* '],

'Q': [' \*\*\* ', '\* \*', '\* \*', '\* \*\*', ' \*\* \*'],

'R': ['\*\*\*\* ', '\* \*', '\*\*\*\* ', '\* \* ', '\* \*'],

'S': [' \*\*\*\*', '\* ', '\*\*\*\* ', ' \*', '\*\*\*\* '],

'T': ['\*\*\*\*\*', ' \* ', ' \* ', ' \* ', ' \* '],

'U': ['\* \*', '\* \*', '\* \*', '\* \*', ' \*\*\* '],

'V': ['\* \*', '\* \*', '\* \*', ' \* \* ', ' \* '],

'W': ['\* \*', '\* \*', '\* \* \*', '\*\* \*\*', '\* \*'],

'X': ['\* \*', ' \* \* ', ' \* ', ' \* \* ', '\* \*'],

'Y': ['\* \*', ' \* \* ', ' \* ', ' \* ', ' \* '],

'Z': ['\*\*\*\*\*', ' \* ', ' \* ', ' \* ', '\*\*\*\*\*'],

}

string = input("ENTER THE STRING WITHOUT SPACE AND SPECIAL CHAR \n ")

try:

#print(len(string))

for i in range(5):

for word in range(len(string)):

current\_word = string[word].upper()

#print(current\_word)

if word == len(string)-1 :

print(letters[current\_word][i])

else :

print(letters[current\_word][i],end=' ')

except KeyError:

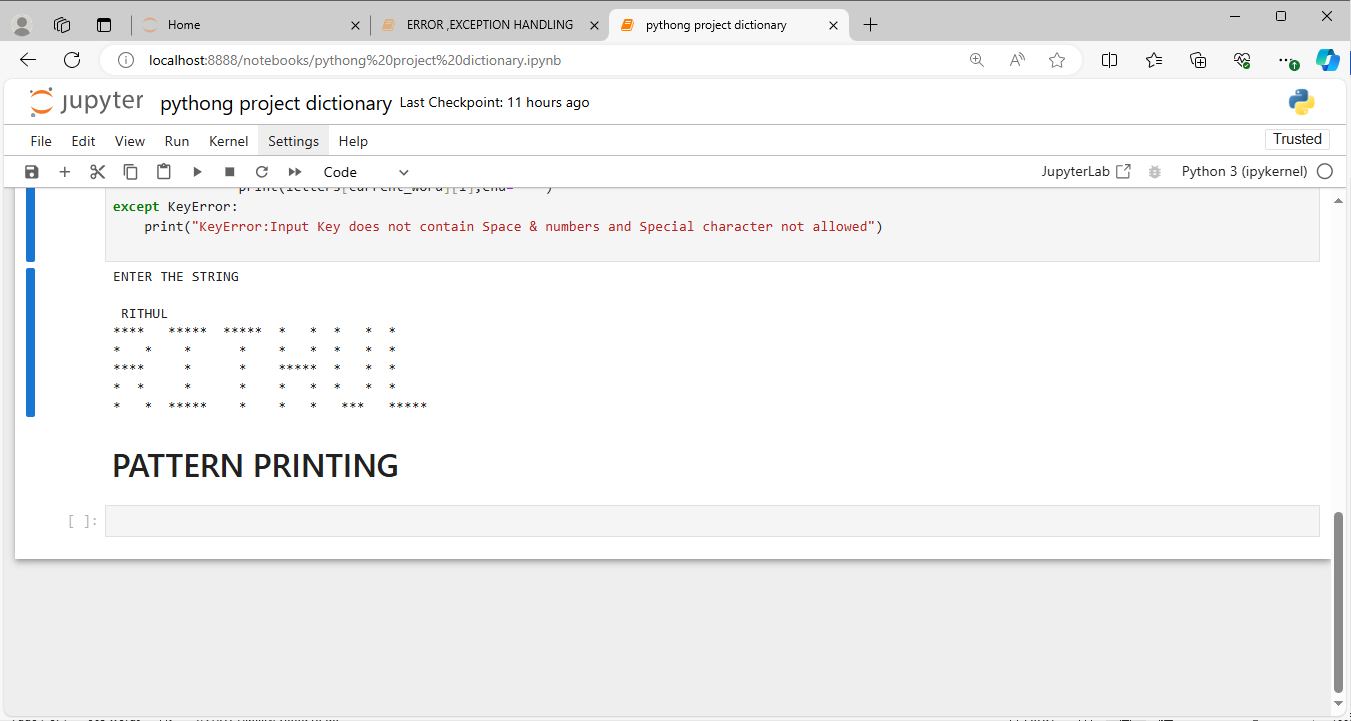
print("KeyError:Space & numbers and Special charetor not allowed")

