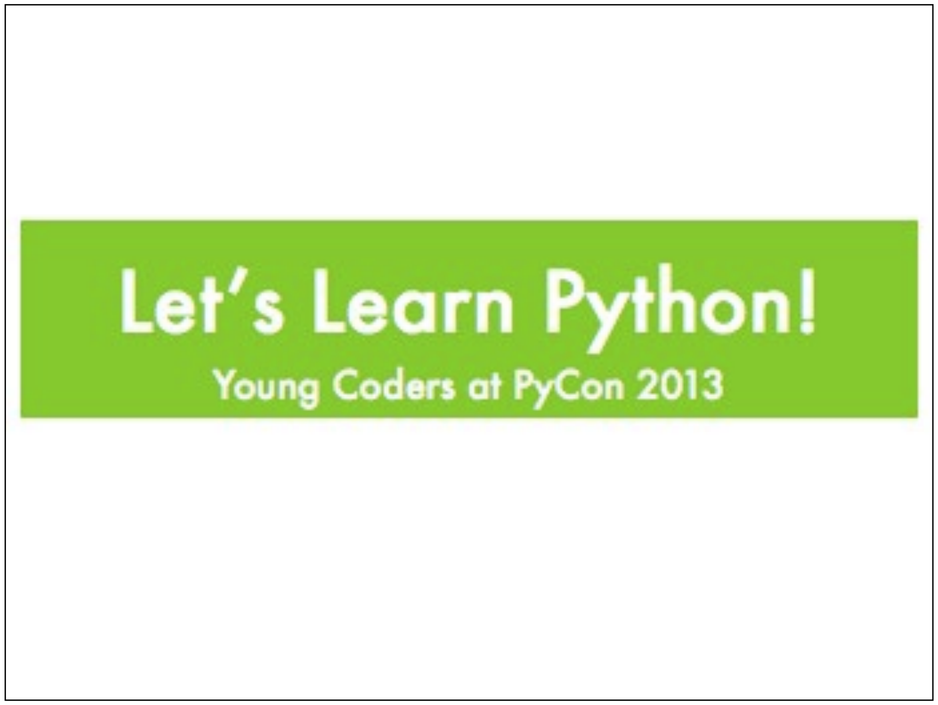


Let's Learn Python!

Young Coders at PyCon 2013

Pass out name tags for everyone

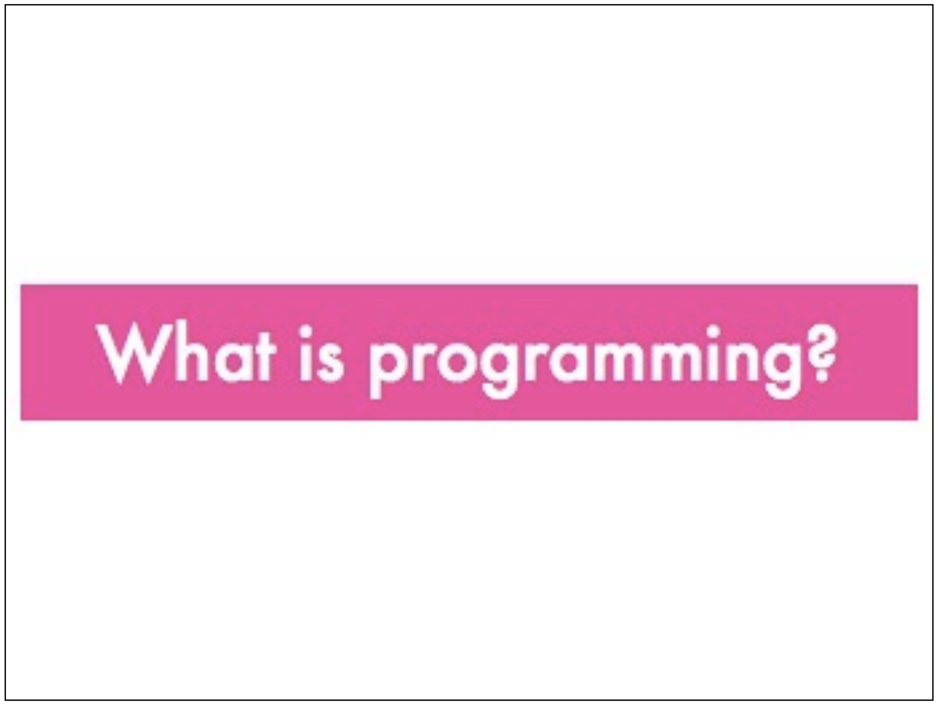
Meet your teachers:

Katie Cunningham
Barbara Shaurette

Each teacher should introduce herself/himself, talk about what they do with programming

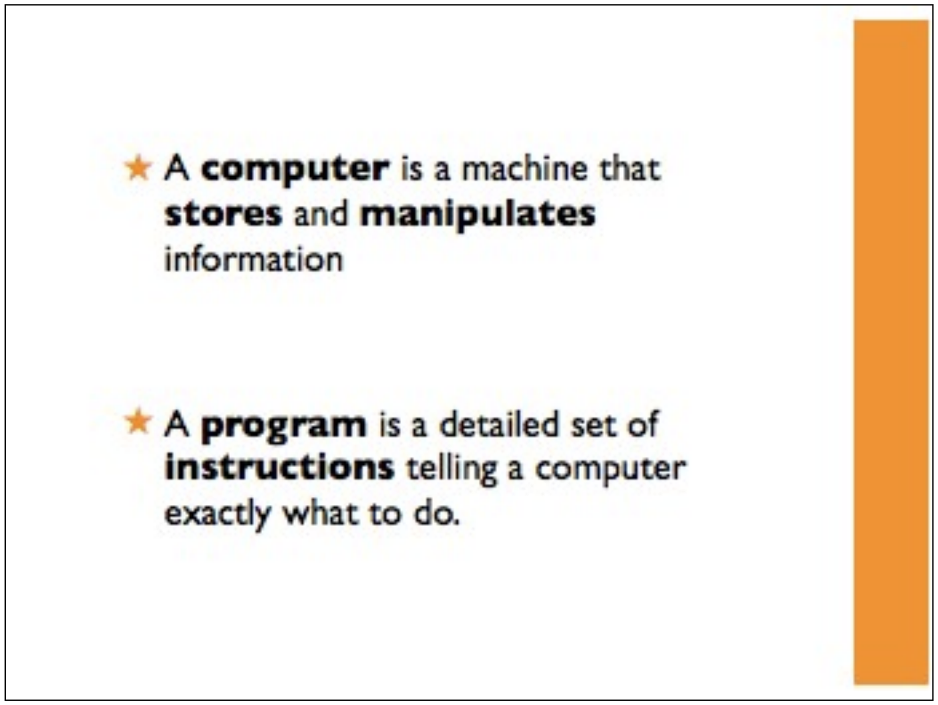
Have them open Idle here

Terms to introduce: shell/interpreter, prompt

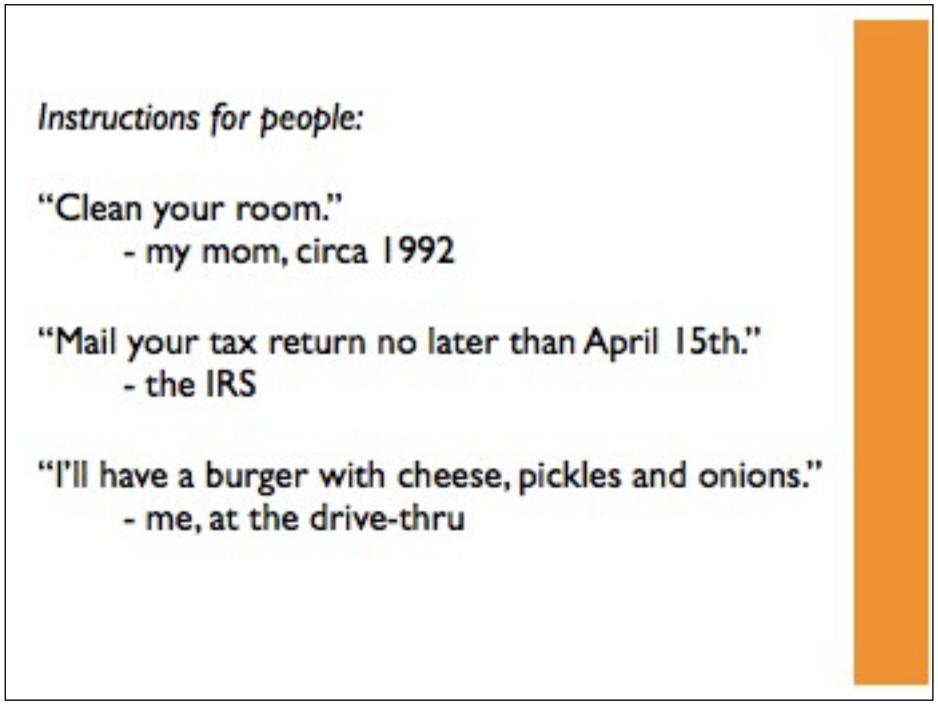


What is programming?

“First we’ll introduce some basic programming concepts, then we’ll talk about some of the specific ways we use the Python language to write programs.”



- ★ A **computer** is a machine that **stores** and **manipulates** information
- ★ A **program** is a detailed set of **instructions** telling a computer exactly what to do.



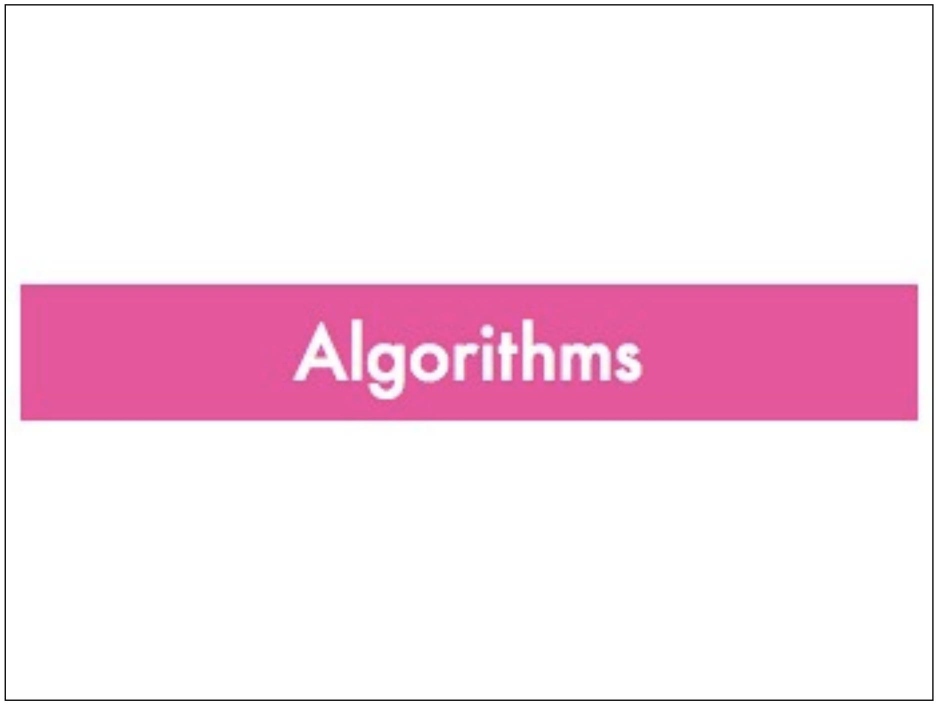
Instructions for people:

"Clean your room."
- my mom, circa 1992

"Mail your tax return no later than April 15th."
- the IRS

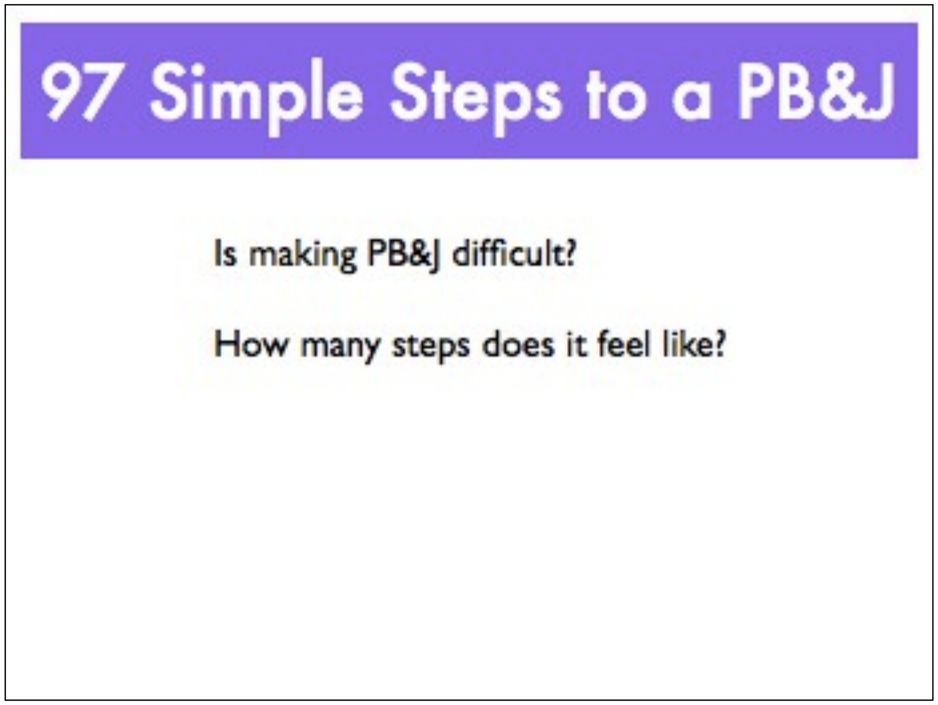
"I'll have a burger with cheese, pickles and onions."
- me, at the drive-thru

"Computers are really dumb – they'll only do exactly what you tell them to do. So you have to be the smart one and give them good instructions. That's what we're going to learn about here today."



Algorithms

"'Algorithm' is a name for the instructions we give to computers – they're like recipes, with specific steps to follow."



97 Simple Steps to a PB&J

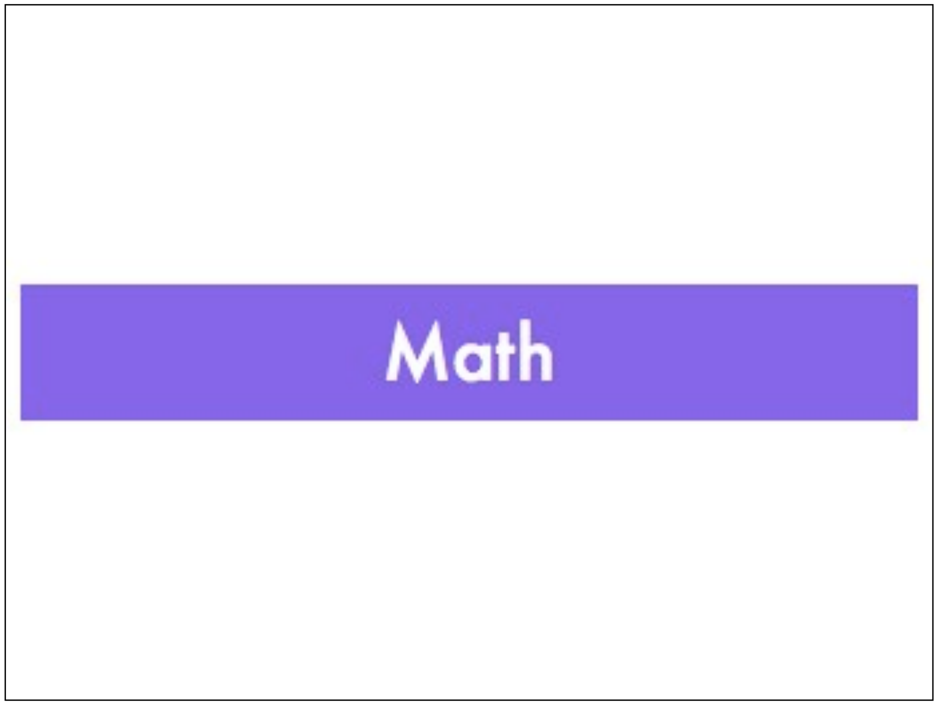
Is making PB&J difficult?

How many steps does it feel like?

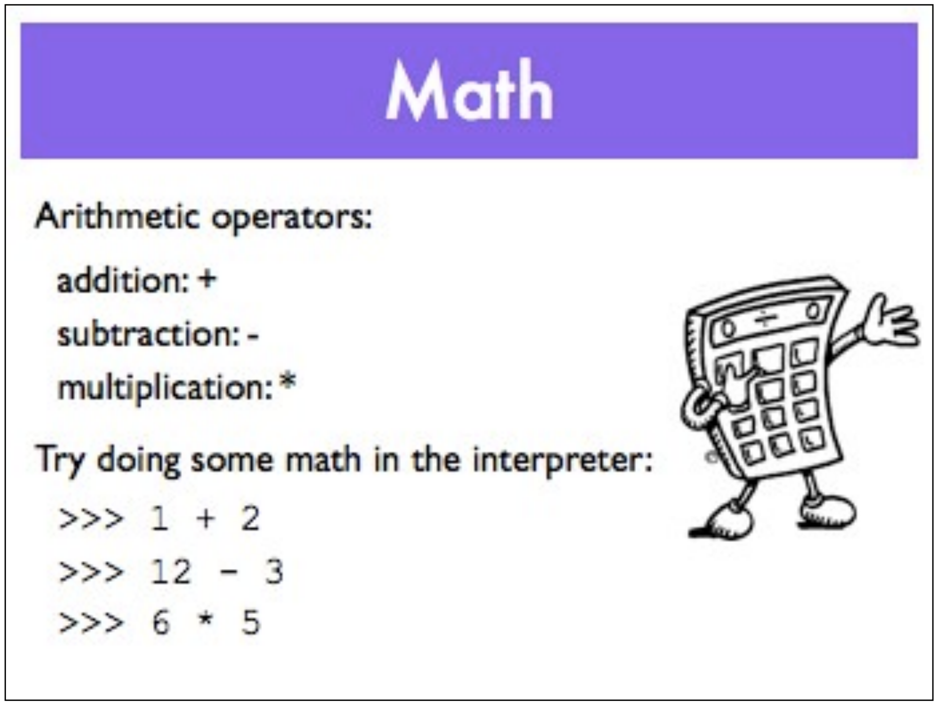
Demonstration: go through the steps of making a peanut butter and jelly sandwich with one team member giving instructions and another performing the actions. Involve the kids by asking them to call out instructions.

Let's talk to Python!

"Python is just one of the languages we use to talk to computers, but there are many others."



