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:

```python
>>> 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 use "text based" programming?

<demo>
BYOB ↔ Python
BYOB ↔ Python

Variables

set var to 0

>>> var = 0

>>>
BYOB ↔ Python

Variables

set var to 1

>>> var = 1
>>> var
1
BYOB ↔ Python

Variables

NOTE:
Assignment doesn’t “evaluate” to anything, so nothing is printed!
BYOB ↔ Python

Variables

```python
>>> var = var + 1
```
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”)

We’ll talk about this “script” (or function) later...
• `bool` is short for boolean
• `bool`’s can have two values:
  • True
  • False

```
>>> True
True
>>> False
False
>>> type(True)
<type 'bool'>
```
• ‘bool’ is short for boolean
• ‘bool’ s can have two values:
  • True
  • False

NOTE: Upper case is important!
BYOB ↔ Python

Types: function type()

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

```python
>>> 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

```python
>>> True and False
False
>>> True and True
True
>>> True or False
True
>>> not True
False
>>> not False
True
```
BYOB ↔ Python

Operators

```python
>>> 1 < 2
True
>>> 3 == 3
True
>>> 2 > 3
False
```
BYOBY 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
Sidebar: Division (integer vs. real/float)

Python 2

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

Python 3

```python
>>> 5/6
0.8333333333333334
>>> 5.0/6.0
0.8333333333333334
>>> 5.0//6.0
0.0
```
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 any base & exponent!

```python
>>> 2**8
256
>>> 2**10
1024
>>> 2**100
1267650600228229401496703205376
```
BYOB ↔ Python

Conditionals

```python
>>> if (True):
    ...    print "True"
    ...
    True

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

```python
if (True):
    print "True"

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

```python
if (True):
    ... print "True"

if (False):
    ... print "False"
else:
    ... print "Guess what? True!"
```
BYOB ↔ Python

Conditionals

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

Notice the colon and indentation syntax!
BYOB ↔ Python

Conditionals

```python
>>> if (True):
...   print "True"
...
True
```

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

Notice the colon and indentation syntax!
BYOB ↔ Python

Conditionals

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

Conditionals

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

Loops

```python
>>> var = 0
>>> while(True):
...     print var
...     var = var + 1
...     print var
data
```

```
0
1
2
3
4
5
6
7
8
9
```
BYOB ↔ Python

Loops

```python
>>> var = 0
>>> while(True):
...   print var
...   var = var + 1
...```

Note the indentation (again)!
BYOB ↔ Python
Loops

```python
>>> 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
```
More 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:

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

• Equivalent to creating & running a BYOB block:
BYOB ↔ Python

Functions: Calling

• Calling functions (the syntax) looks like this:

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

• Equivalent to creating & running a BYOB block:
 Calling functions (the syntax) looks like this:

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

Equivalent to creating & running a BYOB block:
BYOB ↔ Python

Functions: Calling

• Calling functions (the \textit{syntax}) looks like this:

\begin{verbatim}
>>> func(1,2,3)
\end{verbatim}

• Equivalent to creating & running a BYOB block:
BYOB <-> Python

Functions: Defining

```python
>>> def func(arg1, arg2, arg3):
    ...
    pass
    ...
    pass
>>> 
Keyword: DEF
```
BYOB ↔ Python

Functions : Defining

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

Name of the function
BYOB ↔ Python
Functions : Defining

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

“Arguments,” or inputs to the function
BYOB ↔ Python

Functions: Defining

Indentation: the key to Python and “scope.”!

In Python: **Indentation matters!!**

We’ll talk about “scope” later...
pass: Python’s “placeholder”
Skip this, and do nothing.
Functions must have a body!
BYOB ↔ Python

Functions : Defining

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

>>> 
```
BYOB ↔ Python

Functions : Defining

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

Hitting Return/Enter (on an empty line) “closes” (finishes) the definition.
BYOB ↔ Python

Sidebar: Keywords

- Words reserved by Python
- List at: [docs.python.org/reference/lexical_analysis.html](https://docs.python.org/reference/lexical_analysis.html)

<table>
<thead>
<tr>
<th>and</th>
<th>del</th>
<th>from</th>
<th>not</th>
<th>while</th>
</tr>
</thead>
<tbody>
<tr>
<td>as</td>
<td>elif</td>
<td>global</td>
<td>or</td>
<td>with</td>
</tr>
<tr>
<td>assert</td>
<td>else</td>
<td>if</td>
<td>pass</td>
<td>yield</td>
</tr>
<tr>
<td>break</td>
<td>except</td>
<td>import</td>
<td>raise</td>
<td></td>
</tr>
<tr>
<td>class</td>
<td>exec</td>
<td>in</td>
<td>return</td>
<td></td>
</tr>
<tr>
<td>continue</td>
<td>finally</td>
<td>is</td>
<td>try</td>
<td></td>
</tr>
<tr>
<td>def</td>
<td>for</td>
<td>lambda</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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()
int()
open()
str()
bool()
exec()
isinstance()
ord()
slice()
ord()
slice()
ord()
slice()
ord()
slice()
ord()
slice()
ord()
slice()
ord()

float()
iter()
print()
tuple()
callable()
format()
len()
property()
type()
chr()
frozenset()
list()
range()
vars()

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()`
- `help()`
- `min()`
- `max()`
- `print()`
- `range()`
- `str()`
- `chr()`
- `ord()`
- `bool()`
- `float()`
- `int()`
- `tuple()`
- `type()`
- `str()`
- `chr()`
- `ord()`
- `true()`
- `float()`
- `int()`
- `tuple()`
- `type()`
- `str()`
- `chr()`
- `ord()`
- `bool()`
- `float()`
- `int()`
- `tuple()`

### Additional Resources

- [http://docs.python.org/3.3/library/functions.html](http://docs.python.org/3.3/library/functions.html)
BYOB ↔ Python

Functions : Returning Values

```python
>>> def sum(a, b):
...     return (a + b)
...

>>> c = sum(5, 7)
>>> print(c)
12
```
BYOB ↔ Python

Functions : Returning Values

```python
>>> 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!

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

>>> sum(5)
15
```
>>> 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
  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 Python Tutor: http://www.pythonontutor.com
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/