Beyond Blocks: Python Session #1

CS10 Spring 2013
Thursday, April 30, 2013
Michael Ball

Goals

- Quick introduction to Python
  - *Not* a tutorial or "how to"
  - Hope is that you'll want to learn (more)
- Advantages over higher level languages
- Challenges of programming syntax
  - It's really like "writing BYOB on Paper"!

Beyond Blocks: Python #1

Installation: Mac Check

- Open Terminal
- Type `python3` and hit *`return`*
- Type `print("hello world")` and hit *`return`*
- The result should be:

```
>>> print("hello world")
hello world
>>>
```

Beyond Blocks: Python #1

Installation: Windows Check

- Get Python to "`print`" something with these instructions:

  http://docs.python.org/faq/windows.html

(You only have to get to the "Many people use the interactive mode as a convenient yet highly programmable calculator" paragraph)

Beyond Blocks: Python #1

Installation: More Information

- Computer Science Circles : Run Python at Home

  cemclinux1.math.uwaterloo.ca/~cscircles/wordpress/run-at-home/

Beyond Blocks: Python #1

Installation: Version Check

```
Michael> python3 -V
Python 3.3.4
```

We'll be talking about version 3.3.x here, although version 2.7.x (which is more common) works just as well!

If curious, there's more version info at:
http://docs.python.org/whatsnew/index.html

Beyond Blocks: Python #1

Why used "text based" programming?

<demo>

BYOB  Python

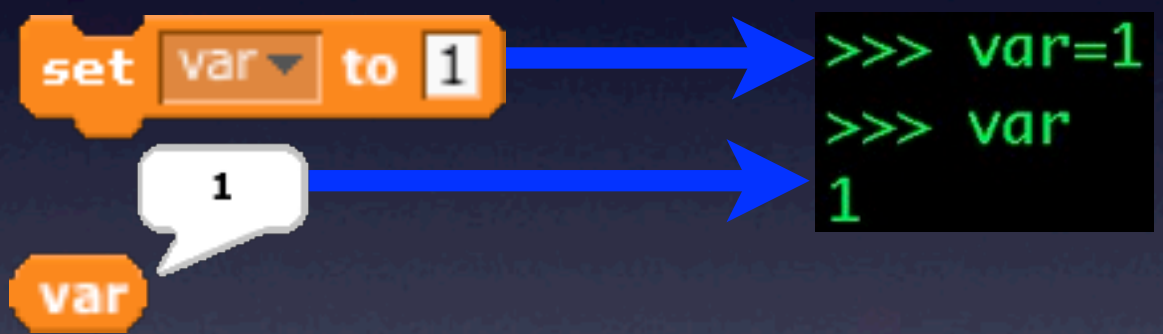
BYOB ⟷ Python

Variables


```
>>> var = 0
>>>
```

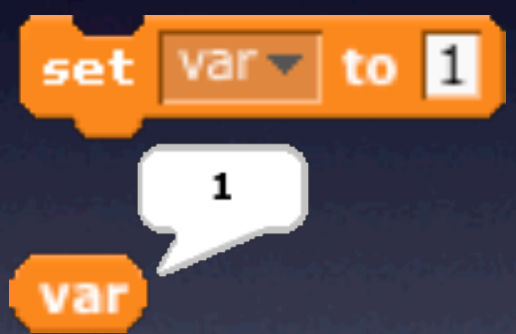
BYOB ↔ Python

Variables



BYOB ↔ Python

Variables

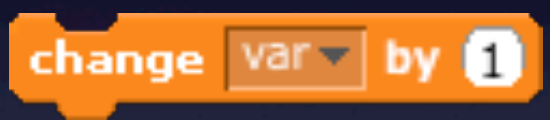



```
>>> var=1
>>> var
1
```

NOTE:
Assignment doesn't "evaluate" to anything, so nothing is printed!

BYOB ⟷ Python

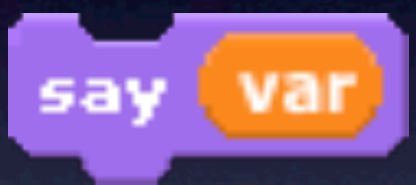
Variables




```
>>> var = var + 1
```

BYOB ⟷ Python

Variables




```
print(var)
```

NOTE:
Printing is one of the big differences between Python 2 and 3. Python 3 requires () with print!
For the sake of humanity, just use print()! :)

BYOB ⟷ Python

Operators


```
>>> 1+1
2
>>> 2-1
1
>>> 2*2
4
>>> 6/2
3
```

BYOB ⟷ Python

Types

- Everything in Python has an internal "type"
- Types are determined *dynamically*
  - `x = 1`
  - x now has the `type "int"`:
    - (short for "integer")

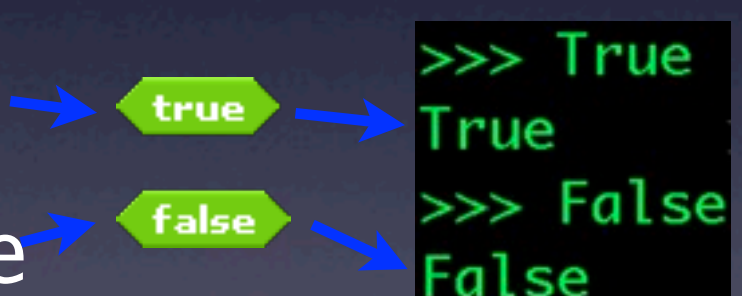
```
>>> x=1
>>> type(x)
<type 'int'>
>>>
```

We'll talk about this "script" (or function) later...

