香港浸會大學
HONG KONG BAPTIST UNIVERSITY

Data Structures

JOUR7280/COMM7780

Big Data Analytics for Media and Communication

Instructor: Dr. Xiaoyi Fu

Agenda

- List
- Dictionaries
- Tuples
- Sets

List

Programming

- Algorithms
  - A set of rules or steps used to solve a problem
- Data Structures
  - A particular way of organizing data in computer

What is Not a Collection

- Most of our variables have one value in them
  - When we put a new value in the variable, the old value is overwritten

```
x = 2
x = 4
print(x)
```

```
4
```

7 data structures. ipynb

A List is a Kind of Collection

- A collection allows us to put many values in a single "variable"
- A collection is nice because we can carry many values around in one convenient package.

```
friends = ['Tony', 'Peter', 'Natasha']
carryon = ['socks', 'shirt', 'perfume']
```

List Constants

- List constants are surrounded by square brackets and the elements in the list are separated by commas
- A list element can be any Python object
  - Even another list
- A list can be empty

```
print([1, 24, 87])
print(['red', 'yellow', 'blue'])
print(['red', 522, 98.6])
print([1, [5, 6], 7])
print([])
```

```
[1, 24, 87]
['red', 'yellow', 'blue']
['red', 522, 98.6]
[1, [5, 6], 7]
[]
```

7 data structures. ipynb

We Already Use Lists

```
for i in [5, 4, 3, 2, 1]:
    print(i)
print('Blastoff!')
```

Output:

5
4
3
2
1
Blastoff!

Lists and Definite Loops – Best Pal

```
friends = ['Tony', 'Peter', 'Natasha']
for friend in friends:
    print('Happy New Year:', friend)
print('Done!')
```

Output:

Happy New Year: Tony

Happy New Year: Peter

Happy New Year: Natasha

Done!

```
z = ['Tony', 'Peter', 'Natasha']
for x in z:
    print('Happy New Year:', friend)
print('Done!')
```

Look Inside Lists

- Just like strings, we can get any single element in a list using an index specified in square brackets.


```
friends = ['Tony', 'Peter', 'Natasha']
print(friends[1])
```

```
Peter
```

7 data structures. ipynb

Lists are Mutable

- Strings are "immutable"
  - We cannot change the contents of a string
  - We must make a new string to take any change
- Lists are "mutable"
  - We can change any element of a list using the index operator

```
fruit = 'Banana'
fruit[0] = 'b'
```

```
---------------------------------------
---------------------------------
TypeError                                 Traceback
(most recent call last)
<ipython-input-6-2bc78b004470> in <module>
      1 fruit = 'Banana'
----> 2 fruit[0] = 'b'

TypeError: 'str' object does not support i
tem assignment
```

```
x = fruit.lower()
print(x)
lotto = [2, 14, 26, 41, 63]
print(lotto)
lotto[2] = 28
print(lotto)
```

```
banana
[2, 14, 26, 41, 63]
[2, 14, 28, 41, 63]
```

7 data structures. ipynb

How long is a list

- The len() function takes a list as a parameter and returns the number of elements in the list
- Actually len() tells us the number of any set or sequence
  - Such as string …

```
greet = 'Hello Tony'
print(len(greet))
x = [1, 2, 'red', 99]
print(len(x))
```

```
10
4
```

7 data structures. ipynb

Use the range Function

- The range function returns a list of numbers that range from 0 to one less than the parameter
- We can construct an index loop using for and an integer iterator

```
for i in range(4):
    print(i)
```

```
0
1
2
3
```

7 data structures. ipynb

A Table of 2 Loops

```
friends = ['Tony', 'Peter', 'Natasha']
for friend in friends:
    print('Happy New Year:', friend)

for i in range(len(friends)):
    friend = friends[i]
    print('Happy New Year:', friend)
```

Output:

Happy New Year: Tony

Happy New Year: Peter

Happy New Year: Natasha

7 data structures. ipynb

Concatenate Lists Using +

- We can create a new list by adding two existing lists together

```
a = [1, 2, 3]
b = [4, 5, 6]
c = a + b
print(c)
print(a)
print(b)
```

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

7 data structures. ipynb

Lists can be Sliced Using

- Just like in strings, the second number is "up to but not including"

```
t = [9, 41, 12, 3, 74, 15]
print(t[1:3])
print(t[:4])
print(t[3:])
print(t[:])
```

```
[41, 12]
[9, 41, 12, 3]
[3, 74, 15]
[9, 41, 12, 3, 74, 15]
```

7 data structures. ipynb

Build a List from Scratch

- We can create an empty list then add elements using append method
- The list stays in order and new elements are added at the end of the list

```
stuff = list()
stuff.append('book')
stuff.append(99)
print(stuff)
stuff.append('cookie')
print(stuff)
```

```
['book', 99]
['book', 99, 'cookie']
```

7 data structures. ipynb

Is Something in a List?

