Lecture 5. Python Primer – Part 5

Chao Chen

Stony Brook University

Oct. 17, 2022

List vs str

- List and string – similar but different (str is immutable)
- Convert a string to a list of chars

```
>>> s = 'spam'
>>> t = list(s)
>>> print(t)
['s', 'p', 'a', 'm']
```

String to list (cont'd)

- Splitting a string to words

```
>>> s = 'pining for the fjords'
>>> t = s.split()
>>> print(t)
['pining', 'for', 'the', 'fjords']
>>> print(t[2])
the
```

- Specify delimiter

```
>>> s = 'spam-spam-spam'
>>> delimiter = '-'
>>> s.split(delimiter)
['spam', 'spam', 'spam']
```

List to str

- Joins list into a string

```
>>> t = ['pining', 'for', 'the', 'fjords']
>>> delimiter = ' '
>>> delimiter.join(t)
'pining for the fjords'
```

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if not line.startswith('From '): continue
    words = line.split()
    print(words[2])

# Code: http://www.py4e.com/code3/search5.py
```

Aliasing

```
>>> a = [1, 2, 3]
>>> b = [1, 2, 3]
>>> a is b
False
```

```
>>> a = [1, 2, 3]
>>> b = a
>>> b is a
True
```

If the aliased object is mutable, changes made with one alias affect the other:

```
>>> b[0] = 17
>>> print(a)
[17, 2, 3]
```

Although this behavior can be useful, it is error-prone. In general, it is safer to avoid aliasing when you are working with mutable objects.

List as an argument

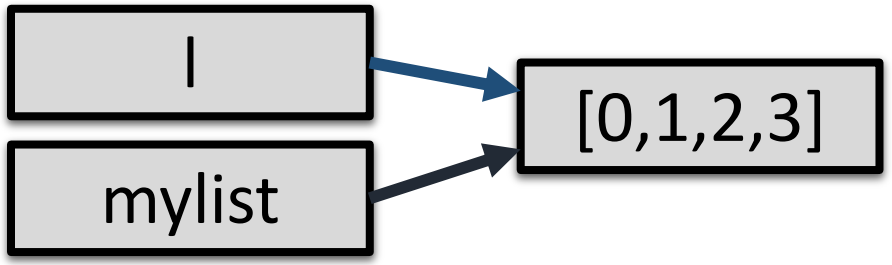
- Passing lists as arguments to a function
- Will changes made to a list inside a function remain effective?

```
def delete_head(t):
    del t[0]

>>> letters = ['a', 'b', 'c']
>>> delete_head(letters)
>>> print(letters)
['b', 'c']
```

```
1  def change_my_list(mylist):
2      mylist[1] = None
3      mylist.append('The End')
4
5  l = [0,1,2,3]
6  change_my_list(l)
7  print(l)
```

```
[0, None, 2, 3, 'The End']
```



List as an argument

- What about this example?

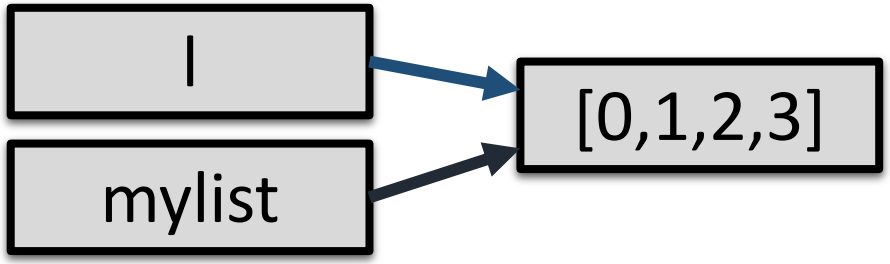
```
1  def change_my_list_2(mylist):
2      tail = [4,5,6]
3      mylist = mylist + tail
4
5  l = [0,1,2,3]
6  change_my_list_2(l)
7  print(l)
```

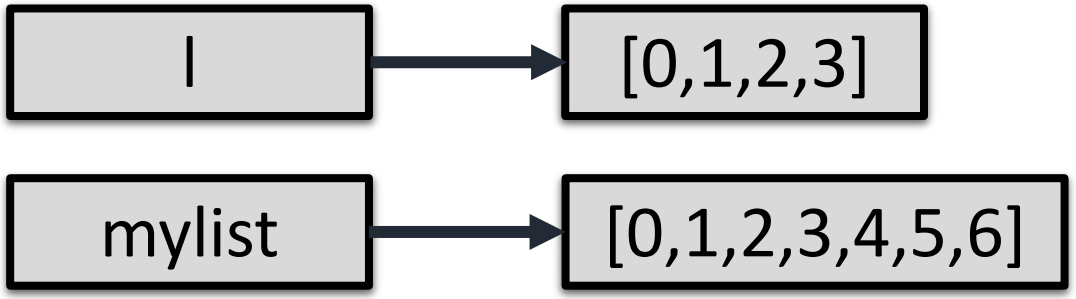
```
[0, 1, 2, 3]
```

