Jonathan McKinsey

The Beauty and Joy of Computing

Lecture #18
Besides Blocks I: Intro to Python

How do you feel about learning Python?

a) Very Excited. I am ready for something new.
b) Sort of Excited.
c) I don't feel strongly either way.
d) Sort of dreading it.
e) Dread. I just got used to Snap!

McKinsey

Why Learn Python?

The Goals of Beauty and Joy of Computing (BJC)

- BJC's goal is not to teach you about a specific programming language, but to teach you about:
  - critical thinking about social implications of computing
  - how to program and help you succeed in the future
  - how to think like a computer scientist also known as computational thinking

McKinsey

What is Computational Thinking?

- Using abstraction
  - removing detail
  - generalization
- Understanding the value of a "specification" that defines a contract
- The iterative design cycle: design, proof-of-concept, prototype, test, repeat
- Thinking about how solutions scale and trying to foresee the unintended consequences!

McKinsey

Why Learn Python?

- Python runs everywhere
  - Operating Systems (OS X, Windows, Linux, iOS, Android)
  - Websites
- Large user community and online support
- Plenty of advanced libraries
  - Everything from graphics processing to AI to games!
- Used in industry and academia

What You'll Learn

- New syntax
  - Different way to express algorithms
- A little bit about the command line
  - A text-based interface that exposes you to the internals of how a computer works
- (next time) A little more about Object Oriented Programming

McKinsey

Intro to Python

Getting Python 3

- We'll be using Python 3 for this class.
  - Python 3 is not backwards-compatible with Python 2.
  - For this class, you don't need to worry about the differences between Python 2 and 3.
- Download Python 3 from
  - https://python.org/downloads
  - Run the graphical installer
- All official Python documentation is at
  - https://docs.python.org

McKinsey

http://www.wired.com/2012/07/command-line/

Intro to the Command Line

- Naming
  - *Terminal* on OS X and Linux
  - *Command Prompt* on Windows
- History
  - Proceeded the graphical user interface (GUI)
- How to Open:
  - OS X - Open: /Applications/Utilities/Terminal.app
  - Windows - Use the search bar to look for "Command Prompt"

OS X/Linux	Windows
Terminal	Command Line

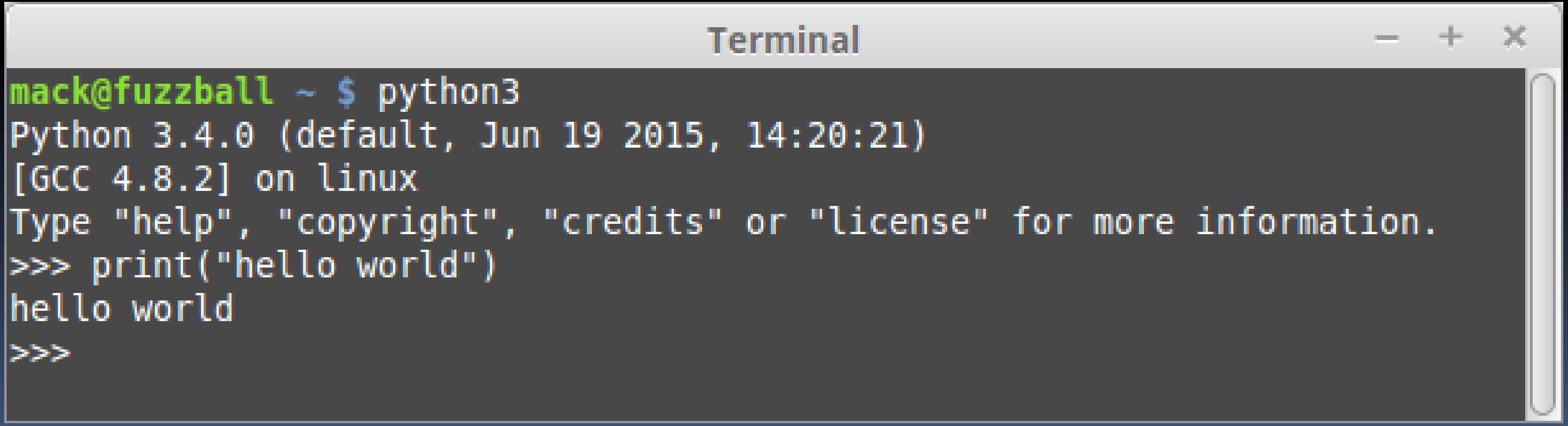
McKinsey

Python Programs

- Python programs are just a text file with Python syntax.
- To run a program you type:
  - `python3  file_name.py`
  - (See table for variations)

	OS X/Linux	Windows
Python 2	`python`	`py -2`
Python 3	`python3`	`py -3`

- Python has two modes – *normal* and *interactive*
  - Interactive mode happens if you don't provide a file to run.
  - After each command Python evaluates your code and returns the response.
    - Kind of like clicking a block in Snap*!*



