INLS 560
Programming for Information Professionals

Exceptions

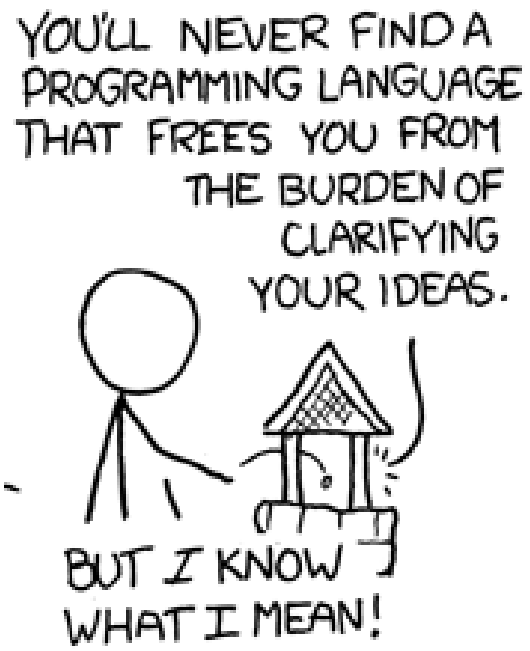
UNC
SCHOOL OF INFORMATION AND LIBRARY SCIENCE

Joan Boone
jpboone@email.unc.edu

Part 1: Types of exceptions

Part 2: Handling exceptions

Errors and Debugging



3 kinds of errors can occur in a program

- Syntax: Language syntax is incorrect, so the program won't run
- Runtime: Error occurs after the program starts running
- Semantic: Program runs successfully, but produces wrong results

Debugging

- Process of figuring out what went wrong and fixing it
- Can be both frustrating and interesting
- One of the best ways to learn programming

Source: http://xkcd.com/568/

Many Types of Exceptions

- Exceptions are errors that occur while a program is running, and will cause a program to stop where the error occurred.
- Exceptions are not user errors, such as entering invalid input; they are program errors that you, the developer, must fix

A few of the more common ones are

`SyntaxError:` invalid syntax found in program

`ValueError`: an operation or function receives an invalid value

`NameError`: when a local or global name is not defined

`TypeError:` an operation on an object is not appropriate

`OSError`: a general exception for input and output errors

`IndexError`: a subscript or index is out of range

`Exception`: general purpose, non-specific error

Complete list: Python Built-in Exceptions, Exception Class Hierarchy

SyntaxError Exception

- Occurs when the Python interpreter encounters a syntax error
- Python will display 'traceback' messages in the Run window that describe the type of error and where it occurred in your code
- These errors must be fixed; otherwise, your program will not function correctly

```
File "C:/Users/.../average_steps.py", line 5
    steps_file = open('steps.txt', 'r)
                                      ^
SyntaxError: unterminated string literal
             (detected at line 5)

Process finished with exit code 1
```

Line number and statement in error

Type of Python exception and a descriptive message

An exit code > 0 usually means the program terminated with an error

`ValueError` Exception

Raised when an operation, or function, receives an argument that has the right type, but an inappropriate value

```
Traceback (most recent call last):
  File "C:/Users/.../average_steps.py", line 24, in <module>
    main()
  File "C:/Users/.../average_steps.py", line 5, in main
    steps_file = open('steps.txt', 'z')
ValueError: invalid mode: 'z'

Process finished with exit code 1
```

NameError Exception

Raised when a local or global name is not found.

```
Traceback (most recent call last):
  File "C:/Users/.../average_steps.py", line 24, in <module>
    main()
  File "C:/Users/.../average_steps.py", line 18, in main
    print('The average number of steps taken in', line_count,
          'days was', format(avrage, ',.1f'))
NameError: name 'avrage' is not defined.
           Did you mean: 'average'?
```

`TypeError` Exception

Raised when an operation, or function, is applied to an object of inappropriate type.

```
Traceback (most recent call last):
  File "C:/Users/.../average_steps.py", line 24, in <module>
    main()
  File "C:/Users/.../average_steps.py", line 13, in main
    total_steps = total_steps + line
TypeError: unsupported operand type(s) for +: 'int' and 'str'


Process finished with exit code 1
```

`OSError` Exception

This exception is raised when a system function returns a system-related error, including I/O failures such as “file not found” or “disk full”

```
Traceback (most recent call last):
  File "C:/Users/.../average_steps.py", line 24, in <module>
    main()
  File "C:/Users/.../average_steps.py", line 5, in main
    steps_file = open('steps*txt', 'r')
OSError: [Errno 22] Invalid argument: 'steps*txt'


Process finished with exit code 1
```

FileNotFoundError Exception

```
def main():
    # Open the file.
    steps_file = open('step.txt', 'r')

    # Initialize counters
    total_steps = 0
    line_count = 0

    # Read each line in the file
    for line in steps_file:
        total_steps = total_steps + int(line)
        line_count = line_count + 1

    # Calculate the average and display
    average = total_steps / line_count
    print('The average number of steps taken in',
        line_count, 'days was', format(average, ',.1f'))

    # Close the file.
    steps_file.close()

main()
```

If you specify a file name that meets the file naming conventions, but the file does not exist, the following exception occurs:

```
FileNotFoundError: [Errno 2] No such file or directory: 'step.txt'
```

Part 1: Types of exceptions

Part 2: Handling exceptions

Handling Exceptions

- Python, like most programming languages, allows you to handle exceptions so that your program doesn't abruptly stop
- Define an *exception handler* with **try/except** statements
- General format:

```
try:
    statement1
    statement2
except ExceptionName:
    statement3
    statement4

statement5
statement6
```

Statements that can potentially raise an exception are placed inside the **try** clause.

