

TASK – 1

TITLE

CodTech IT Solutions Internship - Task Documentation: “Simple calculators with Advance Features”

INTERN INFORMATION

Name: Kandagatla Neetha

ID: C0D5998

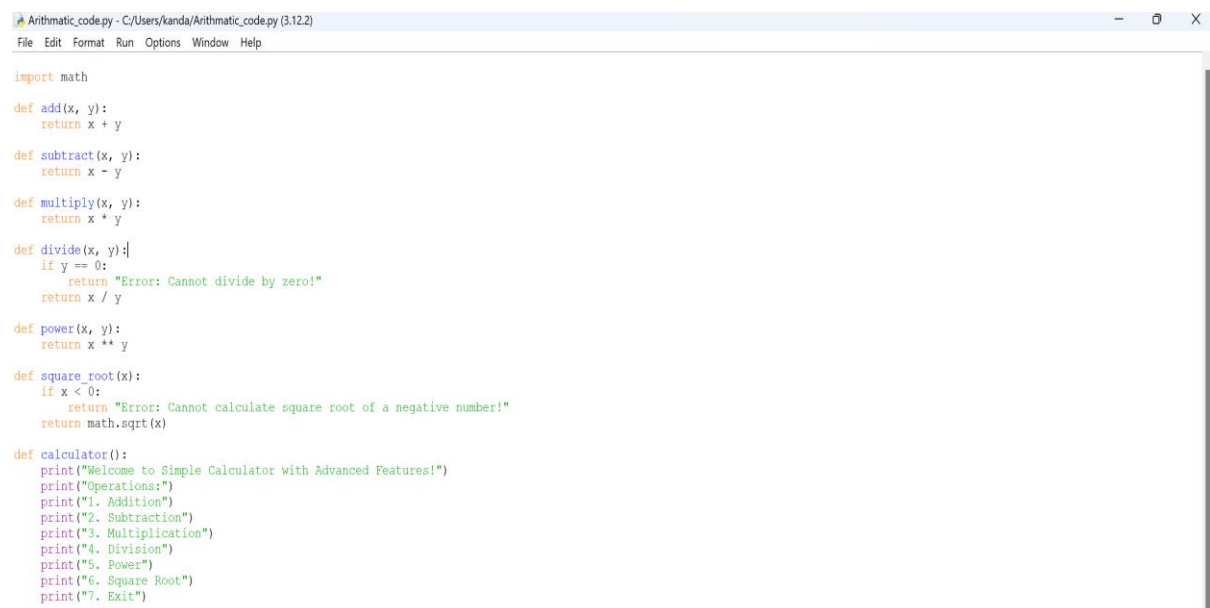
INTRODUCTION

Simple Calculator with Advanced Features:

This calculator is designed to provide more than just basic arithmetic operations. With additional functionalities such as exponentiation, square roots, and error handling for invalid inputs, calculator aims to meet diverse calculation needs.

Whether performing simple additions or exploring complex mathematical operations, this user-friendly interface ensures a seamless experience. Navigate through various operations effortlessly and receive accurate results promptly.

Implementation (Code)



```
Arithmetic_code.py - C:/Users/kanda/Arithmetic_code.py (3.12.2)
File Edit Format Run Options Window Help

import math

def add(x, y):
    return x + y

def subtract(x, y):
    return x - y

def multiply(x, y):
    return x * y

def divide(x, y):
    if y == 0:
        return "Error: Cannot divide by zero!"
    return x / y

def power(x, y):
    return x ** y

def square_root(x):
    if x < 0:
        return "Error: Cannot calculate square root of a negative number!"
    return math.sqrt(x)

def calculator():
    print("Welcome to Simple Calculator with Advanced Features!")
    print("Operations:")
    print("1. Addition")
    print("2. Subtraction")
    print("3. Multiplication")
    print("4. Division")
    print("5. Power")
    print("6. Square Root")
    print("7. Exit")
```

```

while True:
    choice = input("Enter operation number (1-7): ")

    if choice in ('1', '2', '3', '4', '5'):
        num1 = float(input("Enter first number: "))
        num2 = float(input("Enter second number: "))
    elif choice == '6':
        num = float(input("Enter a number: "))
    elif choice == '7':
        print("Exiting the calculator. Goodbye!")
        break
    else:
        print("Invalid input! Please enter a number between 1 and 7.")
        continue

    if choice == '1':
        print("Result:", add(num1, num2))
    elif choice == '2':
        print("Result:", subtract(num1, num2))
    elif choice == '3':
        print("Result:", multiply(num1, num2))
    elif choice == '4':
        print("Result:", divide(num1, num2))
    elif choice == '5':
        print("Result:", power(num1, num2))
    elif choice == '6':
        print("Result:", square_root(num))

calculator()

