Python 1: Intro!

Max Dougherty

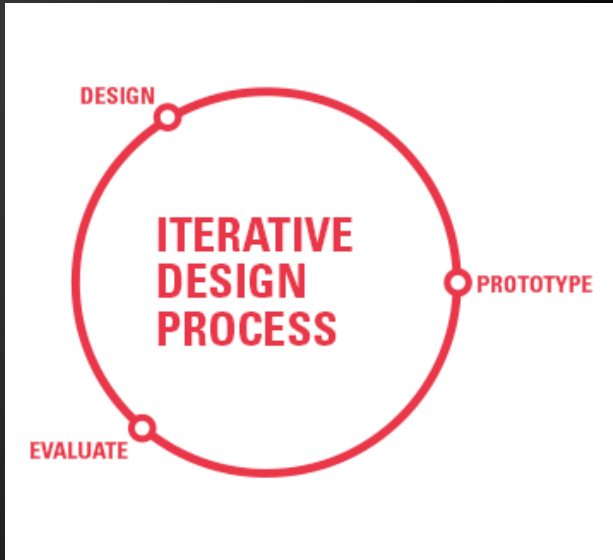
Andrew Schmitt

Computational Thinking

- Two factors of programming:
  - The conceptual solution to a problem.
  - Solution syntax in a programming language
- BJC tries to isolate and strengthen the first.
  - Snap! helps us (more or less) remove the worry of syntax
- Our goal is not to teach Snap!, but instead to teach “computational thinking”

Computational Thinking is:

- using abstraction to generalize problems.
- logical analysis of data
- identification, implementation, and testing of possible solutions
  - a.k.a The Iterative Design Process



Why learn Python?

- Python is learn and use
  - Looks like pseudo-code!
  - Quickly implement programs
- Widely used as a teaching tool
  - Tons of online support
- Powerful and Fast, with hundreds of community supported code libraries.

```
def fact(n):
    if (n < 1):
        return 1
    else:
        return n*fact(n-1)
```

(Familiar?)

Python in the World

- Parts of Google web search are written in Python.
- Battlefield 2 used python for core functions.
- Walt Disney uses python for animation tools and scripts.

Getting Started: Opening Interpreter

(on Mac OSX)

- Open the Command Line in Terminal
  - The Command Line is a text based computer interface
- Type **`python3`** and press **`return`**.

```
Maxs-MacBook-Pro-3:~ iMax$ python3
Python 3.4.0 (v3.4.0:04f714765c13, Mar 15 2014, 23:02:41)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Aside: We will be using python version 3.4 in this course.

Getting Started: “Hello World”

- The “>>>” at the bottom indicates that you are in the interpreter
- Type **`print(“Hello World”)`** and press the return key.
- Putting quotes around “Hello World” turns it into a string.
- Print returns a string

```
>>>
```

```
>>> print("Hello World")
Hello World
```

From Snap! to Python: Variables


```
>>> foo = 5
>>>
```

(Notice how, unlike **`print`**, assigning a value to `foo` does not return anything)

From Snap! to Python: Variabless


```
>>> foo = 5
>>> foo
5
>>>
```

From Snap! to Python: Operators


```
>>> 2 + 4
6
```


```
>>> 5 - 3
2
```


```
>>> 3 * 4
12
```


```
>>> 5 / 2
2.5
```

Aside:
>>> 5 // 2
2 ← Rounds down

From Snap! to Python: Operatorss


