

# Exception Handling

---

Dan Fleck

Coming up: Long ago we wrote  
this

# Long ago we wrote this

---

```
def addGraphicalButton(parent):  
    # Create the PhotoImage widget  
    im = PhotoImage(file='cake.gif')  
    button2 = Button(root, text="Potato", image=im)  
    button2.image = im  
    button2.pack()
```

But... there was a problem. If the cake.gif file doesn't exist, we crash

# Long ago we wrote this

```
def addGraphicalButton(parent):  
    # Create the PhotoImage widget  
    im = PhotoImage(file='cake.gif')  
    button2 = Button(root, text="Potato", image=im)  
    button2.image = im  
    button2.pack()
```

Traceback (most recent call last):

File "images\_sample.py", line 20, in <module>

im = PhotoImage(file='cake.gif')

File "/Library/Frameworks/Python.framework/Versions/2.5/lib/python2.5/lib-tk/Tkinter.py", line 3270, in \_\_init\_\_

Image.\_\_init\_\_(self, 'photo', name, cnf, master, \*\*kw)

File "/Library/Frameworks/Python.framework/Versions/2.5/lib/python2.5/lib-tk/Tkinter.py", line 3226, in \_\_init\_\_

self.tk.call(('image', 'create', imgtype, name,) + options)

**\_tkinter.TclError: couldn't open "cake.gif": no such file or directory**



# Exceptions

---

Normally, we'll assume the cake file should exist. However, on a rare occasion it may not... this is an exceptional case. We think it will happen VERY rarely, but need to deal with it

Exceptions are the way to handle these types of problems.

# try / except

try:

    <body>

except <ErrorType>:

    <error handler>

Python attempts to execute the statements in the body. If no error occurs, control will continue after the try/except block

If an error does occur in the body, Python will look for an except block with the matching error-type. If found, the error handler in that block will execute

When an error occurs this means an exception was **'raised'**



## addGraphicalButton with exception handling

```
def addGraphicalButton(parent):  
    # Create the PhotoImage widget  
    try:  
        im = PhotoImage(file='cake.gif')  
        button2 = Button(root, text="Potato", image=im)  
        button2.image = im  
        button2.pack()  
    except TclError:  
        print 'There was an error. No file was found'
```

Better! The program just continues without the button if an exception is raised. But lets try to do more, lets solve the problem!

## addGraphicalButton with exception handling

```
def addGraphicalButton(parent):  
    # Create the PhotoImage widget  
    try:  
        im = PhotoImage(file='cake.gif')  
        button2 = Button(root, text="Potato", image=im)  
        button2.image = im  
        button2.pack()  
    except TclError:  
        print 'There was an error. No file was found... creating a standard button  
        instead'  
        button2 = Button(root, text="Potato")  
        button2.pack()
```

Even better! If the image isn't there, at least we still have a button, just not a graphical button... our program still works though!

Coming up: Get more information about the exception



# Get more information about the exception

```
def addGraphicalButton(parent):  
    # Create the PhotoImage widget  
    try:  
        im = PhotoImage(file='cake.gif')  
        button2 = Button(root, text="Potato", image=im)  
        button2.image = im  
        button2.pack()  
    except TclError, moreDetails:  
        print 'There was an error. No file was found... creating a standard \'  
            button instead. The error was:', moreDetails  
        button2 = Button(root, text="Potato")  
        button2.pack()
```

The variable here will hold the instance of the exception class used to represent the exception. This is useful to give people more information about what happened.

```
>>> There was an error. No file was found... creating a standard button instead.  
The error was: couldn't open "cake.gif": no such file or directory
```



# Lets try it

---

- Lets try writing a function to get a number again, but make sure we have a valid number

# Exceptions propagate

---

- When an exception is raised, it moves from function to function until someone handles it.
- If it is never handled, it is printed to the screen and the program terminates.



