Exception Handling

Exceptions

- When an error occurs at runtime, an exception can be thrown.
- When an exception is thrown, Python terminates the program and prints information about the exception.
- Some exceptions occur due to programming errors, you should fix those errors.
- Some exception occurs due to causes outside of the program. Your program needs these exceptions to prevent your program crashes.

- Two ValueError exception
  - int(data) Can’t convert to an integer
  - float(data) Can’t convert to a float value

Using a try statement to handle exception

- In a try statement, you code any statements that may throw an exception in the try clause. Then, you can code an except clause that handles any exceptions that occur.
- If you don’t code the name of an exception type in the except clause, the except clause handles all types of exceptions that can occur.
- When an exception occurs, Python skips any remaining statements in the try clause and executes the statements in the except clause.

```
try:
    quantity = int(input("Enter quantity to buy: "))
    return quantity
except ValueError:
    print("Invalid integer. Please try again.")
```

Common Exceptions

- Common Exceptions:
  - Exception
    - OSError
      - FileExistsError
      - FileNotFoundError
    - ValueError
- Multiple except blocks

  try:
      statements
  except ExceptionName:
      statements
  [except ExceptionName:
      statements]..
- The except clauses must be coded in sequence starting with the most specific exception and ending with the least specific.

```
try:
        courses = []
        with open(self.__filename, newline="") as file:
                reader = csv.reader(file)
                for row in reader:
                        courses.append(row)
                return courses
except FileNotFoundError as e:
        print("Could not find " + FILENAME + " file.")
        print("Terminating program.")
        sys.exit()
except Exception as e:
        print(type(e), e)
        print("Terminating program.")
        sys.exit()
```

Get the exception information

- When an exception occurs, an exception object is created. You can use the as keyword in an except clause to provide a name for accessing that object.

  except [ExceptionName] [as name]:

      statements

- To cancel a program as part of your exception handling routine, you can use the exit() function of the sys module.
- type(object) - returns the class for the specified object.

<class 'FileNotFoundError'>
message = [Errno 2] No such file or directory: 'courses2.csv'
Terminating program.

```
try:
        courses = []
        with open(self.__filename, newline="") as file:
                reader = csv.reader(file)
                for row in reader:
                        courses.append(row)
                return courses
except FileNotFoundError as e:
        print("Could not find " + FILENAME + " file.")
        print("Terminating program.")
        sys.exit()
except Exception as e:
        print(type(e),  ‘message =‘, e)
        print("Terminating program.")
        sys.exit()
```

More Exception Handling

- Some objects such as a file object define standard clean-up actions. For them, you can use a with statement to automatically clean up the resources that they're using, even if an exception occurs during the execution of the with statement.
- For objects that don't define standard clean-up actions, you can use a finally clause to manually clean up the resources that the object is using.
- A finally clause is always executed, even if an exception occurs statement is executed in the try block.

```
try:
    statements
except ExceptionName:
    statements
[except ExceptionName:
    statements]..
[finally:
    statements]
```

Raising an exception

- To raise an exception, you can use a raise statement that creates an exception object.
- You can raise an exception for the Exception class or any class that's a child class of the Exception class.
- raise Exception("Error message")
- Example
  - raise ValueError("Invalid value")
  - You should raise a ValueError() when other code tries to set a property value to an invalid value.

```
@gpa.setter
def gpa(self, gpa):
    if gpa < 0.0 or gpa > 4.0:
        raise ValueError("GPA must be form 0.0 to 4.0")
    else:
        self.gpa = gpa
```