



# Testing Exceptions



Copyright © Software Carpentry 2010

This work is licensed under the Creative Commons Attribution License

See <http://software-carpentry.org/license.html> for more information.

## Option #1: return status code

## Option #1: return status code


```
params, status = read_params(param_file)
if status != OK:
    log.error('Failed to read', param_file)
    sys.exit(ERROR)
```

```
grid, status = read_grid(grid_file)
if status != OK:
    log.error('Failed to read', grid_file)
    sys.exit(ERROR)
```

## Option #1: return status code

```
params, status = read_params(param_file)
if status != OK:
    log.error('Failed to read', param_file)
    sys.exit(ERROR)

grid, status = read_grid(grid_file)
if status != OK:
    log.error('Failed to read', grid_file)
    sys.exit(ERROR)
```




*This is what we really care about*

## Option #1: return status code

```
params, status = read_params(param_file)
if status != OK:
    log.error('Failed to read', param_file)
    sys.exit(ERROR)
```

```
grid, status = read_grid(grid_file)
if status != OK:
    log.error('Failed to read', grid_file)
    sys.exit(ERROR)
```



*The rest is error handling*

## Option #2: use *exceptions*

Option #2: use *exceptions*

Separate "normal" operation from code that handles "exceptional" cases

Option #2: use *exceptions*

Separate "normal" operation from code that handles "exceptional" cases

Make both easier to understand



## Rearrange this...

```
params, status = read_params(param_file)
if status != OK:
    log.error('Failed to read', param_file)
    sys.exit(ERROR)
```

```
grid, status = read_grid(grid_file)
if status != OK:
    log.error('Failed to read', grid_file)
    sys.exit(ERROR)
```

...to put "normal" code in one place...

```

params, status = read_params(param_file)
if status != OK:
    log.error('Failed to read', param_file)
    sys.exit(ERROR)

grid, status = read_grid(grid_file)
if status != OK:
    log.error('Failed to read', grid_file)
    sys.exit(ERROR)

```

→ params = read\_params(param\_file)  
→ grid = read\_grid(grid\_file)

...and error handling code in another

```

params, status = read_params(param_file)
if status != OK:
    log.error('Failed to read', param_file)
    sys.exit(ERROR)

grid, status = read_grid(grid_file)
if status != OK:
    log.error('Failed to read', grid_file)
    sys.exit(ERROR)

```

params = read\_params(param\_file)  
grid = read\_grid(grid\_file)

log.error('Failed to read', filename)  
sys.exit(ERROR)

...and error handling code in another

```

params, status = read_params(param_file)
if status != OK:
    log.error('Failed to read', param_file)
    sys.exit(ERROR)

grid, status = read_grid(grid_file)
if status != OK:
    log.error('Failed to read', grid_file)
    sys.exit(ERROR)
    
```

Diagram illustrating code reuse for error handling:

- The first code block shows a function `read_params` that returns `params` and `status`. If `status` is not OK, it logs an error and exits.
- The second code block shows a function `read_grid` that returns `grid` and `status`. If `status` is not OK, it logs an error and exits.
- Red arrows indicate that the error handling code (the `if status != OK:` block) is shared between the two functions.

Only need one copy of the error handling code

## Join the two parts with try and except

```
params, status = read_params(param_file)
if status != OK:
    log.error('Failed to read', param_file)
    sys.exit(ERROR)
```

```
grid, status = read_grid(grid_file)
if status != OK:
    log.error('Failed to read', grid_file)
    sys.exit(ERROR)
```

**try:**

```
params = read_params(param_file)
grid = read_grid(grid_file)
```

**except:**

```
log.error('Failed to read',
          filename)
sys.exit(ERROR)
```

You have seen exceptions before

You have seen exceptions before

```
>>> open('nonexistent.txt', 'r')
```

***`IOError: No such file or directory: 'nonexistent.txt'`***

You have seen exceptions before

```
>>> open('nonexistent.txt', 'r')  
IOError: No such file or directory: 'nonexistent.txt'
```

```
>>> values = [0, 1, 2]  
>>> values[99]  
IndexError: list index out of range
```



