One-on-One Sessions

- Starting tomorrow: 1/2-hour one-on-one sessions
  - Bring computer to work with instructor, TA or consultant
  - Hands on, dedicated help with Lab 2 and/or Lab 3
  - To prepare for assignment, not for help on assignment
- Limited availability: we cannot get to everyone
  - Students with experience or confidence should hold back
- Sign up online in CMS: first come, first served

- Hands on, dedicated help with Lab 2 and/or Lab 3
- To prepare for assignment, not for help on assignment

- Limited availability: we cannot get to everyone
  - Students with experience or confidence should hold back
- Sign up online in CMS: first come, first served
  - Choose assignment One-on-One

Recall: The Python API

- This is a specification
  - Enough info to use func.
  - But not how to implement
- Write them as docstrings

Anatomy of a Specification

def greet(n):
    """Prints a greeting to the name n"
    Greeting has format: 'Hello <n>!
    Followed by conversation starter.
    Parameter n: person to greet
    Precondition: n is a string""
    print 'Hello '+n+'!'
    print 'How are you?'

Anatomy of a Specification

def to_centigrade(x):
    """Returns x converted to centigrade"
    Value returned has type float.
    Parameter x: temp in fahrenheit
    Precondition: x is a float
    return 5*(x-32)/9.0

Preconditions

- Precondition is a promise
  - If precondition is true, the function works
  - If precondition is false, no guarantees at all
- Get software bugs when
  - Function precondition is not documented properly
  - Function is used in ways that violates precondition

Test Cases: Finding Errors

- Bug: Error in a program. (Always expect them!)
- Debugging: Process of finding bugs and removing them.
- Test case: A set of input values, together with the expected output.

Get in the habit of writing test cases for a function from the function’s specification—even before writing the function’s body.

```python
def number_vowels(w):
    """Returns: number of vowels in word w."
    Precondition: w string w/ at least one letter and only letters""
    pass # nothing here yet!
```
Representative Tests

- Cannot test all inputs
  - “Infinite” possibilities
- Limit ourselves to tests that are representative
  - Each test is a significantly different input
  - Every possible input is similar to one chosen
- An art, not a science
  - If easy, never have bugs
  - Learn with much practice

Representative Tests for number_vowels(w)

- Word with just one vowel
  - For each possible vowel!
- Word with multiple vowels
  - Of the same vowel
  - Of different vowels
- Word with only vowels
- Word with no vowels

Unit Test: A Special Kind of Module

- A unit test is a module that tests another module
  - It imports the other module (so it can access it)
  - It imports the cornelltest module (for testing)
  - It defines one or more test procedures
    - Evaluate the function(s) on the test cases
    - Compare the result to the expected value
  - It has special code that calls the test procedures

def assert_equals(expected, received):
  """Quit program if expected and received differ"""

Testing last_name_first(n)

```python
# test procedure
def test_last_name_first():
    """Test procedure for last_name_first(n)"""

    result = name.last_name_first('Walker White')
    cornelltest.assert_equals('White, Walker', result)

    result = name.last_name_first('Walker White')
    cornelltest.assert_equals('White, Walker', result)

# Execution of the testing code
test_last_name_first()

print 'Module name is working correctly'
```

Modules vs. Scripts

Module

- Provides functions, constants
  - Example: temperature.py
- import it into Python
  - In interactive shell...
  - or other module
- All code is either
  - In a function definition, or
  - A variable assignment

Script

- Behaves like an application
  - Example: helloApp.py
- Run it from command line
  - python helloApp.py
  - No interactive shell
  - import acts “weird”
  - Commands outside functions
  - Does each one in order

Finding the Error

- Unit tests cannot find the source of an error
- Idea: “Visualize” the program with print statements

```python
# Modules in this Course
# Our modules consist of
#  - Function definitions
#  - "Constants" (global vars)
#  - Optional script code to call/test the functions
# All statements must
#  - be inside of a function or
#  - assign a constant or
#  - be in the application code
# import will only use the definitions, not app code

def last_name_first(n):
    """Returns: copy of <n> in form <last>, <first>"

    end_first = n.find(' ')
    first = n[:end_first]
    last = n[end_first+1:]
    return last + ', ' + first
```

```python
# temperature.py
# ...
# def to_celsius(x):
#     """Returns: x converted to C"""
# ...
# FREEZING_C = 0.0 # temp. water freezes
# ...
# if __name__ == '__main__':
#     assert floats_equal(0.0, to_celsius(32.0))
#     assert floats_equal(100, to_celsius(212))
#     assert floats_equal(32.0, to_fahrenheit(0.0))
#     assert floats_equal(212.0, to_fahrenheit(100.0))
```

```python
def last_name_first(n):
    """Returns: copy of <n> in form <last>, <first>"

    end_first = n.find(' ')
    print end_first
    first = n[end_first+1:]
    print first
    last = n[:end_first]
    print last
    return last + ', ' + first
```

```python
def assert_equals(expected, received):
    """Quit program if expected and received differ"""
```

```python
# Execution of the testing code
# assert_equals(expected, received):
#     """Quit program if expected and received differ"""
```