Accelerating
Information Technology
Innovation

http://aiti.mit.edu

Kenya Summer 2011
Lecture 08 – Exceptions
Do any of these look familiar to you?

SyntaxError: ...

IndexError: ...

IOError: ...

ZeroDivisionError: ...

TypeError: ...

KeyError: ...

EOFError: ...

ValueError: ...

NameError: ...

AttributeError: ...
Exception Terminology

- **Exceptions** are events that can modify the flow or control through a program.

- **try/except**: catch and recover from the error raised by you or the Python interpreter

- **finally**: perform cleanup actions whether exceptions occur or not

- **raise**: trigger an exception manually in your code
def calculate_infinity():
    infinity = 3/0
    return infinity
**Exceptional Situations**

```python
def calculate_infinity():
    infinity = 3/0
    return infinity
```

OK, it says here I can recover by displaying an error message, then restarting from this line of code...

You can CATCH an exception.

Exception handling code.
Dealing with Problems

Two Ways:

**Look Before Leap**

Easier to Ask Forgiveness than Permission
Look Before You Leap

• Before we execute a statement, we check all aspects to make sure it executes correctly:
  – if it requires a string, check that
  – if it requires a dictionary key, check that

• Tends to make code messy. The heart of the code (what you want it to do) is hidden by all the checking.
Look Before You Leap

Example:

```python
#LBYL, test for the problematic conditions
if not isinstance(s, str) or not s.isdigit:
    return None
elif len(s) > 10: # too many digits to convert
    return None
else:
    return int(str)
```
Easier to Ask Forgiveness than Permission

- Run any statement you want, no checking required.

- However, be ready to “clean up any messes” by catching errors that occur.

- The **try suite code** reflects what you want to do, and the **except code** what you want to do on error. Cleaner separation!

- Python likes EAFP!
Easier to Ask Forgiveness than Permission

Example:

#EAFP, just do it, clean up messes with handlers
try:
    return int(str)
except (TypeError, ValueError, OverflowError):
    return None
Try, Except, Else and Finally

try:
  code to try

except pythonError1:
  exception code
except pythonError2:
  exception code
except:
  default except code

else:
  non exception case

finally:
  clean up code
Nesting Exception Handlers

Once the exception is caught, its life is over.
Nesting Exception Handlers

- But if the ‘finally’ block is present the code in the finally block will be executed, whether an exception gets thrown or not.
Exception Idioms

• All errors are exceptions, but not all exceptions are errors. It could be signals or warnings

```python
>>> while True:
    try:
        line=raw_input()
    except EOFError:
        break
    else:
        # process next line
```

• Functions signal conditions with `raise` (to distinguish success or failure)
Raising Exceptions

try:
    raise NameError('HiThere')
extype NameError:
    print 'An exception flew by!'
User Defined Exceptions

class MyError(Exception):

    def __init__(self, value):
        self.value = value

    def __str__(self):
        return repr(self.value)
User Defined Exceptions

try:
    raise MyError(2*2)
except MyError as e:
    print 'My exception occurred, value:', e.value

My exception occurred, value: 4
User Defined Exceptions

raise MyError('oops!')

Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  __main__.MyError: 'oops!'
Questions?