Besides Blocks: Python Session #1

Instructor: Dan Garcia

Computational Thinking

- It's using abstraction (removing detail and generalization with parameters)
- It's understanding the value of a "spec" that specifies a contract
- It's the iterative design cycle: design, prototype, implement, evaluate (loop)
- It's thinking about how solutions scale, parallelize, generalize, and trying to foresee the unintended consequences!

Codification



- Jens Moenig wrote Snap! with design support from Brian Harvey.
- Here is the coolest program ever.

tinyurl.com/BJCcodification

 Lesson: Snap! is Turing Complete, so anything they can do, we can do.

Outline

- Computational Thinking (5)
- Codification (10)
- Python? (5)
 - Getting Started (10)
- Why? (5)
- Syntax (20)
- Turtle (5)

Why Python? (1/2)

- Up until now, it's just been Snap!
- There's an advantage in just one language, there's only one cognitive tax paid for "learning a new language"
- However, we want CS61A success too!
- The feeling is this will help, it's in Python
- We also see the benefit of another tool, and learning when you'd use Snap! vs Python...
- It's "Besides" blocks, not "Beyond" blocks!

Why Python (2/2)

- Easy to learn and use, looks like pseudocode
- Minimal text-based syntax
- Easy to cut & paste, people type faster than drag
- Has a Turtle mode!
- Popular
- lots of online support
- Incredible libraries
- Makes you marketable

The Goals of BJC • BJC's goal is not to teach you Snap!

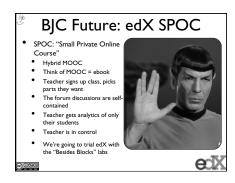
- It's to teach you critical thinking about societal implications of computing
- It's also to teach you how to program (Snap! is the best intro language we know) and help you succeed in CS61A
- More importantly it's to teach you how to think like a computer scientist in life, called "computational thinking"

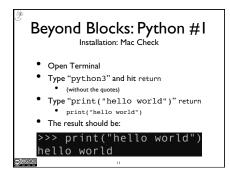
Props to edX and BB teams!

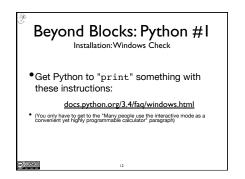
(see website)

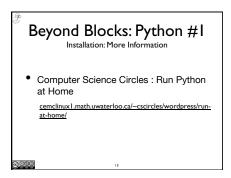
Learning Python

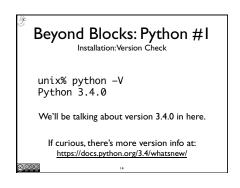
- Quick introduction to Python
- Not a tutorial or "how to"
- Hope is that you'll want to learn (more)
- Advantages over higher level languages
- Challenges of programming syntax
- Hope is that "foreign" syntax becomes less intimidating and more approachable
- Plan: Lec, Lab, Dis, Lec, Lab

