INCREMENTAL DEVELOPMENT, DEBUGGING & TESTING

COMP130 – INTRODUCTION TO COMPUTING

DICKINSON COLLEGE

INCREMENTAL DEVELOPMENT

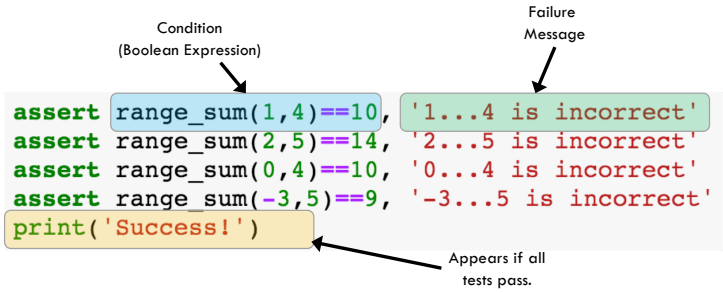
- In ***Incremental development*** only small changes or small amounts of code are *added to a working program* at a time and each change is tested when it is made.
  - Useful for building larger, more complex programs and functions.
  - Can prevent long tedious debugging sessions.
    - If the program was working before the small change, but not after, where is the problem likely to be?
  - Complementary to the *Development Process* defined earlier.
    - Sketch, encapsulate, generalize, repeat, refactor.
    - ***Particularly useful when you can readily identify functions.***

INCREMENTAL DEVELOPMENT OF FUNCTIONS

- **Sketch**: Sketch out the program and identify functions.
- **Stub**: Write a ***function stub*** (name, parameters, prints, return dummy value) and call it with ***dummy arguments.***
- **Expand**: Expand and hand test each small bit of functionality as it is added.
  - KISS: Keep It Simple Stupid
  - ***Debug,*** adding ***scaffolding*** (intermediate variables and prints) as necessary
  - Iterate
- **Test**: Produce ***automated tests*** for the function to facilitate refactoring.
- **Guard**: Add `assert` based ***guardians*** to ensure function can complete its task.
- **Improve**: Refactor the program and the functions.
  - Readability, Encapsulation, Generalization, Efficiency

AUTOMATED TESTS

- Python's `assert` statement is useful for creating ***automated test cases*** for functions.



PICKING TEST CASES

- When choosing test cases consider:
  - ***Normal case*** (middle values)
  - ***Edge (boundary) cases*** (lowest value, highest value)
  - ***Special cases*** (specific values)
  - ***Statement coverage*** (ensure that all statements are executed by at least one test)
    - E.g. all branches of an if/else or a chained conditional

GUARDIAN PATTERN

- A ***guardian*** is a sequence of assertions at the start of the function that terminate the program if the function's ***preconditions*** do not hold (i.e. it will not be able to complete its computation).
  - The conditions should be expressed in the doc string and implemented with `assert`.

```
def range_sum(low, high):
    """Compute the sum of all integers from low to high, inclusive.
    low and high must be integers and low < high
    """
    assert isinstance(low, int), 'low must be an int'
    assert isinstance(high, int), 'high must be an int'
    assert low<high, 'low must be less than high'

    sum = 0
```

Guardians