BYOB ⟷ Python

Types: bool

- 'bool' is short for boolean
- 'bool's can have two values:
  - True
  - False




```
>>> type(True)
<type 'bool'>
```

BYOB ⟷ Python

Types: bool

- 'bool' is short for boolean
- 'bool's can have two values:
  - True
  - False

```
>>> type(True)
<class 'bool'>
```


```
>>> True
True
```


```
>>> False
False
```

NOTE: Upper case is important!

BYOB ⟷ Python

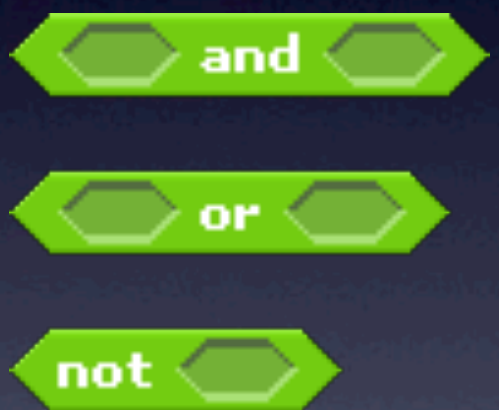
Types: function type()

This function returns the type that Python has assigned the *identifier*.

```
>>> type(True)
<class 'bool'>
>>> type(1)
<class 'int'>
>>> type(1.0)
<class 'float'>
>>> type("Hello, there!")
<class 'str'>
>>> type("1")
<class 'str'>
```

BYOB ↔ Python

Operators




```
>>> True and False
False
>>> True and True
True
>>> True or False
True
>>> not True
False
>>> not False
True
```

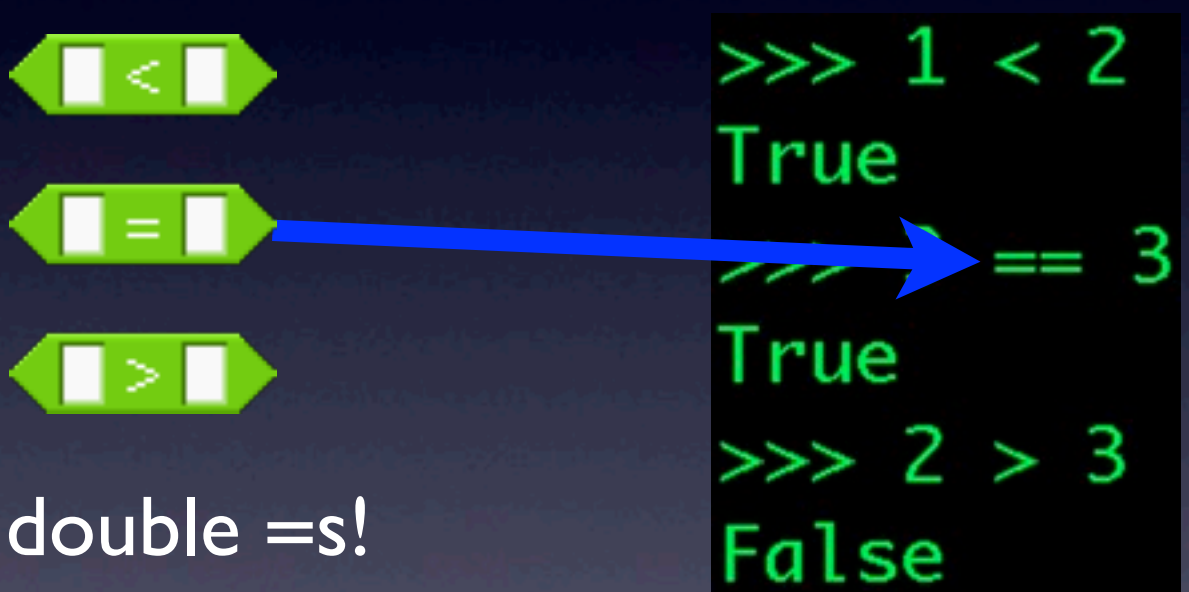
BYOB ⟷ Python

Operators


```
>>> 1 < 2
True
>>> 3 == 3
True
>>> 2 > 3
False
```

BYOB ⟷ Python

Operators



- Note the double =s!
- = means ***assign***, == means ***compare***
- Very common source of bugs!