# Use try and except to deal with them yourself

Use try and except to deal with them yourself

```
>>> try:
...     reader = open('nonexistent.txt', 'r')
... except IOError:
...     print 'Whoops!'
```

*Whoops!*

Use try and except to deal with them yourself

```
>>> try:
...     reader = open('nonexistent.txt', 'r')
... except IOError:
...     print 'Whoops!'
```

*Whoops!* ← *Blue indicates regular output*

Use try and except to deal with them yourself

```
>>> try:
...     reader = open('nonexistent.txt', 'r')
... except IOError:
...     print 'Whoops!'
```

*Whoops!*

*Try to do this...*

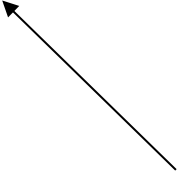


Use try and except to deal with them yourself

```
>>> try:
...     reader = open('nonexistent.txt', 'r')
... except IOError:
...     print 'Whoops!'
```

*Whoops!*

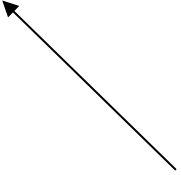
*...and do this if  
an IO error occurs*



Use try and except to deal with them yourself

```
>>> try:
...     reader = open('nonexistent.txt', 'r')
... except IOError:
...     print 'Whoops!'
```

*Whoops!*



*...and do this if  
an IO error occurs  
'IOError' is how Python  
reports 'file not found'*

Can put many lines of code in a try block

Can put many lines of code in a try block  
And handle several errors afterward



Can put many lines of code in a try block  
And handle several errors afterward

```
try:
    params = read_params(param_file)
    grid = read_grid(grid_file)
    entropy = lee_entropy(params, grid)
    write_entropy(entropy_file, entropy)
except IOError:
    print('IO error')
except ArithmeticError:
    print('Arithmetic error')
```

Can put many lines of code in a try block  
And handle several errors afterward

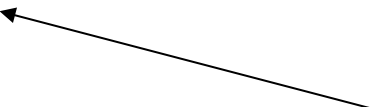
```
try:
    params = read_params(param_file)
    grid = read_grid(grid_file)
    entropy = lee_entropy(params, grid)
    write_entropy(entropy_file, entropy)
except IOError:
    print('IO error')
except ArithmeticError:
    print('Arithmetic error')
```

*Try to do this*

Can put many lines of code in a try block  
And handle several errors afterward

```
try:
    params = read_params(param_file)
    grid = read_grid(grid_file)
    entropy = lee_entropy(params, grid)
    write_entropy(entropy_file, entropy)
except IOError:
    print('IO error')
except ArithmeticError:
    print('Arithmetic error')
```

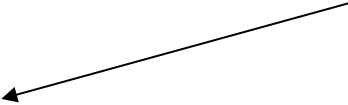
*Handle I/O  
errors here*



Can put many lines of code in a try block  
And handle several errors afterward

```
try:
    params = read_params(param_file)
    grid = read_grid(grid_file)
    entropy = lee_entropy(params, grid)
    write_entropy(entropy_file, entropy)
except IOError:
    print('IO error')
except ArithmeticError:
    print('Arithmetic error')
```

