MIT World Peace University

Python Programming
Second Year B. Tech, Semester 4

Exception Handling in Python

Assignment No. 3

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 20

April 24, 2023

Contents

1 Aim

To write a python program that accepts the length of three sides of a triangle as inputs. The program should indicate whether or not the triangle is a right-angled triangle using function with exception handling.

2 Objectives

1. To learn and Implement Exception and Function handling in Python.

3 Problem Statement

To check whether or not the triangle is a right-angled triangle using function with exception handling.

4 Theory

4.1 Python Functions

In Python, a function is a block of reusable code that performs a specific task. Functions are used to organize code into modules, which can be easily reused and maintained.

```
def multiply(x, y):
  """
  This function multiplies two numbers and returns the result
  """
  return x * y


# To call it,
result = multiply(3, 4)
print(result)

# This will output 12, since 3 * 4 is 12.
```

Functions can also have optional arguments, default values, and can return multiple values. Here's an example of a function that takes an optional argument:

```
def greet(name, greeting='Hello'):
  """
  This function greets a person with a given message.
  """
  print(greeting, name)

greet('John') # Output: Hello John
greet('Mary', 'Hi') # Output: Hi Mary
```

In this example, the greeting parameter is optional and has a default value of 'Hello'. If we don't specify a greeting when calling the function, it will use the default value. If we do specify a greeting, it will use the value we provide.

Functions are an important tool in Python programming and are used extensively in many applications. They allow us to write reusable code that can be easily maintained and modified.

4.2 Exception Handling

In Python, exceptions are used to handle runtime errors that occur during program execution. When an error occurs, Python raises an exception, which can be caught and handled by the program.

4.2.1 Raising Exceptions

Python exceptions can be raised explicitly using the raise keyword. This can be done in response to an error condition that the program encounters. For example:

```
if x < 0:
  raise ValueError("x must be positive")
```

This code raises a ValueError exception with the message "x must be positive" if the value of x is negative.

4.2.2 Catching Exceptions

Exceptions can be caught and handled using the try and except keywords. The code that might raise an exception is placed inside a try block, and any exceptions that occur are caught and handled in the corresponding except block. For example:

```
try:
    result = 10 / 0
except ZeroDivisionError:
    print("Division by zero error")
```

here, this code attempts to divide 10 by zero, which raises a ZeroDivisionError exception. The exception is caught by the except block, which prints an error message.

4.2.3 Handling Multiple Exceptions

Python allows handling multiple exceptions in a single try block. This is done by providing multiple except blocks, each handling a different type of exception. For example:

```
try:
    result = int("not a number")
except ValueError:
    print("ValueError: invalid literal for int() with base 10")
except TypeError:
    print("TypeError: int() argument must be a string or a number")
This code attempts to convert the string "not a number" to an integer, which
   raises a ValueError exception. The except block for ValueError is executed,
   printing an error message.
```

4.2.4 Finally Block

Python provides a finally block that can be used to execute cleanup code, regardless of whether an exception was raised or not. The code inside the finally block is executed after the try and except blocks. For example:

```python
try:
    file = open("example.txt")
    # perform some operations on the file
except IOError:
    print("Error: could not open file")
finally:
    file.close()
```

This code opens a file and performs some operations on it, which may raise an IOError exception. The finally block ensures that the file is closed, regardless of whether an exception was raised or not.

4.3 How does Exception handling happen in python?

When an error occurs in a program, Python raises an exception. This exception can be caught and handled by the program, allowing it to recover from the error and continue executing.

Behind the scenes, Python maintains a stack of "frames" representing the current execution context. When an exception occurs, Python looks through the frames on the stack in reverse order to find a "try" block that can handle the exception. A try block is a section of code that may raise an exception, but also contains code to handle the exception if it occurs.

If Python finds a try block that matches the type of exception raised, it jumps to the corresponding "except" block and executes it. The except block may contain code to handle the exception and recover from the error, or it may simply log the error and exit the program.

If Python does not find a matching try block on the stack, the program terminates and prints a traceback message showing the location where the exception occurred and the sequence of function calls that led to the error.

5 Platform

Operating System: Arch Linux x86-64
IDEs or Text Editors Used: Visual Studio Code with Jupyter
Interpreter: python 3.10.8

6 Libraries Used with pip

No additional Libraries are used with pip. The only libraries used are the default libraries that come with python.

7 Input

1. Accepts the length of three sides of a triangle as inputs from the user

8 Output

1. Display whether or not the triangle is a right-angled triangle.

9 Code

Write a python program that accepts the length of three sides of a triangle as inputs. The program should indicate whether or not the triangle is a right angled triangle using a user defined function with exception handling.