Math



Math

Arithmetic operators:

addition: +

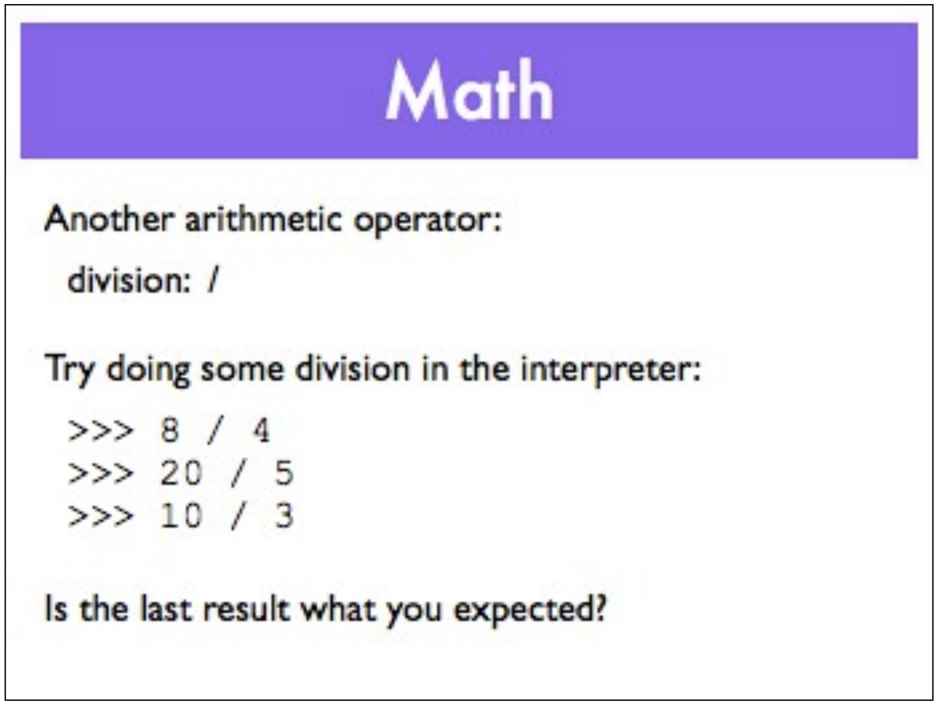
subtraction: -

multiplication: *

Try doing some math in the interpreter:

```
>>> 1 + 2
>>> 12 - 3
>>> 6 * 5
```

Term to introduce: expression



Math

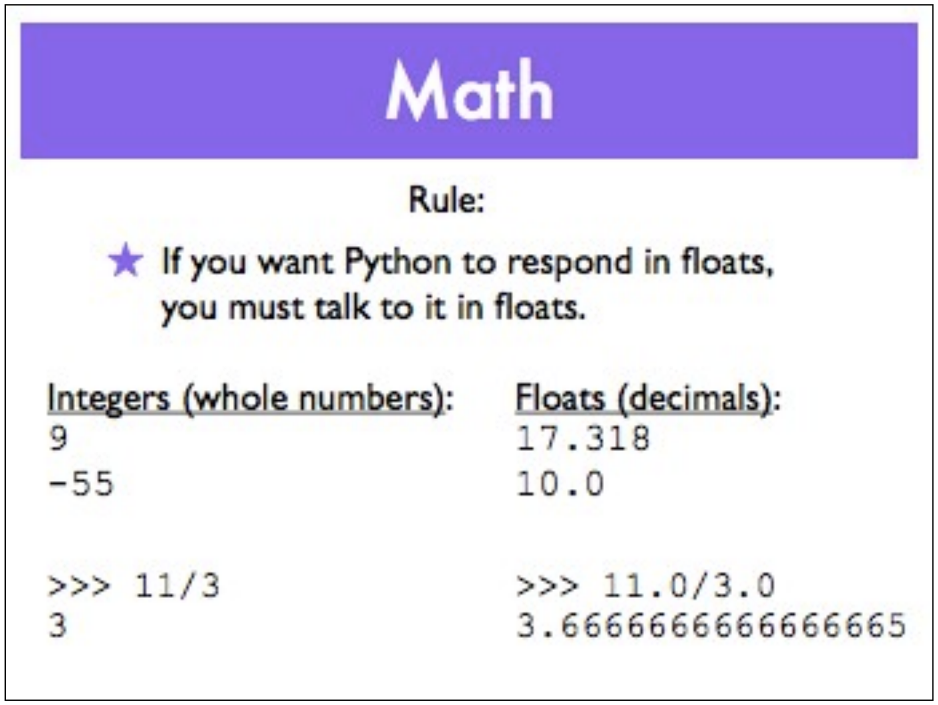
Another arithmetic operator:

division: /

Try doing some division in the interpreter:

```
>>> 8 / 4
>>> 20 / 5
>>> 10 / 3
```

Is the last result what you expected?



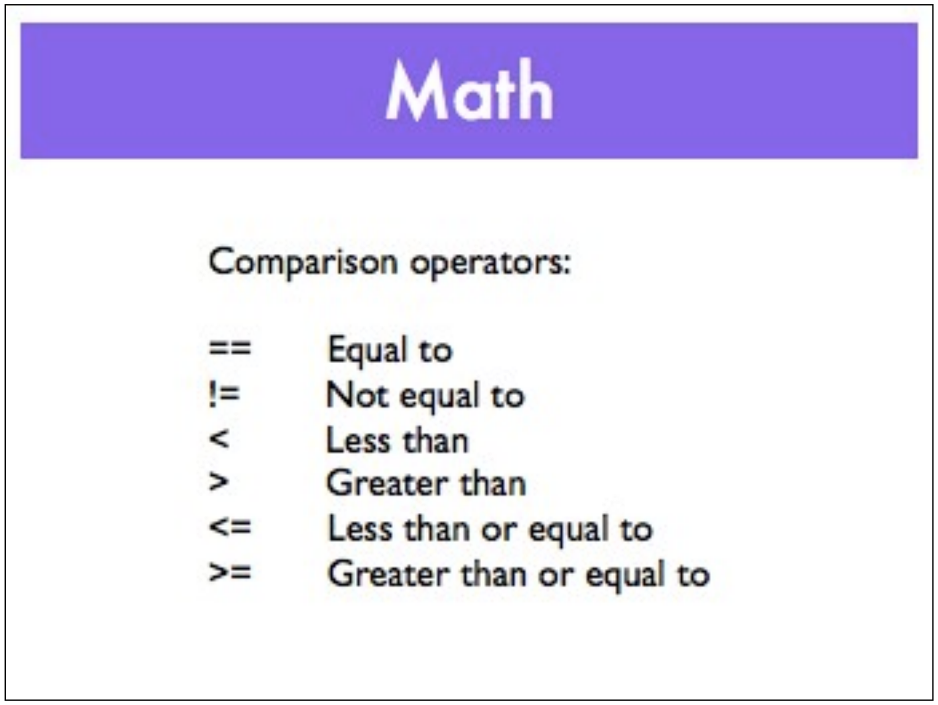
Math

Rule:

★ If you want Python to respond in floats, you must talk to it in floats.

```
Integers (whole numbers):   Floats (decimals):
9                           17.318
-55                         10.0

>>> 11/3                    >>> 11.0/3.0
3                           3.6666666666666665
```

"Here's a new type of number – a 'float', or a decimal number."

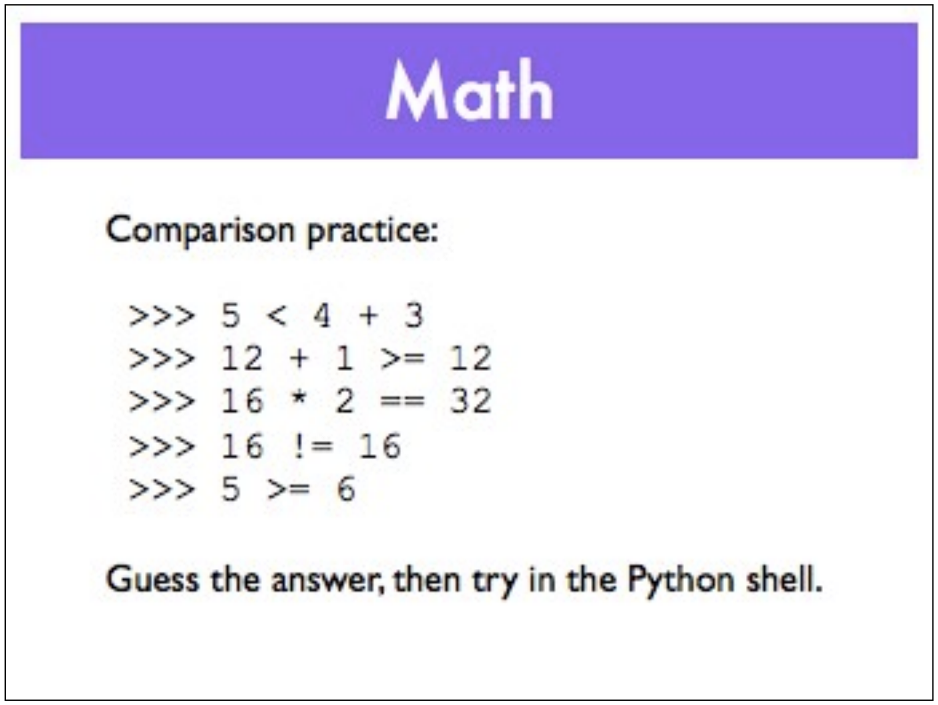


Math

Comparison operators:

==	Equal to
!=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

"The exclamation point is also sometimes called a 'bang'."



Math

Comparison practice:

```
>>> 5 < 4 + 3
>>> 12 + 1 >= 12
>>> 16 * 2 == 32
>>> 16 != 16
>>> 5 >= 6
```

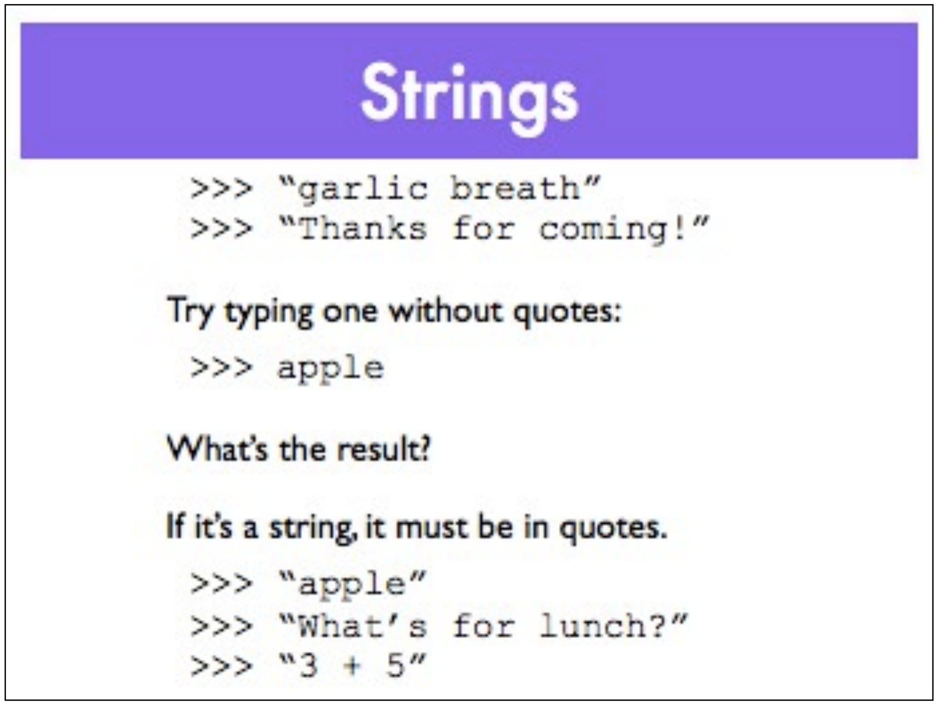
Guess the answer, then try in the Python shell.

Math

Comparison practice:

```
>>> 5 < 4 + 3            True
>>> 12 + 1 >= 12         True
>>> 16 * 2 == 32         True
>>> 16 != 16             False
>>> 5 >= 6               False
```

Strings



Strings

```
>>> "garlic breath"
>>> "Thanks for coming!"
```

Try typing one without quotes:

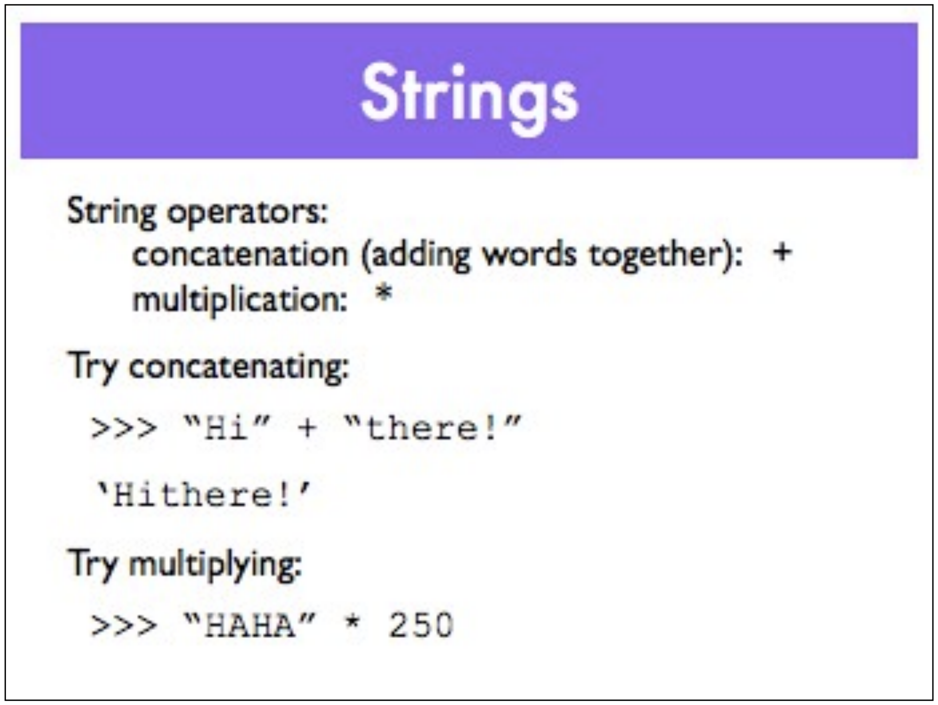
```
>>> apple
```

What's the result?

If it's a string, it must be in quotes.

```
>>> "apple"
>>> "What's for lunch?"
>>> "3 + 5"
```

A string is a character, or a sequence of characters
A number can be a string, if it's wrapped in quotes



Strings

String operators:
  concatenation (adding words together): +
  multiplication: *

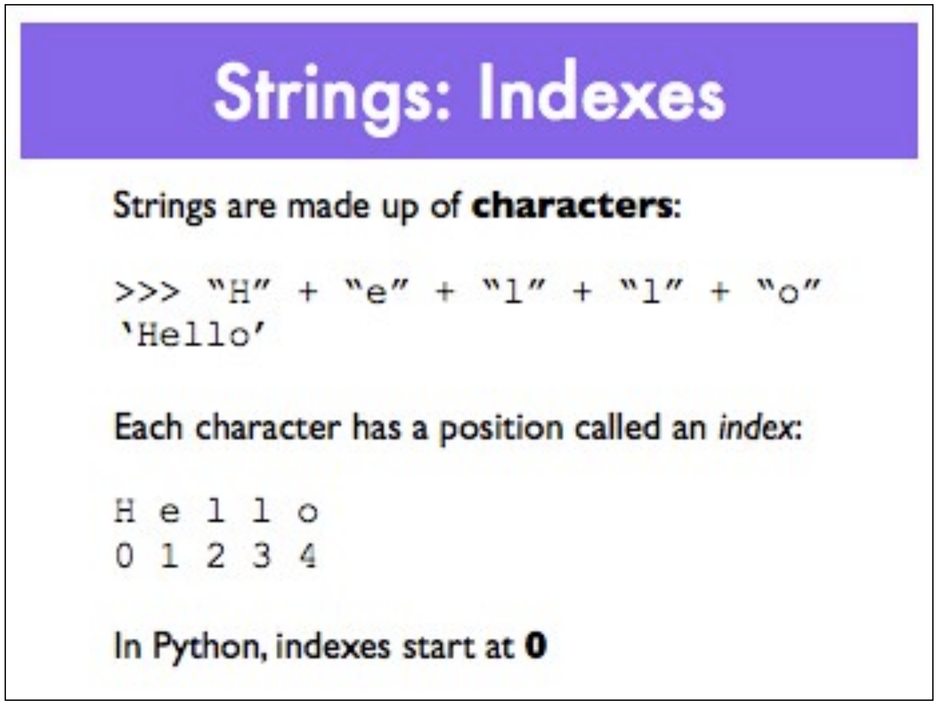
Try concatenating:

```
>>> "Hi" + "there!"
'Hithere!'
```

Try multiplying:

```
>>> "HAHA" * 250
```

Warn them about memory errors if they make the multiplier too high (this was a concern when teaching on Raspberry Pi)



Strings: Indexes

Strings are made up of **characters**:

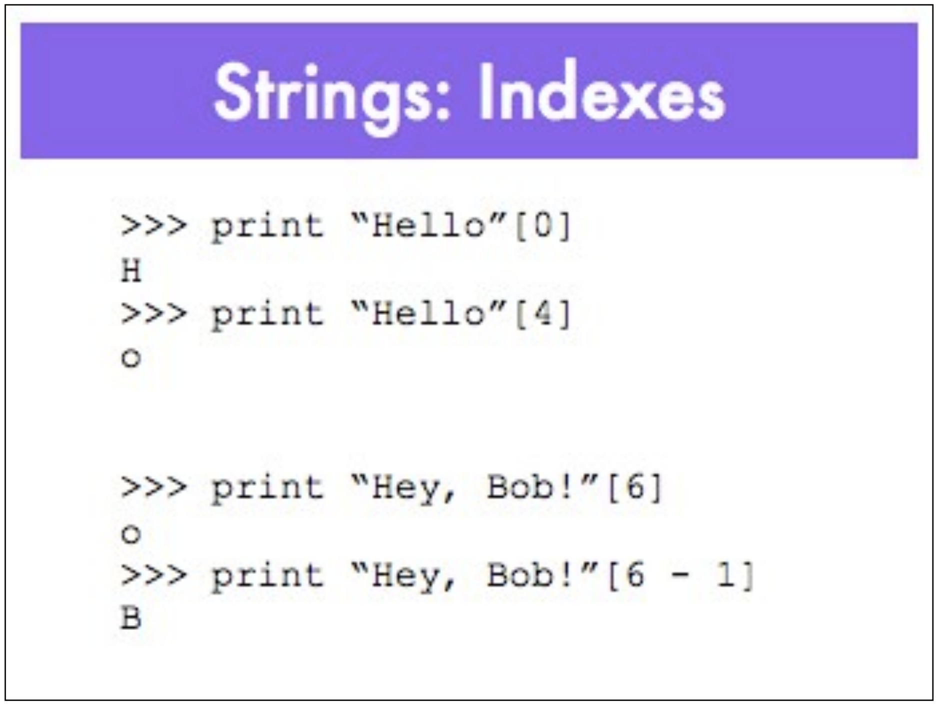
```
>>> "H" + "e" + "l" + "l" + "o"
'Hello'
```

Each character has a position called an *index*:

```
H e l l o
0 1 2 3 4
```

In Python, indexes start at **0**

In the class with older kids, some of them asked "why?" indexing starts at zero – we felt it was too much to address in this class. You might give a brief explanation, then suggest that the kids to research it (via reading or Google) for something deeper.



Strings: Indexes

```
>>> print "Hello"[0]
H
>>> print "Hello"[4]
o


>>> print "Hey, Bob!"[6]
o
>>> print "Hey, Bob!"[6 - 1]
B
```

Strings: Indexes

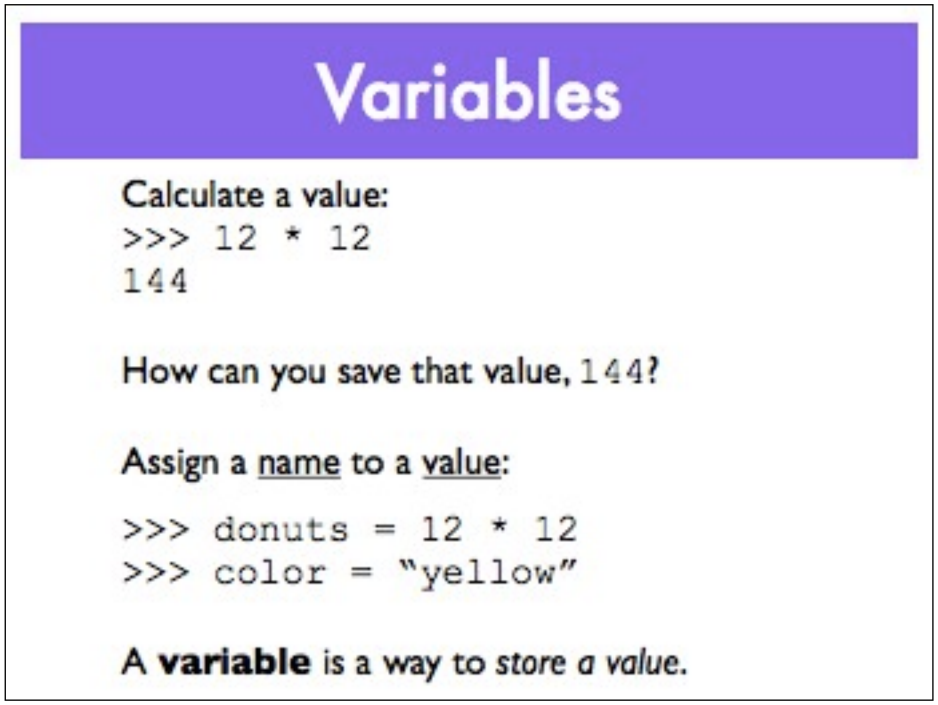
```
>>> print "Hey, Bob!"[4]
```

What did Python print?

Rules:

- ★ Each character's position is called its *index*.
- ★ Indexes start at 0.
- ★ Spaces inside the string are counted.

