Introduction to Programming for Public Policy Week 3 (Lists, Strings, and Dictionaries)

Eric Potash

April 10, 2018

Lists

List operations: +

Can concatenate lists just like strings:

```
>>> [1, 2, 3] + ['a', 'b', 'c']
[1, 2, 3, 'a', 'b', 'c']
```

List operations: *

Can also repeat a list with *:

```
>>> [0] * 5
[0, 0, 0, 0, 0]
>>> [1,2,3] * 2
[1, 2, 3, 1, 2, 3]
```

List operator: `in`

You can check whether a value is in a list using the `in` operator:

```
>>> ls = [1,3,5,7]
>>> 3 in ls
True
>>> 4 in ls
False
```

List slices

As with strings we can slice lists. Note that, as with strings, we can omit either end: the slice will then start at the beginning or end at the end of the list.

```
>>> a = ['a', 'b', 'c', 'd', 'e', 'f']
>>> a[2:]
['c', 'd', 'e', 'f']
>>> a[:3]
['a', 'b', 'c']
```

List append

```
>>> a = ['a', 'b', 'c']
>>> a.append('d')
>>> a
['a', 'b', 'c', 'd']
```

List extend

```
>>> a = ['a', 'b', 'c']
>>> a.extend(['d', 'e', 'f'])
>>> a
['a', 'b', 'c', 'd', 'e', 'f']
```

List sort

The `sort` function sorts the list:

```
>>> a = [15, 11, 2, 23, 13]
>>> a.sort()
>>> a
[2, 11, 13, 15, 23]
```

Inplace

Note that the above list functions (`append`, `extend`, `sort`) modify the functions *inplace* and return `None`.

Removing elements

Two ways to remove elements from a list:

- Remove by value:

```
>>> a = ['a','b','c','d']
>>> a.remove('d')
>>> a
['a', 'b', 'c']
```

Removing elements

Two ways to remove elements from a list:

- Remove by value:

```
>>> a = ['a','b','c','d']
>>> a.remove('d')
>>> a
['a', 'b', 'c']
```

- Remove by index and return value:

```
>>> a.pop(1)
'b'
>>> a
['a', 'c']
```

Median

We can use sorting to find the median in a list. Sort and take the middle value:

```
>>> a = [15, 11, 2, 23, 13]
>>> b = sorted(a)
>>> b[int(len(b)/2)]
13
```

Here `int(len(b)/2) = 2`, the middle index. Note that this code will give the wrong answer for lists of even length.

Percentile

More generally, we can find an arbitrary percentile p:

```
>>> b = sorted(a)
>>> b[int(len(b)*p)]
```

E.g. when p=.5 this is the median. Note that this is a crude version of percentile– in practice we use interpolation to refine.

String parsing examples

What is parsing?

The process of reading through a string to break it down or interpret it is called *parsing*.

- Command line programs do this in order to interpret options and arguments
- Python does this to execute your code
- Google does it to execute a search query
- Etc.

String parsing example

On the next slide we'll write a script that runs like this:

-
  ```
  $ python city_state.py
  Enter City, State: Chicago, IL

  City: Chicago
  State: IL
  ```

String parsing example

On the next slide we'll write a script that runs like this:

-
  ```
  $ python city_state.py
  Enter City, State: Chicago, IL

  City: Chicago
  State: IL
  ```

-
  ```
  $ python city_state.py
  Enter City, State: New York City
  Traceback (most recent call last):
    File "parse_city.py", line 4, in <module>
      raise ValueError('no comma')
  ValueError: no comma
  ```

String functions

- `string.endswith(ending)`: does `string` end in `ending`?

```
>>> 'Chicago, IL'.endswith('IL')
True
```

String functions

- `string.endswith(ending)`: does `string` end in `ending`?

```
>>> 'Chicago, IL'.endswith('IL')
True
```

- `string.startswith(beginning)`: does `string` start with `beginning`?

```
>>> 'Chicago, IL'.startswith('New York')
False
```

