Lecture 15

Queries

1 Command line Arguments

We've seen lots of commands (ls, mkdir, cd, cp, cat, less, rm, mv) that take input via the command line; they don't make you run them and then wait for input. Let's see how to do that!

```
# First we import the 'sys' module.  It contains a couple things that
# we'll be using
import sys


# This is the argument vector (old term for list).  It is a list of
# all the arguments passed to your program on the command line.
print(sys.argv)

# NOTE: We notice that index 0 is the name of our program.  This will
# ALWAYS be the case.

# We can also print it's length.
print(len(sys.argv))

# We can loop over all the arguments
for arg in sys.argv:
    print(arg)
```

Great! Let's use our new found knowledge to change an earlier example.

2 I/O Example with Command Line Args and Error Handling

Let's write a piece of code that takes the name of a file as a command line argument. It will then print out the line along with its line number.

```
import sys

# NOTE: Want to be able to type:
#
# python3 files.py <file_to_read>

my_file = open(sys.argv[1], "r")

line_number = 1
for line in my_file:
    print("{:2d}: {:s}".format(line_number, line.rstrip()))
    line_number += 1

my_file.close()
```

This will work great… as long as the user gives me the name of a file (otherwise `IndexError`). And not just any file, it also has to exist (otherwise `FileNotFoundError`), and I have to have the permission to read it (otherwise `PermissionError`). So… let's try again and handle some errors!

```
import sys

# NOTE: Want to be able to type:
#
# python3 files.py <file_to_read>

try:
    my_file = open(sys.argv[1], "r")
except:
    print("Something terrible happened!")
    sys.exit(1)

line_number = 1
for line in my_file:
    print("{:2d}: {:s}".format(line_number, line.rstrip()))
    line_number += 1
my_file.close()
```

Now, if an exception happens, we handle it… but we have no idea *what* went wrong. We can tell Python to handle each type of exception differently!

```
import sys

# NOTE: Want to be able to type:
#
# python3 files.py <file_to_read>

try:
    my_file = open(sys.argv[1], "r")
except IndexError:
    print("Usage: python3 files.py <file_to_read>")
    sys.exit(1)
except FileNotFoundError:
    print("No such file or directory: '{}'".format(sys.argv[1]))
    sys.exit(1)
except PermissionError:
    print("Permission denied: '{}'".format(sys.argv[1]))
    sys.exit(1)
except:
    print("Some other exception happened")
    sys.exit(1)

line_number = 1
for line in my_file:
    print("{:2d}: {:s}".format(line_number, line.rstrip()))
    line_number += 1
my_file.close()
```

3 Finally

In addition to what we've already seen, there is another thing we can do with exception handling, the **finally** clause.

```
try:
    print("Try to do some dangerous stuff.\n")
except TypeError:
    print("How to handle a TypeError.\n")
except ValueError:
    print("How to handle a ValueError.\n")
finally:
    print("This will always happen!!!!!\n")
```

Code in the **finally** will always happen, whether or not an exception was raised. It will also run whether or not we handled the exception. (**raise** various errors to demonstrate.)

4 Sorting

If time, talk about insertion and/or selection sort. Ask the audience how to sort a list and go with it?

$$[25,\ 13,\ -100,\ -12,\ 60,\ 31,\ -76,\ 2]$$

4.1 Insertion Sort

- Start at index 1, swap with lower one until it gets to the correct location (insert it).
- Each iteration, the first i elements of the list are sorted, each subsequent element is then inserted.

Go through example above. Write out code?

```
def insertion_sort(lst):
    for index in range(1, len(lst)):
        insert(lst, index, lst[index])


def insert(lst, location, to_insert):
    while location > 0 and to_insert < lst[location - 1]:
        lst[location] = lst[location - 1]
        location -= 1

    lst[location] = to_insert
```

4.2 Selection Sort

- Find the minimum (select it), swap with index 0
- Repeat for spots $1,\ 2,\ \ldots,\ n-1$

Go through example above. Write out code?

```
def selection_sort(lst):
    for index in range(len(lst) - 1):
        swap(lst, index, index_of_smallest(lst, index))


def swap(lst, i, j):
    tmp = lst[i]
    lst[i] = lst[j]
    lst[j] = tmp


def index_of_smallest(lst, start):
    min_index = start

    for index in range(start + 1, len(lst)):
        if lst[index] < lst[min_index]:
            min_index = index

    return min_index
```