```
1  def change_my_list_2(mylist):
2      tail = [4,5,6]
3      mylist.extend(tail)
4
5  l = [0,1,2,3]
6  change_my_list_2(l)
7  print(l)
```

```
[0, 1, 2, 3, 4, 5, 6]
```

- The “assignment” ruins the aliasing!





Difference between operations that modifies a list or creates a list

- Operations that modifies: append, extend, del, etc.
- Operations that creates: assignment

```
def bad_delete_head(t):
    t = t[1:]                    # WRONG!
```

- Taking a slice of t, it will not delete the head of t

Exercise 1:

Write a function called **chop** that takes a list and modifies it, removing the first and last elements, and returns **None**.

Then write a function called **middle** that takes a list and returns a new list that contains all but the first and last elements.

Difference between string operations and list operations

- String: return new string, original string unchanged
- List: return None, original list updated

```
1  origin_str = "  \n\t Hello World \n\t"
2  short_str = origin_str.strip()
3  print("short_str = ", short_str)
4  print("origin_str = ", origin_str)
```

```
short_str =  Hello World
origin_str =
         Hello World
```

```
1  origin_l = [3,2,1,0]
2  sorted_l = origin_l.sort()
3  print("sorted_l = ", sorted_l)
4  print("origin_l = ", origin_l)
```

```
sorted_l =  None
origin_l =  [0, 1, 2, 3]
```

List modifications

- Adding an element x to the end of a list t. Which ones of the following work?

```
t.append([x])
t = t.append(x)
t + [x]
t = t + x
```

- Pass in the element to append, when “+” all arguments are lists

```
t.append(x)
t = t + [x]
```

Debugging

- What could go wrong?

```
fhand = open('mbox-short.txt')
for line in fhand:
    words = line.split()
    if words[0] != 'From' : continue
    print(words[2])
```

Debugging

- What could go wrong?

```
fhand = open('mbox-short.txt')
count = 0
for line in fhand:
    words = line.split()
    # print 'Debug:', words
    if len(words) == 0 : continue
    if words[0] != 'From' : continue
    print(words[2])
```

Debugging

Exercise 2: Figure out which line of the above program is still not properly guarded. See if you can construct a text file which causes the program to fail and then modify the program so that the line is properly guarded and test it to make sure it handles your new text file.

Exercise 3: Rewrite the guardian code in the above example without two `if` statements. Instead, use a compound logical expression using the **`and`** logical operator with a single `if` statement.

Exercise 4: Download a copy of the file from www.py4e.com/code3/romeo.txt

Write a program to open the file `romeo.txt` and read it line by line. For each line, split the line into a list of words using the `split` function.

For each word, check to see if the word is already in a list. If the word is not in the list, add it to the list.

When the program completes, sort and print the resulting words in alphabetical order.

```
Enter file: romeo.txt
['Arise', 'But', 'It', 'Juliet', 'Who', 'already',
'and', 'breaks', 'east', 'envious', 'fair', 'grief',
'is', 'kill', 'light', 'moon', 'pale', 'sick', 'soft',
'sun', 'the', 'through', 'what', 'window',
'with', 'yonder']
```

Exercise

- https://www.py4e.com/code3/mbox.txt

Exercise 5: Write a program to read through the mail box data and when you find line that starts with "From", you will split the line into words using the `split` function. We are interested in who sent the message, which is the second word on the From line.

