

# INFO1110 & COMP9001: Introduction to Programming

School of Information Technologies, University of Sydney



## COMMONWEALTH OF AUSTRALIA

Copyright Regulations 1969

### **WARNING**

This material has been reproduced and communicated to you by or on behalf of the University of Sydney pursuant to Part VB of the Copyright Act 1968 (**the Act**).

The material in this communication may be subject to copyright under the Act. Any further copying or communication of this material by you may be the subject of copyright protection under the Act.

**Do not remove this notice.**

**We will cover:** Need for Exceptions, try/except/finally, passing the exception, binary and text files, writing to and reading from a file, streams

**You should read:** Python3 documentation: [Exceptions & Files](#)

## Lecture 11: Exceptions

*Your friend when things go badly*  
*rongQ#((Q#%\_)U\*((\_))..*

Whenever a Python program deals with the external environment, many things can go wrong

- unavailable resource
- not allowed to change
- wrong format when reading
- index out of bounds
- etc.

How will the program cope?

- have many different return values, indicating error conditions; then have the calling program test for each?

Exceptions are a mechanism to deal with such cases in a much cleaner way

# What is an exception?

*An **exception** is an object that signals the occurrence of an **unusual** event during the execution of a program.*

It's a kind of error message that is passed around around a program when something's gone unexpectedly wrong.

Passing an exception is called “throwing” it.

Under normal circumstances, exceptions are very rarely thrown: mostly, things work as expected.

# Exceptions in the real world

In your everyday life you may come across exceptions in a similar way to the way in which programs meet them.

Imagine you're a postal worker...

# Exceptions in the real world

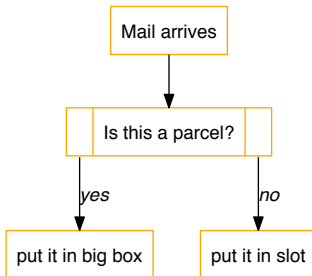
In your everyday life you may come across exceptions in a similar way to the way in which programs meet them.

Imagine you're a postal worker.....and your name is Alfred.

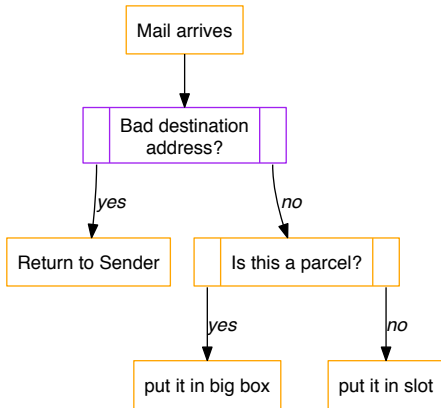


# Handling the mail

Mostly, Alfred's job was simple.

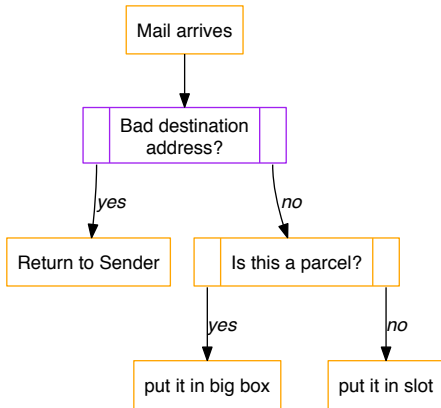


But sometimes, things were a bit more interesting.



Then, he had to think quite hard to know what to do, but he could handle it.

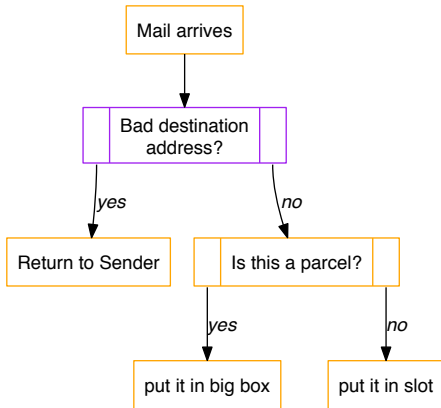
But sometimes, things were a bit more interesting.



Then, he had to think quite hard to know what to do, but he could handle it.

These circumstances were still “normal”. It wasn’t unexpected, and it was definitely something he could handle.

But sometimes, things were a bit more interesting.



Then, he had to think quite hard to know what to do, but he could handle it.

These circumstances were still “normal”. It wasn’t unexpected, and it was definitely something he could handle.

Exceptions are for when you *can't* handle it.