Variables



Variables

Calculate a value:

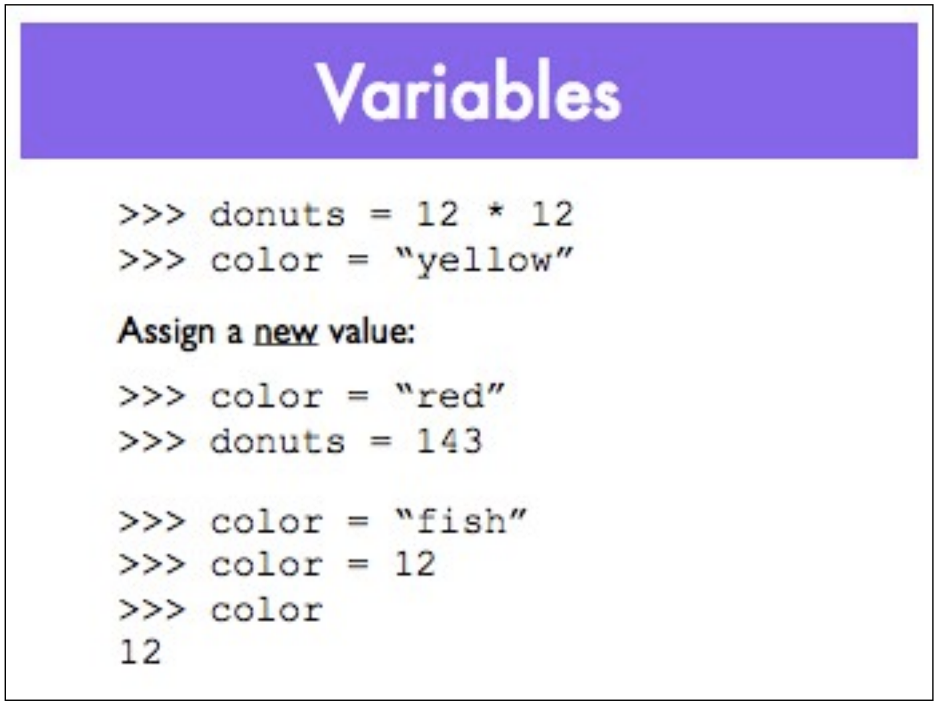
```
>>> 12 * 12
144
```

How can you save that value, 144?

Assign a name to a value:

```
>>> donuts = 12 * 12
>>> color = "yellow"
```

A **variable** is a way to *store a value*.



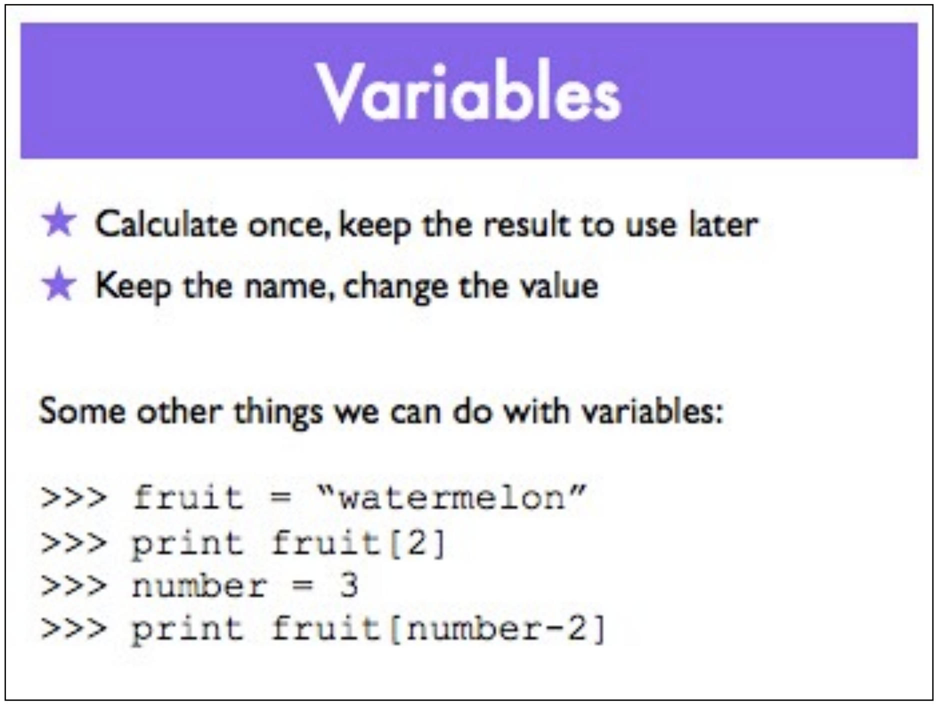
Variables

```
>>> donuts = 12 * 12
>>> color = "yellow"
```

Assign a new value:

```
>>> color = "red"
>>> donuts = 143

>>> color = "fish"
>>> color = 12
>>> color
12
```

Once you have a variable, you can change its value to anything else

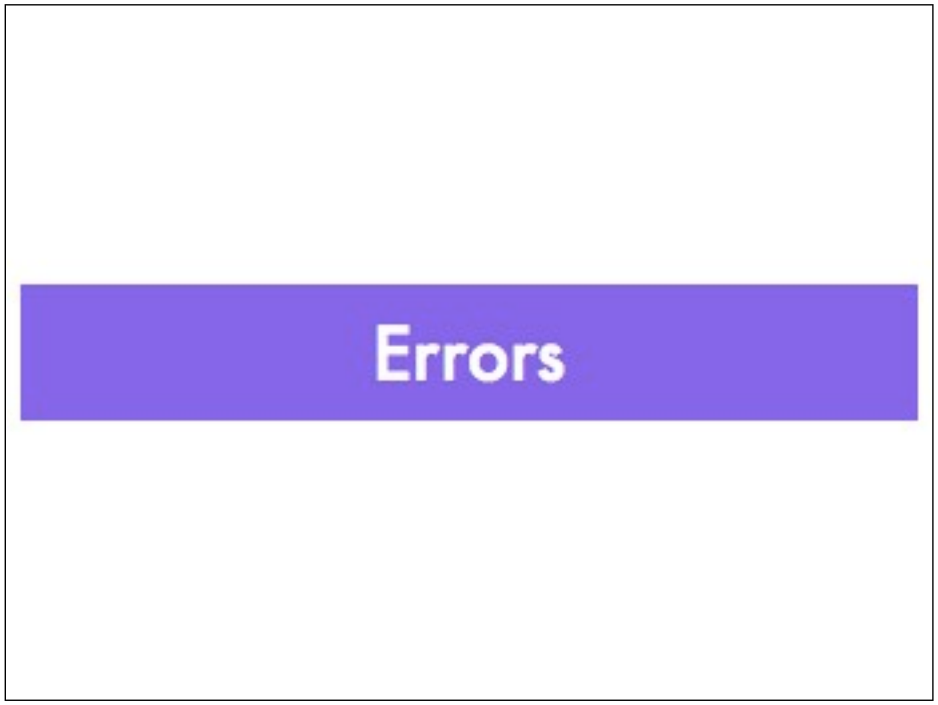

Variables

- ★ Calculate once, keep the result to use later
- ★ Keep the name, change the value

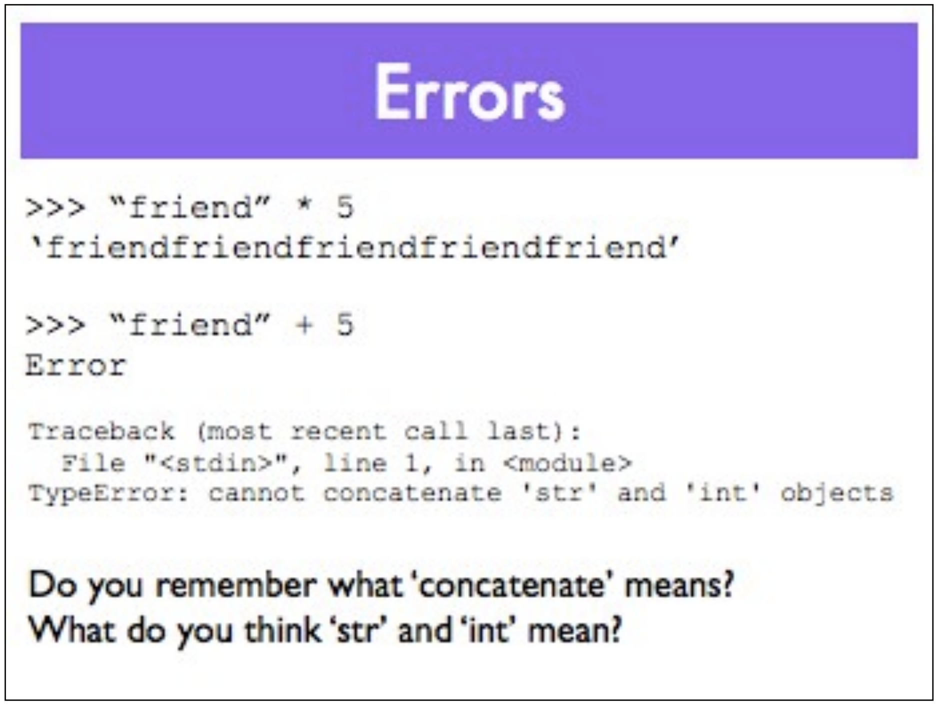
Some other things we can do with variables:

```
>>> fruit = "watermelon"
>>> print fruit[2]
>>> number = 3
>>> print fruit[number-2]
```

Demonstrates using a variable as a value in mathematical operations


Errors

“Errors are our friends – errors tell us what went wrong. Without error messages, it’s hard to fix something that’s broken.”

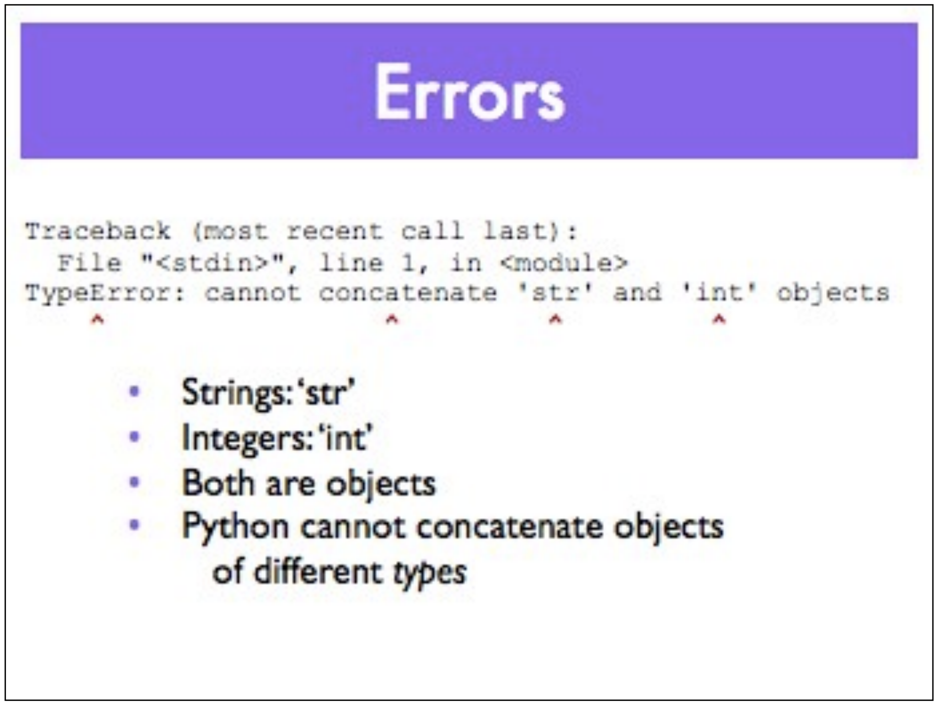


Errors

```
>>> "friend" * 5
'friendfriendfriendfriendfriend'

>>> "friend" + 5
Error

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
```

Do you remember what 'concatenate' means?
What do you think 'str' and 'int' mean?

Python calls these pieces of data "objects"

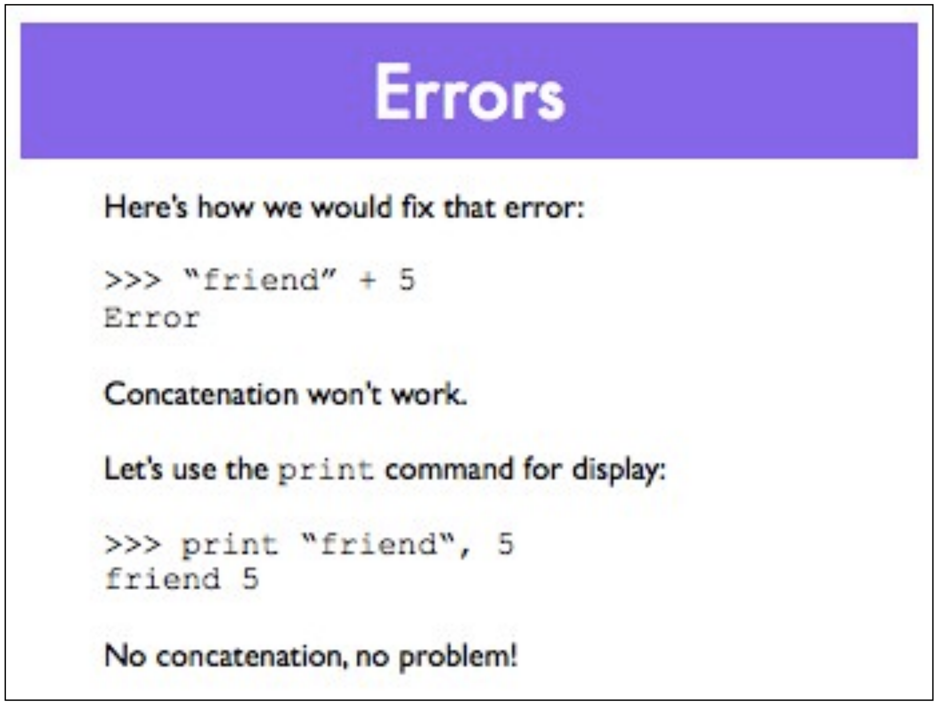


Errors

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
    ^                ^         ^         ^
```

- Strings: 'str'
- Integers: 'int'
- Both are objects
- Python cannot concatenate objects of different *types*

"We'll go into some more detail about objects and types in a few minutes."



Errors

Here's how we would fix that error:

```
>>> "friend" + 5
Error
```

Concatenation won't work.

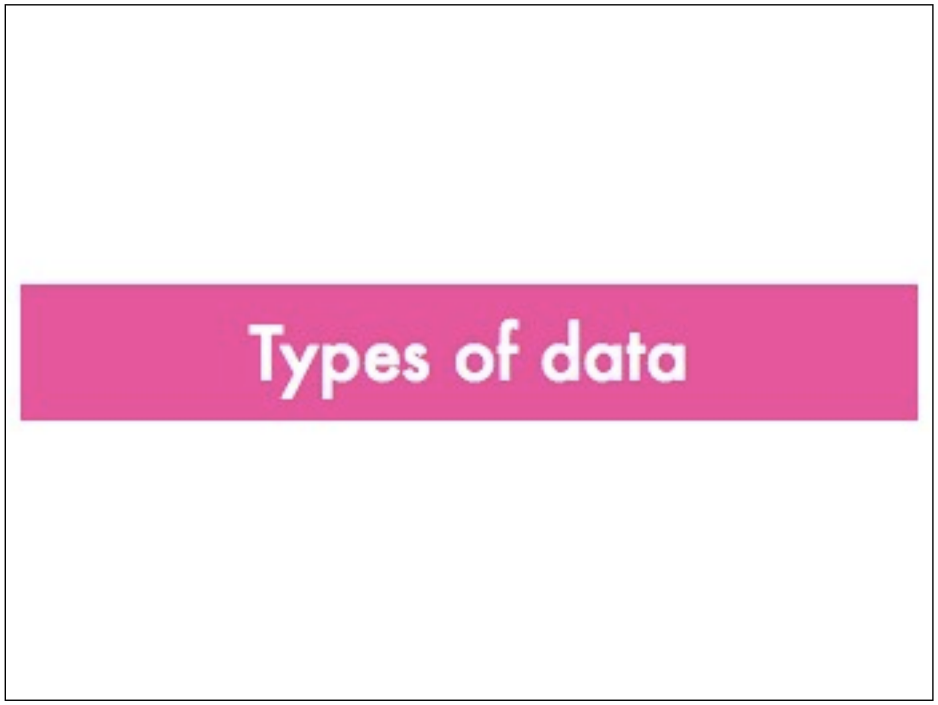
Let's use the `print` command for display:

```
>>> print "friend", 5
friend 5
```

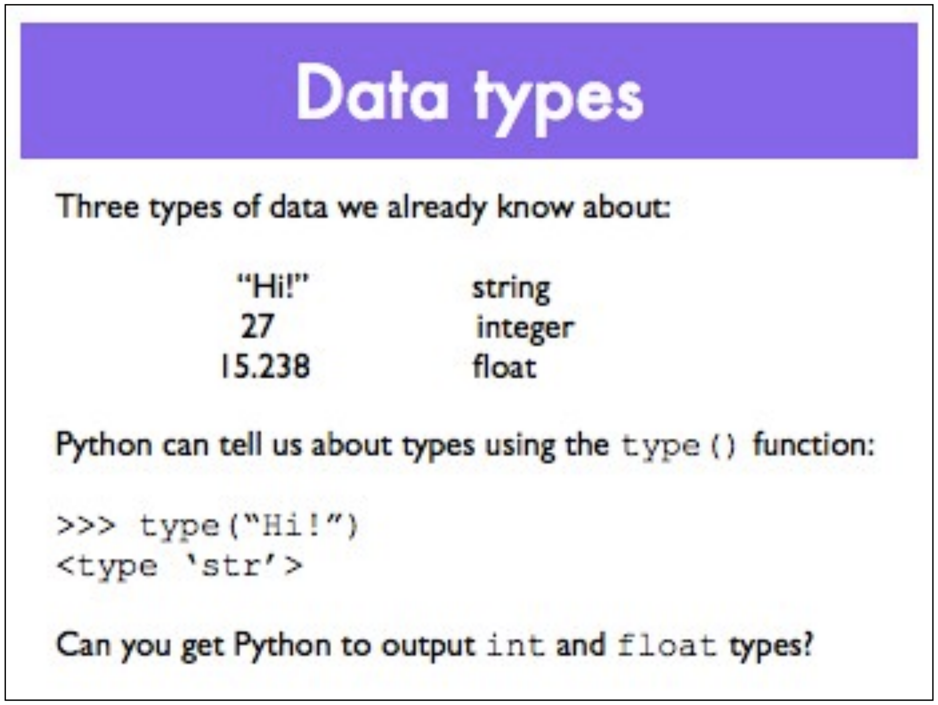
No concatenation, no problem!

"Here's how we would fix that kind of error."

(This is where we introduce the `print` command, but we do talk about it a little more later)



Types of data



Data types

Three types of data we already know about:

"Hi!"	string
27	integer
15.238	float

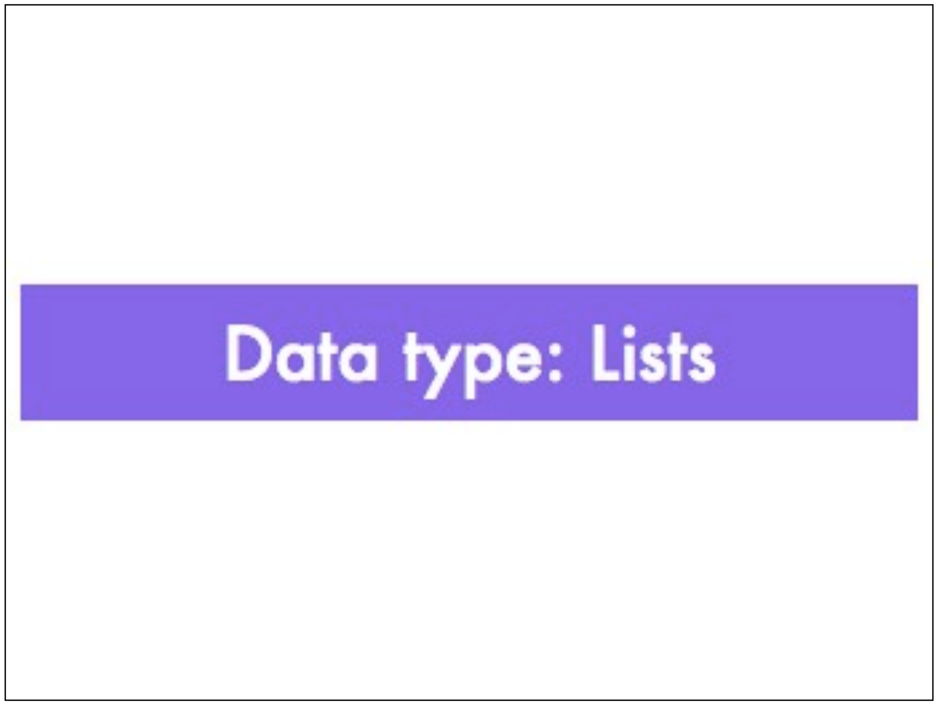
Python can tell us about types using the `type()` function:

```
>>> type("Hi!")
<type 'str'>
```

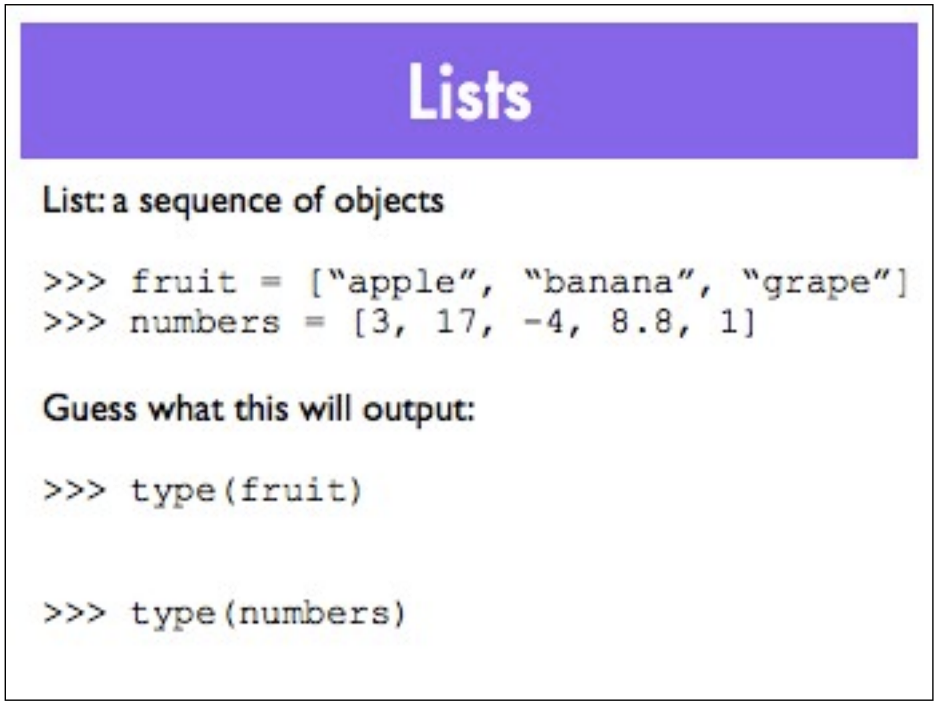
Can you get Python to output `int` and `float` types?

"`type()` is a built-in function that comes with Python."

"Python has a lot of built-in functions, which we'll learn about later."



Data type: Lists



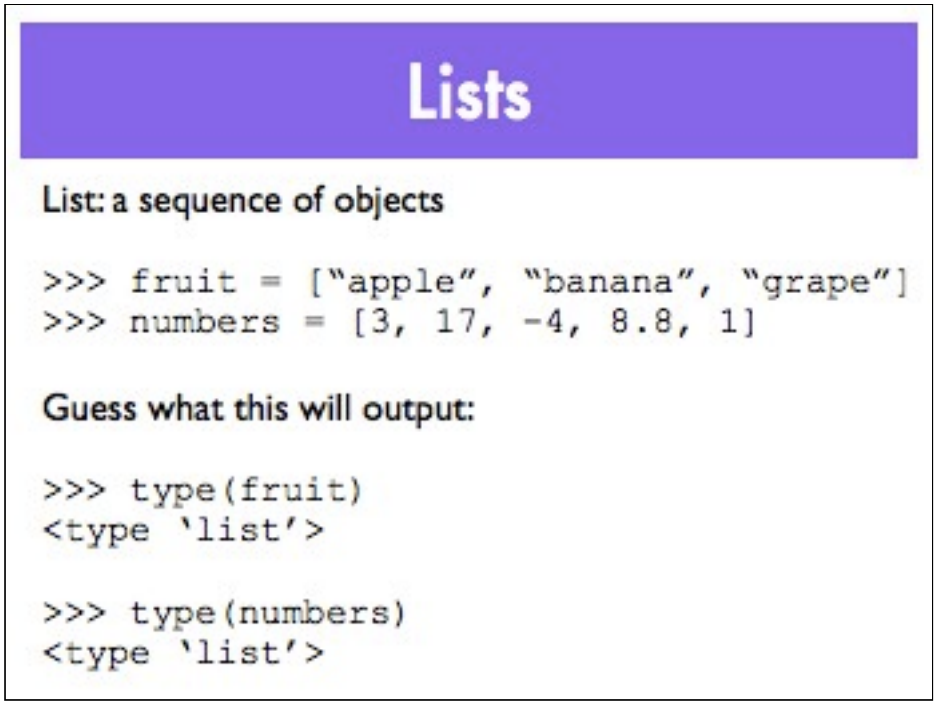
Lists

List: a sequence of objects

```
>>> fruit = ["apple", "banana", "grape"]
>>> numbers = [3, 17, -4, 8.8, 1]
```

Guess what this will output:

```
>>> type(fruit)


>>> type(numbers)
```



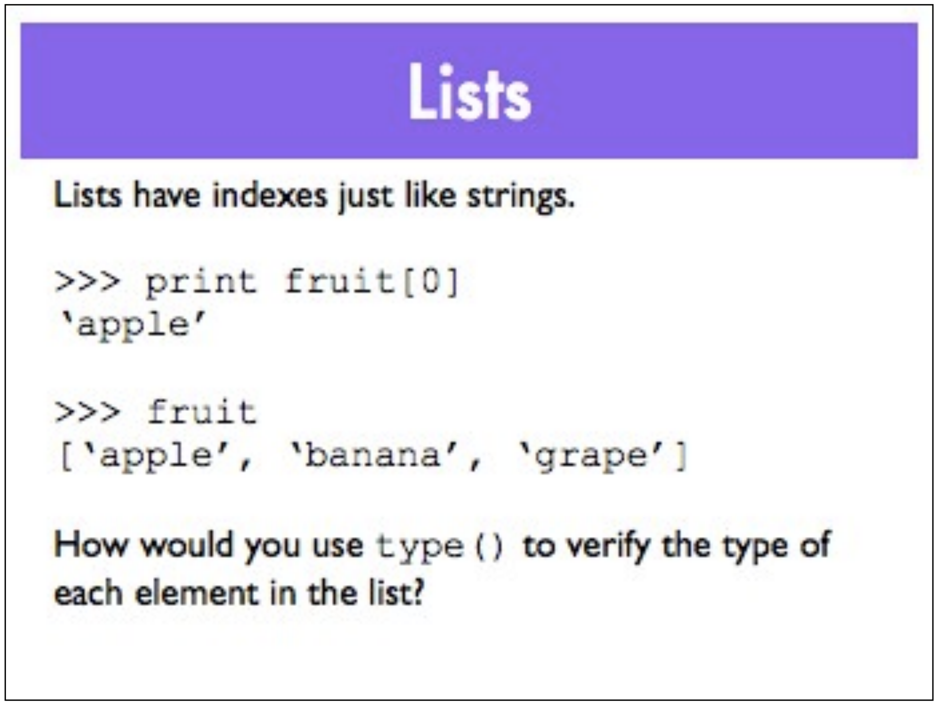
Lists

List: a sequence of objects

```
>>> fruit = ["apple", "banana", "grape"]
>>> numbers = [3, 17, -4, 8.8, 1]
```

Guess what this will output:

```
>>> type(fruit)
<type 'list'>

>>> type(numbers)
<type 'list'>
```



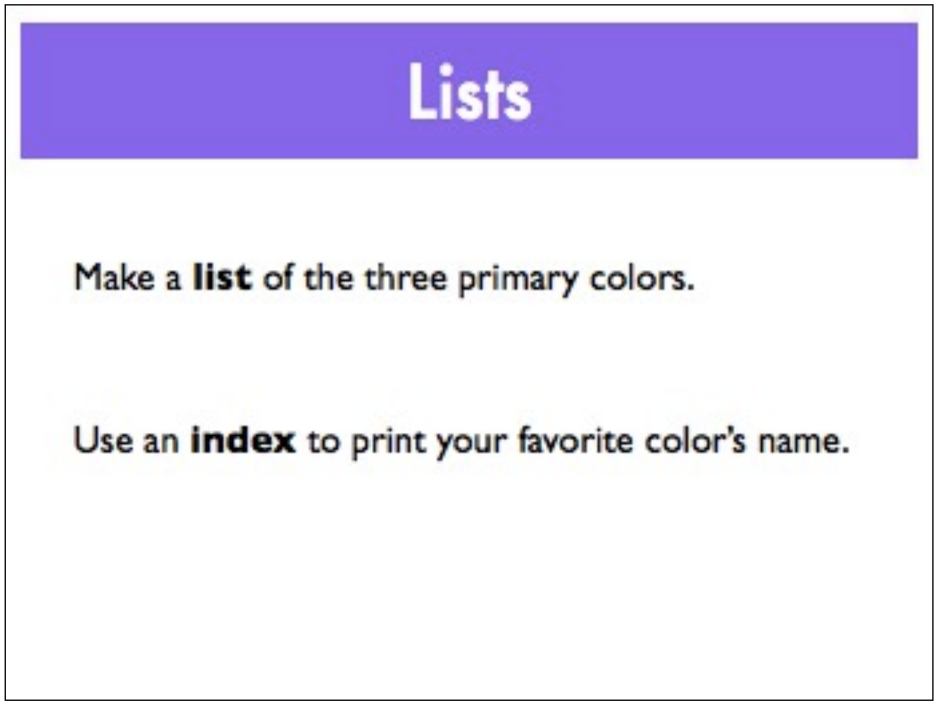
Lists

Lists have indexes just like strings.

```
>>> print fruit[0]
'apple'

>>> fruit
['apple', 'banana', 'grape']
```

How would you use `type()` to verify the type of each element in the list?

```
>>> type(fruit[0])
>>> type(fruit[1])
>>> type(fruit[2])
```

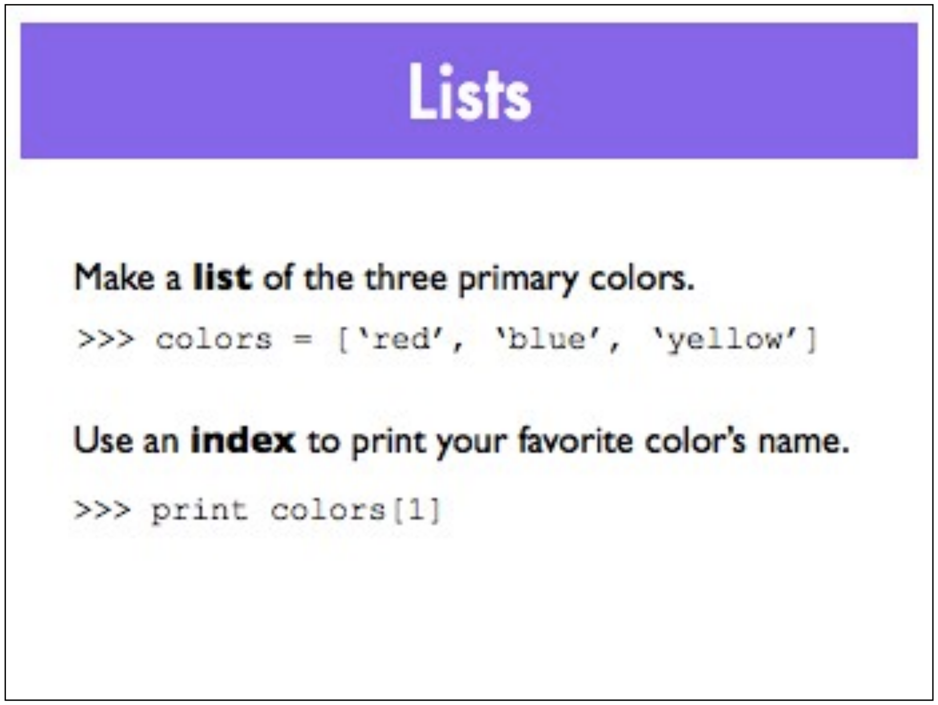


Lists

Make a **list** of the three primary colors.

Use an **index** to print your favorite color's name.

Give the students a few minutes to work out the solution for themselves, then switch over to Idle and demonstrate the solution (as seen in the next slide).



Lists

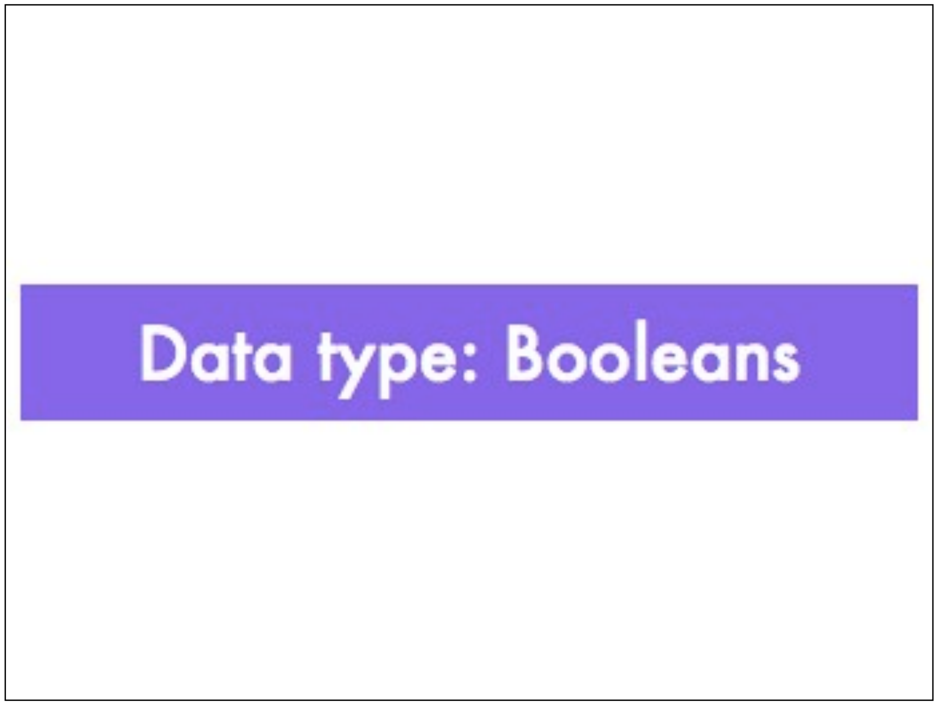
Make a **list** of the three primary colors.

```
>>> colors = ['red', 'blue', 'yellow']
```

Use an **index** to print your favorite color's name.

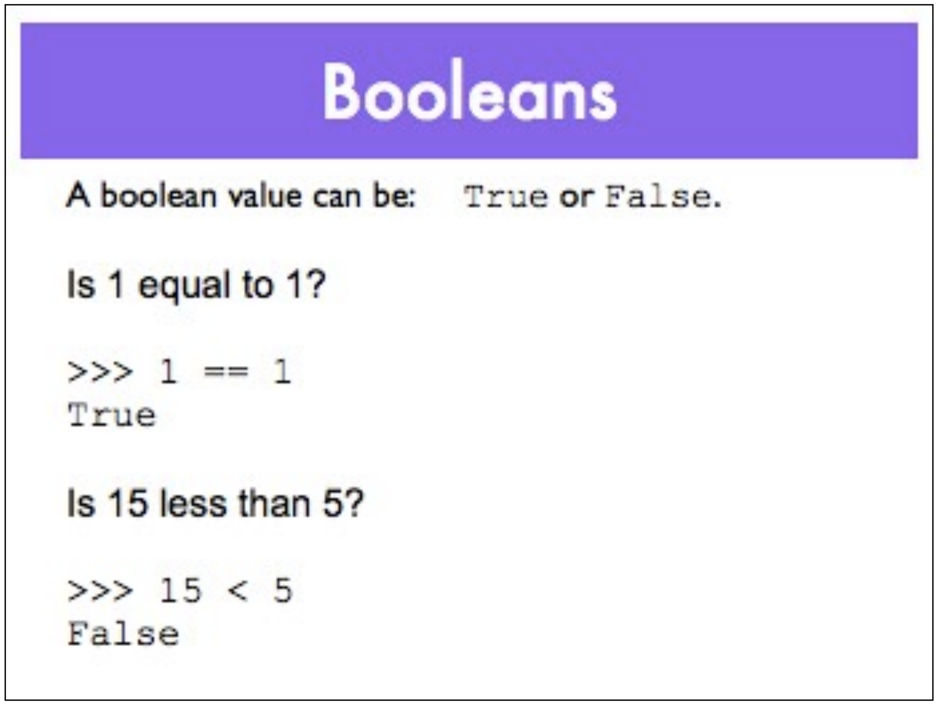
```
>>> print colors[1]
```

If there's time and interest, this is a good place to switch over to Idle and demonstrate .append() to add items to the list.



Data type: Booleans

This was one of the more difficult sections to explain, particularly to younger kids. If you can find a way to simplify this, please do. :)



Booleans

A boolean value can be: `True` or `False`.

Is 1 equal to 1?

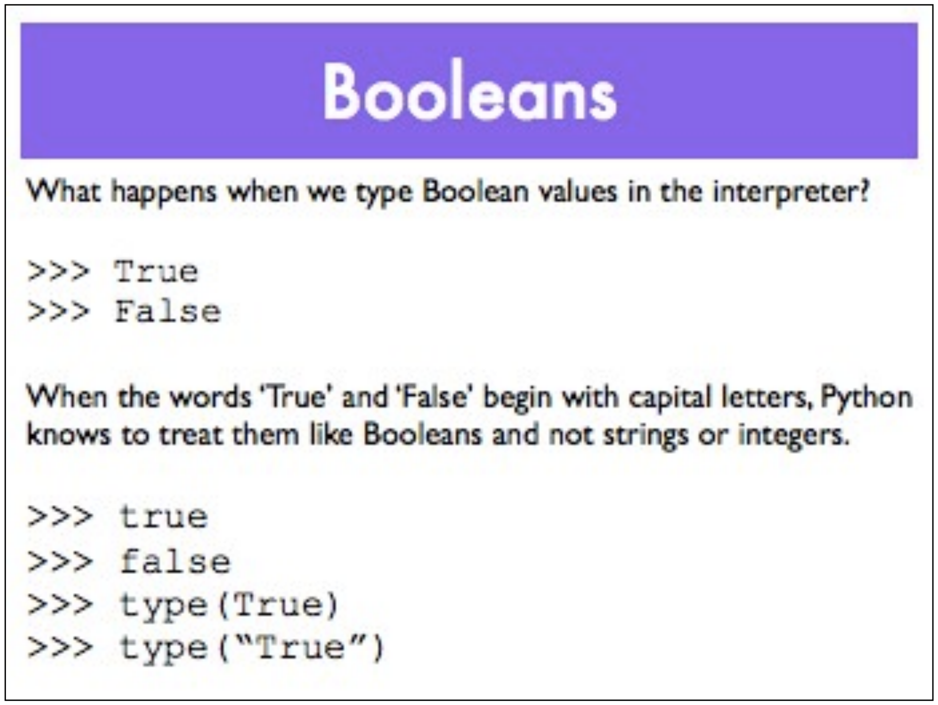
```
>>> 1 == 1
True
```

Is 15 less than 5?

```
>>> 15 < 5
False
```

"'Booleans' are a pretty simple idea, but we use them in programming a lot to make decisions in our code."

"If something is True, do this; if something is False, do something else"



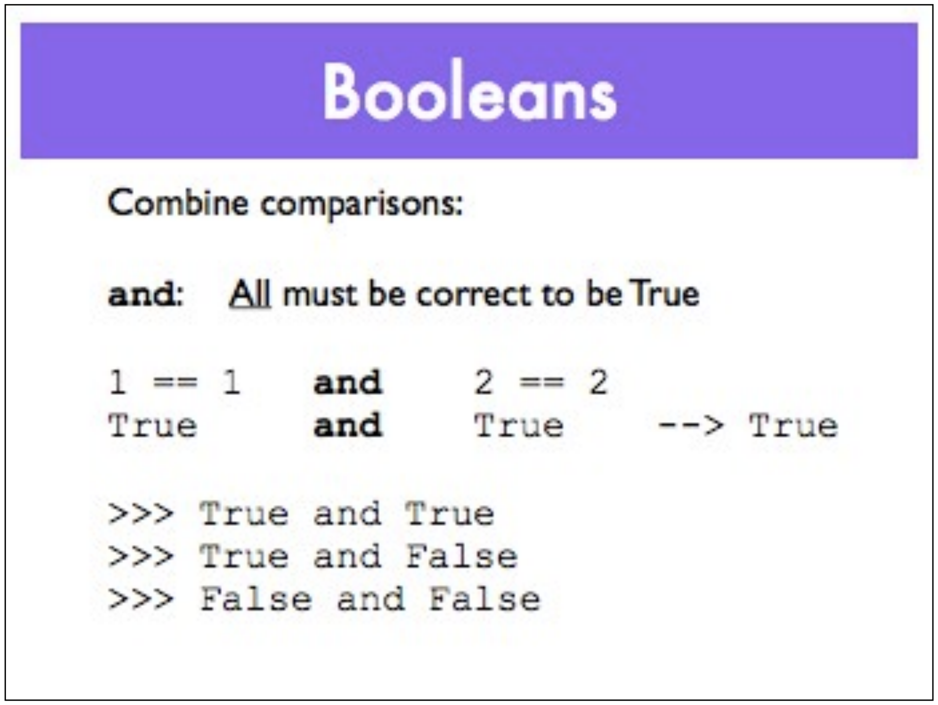
Booleans

What happens when we type Boolean values in the interpreter?

