

# CSCI 141

Lecture 13:  
Midterm review; Functions, continued

# Announcements

- A2 grades out this afternoon.
- Exam material: range(functions)
  - that is, 0 up to but not including writing your own functions

# Announcements

- Sample study problems are posted. Solutions are not verified.
  - Many “execute this code” questions.
  - Underrepresented topics:
    - binary/decimal
    - distinction between statements and expressions
    - computer hardware (CPU, main memory, fetch/decode/execute)
    - Syntax errors
    - Modules and imports
    - Algorithm development

# Goals

- Review for the midterm.
- As time allows:
  - Know the syntax for defining your own **functions**
  - Know how to define and use functions that take no arguments and return no values
  - Know how to use **parameters** to refer to the input arguments of a function
  - Know the meaning of **local variables** and **variable scope** and how it relates to function parameters.
  - Know how to **return** a value from a function.

# Midterm Review: Questions

# Functions, Revisited

What **is** a function, anyway?

- As a user, you can treat a function as a “**black box**”:  
all you need to know is:
  - the **inputs, effects,** and **return value.**
- Functions are named chunks of code.



A bunch of (complicated)  
stuff is wrapped up in a nice,  
easy-to-use package.

# Writing Functions: Syntax

Input(s)

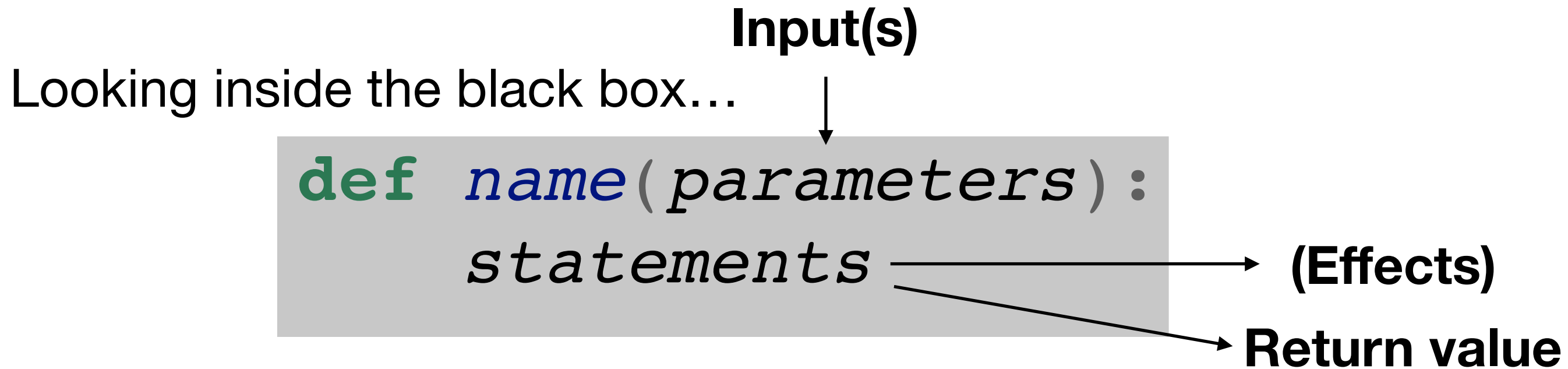
Looking inside the black box...

```
def name(parameters):  
    statements
```

(Effects)

