Writing Your Own Exception Classes

In Python, you can create your own custom exception classes by inheriting from the built-in `Exception` class or one of its subclasses. This allows you to define and raise exceptions that are specific to your application or module. Here's a guide on how to write your own exception classes with examples:

### Writing a Simple Custom Exception:

```python
class CustomError(Exception):
    pass

# Raise the custom exception
raise CustomError("This is a custom exception")
```

In this example, we define a custom exception named `CustomError` that inherits from the built-in `Exception` class. Then, we raise an instance of this custom exception with a specific error message.

### Adding Custom Attributes to Exceptions:

You can add additional attributes to your custom exception by defining an `__init__` method.

```python
class CustomValueError(Exception):
    def __init__(self, value, message="Custom ValueError occurred"):
        self.value = value
        self.message = message
        super().__init__(self.message)

# Raise the custom exception with custom attributes
raise CustomValueError(42, "Invalid value")
```

Here, `CustomValueError` has a custom `__init__` method that takes a `value` parameter and an optional `message` parameter. This allows you to attach additional information to your exception.

### Creating a Hierarchy of Custom Exceptions:

You can create a hierarchy of custom exceptions by defining subclasses.

```python
class CustomError(Exception):
    pass

class CustomValueError(CustomError):
    pass
```

```
class CustomTypeError(CustomError):
    pass

# Raise a specific custom exception
raise CustomTypeError("This is a custom TypeError")
```

In this example, `CustomValueError` and `CustomTypeError` are subclasses of `CustomError`. You can catch these exceptions separately or catch the more general `CustomError` if you want to handle them in a more unified way.

### Using Custom Exceptions in Your Code:

```python
class WithdrawError(Exception):
    def __init__(self, amount, balance):
        self.amount = amount
        self.balance = balance
        super().__init__(f"Withdrawal of {amount} is not allowed. Balance: {balance}")

def withdraw(amount, balance):
    if amount > balance:
        raise WithdrawError(amount, balance)
    else:
        # Perform the withdrawal logic
        print(f"Withdrawal of {amount} successful. New balance: {balance - amount}")

# Example usage
try:
    withdraw(200, 150)
except WithdrawError as e:
    print(f"Error: {e}")
```

In this example, the `withdraw` function raises a `WithdrawError` if the withdrawal amount exceeds the account balance. The `try-except` block demonstrates how to catch and handle this specific custom exception.

Creating custom exception classes allows you to provide more context and specificity to errors in your code, making it easier to identify and handle different exceptional situations.