```
>>> 2 < 4
True
```


```
>>> 2 > 4
False
```


```
>>> 3 == 3
True
```

From Snap! to Python: Operatorsss


```
>>> (2 < 3) and (3 < 4)
True
```


```
>>> (5 == 4) or (7 > 6)
True
```


```
>>> not (3 == 3)
True
```

From Snap! to Python: Operatorssss


```
>>> 5 % 3
2
```


```
>>> “Hello” + “World”
HelloWorld
```


```
>>> len(“fish”)
4
```

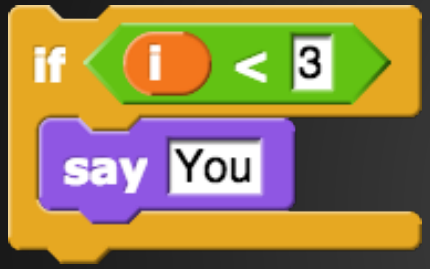

```
>>> “world”[0]
‘w’
```

New Python Function!: Substring

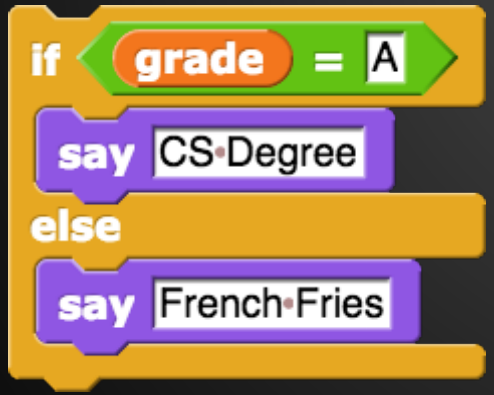
- Substring returns part of a string
- the “:” is used to separate the first and separate index
- If no number exists on either side of colon, the substring will extend as far as possible.

```
>>> name = “Alonzo”
>>> name[1:4]
‘lon’
>>> name[2:]
‘onzo’
>>> name[:3]
‘Alo’
```

From Snap! to Python: Conditionals




```
if (i < 3):
   print("You")
```




```
if (grade == "A"):
   print("CS Degree")
else:
   print("French Fries")
```

From Snap! to Python: Conditionalss

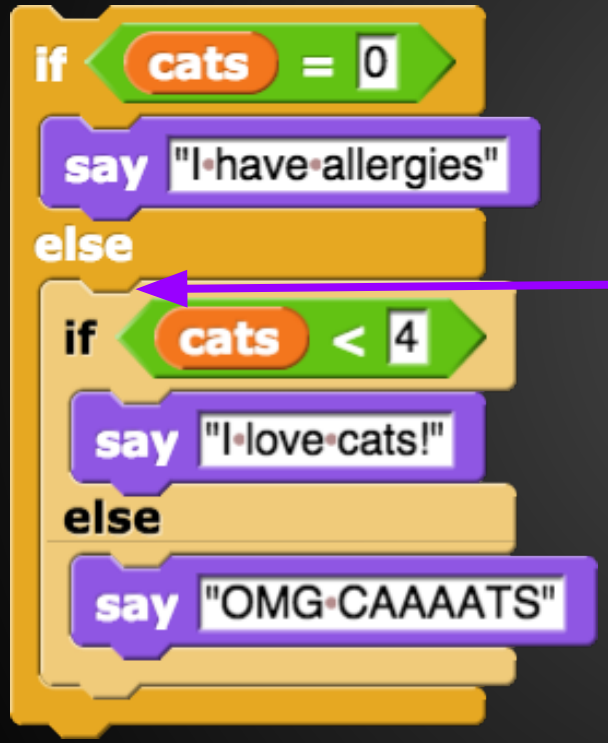
The indentation of lines is super important!

```
if (i < 3):
    print("You")
```

```
if (grade == "A"):
    print("CS Degree")
else:
    print("French Fries")
```

The colon tells Python that any indented lines that follow are inside the “if” or “if/else” condition

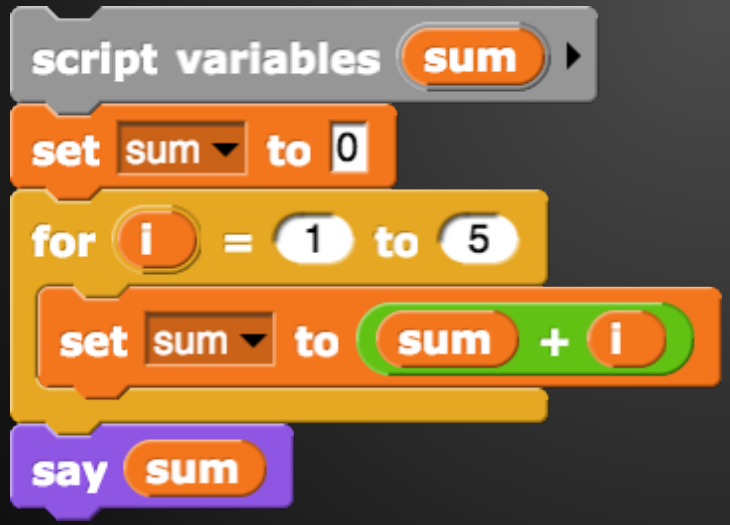
From Snap! to Python: Conditionalsss