String functions

- `string.endswith(ending)`: does string end in ending?

```
>>> 'Chicago, IL'.endswith('IL')
True
```

- `string.startswith(beginning)`: does string start with `beginning`?

```
>>> 'Chicago, IL'.startswith('New York')
False
```

- `substring in string`: does string contain substring?

```
>>> ',' in 'Chicago, IL'
True
```

More string functions

- `string.find(substring)`: what is the (first!) index of `substring` in `string`? (or -1 if `substring not in string`)

```
>>> 'Hello, World'.find(',')
5
```

More string functions

- `string.find(substring)`: what is the (first!) index of `substring` in `string`? (or -1 if `substring not in string`)

```
>>> 'Hello, World'.find(',')
5
```

- `string.split(sep)`: split `string` on `sep` into list of strings

```
>>> 'Chicago, IL, USA'.split(',')
['Chicago', ' IL', ' USA']
```

Errors

- We've seen Python "throw" errors before:

```
NameError: name 'Pi' not found
```

Errors

- We've seen Python "throw" errors before:

```
NameError: name 'Pi' not found
```

- We can throw our own errors:

```
raise ValueError("Invalid value")
```

String parsing example code

```
# city_state.py
city_state = input('Enter City, State: ')
if ',' not in city_state: # substring
    raise ValueError('no comma')

comma_index = city_state.find(',')
city = city_state[:comma_index]
state = city_state[comma_index+2:]

if len(state) != 2:
    raise ValueError('invalid state abbrev: ' + state)

print('\nCity:', city)
print('State:', state)
```

Command Line Arguments

A python script can use command line arguments through the `argv` list in the `sys` module:

```
# cmd_args.py
import sys
print(sys.argv)
```

```
$ python cmd_args.py -o -h arguments
['cmd_args.py', '-o', '-h', 'arguments']
```

Opening a file

Open a file using the open function:

```
>>> file = open('salaries.csv')
>>> print(file)
<_io.TextIOWrapper name='salaries.csv' mode='r' encoding='U
```

This TextIOWrapper object facilitates I/O (input/output).

Reading a line

- Read a line from the file:

```
>>> file = open('salaries.csv')
>>> line = file.readline()
>>> line
'Name,Job Titles,Department,Full or Part-Time,
 Salary or Hourly,Typical Hours,Annual Salary,
 Hourly Rate\n'
```

- `\n` is the *line feed* character. It is a single character. One way to remove it in this example:

```
>>> line[:-1]
'Name,Job Titles,Department,Full or Part-Time,
 Salary or Hourly,Typical Hours,Annual Salary,
 Hourly Rate'
```

Reading many lines

You can iterate over the lines in a file similarly to a list:

```
>>> file = open('salaries.csv')
>>> lines = []
>>> for line in file:
...     lines.append(line[:-1])

['Name,Job Titles,Department,Full or Part-Time,Salary or Ho
 '"AARON,  JEFFERY M",SERGEANT,POLICE,F,Salary,,$101442.00,
 '"AARON,  KARINA ",POLICE OFFICER (ASSIGNED AS DETECTIVE),
...]
```

Splitting fields

This list of lines is not very useful for analysis. The first step is to break up the lines into fields.

```
>>> file = open('salaries.csv')
>>> line = file.readline()[:-1]
>>> line.split(',')
['Name',
 'Job Titles',
 'Department',
 'Full or Part-Time',
 'Salary or Hourly',
 'Typical Hours',
 'Annual Salary',
 'Hourly Rate']
```

2d list

```
>>> file = open('salaries.csv')
>>> lines = []
>>> for line in file:
...     fields = line[:-1].split(',')
...     lines.append(fields)
>>> lines
[['Name',
  'Job Titles',
  'Department',
  'Full or Part-Time',
  'Salary or Hourly',
  'Typical Hours',
  'Annual Salary',
  'Hourly Rate'],
 ['"AARON',
  '  JEFFERY M"'
```

