Assertions

- If we simply want to be sure that assumptions on state of computation are as expected, we can use an `assert` statement
- We can't control response, but will raise an `AssertionError` exception if this happens
- This is good defensive programming

Example

```
def avg(grades, weights):
    assert not len(grades) == 0, 'no grades data'
    newgr = [convertLetterGrade(elt) for elt in grades]
    return dotProduct(newgr, weights)/len(newgr)
```

This will raise an `AssertionError` if it is given an empty list for grades, but otherwise will run properly

- error will print out information `'no grades data'` as part of process

Assertions as defensive programming

- While assertions don't allow a programmer to control response to unexpected conditions, they are a great method for ensuring that execution halts whenever an expected condition is not met
- Typically used to check inputs to procedures, but can be used anywhere
- Can make it easier to locate a source of a bug

Extending use of assertions

- While pre-conditions on inputs are valuable to check, can also apply post-conditions on outputs before proceeding to next stage

Example, extended

```
def avg(grades, weights):
    assert not len(grades) == 0, 'no grades data'
    assert len(grades) == len(weights), 'wrong number grades'
    newgr = [convertLetterGrade(elt) for elt in grades]
    result = dotProduct(newgr, weights)/len(newgr)
    assert 0.0 <= result <= 100.0
    return result
```

Example, extended

- Slight loss of efficiency
- Defensive programming:
  - by checking pre- and post-conditions on inputs and output, avoid propagating bad values

Where to use assertions?

- Goal is to spot bugs early, and make clear where they happened
  - Easier to debug when catch at first point of contact, instead of trying to trace down later
- Not to be used in place of testing, but as a supplement to testing
- Should probably rely on raising exceptions if users supplies bad data input, and use assertions for:
  - Checking types of arguments or values
  - Checking that invariants on data structures are met
  - Checking constraints on return values
  - Checking for violations of constraints on procedure (e.g. no duplicates in a list)