Add statements to handle the exception inside the **except** clause.
For example, display an error message.

How it works:

- If the statements in the **try** clause do not raise an exception specified by ***ExceptionName***, then the statements in the **except** clause are skipped and execution resumes after the **except** clause
- If a statement in the **try** clause does raise an exception specified by ***ExceptionName***, then the statements in the **except** clause are executed
- If a statement in the **try** clause does raise an exception but it is not specified by ***ExceptionName***, then your program will stop with a traceback message

FileNotFoundError Exception Handler

```
def main():
    filename = 'step.txt'
    try:
        steps_file = open(filename, 'r')  # Open the file

        total_steps = 0       # Initialize counters
        line_count = 0

        for line in steps_file:     # Read each line in the file
            total_steps = total_steps + int(line)
            line_count = line_count + 1

        # Calculate the average and display
        average = total_steps / line_count
        print('The average number of steps taken in',
              line_count, 'days was', format(average, ',.1f'))

        steps_file.close()     # Close the file
    except FileNotFoundError:
        print('Error: cannot find file,', filename)

main()
```

Handling Multiple Exceptions

- A program can raise several types of exceptions -- it depends on what the program is doing.
- When reading a file, several types of exceptions could occur
  - File does not exist, or the file name is correct but may have been moved to another directory (**`FileNotFoundError`**)
  - Invalid file name, i.e., it does not meet platform naming conventions (**`OSError`**)
  - Invalid mode for opening the file (**`ValueError`**)
  - File may contain bad data, i.e., non-numeric data (**`ValueError`**)
- It is possible that an exception can be raised that you did not anticipate. To handle this case, always include a “catch all” exception that handles any exception not covered by other handlers.

```
except Exception:
    print('An unknown error occurred')
```

Handling Multiple Exceptions

- Order in which exception handlers are specified can be important
- If an exception occurs, Python will look for the first handler that can handle it

```
def main():
    filename = 'steps.txt'
    try:
        steps_file = open(filename, 'r')  # Open the file
        total_steps = 0       # Initialize counters
        line_count = 0

        for line in steps_file:     # Read each record (line) in the file
            total_steps = total_steps + int(line)
            line_count = line_count + 1

        # Calculate the average and display
        average = total_steps / line_count
        print('The average number of steps taken in',
              line_count, 'days was', format(average, ',.1f'))
        steps_file.close()     # Close the file
    except FileNotFoundError:
        print('Error: cannot find file,', filename)
    except OSError:
        print('Error: cannot access file,', filename)
    except ValueError:
        print('Error: invalid data found in file', filename)
    except Exception:  # catch all error handler
        print('An unknown error occurred')

main()
```

Displaying the Default Error Message for an Exception

Exceptions are objects, and each object usually has an attribute that contains a default error message.

- The message is the same as the one displayed at the end of a traceback when an exception has no handler
- When you write an **except** clause, you can optionally assign the exception object to a variable. Pass the variable to the **print** function and it will display the Exception's default message.

```
except ValueError as err:
    print(err) # ok, but a context-specific message is better
    # Like the message below...
    print('Error: invalid data found in file', filename)
```

A good place to use this approach is with the 'catch all' **except** clause, since the type of error is unknown

```
except Exception as err:
    print(err)
```

Exceptions with Default Messages

```
# Read a file with number of steps taken for each day of the year. Calculate average steps taken
def main():
    filename = 'steps.txt'
    try:
        steps_file = open(filename, 'r')  # Open the file

        total_steps = 0      # Initialize counters
        line_count = 0

        for line in steps_file:    # Read each record (line) in the file
            total_steps = total_steps + int(line)
            line_count = line_count + 1

        # Calculate the average and display
        average = total_steps / line_count
        print('The average number of steps taken in',
              line_count, 'days was', format(average, ',.1f'))
        steps_file.close()    # Close the file
    except FileNotFoundError as err:
        print('Error: cannot find file,', filename)
        print('Error:', err)
    except OSError as err:
        print('Error: cannot access file,', filename)
        print('Error:', err)
    except ValueError as err:
        print('Error: invalid data found in file', filename)
        print('Error:', err)
    except Exception as err:  # catch all error handler, if the above handlers do not apply
        print('An unknown error occurred')
        print('Error:', err)

main()
```

average_steps_exceptions.py

What if my program is not handling exceptions properly?

There are two ways an exception would not be handled

- No `except` clause specifying an exception of the right type
- Exception occurred outside of a `try` clause

In both cases, the exception will cause your program to stop.

To avoid this situation:

- Test your program!!! Try to 'break it' with faulty, or invalid, data. Add exception handlers for the errors that occur.
- Ensure that the statement causing an exception is inside of a `try` clause
- Always include the general purpose `except Exception` handler to catch any problems your testing may not have found.