2d lists

- `lines` above is a two dimensional (2d) list

2d lists

- `lines` above is a two dimensional (2d) list
    - i.e. it is a list in which each item (a row) is itself a list (of fields)

2d lists

- `lines` above is a two dimensional (2d) list
    - i.e. it is a list in which each item (a row) is itself a list (of fields)
- We can subset rows `lines[2]` or a group of columns with a slice `lines[2:4]`

2d lists

- `lines` above is a two dimensional (2d) list
  - i.e. it is a list in which each item (a row) is itself a list (of fields)
- We can subset rows `lines[2]` or a group of columns with a slice `lines[2:4]`
- `lines[1:]` will return everything except the header

Aggregating

We can aggregate the number of full time employees:

```
count = 0
for line in lines:
    if line[4] == 'F':
        count = count + 1
```

Getting a column

We can extract a single column from the 2d list:

```
departments = []
for row in lines:
    departments.append(row[3])
```

Further string parsing

- The 2d list is an improvement but there's more work to do

Further string parsing

- The 2d list is an improvement but there's more work to do
- Extra characters (double quotes in the names)

Further string parsing

- The 2d list is an improvement but there's more work to do
- Extra characters (double quotes in the names)
- Numbers like salaries are still strings

Further string parsing

- The 2d list is an improvement but there's more work to do
- Extra characters (double quotes in the names)
- Numbers like salaries are still strings
- You'll do this in your assignment this week

Further string parsing

- The 2d list is an improvement but there's more work to do
- Extra characters (double quotes in the names)
- Numbers like salaries are still strings
- You'll do this in your assignment this week
- In the future we'll use existing python modules to parse CSVs

Dictionaries

What is a dictionary?

- Another python data structure is a *dictionary* (called a hashmap in some languages).

What is a dictionary?

- Another python data structure is a *dictionary* (called a hashmap in some languages).
- In a list, indices are integers.

What is a dictionary?

- Another python data structure is a *dictionary* (called a hashmap in some languages).
- In a list, indices are integers.
- In a dictionary, indices can take almost any type.

More on dictionaries

- A dictionary can also be thought of as a mapping between *keys* (indices) and *values*.

More on dictionaries

- A dictionary can also be thought of as a mapping between *keys* (indices) and *values*.
- Each key maps to a value. The keys are unique but the values need not be.

More on dictionaries

- A dictionary can also be thought of as a mapping between *keys* (indices) and *values*.
- Each key maps to a value. The keys are unique but the values need not be.
- The combination of a key and a value is called a *key-value pair* or an *item*.

dict construction

- Can construct a dictionary using the `dict` function:

```
>>> d = dict()
>>> d
{}
```

dict construction

- Can construct a dictionary using the `dict` function:

```
>>> d = dict()
>>> d
{}
```

- So don't call your dictionaries `dict`

dict construction

- Can construct a dictionary using the `dict` function:

```
>>> d = dict()
>>> d
{}
```

- So don't call your dictionaries `dict`
- `{}` is an empty dictionary.

dict construction

- Can construct a dictionary using the `dict` function:

```
>>> d = dict()
>>> d
{}
```

- So don't call your dictionaries `dict`
- `{}` is an empty dictionary.
- Curly braces are the analogue for dicts of the square braces `[]` for lists.

Adding items to a dictionary

Add items to a dictionary using square brackets:

```
>>> eng2esp = dict()
>>> eng2esp['one'] = 'uno'
>>> eng2esp
{'one': 'uno'}
```

Now eng2esp maps 'one' to 'uno'.