**Results and outcomes:**

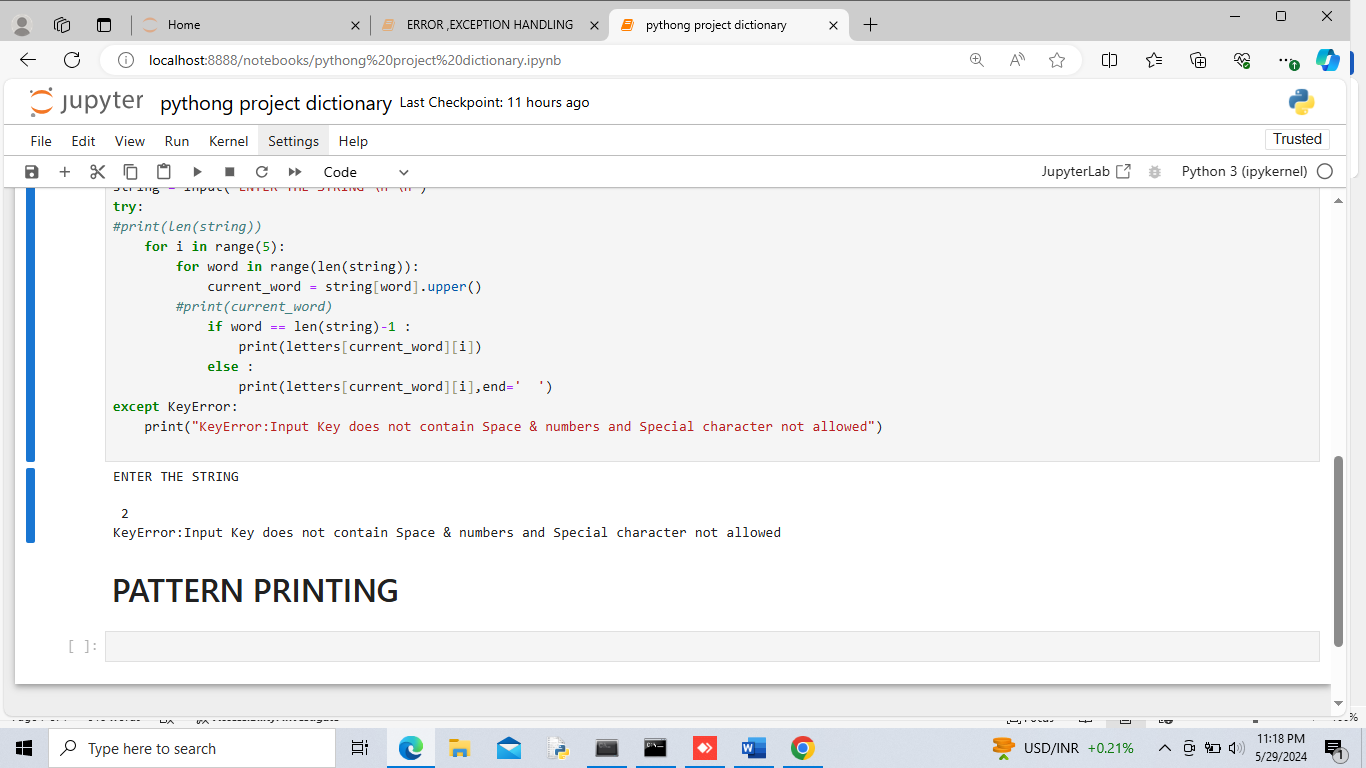
Creating a pattern printing problem statement involves specifying the requirements for a program that generates a specific visual pattern using characters such as asterisks (\*) or other symbols.

**OUTPUT:**

**1: STRING INPUT AND ITS OUTPUT**



**2: Error handling and its Output:**



**Conclusion:**

In this project, we developed a Python program capable of printing patterns representing each letter of the English alphabet using asterisks (\*). The program was designed to be user-friendly and robust, incorporating input versatility, error handling, and exception handling