```
From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008
```

You will parse the From line and print out the second word for each From line, then you will also count the number of From (not From:) lines and print out a count at the end.

This is a good sample output with a few lines removed:

```
python fromcount.py
Enter a file name: mbox-short.txt
stephen.marquard@uct.ac.za
louis@media.berkeley.edu
zqian@umich.edu

[...some output removed...]

ray@media.berkeley.edu
cwen@iupui.edu
cwen@iupui.edu
cwen@iupui.edu
There were 27 lines in the file with From as the first word
```

Exercise

Exercise 6: Rewrite the program that prompts the user for a list of numbers and prints out the maximum and minimum of the numbers at the end when the user enters "done". Write the program to store the numbers the user enters in a list and use the `max()` and `min()` functions to compute the maximum and minimum numbers after the loop completes.

```
Enter a number: 6
Enter a number: 2
Enter a number: 9
Enter a number: 3
Enter a number: 5
Enter a number: done
Maximum: 9.0
Minimum: 2.0
```

Dictionary

- Collection of key-value pairs

```
>>> eng2sp = dict()
>>> print(eng2sp)
{}
```

```
>>> print(eng2sp)
{'one': 'uno'}
```

- Indexing: use “[ ]”

```
>>> eng2sp['one'] = 'uno'
```

```
>>> eng2sp = {'one': 'uno', 'two': 'dos', 'three': 'tres'}
>>> print(eng2sp)
{'one': 'uno', 'three': 'tres', 'two': 'dos'}
```

```
>>> print(eng2sp['two'])
'dos'
```

```
>>> print(eng2sp['four'])
KeyError: 'four'
```

The end

- Length, in – check if a key is in the dictionary

```
>>> len(eng2sp)
3
>>> 'one' in eng2sp
True
>>> 'uno' in eng2sp
False
```

- keys = eng2sp.keys() -- get a collection of keys
- vals = eng2sp.vals() -- get a collection of vals

```python
myd=dict()
for i in range(8):
    myd[str(i)] = i
print("myd = ", myd)
keys = myd.keys()
keys_list = list(myd.keys())
vals = myd.values()
vals_list = list(myd.values())
print("keys = ", keys, " type: ", type(keys))
print("keys_list = ", keys_list, " type: ", type(keys_list))
print("vals = ", vals, " type: ", type(vals))
print("vals_list = ", vals_list, " type: ", type(vals_list))

```

```
myd =  {'0': 0, '1': 1, '2': 2, '3': 3, '4': 4, '5': 5, '6': 6, '7': 7}
keys =  dict_keys(['0', '1', '2', '3', '4', '5', '6', '7'])  type:  <clas
s 'dict_keys'>
keys_list =  ['0', '1', '2', '3', '4', '5', '6', '7']  type:  <class 'lis
t'>
vals =  dict_values([0, 1, 2, 3, 4, 5, 6, 7])  type:  <class 'dict_value
s'>
vals_list =  [0, 1, 2, 3, 4, 5, 6, 7]  type:  <class 'list'>
```

Checking whether a key/val exists in a dict

- Use “in”
- xxx in myd
- keys = myd.keys()
  xxx in keys
- keys_list = list(keys)
  xxx in keys_list

```
1  myd=dict()
2  for i in range(1000000):
3      myd[str(i)] = i
```

```
1  %%time
2
3  mybool = True
4  for i in range(1000000):
5      mybool = str(i) in myd
6  print(mybool)
```

```
True
CPU times: user 518 ms, sys: 3 ms, total: 521 ms
Wall time: 520 ms
```

```
%%time

mybool = True
keys = myd.keys()
for i in range(10000):
    mybool = str(i) in keys
print(mybool)
```

```
True
CPU times: user 6.15 ms, sys: 1.25 ms, total: 7.41 ms
Wall time: 6.31 ms
```

```
%%time

mybool = True
key_list = list(myd.keys())
for i in range(10000):
    mybool = str(i) in key_list
print(mybool)
```

```
True
CPU times: user 759 ms, sys: 7.31 ms, total: 767 ms
Wall time: 780 ms
```

```
%%time

mybool = True
keys = myd.keys()
for i in range(100000):
    mybool = str(i) in keys
print(mybool)
```

```
True
CPU times: user 49.7 ms, sys: 1.99 ms, total: 51.7 ms
Wall time: 50.6 ms
```

```
%%time

mybool = True
key_list = list(myd.keys())
for i in range(100000):
    mybool = str(i) in key_list
print(mybool)
```