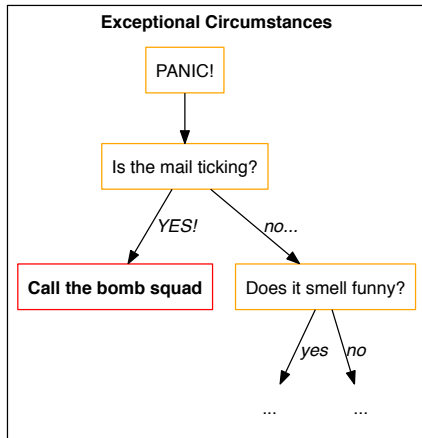
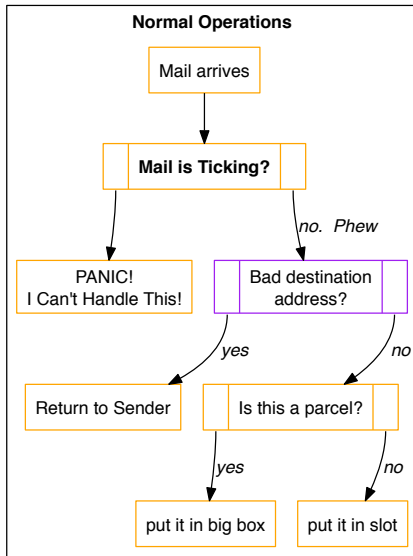
Until one day...

The mail.

Until one day...

The mail.

Is ticking.



Exception mechanism:

When an (exceptional) error is encountered

- 1 *Construct* an exception Object  
(which contains information about the error);
- 2 *Throw* (also called “raise”) the exception;  
This stops normal flow of control.
- 3 The exception may be *caught* somewhere and dealt with; otherwise the execution stops.



# Exceptions syntax

`try` a piece of code that contains a potential rare error

if the rare error occurs a new exception is generated. `raise`

The exception is caught using `except` and handled

Following the `try`, error or not, more code can be executed with a block `finally`.

Code that can't handle a strange error should throw an exception:

```
1 # never expect this situation
2 if numberOfGPU < 1 :
3     raise Exception("Cannot find any GPU. Cannot play.")
```

**Exception** is a class in Python. You need to make an Exception object

**Exception** is very general, it does not indicate what the specific problem is.

# Which Exception do I use?

**BaseException** is *any* kind of exception that can be raised during normal running of the Python program

**ArithmeticError** is a kind of exception that can be raised for  
FloatingPointError, OverflowError, ZeroDivisionError.

1 / 0 OR

1 // 0 OR

5 % 0

**SyntaxError** is a kind of exception that can be raised for IndentationError  
or TabError

x - y = 1

## Which Exception do I use? (cont.)

`LookupError` is a kind of exception that can be raised for `IndexError`,  
`KeyError`  
`x = 1, x[1]`

`AttributeError` `"some text".grasshopper()`

`TypeError` `x = 1, x[0]`

`NameError` `harrypotter('fire')`

`ValueError` `int("xyz")`

Check out the Exception Hierarchy for the language you are using. In Python, there are 64 Exceptions...No.

Throwing a particular kind of exception means that the kind of error that caused it can be used to solve the problem

When generating an Exception, be specific as possible  
e.g. `UnknownGameState(state_info)`

Recall the Exception is an object that can be passed around. The error handler is not always *near* the error itself.

# When to raise an exception

You might raise an exception when:

- an error or a situation occurs that makes the normal flow of processing unsuitable;
- 'someone else' should handle the error.

```
1 # ax^2 + bx + c = 0
2 # returns roots
3 def get_quadratic_roots(a, b, c):
4     result = [ None, None ]
5
6
7
8
9
10
11
12
13     return result
```


# Catching Exceptions

When code catches an exception,

- the error can be dealt with, and execution may continue, or
- it can clean up the damage and exit.

```
1 input = "10a"
2 try:
3     requests = int( input )
4     i = 0
5     while i < requests:
6         dostuff(i)
7         i = i + 1
8 except ValueError as ve:
9     print(str(type(ve)) + \
10         " exception caught: invalid number")
```

## Exception-handling code

- always begins with a `try` block, which should surround all the code you think should normally work – but which could raise an exception;
- must be followed by one or more `except` blocks *in increasing order of generality*: that is, from the most specific to the most general;
- may be followed by a `finally` clause, which contains code that will always be executed if it's there;
-  should not be used for control flow!



# Some exception code

Separate the “normal case” code (in the try block) from the code for weird situations ( the “except” block(s)).

You may have several *different* except clauses, depending on kind of exception thrown in the try block.