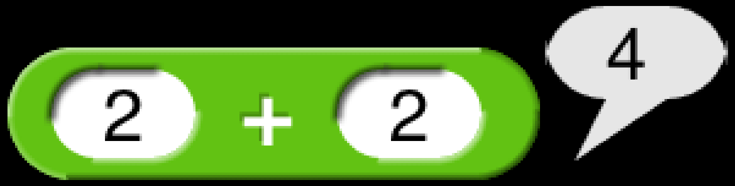
```
mack@fuzzball ~ $ python3
Python 3.4.0 (default, Jun 19 2015, 14:20:21)
[GCC 4.8.2] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("hello world")
hello world
>>>
```

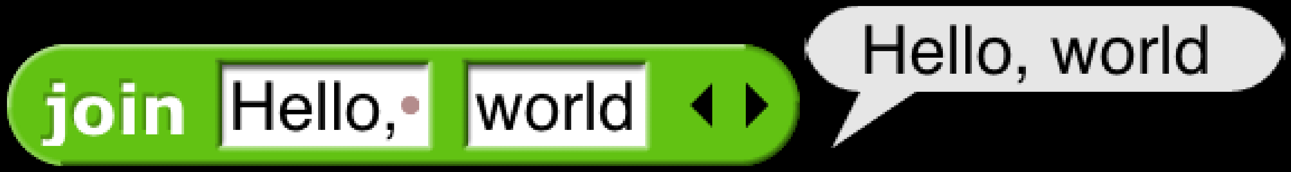
McKinsey

Snap*!* *to* Python

Text and Numbers

- Numbers in Python are called
  - ints (numbers w/o decimals)
  - floats (numbers with decimals)
- Strings:
  - Some text in between quotes “” or ‘’





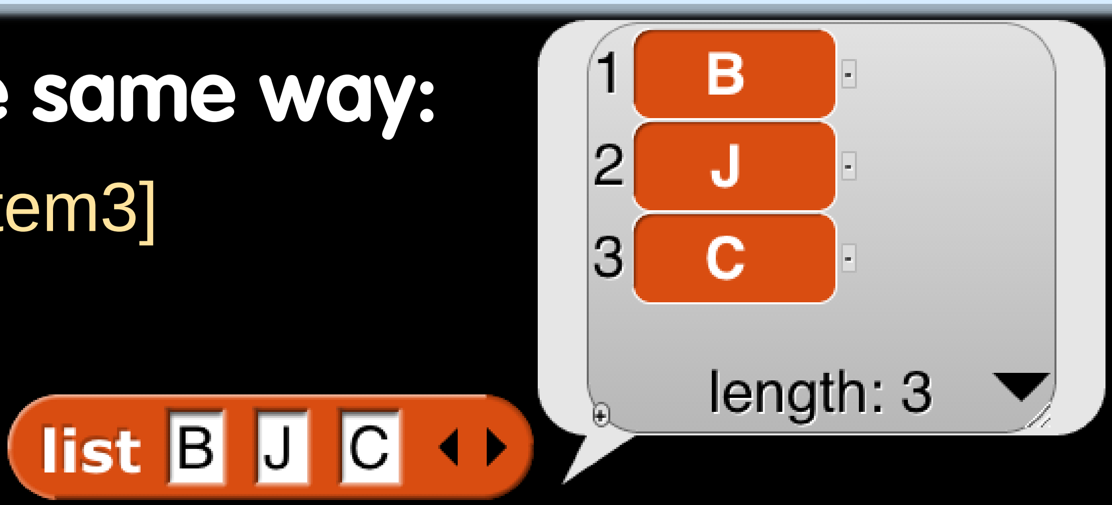
```
>>> 2 + 2
4
```

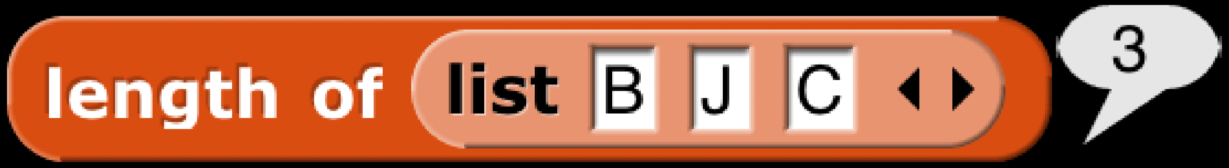
```
>>> "Hello, " + "world"
'Hello, world'
```

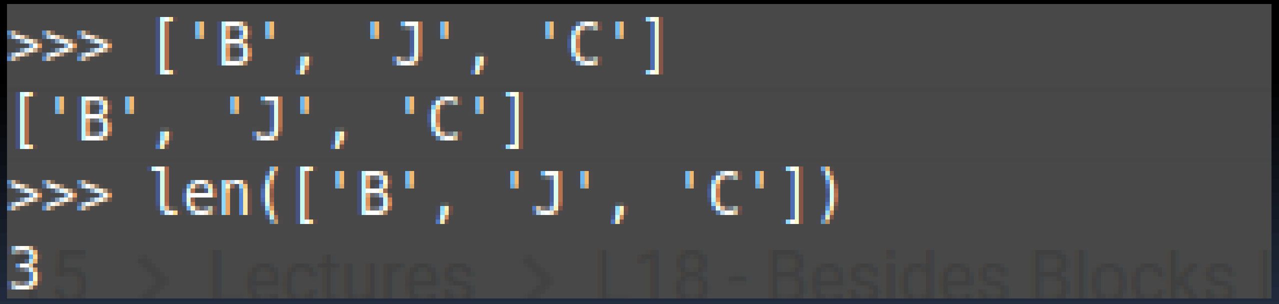
McKinsey

Lists

- **Lists Work in much the same way:**
  - **Syntax**: [item1, item2, item3]