Return value

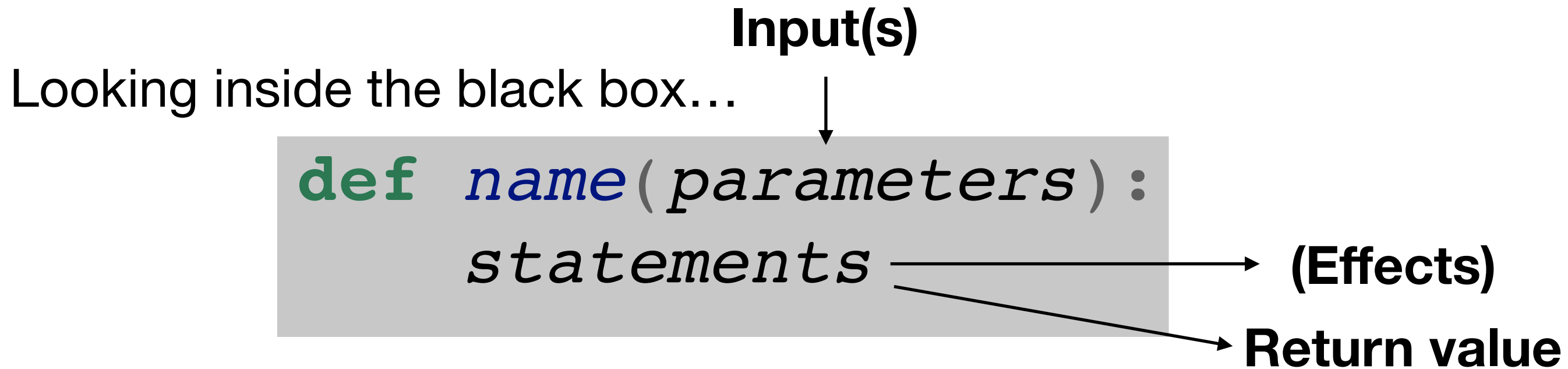
# Writing Functions: Syntax



Two important questions:



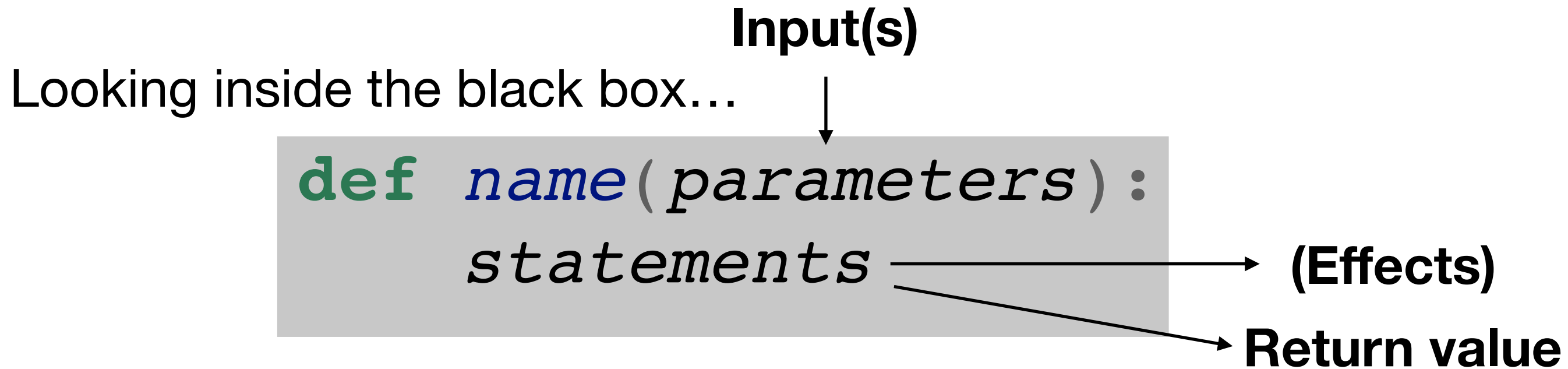
# Writing Functions: Syntax



Two important questions:

1. How does the function use the arguments (inputs) passed to it?

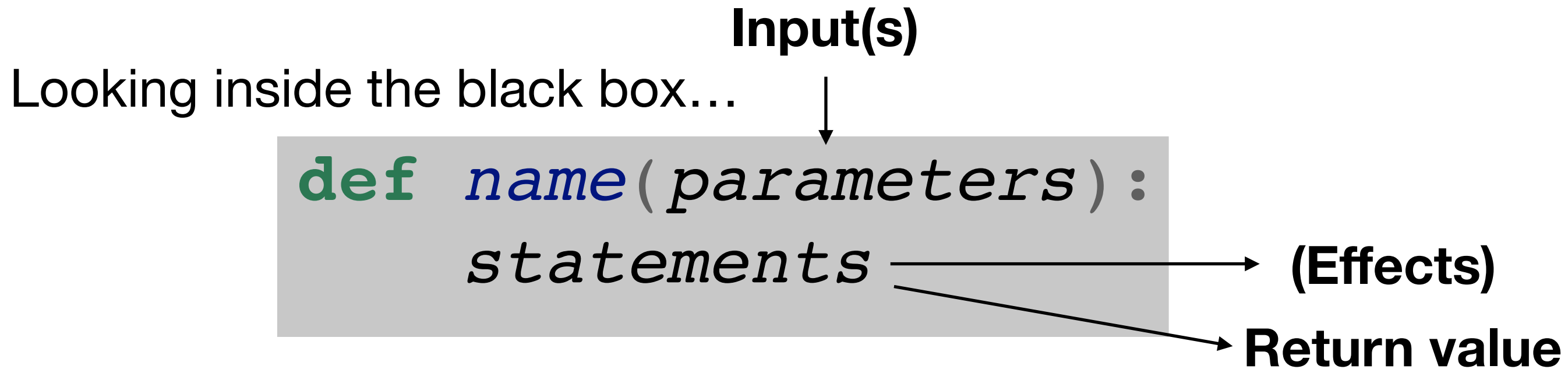
# Writing Functions: Syntax



Two important questions:

1. How does the function use the arguments (inputs) passed to it?
2. How does the function return a value?

# Writing Functions: Syntax



Two important questions:

1. How does the function use the arguments (inputs) passed to it?
2. How does the function return a value?

Let's **dodge** these questions for a moment...

# Functions: the simplest kind

No arguments, no return value:

```
def name() :  
    statements
```

**Example:**

```
def print_hello() :  
    print( "Hello, world!" )
```

# Demo: Function to print a rectangle of # symbols

**Input(s):**

- none

— `print_rectangle` —

**Return value:**

- none

**Effects:** prints a 2x50 rectangle of #s to the screen

# Function to print a rectangle of symbols

```
def print_rectangle():  
    """ Prints a 2x50 rectangle of a  
        user-specified character """  
    user_char = input("What character? ")  
    for i in range(2):  
        print(user_char * 50)
```

# Function to print a rectangle of symbols

```
def print_rectangle():  
    """ Prints a 2x50 rectangle of a  
        user-specified character """  
    user_char = input("What character? ")  
    for i in range(2):  
        print(user_char * 50)
```

Aside: what's `""" this """` about? Two things in one:

# Function to print a rectangle of symbols

```
def print_rectangle():  
    """ Prints a 2x50 rectangle of a  
        user-specified character """  
    user_char = input("What character? ")  
    for i in range(2):  
        print(user_char * 50)
```

Aside: what's `""" this """` about? Two things in one:

- **Multiline strings:** An alternate way to write strings that include newlines.



# Function to print a rectangle of symbols

```
def print_rectangle():  
    """ Prints a 2x50 rectangle of a  
        user-specified character """  
    user_char = input("What character? ")  
    for i in range(2):  
        print(user_char * 50)
```

Aside: what's `""" this """` about? Two things in one:

- **Multiline strings:** An alternate way to write strings that include newlines.
- A **docstring:** The conventional way to write comments that describe the purpose and behavior of a function.

# Multiline Strings and Docstrings: Demo

```
def print_rectangle():  
    """ Prints a 2x50 rectangle of a  
        user-specified character """  
    user_char = input("What character? ")  
    for i in range(2):  
        print(user_char * 50)
```

# Multiline Strings and Docstrings: Demo

```
def print_rectangle():  
    """ Prints a 2x50 rectangle of a  
        user-specified character """  
    user_char = input("What character? ")  
    for i in range(2):  
        print(user_char * 50)
```

- Multiline strings: printing, assigning, etc.
- A string on a line by itself has no effect on the program.
- Docstrings in functions are like comments (but aren't, technically)

# Docstrings

# Docstrings

Docstrings are **not** required by the language.

# Docstrings

Docstrings are **not** required by the language.

Docstrings **are** required by me.

# Docstrings

Docstrings are **not** required by the language.

Docstrings **are** required by me.

- A docstring tells you **what** the function does, but not **how** it does it.

# Docstrings

Docstrings are **not** required by the language.

Docstrings **are** required by me.

- A docstring tells you **what** the function does, but not **how** it does it.
- In other terms, it tells you what you need to know to **use** the function, but not what the function's author needed to know to **write** it.



# Docstrings: Example

The (actual) source code for `turtle.forward`:

Docstring:

```
def forward(self, distance):  
    """Move the turtle forward by the specified distance.  
  
    Aliases: forward | fd  
  
    Argument:  
    distance -- a number (integer or float)  
  
    Move the turtle forward by the specified distance, in the direction  
    the turtle is headed.  
  
    Example (for a Turtle instance named turtle):  
    >>> turtle.position()  
    (0.00, 0.00)  
    >>> turtle.forward(25)  
    >>> turtle.position()  
    (25.00,0.00)  
    >>> turtle.forward(-75)  
    >>> turtle.position()  
    (-50.00,0.00)  
    """
```

