PYTHON IS FOR EVERYONE

Tutorial 11:

PYTHON PROGRAMMING - EXCEPTION HANDLING IN GOOGLE COLAB

Jeff Gentry

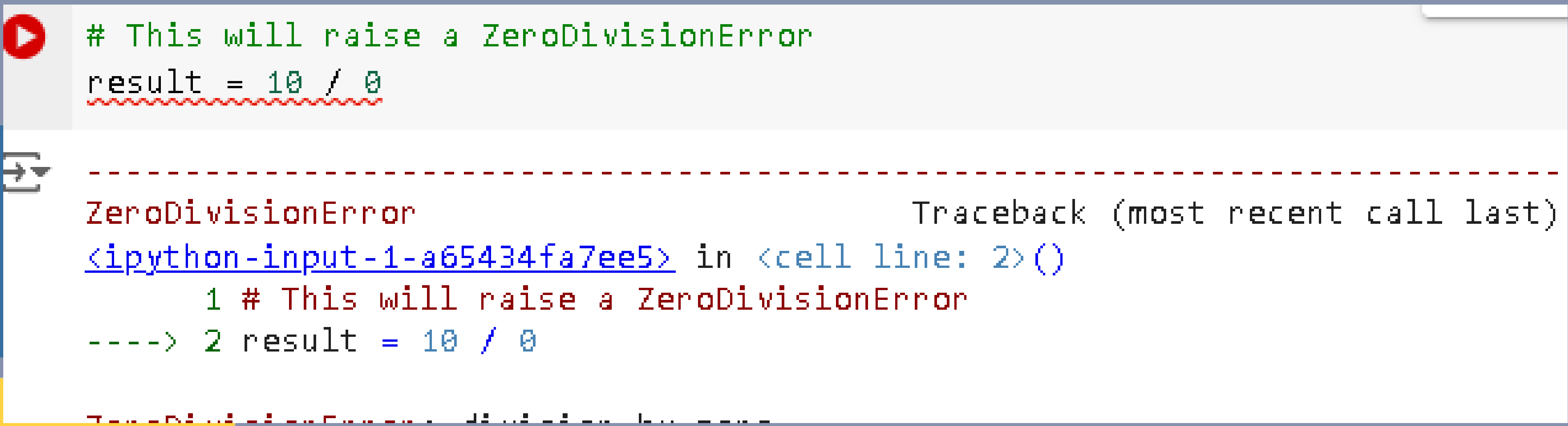
@www.linkedin.com/in/jefferycharlesgentry

Objectives

- Understand what exceptions are and why they occur.
- Learn how to use “try”, “except”, “else”, and “finally” blocks.
- Explore how to raise exceptions and create custom exceptions.
- Practice exception handling through hands-on exercises.

What are Exceptions?

Exceptions are errors that occur during the execution of a program. They can arise from various issues, such as invalid input, file not found, division by zero, etc. If not handled, exceptions can cause the program to crash.



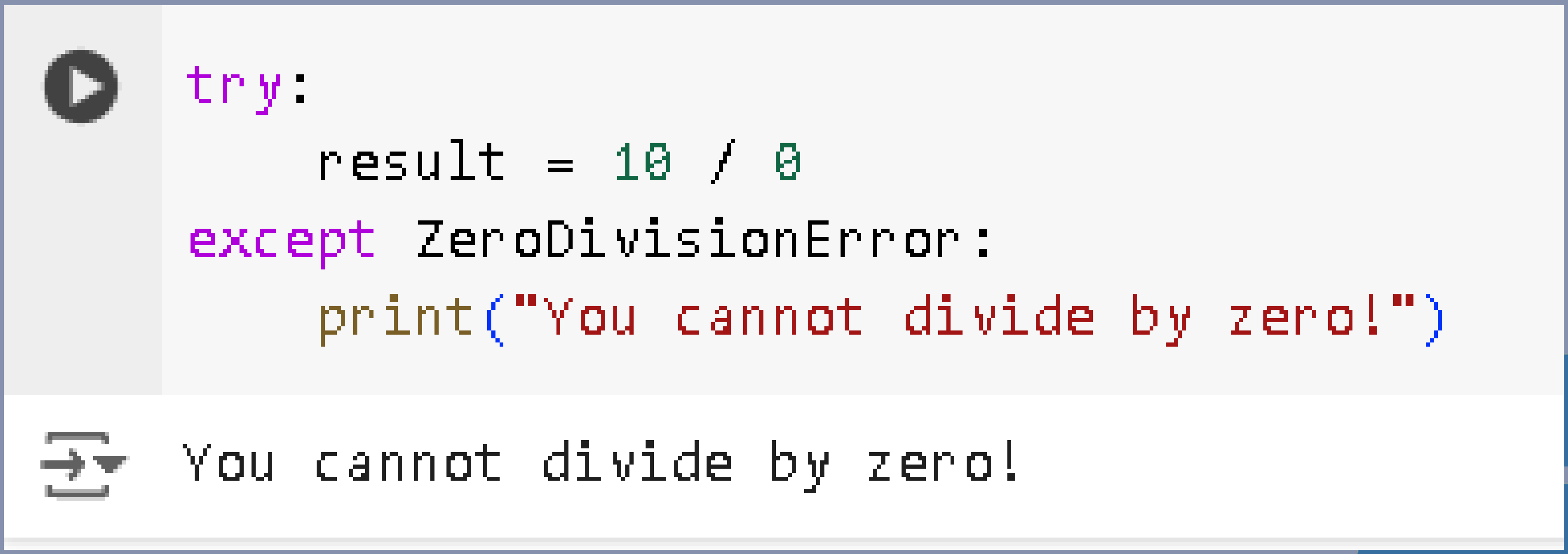
```python
# This will raise a ZeroDivisionError
result = 10 / 0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-1-a65434fa7ee5> in <cell line: 2>()
      1 # This will raise a ZeroDivisionError
----> 2 result = 10 / 0

ZeroDivisionError: division by zero
```