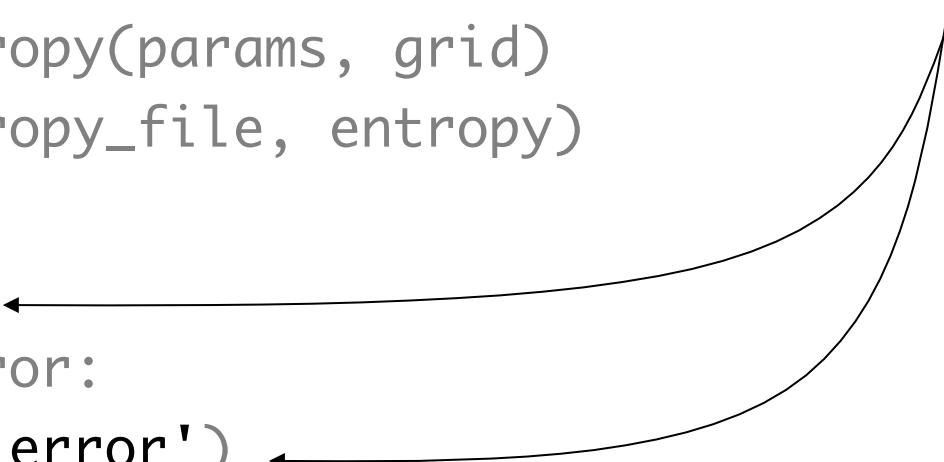
*and numerical  
errors here*



Can put many lines of code in a try block  
And handle several errors afterward

```
try:
    params = read_params(param_file)
    grid = read_grid(grid_file)
    entropy = lee_entropy(params, grid)
    write_entropy(entropy_file, entropy)
except IOError:
    print('IO error')
except ArithmeticError:
    print('Arithmetic error')
```

*These messages  
aren't very helpful*



```
try:  
    x = 1/0  
except Exception as error:  
    print error
```

*Stores information about what went wrong*

*Different information for different kinds of errors*

# Better error messages

## Better error messages

```
try:
    params = read_params(param_file)
    grid = read_grid(grid_file)
    entropy = lee_entropy(params, grid)
    write_entropy(entropy_file, entropy)
except IOError as err:
    print('Cannot read/write' + err.filename)
except ArithmeticError as err:
    print(err.message)
```




## Better error messages

```
try:
    params = read_params(param_file)
    grid = read_grid(grid_file)
    entropy = lee_entropy(params, grid)
    write_entropy(entropy_file, entropy)
except IOError as err:
    print('Cannot read/write' + err.filename)
except ArithmeticError as err:
    print(err.message)
```

## Better error messages

*Help user figure out  
which file*

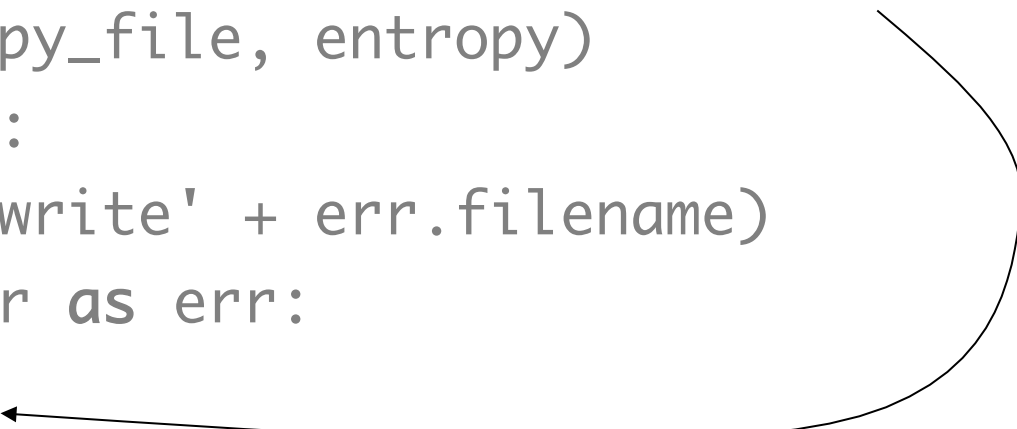
```
try:
    params = read_params(param_file)
    grid = read_grid(grid_file)
    entropy = lee_entropy(params, grid)
    write_entropy(entropy_file, entropy)
except IOError as err:
    print('Cannot read/write' + err.filename)
except ArithmeticError as err:
    print(err.message)
```



