We Write Programs to Do Things

- Functions are the **key doers**

Function Call

- Command to **do** the function

```
greet('Walker')
```

argument to assign to n

Function Definition

- Defines what function **does**

```
def greet(n):
    print 'Hello '+n+'!'
```

Function Header

declaration of parameter n

Function Body (indented)

- **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

```
def greet(n):
    """Prints a greeting to the name n

    Precondition: n is a string
    representing a person's name"""
    print 'Hello '+n+'!'
    print 'How are you?'
```

Function Header

Docstring Specification

Statements to execute when called

The vertical line indicates indentation

Use vertical lines when you write Python on **exams** so we can see 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(2.56,1)`

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
```


Aside: Constants

- Modules often have variables outside a function
  - We call these global variables
  - Accessible once you import the module
- Global variables should be **constants**
  - Variables that never, ever change
  - Mnemonic representation of important value
  - **Example**: math.pi, math.e in math
- In this class, constant names are **capitalized**!
  - So we can tell them apart from non-constants

Module Example: Temperature Converter

```
# temperature.py
"""Conversion functions between fahrenheit and centrigrade"""

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

def to_fahrenheit(x):
    """Returns: x converted to fahrenheit"""
    return 9*x/5.0+32

# Constants
FREEZING_C = 0.0  # temp. water freezes
```

Style Guideline: Two blank lines between function definitions

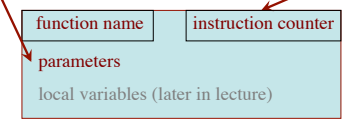
How Do Functions Work?

Draw template on a piece of paper

- **Function Frame**: Representation of a function call
- A **conceptual model** of Python

Draw parameters as variables (named boxes)

- Number of statement in the function body to execute next
- **Starts with 1**

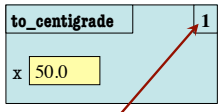


Example: to_centigrade(50.0)

1. Draw a frame for the call
2. Assign the argument value to the parameter (in frame)
3. Execute the function body
   - Look for variables in the frame
   - If not there, look for global variables with that name
4. Erase the frame for the call

```
1 def to_centigrade(x):
      return 5*(x-32)/9.0
```

Initial call frame (before exec body)



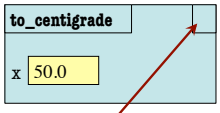
next line to execute

Example: to_centigrade(50.0)

1. Draw a frame for the call
2. Assign the argument value to the parameter (in frame)
3. Execute the function body
   - Look for variables in the frame
   - If not there, look for global variables with that name
4. Erase the frame for the call

```
1 def to_centigrade(x):
      return 5*(x-32)/9.0
```

Executing the return statement



The return terminates; no next line to execute

Example: to_centigrade(50.0)

1. Draw a frame for the call
2. Assign the argument value to the parameter (in frame)
3. Execute the function body
   - Look for variables in the frame
   - If not there, look for global variables with that name
4. Erase the frame for the call

```
1 def to_centigrade(x):
      return 5*(x-32)/9.0
```

ERASE WHOLE FRAME

But don't actually erase on an exam

Call Frames vs. Global Variables

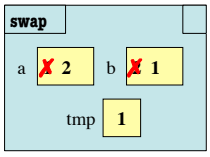
- This does not work:

```
def swap(a,b):
    """Swap vars a & b"""
1   tmp = a
2   a = b
3   b = tmp
```

```
>>> a = 1
>>> b = 2
>>> swap(a,b)
```

Global Variables

Call Frame



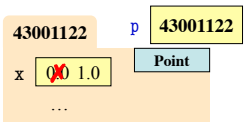
Example with Objects

- Mutable objects can be altered in a function call
  - Object vars hold names!
  - Folder accessed by both global var & parameter
- **Example**:

```
def incr_x(q):
1   q.x = q.x + 1
```

```
>>> p = Point()
>>> incr_x(p)
```

Global Variable



Call Frame