```
>>> True
>>> False
```

When the words 'True' and 'False' begin with capital letters, Python knows to treat them like Booleans and not strings or integers.

```
>>> true
>>> false
>>> type(True)
>>> type("True")
```



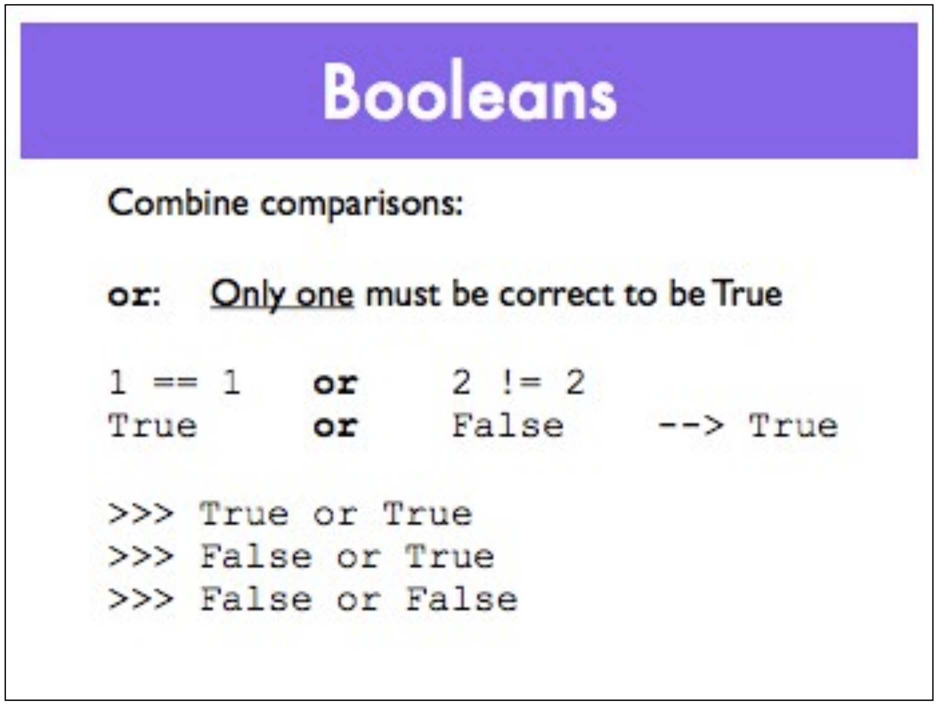
Booleans

Combine comparisons:

and: All must be correct to be True

```
1 == 1    and    2 == 2
True      and    True      --> True

>>> True and True
>>> True and False
>>> False and False
```

Here's where we can start using some of those "comparison operators" we talked about earlier, when we learned about math.

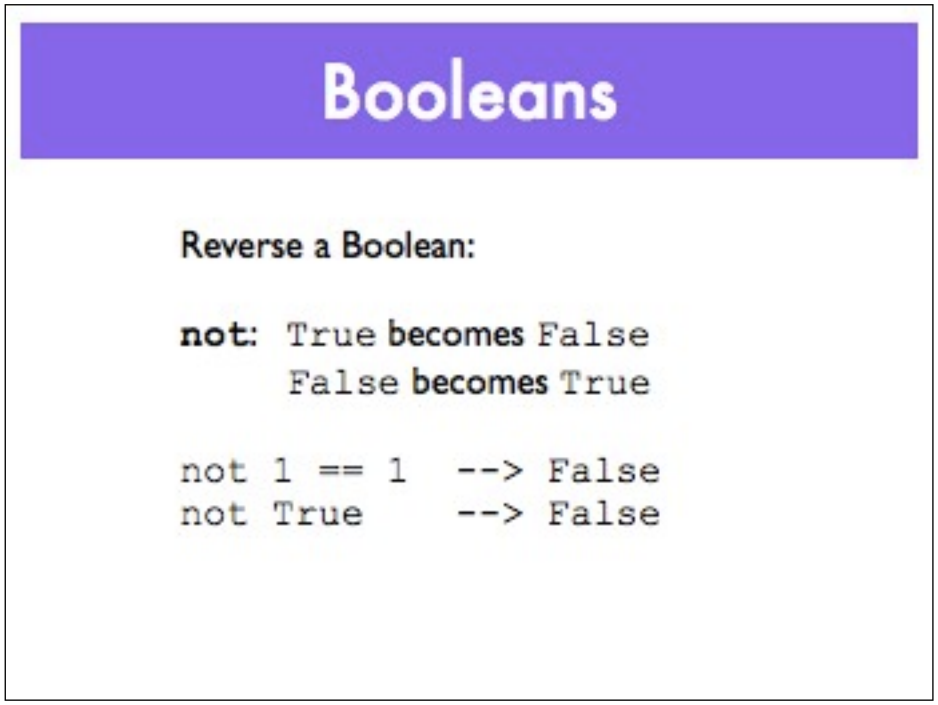


Booleans

Combine comparisons:

or: Only one must be correct to be True

```
1 == 1    or    2 != 2
True      or    False     --> True

>>> True or True
>>> False or True
>>> False or False
```

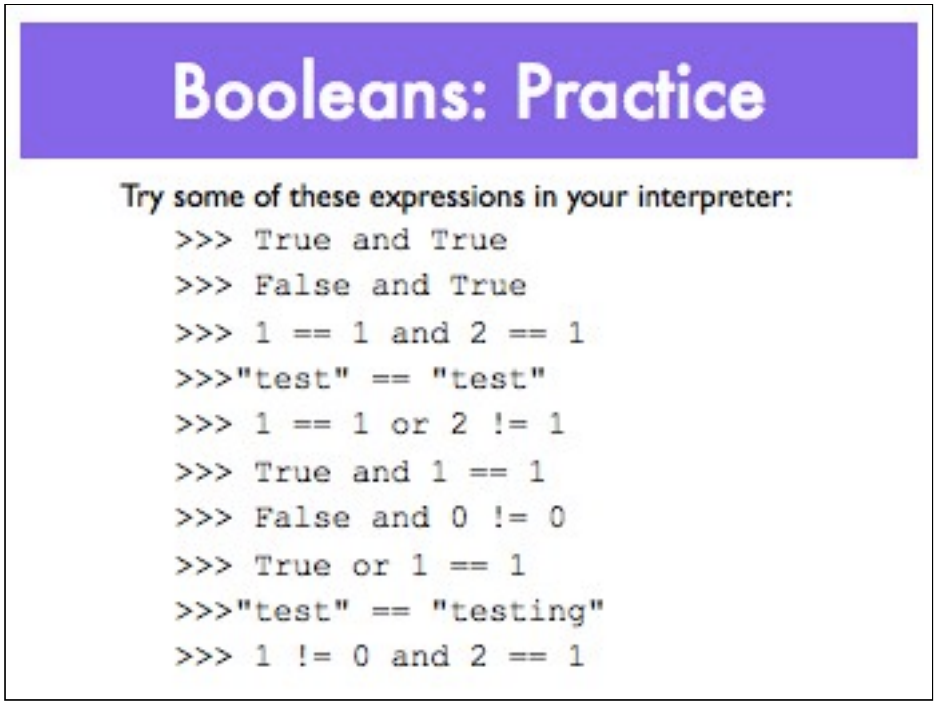


Booleans

Reverse a Boolean:

not: `True` **becomes** `False`
`False` **becomes** `True`

```
not 1 == 1   --> False
not True     --> False
```

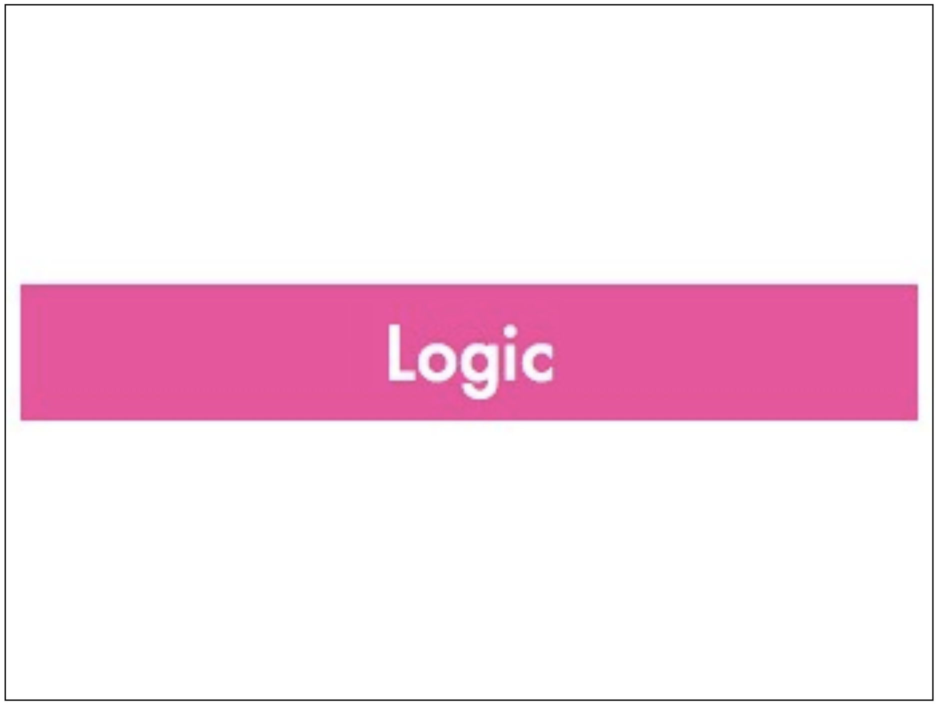


Booleans: Practice

Try some of these expressions in your interpreter:

```
>>> True and True
>>> False and True
>>> 1 == 1 and 2 == 1
>>>"test" == "test"
>>> 1 == 1 or 2 != 1
>>> True and 1 == 1
>>> False and 0 != 0
>>> True or 1 == 1
>>>"test" == "testing"
>>> 1 != 0 and 2 == 1
```

In case we want to send them online later: http://bit.ly/boolean-practice

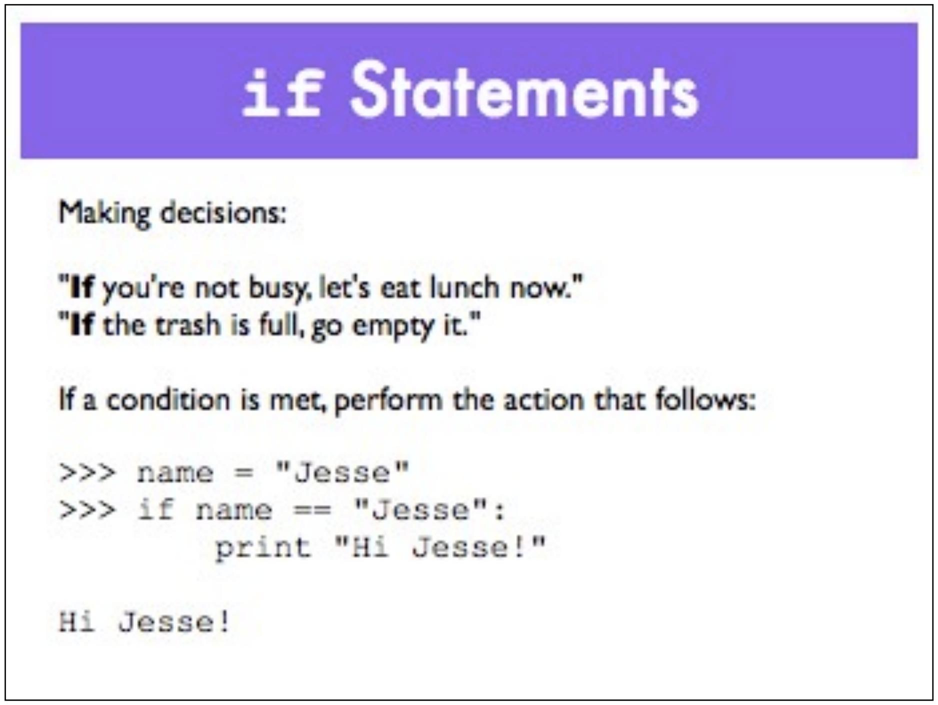


Logic

`if` Statements

When we talk about logic, we're talking about making decisions about what to do next in our code.

One of the ways we do that is with "if statements".



if Statements

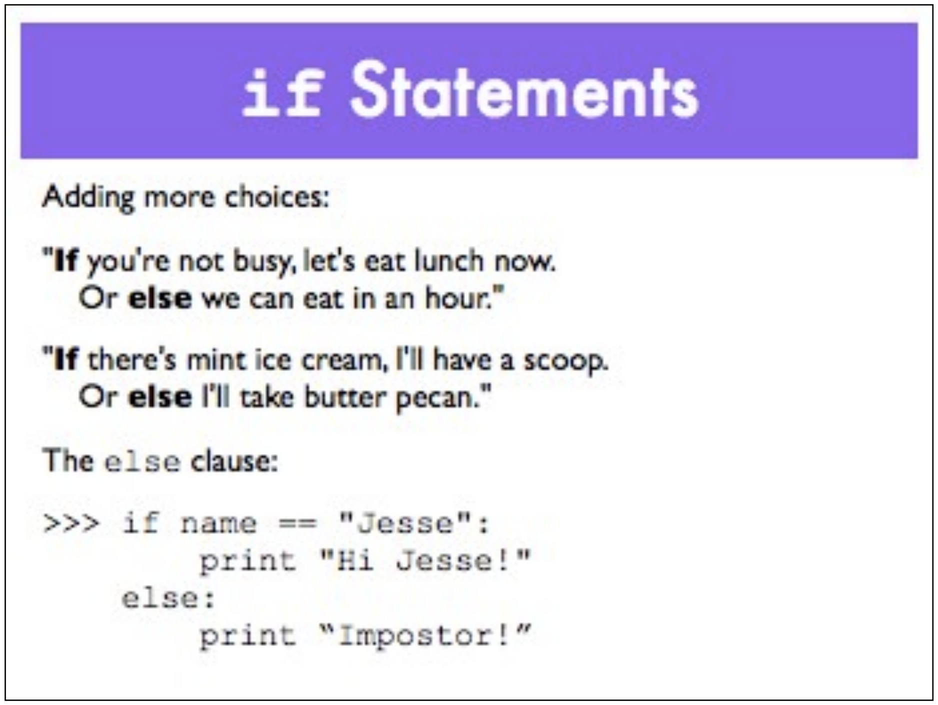
Making decisions:

"**If** you're not busy, let's eat lunch now."
"**If** the trash is full, go empty it."

If a condition is met, perform the action that follows:

```
>>> name = "Jesse"
>>> if name == "Jesse":
        print "Hi Jesse!"

Hi Jesse!
```

Mention indentation here



if Statements

Adding more choices:

"**If** you're not busy, let's eat lunch now.
Or **else** we can eat in an hour."

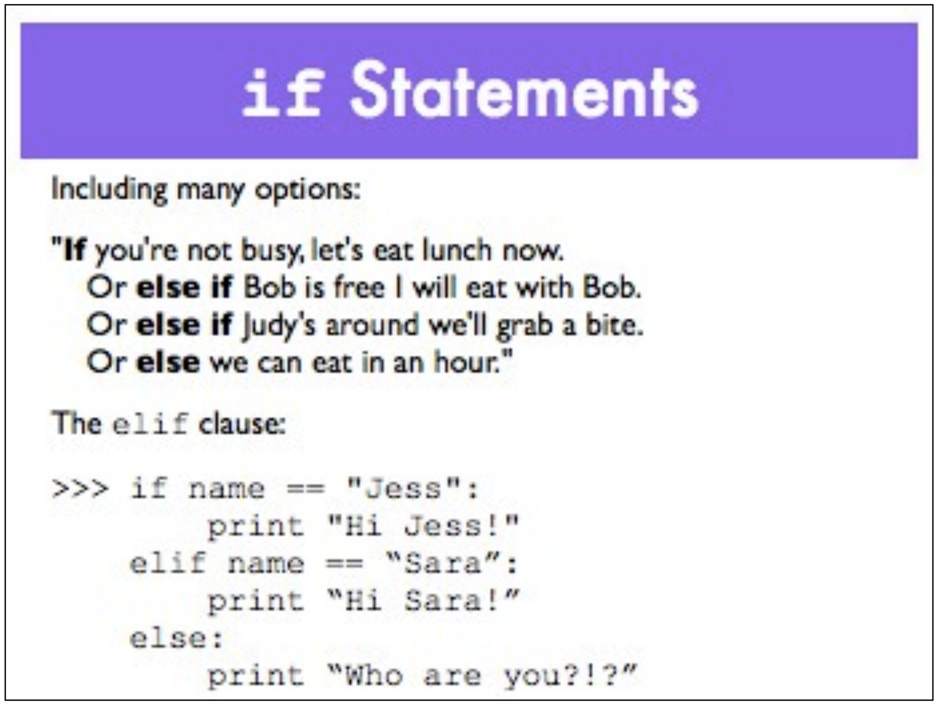
"**If** there's mint ice cream, I'll have a scoop.
Or **else** I'll take butter pecan."

The `else` clause:

```
>>> if name == "Jesse":
        print "Hi Jesse!"
    else:
        print "Impostor!"
```

Another example from the demo:

IF the peanut butter jar has paper, take it off
ELSE scoop out the peanut butter

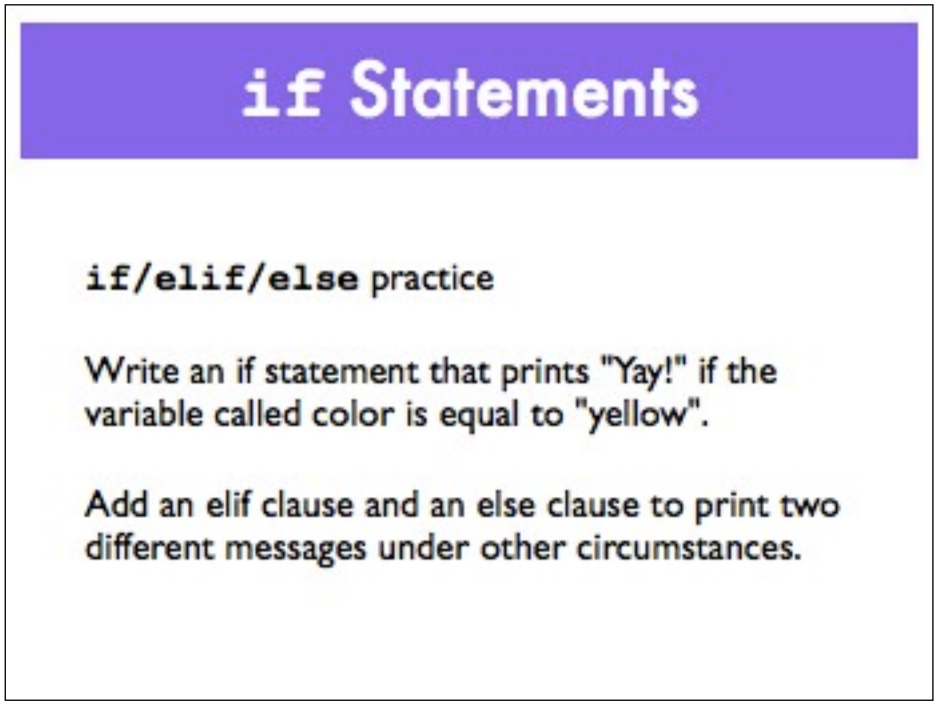

if Statements

Including many options:

"**If** you're not busy, let's eat lunch now.
  Or **else if** Bob is free I will eat with Bob.
  Or **else if** Judy's around we'll grab a bite.
  Or **else** we can eat in an hour."

The `elif` clause:

```
>>> if name == "Jess":
        print "Hi Jess!"
    elif name == "Sara":
        print "Hi Sara!"
    else:
        print "Who are you?!?"
```


if Statements

`if/elif/else` practice

Write an if statement that prints "Yay!" if the variable called color is equal to "yellow".

Add an elif clause and an else clause to print two different messages under other circumstances.