BYOB ⟷ Python

Operators

= means ***assign***,

== means ***compare***

BYOB ⟷ Python

Operators

= means ***assign***,

== means ***compare***

BYOB ↔ Python

Operators


```
>>> 3 % 2
1
>>> 12345 % 678
141
```

BYOB ↔ Python

Sidebar: Division (integer vs. real/float)

Python 2

```
>>> 5/6
0
>>> 5.0/6.0
0.8333333333333334
>>> 5.0//6.0
0.0
```

Python 3

```
>>> 5/6
0.8333333333333334
>>> 5.0/6.0
0.8333333333333334
>>> 5.0//6.0
0.0
```

BYOB ↔ Python

Sidebar: Division (integer vs. real/float)


```
>>> 5/6
0.8333333333333334
>>> 5.0/6.0
0.8333333333333334
>>> 5.0//6.0
0.0
```

"Force" integer division

BYOB ⟷ Python

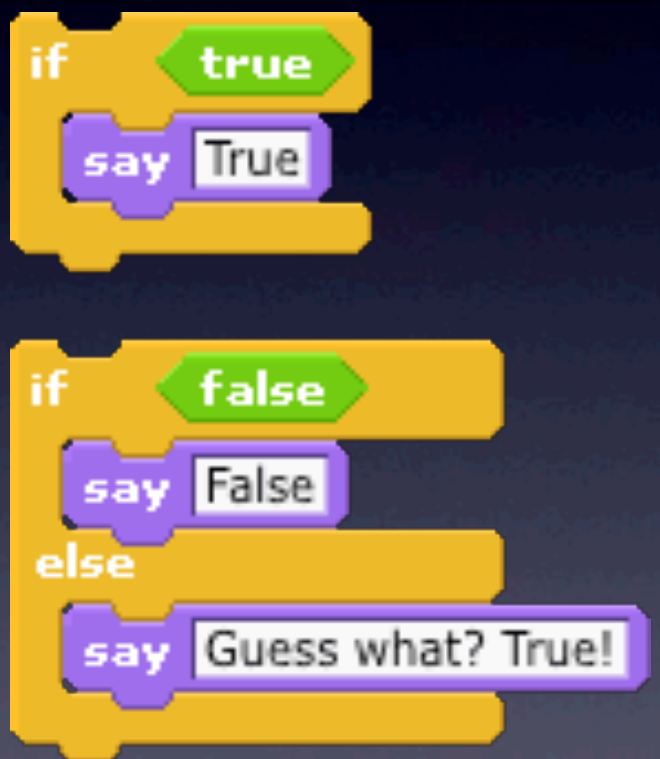
Sidebar: Exponent

BYOB has e^x and 10^x,
but Python can do <u>any</u> base & exponent!

```
>>> 2**8
256
>>> 2**10
1024
>>> 2**100
1267650600228229401496703205376
```

BYOB ↔ Python

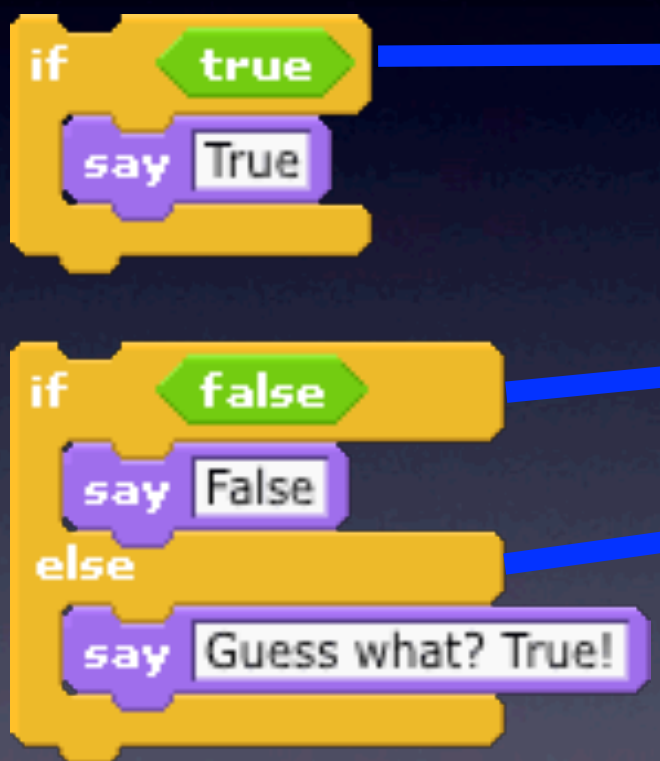
Conditionals




```
>>> if (True):
...     print "True"
...
True
>>> if (False):
...     print "False"
... else:
...     print "Guess what? True!"
...
Guess what? True!
```

BYOB ⟷ Python

Conditionals




```
>>> if (True):
...     print "True"
...
True
>>> if (False):
...     print "False"
... else:
...     print "Guess what? True!"
...
Guess what? True!
```

BYOB ↔ Python

Conditionals

```
>>> if (True):
...     print "True"
...
True
>>> if (False):
...     print "False"
... else:
...     print "Guess what? True!"
...
Guess what? True!
```

Notice the colon and indentation *syntax*!