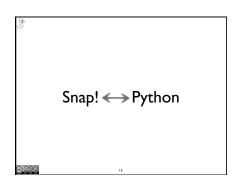


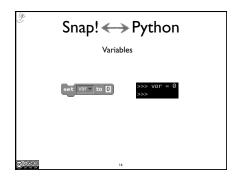


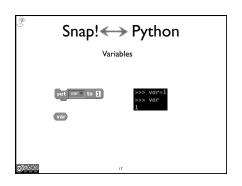


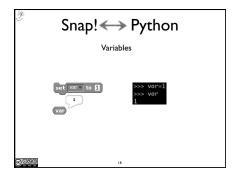


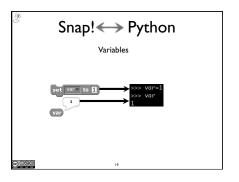


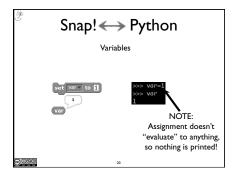


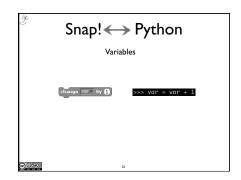


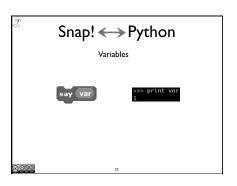


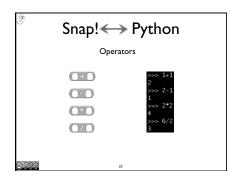


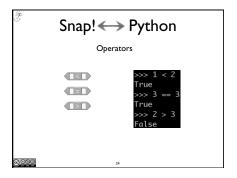


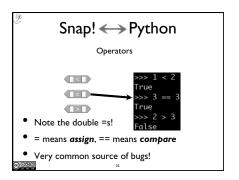


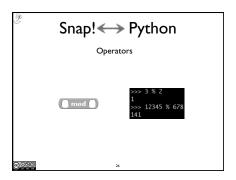


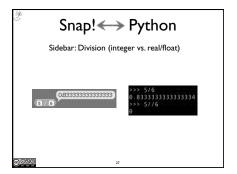


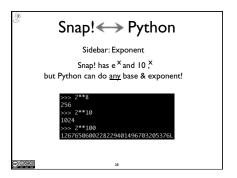


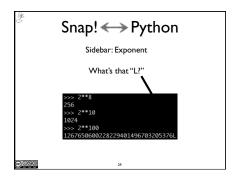


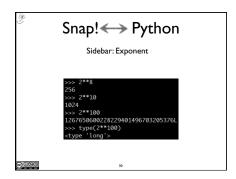


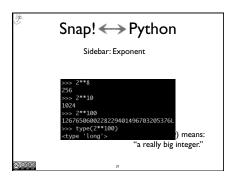


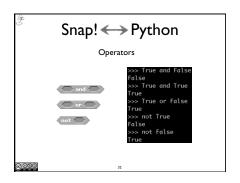


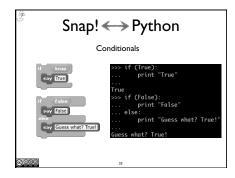


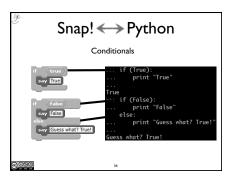


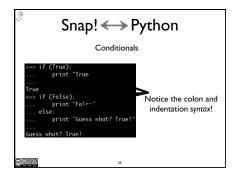


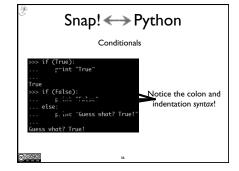


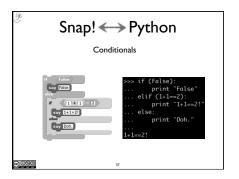


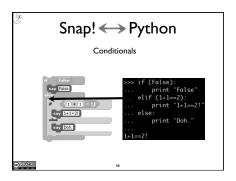


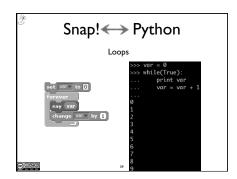


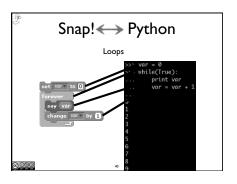


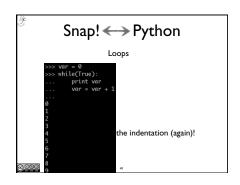


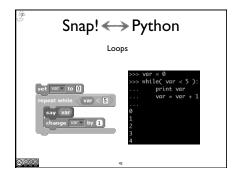


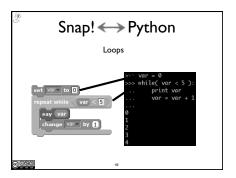


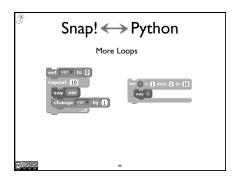


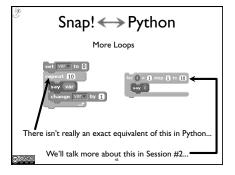


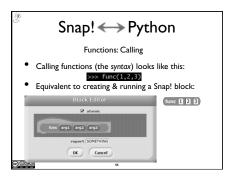


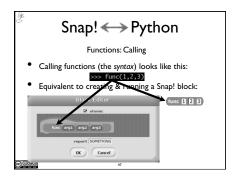


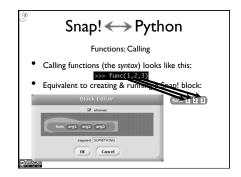


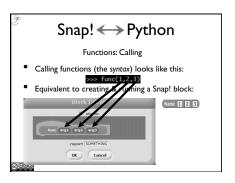


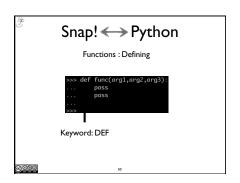


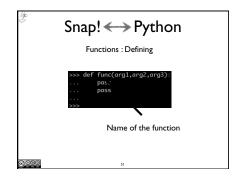


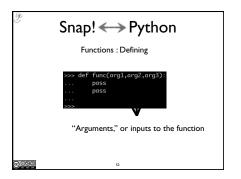


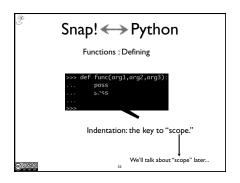


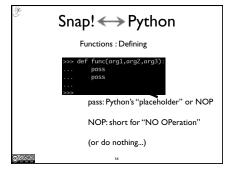


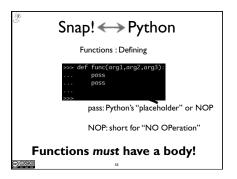


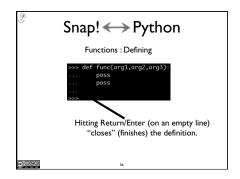