```
color = "blue"
if color == "yellow":
        print "Yay!"
    elif name == "purple":
        print "Try again"
    else:
        print "We want yellow!"
```

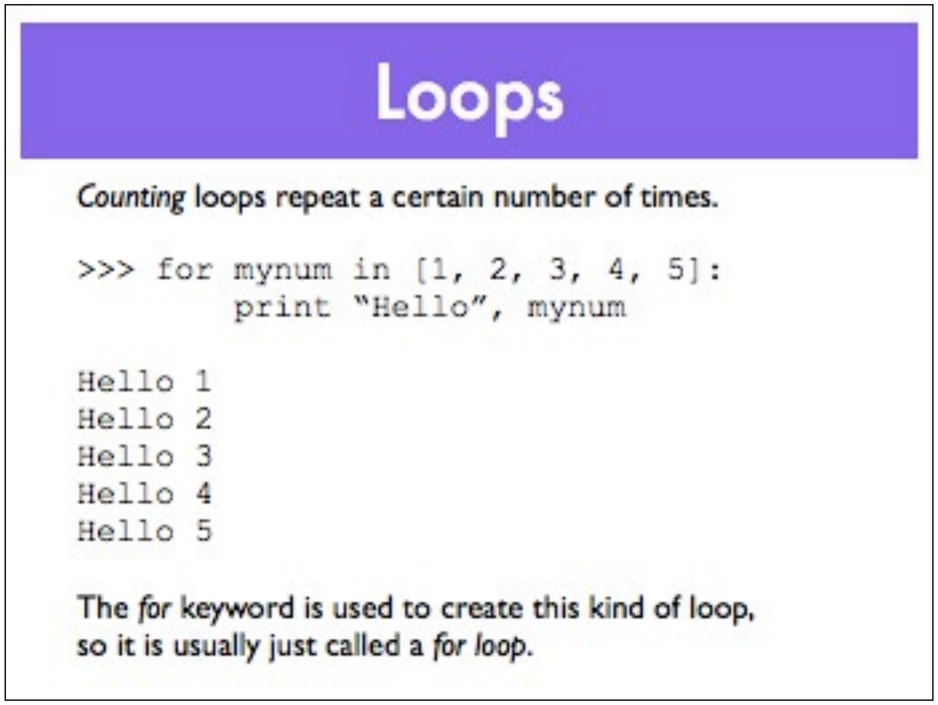
Loops

Loops

Loops are chunks of code
that repeat a task over and over again.

★*Counting* loops repeat a
certain number of times.

★*Conditional* loops keep
going until a certain thing happens
(or as long as some condition is True).



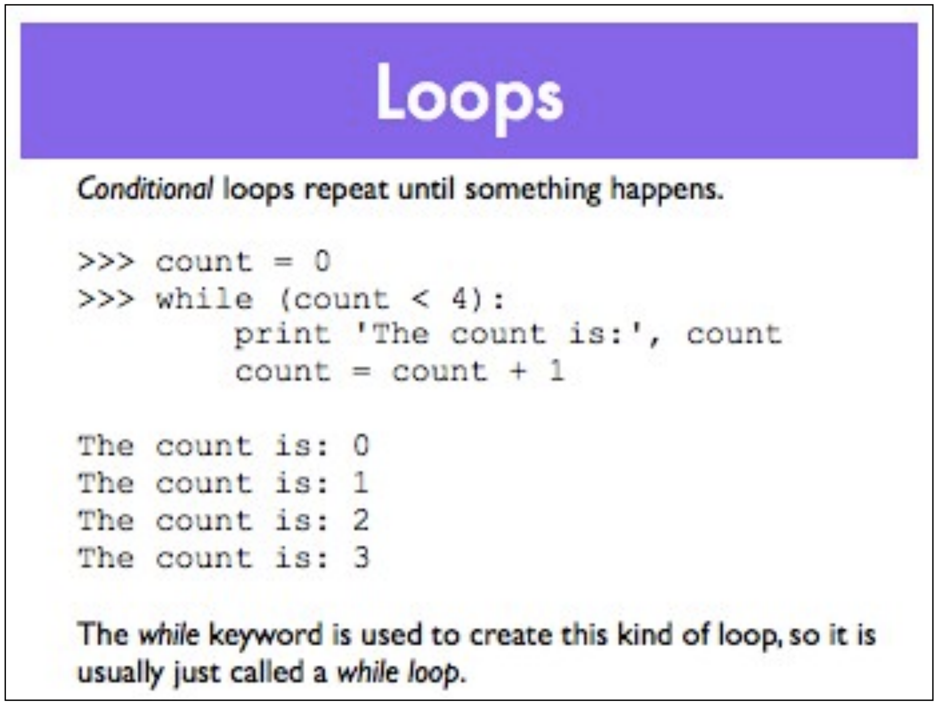
Loops

Counting loops repeat a certain number of times.

```
>>> for mynum in [1, 2, 3, 4, 5]:
        print "Hello", mynum

Hello 1
Hello 2
Hello 3
Hello 4
Hello 5
```

The *for* keyword is used to create this kind of loop, so it is usually just called a *for loop*.

Break down the “for” statement – for each element in the list



Loops

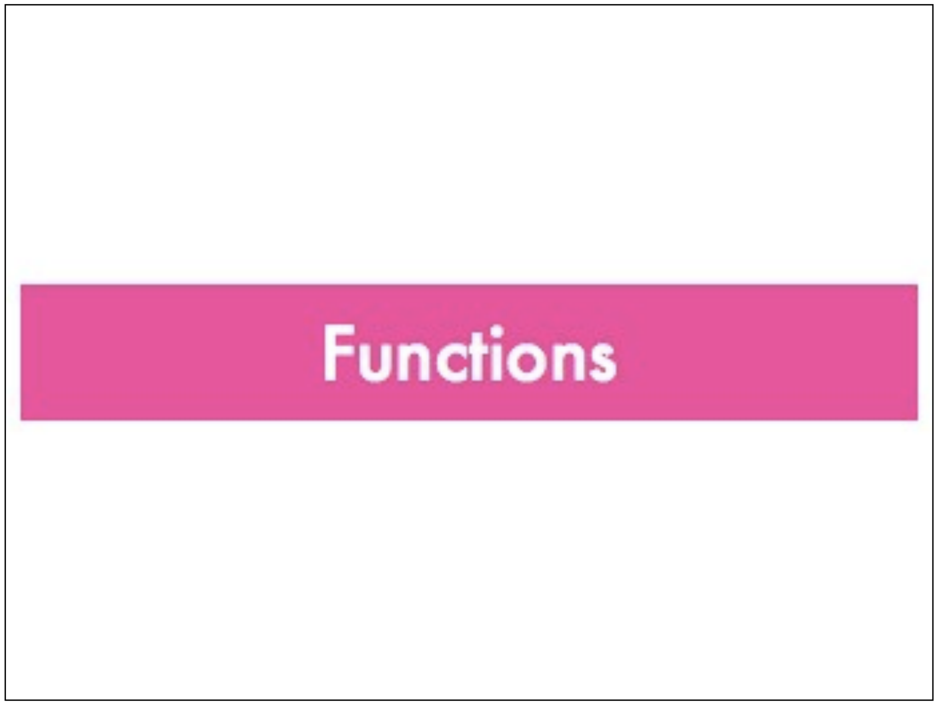
Conditional loops repeat until something happens.

```
>>> count = 0
>>> while (count < 4):
        print 'The count is:', count
        count = count + 1

The count is: 0
The count is: 1
The count is: 2
The count is: 3
```

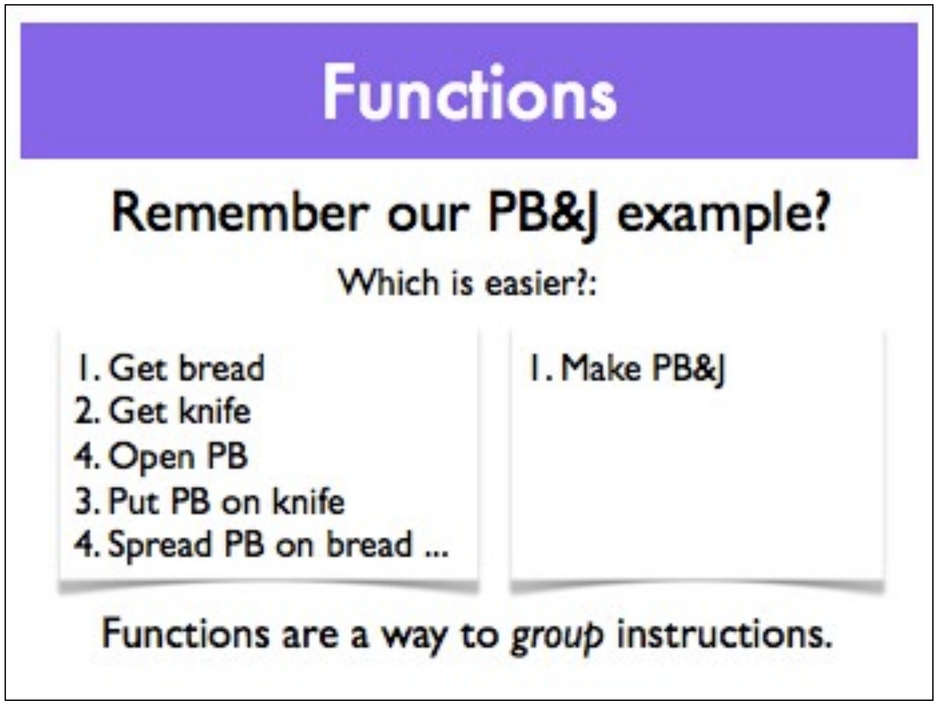
The *while* keyword is used to create this kind of loop, so it is usually just called a *while loop*.

Point out how the counter is increasing each time we go through the loop

Talk about infinite loops



Functions



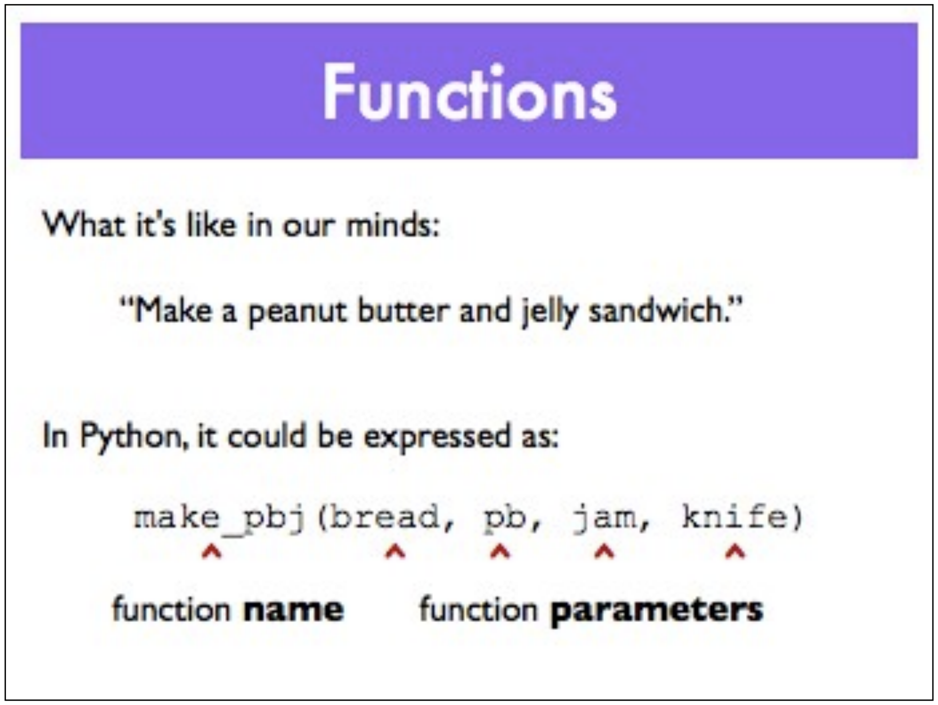
Functions

Remember our PB&J example?

Which is easier?:

1. Get bread 2. Get knife 4. Open PB 3. Put PB on knife 4. Spread PB on bread ...	1. Make PB&J

Functions are a way to *group* instructions.



Functions

What it's like in our minds:

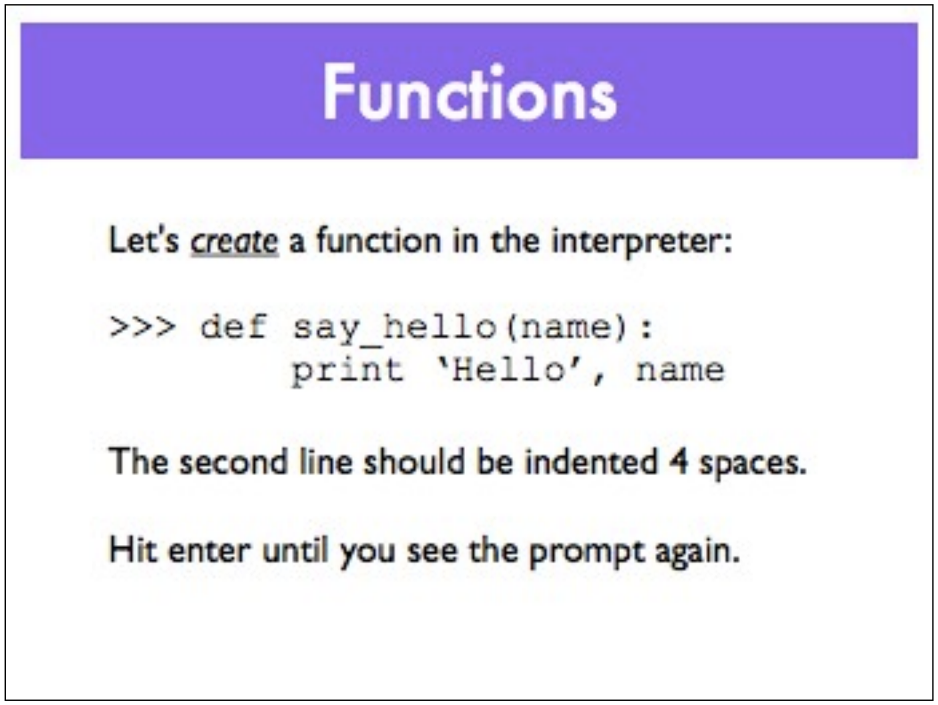
"Make a peanut butter and jelly sandwich."

In Python, it could be expressed as:

```
make_pbj(bread, pb, jam, knife)
    ^         ^    ^    ^     ^
```

function **name** function **parameters**

We all know how to make a PB&J, but we don't have to think about all the steps it takes every time, because the steps are grouped together in our minds as "make a peanut butter and jelly sandwich".



Functions

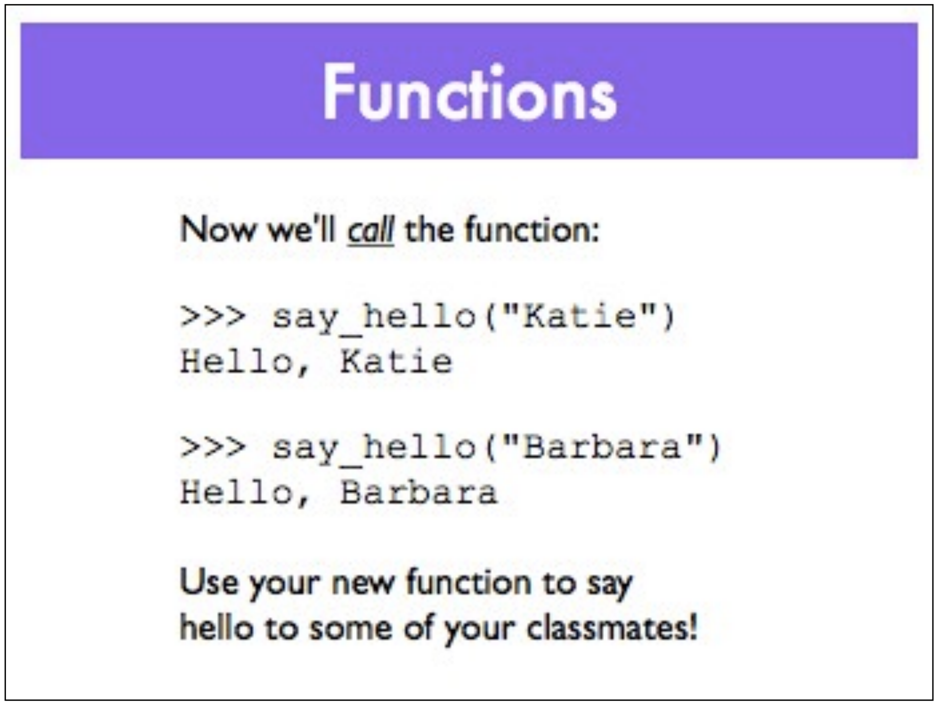
Let's *create* a function in the interpreter:

```
>>> def say_hello(name):
        print 'Hello', name
```

The second line should be indented 4 spaces.

Hit enter until you see the prompt again.

Reinforce the notion of indentation, make sure the students are checking that each time they enter a line.



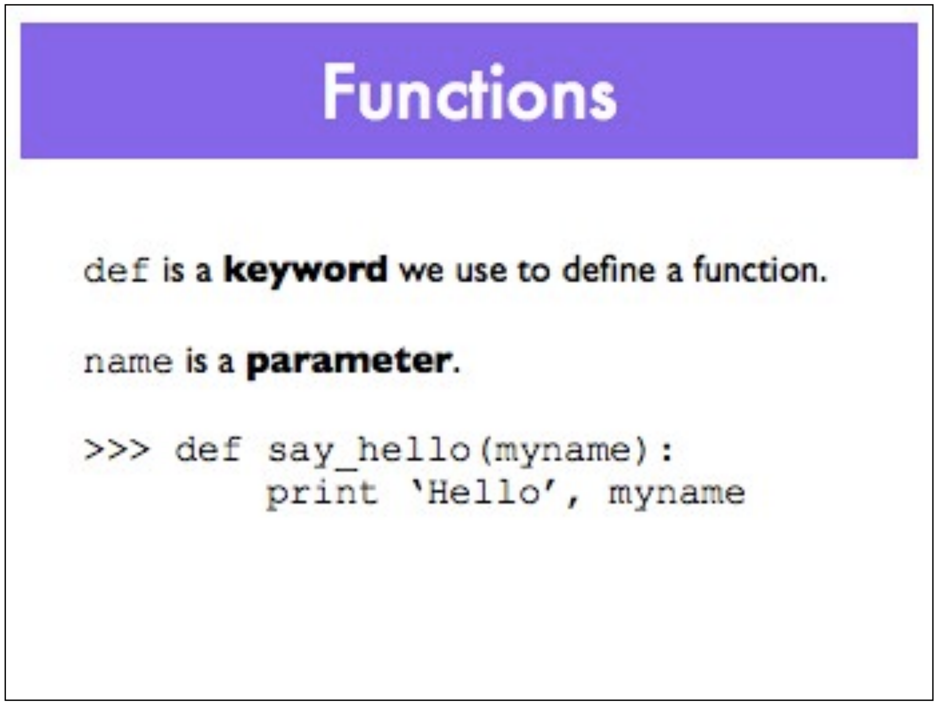
Functions

Now we'll *call* the function:

```
>>> say_hello("Katie")
Hello, Katie

>>> say_hello("Barbara")
Hello, Barbara
```

Use your new function to say hello to some of your classmates!

Emphasize the difference between **defining** and **calling** the function.



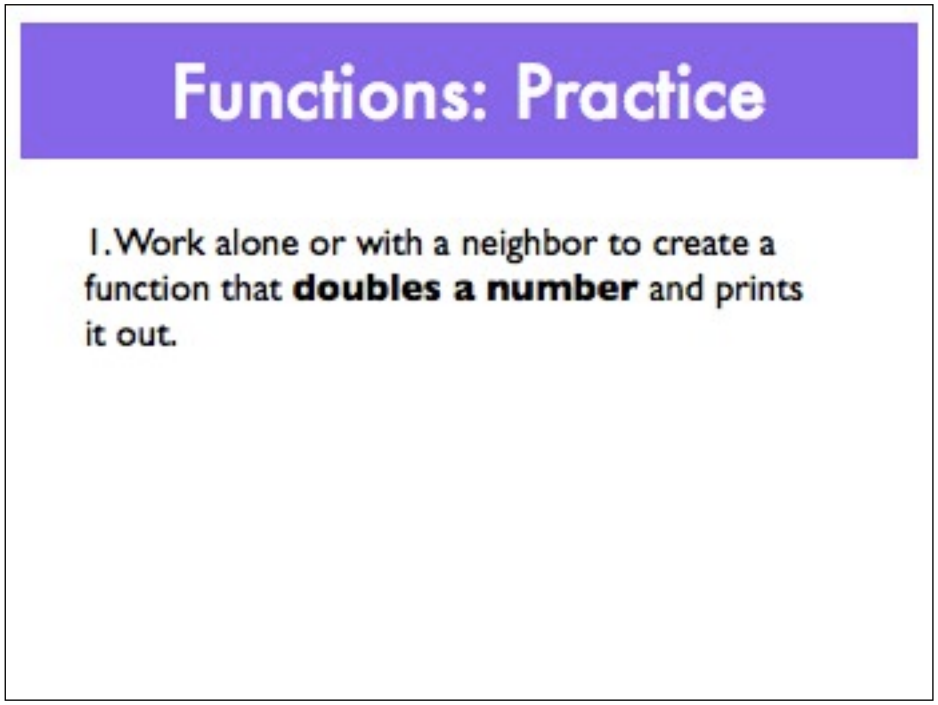
Functions

`def` is a **keyword** we use to define a function.

`name` is a **parameter**.

