Functions

CSC120

Mark Kazakevich

Functions seem useful!

- Without using functions in Python, all we have is a few math operators:
  - +, -, *, //, %, etc.
- But we want to do more than that!
- The functions we've used so far demonstrate that indeed we can do more than just basic operations in Python

But notice something...

- When I told you about functions like abs(x), how much time did you spend thinking about **how** they work?

- Most likely, not much.
  - You probably just thought "Ok, cool. Python can find the absolute value" and just used the function to do your work

- That's a good thing! And….
  - It's actually part of what functions are all about!

Functions are meant to hide the details

- We don't know what the computer actually does to calculate the absolute value of a number
- If we needed to know how every function works before we can use it, we would never get any work done!
- Right now, we want to use functions to do interesting things
- When we see something like abs(x), we assume it's going to work as expected:
  - We assume the people who made Python also made a good absolute value function for it

So how do functions in Python work?

How functions work

The three parts to a what a **function in Python** does:

1. ‘Call’ the function with some arguments
2. Do something with the argument values
3. Produce a ‘return value’

Part 1: “Calling” the function

- Whenever we type a function in the shell and press enter, we are **‘calling’** that function

```
> abs(9)
```

“I called the function `abs` with the value 9”

- The ‘9’ is called an **argument** to the function

Part 1: “Calling” the function

- Some functions can take more than one argument

```
> max(3, 5)
```

“I called the function `max` with the values 3 and 5”

Arguments (con't)

- Argument can be a **literal** value (no further evaluation needed)

```
> abs(-2)
```

Argument value: -2

- Or, it can be an expression

```
> abs(10 + 6)
```

Argument value: 10 + 6 = 16

- Because arguments can be expressions, we have to **evaluate** them before the function can work with them

So that's part 1

The three parts to a what a **function in Python** does:

1. 'Call' the function with some arguments
2. Do something with the argument values
3. Produce a 'return value'

Part 2: Do something with the input

- A function that is called with an argument should do something with that argument

```
> abs(-2)
```

- Python does something to find the absolute value of -2

and that's part 2!

The three parts to a what a **function in Python** does:

1. ‘Call’ the function with some arguments
2. Do something with the argument values
3. Produce a ‘return value’

Part 3: Produce a “return value”

- In the shell, after we press Enter and the function is called, we saw that we get back a value

```
> abs(-2)
2
```

  - The ‘2’ that we get back after calling abs(-2) is called the **return value** of the function

Return Values

- The return value of a function is the value that the function **evaluates** to.

- For example, we say `abs(-2)` **evaluates** to 2

- We can use the return value of the function the same way we use the value we get from an expression
  - We can assign it to a variable
  - We can use it as an argument to another function

We can assign the return values to variables

```
> a = abs(-2)
> a
2
```

We can use them as arguments to other functions

```
> max(abs(-2), 1)
2
```

Let's say we wanted a function that adds 3 to an int.

There is no Python built-in function for that.

So we create our own!

Let’s see how to define our own functions