```
if (cats == 0):
   print("I have allergies!")
elif (cats < 4):
   print("I love cats!")
else:
   print("OMG CAAAATS")
```

From Snap! to Python: Loops

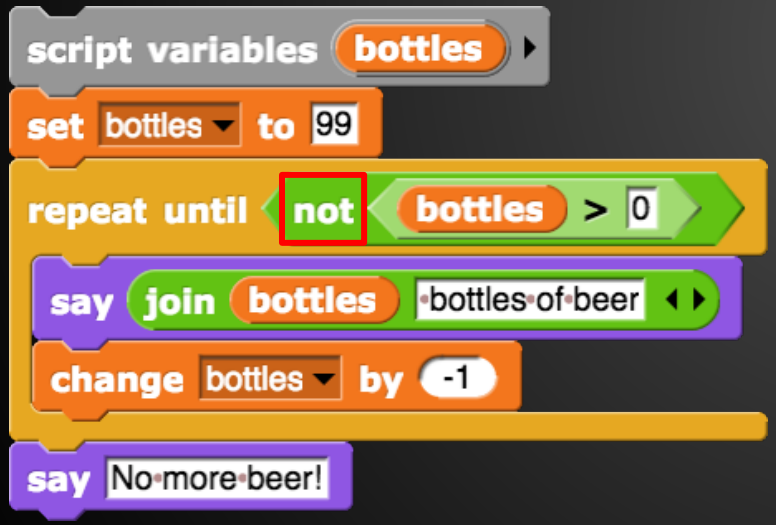
- Python “for” requires a **list of values** instead of a start and end value.
  - This list is more properly known as an “iterable”
- Range(1,5) returns the list [1,2,3,4]
  - Notice how 5 is NOT in the list!




```
sum = 0
for i in range(1,6):
    sum = sum + i
print(sum)
```

From Snap! to Python: Loopss

- Python “while” is similar to repeat until.
  - Difference: “Repeat until” ends on TRUE, “while” ends on FALSE
- Can both cause infinite loops if the index (bottles) is not updated.




```
num = 99
while bottles > 0:
    print(num +“ bottles of beer”)
    bottles = bottles - 1
print(“No more beer”)
```

Snap! to Python: Calling Functions

- Calling a function requires a name **`func`** and the required comma separated arguments in parentheses **`(arg1, arg1)`**


```
>>> func("foo", "bar")
CS Rules!
>>>
```

Snap! to Python: Defining Functions

- Instead of using a pop-up window like Snap!, python functions are defined in text.
  - prefaced by the keyword, **`def`**


```
def func(arg1, arg2):
    if (arg1 < arg2):
        return "Arg! I'm a Pirate"
    else:
        return "CS Rules!"
```

- Uses same indentation system as loops and conditionals (colons and indentation)

Snap! to Python: Defining Functions

Function Name

Arguments

```
def func(arg1, arg2):
    if (arg1 < arg2):
        return "Arg! I'm a Pirate"
    else:
        return "CS Rules!"
```

`return` works exactly like Snap! "report"

Note: return does not require parentheses

Types in Python

- Functions in Python have **Dynamic type**
  - Simply: A function can return any type of data!
- `type` is a Python function that returns the object's class

Integer:

```
>>> output = sum(2,3)
>>> type(output)
<class 'int'>
```

```
def sum(x, y):
    return x + y
```

Floating point number:

```
>>> output = sum(3.14, 2.718)
>>> type(output)
<class 'float'>
```

Aside: Floating Point Numbers

$$3.14_{10} = ???_{2}$$

- Doesn’t look so pretty for a programmer.
- Computer Scientists decided to use an alternate representation for non-integer values
  - Represent numbers to great accuracy
- Not responsible for knowing how floating point numbers work in computers

Types in Python

- Functions in Python have **Dynamic type**
  - Simply: A function can return any type of data!
- `type` is a Python function that returns the object's class

```
def sum(x, y):
    return x + y
```

Integer:

```
>>> output = sum(2,3)
>>> type(output)
<class 'int'>
```

Floating point number:

```
>>> output = sum(3.14, 2.718)
>>> type(output)
<class 'float'>
```

String:

```
>>> output = sum("cat", "dog")
>>> type(output)
<class 'str'>
```

Snap! to Python: Importing

- Importing tools in Snap! gave us a more advanced set of functions
- Python also allows importing, but from a much, much larger library

```
>>> import math
>>>
```

Importing in Python

```
>>> sin(0.5)  ← # standard Python doesn't know what 'sin' is
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'sin' is not defined

>>> import math ← # adds tons of functions for math operations
>>> math.sin(0.5)
0.479425538604203
```

Python Demo: Turtle Power!