```
>>> def say_hello(myname):
        print 'Hello', myname
```

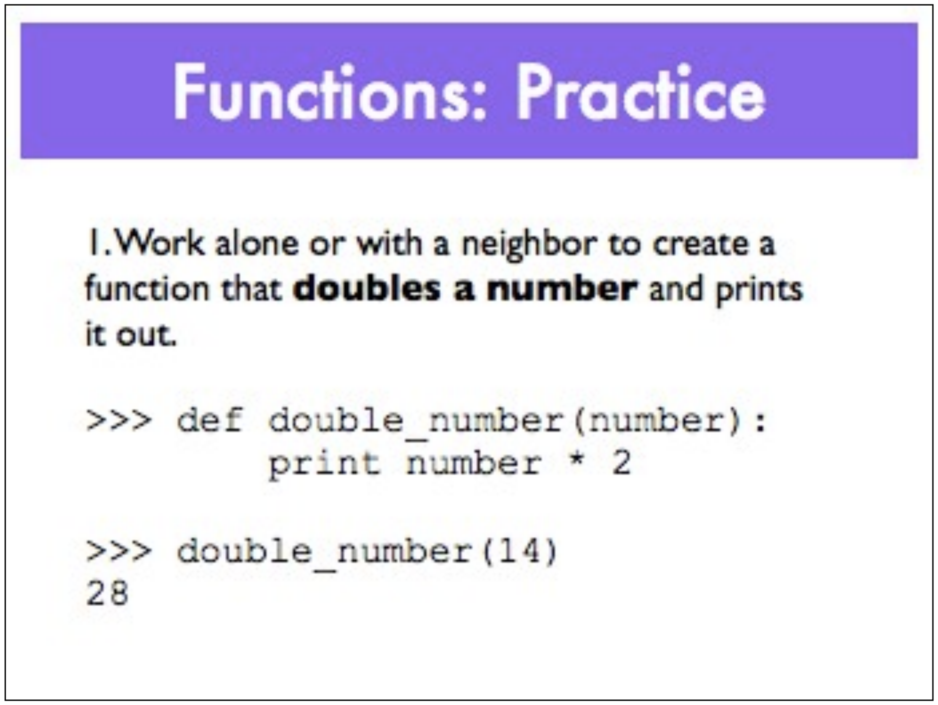
Emphasize that the ‘myname’ variable is a parameter, and that it can be anything, as long as it’s the same in the body of the function



Functions: Practice

1. Work alone or with a neighbor to create a function that **doubles a number** and prints it out.

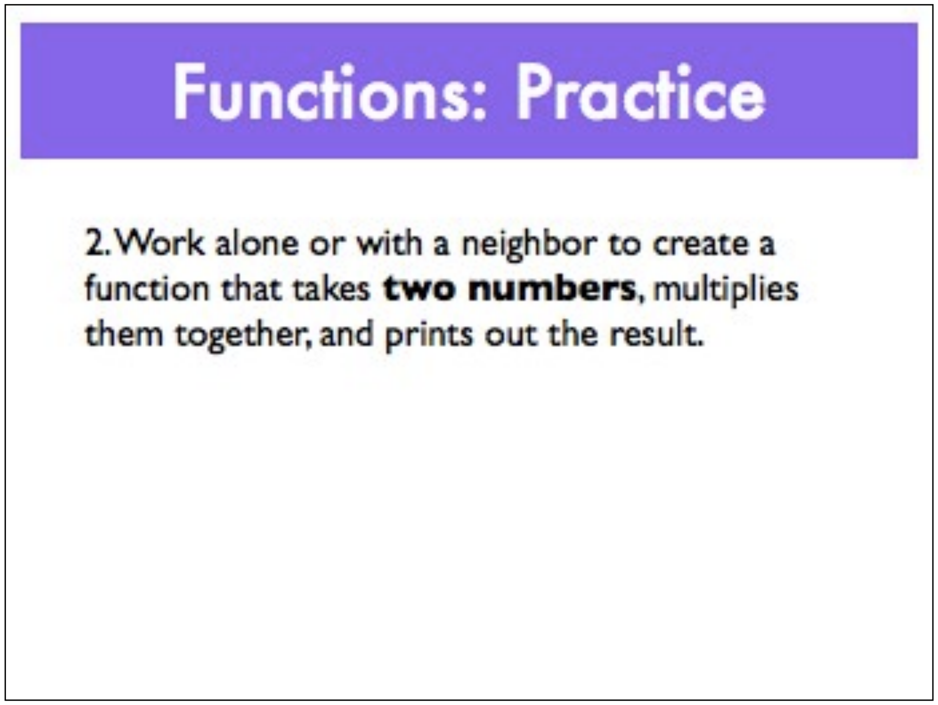
Give the students some time to work it out for themselves, then move to Idle and demonstrate the solution for them.



Functions: Practice

1. Work alone or with a neighbor to create a function that **doubles a number** and prints it out.

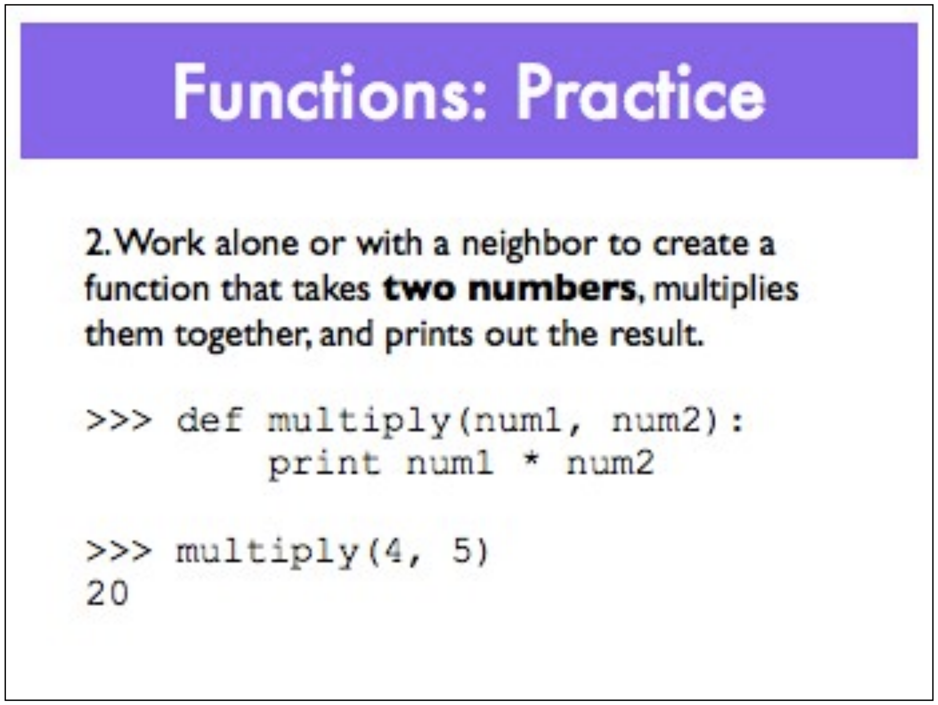
```
>>> def double_number(number):
        print number * 2

>>> double_number(14)
28
```



Functions: Practice

2. Work alone or with a neighbor to create a function that takes **two numbers**, multiplies them together, and prints out the result.

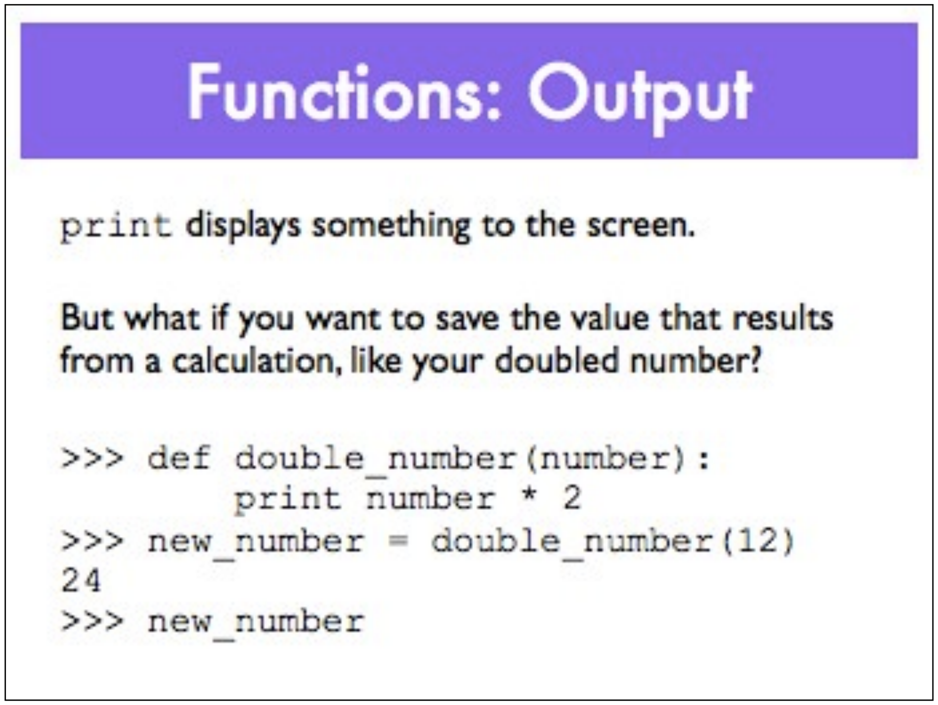
Give the students some time to work it out for themselves, then



Functions: Practice

2. Work alone or with a neighbor to create a function that takes **two numbers**, multiplies them together, and prints out the result.

```
>>> def multiply(num1, num2):
        print num1 * num2

>>> multiply(4, 5)
20
```



Functions: Output

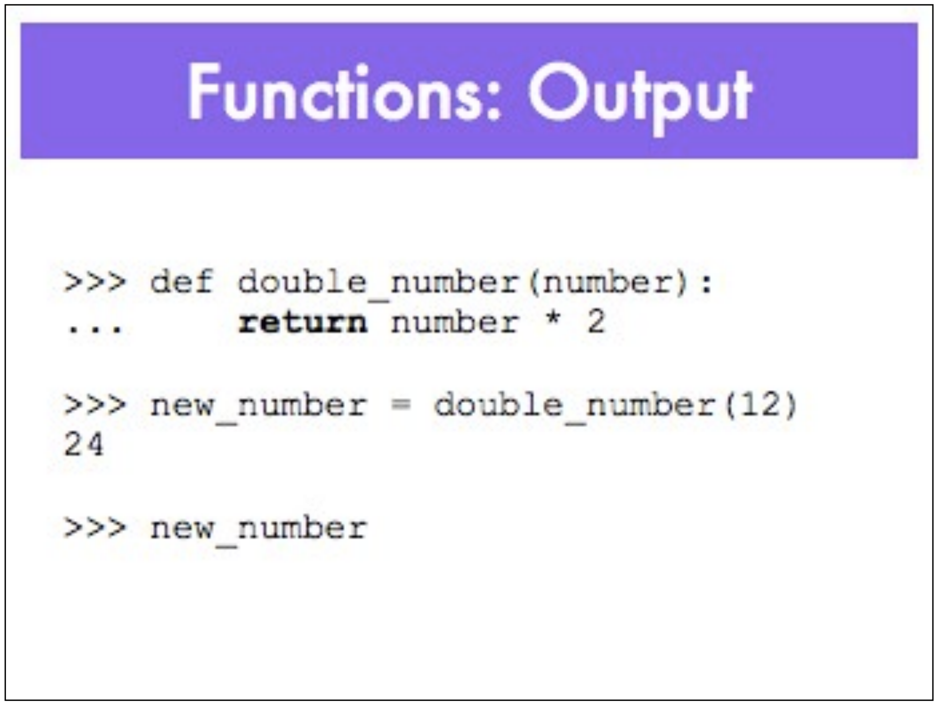
`print` displays something to the screen.

But what if you want to save the value that results from a calculation, like your doubled number?

```
>>> def double_number(number):
        print number * 2
>>> new_number = double_number(12)
24
>>> new_number
```

The first time you give 'new_number' a value, it will return that value (24).

But the next time you enter 'new_number', it doesn't have that value (24) anymore – the value isn't saved.



Functions: Output

```
>>> def double_number(number):
...     return number * 2

>>> new_number = double_number(12)
24

>>> new_number
```

This time when you give 'new_number' a value, it will return that value (24), and now the value is saved.

When you type 'new_number' again, you'll see the same value (24) until you decide to change it.

Functions

Rules:

- ★ Functions are **defined** using **`def`**.
- ★ Functions are **called** using **parentheses**.
- ★ Functions take **parameters** and can return **outputs**.
- ★ `print` **displays** information, but does not give a value
- ★ `return` gives a **value** to the caller (you!)

Input

Input

Input is information that we enter into a function so that we can do something with it.

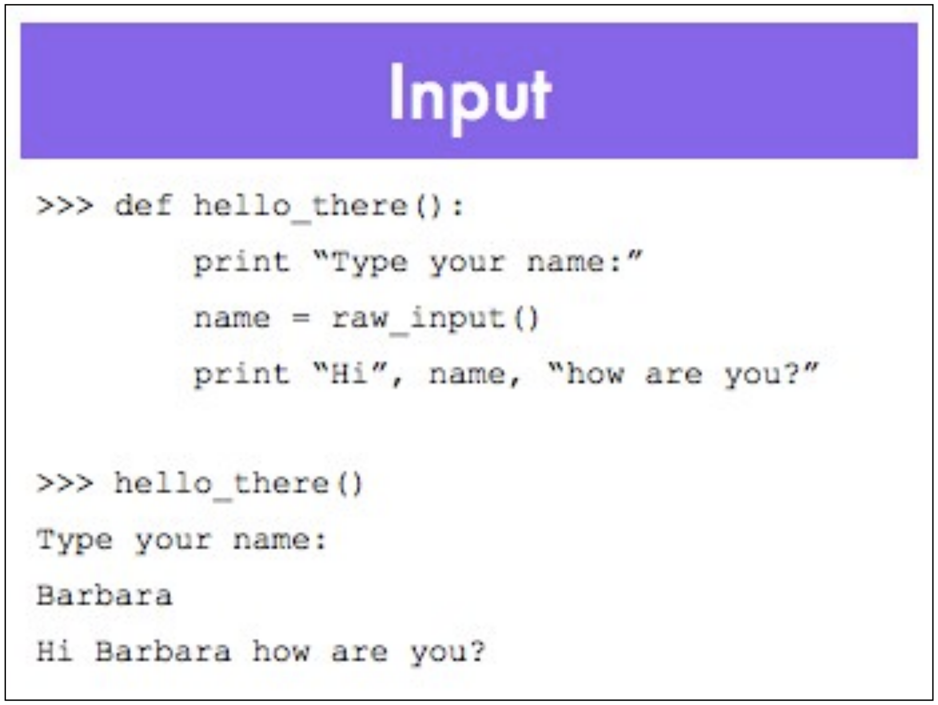
```
>>> def hello_there(name):
        print "Hello there", name
>>> hello_there("Katie")
"Hello there Katie"
```

But what if you want to enter a different name? Or let another user enter a name?

Input

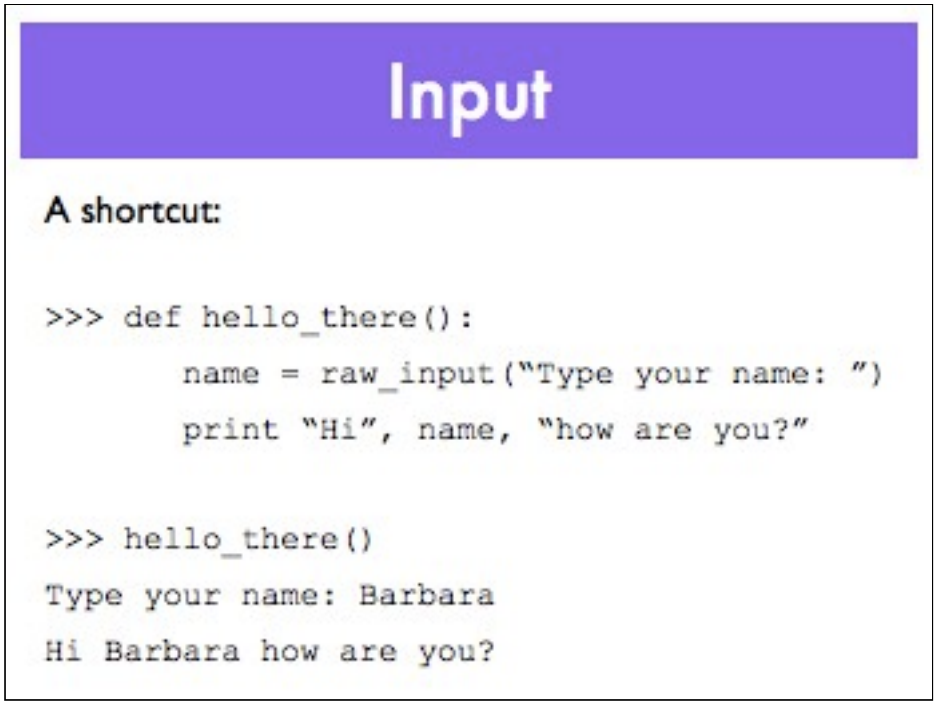
The `raw_input()` function takes *input* from the user - you give that input to the function by typing it.

```
>>> def hello_there():
        print "Type your name:"
        name = raw_input()
        print "Hi", name, "how are you?"
```



Input

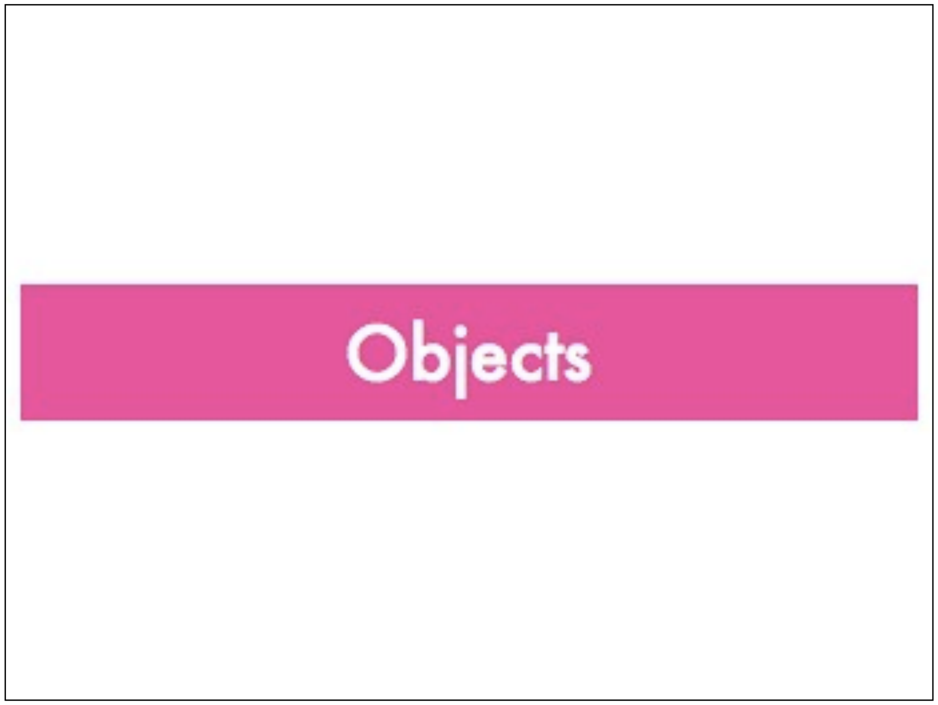
```
>>> def hello_there():
        print "Type your name:"
        name = raw_input()
        print "Hi", name, "how are you?"