```

Ln: 13 Col: 17

CODE EXPLANATION

1. Function Definitions: The code begins with the definition of several mathematical functions:

- add (x, y): Adds two numbers x and y.
- subtract (x, y): Subtracts y from x.
- multiply (x, y): Multiplies x and y.
- divide (x, y): Divides x by y, handling division by zero error.
- power (x, y): Raises x to the power of y.
- square root (x): Calculates the square root of x, handling negative input error.

2. Calculator Function:

- Calculator() : This function contains the main logic of the calculator.
- It starts by printing the introduction message, explaining the calculator's features.
- It then displays the available operations and prompts the user to choose an operation.
- Based on the user's choice, it collects the necessary input numbers and performs the selected operation.
- It repeats this process until the user chooses to exit.

3. Introduction Text:

- The introduction message is displayed at the beginning of the calculator() function.
- It welcomes the user to the calculator and explains its purpose and features, including basic arithmetic operations, exponentiation, and square roots.

4. User Input and Output:

- The program prompts the user to enter the operation number and input numbers as needed.
- It validates user input and handles errors gracefully, such as division by zero or taking the square root of a negative number.
- It displays the result of each operation performed by calling the appropriate function based on the user's choice.

5. Loop and Exit:

- The calculator function runs in a continuous loop until the user chooses to exit by entering '7'.
- When the user decides to exit, the program prints a farewell message and terminates.

Overall, this code provides a user-friendly calculator interface with advanced mathematical functionalities while ensuring robust error handling and ease of use.

USAGE

You can use the provided Python code for the Simple Calculator with Advanced Features in various scenarios where basic arithmetic calculations and more advanced mathematical operations are needed. Here are some examples of how you can use this code:

1. Personal Use: We can run the calculator program on our local machine whenever we need to perform calculations, whether it's simple addition or more complex operations like exponentiation or square roots.

2. Educational Purposes: Teachers can utilize this calculator code as an educational tool to demonstrate how basic arithmetic operations are implemented in Python. Additionally, it can serve as an example of error handling and user input validation.

3. Integration into Larger Projects: You can integrate this calculator code into larger Python projects where mathematical computations are required. For example, you could incorporate it into a scientific or engineering application that needs to perform calculations.

4. Training and Learning: Beginners learning Python programming can study this code to understand concepts such as functions, user input, conditional statements, loops, and error handling. They can also modify and extend the code to practice their programming skills.

5. Testing and Debugging: Developers can use this calculator code for testing and debugging purposes in software projects. They can verify the correctness of mathematical computations or use it to troubleshoot issues related to calculations.

Overall, the usage of this code extends to a wide range of applications, including personal, educational, professional, and developmental contexts, making it a versatile tool for performing various calculations in Python.

CONCLUSION

In conclusion, the provided Python code for the Simple Calculator with Advanced Features offers a versatile solution for performing basic arithmetic operations and more advanced mathematical calculations. Here's a summary of the key points:

1. Features: The calculator supports basic arithmetic operations such as addition, subtraction, multiplication, and division, along with advanced functionalities like exponentiation and square roots.

2. User-Friendly Interface: The program provides a clear and intuitive user interface with a menu of available operations, guiding users through the calculation process.

