Exceptions and Assertions

Chapter 9

What is an Exception

- Indication something “unexpected” has occurred while the program is running
  - A **semantic** rather than a **syntax errors**
  - Exceptions are common program constructs – that hopefully rarely occur
  - Exceptions are **`raise`**d or “thrown” when an error occurs
  - Exceptions are “caught” or “handled”
- We have already seen several exceptions
  - TypeError
  - ValueError
  - NameError

So what is an exception?

- Class that inherits from **Exception**
  - Includes a tuple of arguments
- Python includes many built-in exceptions
  - https://docs.python.org/3.7/library/exceptions.html#bltin-exceptions
- Can create user-defined exceptions that subclass from Exception
- One exception can inherit from another
  - ArithmeticError
    - FloatingPointError
    - OverFlowError
    - ZeroDivisionError

Handling exceptions

- First the **try** clause is executed
  - Statements between **try** and **except** keywords
- If no *exception* occurs, the **except** clause is skipped and the execution of the **try** statement is complete
- If an *exception* occurs the rest of the **try** clause is skipped. If the **exception type** matches an **except** statement then that clause is executed and control passes to the next statement after the **try**
- If the exception does not match any handler, execution is passed to the next outer try. If no try if found then execution stops with an *unhandled exception*
  - Notice that try/except blocks can be nested

Exception handling

- We have seen this form:
  - `except` *`exception_name`* `[as` *`messageName)`*`:`
- We can handle multiple exception types:
  - `except (`*`exception_name1, exception_name2, … exception_name[n]`*`):`
- We can handle any exception:
  - `except:`

`try … except … else`

- A try/except block can have an **else** clause
- Must follow the last **except** statement
- Executed if the **try** clause does not raise an exception
- Using an **else** helps isolate the code being tested by the **try**

“Finger Exercise”
(p. 169)

- Implement a function that meets the specification below. Use a try-except block.

```
def sum_digits(s):
    """Assumes s is a string
       Returns the sum of the decimal digits in
       s For example, if s is 'a2b3c' it
       returns 5"""
```

A generalized inputter

```
def read_val(val_type, request_msg, error_msg):
    while True:
        val = input(request_msg + ' ')
        try:
            return(val_type(val)) #convert str to val_type
        except ValueError:
            print(val, error_msg)
```

Polly Who?

- Our **`read_val`** function is **polymorphic** in that it handles arguments of many types
- Avoid writing `read_int`, `read_float`, `read_boolean`, `read_string` …

Finally, `finally`

- The **finally** clause is always executed whether an exception occurs or not
- If an exception was raised then it will be re-raised after the finally clause is executed

Controlling the flow

- We can also **raise** exceptions
  - `raise exception_name[(arguments)]`

- Can be used to raise an exception “in flight”
- Can be used to re-raise or reclassify a handled exception
  - Useful in logging
- Example (p. 106)

Assertions

- Tests whether some condition is True
  - If not raise an AssertionError

- These are equivalent

```
assert Boolean expression[, arguments]
```

```
if Boolean expression:
    raise AssertionError(arguments)
```

Asserts and testing

- Last week we had this code:

```
def test_abs(x, abs_x):
    if abs(x) == abs_x:
        print('Yippee Skippy abs(', x, ') == ', abs_x)
    else:
        print('Yikes abs(', x, ') != ', abs_x)

test_abs(2, 2)
test_abs(-2, 2)
test_abs(0, 0)
test_abs(-1, 1)
```

Silent testing

```
def test_abs(x, abs_x):
    assert abs(x) == abs_x, 'Yikes abs(` + 
str(x) + ') != ` + str(abs_x)


test_abs(2, 2)
test_abs(-2, 2)
test_abs(0, 0)
test_abs(-1, 1)
```

Design considerations

- What types of implicit checks do you want
  - When to evaluate
- What types of exceptions do you want to raise
- Clarity vs. Cleverness

`unittest`

- Create a test class that is a child of `unittest`
- Import class under test
- Create test cases
- Separates implementation and testing
- `unittest` provides several methods to support testing and several types of assert methods
  - `assertEquals, assertNotEquals`
  - `assertIn, assertNotIn`
- https://docs.python.org/3.7/library/unittest.html