```
True
CPU times: user 1min 18s, sys: 420 ms, total: 1min 18s
Wall time: 1min 19s
```

“in” – scalability issue

- range(10000)
  in keys – 6.3 ms
  in keys_list – 780 ms
- range(100000)
  in keys – 50.6 ms
  in keys_list – 1min 19s = 79,000 ms
- keys – implemented using hash table
  constant operation for each ”in” operation
- keys_list – linear operation
  can be as expensive as the list length (1,000,000 here)

Exercise

Exercise 1: [wordlist2]

Write a program that reads the words in `words.txt` and stores them as keys in a dictionary. It doesn't matter what the values are. Then you can use the `in` operator as a fast way to check whether a string is in the dictionary.

Suppose you are given a string and you want to count how many times each letter appears. There are several ways you could do it:

1. You could create 26 variables, one for each letter of the alphabet. Then you could traverse the string and, for each character, increment the corresponding counter, probably using a chained conditional.

2. You could create a list with 26 elements. Then you could convert each character to a number (using the built-in function `ord`), use the number as an index into the list, and increment the appropriate counter.

3. You could create a dictionary with characters as keys and counters as the corresponding values. The first time you see a character, you would add an item to the dictionary. After that you would increment the value of an existing item.

Counting letters

- Dictionary implementation:
  - do not need to know what letters are,
  - only create entries for letters appeared

```
word = 'brontosaurus'
d = dict()
for c in word:
    if c not in d:
        d[c] = 1
    else:
        d[c] = d[c] + 1
print(d)
```

Counting letters

- Simplify with get (return the second arg if the key does not exist)

```
>>> counts = { 'chuck' : 1 , 'annie' : 42, 'jan': 100}
>>> print(counts.get('jan', 0))
100
>>> print(counts.get('tim', 0))
0
```

```
word = 'brontosaurus'
d = dict()
for c in word:
    d[c] = d.get(c,0) + 1
print(d)
```

Counting words in a document

```
fname = input('Enter the file name: ')
try:
    fhand = open(fname)
except:
    print('File cannot be opened:', fname)
    exit()

counts = dict()
for line in fhand:
    words = line.split()
    for word in words:
        if word not in counts:
            counts[word] = 1
        else:
            counts[word] += 1

print(counts)

# Code: http://www.py4e.com/code3/count1.py
```

```
But soft what light through yonder window breaks
It is the east and Juliet is the sun
Arise fair sun and kill the envious moon
Who is already sick and pale with grief
```

```
python count1.py
Enter the file name: romeo.txt
{'and': 3, 'envious': 1, 'already': 1, 'fair': 1,
'is': 3, 'through': 1, 'pale': 1, 'yonder': 1,
'what': 1, 'sun': 2, 'Who': 1, 'But': 1, 'moon': 1,
'window': 1, 'sick': 1, 'east': 1, 'breaks': 1,
'grief': 1, 'with': 1, 'light': 1, 'It': 1, 'Arise': 1,
'kill': 1, 'the': 3, 'soft': 1, 'Juliet': 1}
```

Enumerate through a counting dictionary

```
counts = { 'chuck' : 1 , 'annie' : 42, 'jan': 100}
for key in counts:
    print(key, counts[key])
```

```
jan 100
chuck 1
annie 42
```

- How to:
  - Only output the ones with >10 counts

```
jan 100
annie 42
```

  - Out the words in alphabetical order

```
annie 42
chuck 1
jan 100
```

Solutions

```
counts = { 'chuck' : 1 , 'annie' : 42, 'jan': 100}
for key in counts:
    if counts[key] > 10 :
        print(key, counts[key])
```

```
jan 100
annie 42
```

```
counts = { 'chuck' : 1 , 'annie' : 42, 'jan': 100}
lst = list(counts.keys())
print(lst)
lst.sort()
for key in lst:
    print(key, counts[key])
```

```
['jan', 'chuck', 'annie']
annie 42
chuck 1
jan 100
```

Advanced parsing: removing all punctuations

```
1 import string
2 print(string.punctuation)
3 line = "But, soft!!!! what light through yonder window breaks?"
4 print(line)
5 print(line.translate(line.maketrans('','',string.punctuation)))
```

```
!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
But, soft!!!! what light through yonder window breaks?
But soft what light through yonder window breaks
```

```
1 print(line.translate(line.maketrans(",!?","###")))
2 print(line.translate(
3     line.maketrans(string.punctuation,"#"*len(string.punctuation))))
4 print(line.translate(line.maketrans(",!?","###",'i')))
```

