Week 7: COMP-801 - Integrated Computing Practice

Agenda

- Exception handling
- Problem solving strategies
- Text files
- Due next week

Exception Handling

Why? To separate

- normal course of problem solving process
- from signaling potential run-time errors

How? What's the syntax of the exception handling construct

```
try:
    <try clause code block>
except <ErrorType>:
    <exception handler code block>
```

Example 1

Code block prone to run-time error

```
items = ['a', 'b']
third = items[idx]
```

`idx` may be 2 or other invalid value, so:

```
try:
    items = ['a', 'b']
    third = items[idx]
except IndexError e:
    print(e)
```

Example 2

Code block prone to run-time error

```
x = 5
y = x / some_num
```

`some_num` may be 0, so:

```
try:
    x = 5
    y = x / some_num
except ZeroDivisonError e:
    print(e)
```

Standard Exception Classes - 1

Language Exceptions

- `ImportError` - Raised when the imported module is not found.
- `IndentationError` - Raised when there is incorrect indentation.
- `IndexError` - Raised when the index of a sequence is out of range.
- `KeyError` - Raised when a key is not found in a dictionary.

Standard Exception Classes - 2

Language Exceptions

- `MemoryError` - Raised when an operation runs out of memory.
- `NameError` - Raised when a variable is not found in local or global scope.
- `SyntaxError` - Raised by parser when syntax error is encountered.
- `SystemError` - Raised when interpreter detects internal error.

Standard Exception Classes - 3

Language Exceptions

- `UnboundLocalError` - Raised when a reference is made to a local variable in a function or method, but no value has been bound to that variable.
- `ValueError` - Raised when a function gets an argument of correct type but improper value.
- `TypeError` - Raised when a function or operation is applied to an object of incorrect type.
- `RuntimeError` - Raised when an error does not fall under any other category.

Standard Exception Classes - 4

Math Exceptions

- `ArithmeticError` - All errors that occur for numeric calculation.
- `OverflowErrror` - Raised when a calculation exceeds max limit for a numeric type.
- `FloatingPointError` - Raised when a floating point operation fails.
- `ZeroDivisionError` - Raised when the second operand of division or modulo operation is zero.

Standard Exception Classes - 5

I/O Exceptions

- `FileNotFound` - Raised when a file or directory is requested but doesn't exist.
- `EOFError` - Raised when the `input()` function hits end-of-file condition.
- `IOError` - Raised when an input/output operation fails, e.g., `print()` or `open()`, or for OS-related errors.
- `PermissionError` - Raised when running an operation without the adequate access rights.

Standard Exception Classes - 6

Other Exceptions

- `AssertionError` - Raised when an `assert` statement fails.
- `AttributeError` - Raised when attribute assignment or reference fails.
- `TabError` - Raised when indentation consists of inconsistent tabs and spaces.
- `UnicodeError` - Raised when a Unicode-related encoding or decoding error occurs.

Problem Solving Strategies

Review Polya's Problem Solving Techniques.

Also, read How to Solve It.

Polya's Problem Solving Techniques

Step 1 : *Understand* the problem

- Read **problem statement** carefully.
- Come up examples of problem **input** and problem **output**

Step 2 : Devise a plan (*design*)

- Apply appropriate problem solving **patterns**
- Translate plan steps into **computational steps**

Polya's Problem Solving Techniques

Step 3 : Carry out the plan (*solve*)

- Implement (*write the code*)

Step 4 : Look back: check and interpret (*evaluate*)

- Test, debug, fix
- Reflect on the strategy
- What have you learned?
- Can the solution be used for other problems?

Problem Solving Strategy: Divide and Conquer

- Break down the problem into sub-problems
- Find solution to sub-problems
  - Apply known problem solving patterns
  - Take advantage of data structures' properties and behavior
- Combine the sub-problems' solutions into the final solution

Text Files

A text file is:

- a *sequence of lines of text* stored on a permanent medium

A line in a text file is:

- a *sequence of characters* up to and including `\n` (newline)

*** Note**: The newline character(s) are different on different operating systems.

File Locations

File location includes:

- Directory *path* and *file name* where the file resides in the file system

Examples:

- Relative path: 'labs/lab5/input.txt'
  - relative to the current directory, which is the directory that has `labs` sub-directory
- Absolute path: '/Users/mcs/comp801/lab7/input.txt'
  - full path starting with **root** directory (`/`)

File Objects

- Create a **file object** to either read from or write to a file.
- Syntax: call `open()` built-in function
  - `fin = open('input.txt', 'r') # for reading`
  - `fout = open('output.txt', 'w') # for writing`

File objects must be closed after use:

- `fin.close() # file read is complete`
- `fout.close() # file write is complete`

Open a Text File

```
fin = open('filename.txt', 'r')  # opens file for reading
```

- `fin` is a text file object
  - created with the `open()` function call
  - to read from 'filename.txt' text file
  - can be used for subsequent calls

Method to Read a Line From a Text File

```
some_line = fin.readline(n)
```

- returns a **string** representing the next line including `\n`
- `n` is optional: how many characters to read from a line

Other Methods to Read from a Text File

```
lines_lst = fin.readlines()
```

- returns a **list** of strings, each representing a line in the file

  ```
  all_content = fin.read(n)
  ```
- returns a **string** of the entire file
- `n` is optional: how many characters to read from the entire file

Closing Text Files

```
fin.close()
```

- SHOULD be called after reading is complete

`with` ... `as` ... **statement**

- See examples in sections 10.6 and 10.7 in text book

Use `zip` To Combine iterables

- `zip` is built-in into Python
- Takes 2 or more **iterables** and returns an **iterator**
- `zip` **iterator** allows us to get one tuple at a time
- Tuple elements correspond to elements in the **iterable** arguments that are at the same index

Read Before Next Week

- Read handouts
  - Python Coding Style Guide
  - Polya's Problem Solving Techniques