We Write Programs to Do Things

- Functions are the **key doers**

Function Call

- Command to **do** the function

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

Function Definition

- Defines what function **does**

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

Function **Header**

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 plus(n):
|   """Returns the number n+1
|
|     Parameter n: number to add to
|     Precondition: n is a number"""
|   x = n+1
|   return x
```

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

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

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)
```

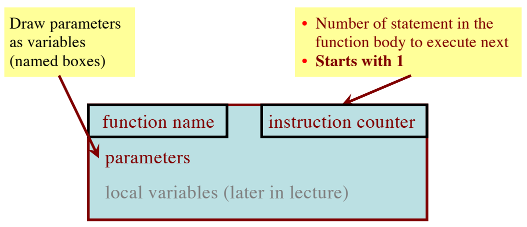
x ?

What is in the box?

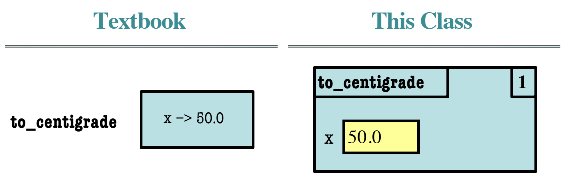
A: 2
B: 3
C: 16
D: Nothing!
E: I do not know

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
function name
instruction counter
parameters
local variables (later in lecture)

Text (Section 3.10) vs. Class



Definition:

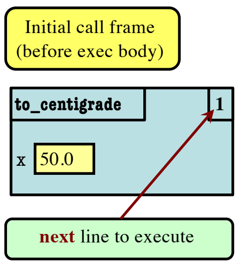
```
def to_centigrade(x):
|   return 5*(x-32)/9.0
```

Call: to_centigrade(50.0)

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

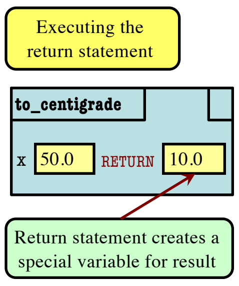
```
  def to_centigrade(x):
1 |  return 5*(x-32)/9.0
```



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

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

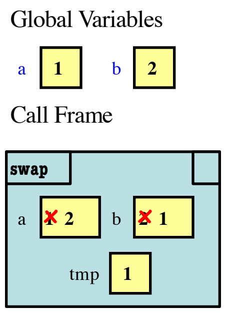

Executing the return statement
to_centigrade
x
50.0
RETURN
10.0
Return statement creates a special variable for result

Call Frames vs. Global Variables

The specification is a **lie**:

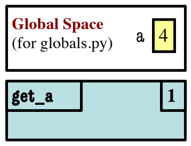
```
  def swap(a,b):
    """Swap global a & b"""
1 |  tmp = a
2 |  a = b
3 |  b = tmp
```

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



Function Access to Global Space

- All function definitions are in some module
- Call can access global space for **that module**
  - math.cos: global for math
  - temperature.to_centigrade uses global for temperature
- But **cannot** change values
  - Assignment to a global makes a new local variable!
  - Why we limit to constants

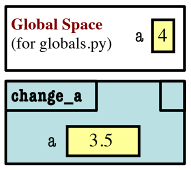


```
# globals.py
"""Show how globals work"""
a = 4 # global space

def get_a():
  return a # returns global
```

Function Access to Global Space

- All function definitions are in some module
- Call can access global space for **that module**
  - math.cos: global for math
  - temperature.to_centigrade uses global for temperature
- But **cannot** change values
  - Assignment to a global makes a new local variable!
  - Why we limit to constants



```
# globals.py
"""Show how globals work"""
a = 4 # global space
def change_a():
  a = 3.5 # local variable
  return a
```

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**?