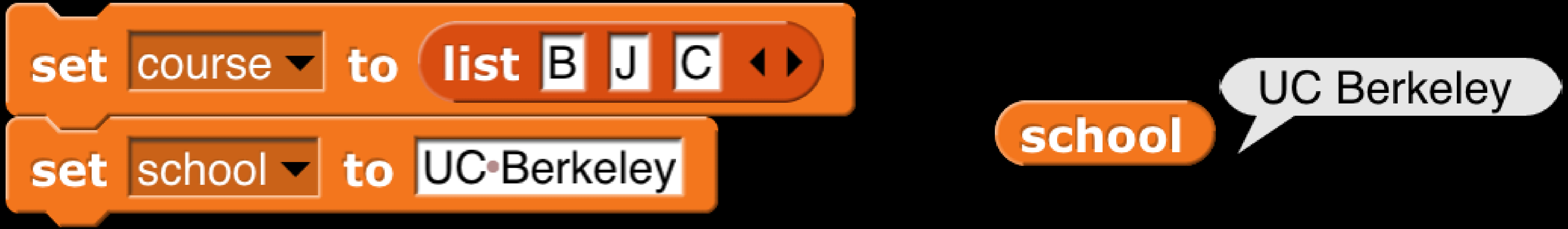


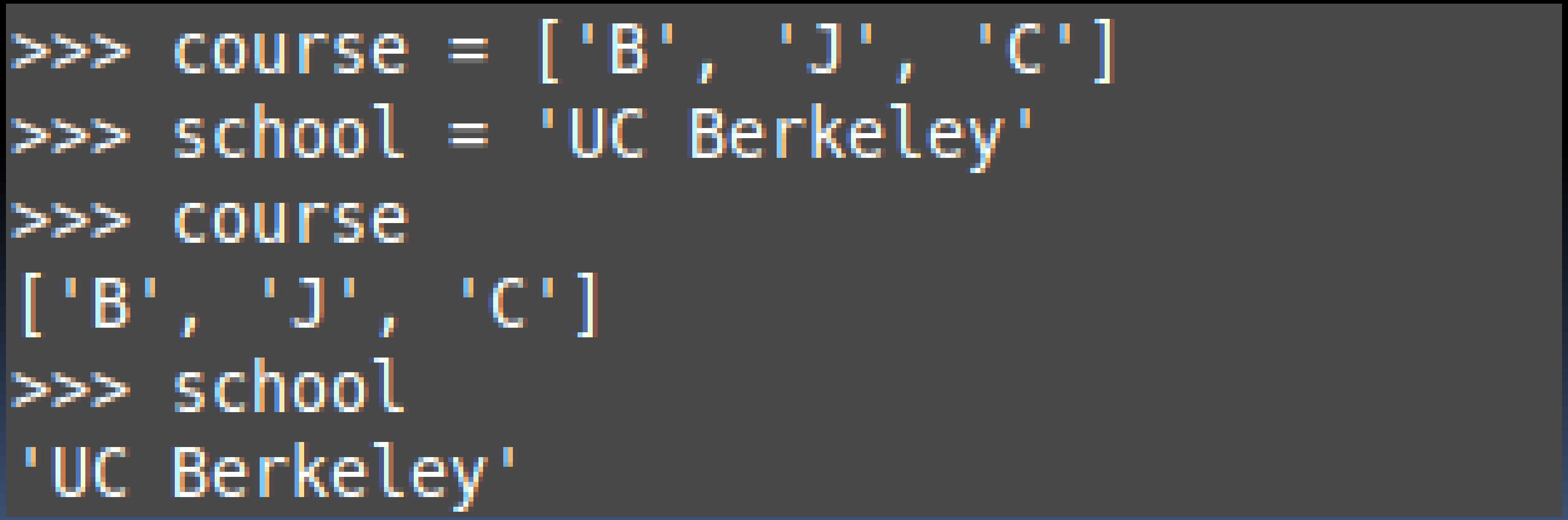


```
>>> ['B', 'J', 'C']
['B', 'J', 'C']
>>> len(['B', 'J', 'C'])
3
```

Variables

- No need to "declare" variable in Python,
  - Just use = for assignment
  - To access a variable, type its name