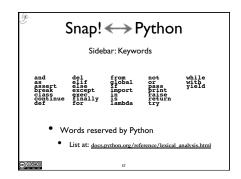


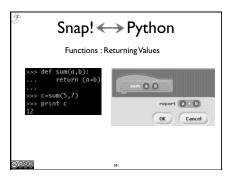


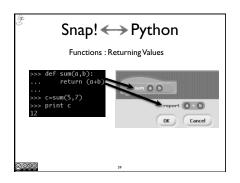


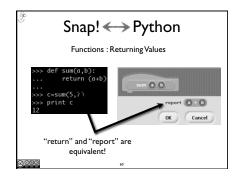












```
Snap! Python

Functions: Returning Values

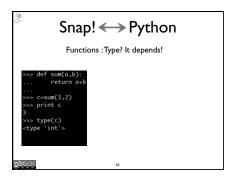
>>> def sum(a,b):
    return (a-b)

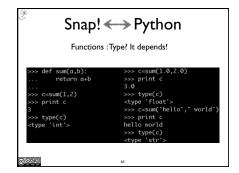
>>> c=sum(5,7)

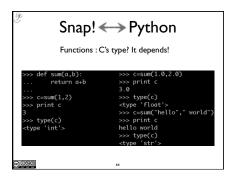
>>> print c

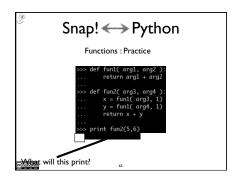
12

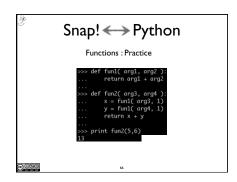
What is the type of the variable 'c'?
```

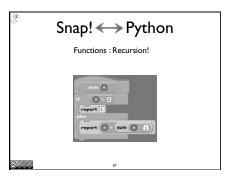


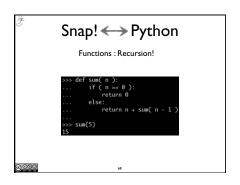


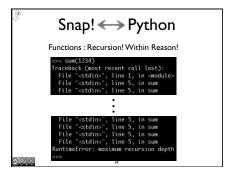


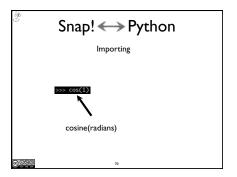


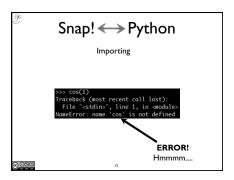


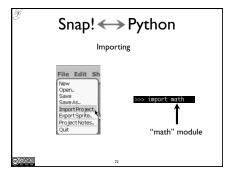


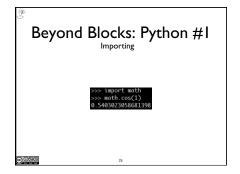


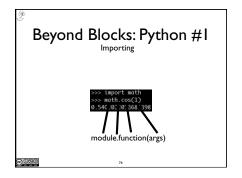


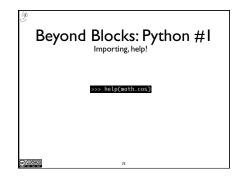


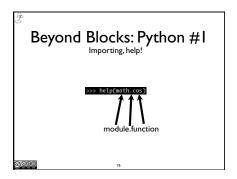


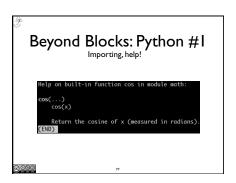


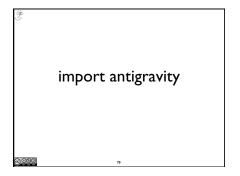


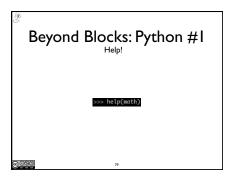


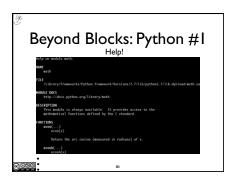


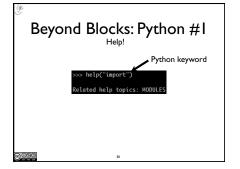


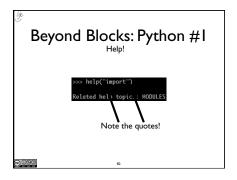


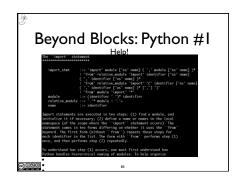


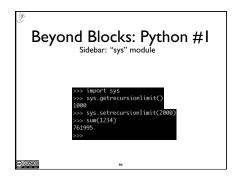












from turtle import *
color('red', 'yellow')
begin_fill()
while True:
 forward(200)
 left(170)
 if abs(pos()) < 1:
 break
end_fill()
done()</pre>

Beyond Blocks: Python #I

More Information

*Python.org: www.python.org

*Python Docs: www.python.org/doc/

*Python Modules: docs.python.org/modindex.html

Beyond Blocks: Python #I

More Information

*Computer Science Circles: Python
cemclinux I.math.uwaterloo.cal~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/