# Exceptions propagate

```
def f1():  
    print 'IN F1'  
    f2()
```

```
def f2():  
    print 'IN F2'  
    f3()
```

```
def f3():  
    print 'IN F3'  
    y = 10 / 0 ## Oops!
```

```
def main():  
    f1()
```

```
main()
```

IN F1

IN F2

IN F3

Traceback (most recent call last):

File "propagate.py", line 19, in <module>

main()

File "propagate.py", line 17, in main

f1()

File "propagate.py", line 5, in f1

f2()

File "propagate.py", line 9, in f2

f3()

File "propagate.py", line 13, in f3

y = 10 / 0

ZeroDivisionError: integer division or modulo by

z

Exception begins

Exception is not handled... so  
continues

# Exceptions propagate

```
def f1():  
    print 'IN F1'  
    f2()
```

IN F1

IN F2

IN F3

```
def f2():  
    print 'IN F2'  
    f3()
```

Traceback (most recent call last):

File "propagate.py", line 19, in <module>

main()

File "propagate.py", line 17, in main

f1()

File "propagate.py", line 5, in f1

f2()

File "propagate.py", line 9, in f2

f3()

File "propagate.py", line 13, in f3

y = 10 / 0

ZeroDivisionError: integer division or modulo by

```
def f3():  
    print 'IN F3'  
    y = 10 / 0 ## Oops!
```

```
def main():  
    f1()
```

```
main()
```

Exception is not handled... so  
continues



# Exceptions propagate

```
def f1():  
    print 'IN F1'  
    f2()  
  
def f2():  
    print 'IN F2'  
    f3()  
  
def f3():  
    print 'IN F3'  
    y = 10 / 0 ## Oops!
```

```
def main():  
    f1()
```

```
main()
```

IN F1

IN F2

IN F3

Traceback (most recent call last):

File "propagate.py", line 19, in <module>  
 main()

File "propagate.py", line 17, in main  
 f1()

File "propagate.py", line 5, in f1  
 f2()

File "propagate.py", line 9, in f2  
 f3()

File "propagate.py", line 13, in f3  
 y = 10 / 0

ZeroDivisionError: integer division or modulo by

Exception is not handled... so  
continues

# Exceptions propagate

```
def f1():  
    print 'IN F1'  
    f2()
```

IN F1

IN F2

IN F3

```
def f2():  
    print 'IN F2'  
    f3()
```

Traceback (most recent call last):

File "propagate.py", line 19, in <module>

main()

File "propagate.py", line 17, in main

```
def f3():  
    print 'IN F3'  
    y = 10 / 0 ## Oops!
```

↑ f1()

File "propagate.py", line 5, in f1

f2()

File "propagate.py", line 9, in f2

f3()

File "propagate.py", line 13, in f3

y = 10 / 0

```
def main():  
    f1()
```

```
main()
```

ZeroDivisionError: integer division or modulo by

**Exception is not handled... so  
continues**



# Exceptions propagate

```
def f1():  
    print 'IN F1'  
    f2()
```

```
def f2():  
    print 'IN F2'  
    f3()
```

```
def f3():  
    print 'IN F3'  
    y = 10 / 0 ## Oops!
```

```
def main():  
    f1()
```

```
main() ←
```

IN F1

IN F2

IN F3

Traceback (most recent call last):

File "propagate.py", line 19, in <module>

main()

File "propagate.py", line 17, in main

f1()

File "propagate.py", line 5, in f1

f2()

File "propagate.py", line 9, in f2

f3()

File "propagate.py", line 13, in f3

y = 10 / 0

ZeroDivisionError: integer division or modulo by

Oops..Exception is never  
handled... end program

# Exceptions propagate

```
def f1():  
    print 'IN F1'  
    f2()  
    print 'F1 NORMAL EXIT'
```

IN F1

IN F2

IN F3

```
def f2():  
    try:  
        print 'IN F2'  
        f3()  
        print 'F2 NORMAL EXIT'  
    except ZeroDivisionError, detail:  
        print 'There was a divide by zero error!',  
            detail
```

There was a divide by zero error! integer division  
or modulo by zero

F1 NORMAL EXIT

```
def f3():  
    print 'IN F3'  
    y = 10 / 0  
    print 'hello world'
```

Notice: F2 did not have a normal exit, why?

```
def main():  
    f1()  
main()
```



# Exceptions Propagate

So, if a function has an unhandled exception, the exception moves up the call stack until it is handled.

Call Stack:

main

F1

F2

then, how about here?  
here?

F3

Exception occurred, is it  
handled in here?

Yes, F2 handles the  
exception and all other  
code continues normally

Coming up: What prints?

# What prints?

```
def test():  
    print 'A',  
    try:  
        x = 10 / 0  
        print 'B',  
    except ZeroDivisionError:  
        print 'OOPS',  
    print 'C',
```

A OOPS C



# What prints?

```
def test():  
    print 'A',  
    try:  
        x = 10 / someVar  
        print 'B',  
    except ZeroDivisionError:  
        print 'OOPS',  
    print 'C',
```

A

Traceback (most recent call last):

File "test2.py", line 12, in

<module>

test()

File "test2.py", line 6, in test

x = 10 / someVar

NameError: global name

'someVar' is not defined

We handled Zero error... NOT

NameError!!