```
import turtle
t = turtle.Turtle()

turtle.showturtle()
turtle.pendown()
recursive_tree(5, 40)
```

Thank You!

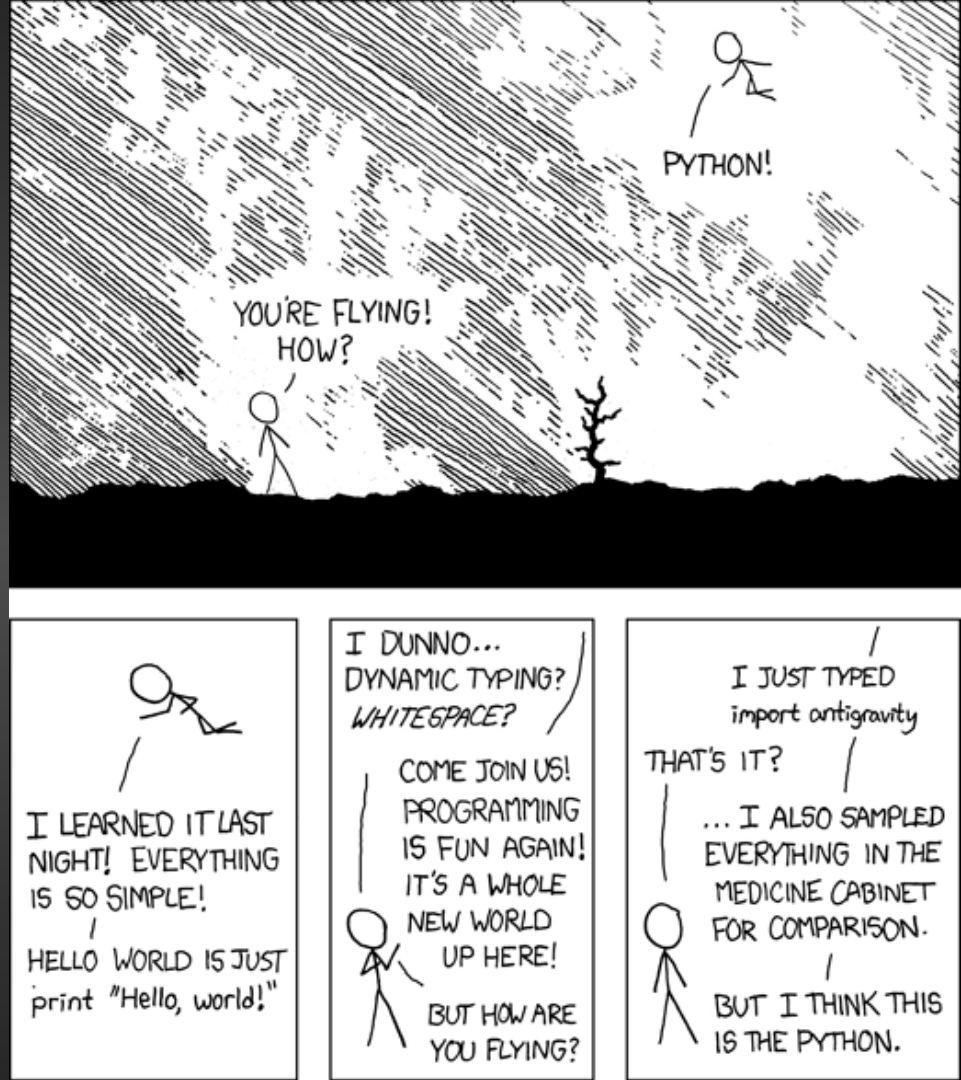


Photo at xkcd.com/353