[9]:
```python
import math


def check_right_angled_triangle(side_1, side_2, side_3):
    """This function takes three sides of a triangle as input and returns True␣
 ↪if the triangle is right-angled and False otherwise.
    """
    longest_side = max(side_1, side_2, side_3)
    print(longest_side)
    if math.pow(longest_side, 2) == (math.pow(side_1, 2) + math.pow(side_2, 2) +␣
 ↪math.pow(side_3, 2) - math.pow(longest_side, 2)):
        return True
    else:
        return False
```

[16]:
```python
def main():
    print("Welcome to Python Assignment 3")
    print("Enter the sides of the triangle to check if it is a right angled␣
 ↪triangle")
    try:
        side_1 = int(input("Enter the First side: "))
        side_2 = int(input("Enter the Second side: "))
        side_3 = int(input("Enter the Third side: "))
        if check_right_angled_triangle(side_1, side_2, side_3):
            print("The triangle is a right angled triangle")
        else:
            print("The triangle is not a right angled triangle")
    # except ValueError:
    #     print("Value Error Occured: Please enter a valid number")
    except Exception as e:
        print("An error occurred: ", e)
```

[18]:
```python
main()
```

```
Welcome to Python Assignment 3
Enter the sides of the triangle to check if it is a right angled triangle
An error occurred:  invalid literal for int() with base 10: 'hj'
```

[15]:
```
main()
```

```
Welcome to Python Assignment 3
Enter the sides of the triangle to check if it is a right angled triangle
5
The triangle is a right angled triangle
```

[20]:
```
main()
```

```
Welcome to Python Assignment 3
Enter the sides of the triangle to check if it is a right angled triangle
An error occurred:  invalid literal for int() with base 10: '-af'
```

9.1 Assignment 3 - Part 2

Given a list l1, find reciprocal of each element in the list. Use Error Handling

[34]:
```python
print("What do you want in the list? ")
input_list = [10, 5.3, 'x', 4, 0]
print("The Input list is:", input_list)
```

```
What do you want in the list?
The Input list is: [10, 5.3, 'x', 4, 0]
```

[35]:
```python
i = 0
print("The reciprocals are: ")
while i < len(input_list) - 1:
    try:
        i += 1
        print(1/input_list[i])
    except TypeError:
        print("TypeError: Please enter a valid number")
    except ZeroDivisionError:
        print("ZeroDivisionError: Please enter a number other than 0")
    except Exception as e:
        print("An error occurred: ", e)
```

```
The reciprocals are:
0.18867924528301888
TypeError: Please enter a valid number
0.25
ZeroDivisionError: Please enter a number other than 0
```

[ ]:
```
```

10 Conclusion

Thus, We have studied python funtions along with exception handling and implemented it in the above program.

11 FAQ

1. **Why should use an exception?**

   Exceptions are convenient in many ways for handling errors and special conditions in a program. When you think that you have a code which can produce an error then you can use exception handling. It handles the error by its own and your code will be executed without any interruption.

   There are many reasons to use exceptions, not just in python, but in all programming languages.

   (a) **Separation of Error Handling Code from Regular Code:** Exceptions allow you to separate error handling code from regular code. This is a good thing, because it allows you to focus on the regular code, and not worry about error handling until something goes wrong.

   (b) **Propagating Errors Up the Call Stack:** Exceptions allow you to propagate errors up the call stack, until you find a function that can handle the error. This is a good thing, because it allows you to write code that can handle errors in a centralized location, instead of having to handle errors in every function.

   (c) **Handling Errors in a Central Location:** Exceptions allow you to handle errors in a central location, instead of having to handle errors in every function. This is a good thing, because it allows you to write code that can handle errors in a centralized location, instead of having to handle errors in every function.

2. **Why functions are needed in Python?**

   Functions are a convenient way to divide your code into useful blocks, allowing us to order our code, make it more readable, reuse it and save some time. Also functions are a key way to define interfaces so programmers can share their code.

   There are many reasons to use functions:

   (a) **Modularity:** Functions allow you to break down the problem into smaller parts. This makes your code easier to read and maintain.

   (b) **Reuseability:** Functions can be reused in different programs. This saves time and effort.

   (c) **Scoping:** Functions allow you to define variables that are only used within the function's scope. This avoids name collisions with variables in other parts of the program.

   (d) **Namespacing:** Functions allow you to define variables that are only used within the function's scope. This avoids name collisions with variables in other parts of the program.

   (e) **Testing and Debugging:** Functions allow you to test parts of your program in isolation. This makes it easier to find and fix bugs.

   (f) **Composition:** Functions allow you to break down the problem into smaller parts. This makes your code easier to read and maintain.