DSC 201: Data Analysis & Visualization

Python Containers & Tables

Dr. David Koop
Python Math and String "Math"

- Standard Operators: +, -, *, /, %
- Division "does what you want" (new in v3)
  - \( \frac{-5}{2} = 2.5 \)
  - \( -5 \div 2 = 2 \) # 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:
  
  ```python
  a = "abcdef"
  a[0]
  ```

- Slicing: a[1:3]

- Format: name = "Jane"
  
  ```python
  print("Hello, {}".format(name))
  ```
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)
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")
  ```
Calling Functions

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

• \texttt{print} is a function
  
  \begin{verbatim}
  print("This line doesn't end", end="")
  print("See it continues")
  \end{verbatim}
  
  - \texttt{end} is also a parameter, but this has a different syntax
  - Keyword argument!
Jupyter Notebook & JupyterLab @ UMassD

- https://rps.cscvr.umassd.edu:8000/
- Just log in and everything should be setup
- For JupyterLab, click the corresponding button after you log in
- Advanced:
  - Can add your own conda environments in your user directory
Assignment 1

• **Link**

• Analyze Atlantic hurricane data

• You need to manipulate lists and strings to extract the answers

• Find:
  - Number of unique names
  - Hurricane with top sustained wind

• Code to read data is provided

• Code to check answers is provided

• Due Tuesday, Sept. 25 at 11:59pm

• **Get started!**
Python Containers

• Container: store more than one value
• Mutable versus immutable: Can we update the container?
  - Yes → mutable
  - No → 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:
    
    [<start>:<end>]

  - Returns a **new** list \( b = a[::] \)
  - Don't need to specify the beginning or end
  - Can add a second colon to specify the increment
    
    [<start>:<end>::<step>]
Examples

• Suppose $a = ['a', 'b', 'c', 'd']$

• What are?
  - $a[0]$
  - $a[1:2]$
  - $a[3:]$
  - $a[:-2]$
  - $a[:::1]$
Examples

• Suppose `a = ['a', 'b', 'c', 'd']`

• What are?
  - `a[0]` # 'a'
  - `a[1:2]` # ['b']
  - `a[3:]` # ['d']
  - `a[::2]` # ['a', 'b']
  - `a[2::2]` # ['c']
  - `a[:::-1]` # ['d', 'c', 'b', 'a']
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 # v points to a new object

  - t += u # t is a new object
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 a list

• \( v = [1, 2, 3] \)
  \( w = [4, 5, 6] \)

• \( x = v + w \) # \( x \) is a **new** list \([1, 2, 3, 4, 5, 6]\)

• \( v.extend(w) \) # \( v \) is mutated to \([1, 2, 3, 4, 5, 6]\)

• \( v += w \) # \( v \) is mutated to \([1, 2, 3, 4, 5, 6]\)

• \( v.append(w) \) # \( v \) is mutated to \([1, 2, 3, [4, 5, 6]]\)

• \( x = v + 4 \) # **error**

• \( v += 4 \) # **error**

• \( v += [4] \) # \( v \) is mutated to \([1, 2, 3, 4]\)
in: Checking for a value

• The \texttt{in} operator:
  - \texttt{a in l}
  - \texttt{a not in l}

• Not very fast for lists
Queue and Stack

- Queue: add to the end, remove from the front
- Stack: add to the end, remove from the end
- Which operations would we use for add/remove for Python lists?
Queue and Stack Solution

- **Queue**: add to the end, remove from the front
  - create: `queue = []`
  - enqueue: `queue.append(v)`
  - dequeue: `queue.pop(0)`

- **Stack**: add to the end, remove from the end
  - create: `stack = []`
  - push: `stack.append(v)`
  - pop: `stack.pop(-1)`
For loops

- Used much more frequently than while loops
- Is actually a "for-each" type of loop
- In Java, this is:
  ```java
  for (String item : someList) {
    System.out.println(item);
  }
  ```
- In Python, this is:
  ```python
  for item in someList:
    print(item)
  ```
- Grabs each element of `someList` in order and puts it into `item`
- Be very careful about modifying the container when using it in a for loop! (e.g. `someList.append(new_item)`)
What about counting?

• In C++:
  
  for(int i = 0; i < 100; i++) {
    cout << i << endl;
  }

• In Python:
  
  for i in range(0,100): # or range(100)
    print(i)

  range(100) vs. list(range(100))

• What about only even integers?
Dictionaries

- One of the most useful features of Python
- Also known as associative arrays
- Exist in other languages but a core feature in Python
- Associate a key with a value
- When I want to find a value, I give the dictionary a key, and it returns the value
- Example: InspectionID (key) → InspectionRecord (value)
- Keys must be immutable (technically, hashable):
  - Normal types like numbers, strings are fine
  - Tuples work, but lists do not (TypeError: unhashable type: 'list')
- There is only one value per key!
Dictionaries

• Defining a dictionary: curly braces

states = {'MA': 'Massachusetts, 'RI': 'Road Island', 'CT': 'Connecticut'}

• Accessing a value: use brackets!

states['MA'] or states.get('MA')

• Adding a value:

states['NH'] = 'New Hampshire'

• Checking for a key:

'ME' in states → returns True or False

• Removing a value: states.pop('CT') or del states['CT']

• Changing a value: states['RI'] = 'Rhode Island'
Dictionaries

- Combine dictionaries: `d1.update(d2)`
  - `update` overwrites any key-value pairs in `d1` when the same key appears in `d2`
- `len(d)` is the number of entries in `d`
Extracting Parts of a Dictionary

- `d.keys()`: the keys only
- `d.values()`: the values only
- `d.items()`: key-value pairs as a collection of tuples: `[(k1, v1), (k2, v2), ...]`

- Unpacking a tuple or list
  - `t = (1, 2)`
  - `a, b = t`

- Iterating through a dictionary:
  - `for (k,v) in d.items():`
    - `if k % 2 == 0:`
      - `print(v)`

- Important: keys, values, and items are not in any specific order!
Example: Counting Letters

• Write code that takes a string s and creates a dictionary with that counts how often each letter appears in s

• `count_letters("Mississippi") → {'s': 4, 'i': 4, 'p': 2, ...}`
Solution using Counter

• Use an existing library made to count occurrences
• from collections import Counter
  Counter("Mississippi")
• produces
• Counter({'M': 1, 'i': 4, 's': 4, 'p': 2})
• Improve: convert to lowercase first