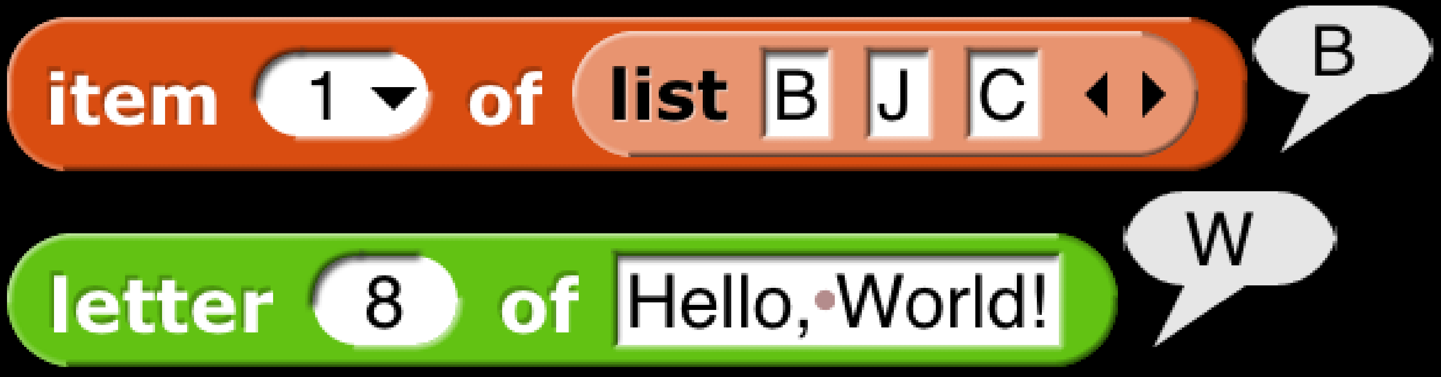


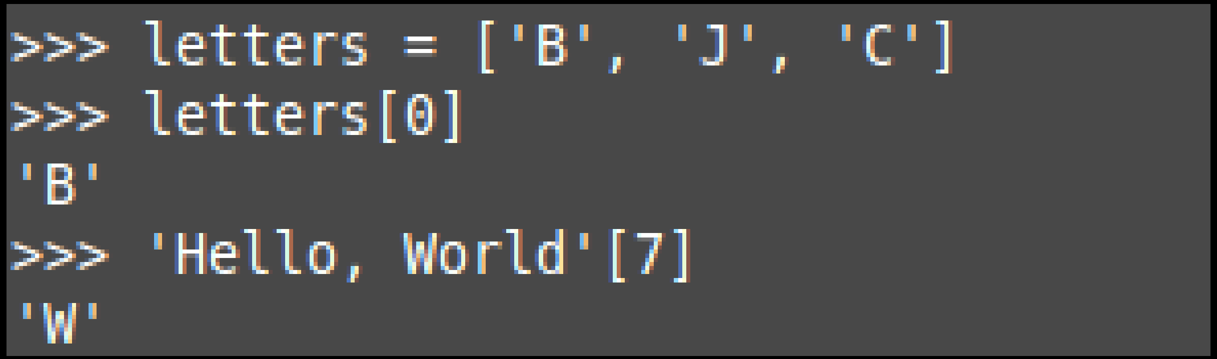


```
>>> course = ['B', 'J', 'C']
>>> school = 'UC Berkeley'
>>> course
['B', 'J', 'C']
>>> school
'UC Berkeley'
```

Zero-based Versus One-based Indexing

- Python is zero-based
  - when indexing, the first index is 0.
- Snap! Is one-based.
  - when indexing, the first index is 1.
- Access items using [#]

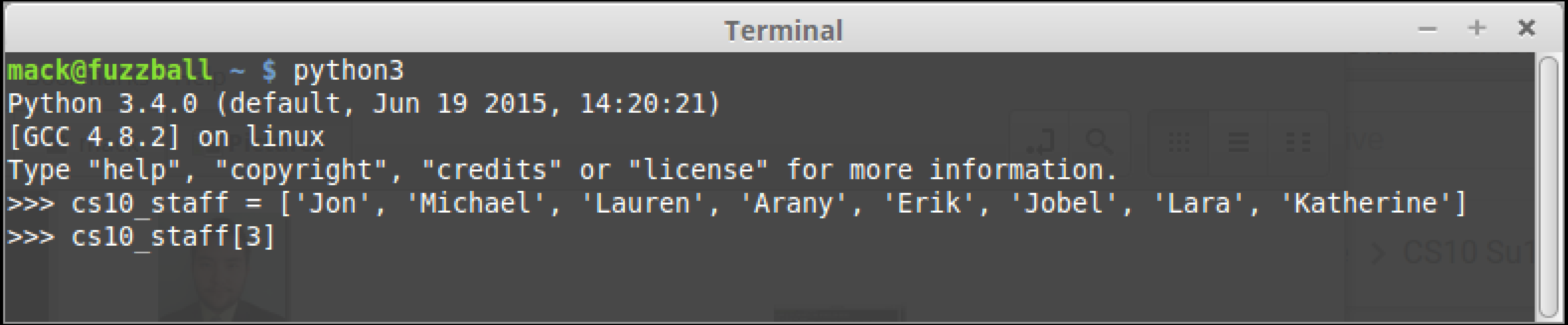




```
>>> letters = ['B', 'J', 'C']
>>> letters[0]
'B'
>>> 'Hello, World'[7]
'W'
```

McKinsey

What is the output?



