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
  ▪ Atom Editor to *make* a module
  ▪ Python to *use* the module

Two different programs
We Write Programs to Do Things

• Functions are the **key doers**

<table>
<thead>
<tr>
<th>Function Call</th>
<th>Function Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Command to <strong>do</strong> the function</td>
<td>• Defines what function <strong>does</strong></td>
</tr>
</tbody>
</table>

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

• **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

9/10/18
Functions are the **key doers**

**Function Call**
- Command to **do** the function

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

**Function Definition**
- Defines what function **does**

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

- **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
We Write Programs to Do Things

- Functions are the **key doers**

**Function Call**
- Command to **do** the function

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

**Function Definition**
- Defines what function **does**

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

- **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
We Write Programs to Do Things

• Functions are the **key doers**

**Function Call**

• Command to **do** the function

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

**Function Definition**

• Defines what function **does**

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

- **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

def plus(n):
    
    """Returns the number n+1
    Parameter n: number to add to
    Precondition: n is a number"
    
    x = n+1
    return x
Anatomy of a Function Definition

\[ \text{def plus(n):} \]

"""Returns the number n+1
Parameter n: number to add to
Precondition: n is a number"""

\[ x = n+1 \]

\[ \text{return } x \]

The vertical line indicates indentation

Use vertical lines when you write Python on \textbf{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

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

9/10/18

Defining Functions
Defining a String Function

- Start w/ string variable
  - Holds string to work on
  - Make it the parameter
- Body is all assignments
  - Make variables as needed
  - But last line is a return
- Try to work in reverse
  - Start with the return
  - Figure ops you need
  - Make a variable if unsure
  - Assign on previous line

```python
def middle(text):
    """Returns: middle 3rd of text
    Param text: a string"
    # Get length of text
    # Start of middle third
    # End of middle third
    # Get the text
    # Return the result
    return result
```
Defining a String Function

- Start w/ string variable
  - Holds string to work on
  - Make it the parameter

- Body is all assignments
  - Make variables as needed
  - But last line is a return

- Try to work in reverse
  - Start with the return
  - Figure ops you need
  - Make a variable if unsure
  - Assign on previous line

```python
def middle(text):
    """Returns: middle 3rd of text
    Param text: a string""
    # Get length of text
    # Start of middle third
    # End of middle third
    # Get the text
    result = text[start:end]
    # Return the result
    return result
```
Defining a String Function

- Start w/ string variable
  - Holds string to work on
  - Make it the parameter
- Body is all assignments
  - Make variables as needed
  - But last line is a return
- Try to work in reverse
  - Start with the return
  - Figure ops you need
  - Make a variable if unsure
  - Assign on previous line

```python
def middle(text):
    """Returns: middle 3rd of text
    Param text: a string"
    # Get length of text
    # Start of middle third
    end = 2*len(text)//3
    # End of middle third
    result = text[start:end]
    # Get the text
    # Return the result
    return result
```

9/10/18
Defining a String Function

- Start w/ string variable
  - Holds string to work on
  - Make it the parameter

- Body is all assignments
  - Make variables as needed
  - But last line is a return

- Try to work in reverse
  - Start with the return
  - Figure ops you need
  - Make a variable if unsure
  - Assign on previous line

```python
def middle(text):
    """Returns: middle 3rd of text
    Param text: a string""
    # Get length of text
    # Start of middle third
    start = size//3
    # End of middle third
    end = 2*size//3
    # Get the text
    result = text[start:end]
    # Return the result
    return result
```

9/10/18 Defining Functions 13
Defining a String Function

• Start w/ string variable
  ▪ Holds string to work on
  ▪ Make it the parameter

• Body is all assignments
  ▪ Make variables as needed
  ▪ But last line is a return

• Try to work in reverse
  ▪ Start with the return
  ▪ Figure ops you need
  ▪ Make a variable if unsure
  ▪ Assign on previous line

```python
def middle(text):
    """Returns: middle 3rd of text
    Param text: a string""
    # Get length of text
    size = len(text)
    # Start of middle third
    start = size // 3
    # End of middle third
    end = 2 * size // 3
    # Get the text
    result = text[start:end]
    # Return the result
    return result
```
Defining a String Function

```python
def middle(text):
    """Returns: middle 3rd of text
    Param text: a string""
    # Get length of text
    size = len(text)
    # Start of middle third
    start = size // 3
    # End of middle third
    end = 2 * size // 3
    # Get the text
    result = text[start:end]
    # Return the result
    return result
```

```plaintext
>>> middle('abc')
'b'
>>> middle('aabbcc')
'bb'
>>> middle('aaabbbccc')
'bbb'
```
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
parameters
local variables (later in lecture)
instruction counter
```
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

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

Initial call frame (before exec body)
**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

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

- Executing the return statement
- Return statement creates a special variable for result
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):
    return 5*(x-32)/9.0
```

Executing the return statement

```
def to_centigrade(x):
    x = 50.0
    return 10.0
```

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

```python
def to_centigrade(x):
    return 5*(x-32)/9.0
```
Visualizing Frames: The Python Tutor

```python
def max(x, y):
    if x > y:
        return x
    return y

a = 1
b = 2
max(a, b)
```

Frames

Objects

Global frame

function max(x, y)

<table>
<thead>
<tr>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
</tr>
<tr>
<td>y</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
</tr>
</tbody>
</table>
Visualizing Frames: The Python Tutor

Global Space

Call Frame

1. def max(x, y):
   2.     if x > y:
   3.         return x
   4. return y
5.
6. a = 1
7. b = 2
8. max(a, b)
Visualizing Frames: The Python Tutor

Global Space

Call Frame

Variables from second lecture go in here
Visualizing Frames: The Python Tutor

```python
1  def max(x,y):
2      if x > y:
3          return x
4      return y
5
6  a = 1
7  b = 2
8  max(a,b)
```

Frames

Objects

Global frame:

```
max
```

```
x | 1
y | 2
```
Visualizing Frames: The Python Tutor

Line number marked here (sort-of)

```
1 def max(x,y):
2     if x > y:
3         return x
4     return y
5
6 a = 1
7 b = 2
8 max(a,b)
```

Missing line numbers!