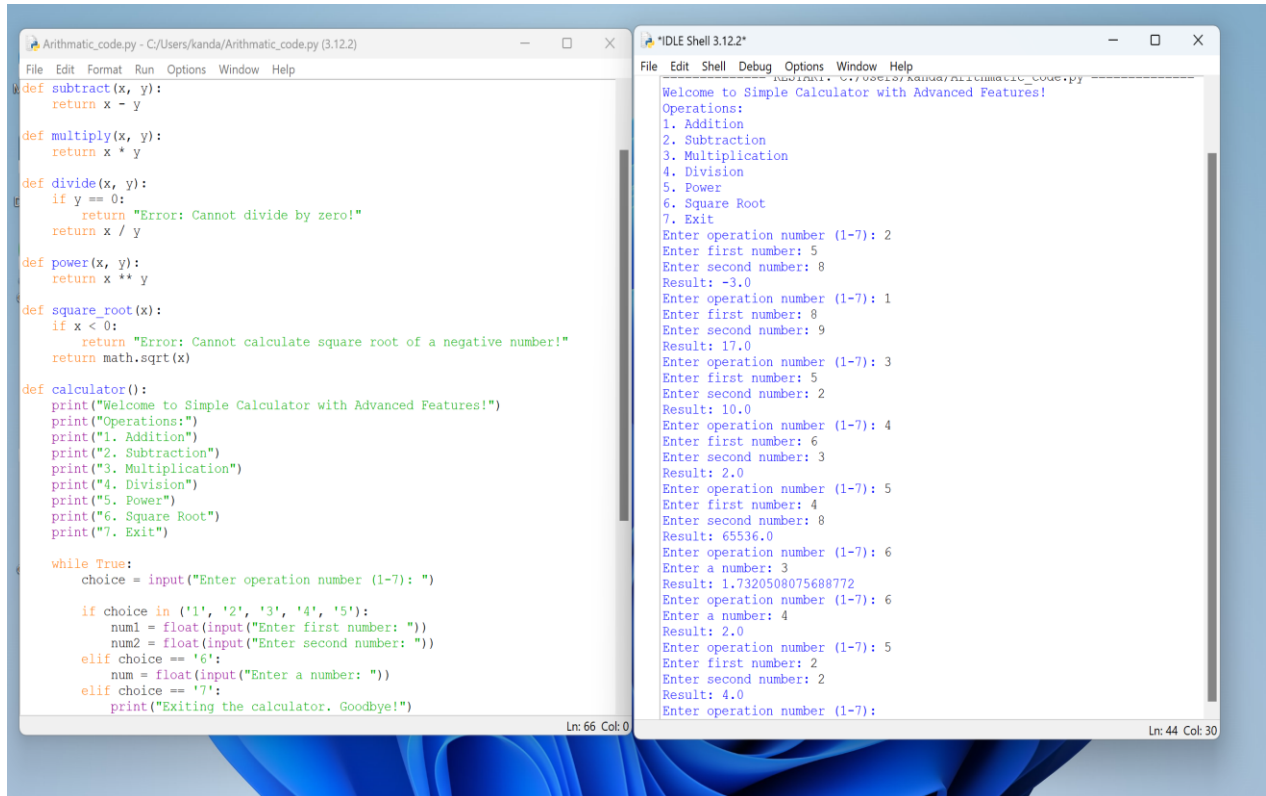
3. Error Handling: Robust error handling ensures that the program gracefully handles invalid inputs, such as division by zero or taking the square root of a negative number, providing informative error messages to the user.

4. Educational Value: The code serves as an educational resource for learning Python programming concepts, including functions, user input, conditional statements, loops, and error handling.

5. Versatile Usage: The calculator code can be used for personal calculations, educational purposes, integration into larger projects, training and learning, as well as testing and debugging.

Overall, the Simple Calculator with Advanced Features offers a practical and efficient solution for a wide range of mathematical calculations, making it a valuable tool for Python programmers and users alike.

OUTPUT



The image shows a screenshot of a Python IDE with two windows. The left window, titled 'Arithmetic_code.py - C:/Users/kanda/Arithmetic_code.py (3.12.2)', contains the source code for a simple calculator. The right window, titled 'IDLE Shell 3.12.2*', shows the output of the program, which is a text-based menu and a series of user inputs and calculations.

```
def subtract(x, y):  
    return x - y  
  
def multiply(x, y):  
    return x * y  
  
def divide(x, y):  
    if y == 0:  
        return "Error: Cannot divide by zero!"  
    return x / y  
  
def power(x, y):  
    return x ** y  
  
def square_root(x):  
    if x < 0:  
        return "Error: Cannot calculate square root of a negative number!"  
    return math.sqrt(x)  
  
def calculator():  
    print("Welcome to Simple Calculator with Advanced Features!")  
    print("Operations:")  
    print("1. Addition")  
    print("2. Subtraction")  
    print("3. Multiplication")  
    print("4. Division")  
    print("5. Power")  
    print("6. Square Root")  
    print("7. Exit")  
  
    while True:  
        choice = input("Enter operation number (1-7): ")  
  
        if choice in ('1', '2', '3', '4', '5'):  
            num1 = float(input("Enter first number: "))  
            num2 = float(input("Enter second number: "))  
            if choice == '1':  
                num = float(input("Enter a number: "))  
            elif choice == '7':  
                print("Exiting the calculator. Goodbye!")  
                return
```

