DSC 201: Data Analysis & Visualization

Python Loops and Containers

Dr. David Koop
Python

• Started in December 1989 by Guido van Rossum
• “Python has surpassed Java as the top language used to introduce U.S. students to programming…” (ComputerWorld, 2014)
• Python and R are the two top languages for data science
• High-level, interpreted language
• Supports multiple paradigms (OOP, procedural, imperative)
• Help programmers write readable code
• Use less code to do more
• Lots of libraries for python
  - Designed to be extensible
  - Easy to wrap code from other languages like C/C++
• Open-source with a large, passionate community
Python Compared to Java

• Dynamic Typing
  - A variable does not have a fixed type
  - Example: \( a = 1; \ a = \text{“abc”} \)

• Indentation
  - Braces define blocks in Java, good style is to indent but not required
  - Indentation is critical in Python
  - Example:
    
    ```python
    z = 20
    if x > 0:
        if y > 0:
            z = 100
        else:
            z = 10
    ```
Jupyter Notebook

- An interactive programming environment
- Runs in your web browser…
- … but does not require an internet connection
- Displays results (even interactive maps) inline
- Originally designed for Python
- Supports other languages, too
- You decide how to divide code into executable cells
- Shift+Enter to execute a cell
Obtaining Python (and Jupyter Notebook)

- www.anaconda.com/download/
- Anaconda has Jupyter Notebook
- Use Python 3.6 version (not 2.7)
- Anaconda Navigator
  - GUI application for managing Python environment
  - Can install packages
  - Can start Jupyter Notebook
- Can also use the shell to do this:
  - $ jupyter-notebook
  - $ conda install <pkg_name>
Python Variables and Types

- No type declaration necessary
- Variables are names, not memory locations
  
  ```python
  a = 0
  a = "abc"
  a = 3.14159
  ```

- Don't worry about types, but think about types
- Strings are a type
- Integers are as big as you want them
- Floats can hold large numbers, too (double-precision)
Python Math and String "Math"

• Standard Operators: +, -, *, /, %
• Division "does what you want" (new in v3)
  -5 / 2 = 2.5
  -5 // 2 = 2 # use // for integer division
• Shortcuts: +=, -=, *=
• No ++, --
• Exponentiation (Power): **
• Order of operations and parentheses:
  - 4 - 3 - 1
  - 4 - (3 - 1)
• "abc" + "def"
• "abc" * 3
Python Strings

- Strings can be delimited by single or double quotes
  - "abc" and 'abc' are exactly the same thing
  - Easier use of quotes in strings: "Joe's" or 'He said "Stop!"'
- String concatenation: "abc" + "def"
- Repetition: "abc" * 3
- Special characters: \n \t like Java
- Indexing:
  a = "abcdef"
  a[0]
- Slicing: a[1:3]
- Format: name = "Jane"
  print("Hello, {}".format(name))
Textbook Available

- eBook now available
- Paperback later this month
- I will be referring to this text a lot during the rest of the course
Assignment 2

- Python and Jupyter Notebook
- Dataset of MA, CT, RI towns
- Data exploration
  - Find max, min
  - Count
  - String processing
- Due next Wednesday
- Start now!
Loops

- while <condition>: 
  <indented block>
  # end of while block (indentation done)

- Remember the colon!

- a = 5
  while a > 0:
    print(a)
    a -= 2

- a > 0 is the condition

- Python has standard boolean operators (<, >, <=, >=, ==, !=)
  - What does a boolean operation return?
  - Linking boolean comparisons (and, or)
break and continue

• **break** stops the execution of the loop
• **continue** skips the rest of the loop and goes to the next iteration
Conditionals

• if, else
  - Again, indentation is required
• elif
  - Shorthand for else: if:
• Same type of boolean expressions
True and False

- True and False (captialized) are defined values in Python
- v == 0 will evaluate to either True or False
Functions

• Calling functions is as expected:
  
  `mul(2,3) # computes 2*3 (mul from operator package)`
  
  - Values passed to the function are parameters
  
  - May be variables!
    
