Mini-Lecture 8

Specifications
Recall: The Python API

Function name
Possible arguments
What the function evaluates to
Recall: The Python API

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

```python
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
```

One line description, followed by blank line

More detail about the function. It may be many paragraphs.

Parameter description

Precondition specifies assumptions we make about the arguments
Anatomy of a Specification

```python
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
```

"""Returns"" indicates a fruitful function

More detail about the function. It may be many paragraphs.

Parameter description

Precondition specifies assumptions we make about the arguments
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

```python
>>> to_centigrade(32.0)
0.0
>>> to_centigrade(212)
100.0
```
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

>>> to_centigrade(32.0)
0.0

>>> to_centigrade(212)
100.0

>>> to_centigrade('32')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "temperature.py", line 19 ...
TypeError: unsupported operand type(s) for -: 'str' and 'int'

Precondition violated
def firstparens(text):
    """Returns: substring in ()
    Uses the first set of parens
    Param text: a string with ()"""
    # Find the open parenthesis

    # Store part AFTER paren

    # Find the close parenthesis

    # Return the result

>>> s = 'Prof (Walker) White'

>>> firstparens(s)
'Walker'

>>> t = '(A) B (C) D'

>>> firstparens(t)
'A'
# String Extraction Example

```python
def firstparens(text):
    
    """Returns: substring in ()
    Uses the first set of parens
    Param text: a string with ()""

    # Find the open parenthesis
    start = introcs.index_str(s,'(')

    # Store part AFTER paren
    # Find the close parenthesis

    # Return the result

# Testing

>>> s = 'Prof (Walker) White'
>>> firstparens(s)
'Walker'

>>> t = '(A) B (C) D'
>>> firstparens(t)
'A'
```
def firstparens(text):
    """Returns: substring in ()
    Uses the first set of parens
    Param text: a string with ()""

    # Find the open parenthesis
    start = intros.index_str(s,'(')
    # Store part AFTER paren
    tail = s[start+1:]
    # Find the close parenthesis
    # Return the result

    >>> s = 'Prof (Walker) White'
    >>> firstparens(s)
    'Walker'
    >>> t = '(A) B (C) D'
    >>> firstparens(t)
    'A'
def firstparens(text):

    """Returns: substring in ()
    Uses the first set of parens
    Param text: a string with ()""

    # Find the open parenthesis
    start = introcs.index_str(s,'(')
    # Store part AFTER paren
    tail = s[start+1:]
    # Find the close parenthesis
    end = introcs.index_str(tail,')')
    # Return the result

    return tail[end]

>>> s = 'Prof (Walker) White'

>>> firstparens(s)
'Walker'

>>> t = '(A) B (C) D'

>>> firstparens(t)
'A'
def firstparens(text):
    """Returns: substring in ()
    Uses the first set of parens
    Param text: a string with ()""
    # Find the open parenthesis
    start = introcs.index_str(s,'(')
    # Store part AFTER paren
    tail = s[start+1:]
    # Find the close parenthesis
    end = introcs.index_str(tail,')')
    # Return the result
    return tail[:end]

>>> s = 'Prof (Walker) White'
>>> firstparens(s)
'Walker'

>>> t = '(A) B (C) D'
>>> firstparens(t)
'A'
def second(thelist):
    """Returns: second elt in thelist
    Ex: second('A, B, C') => 'B'
    Param thelist: a list of words
    Precond: thelist has words sep.
    by commas, spaces.""

>>> second('cat, dog, mouse, lion')
'dog'

>>> second('apple, pear, banana')
'pear'
def second(thelist):
    """Returns: second elt in thelist
    Ex: second('A, B, C') => 'B'
    Param thelist: a list of words
    Precond: thelist has words sep. by commas, spaces."""

    # Find start of second elt
    # Find end of second elt
    # Slice from start to end
    # Return result

>>> second('cat, dog, mouse, lion')
'dog'

>>> second('apple, pear, banana')
'pear'
def second(thelist):
    """Returns: second elt in thelist
    Ex: second('A, B, C') => 'B'
    Param thelist: a list of words
    Precond: thelist has words sep. by commas, spaces."

    # Find FIRST comma
    # Find SECOND COMMA
    # Slice from comma to comma
    # Return result

    >>> second('cat, dog, mouse, lion')
    'dog'
    >>> second('apple, pear, banana')
    'pear'
def second(thelist):
    """Returns: second elt in thelist
Ex: second('A, B, C') => 'B'
Param thelist: a list of words
Precond: thelist has words sep. by commas, spaces."""

    s = introcs.index_str(thelist,',',)
    e = introcs.index_str(thelist,',',s+1)
    result = thelist[s+1:e]
    return result

>>> second('cat, dog, mouse, lion')
'dog'
>>> second('apple, pear, banana')
'pear'
String Extraction Example

```python
def second(thelist):
    """Returns: second elt in thelist
    Ex: second('A, B, C') => 'B'
    Param thelist: a list of words
    Precond: thelist has words sep. by commas, spaces.""
    s = introcs.index_str(thelist,',')
    e = introcs.index_str(thelist,',',s+1)
    result = thelist[s+1:e]
    return result

>>> second('cat, dog, mouse, lion')
'dog'
>>> second('apple, pear, banana')
'pear'
```

Where is the error?

A: Line 1
B: Line 2
C: Line 3
D: Line 4
E: There is no error
String Extraction Example

def second(thelist):
    """Returns: second elt in thelist
    Ex: second('A, B, C') => 'B'
    Param thelist: a list of words
    Precond: thelist has words sep. by commas, spaces.""
    
    s = introcs.index_str(thelist,',',')
e = introcs.index_str(thelist,',','.s+]')
    result = thelist[s+1:e]
    return result

>>> second('cat, dog, mouse, lion')
'dog'

>>> second('apple, pear, banana')
'pear'

result = thelist[s+2:end]
OR
result = introcs.strip(result)