Data Science

Python Review

Chris Jermaine, Risa Myers, Marmar Orooji

Rice University

Python

- Old language, first appeared in 1991
  - But updated often over the years
- Important characteristics
  - Interpreted
  - Dynamically-typed
  - High level
  - Multi-paradigm (imperative, functional, OO)
  - Generally compact, readable, easy-to-use
  - Everything is an object
    - Scalar (indivisible): int, float, etc.
    - Non-scalar (having internal structure): list, dictionary, ...
- Boom in popularity recently
  - Now the first programming language learned in many CS departments

Python for data science

- Dynamic typing/interpreted
  - Type a command, get a result
  - No need for compile/execute/debug cycle
- Quite high-level: easy for non-CS people to pick up
  - Statisticians, mathematicians, physicists...
- More of a general-purpose programming language than R
  - More reasonable target for larger applications
  - More reasonable as API for platforms such as Spark
- Can be used as lightweight wrapper on efficient numerical codes

Python basics

- Since Python is interpreted, can just fire up Python shell, ipython, or a Notebook
- Then start typing

Python basics continued

- Spacing and indentation
  - Indentation is important
  - No begin/end nor {}
  - Indentation signals code block
- Variables
  - No declaration
  - All type checking is dynamic
  - Just use them

Python basics: List

- Ordered sequence of objects
- Mutable object
- Example: `myList = [ ]` creates an empty list
- Useful methods
  - `myList.append(x)`
  - `myList.insert(index, x)` inserts x before position *index*
  - `myList.remove(x)` removes first item in `myList` whose value is x
  - iterator

```
for x in myList:
    print(x)
```

Poll

- Is `myList = ['a', 1, True]` a valid Python list?

Python basics: Dictionary

- Think of as a list, indexed by key
- Mutable object
- Example: `wordsInDoc = {}` creates an empty dictionary
- Adding Data
  - Add data by saying `wordsInDoc[23] = 16`
  - Now can write something like `if wordsInDoc[23] == 16:` ...
  - ? What if `wordsInDoc[23]` is not there?

Python basics: Dictionary

- What if `wordsInDoc[23]` is not there?
- Error!
- Prevent with `wordsInDoc.get(23, 0)`
- Returns `0` if key `23` is not defined

? When might we choose to use this type of statement?

Functions / procedures

- Use for decomposition
  - Breaking a problem down into smaller, self-contained parts
- Use for abstraction
  - Hide implementation details
- Defined using `def myFunc (arg1, arg2):`
- Procedure: no return statement
- Function: return statement
- Remember:
  - Make sure to indent!
  - No marker to end function or procedure
  - It ends when you stop indenting

Sample function

```
def Factorial (n):
     if n == 1 or n == 0:
          return 1
     else:
          return n * Factorial (n - 1)
```

- Called by

```
Factorial (12)
```

Loops

- Looping through a range of values
  - `for var in range (0, 50)`
  - Loops for `var` in `{0, 1, ..., 49}`
- Looping while a condition evaluates to True
  - `while <condition>:`
- Looping through data structures
  - Example: `for var in dataStruct`
  - Loops through each entry in `dataStruct`
  - `dataStruct` can be a list or dictionary, etc.
  - If list, you loop through the entries
  - If dictionary, you loop through the keys
- `break`
  - Exits the innermost loop
- `continue`
  - Skips the remaining statements in the loop body and goes to the next iteration of the innermost loop

Loops continued

- An example

```
>>> a = {}
>>> a[1] = 'this'
>>> a[2] = 'that'
>>> a[3] = 'other'
for b in a:
   print(a[b])

this
that
other
```

Wrap up

1. Python review

? How can we use what we learned today?

? What do we know now that we didn't know before?