I/O,
Exception Handling

CPE101 Winter 2019
@ Cal Poly SLO
By
Toshi

Learning Objectives

1. Introduction to I/O
    a. Will have already used print.
    b. Talk briefly about input.
2. Introduction to basic file I/O
3. Command Line arguments
4. Introduce using exceptions (handling) with emphasis on when exceptions are appropriate and when they are not (i.e., not a substitute for conditionals)

Getting Input from Keyboard in Python

- Getting input from keyboard
  - input()

```
inputs = []

# get three inputs
for i in range(3):
    # read a line from prompt
    line = input("Please type something and
hit [ENTER]: ")
    inputs.append(line)

print "inputs are:", inputs
```

```
Please type something and hit [ENTER]:
abcd
Please type something and hit [ENTER]:
123
Please type something and hit [ENTER]:
!@#$
inputs are: ['abcd', '123', '!@#$']
```

File I/O in Python

- open(file_name, mode)
  - Takes two arguments
    - file_name: string of a file name including its path
    - mode: string specifying the mode of operation
      - ‘r’: read
      - ‘w’: write
        - Will create a new file if the file does not exist.
        - The content of the existing file will be lost.
      - ‘x’: create a file
        - Returns an error if the file exists.
      - ‘a’: append
        - Will create a file if the file does not exist.
  - Returns a file object
  - Raises IOError

Python File object

- Methods in file object
  - read([size])
    - Read at most size bytes from the file (less if the read hits EOF before obtaining size bytes).
    - If the size argument is negative or omitted, read all data until EOF is reached.
  - readline([size])
    - Read one entire line from the file. A trailing newline character is kept in the string.
  - readlines([sizehint])
    - Read until EOF using readline() and return a list containing the lines thus read.
  - write(str)
    - Write a string to the file. There is no return value.
  - writelines(list)
    - Write a list of strings to the file. There is no return value.
  - close()
    - Close the file. A closed file cannot be read or written anymore.
    - Do not forget to close the file!

More Examples of File I/O

```
f = open('txt/test.txt', 'w') # open or create test.txt in txt directory

# you can pass any python strings to write()
f.write("\\n inserts a new line char.\n")
f.write("\\t inserts a tab.\tSee?\n")
f.write("By the way, \ let you escape a character.\n")
f.write("Mustangs defeated Southern Utah by a final score of %d-%d.\n" % (38, 24))

# create a list of strings and pass it to writelines()
lines = []
lines.append("Yesterday\n")
lines.append("All my troubles seemed so far away\n")
lines.append("Now it looks as though they're here to stay\n")
lines.append("Oh, I believe in yesterday\n")
f.writelines(lines)

#do not foreget to close the file
f.close()
```

Command Line Arguments

You can have some arguments passed to your program in command line.

```
# my_program.py
# you need to import sys module
import sys

print "command line args passed are:"
for i,arg in enumerate(sys.argv):
    print "args[%s] = %s" % (i, arg)
    print type(arg)
```

```
python my_program.py 1 two 3
command line args passed are:
args[0] = my_program.py
<type 'str'>
args[1] = 1
<type 'str'>
args[2] = two
<type 'str'>
args[3] = 3
<type 'str'>
```

Exceptions

An exception is an error that happens during execution of a program. When that error occurs, Python generate an exception that can be handled, which avoids your program to crash.

- Why use exceptions?
  - Exceptions are convenient in many ways for handling errors and special conditions in a program. When you think that you have a code which can produce an error then you can use exception handling.
- Raising an exception
  - You can raise an exception in your own program by using the raise exception statement.
  - Raising an exception breaks current code execution and returns the exception back until it is handled.

Common Exception Errors in Python

- IOError
  - If a file can not be opened.
- ImportError
  - If importing a module failed.
- ValueError
  - Raised when an inappropriate value is received.
- KeyboardInterrupt
  - Raised when the user hits the interrupt key (normally Control-C or Delete)
- EOFError
  - Raised when one of the built-in functions (input() or raw_input()) hits an end-of-file condition (EOF) without reading any data

Catching Exceptions

- try
  - Enclose lines of code you think might raise exceptions within try block.
- except
  - Catch exceptions raised.
  - You can either catch a specific exception or catch all.
- finally
  - Do something finally in any case.

```
try:
    # do something
except IOError:
    # handle the I/O error
except:
    # catch all other errors here
finally:
    # do something finally in any cases.
```

Exception Handling Example

```
import sys
try:
    # get the file name from the command line arg
    file_name = sys.argv[1]
    # open a file and read
    f = open(file_name, 'r')
    # read lines from the file
    for line in f.readlines():
        # print each line
        print line
    #close the file
    f.close()
except IOError:
    # handle the error
    print "Failed to open the file %s !" % (file_name)
except:
    # catch all other errors here
    print "Other error occurred!"
```

```
>> python exception_example.py
Other error occurred!
>> python exception_example.py xxx
Failed to open the file xxx !
>> python exception_example.py
exception_example.py
import sys
try:
   ...
```

Exception Handling Example with finally

```
import sys
f = None
try:
    # get the file name from the command line arg
    file_name = sys.argv[1]
    # open a file and read
    f = open(file_name, 'r')
    # read lines from the file
    for line in f.readlines():
        # print each line
        print line
except IOError:
    # handle the error
    print "Failed to open the file %s !" % (file_name)
except:
    # catch all other errors here
    print "Other error occurred!"
```

```
finally:
    if f:
        #close the file
        f.close()
```

Printing Stack Trace

Import traceback module

```
import sys
import traceback

try:
    #do something

except:
   #print stack trace
    traceback.print_exc(file=sys.stdout)
   # raise a new error
   # format_exc returns a string
    raise RuntimeError(traceback.format_exc())
```

Raising Exceptions

- To raise an exception, use raise statement.

```
try:
    x = y / z
except ZeroDivisionError:
    raise ValueError(“The value of z must not be zero!”)

# raise your own custom Error
If grade == ‘F’:
    raise BadGradeError(“OMG! I’ve got an F!”)

class BadGradeError:
    def __init__(self, message):
        self.msg = message
```