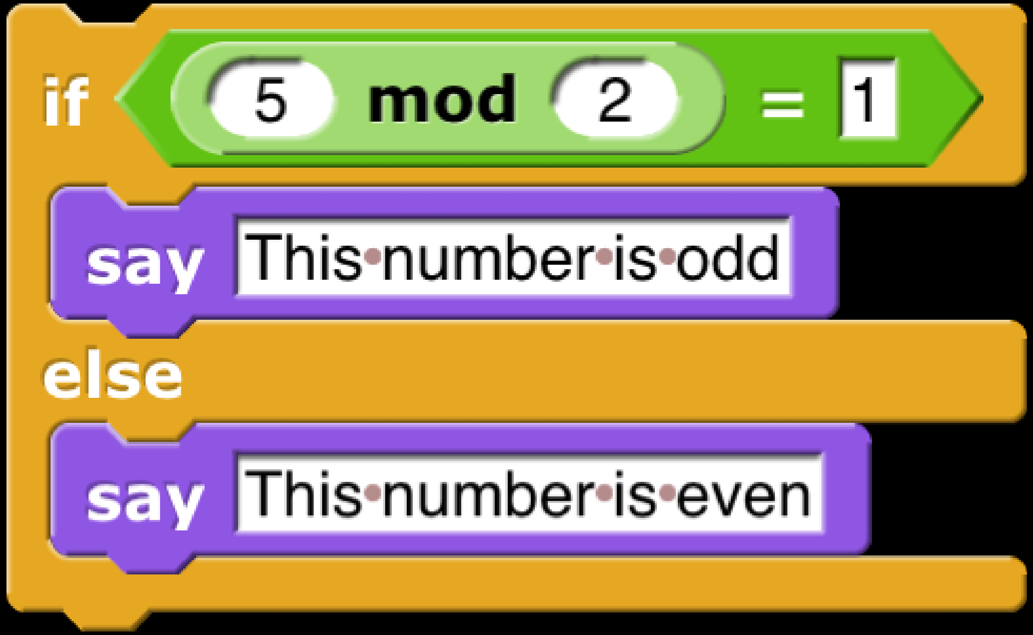
```
mack@fuzzball ~ $ python3
Python 3.4.0 (default, Jun 19 2015, 14:20:21)
[GCC 4.8.2] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> cs10_staff = ['Jon', 'Michael', 'Lauren', 'Arany', 'Erik', 'Jobel', 'Lara', 'Katherine']
>>> cs10_staff[3]
```

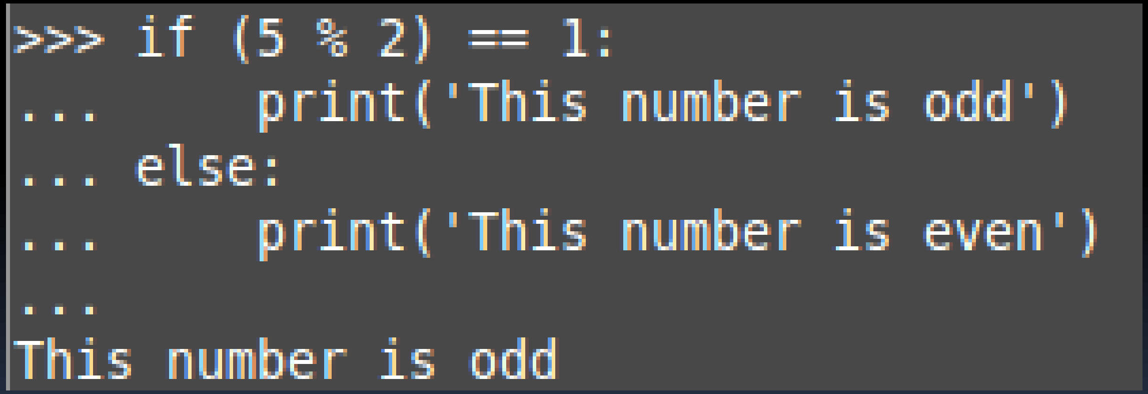
a) Michael
b) Lauren
c) Arany
d) Erik
e) Jobel

McKinsey

Conditionals

- Syntax:
  - End the condition with :
    - Parentheses are optional around the condition.
  - Indent the body one "level" (usually 4 spaces)
    - Indentation matters in Python!
  - To end a condition, just un-indent your code
- mod in Python is a %
- equivalence check is ==
- Python also supports an if (without the else) just like Snap*!*



