10. Exception Handling

September 17, 2025

1 10. Exception Handling (`try` statement)

Exception handling in Python is a way to deal with errors that might occur during the execution of your code. It allows you to gracefully handle these errors and prevent your program from crashing unexpectedly.

1.0.1 What is an exception?

An exception is an error that occurs during the execution of a program. These can be caused by various reasons such as invalid input, file not found, division by zero, etc.

1.0.2 Why handle exceptions?

Handling exceptions is important because it allows your program to continue running even when it encounters errors. Instead of crashing, you can catch these errors and take appropriate actions, such as displaying a user-friendly error message or trying an alternative approach.

1.1 10.1 The `try` statement

In Python, you handle exceptions using a try-except block. Here's the basic structure:

```
try:
    # Code that might raise an exception
except ExceptionName:
    # Code to handle the exception
```

Let's say you're trying to open a file. If the file doesn't exist, it will raise a `FileNotFoundError`. You can handle it like this:

[1]:
```
try:
    file = open("myfile.txt", "r")
except FileNotFoundError:
    print("File not found!")
```

```
File not found!
```

You can handle different types of exceptions separately or together:

"'python try: # Code that might raise an exception except (Exception1, Exception2): # Code to handle Exception1 or Exception2 except Exception3: # Code to handle Exception3 ...

[2]:
```
try:
    file = open("myfile.txt", "r")
except FileNotFoundError:
    print("File not found!")
except PermissionError:
    print("Permission denied.")
```

```
File not found!
```

You can also use a finally block to execute code regardless of whether an exception was raised or not. It's useful for cleanup operations (like closing a file, closing a database connection):

```
try:
    # Code that might raise an exception
except Exception:
    # Code to handle the exception
finally:
    # Code that will always execute
```

1.2 Exercises 10.1

1. Write a program to read two numbers: `x` and `y` from standard input and print the result of `x / y`. If the user inputs invalid data (non-numeric input or `0` for `y`), display an error message and exit gracefully.
2. Modify the previous program so that it asks the user to re-enter data until valid.

1.3 10.2 Explicitly raising exceptions (`raise` statement)

In Python, exceptions are raised when errors occur at runtime. We can also manually raise exceptions using the `raise` statement.

[3]:
```
raise ValueError
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[3], line 1
----> 1 raise ValueError

ValueError:
```

We can optionally pass values to the exception to clarify why that exception was raised.

[4]:
```
raise ValueError('Invalid value.')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[4], line 1
----> 1 raise ValueError('Invalid value.')
```

```
ValueError: Invalid value.
```

You will usually want to raise an exception when a certain business rule is violated:

[5]:
```python
def calculate_area(length, width):
    if length <= 0 or width <= 0:
        raise ValueError("Length and width must be positive numbers.")
    return length * width
```

[6]:
```python
calculate_area(10, 0)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[6], line 1
----> 1 calculate_area(10, 0)

Cell In[5], line 3, in calculate_area(length, width)
      1 def calculate_area(length, width):
      2     if length <= 0 or width <= 0:
----> 3         raise ValueError("Length and width must be positive numbers.")
      4     return length * width

ValueError: Length and width must be positive numbers.
```

1.4 Exercises 10.2

1. Write a function `convert_temperature(temp, scale)` that converts a temperature from one scale to another. The function should accept two arguments:

- `temp`: The temperature value to be converted (a `float` or an `int`).
- `scale`: The scale of the temperature, which can be either `"C"` for Celsius or `"F"` for Fahrenheit.

The conversion formulas are as follows:

```
F = 9 / 5 * C + 32
C = (F - 32) * 5 / 9
```

If the scale provided is neither `"C"` nor `"F"`, the function should raise a `ValueError` with an appropriate error message.