Example of an Exception

Using “try” and “except”

You can handle exceptions using try and except blocks. The code that may raise an exception is placed inside the try block, and the code to handle the exception is placed in the except block.

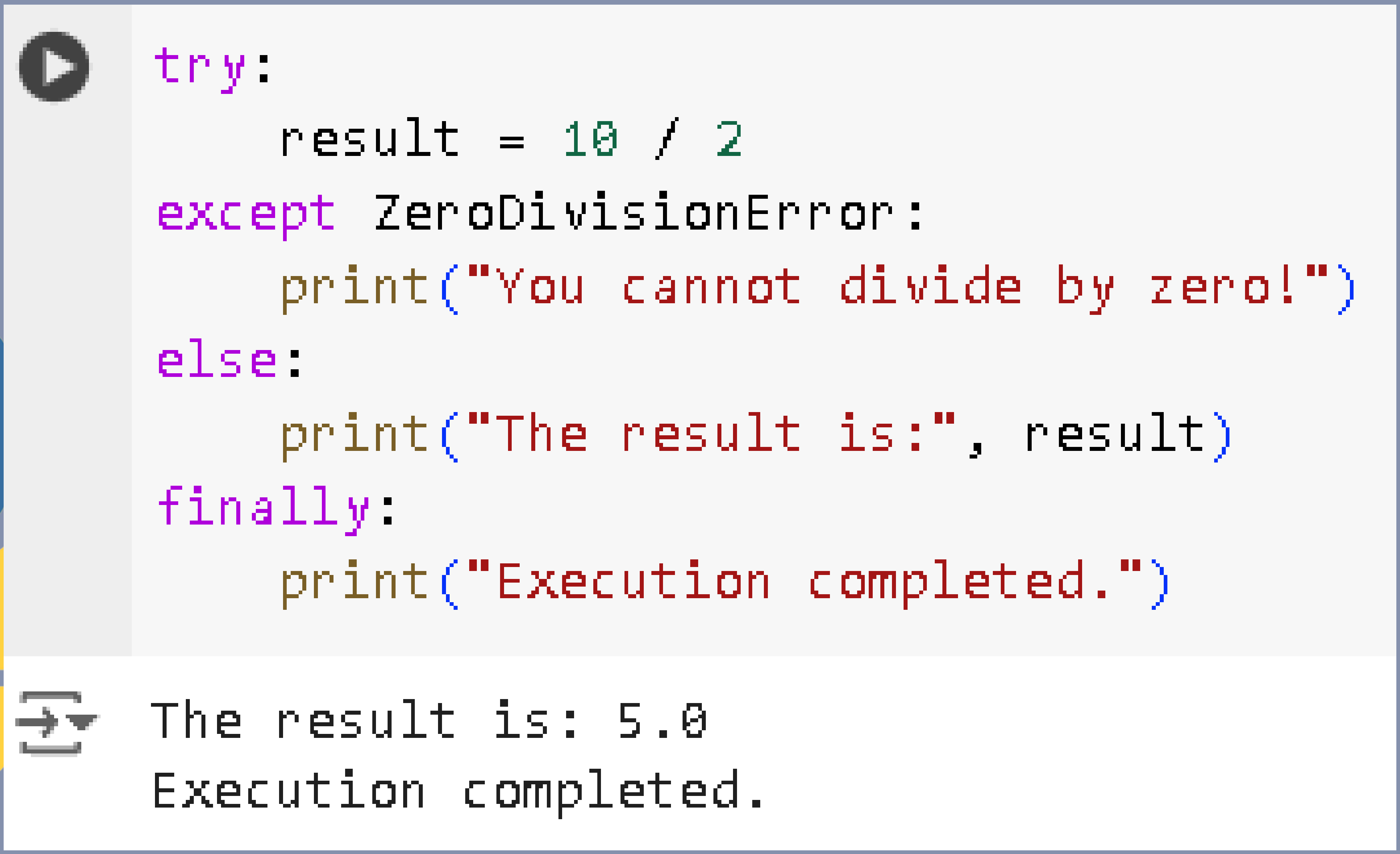


```
try:
    result = 10 / 0
except ZeroDivisionError:
    print("You cannot divide by zero!")
```

```
You cannot divide by zero!
```

Using “else” and “finally”

- “else”: The code inside the “else” block runs if the “try” block does not raise an exception.
- “finally”: The code inside the “finally” block runs regardless of whether an exception occurred or not. It is often used for cleanup actions.