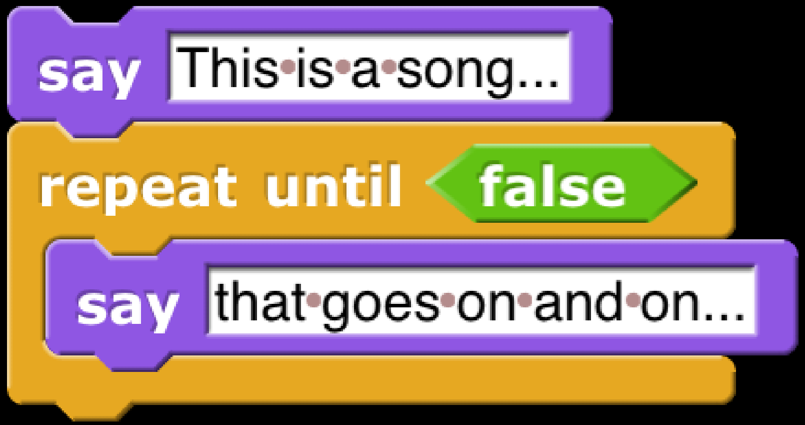


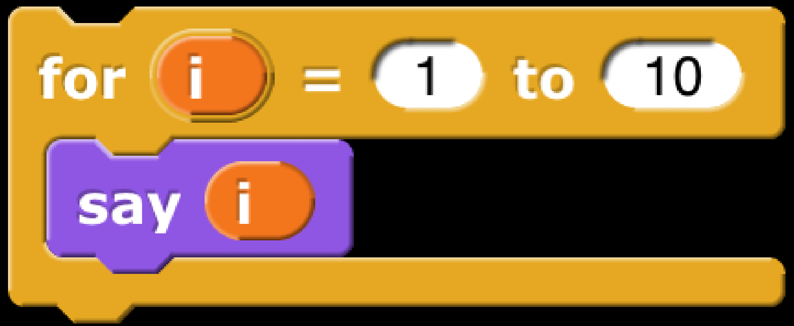
```
>>> if (5 % 2) == 1:
...     print('This number is odd')
... else:
...     print('This number is even')
...
This number is odd
```

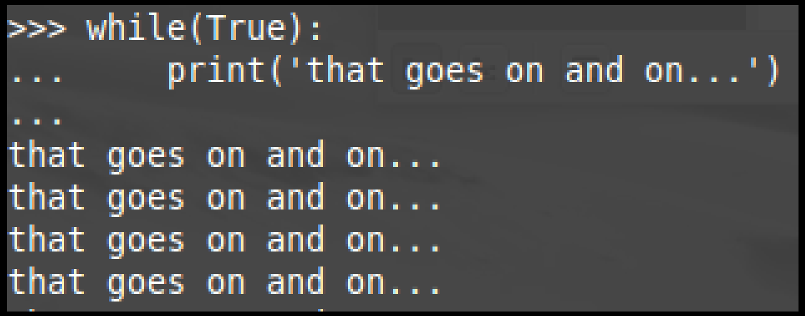
McKinsey

Loops

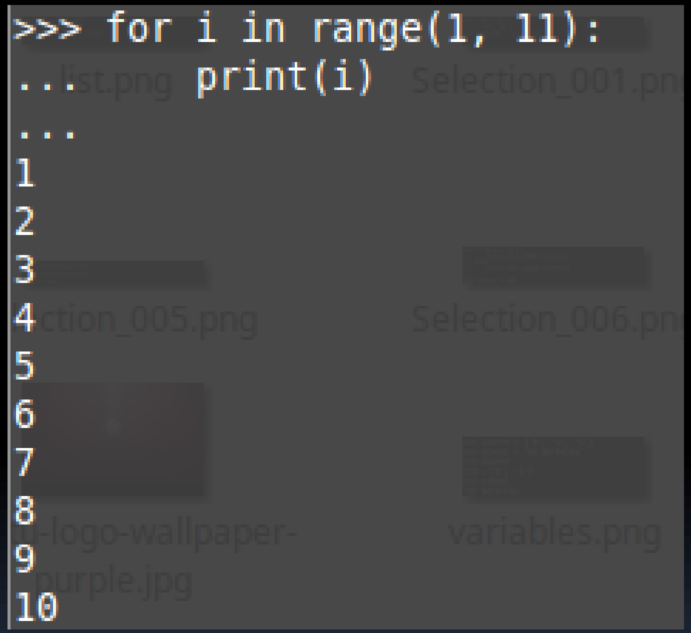
- Instead of a `repeat until` loop, Python has a `while` loop.
- Python is missing the `repeat` loop and the `forever` loop, but you can make these with while and for loops.
- Note: `range()` is a built-in function which is includes the first item, but not the last!
  - `range(1,11)` counts from 1 to 10!







