Introduction to: Computers & Programming: Exception Handling

Adam Meyers

New York University

Summary

- What kind of error *raises an exception*?
- Preventing errors
- How to raise an exception on purpose
- How to catch an exception and what to do with one once you caught it

Errors that Raise Exceptions

- These are errors that cause the program to halt.
- Special 'error' messages are printed to the screen.
- Examples
  - TypeError: if a function or operator is called with the wrong type of argument
    - 'The book' * 'The book'
    - Len(5)
  - ValueError: similar to TypeError, except the argument is the correct type but inappropriate for another reason:
    - int('hello')
      - Argument cannot be converted to an integer
  - IOError: if a file or path doesn't exist
    - Instream = read('abc','r')
  - IndexError: An index refers to a nonexistant position in a sequence
    - 'hello'[5]

Preventing Errors

- while answer != 'yes' and answer != 'no':

  answer = input('Answer yes or no: ')
- The function isinstance(object, type)
  - Possible types: int,str,list,tuple,dict...
  - Can be used to prevent type errors
  - while (not isinstance(input_string,str)):

    input_string('Provide a valid string: ')
- The functions: os.path.isfile(*path*), os.path.isdir(*path*)
  - Can be used to prevent IOError
- And so on

Preventing Errors

- while answer != 'yes' and answer != 'no':

  answer = input('Answer yes or no: ')
- The function isinstance(object, type)
  - Possible types: int,str,list,tuple,dict...
  - Can be used to prevent type errors
  - while (not isinstance(input_string,str)):

    input_string('Provide a valid string: ')
- The functions: os.path.isfile(*path*), os.path.isdir(*path*)
  - Can be used to prevent IOError
- And so on

Raising Exceptions

- If you decide that a certain situation warrants an error message, you can put it in your code.
- The syntax is as follows:

```
def foul_language(string):
    if string in ['chicken','turkey','pheasant']:
        raise Exception('Foul Language')
```

- You can use a more specific type of exception as well:
  - IOError(string), TypeError(string), etc.

Catching Exceptions

- If you are aware of possible exceptions, you can:
  - Let your program crash OR
  - Design your code to elegantly handle each type of possible exception
    - *** Preferred if other people are going to use your program
- try & except
  - Put your code in a block under 'try:'
  - Put what to do for each exception in blocks of code under 'except:' statement.

For any Type of Exception

- def get_an_integer():

```
    while True:
        try:
            number = int(input('Pick an integer: '))
            return(number)
        except:
            print('That wasn\'t an integer!')
```

- This will continually ask the user for an integer until they put one in. Note that the *return* statement causes the function to end.

For Specific Types of Exceptions

```
def divide_10_by_an_integer ():
    while True:
        try:
            number = int(input('Pick an integer: '))
            output = 10/number
            return(output)
        except ValueError:
            print('That wasn\'t an integer!')
        except ZeroDivisionError:
            print('You can\'t divide by zero!')
        except:
            print('Something is wrong! Try again!')
```

Using Default Error Messages

```
def divide_10_by_an_integer ():
    while True:
        try:
            number = int(input('Pick an integer: '))
            output = 10/number
            return(output)
        except ValueError as err:  ### using default message
            print(err)
        except ZeroDivisionError:  ### using my message
            print('You can\'t divide by zero!')
        except:
            print('Something is wrong! Try again!')
```

Else: Executes if there is no Exception

```
def divide_10_by_an_integer ():
    while True:
        try:
            number = int(input('Pick an integer: '))
            output = 10/number
        except ValueError:
            print('That wasn\'t an integer!')
        except ZeroDivisionError:
            print('You can\'t divide by zero!')
        except:
            print('Something is wrong! Try again!')
        else:
            return(output) ## equivalent to putting the return statement last in the try block
```

'finally' statements: execute at the end no matter whay

```
def divide_10_by_an_integer ():
    while True:
        try:
            number = int(input('Pick an integer: '))
            output = 10/number
            return(output)
        except ValueError:
            print('That wasn\'t an integer!')
        except ZeroDivisionError:
            print('You can\'t divide by zero!')
        except:
            print('Something is wrong! Try again!')
        finally:
            print('''This program was sponsored by NYU\'s CS Division. It is being released 'as is' and NYU is not responsible for any bugs.''')
```

Summary

- Exception or Error Handling is a necessary part of writing code, particularly if it is going to be used by people other than yourself.
- Python's exception handling system is very similar syntactically to *if/elif/else* statements
- *try:* – used before main block of code
- *except* Exception: – like elif: statement conditioned on Exception (a particular type of exception)
- *except*: – all other types of exceptions
- raise Exception('ABC') – to raise exception of your own
  - If you "catch" an exception and don't raise one, the program will not halt
- else – at the end, if no exception is raised
- finally – at the end, whether an exception is raised or not.
- http://docs.python.org/release/3.0.1/c-api/exceptions.html

Homework 8– Part 1 – Due December 2, 2015

- Read Chapter 9
- Do Module 10, Quiz 10

Homework – Part 2 Due Dec 2, 2015
Question 1

- Write a function that solicits a yes or no answer from a user using the function Input.
- If the user inputs: 'Yes,' 'yes', 'Y' or 'y', the function should return ***True***
- If the user inputs: 'No','no', 'N', or 'n', the function should return False
- Otherwise, it should raise an exception. The error message can be anything that makes sense, e.g., 'Yes-No Error: only yes or no answers are permitted'

Question 2

- Write a function that prints out the ratio of registered democrat to republican voters for a particular voting district.
- For districts with all republican or all democrat voters, the function should print out “All Republicans” or “All Democrats” instead of a ratio
- Use simple (float) division to calculate the ratio, but use try, except and the ZeroDivisionError to catch the “All Democrat” cases.

Homework Part 3

- Described at end of Input_Output Lecture