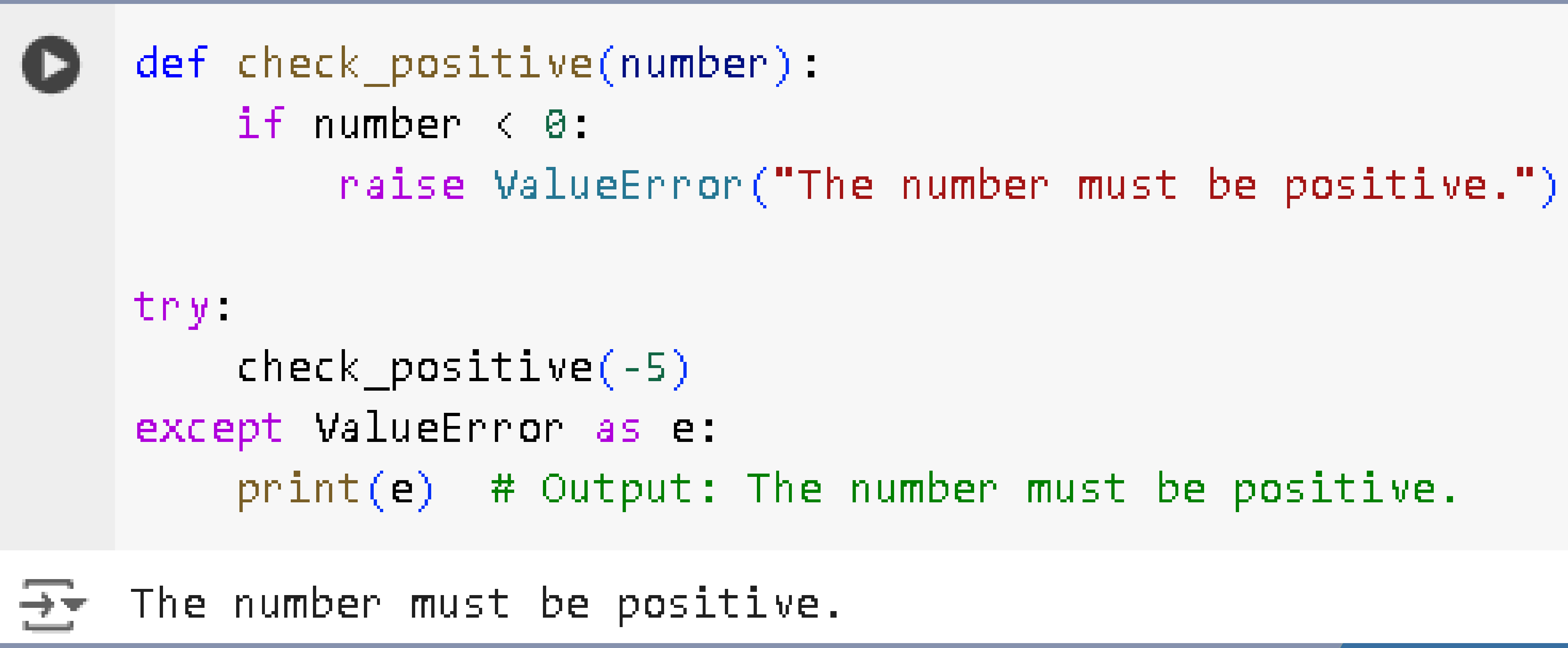


```
try:
    result = 10 / 2
except ZeroDivisionError:
    print("You cannot divide by zero!")
else:
    print("The result is:", result)
finally:
    print("Execution completed.")
```

```
The result is: 5.0
Execution completed.
```

Raising Exceptions

You can raise exceptions intentionally using the “raise” statement. This is useful for enforcing certain conditions in your code.