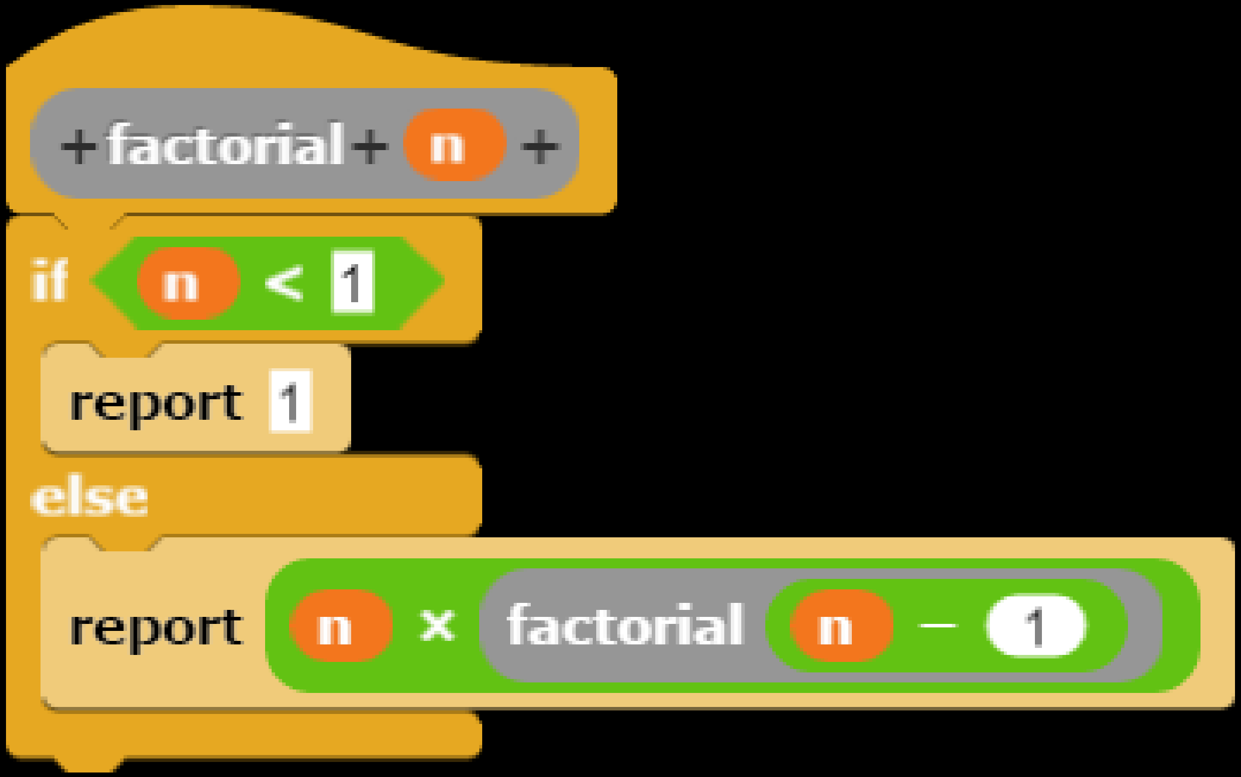
```
>>> while(True):
...     print('that goes on and on...')
...
that goes on and on...
that goes on and on...
that goes on and on...
that goes on and on...
```

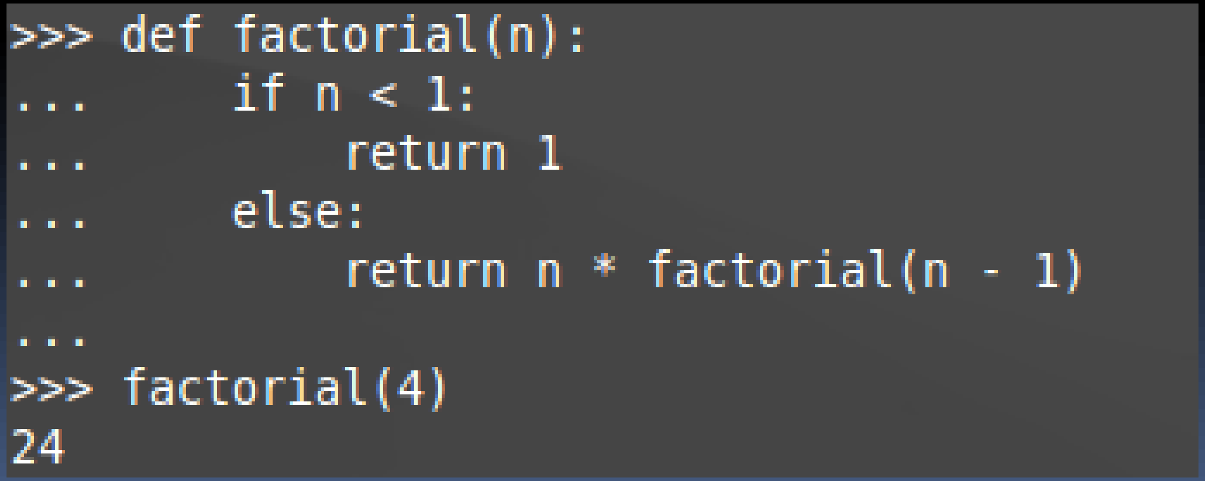


```
>>> for i in range(1, 11):
...     print(i)
...
1
2
3
4
5
6
7
8
9
10
```

Functions

- There is no distinction between a command, reporter or predicate.
  - You can simply use: `return None` or just `return`
- Python uses the word `def`
- The **body** of function is indented
- All **arguments** are specified in () and must come at the end of the function name
- `report` = `return`
- Recursion works exactly the same as in Snap*!*
- Call a function like this:
  name(arg1, arg2..)





```
>>> def factorial(n):
...     if n < 1:
...         return 1
...     else:
...         return n * factorial(n - 1)
...
>>> factorial(4)
24
```