- Python provides two operators that let you check if an item is in a list
- These are logical operators that return True or False
- They do NOT modify list

```
t = [9, 41, 12, 3, 74, 15]
print(12 in t)
print(20 in t)
print(20 not in t)
```

```
True
False
True
```

7 data structures. ipynb

Lists are in Order

- A list can hold many items and keeps those items in the order until we do something to change the order
- A list can be sorted (i.e., change the order)
- The sort method means "sort yourself"

```
friends = ['Tony', 'Peter', 'Natasha']
friends.sort()
print(friends)
print(friends[0])
```

```
['Natasha', 'Peter', 'Tony']
Natasha
```

7 data structures. ipynb

Built-in Functions and Lists

- There are a number of functions build into Python that take lists as parameters
- Remember the loops we built?
  - There are much simpler

```
nums = [9, 41, 12, 3, 74, 15]
print(len(nums))
print(max(nums))
print(min(nums))
print(sum(nums))
print(sum(nums)/len(nums))
```

```
6
74
3
154
25.666666666666668
```

7 data structures. ipynb

Strings and Lists

- Split breaks a string into parts and produces a list of strings
  - We think of these as words
- We can access a particular word or loop through all the words

```
abc = 'with three words'
stuff = abc.split()
print(stuff)
print(len(stuff))
print(stuff[0])
```

```
['with', 'three', 'words']
3
with
```

```
for w in stuff:
    print(w)
```

```
with
three
words
```

7 data structures. ipynb

Strings and Lists

- When you do not specify a delimiter, multiple spaces are treated like one delimiter
- You can specify what delimiter character to use in the splitting

```
line = 'a lot          of spaces'
etc = line.split()
print(etc)
```

```
['a', 'lot', 'of', 'spaces']
```

```
line = 'first;second;third'
thing = line.split()
print(thing)
print(len(thing))
```

```
['first;second;third']
1
```

```
thing = line.split(';')
print(thing)
```

```
['first', 'second', 'third']
```

7 data structures. ipynb

The Double Split Pattern

- Sometimes we split a line one way, and then grab one of the pieces of the line and split that piece again

```
data = 'From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008'
words = data.split()
print(words)
email = words[1]
pieces = email.split('@')
print(pieces)
```

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

7 data structures. ipynb

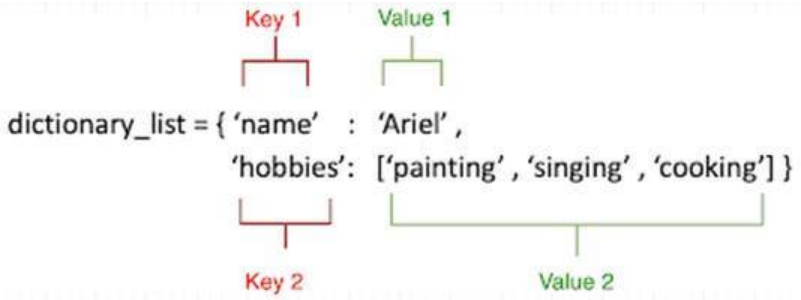
Dictionaries

A Story of Two Collections

- List
  - A linear collection of values that stay in order
- Dictionary
  - A "bag" of values, each with its own label

Dictionaries

- Dictionaries are Python's most powerful data collection
- Dictionaries allow us to do fast database-like operations in Python
- Dictionaries have different names in different languages
  - Associative arrays – Perl / PHP
  - Map or HashMap – Java
  - Property Bag – C# / .Net



Dictionaries

- Lists index their elements based on the position in the list
- Dictionaries are like bags
  - No order
- So we index the things we put in the dictionary with a "lookup tag"
- No two same keys in one dictionary

```
purse = dict()
purse['money'] = 12
purse['candy'] = 3
purse['tissues'] = 75
print(purse)
print(purse['candy'])
purse['candy'] = purse['candy']+2
print(purse)
```

```
{'money': 12, 'candy': 3, 'tissues': 75}
3
{'money': 12, 'candy': 5, 'tissues': 75}
```