BYOB ↔ Python

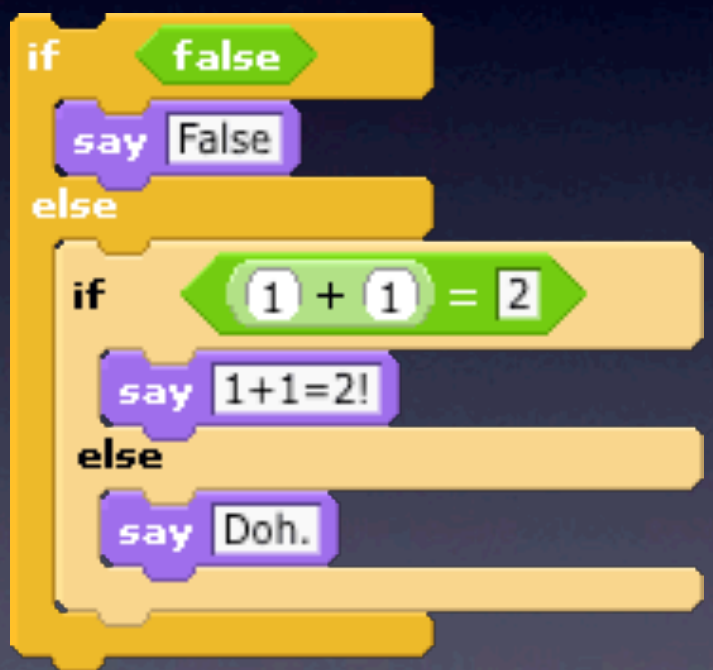
Conditionals

```
>>> if (True):
...     print "True"
...
True
>>> if (False):
...     print "False"
... else:
...     print "Guess what? True!"
...
Guess what? True!
```

Notice the colon and indentation *syntax*!

BYOB ↔ Python

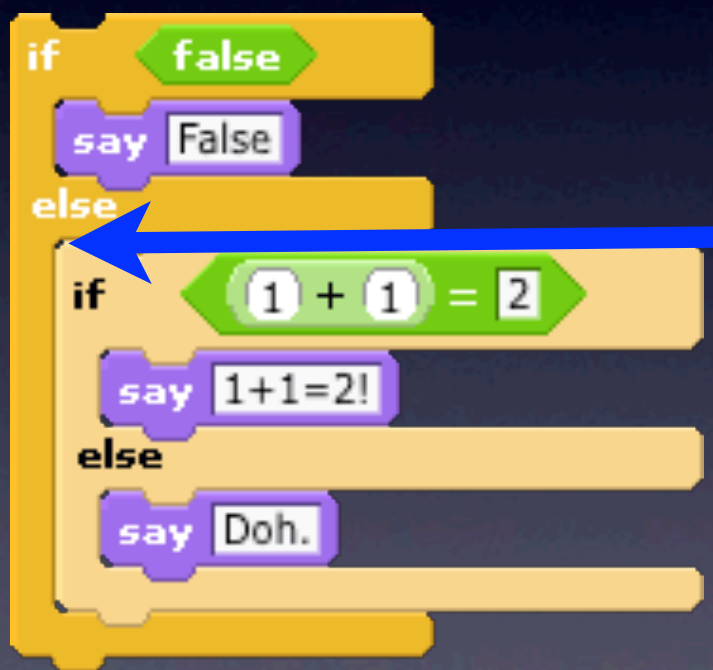
Conditionals




```
>>> if (False):
...     print "False"
... elif (1+1==2):
...     print "1+1==2!"
... else:
...     print "Doh."
...
1+1==2!
```

BYOB ↔ Python

Conditionals




```
>>> if (False):
...     print "False"
... elif (1+1==2):
...     print "1+1==2!"
... else:
...     print "Doh."
...
1+1==2!
```

BYOB ↔ Python

Loops


```
>>> var = 0
>>> while(True):
...     print var
...     var = var + 1
...
0
1
2
3
4
5
6
7
8
9
```

BYOB ↔ Python

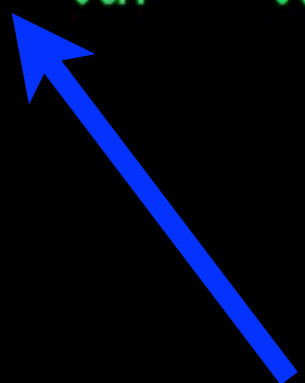
Loops


```
>>> var = 0
>>> while(True):
...     print var
...     var = var + 1
...
0
1
2
3
4
5
6
7
8
9
```

BYOB ↔ Python

Loops

```
>>> var = 0
>>> while(True):
...     print var
...     var = var + 1
...
0
1
2
3
4
5
6
7
8
9
```



Note the indentation (again)!

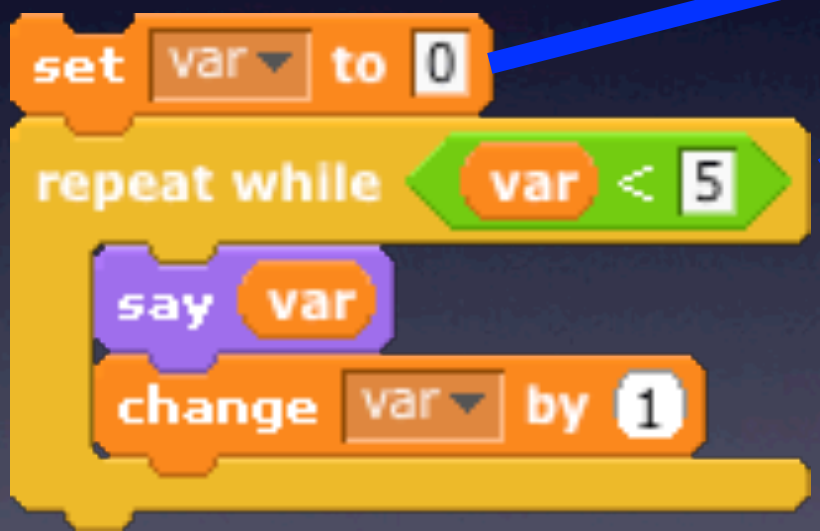
BYOB ⟷ Python

Loops


```
>>> var = 0
>>> while( var < 5 ):
...     print var
...     var = var + 1
...
0
1
2
3
4
```

