DSC 201: Data Analysis & Visualization

Dictionaries

Dr. David Koop
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)
Functions

• Calling functions is as expected:
  
  \texttt{mul(2,3)} \# computes 2*3 (\texttt{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}

• 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!
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")
  ```
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)`
Lists

- New list: \( l = [ ] \) or \( l = \text{list()} \)
- Add to a list \( l \):
  - \( l\text{.append}(v) \): add one value \( (v) \) to the end of the list
  - \( l\text{.extend}(v\text{list}) \): add multiple values \( (v\text{list}) \) to the end of \( l \)
  - \( l\text{.insert}(i, v) \): add one value \( (v) \) at index \( i \)
- Remove from a list \( l \):
  - \( \text{del } l[i] \): deletes the value at index \( i \)
  - \( l\text{.pop}(i) \): removes the value at index \( i \) (and returns it)
  - \( l\text{.remove}(v) \): removes the \textbf{first} occurrence of value \( v \) (careful!)
- Changing an entry:
  - \( l[i] = v \): changes the value at index \( i \) to \( v \)
Quiz

• Suppose $a = ['a', 'b', 'c', 'd']$ and $b = (1, 2, 3)$

• What happens with?
  - $a[0]$
  - $b[:-2]$
  - $b$.append(4)
  - $a$.extend($b$)
  - $a$.pop(0)
  - $b[0] = "100"
  - $b + (4,)$
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?
Assignment 2

• [www.cis.umassd.edu/~dkoop/dsc201-2017fa/assignment2.html](http://www.cis.umassd.edu/~dkoop/dsc201-2017fa/assignment2.html)
• Python and Jupyter Notebook
• Dataset of MA, CT, RI towns
• Data exploration
  - Find max, min
  - Count
  - String processing
• Due Tomorrow!
For loops

• Used much more frequently than while loops
• Is actually a "for-each" type of loop
• In Java, this is:
  
  - for (String item : someList) {
        System.out.println(item);
  }

• In Python, this is:
  
  - 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++:
  ```cpp
  for(int i = 0; i < 100; i++) {
    cout << i << endl;
  }
  ```

- In Python:
  ```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) $\rightarrow$ 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:
  \[ [(k1, v1), (k2, v2), \ldots] \]

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

- Iterating through a dictionary:
  ```python
  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

```python
count_letters("Mississippi") →
{'s': 4, 'i': 4, 'p': 2', ...}
```
Example: Counting Letters

• def count_letters(s):
    letter_counts = {}
    for ch in s:
        if ch not in letter_counts:
            letter_counts[ch] = 1
        else:
            letter_counts[ch] += 1

• Cases? s.upper, s.lower

• get method