>>> hello_there()
Type your name:
Barbara
Hi Barbara how are you?
```



Input

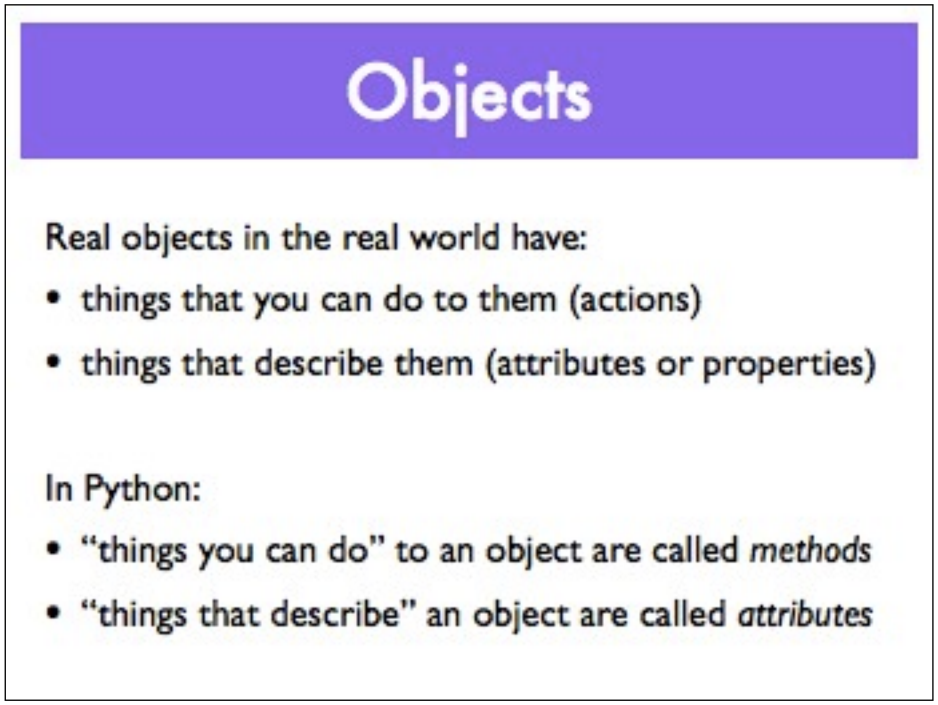
A shortcut:

```
>>> def hello_there():
       name = raw_input("Type your name: ")
       print "Hi", name, "how are you?"

>>> hello_there()
Type your name: Barbara
Hi Barbara how are you?
```



Objects

Note: This whole section was a little too abstract for the younger students (9-12), but worked well with the older students (13-16).



Objects

Real objects in the real world have:

- things that you can do to them (actions)
- things that describe them (attributes or properties)

In Python:

- "things you can do" to an object are called *methods*
- "things that describe" an object are called *attributes*

Objects

This ball object might have these *attributes*

```
myBall.color
myBall.size
myBall.weight
```

You can display them:

```
print myBall.size
```

You can assign values to them:

```
myBall.color = 'green'
```

You can assign them to attributes in other objects:

```
anotherBall.color = myBall.color
```

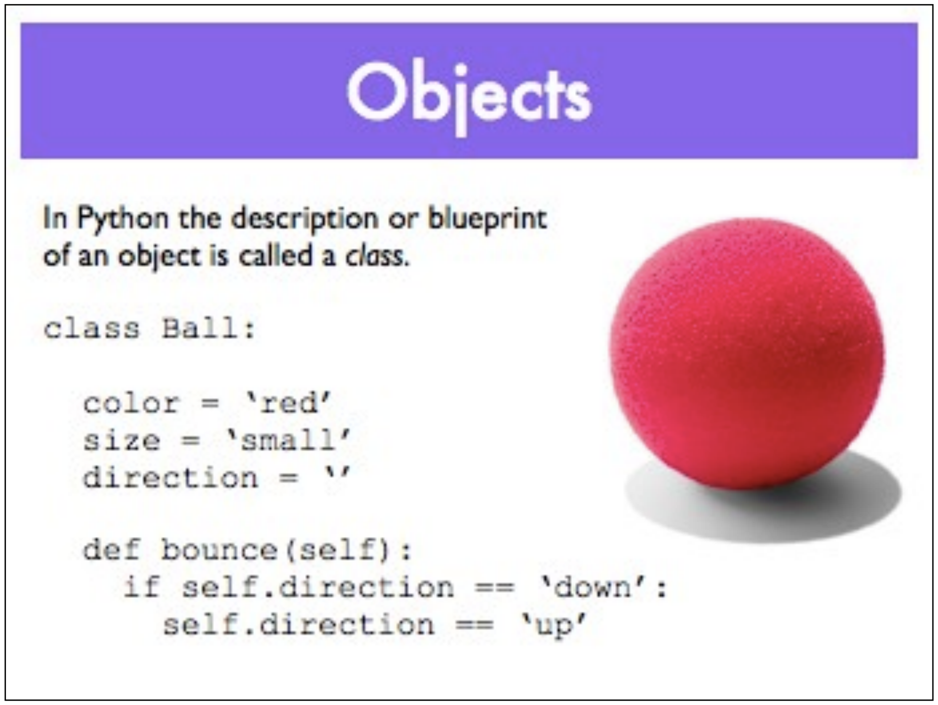
Objects

The ball object might have these *methods*:

```
ball.kick()
ball.throw()
ball.inflate()
```

Methods are the things you can *do* with an object.

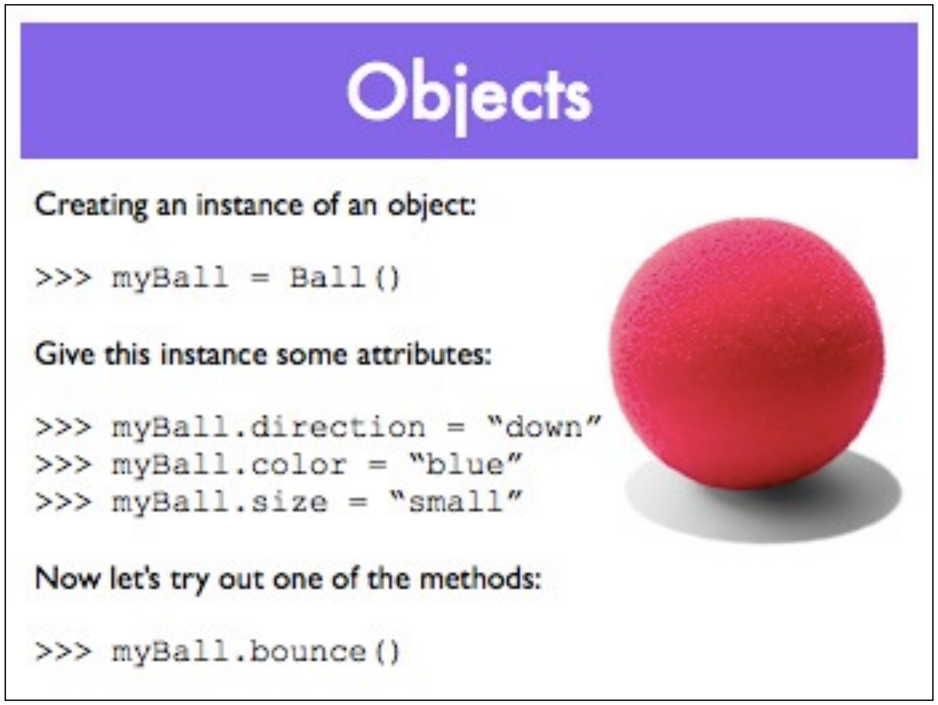
Methods are chunks of code - *functions* - that are included *inside* the object.



Objects

In Python the description or blueprint of an object is called a *class*.

```
class Ball:

  color = ‘red’
  size = ‘small’
  direction = ‘’

  def bounce(self):
    if self.direction == ‘down’:
      self.direction == ‘up’
```



Objects

Creating an instance of an object:

```
>>> myBall = Ball()
```

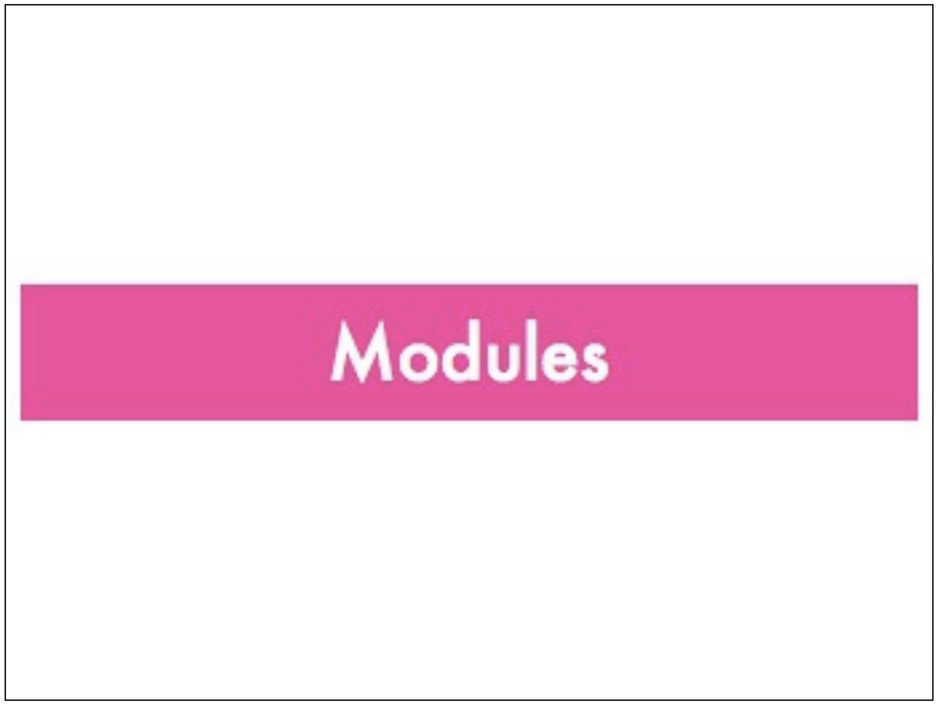
Give this instance some attributes:

```
>>> myBall.direction = “down”
>>> myBall.color = “blue”
>>> myBall.size = “small”
```

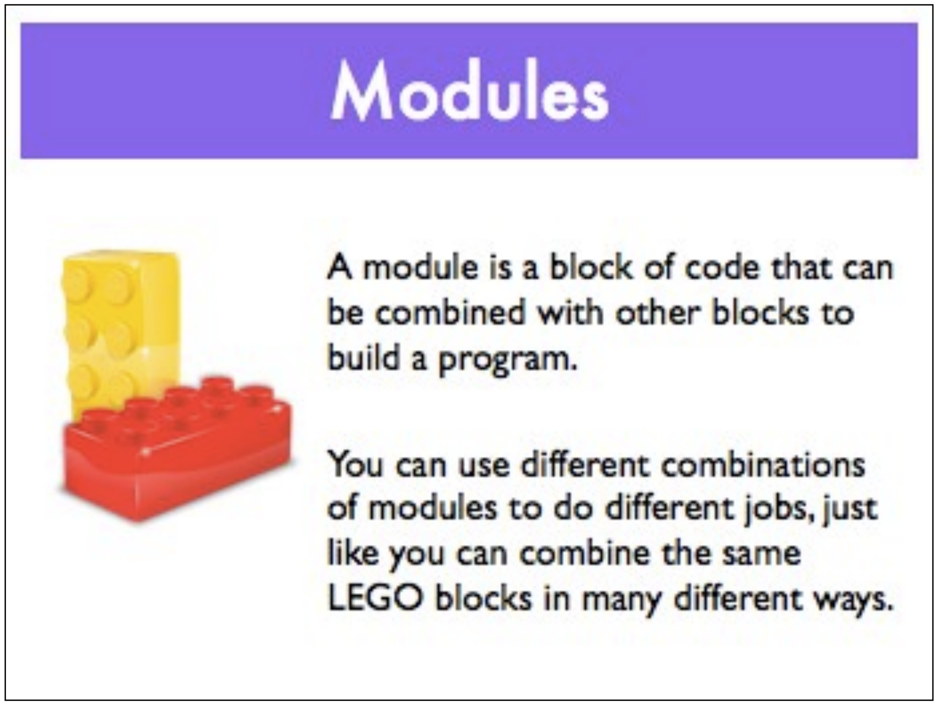
Now let's try out one of the methods:

```
>>> myBall.bounce()
```

Here’s how we create a new object.



Modules

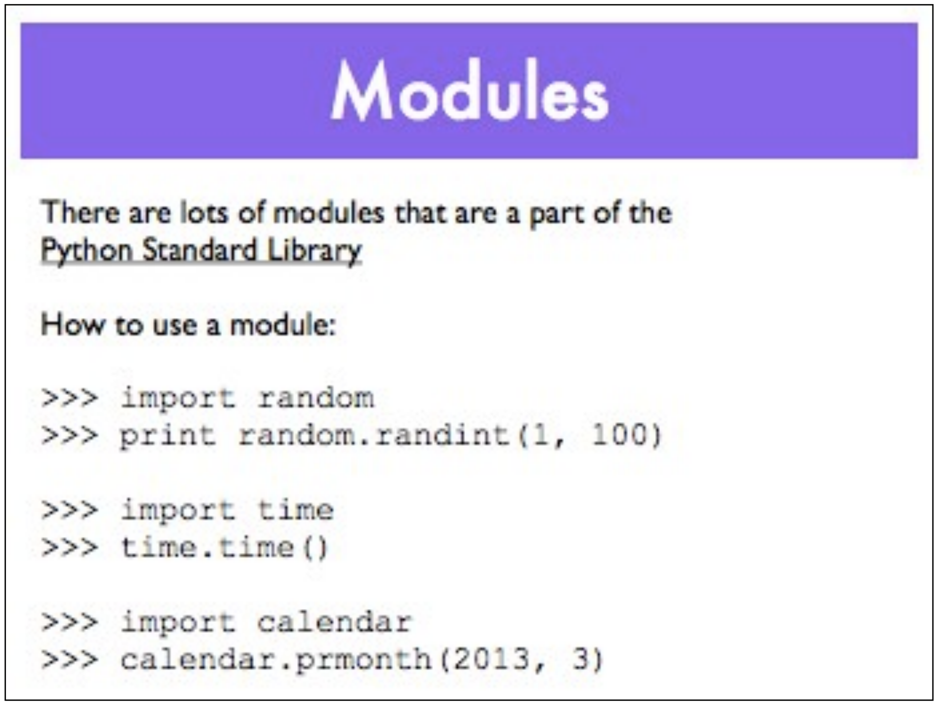


Modules

A module is a block of code that can be combined with other blocks to build a program.

You can use different combinations of modules to do different jobs, just like you can combine the same LEGO blocks in many different ways.

“We’ve already defined some functions of our own, but Python has a lot of functions built in – here’s how we use them.”



Modules

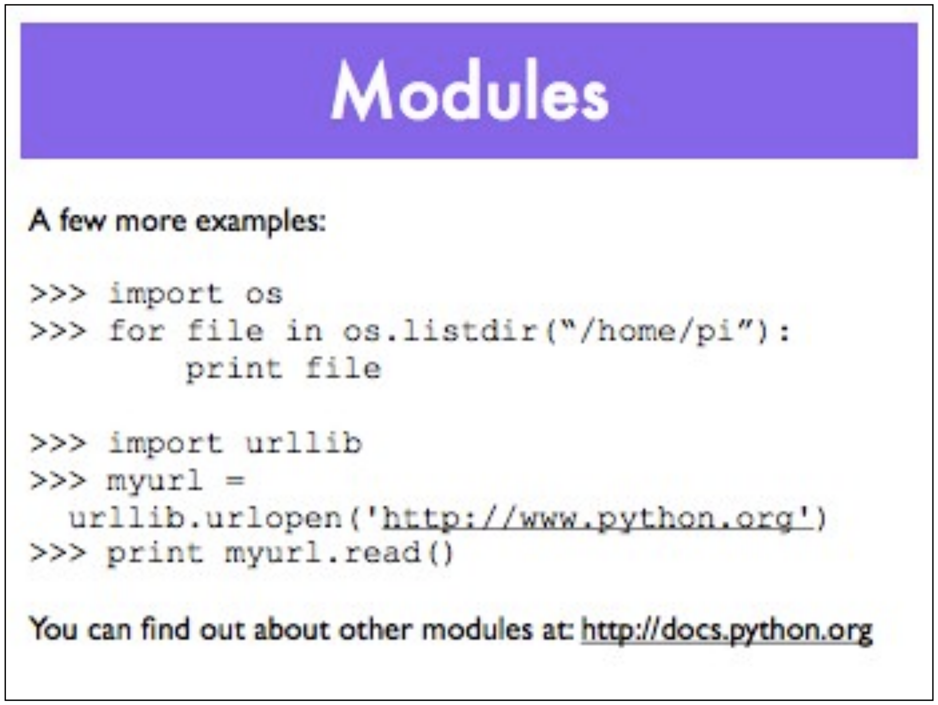
There are lots of modules that are a part of the Python Standard Library

How to use a module:

```
>>> import random
>>> print random.randint(1, 100)

>>> import time
>>> time.time()

>>> import calendar
>>> calendar.prmonth(2013, 3)
```

"When you 'import' a module, you can use all the functions inside that module."

New term: "dot notation"
Emphasize the distinction between the module name and the function name.
Explain what each of these examples does.



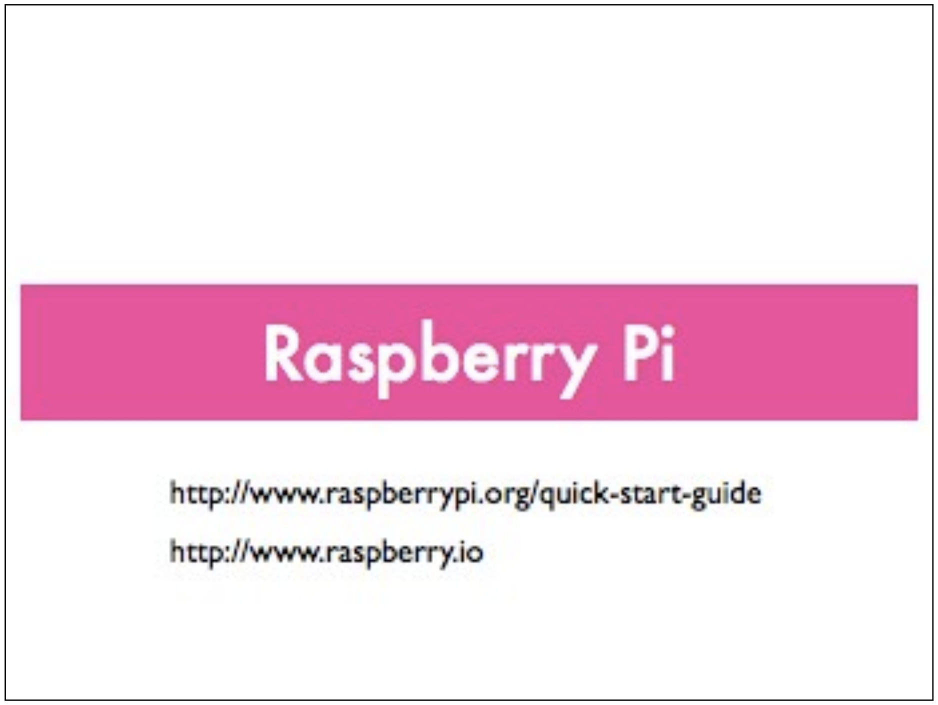
Modules

A few more examples:

```
>>> import os
>>> for file in os.listdir("/home/pi"):
        print file

>>> import urllib
>>> myurl =
  urllib.urlopen('http://www.python.org')
>>> print myurl.read()
```

You can find out about other modules at: http://docs.python.org

Talk about how many other modules can be found in the standard library.


Raspberry Pi

http://www.raspberrypi.org/quick-start-guide

http://www.raspberry.io

Let's make a game!

Introduce the Raspberry Rogue code here

Go over the guessing game example (see the attached code samples)