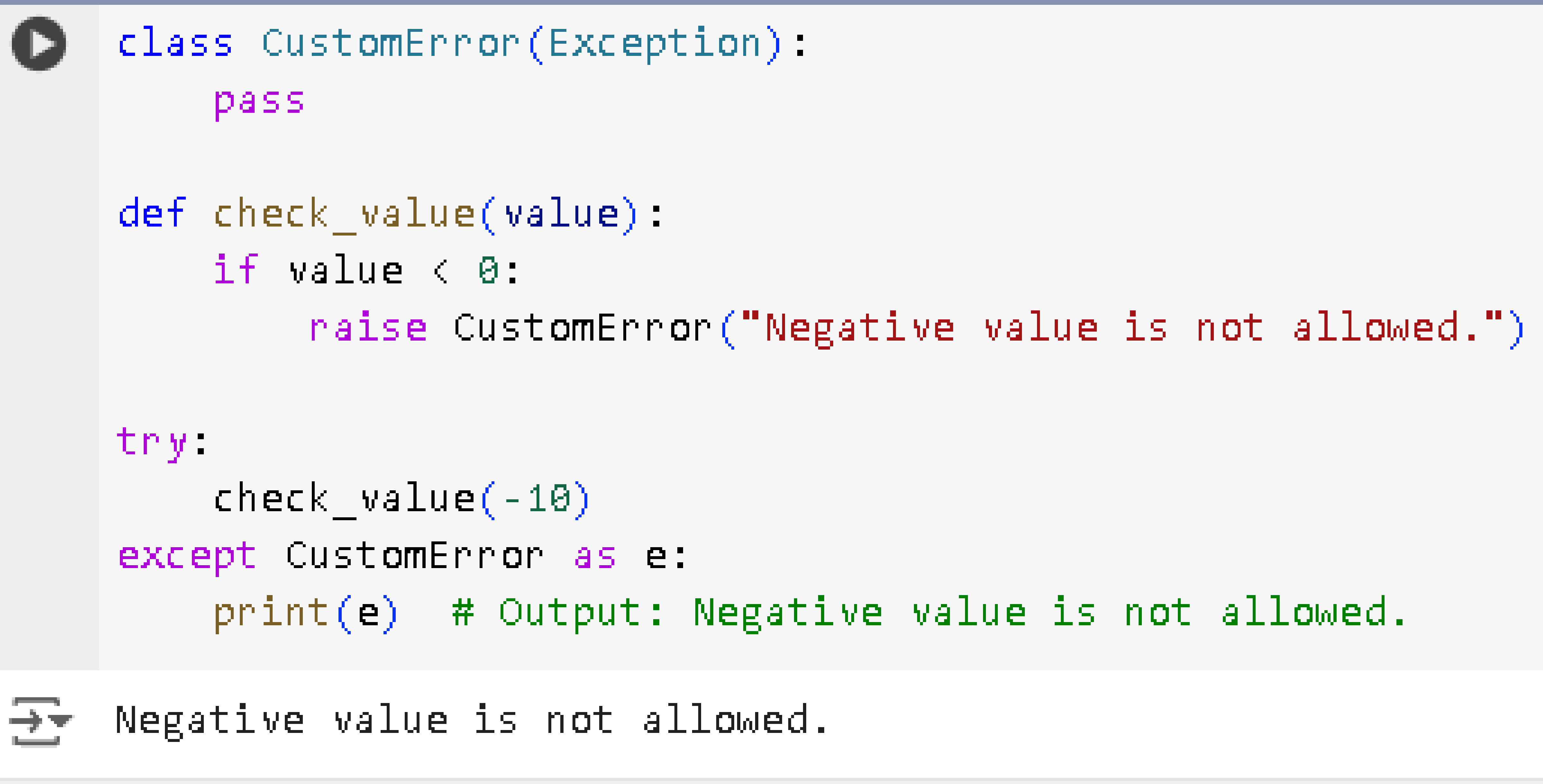


```python
def check_positive(number):
    if number < 0:
        raise ValueError("The number must be positive.")

try:
    check_positive(-5)
except ValueError as e:
    print(e)  # Output: The number must be positive.
```

```
The number must be positive.
```

Creating Custom Exceptions

You can create your own exception classes by inheriting from the built-in “Exception” class. This allows you to define specific error types for your application.

