Exceptions

George Mason University
Today’s topics

• Review of Chapter 8: Exceptions
• Go over examples and questions
• exceptions in Python
Exceptions review

• What is an exception? Can you give an example?
• How is your example different than something that can be handled by an if-else?
• How do we catch exceptions? Do they have to be inside a try block?
• What happens if there are multiple catches? Nested try blocks and catches?
• What else can we do with exceptions?
Let’s go over the exercises
Exceptions in python

```python
print "start"
try:
    denom = 0
    print "attempt divide"
    print 5 / denom
    print "will never get here"
except ZeroDivisionError:
    print "Oops!"
print "finished catching exception!"
print "Force crash: " + str(5/0)
```

- **try** is a keyword that declares the following block of indented statements will be protected by the subsequent catch block
- **except** is a keyword that declares the catch block
- an exception not protected in some try block will always crash the program
Multiple exceptions

Recall, a `try` block can try to match multiple types of exceptions.

Exceptions are classes and form a hierarchy—ordering is important of `catch` blocks, like `elif`.
Important exception types in yellow

- **IOError** when dealing with files
  - file not found
- **NameError** when you use a variable before giving it a value
- **SyntaxError** when you’re probably missing a : or }
- **TypeError** when you call a function with the wrong type
  - int(“kitten”)
Exceptions as objects

The catch clauses are matching a type (or class)

- **as** is a keyword that allows you to label the actual exception object with a variable name
  - useful for printing out information about that particular exception object
Raising an exception

```python
print "start"
try:
    denom = 0
    print "attempt divide"
    raise ZeroDivisionError("Bieber was here")
    print "will never get here"
except Exception as e:
    print "Problem: " + str(e)
print "finished catching exception!"
```

- `raise` is a keyword that we call in conjunction with the exception’s constructor
- Can specify our own message
Sweeping up after exceptions

```python
def openThisManyBuffers(number):
    try:
        # start opening buffers
        # ... something goes wrong
        raise IOError("couldn't open 42,997")
    except ZeroDivisionError:
        print "divided by zero"
    finally:
        print "always close all buffers"
        print "will not get here with IOError"

#MAIN
openThisManyBuffers(50000)
print "can't close buffers here"
```

- **finally** is a keyword that allows us to execute code regardless if an exception was raised or not in a **try** block
  - for example, leaving too many buffers open will crash the program later
Questions?