We Write Programs to Do Things

- Functions are the **key doers**

**Function Call**
- Command to do the function
  ```python
greet('Walker')
  ```

**Function Definition**
- Defines what function does
  ```python
def greet(n):
    # docstring
    # print 'Hello ' + n + '!
    # print 'How are you?'
    print 'Hello ' + n + '!
    print 'How are you?'

    # function body
    # The vertical line indicates indentation
  ```

- **Parameter**: variable that is listed within the parentheses of a method header.
- **Argument**: a value to assign to the method parameter when it is called

Anatomy of a Function Definition

- **name**
- **parameters**
- **Function Header**
- **Function Body (indented)**

Procedures vs. Fruitful Functions

- **Procedures**
  - Functions that do something
  - Call them as a **statement**
  - Example: `greet('Walker')`

- **Fruitful Functions**
  - Functions that give a **value**
  - Call them in an **expression**
  - Example: `x = round(2.56, 1)`

Historical Aside

- Historically “function” = “fruitful function”
- But now we use “function” to refer to both

Print vs. Return

- **Print**
  - Displays a value on screen
  - Used primarily for **testing**
  - Not useful for calculations

  ```python
def print_plus(n):
    print(n+1)

>>> x = print_plus(2)
3
```  

- **Return**
  - Defines a function’s value
  - Important for calculations
  - But does not display anything

  ```python
def return_plus(n):
    return(n+1)

>>> x = return_plus(2)
>>> x
3
```  

The return Statement

- Fruitful functions require a **return statement**
  - **Format**: `return <expression>`
    - Provides value when call is used in an expression
    - Also stops executing the function!
    - Any statements after a `return` are ignored

  - **Example**: temperature converter function
    ```python
def to_centigrade(x):
    # docstring
    # returns: x converted to centigrade
    return 5*(x-32)/9.0
    ```

Recall: The Python API

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

- **Function name**
- **Number of arguments**
- **Return**
  - `ceil(x)`
    - Return the ceiling of `x` as a float, the smallest integer value greater than or equal to `x`.
  - `print_plus(n)`
    - `n` must be a **positive integer**
  - `sum(range(1000000))`
    - `sum(range(n))` returns the sum of all integers up to and including `n`.
  - `to_centigrade(x)`
    - `x` must be a **real number**

- `assert not x`
  - Test `x` is **not** `None`

- `assert x < 10`
  - Test `x` is **strictly less than** 10
Anatomy of a Specification

```python
def to_centigrade(x):
    """Returns x converted to centigrade."""
    return 5.0 * (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 violate 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.

Running Example

- The following function has a bug:

```python
def last_name_first(n):
    """Returns copy of <n> but in the form <last-name>, <first-name>"
    Precondition: <n> is in the form <first-name> <last-name> with one or more blanks between the two names"
    end_first = n.find(' ')
    first = n[:end_first]
    last = n[end_first + 1 :]
    return last, ' ' + first
```

Representative Tests:
- `last_name_first('Walker White')` gives `White, Walker`
- `last_name_first('Walker White')` gives `White, Walker`

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

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
  - The test procedures use the `cornelltest` function

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