BYOB ↔ Python

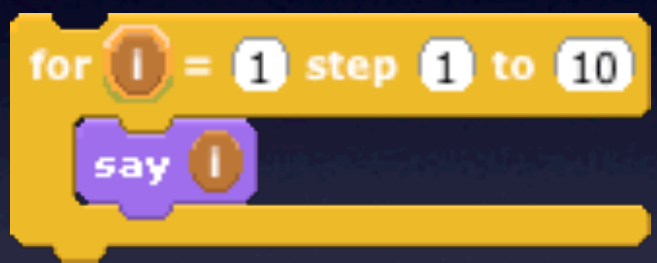
Loops




```
>>> var = 0
>>> while( var < 5 ):
...     print var
...     var = var + 1
...
0
1
2
3
4
```

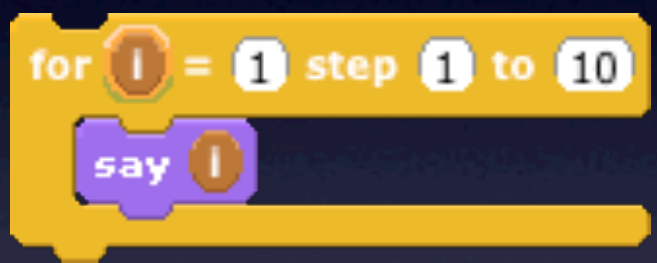
BYOB ↔ Python

More Loops



BYOB ⟷ Python

Moar [sic] Loops



There isn't really an exact equivalent of this in Python...

We'll talk more about this in Session #2...

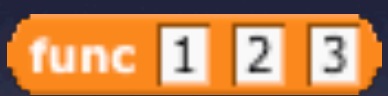
BYOB ↔ Python

Functions: Calling

- Calling functions (the *syntax*) looks like this:

```
>>> func(1,2,3)
```

- Equivalent to creating & running a BYOB block:



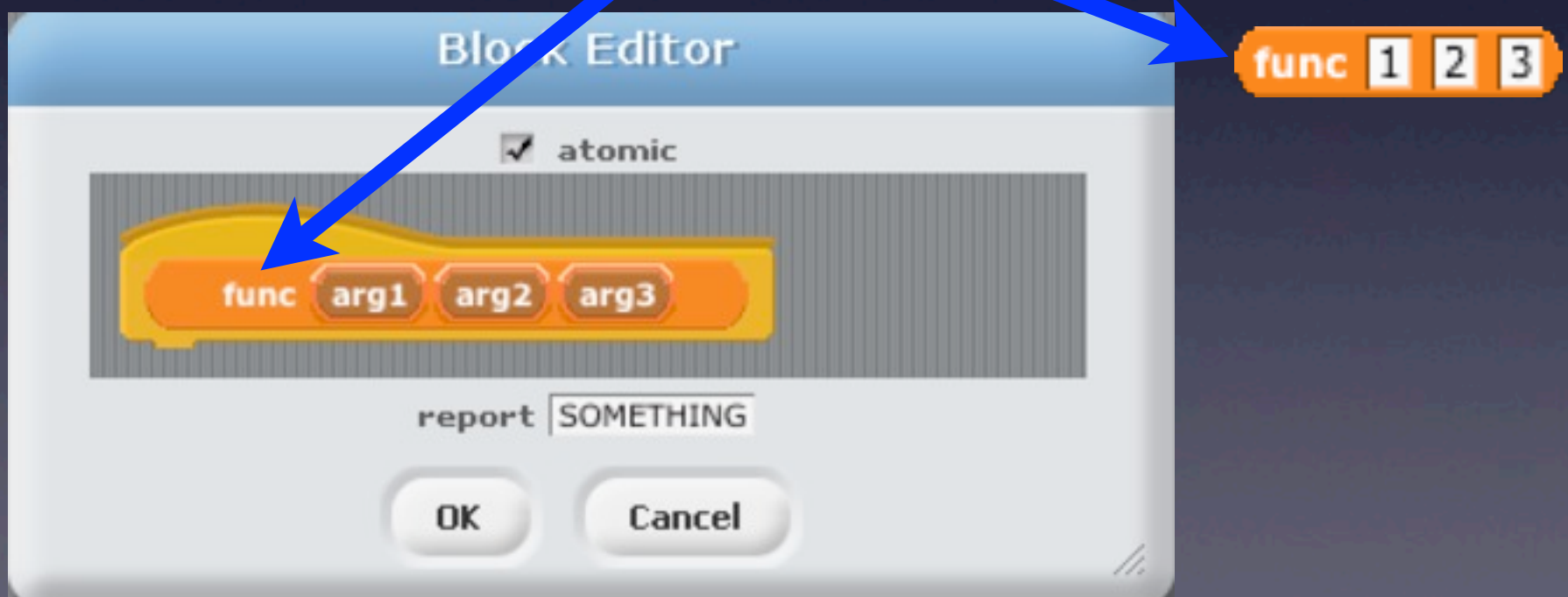
BYOB ↔ Python

Functions: Calling

- Calling functions (the *syntax*) looks like this:

```
>>> func(1,2,3)
```

- Equivalent to creating & running a BYOB block:



BYOB ↔ Python

Functions: Calling

- Calling functions (the *syntax*) looks like this:

```
>>> func(1,2,3)
```

- Equivalent to creating & running a BYOB block:

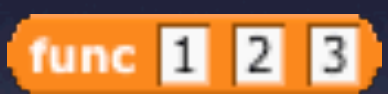
BYOB ↔ Python

Functions: Calling

- Calling functions (the *syntax*) looks like this:

```
>>> func(1,2,3)
```

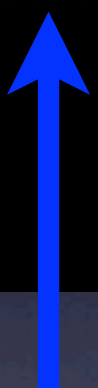
- Equivalent to creating & running a BYOB block:



BYOB ↔ Python

Functions : Defining

```
>>> def func(arg1,arg2,arg3):
...     pass
...     pass
...
>>>
```

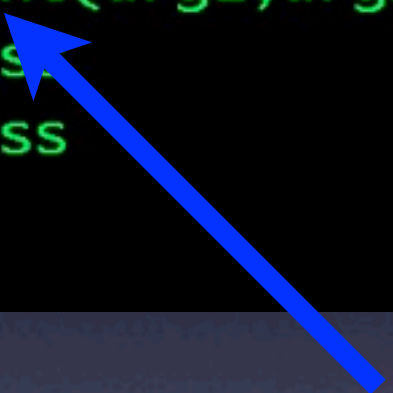


Keyword: DEF

BYOB ↔ Python

Functions : Defining

```
>>> def func(arg1,arg2,arg3):
...     pas
...     pass
...
>>>
```

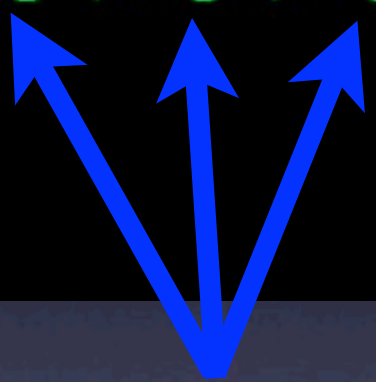


Name of the function

BYOB ↔ Python

Functions : Defining

```
>>> def func(arg1,arg2,arg3):
...     pass
...     pass
...
>>>
```



"Arguments," or inputs to the function

BYOB ↔ Python

Functions : Defining

```
>>> def func(arg1,arg2,arg3):
...     pass
...     pass
...
>>>
```

Indentation: the key to Python and "scope."!

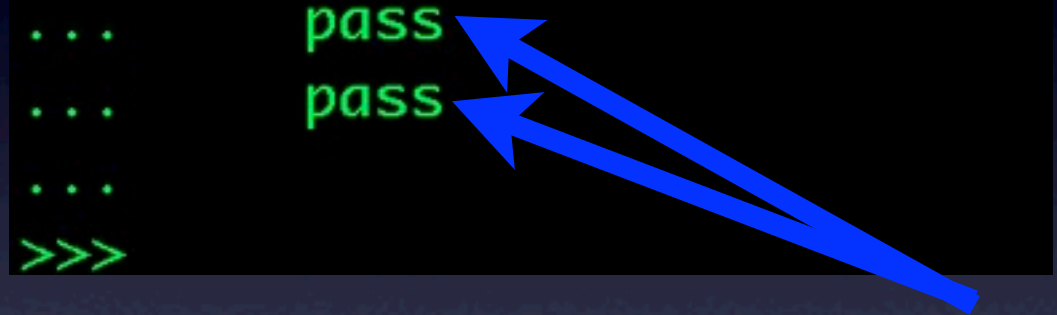
In Python: **Indentation matters!!**

We'll talk about "scope" later...

BYOB ↔ Python

Functions : Defining

```
>>> def func(arg1,arg2,arg3):
...     pass
...     pass
...
>>>
```

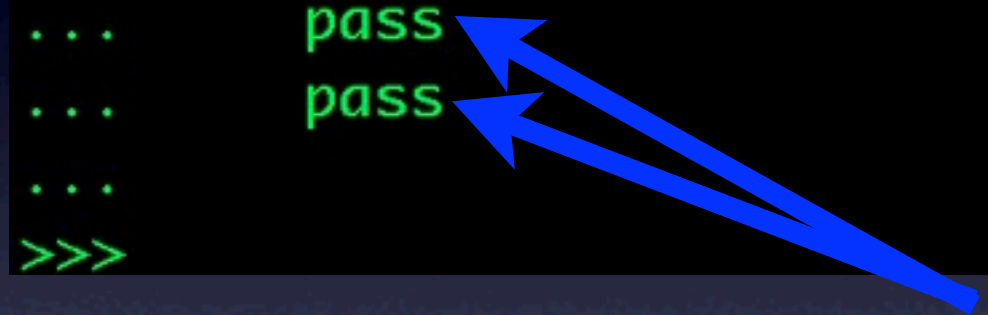


pass: Python's "placeholder"
Skip this, and do nothing.

BYOB ↔ Python

Functions : Defining

```
>>> def func(arg1,arg2,arg3):
...     pass
...     pass
...
>>>
```



pass: Python's "placeholder"
Skip this, and do nothing.

