Anatomy of a Function Definition

- **Name**: `greet(n)
- **Parameters**: `n`
- **Docstring**:
  "Prints a greeting to the name n
  Precondition: n is a string representing a person's name"
  
  `print 'Hello ' + n + '!
  print 'How are you?'`

- **Use vertical lines when you write Python on exams so we can see indentation**

The vertical line indicates indentation.

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(2.66,1)`

Historical Aside

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

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

```
def to_centigrade(x):
    """Returns: x converted to centigrade""
    return 5*(x-32)/9.0
```

Print vs. Return

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

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

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

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

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

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

Recall: The Python API

- **Function name**: `to_centigrade`
- **Number of arguments**: 1
- **What the function evaluates to**: `5*(x-32)/9.0`
- **Precondition**: `x` is a float measuring temperature in fahrenheit
- **Value returned has type float.**

Anatomy of a Specification

- **Function name**: `to_centigrade`
- **What the function evaluates to**: `5*(x-32)/9.0`
- **Precondition**: `x` is a float measuring temperature in fahrenheit

- **This is a specification**
  - Enough info to use func.
  - But not how to implement
  - Write them as docstrings
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.
- **Testing**: Process of analyzing, running program, looking for bugs.
- **Test case**: A set of input values, together with the expected output.

Representative Tests

- Cannot test all inputs
  - “Infinite” possibilities
- Limit ourselves to tests that are representative
  - Each test is 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

Running Example

- The following function has a bug:

```
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’

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

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

Testing last_name_first(n)

```
# test procedure
def test_last_name_first():
    """Test procedure for last_name_first()"
cornelltest.assert_equals( 'White, Walker',
            last_name_first( 'Walker White' ))
cornelltest.assert_equals( 'White, Walker',
            last_name_first( 'Walker White' ))
```

```
# Application code
if __name__ == '__main__':
    test_last_name_first()
```