```
But# soft#### what light through yonder window breaks#
But# soft#### what light through yonder window breaks#
But# soft#### what lght through yonder wndow breaks#
```

```
import string

fname = input('Enter the file name: ')
try:
    fhand = open(fname)
except:
    print('File cannot be opened:', fname)
    exit()

counts = dict()
for line in fhand:
    line = line.rstrip()
    line = line.translate(line.maketrans('', '', string.punctuation))
    line = line.lower()
    words = line.split()
    for word in words:
        if word not in counts:
            counts[word] = 1
        else:
            counts[word] += 1

print(counts)
```

Advanced parsing:

- removing all punctuations
- convert all words to lowercases

```
Enter the file name: romeo-full.txt
{'swearst': 1, 'all': 6, 'afeard': 1, 'leave': 2, 'these': 2,
'kinsmen': 2, 'what': 11, 'thinkst': 1, 'love': 24, 'cloak': 1,
a': 24, 'orchard': 2, 'light': 5, 'lovers': 2, 'romeo': 40,
'maiden': 1, 'whiteupturned': 1, 'juliet': 32, 'gentleman': 1,
'it': 22, 'leans': 1, 'canst': 1, 'having': 1, ...}
```

Exercise

- https://www.py4e.com/code3/mbox.txt
- https://www.py4e.com/code3/mbox-short.txt

Exercise 2: Write a program that categorizes each mail message by which day of the week the commit was done. To do this look for lines that start with "From", then look for the third word and keep a running count of each of the days of the week. At the end of the program print out the contents of your dictionary (order does not matter).

Sample Line:

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
```

Sample Execution:

```
python dow.py
Enter a file name: mbox-short.txt
{'Fri': 20, 'Thu': 6, 'Sat': 1}
```

Exercise

Exercise 3: Write a program to read through a mail log, build a histogram using a dictionary to count how many messages have come from each email address, and print the dictionary.

```
Enter file name: mbox-short.txt
{'gopal.ramasammycook@gmail.com': 1, 'louis@media.berkeley.edu': 3,
'cwen@iupui.edu': 5, 'antranig@caret.cam.ac.uk': 1,
'rjlowe@iupui.edu': 2, 'gsilver@umich.edu': 3,
'david.horwitz@uct.ac.za': 4, 'wagnermr@iupui.edu': 1,
'zqian@umich.edu': 4, 'stephen.marquard@uct.ac.za': 2,
'ray@media.berkeley.edu': 1}
```

Exercise

Exercise 4: Add code to the above program to figure out who has the most messages in the file.

After all the data has been read and the dictionary has been created, look through the dictionary using a maximum loop (see Section [maximumloop]) to find who has the most messages and print how many messages the person has.

```
Enter a file name: mbox-short.txt
cwen@iupui.edu 5

Enter a file name: mbox.txt
zqian@umich.edu 195
```

Exercise

Exercise 5: This program records the domain name (instead of the address) where the message was sent from instead of who the mail came from (i.e., the whole email address). At the end of the program, print out the contents of your dictionary.

```
python schoolcount.py
Enter a file name: mbox-short.txt
{'media.berkeley.edu': 4, 'uct.ac.za': 6, 'umich.edu': 7,
'gmail.com': 1, 'caret.cam.ac.uk': 1, 'iupui.edu': 8}
```

Homework

- Given data files from UCI (will give a list)
- Convert the data into a list of lists
  - A list of data, each datum is a patient or other person/object instances
  - Each datum is a list of attributes (same length, same correspondence)
  - If continuous value, keep the value
  - If categorical value/nominal value, construct a set/dictionary with the attribute being the key
  - If missing value “?”, use None

Homework

- Compute basic statistics of each attribute
  - All statistics of the attributes form a list (one attribute per item)
  - If a continuous-valued attribute: save max, min, mean, standard deviation
  - If a categorical: construct a dictionary to count frequencies
  - Skip missing value ”None” when you calculate these statistics
- Generate a normalized data
  - Filling missing value:
    - for continuous-valued attributes: use mean
    - For categorical-valued: use maximal frequency value
  - Normalize the data
    - For continuous-valued attributes: new val = (org val – mean) / std
    - For categorical: map values to 0, 1, 2, …., L, have a dictionary mapping org val to new val

The End

- Thank you