Goals

• Know the syntax for triple-quoted strings

• Know the convention for writing docstrings that describe a function’s specification

• Know what does and does not belong in a function specification

• Know the definition and purpose of preconditions and postconditions
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.

\[
\text{Input(s)} \rightarrow \text{(Effects)} \rightarrow \text{Return value}
\]

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

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

- **def** keyword
- **function name**
- **parameters**: comma-separated list of variable names that will get assigned to the arguments
- **statements**: An indented code block that does any computation, executes any effects, and (optionally) returns a value
Why are functions great?

- **Concise** - wrap something complicated in an easy-to-use package

- **Customizable** - make the easy-to-use package do different things

- **Composable** - use the result of one computation as input to (or as one step in) another
Demo: Function to draw a square using a turtle
Demo: Function to draw a square using a turtle

- Concise: `turtle_square` call tells the turtle to do a bunch of things

- Customizable: `turtle_rectangle(t, w, h)` function draws a w-by-h rectangle

- add docstrings at the end!
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.

```python
def turtle_rectangle(t, w, h):
    """ Draw a w-by-h rectangle using turtle t """
    for i in range(2):
        t.forward(w)
        t.left(90)
        t.forward(h)
        t.left(90)
```

Multiline Strings and Docstrings: Demo
Multiline Strings and Docstrings: Demo

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

Docstrings **are** required by me from now on.

- 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:

```python
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)
    '''
    self._go(distance)
```

Implementation: `self._go(distance)`
Docstrings: Example

Python documentation is generated from the docstrings in the code!

turtle.\texttt{forward}(\textit{distance})
turtle.\texttt{fd}(\textit{distance})

\textbf{Parameters:} \textit{distance} – a number (integer or float)

Move the turtle forward by the specified \textit{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)
What belongs in a docstring?

As a user, you can treat a function as a “black box”: all you need to know is:
- the inputs, effects, and return value.

Docstrings give a user of your function everything they need to know to call it.

A docstring should explain what the function does, but not how the function works.
What belongs in a docstring?

![Diagram showing Input(s) → f → Return value (Effects)]

**preconditions:**
Things the **caller** is responsible for ensuring before the function is called.

*like comments, these are human constructs, not part of Python*

**postconditions:**
Things the **function** is responsible for ensuring by the time the function returns.
Preconditions: why?

- Demo: abs.py

- Absolute value only makes sense on numbers, so specify a precondition that the input must be a number.
Postconditions: why?

- Demo: turtle_rectangle.py

- It's important to know where a Turtle function leaves the turtle so you know how to continue with your drawing.

- Specify a **postcondition** that the turtle ends up in the same position and direction as it started.
Preconditions and Postconditions: Assigning Blame

Example. Suppose you wrote this function:

```python
def split_bill(bill_amt, tip_pct, num_diners):
    """ Return the total owed by each diner for a restaurant bill of bill_amt, assuming a tip percent of tip_pct and splitting the bill evenly among num_diners people. """

    total = bill_amt + (bill_amt * tip_pct/100)
    return total / num_diners

>>> split_bill(34.78, 18.0, 0)
ZeroDivisionError: float division by zero

Bad news: This is your fault.
Preconditions and Postconditions: Assigning Blame

Example. Suppose you wrote this function:

```python
def split_bill(bill_amt, tip_pct, num_diners):
    """ Return the total owed by each diner for a restaurant bill of bill_amt, assuming a tip percent of tip_pct and splitting the bill evenly among num_diners people."
    """
    total = bill_amt + (bill_amt * tip_pct/100)
    return total / num_diners
```

>>> split_bill(34.78, 18.0, 0)
ZeroDivisionError: float division by zero

Good news: This is my fault.