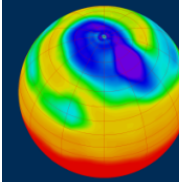




**National Centre for  
Atmospheric Science**  
NATURAL ENVIRONMENT RESEARCH COUNCIL



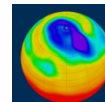
**Centre for Environmental  
Data Analysis**  
SCIENCE AND TECHNOLOGY FACILITIES COUNCIL  
NATURAL ENVIRONMENT RESEARCH COUNCIL

# Python Errors and Exceptions

CEDA



**National Centre for  
Atmospheric Science**  
NATURAL ENVIRONMENT RESEARCH COUNCIL



**Centre for Environmental  
Data Analysis**  
SCIENCE AND TECHNOLOGY FACILITIES COUNCIL  
NATURAL ENVIRONMENT RESEARCH COUNCIL

# Errors

Computer programmes break. It's a fact of life.

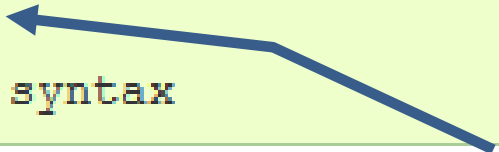
There are (at least) two distinguishable kinds of errors:

***syntax errors*** and ***exceptions***.

# Syntax Errors

- Syntax errors, or *parsing errors*, are very common when learning:

```
>>> while True print 'Hello world'
      File "<stdin>", line 1, in ?
        while True print 'Hello world'
                          ^
SyntaxError: invalid syntax
```



The arrow tells you  
where the error  
was located

# Exceptions

- Even if a statement or expression is syntactically correct, it may cause an error on execution.
- Errors detected during execution are called *exceptions* and are not unconditionally fatal.
- You can *catch* an exception and decide how to *handle* it.

```
>>> '2' + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: cannot concatenate 'str' and 'int' objects
```

# Types of exception

- Exceptions come in different types, and the type is printed as part of the message, e.g.:
  - [ZeroDivisionError](#)
  - [NameError](#)
  - [TypeError](#)
- And **you can define your own exceptions**, e.g.:
  - [MyAppBadUserInputError](#)

# Catching exceptions

- You can catch errors and decide how to handle them using: **try** / **except**

**try:**

```
result = runMyClimateModel(experiment)
```

**except:**

```
# It failed, so do something sensible  
emailMe("No results I'm afraid!")  
print "It's not a good model"
```

- By handling errors appropriately you can change the flow of your programme accordingly.

# Raising exceptions

- You can even trigger your own exceptions using: **raise**

```
if validate(input) == False:
    raise Exception("Bad input provided")
    # Programme will stop here unless this
    # exception is caught
else:
    print "Great input"

...processing input here...
```

# An example please

In this example, I have written some code to read the content from a number of simple text files. Each file should contain a numeric code.

There are two exceptions that I am interested in:

1. File does not have content
2. Contents of the file cannot be converted to an integer.



# An example continued

```
def readIntFromFile(fname):  
    "Returns an integer from a file. "  
    with open(fname) as f:  
        my_int = int(f.read(10))  
  
    return my_int
```

# An example continued

```
for f in ("a.txt", "b.txt", "c.txt"):
```

```
    try:
```

```
        print readIntFromFile(f)
```

```
    except IOError:
```

```
        print "There is nothing in file: {0}".format(f)
```

```
    except ValueError:
```

```
        print "Could not convert to int: {0} ".format(f)
```

```
    except Exception, err:
```

```
        print "Really unexpected error: {0}".format(err)
```

```
# But the script keeps processing all the good files!
```

# Further Reading

Much of this presentation was taken from the python documentation pages:

<https://docs.python.org/2/tutorial/errors.html>