Implementation: `self._go(distance)`

# Docstrings: Example

The (actual) source code for `turtle.forward`:

Docstring:

```
def forward(self, distance):  
    """Move the turtle forward by the specified distance.
```

```
    Aliases: forward | fd
```

```
    Argument:
```

```
    distance -- a number (integer or float)
```

```
    Move the turtle forward by the specified distance, in the direction  
    the turtle is headed.
```

```
    Example (for a Turtle instance named turtle):
```

```
>>> turtle.position()  
(0.00, 0.00)  
>>> turtle.forward(25)  
>>> turtle.position()  
(25.00,0.00)  
>>> turtle.forward(-75)  
>>> turtle.position()  
(-50.00,0.00)  
"""
```

Implementation: `self._go(distance)`

# Docstrings: Example

Python documentation is generated from the docstrings in the code!

```
turtle.forward(distance)
```

```
turtle.fd(distance)
```

**Parameters:** *distance* – a number (integer or float)

Move the turtle forward by the specified *distance*, in the direction the turtle is headed.

```
>>> turtle.position()
(0.00,0.00)
>>> turtle.forward(25)
>>> turtle.position()
(25.00,0.00)
>>> turtle.forward(-75)
>>> turtle.position()
(-50.00,0.00)
```

```
>>>
```

# Docstrings: Example

Python documentation is generated from the docstrings in the code!

```
turtle.forward(distance)
```

```
turtle.fd(distance)
```

**Parameters:** *distance* – a number (integer or float)

Move the turtle forward by the specified *distance*, in the direction the turtle is headed.

```
>>> turtle.position()
(0.00,0.00)
>>> turtle.forward(25)
>>> turtle.position()
(25.00,0.00)
>>> turtle.forward(-75)
>>> turtle.position()
(-50.00,0.00)
```

```
>>>
```

# Worksheet Exercise 1

```
def name():  
    statements
```

Exercise 1: Define a function named `print_word`, which prompts the user to input a word, and also prompts the user to specify how many times that word should be printed. The function should then print that word to the screen as many times as the user has indicated. Invoke the function (hint: the function takes no parameters (no arguments)).

## Input(s):

- none



## Return value:

- none

**Effects:** prompts the user to input a word and a number of repetitions  
prints the word that many times

# Writing Functions: Syntax

```
def name(parameters):  
    statements
```

Two important questions:

- 1. How does the function use the arguments (inputs) passed to it?**
2. How does the function return a value?

Let's **dodge** these questions for a moment...

# Writing Functions: Syntax

1. How does the function use the arguments (inputs) passed to it?

def keyword

function name

**def** *name*(*parameters*):  
*statements*

The diagram illustrates the syntax of a Python function definition. It shows two boxes at the top: 'def keyword' on the left and 'function name' on the right. Arrows point from these boxes to the corresponding parts of the function definition syntax below. The 'def' keyword is highlighted in green, and the 'name' parameter is highlighted in blue. The parameters and statements are shown in italics.

# Writing Functions: Syntax

1. How does the function use the arguments (inputs) passed to it?

def keyword

function name

**def** *name*(*parameters*):  
*statements*

**inputs**

comma-separated  
list of **parameters**:  
variable names that  
will refer to the input  
arguments



# Demo: Function to print a rectangle of a symbol passed in as an argument.

## Input(s):

- character to make a rectangle out of

## Return value:

- none

→ `print_rectangle` →

**Effects:** prints a 2x50 rectangle of the given character to the screen

# Writing Functions: Syntax

1. How does the function use the arguments (inputs) passed to it?

def keyword

function name

**def** *name*( *parameters* ) :  
*statements*

The diagram illustrates the syntax of a Python function definition. It shows two boxes at the top: 'def keyword' on the left and 'function name' on the right. Arrows point from these boxes to the corresponding parts of the function definition syntax below. The 'def' keyword is highlighted in green, and the 'function name' is highlighted in blue. The parameters and statements are shown in italics.