# Adding multiple handlers

```
def test():  
    print 'A',  
    try:  
        x = 10 / someVar  
        print 'B',  
    except ZeroDivisionError:  
        print 'OOPS',  
    except NameError:  
        print 'DOH'  
    print 'C',
```

A DOH C

We handled the NameError

You can have as many exception handlers as you want, but only the first one matching the error will ever execute.

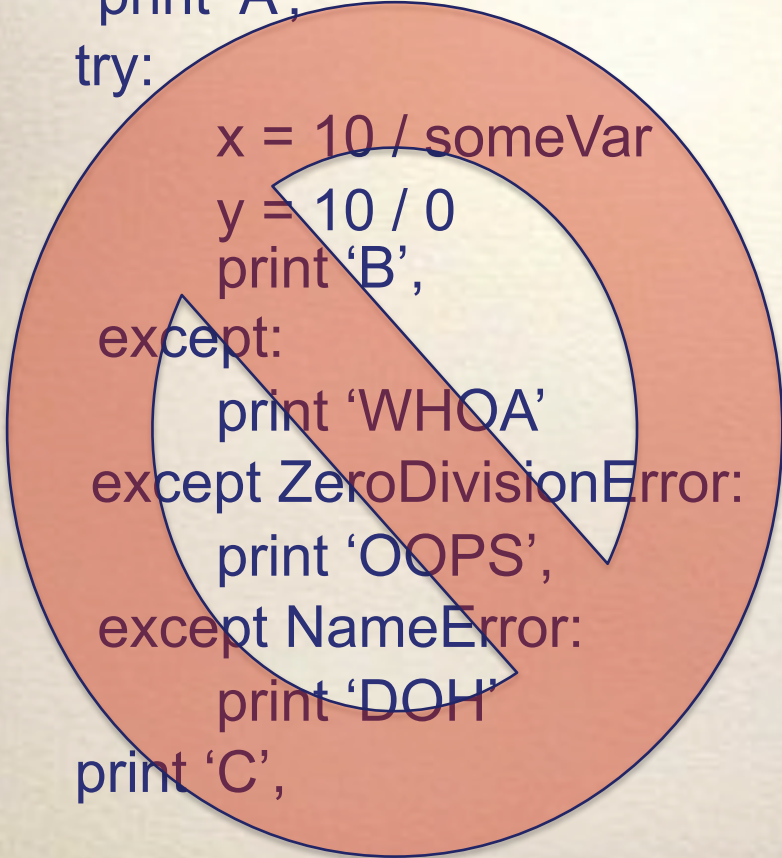
If you leave out the name, the except handler handles ALL exceptions

except: # Handle any exception



# Adding multiple handlers

```
def test():  
    print 'A',  
    try:  
        x = 10 / someVar  
        y = 10 / 0  
        print 'B',  
    except:  
        print 'WHOA'  
    except ZeroDivisionError:  
        print 'OOPS',  
    except NameError:  
        print 'DOH'  
    print 'C',
```



A WHOA C

This is BAD CODE!

except: matches ALL errors...  
so it should be a last resort. If  
used, put it at the end of the error  
handlers!

Most of the time you shouldn't  
use it at all... catch specific  
errors and handle them, let all  
other errors propagate up.

# Adding multiple handlers

```
def test():  
    print 'A',  
    try:  
        x = 10 / someVar  
        y = 10 / 0  
        print 'B',  
    except ZeroDivisionError:  
        print 'OOPS',  
    except NameError:  
        print 'DOH'  
    except:  
        print 'WHOA'  
    print 'C',
```

A DOH C

This is better. But in general you should avoid catching except: if you can. Better to catch specific errors and handle them appropriately.

If you need the safety net of catching except:, do it with caution.



# Summary

---

- Exceptions should be used for unexpected conditions in your program.
- When an exception happens we say the exception was raised
- Use try/except to handle them appropriately
- Exceptions propagate up the call stack until they are handled or the program ends
- You can have multiple exception handlers, and the first matching one is the only one that will execute.
- Without a name 'except:' will handle all exceptions and should be used carefully

# References

---

- <http://docs.python.org/tut/node10.html>
- This reference has much more information on the many other things you can do with exceptions like:
  - Creating your own
  - Raising an exception on purpose
  - Adding a 'finally' block
  - Adding an 'else' block