Australian National University

COMP1730/COMP6730 Programming for Scientists

Exceptions and exception handling

Lecture outline

- **The exception mechanism in python**
- Raising exceptions (`assert` and `raise`)
- Catching exceptions

Reminder: Kinds of errors: Bugs.

- 1. Syntax errors or type errors: the code is not valid e.g. code is syntactically invalid, or is syntactically valid, but you're asking the python interpreter to do something impossible. E.g., apply operation to values of wrong type, call a function that is not defined, etc.
- 2. Semantic/logic errors: code runs without error, but does the wrong thing (for example, returns the wrong answer thus violating a post-condition, or a function is called with arguments violating a pre-condition/assumption).

Reminder: Kinds of errors: Run-time exceptional behaviour.

- 3. The following are exceptional behaviours that even fully correct code may encounter occasionally. Examples include: attempting to open a file that does not exist;
  or writing to a file and the hard disk fills up;
  or user input is of the wrong format;
  or a root-finding function cannot converge with the limits passed in;
  or a matrix inversion routine fails as the input matrix is close to singular (non-invertible)

Error handling in scientific or engineering code.

- Cases 1 and 2 above represent bugs in the code that should be corrected using a full suite of test functions and many assertions in the code to detect violations of preconditions, post-conditions and loop invariants. If such a bug is only detected when running a final experimental analysis then the correct response is to halt the code with an obvious and complete error message on the display or output log file.
- The correct python language construct to handle errors of type 2 is the `assertion`

Error handling.

* Note: for a large data analysis e.g. running on a large cluster you may consider checkpointing the state at regular intervals so you don’t need to restart from the beginning. Also, for certain engineering or commercial applications additional considerations on restarting the code automatically on detecting a bug may be important (e.g. on the Mars rover).
* Note: Case 1 cannot occur at run-time on a strictly typed compiled language.

Error handling.

- Case 3 is the only situation where we might want to detect the exceptional condition and call the affected code after correcting the problem.
- For example, if our search range does not enclose a root when calling a root-finding function, our calling function can probe a new part of the search space until all possible roots are found.

Error handling. Case 3 cont’

- When calling a matrix inversion or factorization routine, for a near-singular matrix we may call a more numerically stable but slower function, or may skip that data point.
- With bad user input, we may ask for the data to input again.

Error handling.

For Case 3 our options are to:

1. return some distinguished value to indicate the exceptional condition (e.g. return NaN for a root-finder). This may not be possible if all possible values may occur in the non-exceptional state.

2. return a tuple with an extra value that is true if the exceptional condition has occurred (may also return extra info about the particular issue).

3. Throw an exception.

Exception names

* Exceptions are types (classes) derived from the type `Exception`:
  - e.g. `ZeroDivisionError` derives from `ArithmenticError` derives from `Exception`.
  - ...and others.

* `https://docs.python.org/3/library/exceptions.html` for full list of exceptions in python standard library.
* Modules can define new exceptions by defining new classes derived from Exception.
* e.g. `LinAlgError` in the numpy library (to be covered next lecture).

Lecture outline

- The exception mechanism in python
- **Raising exceptions (`assert` and `raise`)**
- Catching exceptions

Australian National University

Assertions

- `assert` *`condition`*`, "fail message"`
  - Evaluate *`condition`* (to type `bool`)
  - If the value is not `True`, raise an `AssertionError` with the (optional) message.
- Assertions are used to check the programmer’s assumptions (including correct use of functions).
- Function’s docstring states assumptions; assertions can check them.
- Although assertions are implemented with exceptions in python, they should never be caught and retried.

The `raise` statement

* `raise` *`ExceptionName`*`(...)`
  - Raises the named exception.
  - Exception arguments (required or optional) depend on exception type.
* Can be used to raise any type of runtime error.
* Typically used with programmer-defined exception types.

```
if root_not_found:
    raise MyMathLibraryError('root is not in input range')
```

Reminder: Defensive programming

* It is better to “fail fast” (raise an exception) than to return a nonsense result that will silently affect or analysis.

Lecture outline

- The exception mechanism in python
- Raising exceptions (`assert` and `raise`)
- **Catching exceptions**

Exception handling

Catching exceptions allows us to retry or otherwise handle exceptional conditions:

```
try:
    block
except ExceptionName:
    error-handling block
```

- Execute *block*.
- If no exception arises, skip *error-handling block* and continue as normal.
- If the named exception arises from executing *block* immediately execute *error-handling block*, then continue as normal.
- If any other error occurs, fail as normal.

- An un-caught exeception in a function causes an immediate end to the execution of the function block; the exception passes to the function's caller, arising from the function call.
- The exception stops at the *first* matching `except` clause encountered in the call chain (this allows exceptions from multiple function calls to be handled in one location in higher level code).

```python
def f(x, y):
    try:
        return g(x, x + y)
    except ZeroDivisionError:
        return 0

def g(x, y):
        return x / y
```

When to catch exceptions?

- Never catch an exception unless there is a sensible way to handle it (there is no sensible way to handle syntax or type errors or logical bugs in the code).
- If a function does not raise an exception, it’s return value (or side effect) should be correct.
  - Therefore, if you can’t compute a correct value, raise an exception, or return an indicator of the exceptional condition.

Advantages of exceptions

- Exceptions are difficult to ignore if the exceptional condition occurs: if they are not caught they cause the code to abort (unlike returned error codes which are easy to fail to check for).
- Allow multiple error returns to be caught and handled in one place in higher level calling code (unlike error codes which must be explicitly checked for an propagated to higher calling functions).

Australian National University

Disadvantages of exceptions

* Exceptions are slow when thrown (in principle should have no or low cost when not thrown).
* They add a new invisible return path from functions (this should be documented in docstring) which may make the function more difficult to understand: in particular, for functions that throw exceptions from multiple return points in the function it can be difficult to test all the return possibilities.

Summary

- Consider:
  - What runtime exceptional conditions can occur in your code?
  - Which should be caught, and how should they be handled?
  - What assumptions should be checked?
- Use `assert` to check violated assumptions.
- Use `raise` to throw an exception (or return an error code) for a run-time exceptional condition that may be handled by the caller.
- Never catch an exception unless there is a sensible way to handle it.