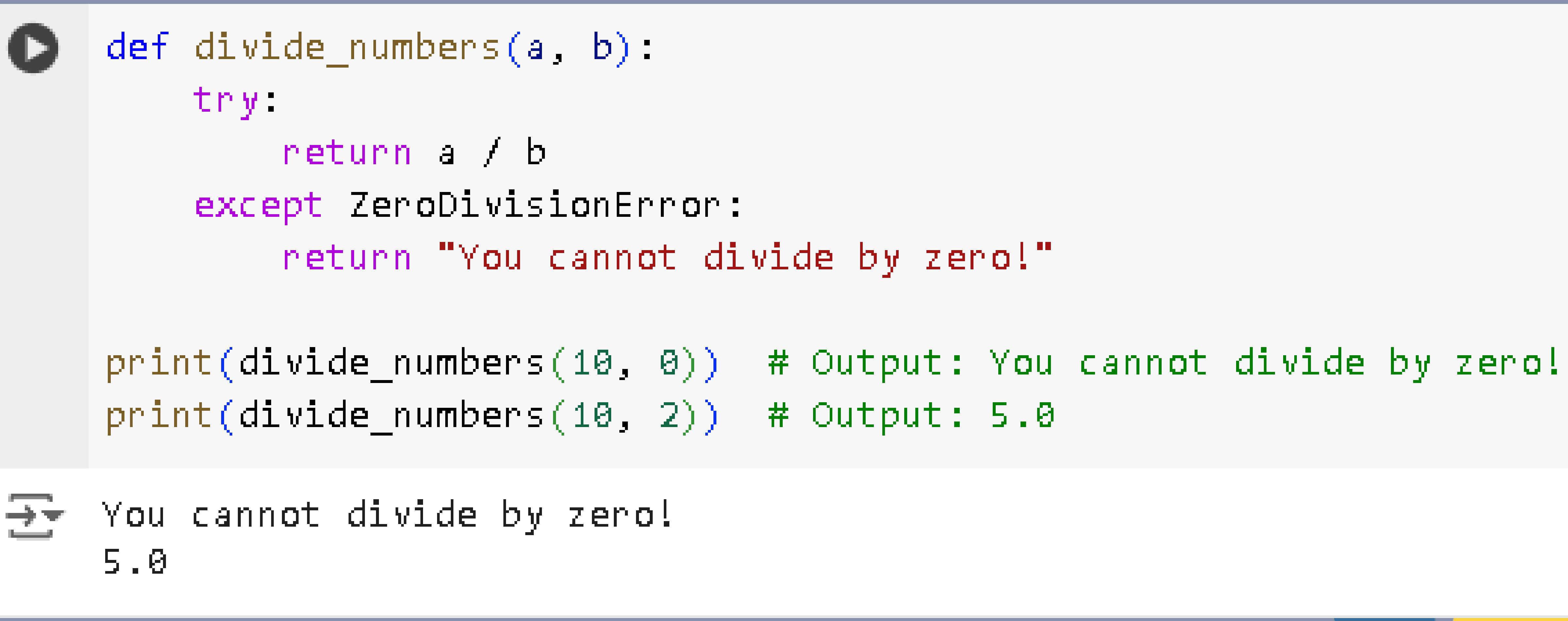


```python
class CustomError(Exception):
    pass

def check_value(value):
    if value < 0:
        raise CustomError("Negative value is not allowed.")

try:
    check_value(-10)
except CustomError as e:
    print(e)  # Output: Negative value is not allowed.
```

```
Negative value is not allowed.
```

Practice Exercises

Division with Exception Handling: Write a program that takes two numbers as input and performs division, handling any division by zero errors.

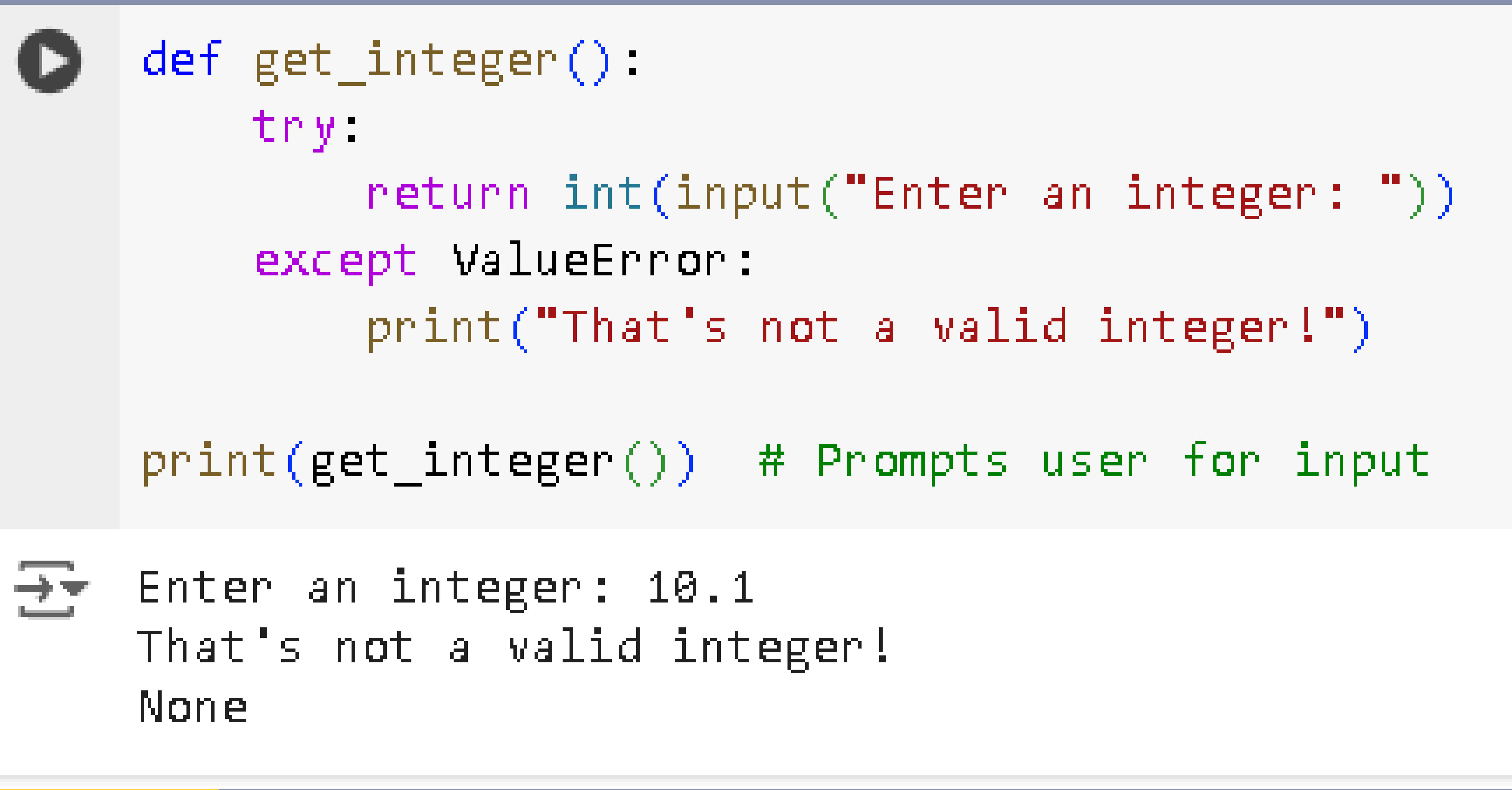


```
def divide_numbers(a, b):
    try:
        return a / b
    except ZeroDivisionError:
        return "You cannot divide by zero!"

print(divide_numbers(10, 0))  # Output: You cannot divide by zero!
print(divide_numbers(10, 2))  # Output: 5.0
```

```
You cannot divide by zero!
5.0
```