Alternative dictionary constructor

You can also create a `dict` with curly brace syntax:

```
>>> eng2esp = {'one':'uno', 'two':'dos', 'three':'tres'}
```

Dictionaries are unordered

- Dictionaries are *unordered* meaning their items do not have a sequence:

```
>>> eng2esp = {'one':'uno', 'two':'dos', 'three':'tres'
>>> eng2esp
{'one':'uno', 'three':'tres', 'two':'dos'}
```

Dictionaries are unordered

- Dictionaries are *unordered* meaning their items do not have a sequence:

```
>>> eng2esp = {'one':'uno', 'two':'dos', 'three':'tres'
>>> eng2esp
{'one':'uno', 'three':'tres', 'two':'dos'}
```

- So you cannot rely on the order of elements in a dictionary. But it's okay because we index the elements using their keys, not their order:

```
>>> eng2esp['two']
'dos'
```

KeyError

If you try to get a value for a non-existent key you'll get a `KeyError`:

```
>>> eng2esp['four']
KeyError: 'four'
```

in operator

To determine whether a key is in a dictionary you can use the `in` operator:

```
>>> 'one' in eng2esp
True
>>> 'uno' in eng2esp
False
```

Associated lists

A dictionary has some useful “lists” associated with it:

- keys() is like a list of the keys:

```
>>> eng2esp.keys()
dict_keys(['three', 'one', 'two'])
```

Associated lists

A dictionary has some useful “lists” associated with it:

- `keys()` is like a list of the keys:

```
>>> eng2esp.keys()
dict_keys(['three', 'one', 'two'])
```

- `values()` is like a list of the values:

```
>>> eng2esp.values()
dict_values(['tres', 'uno', 'dos'])
```

Counts

- A common task is to take a set of things and count the distinct things

Counts

- A common task is to take a set of things and count the distinct things
- e.g. to count the number of times each department appears in the employees file

Counts

- A common task is to take a set of things and count the distinct things
- e.g. to count the number of times each department appears in the employees file
- Here we'll do a related example of counting the number of times each letter appears in a word

Letter counts

- For example, given a word like `'brontosaurus'`, we want to know that there are:
  - `'b'`: 1
  - `'r'`: 2
  - `'o'`: 2
  - etc.
- So we need to map letters to counts
- Dictionaries work well for this

Letter counts code

```
word = 'brontosaurus'
d = {} # store counts in this dictionary
for letter in word:
    if letter not in d:
        d[letter] = 1
    else:
        d[letter] = d[letter] + 1
```

+= operator

Note that

```
x = x +1
```

can be written equivalently and conveniently in python as

```
x += 1
```

Letter counts code +=

```
word = 'brontosaurus'
d = {} # store counts in this dictionary
for letter in word:
    if letter not in d:
        d[letter] = 1
    else:
        d[letter] += 1
```

Maximum value

We can use the `values()` function to find the maximum value among the counts:

```
>>> d
{'a': 1, 'r': 2, 'b': 1, 'n': 1, 'o': 2, 'u': 2, 't': 1, 's
>>> d.values()
dict_values([1, 2, 1, 1, 2, 2, 1, 2])
>>> max(d.values())
2
```

Iterating over a dictionary

- It's useful to iterate over a dictionary:

```
for key in d:
    print(key)
```

- Can lookup the value for each key in the loop:

```
for key in d:
    print(key, ':', d[key])
```

Searching for a value

Use that to search for keys with a given value:

```
search = 2
for key in d:
    if d[key] == search:
        print(key)
```

Dictionary substraction

Suppose we have two letter counts d1 and d2 from two different words and we want to find the letters in d1 that aren't in d2. We can do this with a loop:

```
d1_minus_d2 = []
for letter in d1:
    if letter not in d2:
        d1_minus_d2.append(letter)
```

Inverting a dictionary

We can invert a dictionary, i.e. switch the direction of the mapping so that 'uno' maps to 'one':

```
esp2eng = {}
for eng in eng2esp:
    esp = eng2esp[eng]
    if esp in esp2eng:
        raise ValueError('Duplicate values: ' + esp)
    esp2eng[esp] = eng
```

Note that when the source dictionary (eng2esp) contains duplicate values, the dictionary is not invertible and the code will error.