Ln: 66 Col: 0

```
Welcome to Simple Calculator with Advanced Features!  
Operations:  
1. Addition  
2. Subtraction  
3. Multiplication  
4. Division  
5. Power  
6. Square Root  
7. Exit  
Enter operation number (1-7): 2  
Enter first number: 5  
Enter second number: 8  
Result: -3.0  
Enter operation number (1-7): 1  
Enter first number: 8  
Enter second number: 9  
Result: 17.0  
Enter operation number (1-7): 3  
Enter first number: 5  
Enter second number: 2  
Result: 10.0  
Enter operation number (1-7): 4  
Enter first number: 6  
Enter second number: 3  
Result: 2.0  
Enter operation number (1-7): 5  
Enter first number: 4  
Enter second number: 8  
Result: 65536.0  
Enter operation number (1-7): 6  
Enter a number: 3  
Result: 1.7320508075688772  
Enter operation number (1-7): 6  
Enter a number: 4  
Result: 2.0  
Enter operation number (1-7): 5  
Enter first number: 2  
Enter second number: 2  
Result: 4.0  
Enter operation number (1-7):
```

Ln: 44 Col: 30

TASK – 2

TITLE

CodTech IT Solutions Internship - Task Documentation: “Simple Python Chatbot”

INTERN INFORMATION

Name: Kandagatla Neetha

ID: C0D5998

INTRODUCTION

In the realm of artificial intelligence, chatbots have emerged as powerful tools for engaging users in conversation, providing assistance, and delivering information. These bots are capable of understanding natural language input and responding appropriately, making them invaluable for customer service, information retrieval, and even entertainment purposes.

In this, we'll explore how to develop a basic Python chatbot using natural language processing (NLP) techniques. Our chatbot will be able to understand user queries and respond with relevant information or assistance. To accomplish this, we'll leverage the Natural Language Toolkit (NLTK), a popular library for NLP tasks in Python.

Key Components of the Chatbot:

1. Natural Language Understanding: The chatbot will employ NLP techniques to interpret user input. This includes tokenization, part-of-speech tagging, and pattern matching to identify the user's intent.

2. Response Generation: Based on the analysis of user input, the chatbot will generate appropriate responses. This may involve selecting predefined responses from a set of pattern-response pairs or generating dynamic responses based on the context of the conversation.

3. Reflections: To enhance the conversational experience, the chatbot will utilize reflections to handle personal pronouns. This allows the bot to maintain context and provide more natural-sounding responses.

Implementation (Code)

```
nltk.py - C:/Users/kanda/nltk.py (3.12.2)
File Edit Format Run Options Window Help

import nltk
from nltk.chat.util import Chat, reflections

# Define reflections to handle personal pronouns
reflections = {
    "i am": "you are",
    "i was": "you were",
    "i": "you",
    "i'm": "you are",
    "i'd": "you would",
    "i've": "you have",
    "i'll": "you will",
    "my": "your",
    "you are": "I am",
    "you were": "I was",
    "your": "my",
    "yours": "mine",
    "you": "me",
    "me": "you",
}

# Define pattern-response pairs for the chatbot
pairs = [
    [
        r"my name is (.*)",
        ["Hello %1, how can I help you today?"]
    ],
    [
        r"what is your name?",
        ["My name is ChatBot and I'm here to assist you."]
    ],
    [
        r"how are you?",
        ["I'm doing well, thank you!"]
    ],
    [
        r"(.*) your name(.*)",
        ["My name is ChatBot. How can I assist you?"]
    ],
    [
        r"(.*) (good|great|fine|well)",
        ["That's good to hear!"]
    ],
    [
        r"(.*) (sorry|apologies|excuse me)",
        ["No problem, it's alright."]
    ],
    [
        r"(.*) (help|assist)(.*)",
        ["Sure, I'd be happy to help. What do you need assistance with?"]
    ],
    [
        r"quit",
        ["Bye! Take care."]
    ],
    [
        r"(.*)",
        ["I'm sorry, I didn't quite understand. Can you please rephrase?"]
    ]
]

# Create a chatbot instance
chatbot = Chat(pairs, reflections)

# Start conversation with the user
print("Hello, I'm ChatBot. How can I assist you today?")
while True:
    user_input = input("You: ")
    response = chatbot.respond(user_input)
    print("ChatBot:", response)
```

CODE EXPLANATION

Python chatbot step by step:

1. Importing Libraries:

```
python
import nltk
from nltk.chat.util import Chat, reflections
```

- We import the necessary modules from the NLTK library. Chat is a class that allows us to create a chatbot, and reflections is a dictionary that helps the bot handle personal pronouns.