McKinsey

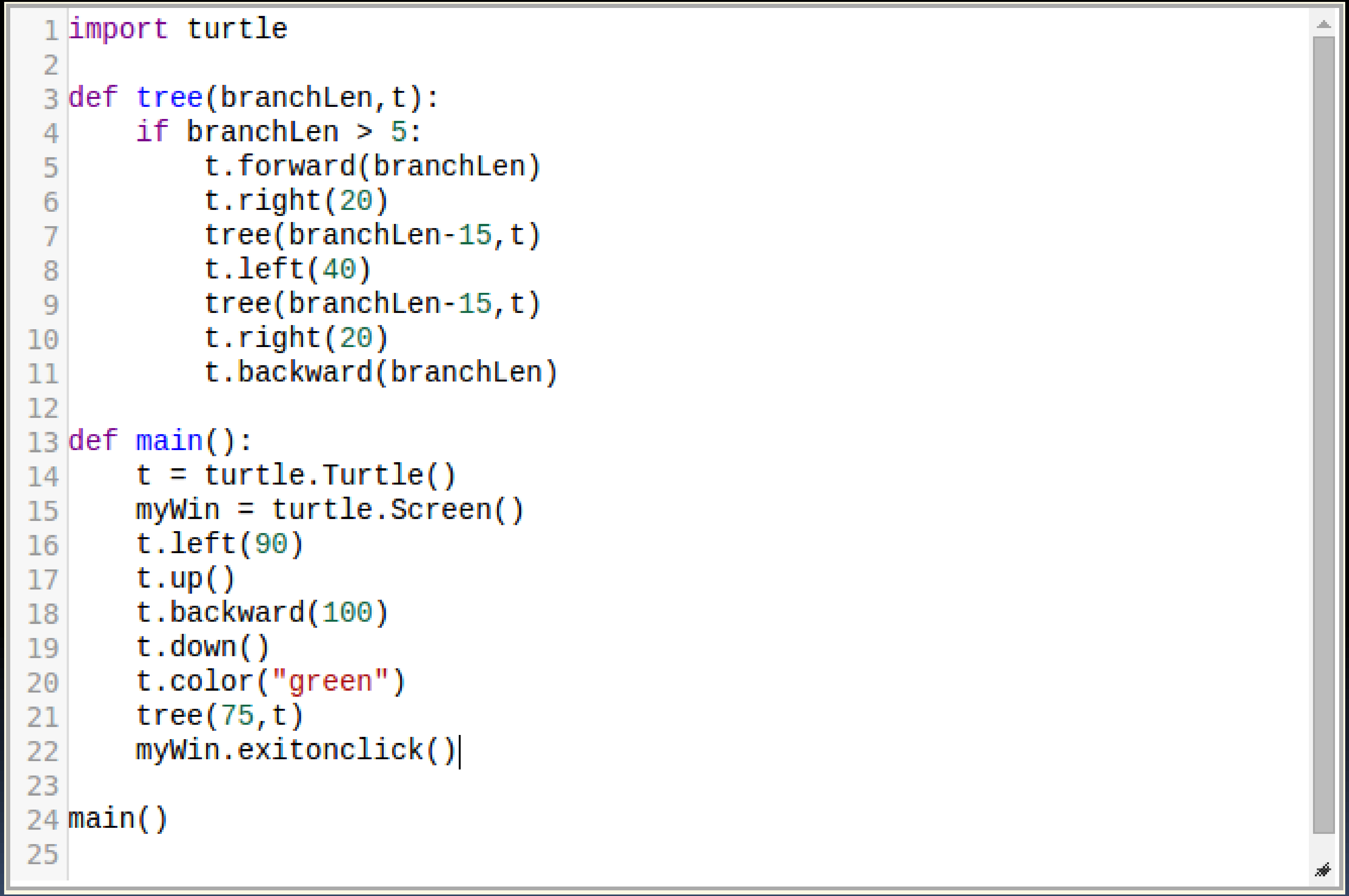
Summary

- Lots of little syntax differences!
  - The Python documentation is your friend
- Don't get too hung up on the differences and don't get discouraged when you get an error!
- There's so much more to Python in the coming weeks:
  - Python has thousands of additional, useful built in tools
  - Python supports HOFs and lambdas
  - Lots of cool libraries to explore (including turtle graphics)

McKinsey

http://interactivepython.org/runestone/static/pythonds/Recursion/graphical.html

Preview



```python
import turtle

def tree(branchLen,t):
    if branchLen > 5:
        t.forward(branchLen)
        t.right(20)
        tree(branchLen-15,t)
        t.left(40)
        tree(branchLen-15,t)
        t.right(20)
        t.backward(branchLen)

def main():
    t = turtle.Turtle()
    myWin = turtle.Screen()
    t.left(90)
    t.up()
    t.backward(100)
    t.down()
    t.color("green")
    tree(75,t)
    myWin.exitonclick()

main()

```

McKinsey