    ```python
    a = 5
    b = 7
    mul(a,b)
    ```

• `print` is a function
  
  ```python
  print("This line doesn't end", end="")
  print("See it continues")
  ```
  
  - `end` is also a parameter, but this has a different syntax
  
  - Keyword argument!
Why do we create and use functions?
Defining Functions

• **def** keyword

• Arguments have names but **no types**
  
  ```python
  def hello(name):
      print("Hello {}".format(name))
  ```

• Can have defaults:
  
  ```python
  def hello(name="Jane Doe"):
      print("Hello {}".format(name))
  ```

• With defaults, we can skip the parameter
  
  ```python
  hello() or hello("John")
  ```

• Also can pick and choose arguments:
  
  ```python
  def hello(name1=\"Joe\", name2=\"Jane\"):
      print("Hello {} and {}".format(name1, name2))
  hello(name2=\"Mary\")
  ```
Return statement

• Return statement gives back a value:

```python
def mul(a, b):
    return a * b
```

• Variables changed in the function won't be updated:

```python
def increment(a):
    a += 1
    return a

b = 12
b = increment(b)
c = increment(b)
print(b, c)
```
Quiz

- Suppose I want to write Python code to print the numbers from 1 to 100. What errors do you see?

```python
// print the numbers from 1 to 100
int counter = 1
while counter < 100 {
    print counter
    counter++
}
```
Python Containers

- Container: store more than one value
- Mutable versus immutable: Can we update the container?
  - Yes \(\rightarrow\) mutable
  - No \(\rightarrow\) immutable
  - Lists are mutable, tuples are immutable
- Lists and tuples may contain values of different types:
  - List: \([1, "abc", 12.34]\)
  - Tuple: \((1, "abc", 12.34)\)
- You can also put functions in containers!
- `len` function: number of items: `len(l)`
Indexing and Slicing

• Just like with strings

• Indexing:
  - Where do we start counting?
  - Use brackets [] to retrieve one value
  - Can use negative values (count from the end)

• Slicing:
  - Use brackets plus a colon to retrieve multiple values:
    \[ \text{[<start>:<end>]} \]
  - Returns a \textbf{new} list \( b = a[:,] \)
  - Don't need to specify the beginning or end
  - Can add a second colon to specify the increment
    \[ \text{[<start>:<end>:<step>] \} } \]
Examples

• Suppose \( a = [\text{`}a\text{', }\text{`}b\text{', }\text{`}c\text{', }\text{`}d\text{'}] \)

• What are?
  - \(a[0]\)
  - \(a[1:2]\)
  - \(a[3:]\)
  - \(a[::-2]\)
  - \(a[:::-1]\)
Tuples

• months = ('January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October', 'November', 'December')

• Useful when you know you're not going to change the contents or add or delete values

• Can index and slice

• Also, can create new tuples from existing ones:
  - t = (1, 2, 3)
  - u = (4, 5, 6)
  - v = t + u
Modifying Lists

- **Add to a list l:**
  - `l.append(v)`: add one value \(v\) to the end of the list
  - `l.extend(vlist)`: add multiple values \(vlist\) to the end of \(l\)
  - `l.insert(i, v)`: add one value \(v\) at index \(i\)

- **Remove from a list l:**
  - `del l[i]`: deletes the value at index \(i\)
  - `l.pop(i)`: removes the value at index \(i\) (and returns it)
  - `l.remove(v)`: removes the first occurrence of value \(v\) (careful!)

- **Changing an entry:**
  - `l[i] = v`: changes the value at index \(i\) to \(v\)
  - Watch out for `IndexError`
Modifying Lists

• Also can create new lists by "addition" as with tuples:
  
  - a = [1, 2, 3]; b = [4, 5, 6]
    c = a + b
  
  - Does not work: c = a + 4
in: Checking for a value

• The `in` operator:
  - 'a' in l
  - 'a' not in l

• Not very fast for lists