Introduction to: Computers & Programming: Exception Handling

Adam Meyers
New York University
Summary

• An Error halts the program and prints out an Error Message, this is called:
  – “raising an exception”

• How & Why your program can overtly raise an exception

• How & Why you can prevent a program from crashing by
  – anticipating errors with an if statement or
  – “catching an exception”
  – What should you do with an exception 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 nonexistent 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
  – if (not isinstance(input_string, str)):
    print('The variable “input_string” is', input_string)
    print('This is not a string')
    input_string = 'invalid_string'
    – if type(input_string) != str:
      • Same as above: type(object) → returns the type of any object

• The functions: os.path.isfile(path), os.path.isdir(path)
  – Can be used to prevent IOError
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:

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

• Generating specific type of exception:

```python
if string in ['chicken','turkey','pheasant']:
    raise TypeError('Foul Language')
```

– IOError(string), ValueError(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 will use your program

• **try & except**
  – similar to if/elif/else
  – Put your code in a block under 'try:'
  – Put what to do for each exception in blocks of code under 'except:' statements.
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.

• In this case, try/except, is little like if/else
For Specific Types of Exceptions (similar to if/elif/else)

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_integer2 ():
    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!')
def divide_10_by_an_integer3 ():
    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

Else: Executes if there is no Exception
'finally' statements: execute at the end (unless a return statement precedes)

def divide_10_by_an_integer4():
    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](http://docs.python.org/release/3.0.1/c-api/exceptions.html)
Homework 8 – Due 25th Class

- [http://cs.nyu.edu/courses/spring17/CSCI-UA.0002-004/hw8.html](http://cs.nyu.edu/courses/spring17/CSCI-UA.0002-004/hw8.html)
- This HW has an Exception portion and an Input-Output (IO) portion.
- This talk is helpful for the former, but not the latter.