Functions must have a body!

BYOB ↔ Python

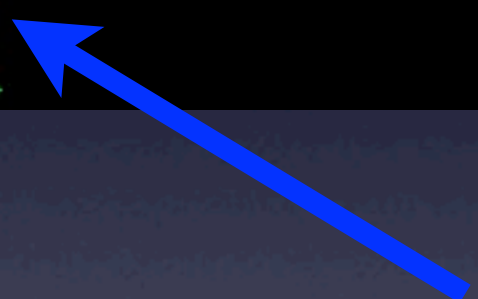
Functions : Defining

```
>>> def func(arg1,arg2,arg3):
...     pass
...     pass
...
>>>
```

BYOB ↔ Python

Functions : Defining

```
>>> def func(arg1,arg2,arg3):
...     pass
...     pass
...
>>>
```



Hitting Return/Enter (on an empty line)
"closes" (finishes) the definition.

BYOB ⟷ Python

Sidebar: Keywords

```
and       del       from      not       while
as        elif      global    or        with
assert    else      if        pass      yield
break     except    import
class     exec      in        raise
continue  finally   is        return
def       for       lambda    try
```

- Words reserved by Python
- List at: docs.python.org/reference/lexical_analysis.html

BYOB ⟷ Python

Built In Functions

```
abs()
dict()
help()
min()
setattr()
all()
dir()
hex()
next()
slice()
any()
divmod()
id()
object()
sorted()
ascii()
enumerate()
input()
```

```
oct()
staticmethod()
bin()
eval()
int()
open()
str()
bool()
exec()
isinstance()
ord()
sum()
bytearray()
filter()
issubclass()
pow()
super()
bytes()
```

```
float()
iter()
print()
tuple()
callable()
format()
len()
property()
type()
chr()
frozenset()
list()
range()
vars()
classmethod()
getattr()
locals()
repr()
```

```
zip()
compile()
globals()
map()
reversed()
__import__()
complex()
hasattr()
max()
round()
delattr()
hash()
memoryview()
set()
```

- http://docs.python.org/3.3/library/functions.html

BYOB ⟷ Python

USEFUL: Built In Functions

```
abs()       type()      map()       tuple()
help()      str()       filter()    list()
min()       chr()       sum()       dict()
max()       ord()                   set()
print()     bool()      open()
range()     float()
            int()       iter()
                        len()
```

- http://docs.python.org/3.3/library/functions.html

BYOB ↔ Python

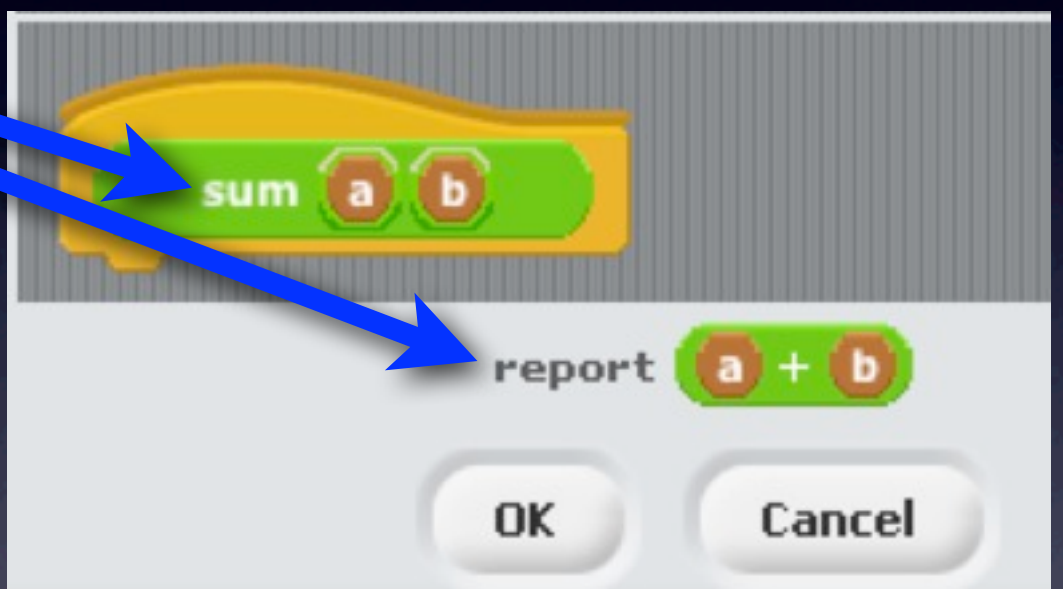
Functions : Returning Values

```
>>> def sum(a,b):
...     return (a+b)
...
>>> c=sum(5,7)
>>> print c
12
```