## Better error messages

```
try:
    params = read_params(param_file)
    grid = read_grid(grid_file)
    entropy = lee_entropy(params, grid)
    write_entropy(entropy_file, entropy)
except IOError as err:
    print('Cannot read/write' + err.filename)
except ArithmeticError as err:
    print(err.message)
```

*No worse than  
the default*



# A question of style

## A question of style

```
try:  
    grid = read_grid(grid_file)  
except IOError:  
    grid = default_grid()
```

## A question of style

```
try:  
    grid = read_grid(grid_file)  
except IOError:  
    grid = default_grid()
```

```
if file_exists(grid_file):  
    grid = read_grid(grid_file)  
else:  
    grid = default_grid()
```

## A question of style

```
try:
    grid = read_grid(grid_file)
except IOError:
    grid = default_grid()
```



```
if file_exists(grid_file):
    grid = read_grid(grid_file)
else:
    grid = default_grid()
```

*Use exceptions for exceptional cases*

## Another question of style



Another question of style

But first...

# Exceptions can be thrown a long way

## Exceptions can be thrown a long way

```
try:
    params = read_params(param_file)
    grid = read_grid(grid_file)
    entropy = lee_entropy(params, grid)
    write_entropy(entropy_file, entropy)
except IOError as err:
    print('Cannot read/write' + err.filename)
except ArithmeticError as err:
    print(err.message)
```

## Exceptions can be thrown a long way

```
try:
    params = read_params(param_file)
    grid = read_grid(grid_file)
    entropy = lee_entropy(params, grid)
    write_entropy(entropy_file, entropy)
except IOError as err:
    print('Cannot read/write' + err.filename)
except ArithmeticError as err:
    print(err.message)
```

*These are  
function calls*

## Exceptions can be thrown a long way

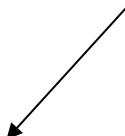
```
try:
    params = read_params(param_file)
    grid = read_grid(grid_file)
    entropy = lee_entropy(params, grid)
    write_entropy(entropy_file, entropy)
except IOError as err:
    print('Cannot read/write' + err.filename)
except ArithmeticError as err:
    print(err.message)
```

*Any errors  
they don't  
catch...*

## Exceptions can be thrown a long way

```
try:
    params = read_params(param_file)
    grid = read_grid(grid_file)
    entropy = lee_entropy(params, grid)
    write_entropy(entropy_file, entropy)
except IOError as err:
    print('Cannot read/write' + err.filename)
except ArithmeticError as err:
    print(err.message)
```

*...are caught  
and handled  
here*



# You can raise exceptions yourself

You ~~can~~ raise exceptions yourself  
should



~~You can~~ raise exceptions yourself  
should

```
def read_grid(grid_file):  
    '''Read grid, checking consistency.'''  
  
    data = read_raw_data(grid_file)  
    if not grid_consistent(data):  
        raise Exception('Inconsistent grid: ' + grid_file)  
    result = normalize_grid(data)  
  
    return result
```

~~You can~~ raise exceptions yourself  
should

```
def read_grid(grid_file):  
    '''Read grid, checking consistency.'''  
  
    data = read_raw_data(grid_file)  
    if not grid_consistent(data):  
        raise Exception('Inconsistent grid: ' + grid_file)  
    result = normalize_grid(data)  
  
    return result
```

You can define new types of exceptions too

You ~~can~~ define new types of exceptions too  
should

You ~~can~~ define new types of exceptions too  
should

Need to understand classes and objects

## Exercise 3:

#1 Using the function `dnaContent` or another of your own, raise Exceptions.

#2 Discuss that in group



Created by Greg Wilson (July 2010)  
Modified by Diego Barneche (Sept 2013)



Copyright © Software Carpentry 2010

This work is licensed under the Creative Commons Attribution License

See <http://software-carpentry.org/license.html> for more information.