2. Reflections:

```
python
reflections = {
    "i am": "you are",
    "i was": "you were",
    # Other reflection mappings...
}
```

- Reflections map first-person pronouns to second-person pronouns and vice versa. This helps the bot respond more naturally.

3. Pattern-Response Pairs:

```
python
pairs = [
    [
        r"my name is (.*)",
        ["Hello %1, how can I help you today?"]
    ],
    # Other pattern-response pairs...
]
```

- These pairs consist of regular expressions that match user input and corresponding responses. For example, if the user says, "my name is Neetha," the bot responds with "Hello Neetha, how can I help you today?"

4. Creating the Chatbot:

```
python  
chatbot = Chat(pairs, reflections)
```

- We create a Chat instance by passing the pattern-response pairs and the reflections dictionary.

5. Starting the Conversation:

```
python  
print("Hello, I'm ChatBot. How can I assist you today?")  
while True:  
    user_input = input("You: ")  
    response = chatbot.respond(user_input)  
    print("ChatBot:", response)
```

- The bot greets the user and enters a loop where it continuously prompts the user for input. It then responds based on the user's input by calling the respond() method of the chatbot instance.

6. Handling User Input:

- The chatbot matches the user input against the defined patterns. If a match is found, it selects a corresponding response. If no match is found, it defaults to a generic response.

Overall, this code provides a basic framework for creating a Python chatbot using NLTK. It demonstrates how to define patterns, generate responses, and handle user interactions in a conversational manner.

USAGE

We can use the above code as a foundation to create our own Python chatbot tailored to our specific use case. Here's how we can utilize the code:

1. Customize Patterns and Responses: Modify the pairs list to include patterns and responses relevant to the domain or topic based on that the chatbot will handle. Add more pattern-response pairs to cover a wider range of user queries.

2. Extend Functionality: Expand the functionality of the chatbot by adding more sophisticated natural language processing techniques or integrating with external APIs for data retrieval. For example, we can incorporate sentiment analysis, entity recognition, or knowledge graph querying to enhance the bot's capabilities.

3. Integrate with Messaging Platforms: Integrate the chatbot with messaging platforms like Slack, Facebook Messenger, or Telegram to make it accessible to users on those platforms.

We can use libraries or frameworks specifically designed for chatbot development in these environments.

4. Deploy to Production: Once your chatbot is ready, deploy it to a server or cloud platform to make it accessible to users 24/7. You can use platforms like Heroku, AWS Lambda, or Google Cloud Functions for deployment.

5. Continuous Improvement: Monitor user interactions and feedback to iteratively improve the chatbot's performance. Analyze user queries that the bot couldn't handle effectively and update the patterns and responses accordingly.

6. Documentation and Support: Provide documentation and support for users who interact with the chatbot. Include instructions on how to use the bot effectively and troubleshoot common issues.

CONCLUSION

The provided Python code serves as a foundational framework for developing a simple chatbot using natural language processing techniques. By following this example, you can create a chatbot capable of understanding user queries and responding appropriately. Here are the key takeaways:

1. Basic Chatbot Functionality: The code demonstrates how to define pattern-response pairs to enable the chatbot to recognize user input and generate relevant responses. It covers common conversational scenarios such as greetings, inquiries about the bot's name, and requests for assistance.

2. Natural Language Understanding: Utilizing the Natural Language Toolkit (NLTK), the chatbot employs natural language processing techniques such as tokenization, part-of-speech tagging, and pattern matching to interpret user queries.

3. Reflections: The inclusion of reflections allows the chatbot to handle personal pronouns, enhancing the conversational experience by maintaining context and providing more natural-sounding responses.

4. Customization and Extension: Users can customize the chatbot by adding or modifying pattern-response pairs to suit their specific use case. Additionally, the code can be extended to incorporate more sophisticated NLP techniques or integrate with external APIs for additional functionality.

5. Deployment and Iterative Improvement: Once developed, the chatbot can be deployed to production environments, such as messaging platforms or web applications. Continuous monitoring and feedback analysis enable iterative improvement to enhance the chatbot's performance and user satisfaction.

Overall, the provided code serves as a starting point for building a Python chatbot and demonstrates the potential of leveraging natural language processing to create engaging conversational interfaces. With further customization and refinement, chatbots can become valuable assets in various domains, providing assistance, information, and support to users.