# Writing Functions: Syntax

1. How does the function use the arguments (inputs) passed to it?

def keyword

function name

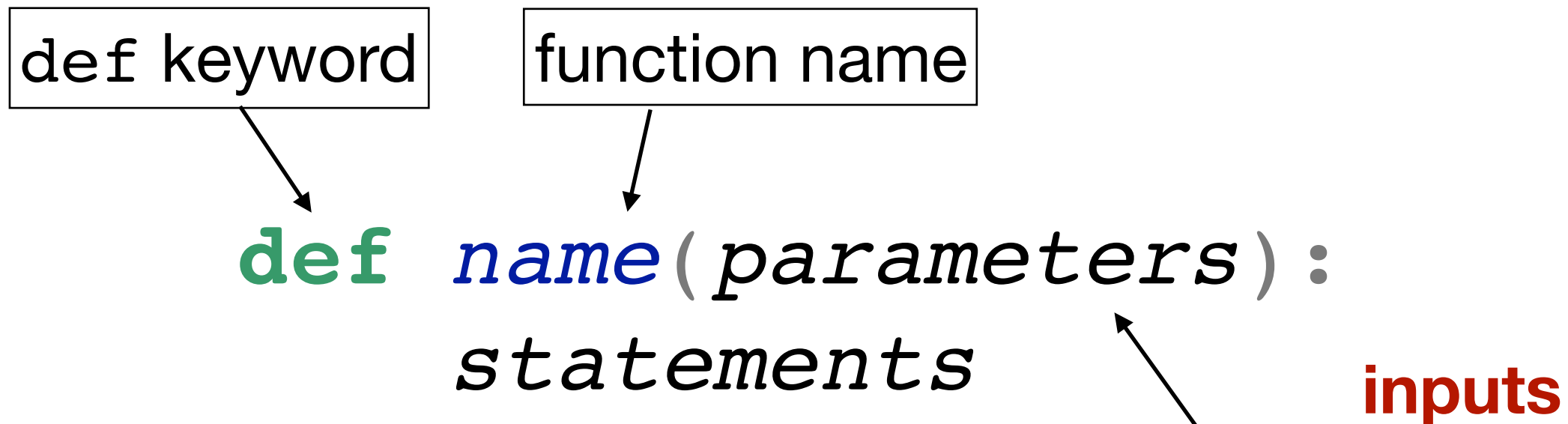
**def** *name*(*parameters*):  
*statements*

**inputs**

comma-separated  
list of **parameters**:  
variable names that  
will refer to the input  
arguments

# Writing Functions: Syntax

1. How does the function use the arguments (inputs) passed to it?



Inside the function, the parameters act as **local variables** that refer to the arguments passed into the function.

comma-separated list of **parameters**: variable names that will refer to the input arguments

# Worksheet Exercise 2

Exercise 2: Write (define) a function that adds two numbers and prints their sum. Then use that function (invoke it) in a python program.

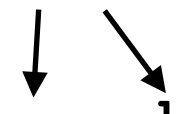
# Parameters vs Arguments

**Parameters:** variable names that will refer to the input arguments.

Parameters (these are new):

**variables** that take on the value of the arguments

```
def add2(a, b):  
    """ Print the sum of a and b """  
    print(a + b)
```



```
add2(4, 10)
```

Arguments (we've seen these before):

**values** passed into a function.



# Parameters are Local Variables

- They **only** exist inside the function.
- Any other variables declared inside a function are also local variables.
- This is an example of a broader concept called **scope**: a variable's scope is the set of statements in which it is visible/usable.
- A local variable's scope is limited to the function inside which it's defined.

# Parameters and Local Variables: Demo

- `add2.py`



# Parameters and Local Variables: Demo

- `add2.py`:
  - parameters as local variables (inaccessible outside fn)
  - other local variables
  - variables getting passed in
  - variables shadowing other variables

# Demo: Function to draw a square using a turtle

# Demo: Function to draw a square using a turtle

- the convenience of repetition:
  - you can define a function once then call it as many times as you want
- the power of *customized* repetition:
  - you can define a function that takes arguments to customize the task it performs: this is powerful!
  - e.g.: one function to draw any size rectangle.

# Writing Functions: Syntax

```
def name(parameters):  
    statements
```

Two important questions:

**2. How does the function return a value?**

# Exercise 3

Has a typo. Should say:

- Defines a function that takes a single argument and **prints** the fourth power of the input argument.

# Returning values

New statement: the `return` statement

Syntax: `return expression`

Behavior:

1. *expression* is evaluated
2. the function stops executing further statements
3. the value of expression is returned

# Function Syntax: Summary

def keyword

function name

**def** *name*(*parameters*):  
*statements*

**inputs**

comma-separated  
list of **parameters**:  
variable names that  
will get assigned to  
the arguments

An indented code block that  
does any computation,  
executes any effects, and  
(optionally) **returns** a value

**effects; return value**

# Returning values: Why?

- Next time:
- Using the result of one computation as the input to another: function composition.