Handling Exceptions

Input Validation: Write a program that prompts the user for a number and raises an exception if the input is not a valid integer.

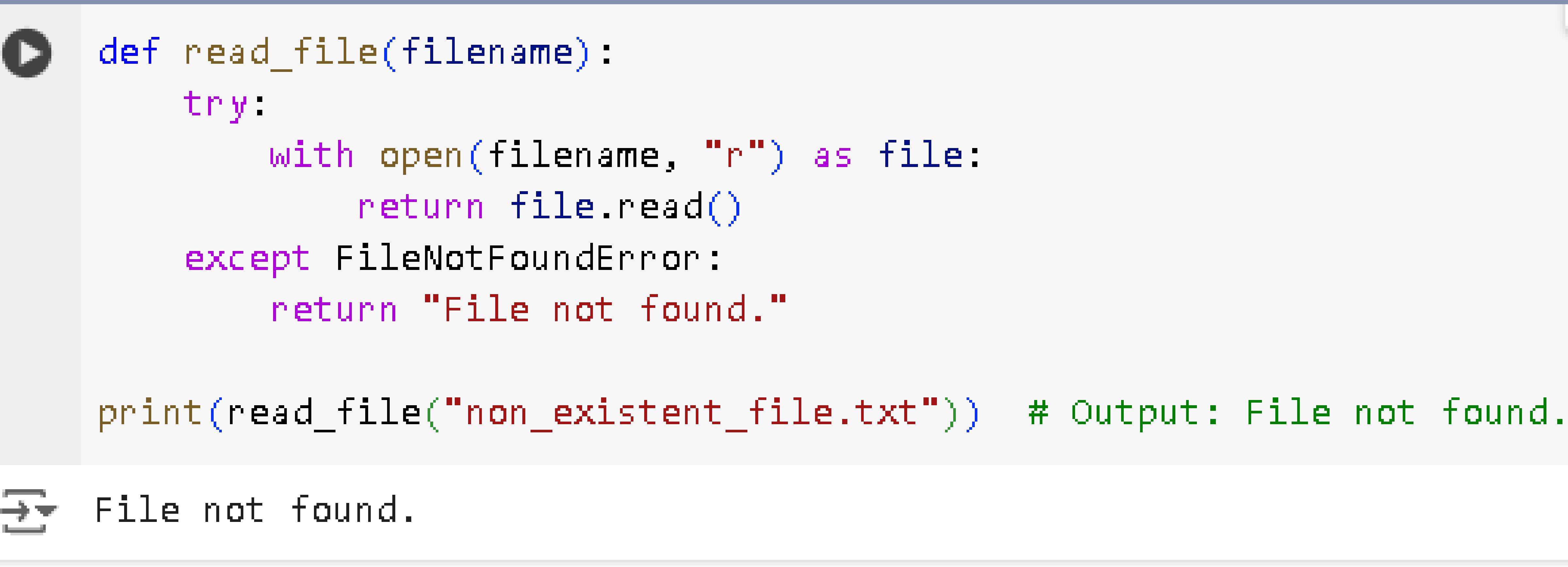


```python
def get_integer():
    try:
        return int(input("Enter an integer: "))
    except ValueError:
        print("That's not a valid integer!")

print(get_integer())  # Prompts user for input
```

```
Enter an integer: 10.1
That's not a valid integer!
None
```

Practice Exercises

File Reading with Exception Handling: Write a program that attempts to read a file and handles the case where the file does not exist.