BYOB ↔ Python

Functions : Returning Values

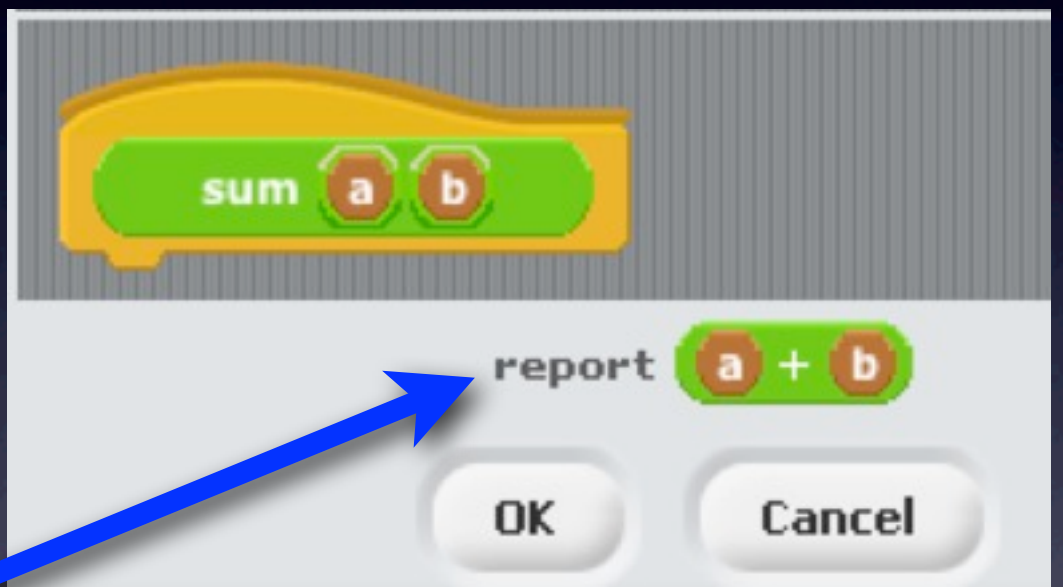
```
>>> def sum(a,b):
...     return (a+b)
...
>>> c=sum(5,7)
>>> print c
12
```



BYOB ↔ Python

Functions : Returning Values

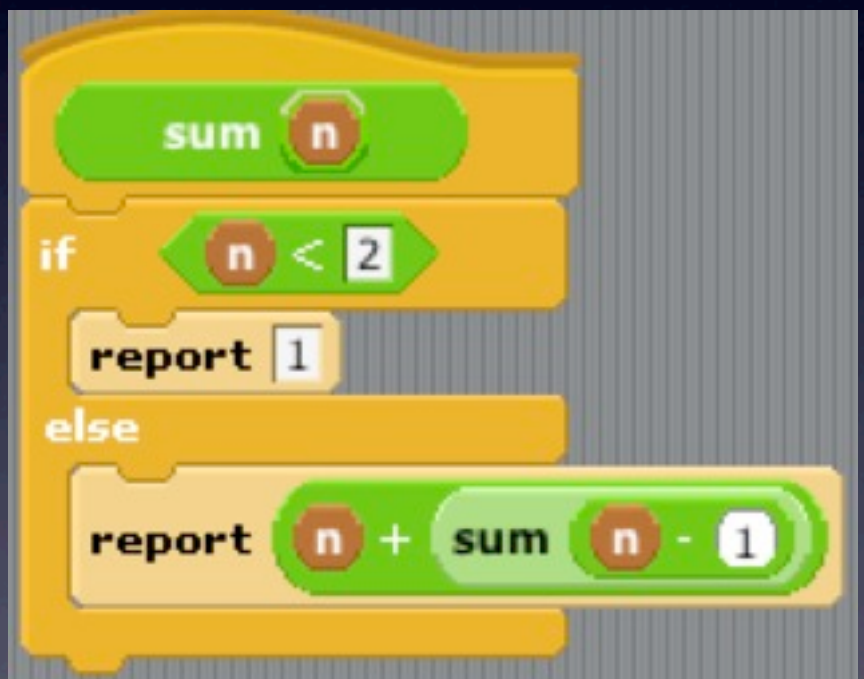
```
>>> def sum(a,b):
...     return (a+b)
...
>>> c=sum(5,7)
>>> print c
12
```



"return" and "report" are equivalent!

BYOB ↔ Python

Functions : Recursion!



BYOB ⟷ Python

Functions : Recursion!

```
>>> def sum( n ):
...     if ( n == 0 ):
...         return 0
...     else:
...         return n + sum( n - 1 )
...
>>> sum(5)
15
```

BYOB ↔ Python

Functions : Recursion! Within Reason!

```
>>> sum(1234)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 5, in sum
  File "<stdin>", line 5, in sum
```

•
•
•

```
  File "<stdin>", line 5, in sum
  File "<stdin>", line 5, in sum
  File "<stdin>", line 5, in sum
  File "<stdin>", line 5, in sum
RuntimeError: maximum recursion depth
>>>
```

Beyond Blocks: Python #1

More Information

- **Python.org**: www.python.org
- **Python Docs**: www.python.org/doc/
- **Python Modules**: docs.python.org/modindex.html
- **CodeAcademy**: codeacademy.com
- **Online PythonTutor**: http://www.pythontutor.com

Beyond Blocks: Python #1

More Information

- **Computer Science Circles: Python**

  cemclinux1.math.uwaterloo.ca/~cscircles/wordpress/using-this-website/

- **Dive Into Python**: diveintopython.org/toc/
- **Cal's Self-Paced Center**:

  inst.eecs.berkeley.edu/~selfpace/class/cs9h/

  How to Think Like a Computer Scientist (Python Version)

  www.greenteapress.com/thinkpython/thinkCSpy/html/