Introduction to Programming for Public Policy Week 3 (Lists and String Parsing)

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 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

```
>>> a = ['a','b','c','d']
>>> a.remove('d')          # remove by value
>>> a
['a', 'b', 'c']
>>> a.pop(1)               # by index and return value
'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[round(len(b)/2)-1]
15
```

Percentile

More generally, we can find an arbitrary percentile p:

```
>>> b = sorted(a)
>>> b[round(len(b)*p)-1]
```

Note that this is a crude version of percentile– in practice we use interpolation to refine.

String parsing

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

```python
city_state = 'Chicago, IL'

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('City:', city)
print('State:', state)
```

More string functions

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

More string functions

- `string.endswith(ending)`: does `string` end in `ending`?
- `string.startswith(beginning)`: does `string` start with `beginning`?

More string functions

- `string.endswith(ending)`: does `string` end in `ending`?
- `string.startswith(beginning)`: does `string` start with `beginning`?
- `substring in string`: does `string` contain `substring`?

More string functions

- `string.endswith(ending)`: does `string` end in `ending`?
- `string.startswith(beginning)`: does `string` start with `beginning`?
- `substring in string`: does `string` contain `substring`?
- `string.find(substring)`: what is the (first!) index of `substring` in `string`? (or -1 if `substring not in string`)

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']
```

Example: Reading a CSV file

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

```
>>> file = open('salaries.csv')
>>> file.readline()
'Name,Job Titles,Department,Full or Part-Time,Salary or Hou
```

The \n character is the *line feed* character. It is a single character. We can remove it by indexing `[:-1]`.

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. “‘python >>> 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',
```

2d lists

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

2d lists

- `lines` above is a two dimensional (2d) list
- We can subset the columns using `lines[2]` or a group of columns with a slice `lines[2:4]`

2d lists

- `lines` above is a two dimensional (2d) list
- We can subset the columns using `lines[2]` or a group of columns with a slice `lines[2:4]`
- We can subset rows using `lines[:][5]` or `lines[:][5:10]`

2d lists

- `lines` above is a two dimensional (2d) list
- We can subset the columns using `lines[2]` or a group of columns with a slice `lines[2:4]`
- We can subset rows using `lines[:][5]` or `lines[:][5:10]`
- We can subset rows and columns using `lines[2:4][5:10]`

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
- 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*.
- 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*.

Counter