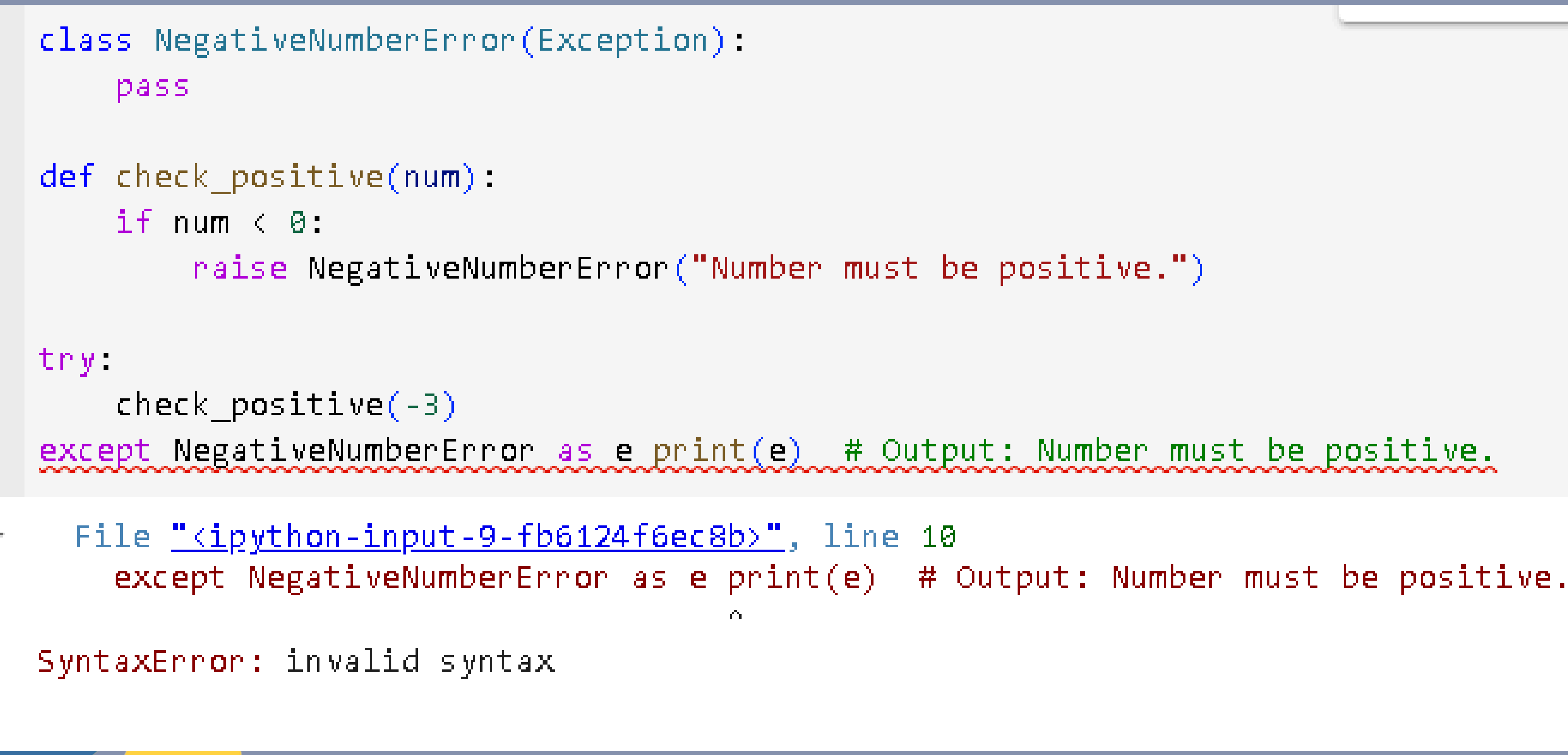


```python
def read_file(filename):
    try:
        with open(filename, "r") as file:
            return file.read()
    except FileNotFoundError:
        return "File not found."

print(read_file("non_existent_file.txt"))  # Output: File not found.
```

```
File not found.
```

Practice Exercises

Custom Exception: Create a program that checks if a number is positive and raises a custom exception if it is not.

