

Recall: Modules

- Modules provide extra functions, variables
 - **Example:** math provides math.cos(), math.pi
 - Access them with the import command
- Python provides a lot of them for us
- **This Lecture:** How to make modules
 - Pulsar to *make* a module
 - Python to *use* the module

Two different programs

1

We Write Programs to Do Things

- Functions are the **key doers**

Function Call

- Command to **do** the function

```
>>> plus(23)
24
>>>
```

Function Header

Function Definition

- Defines what function **does**

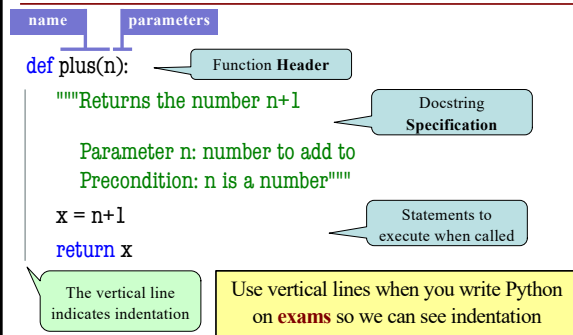
```
def plus(n):
    return n+1
```

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

2

Anatomy of a Function Definition



3

The return Statement

- **Format:** `return <expression>`
 - Used to evaluate *function call* (as an expression)
 - Also stops executing the function!
 - Any statements after a **return** are ignored
- **Example:** temperature converter function

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

4

A More Complex Example

Function Definition

```
def foo(a,b):
    """Return something
    Param a: number
    Param b: number"""
    x = a
    y = b
    return x*y+y
```

Function Call

```
>>> x = 2
>>> foo(3,4)
x ?
```

What is in the box?

```
A: 2
B: 3
C: 16
D: Nothing!
E: I do not know
```

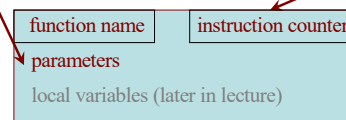
5

Understanding How Functions Work

- **Function Frame:** Representation of 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



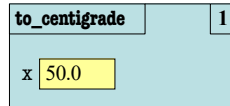
6

Text (Section 3.10) vs. Class

Textbook

`to_centigrade` `x -> 50.0`

This Class



Definition:

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

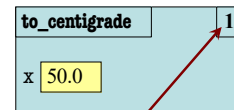
Call: `to_centigrade(50.0)`

7

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

Initial call frame
(before exec body)



next line to execute

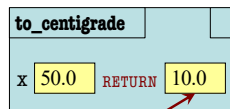
```
def to_centigrade(x):
    return 5*(x-32)/9.0
```

8

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

Executing the
return statement



Return statement creates a
special variable for result

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

9

Call Frames vs. Global Variables

The specification is a **lie**:

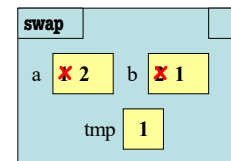
```
def swap(a,b):
    """Swap global a & b"""
    tmp = a
    a = b
    b = tmp
```

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

Global Variables

a 1 b 2

Call Frame



10

Exercise Time

Function Definition

```
def foo(a,b):
    """Return something
    Param x: a number
    Param y: a number"""
    1 x = a
    2 y = b
    3 return x*y+y
```

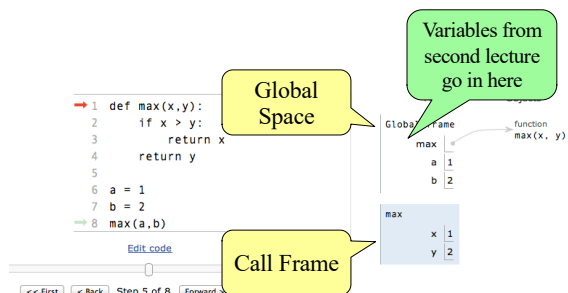
Function Call

```
>>> x = foo(3,4)
```

What does the
frame look like
at the **start**?

11

Visualizing Frames: The Python Tutor



12