Objectives

- Reading numeric data from files
- Writing to files
- Exceptions
- Our own modules

Review

- How do we read from files?
- Why do we need to handle reading numerical data specially?
- How do we write to files?

Review: Handling Numeric Data

- We have been dealing with reading and writing *strings* so far
  - Read from a file: get a string
  - Write to file: use a string
- What do we need to do to **read numbers** from a file?
  - Cast as a numeric type, e.g., `int` or `float`
- How can we **write numbers** to a file?
  - Cast number as a `str` or use `format` method

Problem: Temperature Data

- **Given**: data file that contains the daily high temperatures for last year at one location
  - Data file contains one temperature per line
  - Example: `data/florida.dat`
- **Problem**: What is the average high temperature (to 2 decimal places) for the location?

Rule of Thumb: Always look at data file before processing it

`avgData.py`

Writing to a File

- Create a file object in **write** mode:
  - `myFile = open("myfile.txt", "w")`
- Example: create a file from user input
  - `file_write.py`

What happens if you execute the program again with different user input?

Problem: Cleaning Up Data

- **Given**: a file containing students' names and their class according to the Registrar
- **Problem**: This data file is a little ugly
  - For example, instead of Ugr:Sophomore, the file could just say "Sophomore"
- **Solution:**
  - Read through file "data/years.dat", clean up data
    - remove "Ugr:"
  - Write the cleaned up data to a new file called "data/roster.txt"
    - 1st iteration: name year
    - 2nd iteration: nice tables of data

`cleanRoster.py`

EXCEPTION HANDLING

Handling Exceptions

- Using try/except statements
- Syntax:

```
try:
       <body>
except [<errorType>] :
       <handler>
```

Optional: use this to handle specific error types appropriately

- Example:

```
try:
   age = eval(input("Enter your age: "))
   currentyear = int(input("Enter the current year: "))
except:
   print("ERROR: Your input was not in the correct form.")
   print("Enter integers for your age and the current year")
   sys.exit()
```

yearborn.py

Discussion: `sys.exit()`

- What is `sys.exit()`? Where does it come from?

Discussion: `sys.exit()`

- What is `sys.exit()`? Where does it come from?
  - `import sys`
    - Imports the `sys` module

```
exit(...)
    exit([status])

    Exit the interpreter by raising SystemExit(status).
    If the status is omitted or None, it defaults to zero (i.e., success).
    If the status is an integer, it will be used as the system exit status.
    If it is another kind of object, it will be printed and the system
    exit status will be one (i.e., failure).
```

Handling Exceptions

- Other types of exceptions
  - File exceptions:
    - File doesn't exist
    - Don't have permission to read/write file

`file_handle.py`

CREATING MODULES

Where are Functions Defined?

- Functions can go inside of program script
  - Defined before use/called (if no `main`() function)
  - Or, below the `main`() function (***preferred***)
- Functions can go inside a separate **module**

Creating Modules

- Modules group together related functions and constants
- Unlike functions, no special keyword to define a module
  - A module is named by its filename
- You've used modules in the past
  - graphics.py
  - game.py

Just a Python file!

Typical Use of Modules

- Put your reusable code in a module that can be shared with others
- Example: `game.py`
  - `rollDie(sides)`
  - `rollMultipleDice(numDice, sides)`
- Call `import game` in Python interpreter
  - What happened?

Creating Modules

- Then, to call **`rollDie`** function
  - `game.rollDie(6)`
- To access a defined constants
  - Example: `game.SIDES`

Creating Modules

- So that our program doesn't execute code automatically when it is imported in a program, at bottom, add

```
if __name__ == '__main__' :
    testRollDie()
    testRollMultipleDice()
```

Not important how this works; just know when to use

- Note the sub-directories now listed in the directory

Benefits of Defining Functions in Separate Module

- Reduces code in ***primary*** driver script
- Easier to reuse by importing from a module
- Maintains the "black box"
  - ***Abstraction***
- Isolates testing of function
- Write "test driver" scripts to test functions separately from use in script

Lab 8: Pair Programming

Findley	Jordan
Parker	Margaret
Ryan	Chas
Lizzie	Robert
Olivia	Anna
Lindsey	Ben
Alison	Rachel
Kalady	Mary-Frances
Joseph	Andrew
Harris	Davis
Chase	Ian

Jordan	Findley
Margaret	Parker
Chas	Ryan
Robert	Lizzie
Anna	Olivia
Ben	Lindsey
Rachel	Alison
Mary-Frances	Kalady
Andrew	Joseph
Davis	Harris
Ian	Chase

Same pairing in each table

Looking Ahead

- Lab 8 tomorrow – files!
- Broader Issue – Cryptocurrencies