7 data structures. ipynb

Dictionaries and Lists

```
>>> lst = list()
>>> lst.append(21)
>>> lst.append(183)
>>> print(lst)
[21, 183]
>>> lst[0] = 23
>>> print(lst)
[23, 183]

>>> ddd = dict()
>>> ddd['age'] = 21
>>> ddd['course'] = 182
>>> print(ddd)
{'course': 182, 'age': 21}
>>> ddd['age'] = 23
>>> print(ddd)
{'course': 182, 'age': 23}
```

List

Key	Value
[0]	21
[1]	183

lst

Dictionary

Key	Value
['course']	182
['age']	21

ddd

- Dictionaries are like lists except that they use keys instead of numbers to look up values

Dictionary Literals (Constants)

- Dictionary literals use curly braces and have a list of key : value pairs
- You can make an empty dictionary using empty curly braces

```
jjj = {'chuck': 1, 'fred': 42, 'jan': 100}
print(jjj)
ooo = { }
print(ooo)
```

```
{'chuck': 1, 'fred': 42, 'jan': 100}
{}
```

7 data structures. ipynb

Tuples

Tuples Are Like Lists

- Tuples are another kind of sequence that functions much like a list
  - They have elements which are indexed starting at 0

```
x = ('Tony', 'Peter', 'Natasha')
print(x[2])
y = (1, 8, 2)
print(max(y))
```

```
Natasha
8
```

7 data structures. ipynb

Tuples are Immutable

- Unlike a list, once you create a tuple, you cannot alter its content
  - Similar to a string

```
y = (1, 8, 2)
y[2] = 0
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-39-959605bf499f> in <module>
      1 y = (1, 8, 2)
----> 2 y[2] = 0

TypeError: 'tuple' object does not support item assignment
```

7 data structures. ipynb

Things not to do with Tuples

```
>>> x = (3, 2, 1)
>>> x.sort()
Traceback:
AttributeError: 'tuple' object has no attribute 'sort'
>>> x.append(5)
Traceback:
AttributeError: 'tuple' object has no attribute 'append'
>>> x.reverse()
Traceback:
AttributeError: 'tuple' object has no attribute 'reverse'
```

Tuples and Assignment

- We can also put a tuple on the left-hand side of an assignment statement

```
(x, y) = (4, 'fred')
print(y)
(a, b) = (99, 98)
print(a)
```

```
fred
99
```

7 data structures. ipynb

Tuples and Dictionaries

- The items() method in dictionaries returns a list of (key, value) tuples

```
purse = dict()
purse['money'] = 12
purse['candy'] = 3
purse['tissues'] = 75
for (k, v) in purse.items():
    print(k, v)
```

```
money 12
candy 3
tissues 75
```

7 data structures. ipynb

Tuples are Comparable

- The comparison operators work with tuples and other sequences.
- If the first item is equal, Python goes on to the next element, and so on, until it finds elements that differ.

```
(0, 1, 2) < (5, 1, 2)
```

```
True
```

```
(0, 1, 200000) < (0, 3, 4)
```

```
True
```

```
('Jones', 'Sally') < ('Jones', 'Sam')
```

```
True
```

```
('Jones', 'Sally') > ('Adams', 'Sam')
```

```
True
```

7 data structures. ipynb

Sets

Sets

- Unordered collection of unique values
  - Mutable and unordered

```
set1 = {'crunchy frog', 'ram bladder', 'lark vomit','harry potter', 'sun wu kong'}
set1
```

```
{'crunchy frog', 'harry potter', 'lark vomit', 'ram bladder', 'sun wu kong'}
```

```
# cannot use index to reach element of sets
set1[0]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-3-164047c4169c> in <module>
      1 # cannot use index to reach element of sets
----> 2 set1[0]

TypeError: 'set' object is not subscriptable
```

7 data structures. ipynb

Sets

- All the elements are unique
  - No duplicates

```
# add by value
set2 = set([1,2,3,3,3,4])
print(set2)
print("The length of the set is ", len(set2))

set2.add("B")
print(set2)
```

```
{1, 2, 3, 4}
The length of the set is  4
{1, 2, 3, 4, 'B'}
```

7 data structures. ipynb

Acknowledgements / Contributions

- Some of the slides used in this lecture from:
  - Charles R. Severance - University of Michigan School of Information

Thank You