The optional finally block is always executed at the end of the except region:

- after the try block completes successfully,
- or after a except block is executed.

```
1  try:
2      # do something
3      # which could throw
4      # Exceptions
5  except EOFError as eoferr:
6      # do something to deal
7      # with the situation
8  except FileNotFoundError:
9      # do something else
10 except Exception:
11     # do something general
12 finally:
13     # code that should happen
14     # after normal/weird case
```

# Exception catching – example

```
1 try:
2     roots = get_quadratic_roots(a, b, c);
3     ...
4 except ArithmeticError as ae:
5     print("Oops: " + str(ae))
```

You can refer to the exception, which we've called `ae`, in the catch block.

If an exception occurs in a method

- constructed and thrown explicitly
- or raised by some other method that is called,

if it cannot be handled within the method then that method will *terminate* and throw the same exception to the caller of the method.

The exception will be passed up until someone can *handle* it.

# Passing exceptions to caller of function

Most programming languages declare what exceptions, if any, they will throw.

In python, there is no way of knowing this unless the caller of that method has that knowledge apriori

Other programming languages have more concrete ways of detecting unhandled exceptions at compile time (before executing the code)

This does not mean you should not, or cannot use exceptions. You just have to know about what function you are calling.

# Common Exceptional errors

- Putting `try/except` blocks with too small or too large scope: put them around the entire piece of code you expect to work!
- Returning too general an Exception type: use the appropriate exception if you can.
- Not conveying information with the exception thrown: construct it with a detailed message.



Do not *throw* an exception

- to avoid thinking about flow control in typical situations. e.g. don't deal with the end of the input this way!
- to “simplify” your code!



Do not *catch* an exception except

- if it is not the right place to handle it — maybe let the calling routine handle it;
- to simply prevent crashing at that moment

Use exceptions in exceptional cases.

- `try ... except ... finally ...`

- Exceptions are not suitable for expected situations flow-control!
- Catch and handle the exception if you can do so sensibly, else pass it on.
- Python language depends on use of exceptions by programmer. Know where and when they may occur

# Throwing an exception — example

```
1 def print_weight(weight):
2     """Prints a message about the weight"""
3     # expecting non-negative
4     if weight < 0.0:
5         raise ValueError("negative weight")
6         # this location can never be reached
7     # code here for the normal case
8     print("weight is: " + str(weight))
9
10 try:
11     print_weight(-2)
12     # more code if successful
13 except ValueError as ve:
14     print("Did not expect that value: {0}".format(ve))
```



# Exception Hierarchy

BaseException

- +-- SystemExit
- +-- KeyboardInterrupt
- +-- GeneratorExit
- +-- Exception
  - +-- StopIteration
  - +-- StopAsyncIteration
  - +-- ArithmeticError
    - | +-- FloatingPointError
    - | +-- OverflowError
    - | +-- ZeroDivisionError
  - +-- AssertionError
  - +-- AttributeError
  - +-- BufferError
  - +-- EOFError

## Exception Hierarchy (cont.)

```
+-- ImportError
    +-- ModuleNotFoundError
+-- LookupError
|   +-- IndexError
|   +-- KeyError
+-- MemoryError
+-- NameError
|   +-- UnboundLocalError
+-- OSError
|   +-- BlockingIOError
|   +-- ChildProcessError
|   +-- ConnectionError
|       +-- BrokenPipeError
|       +-- ConnectionAbortedError
|       +-- ConnectionRefusedError
|       +-- ConnectionResetError
```

## Exception Hierarchy (cont.)

```
|    +-- FileNotFoundError
|    +-- InterruptedError
|    +-- IsADirectoryError
|    +-- NotADirectoryError
|    +-- PermissionError
|    +-- ProcessLookupError
|    +-- TimeoutError
+-- ReferenceError
+-- RuntimeError
|    +-- NotImplementedError
|    +-- RecursionError
+-- SyntaxError
|    +-- IndentationError
|        +-- TabError
+-- SystemError
```

## Exception Hierarchy (cont.)

```
+-- TypeError
+-- ValueError
|   +-- UnicodeError
|       +-- UnicodeDecodeError
|       +-- UnicodeEncodeError
|       +-- UnicodeTranslateError
+-- Warning
    +-- DeprecationWarning
    +-- PendingDeprecationWarning
    +-- RuntimeWarning
    +-- SyntaxWarning
    +-- UserWarning
    +-- FutureWarning
    +-- ImportWarning
    +-- UnicodeWarning
    +-- BytesWarning
```

# Exception Hierarchy (cont.)

+-- ResourceWarning