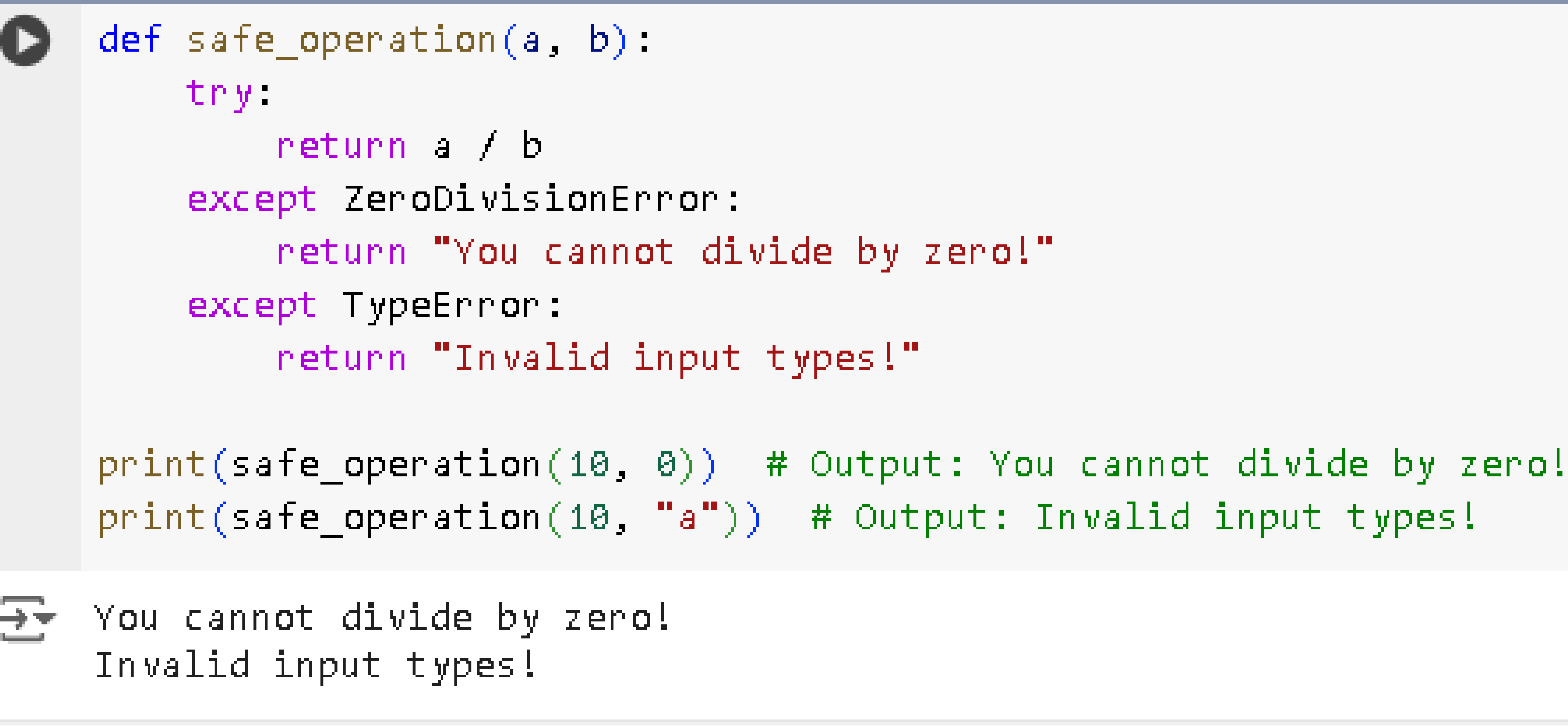


```python
class NegativeNumberError(Exception):
    pass

def check_positive(num):
    if num < 0:
        raise NegativeNumberError("Number must be positive.")

try:
    check_positive(-3)
except NegativeNumberError as e print(e)  # Output: Number must be positive.
```

```
  File "<ipython-input-9-fb6124f6ec8b>", line 10
    except NegativeNumberError as e print(e)  # Output: Number must be positive.
                                        ^
SyntaxError: invalid syntax
```

Practice Exercises

Multiple Exceptions: Write a program that handles multiple types of exceptions when performing operations.



```python
def safe_operation(a, b):
    try:
        return a / b
    except ZeroDivisionError:
        return "You cannot divide by zero!"
    except TypeError:
        return "Invalid input types!"

print(safe_operation(10, 0))  # Output: You cannot divide by zero!
print(safe_operation(10, "a"))  # Output: Invalid input types!
```

```
You cannot divide by zero!
Invalid input types!
```

Conclusion

In this tutorial, you learned about exception handling in Python, including how to use “try”, “except”, “else”, and “finally” blocks. You also explored how to raise exceptions and create custom exceptions. Exception handling is crucial for building robust applications that can gracefully handle errors and unexpected situations.

Next Steps

In tutorial 12, we will discover how to import and use modules, as well as how to leverage Python's standard libraries for various tasks.

FOLLOW ME

for more tips you didn't know you needed

Jeff Gentry

@www.linkedin.com/in/jefferycharlesgentry