LISTS & DICTIONARIES

LECTURE 05-2

JIM FIX, REED COLLEGE CSCI 121

READING FOR PYTHON LISTS

- **Reading:**
  - TP Ch 8-10
  - CP Ch 2.1-2.4

LIST "ARITHMETIC"

- We can build new lists from other list's contents using **+** and *****:

```
>>> [1,2,17] + [111,8]
[1, 2, 17, 111, 8]
>>> [1,2,17] * 4
[1, 2, 17, 1, 2, 17, 1, 2, 17, 1, 2, 17]
>>> [1,2,17] + []
[1, 2, 17]
>>> [] + [1,2,17]
[1, 2, 17]
>>> [1,2,17] * 1
[1, 2, 17]
>>> [1,2,17] * 0
[]
>>> [] * 4
[]
>>> [] + []
[]
```

LIST "SLICING"

- We can build new lists by copying portions of other lists:

```
>>> xs = [45,1,8,17,100,6]
>>> xs
[45, 1, 8, 17, 100, 6]
>>> xs[2:5]           # Build a new list from the 2,3,4 slice.
[8, 17, 100]
>>> xs[2:4]           # Build a new list from the 2,3 slice.
[8, 17]
>>> xs[:4]            # Build a new list from the 0,1,2,3 slice.
[45, 1, 8, 17]
>>> xs[4:]            # Build a new list from the 4,5 slice.
[100, 6]
>>> ys = xs[:]        # Build a new list as a full copy.
>>> xs[1] = 121
>>> xs
[45, 121, 8, 17, 100, 6]
>>> ys
[45, 1, 8, 17, 100, 6]
```

SLICING `[start:stop:step]`

- default values are `[0,len,1]`
- or `[-1,-(len+1),-1]`

LISTS OF LISTS

- Lists can be stored within other lists.

```
>>> lls = [[45,19],[8],[17,100,6],[]]
>>> lls[2]
[17, 100, 6]
>>> lls[2][0]
17
>>> lls[2][0] = 7777
>>> lls
[[45, 19] ,[8] ,[7777, 100, 6], []]
>>> lls[0].pop()
19
>>> lls[0].extend([0,0,0])
>>> lls
[[45, 0, 0, 0],[8],[7777, 100, 6],[]]
>>> lls.append([5,4,3,2])
>>> lls
[[45, 0, 0, 0], [8], [7777, 100, 6], [], [5, 4, 3, 2]]
```

TWO PRINTING PROCEDURES

- This procedure outputs the contents of a list.

```
def output_using_while(xs):
    i = 0
    while i < len(xs):
        print(xs[i])
        i = i + 1
```

- This procedure also outputs the contents of a list.

```
def output_using_for(xs):
    for x in xs:
        print(x)
```

WHILE VS. FOR LOOPS IN GENERAL

- This code snippet prints 0, 1, 2, 3, 4 (one number per line)

```
i = 0
while i < 5:
    print(i)
    i = i + 1
```

- This code snippet also prints 0, 1, 2, 3, 4 (one number per line)

```
for i in range(5)
    print(i)
```

`range(start, stop, step)`

- default values are `start = 0` and `step = 1` and optional
- loop until value is `stop - 1`

WHILE VS. FOR LOOPS

WHILE loops

- unbounded number of iterations
- can end early via break
- can use a counter but must initialize before loop and increment it inside loop
- may not be able to rewrite a while loop using a for loop

FOR loops

- know number of iterations
- can end early via break
- uses a counter or list or dict
- can rewrite a for loop using a while loop

Source: MIT 6.001 Open Course Ware

PYTHON LIST SUMMARY ENHANCED WITH FOR

- List creation via enumeration, concatenation, repetition, slicing:
  `[3,1,7]  []  [1,2]+[3,4,5]  [1,2]*4  xs[3:5]  xs[3:]  xs[:]`
- Accessing contents by index; list length:
  `xs[3]  xs[-1]  len(xs)`
- Updating contents by indexed assignment:
  `xs[3] = 5`
- Modifying/mutating a list object:

```
xs.append(5)    xs.extend([8,9,10])    xs.insert(2,357)
xs.pop()        del xs[6]
```

- Checking membership, content equality, object identity:

```
3 in xs     xs == [1,2,3]     xs is ys
```

- Scan according to index using a **`while`** loop.
- Loop through the contents using a **`for`** loop.

OUR SECOND DATA STRUCTURE: DICTIONARIES

- Python lets you store a collection of *associations*

```
>>> d = {"bob":35, "mel":24, "betty":29}
>>> d
{'bob': 35, 'mel': 24, 'betty': 29}
>>> d['bob']
35
>>> d['mel']
24
```

CREATE

READ

- This is a built-in data structure called a Python ***dictionary***.
  - A dictionary contains a collection of ***entries***.
  - The left part of each entry is called its ***key***.
  - The right part is that key's ***associated value***.
  - There is *at most one entry* for a key.
- A Python dictionary is our 2nd explicit example of a Python (data) ***object***

DICTIONARIES

- Python lets you store a collection of *associations*

```
>>> d = {"bob":35, "mel":24, "betty":29}
>>> d
{'bob': 35, 'mel': 24, 'betty': 29}
>>> d['bob']
35
>>> d['mel']
24
```

- This is a built-in data structure called a Python ***dictionary***.
  - It's also called a "key-value mapping", or sometimes just a "map".
  - Sometimes it's called a "hash table" or just "hashmap"

- In some languages, you mimic a dictionary with an "association list:"

```
d = [[“bob”, 35], [“mel”,24], [“betty”,29]]
```

SIMILARITIES BETWEEN LISTS AND DICTIONARIES

```
# Creating a list and a dictionary with the same information
fruits_list = ['apple', 'banana', 'cherry']
fruits_dict = {‘0': 'apple', ‘1': 'banana', ‘2': 'cherry'}

# Accessing the second item in the list and dictionary
print(fruits_list[1])  # Output: 'banana'
print(fruits_dict[‘1'])  # Output: 'banana'

# Modifying the second item in the list and dictionary
fruits_list[1] = 'orange'
fruits_dict[‘1'] = 'orange'

# Printing the modified list and dictionary
print(fruits_list) # ['apple', 'orange', 'cherry']
print(fruits_dict) # {‘0':'apple',‘1':'orange',‘2':'cherry'}
```

MODIFYING A DICTIONARY'S CONTENTS

- A Python dictionary is also a ***mutable*** data structure.
  - You can add new key-value pairs, or modify the associated value to a key.
  - The syntax for adding a new entry and updating an existing entry is the same

```
>>> d = {"bob":35, "mel":24, "betty":29}
>>> d
{'bob': 35, 'mel': 24, 'betty': 29}
>>> d['mel']
24
>>> d['mel'] = 25                                         UPDATE
>>> d['mel']
25
>>> d
{'bob': 35, 'mel': 25, 'betty': 29}
>>> d['lou'] = 87                                         UPDATE
>>> d
{'bob': 35, 'mel': 24, 'betty': 29, 'lou': 87}
```

DICTIONARY CONTENT CHECKS

```
>>> d = {"bob":35, "mel":24, "betty":29, "lou": 87}
>>> 'mel' in d      # Does the dictionary contain a key?
True
>>> 'jim' in d
False
>>> 35 in d
False
>>> e =  {"lou": 87,"mel":24, "betty":29, "bob":35}
>>> e == d          # Are the dictionary's contents the same?
True
>>> e is d          # Are they the same object?
False
>>> len(d)          # Get the number of entries.
4
```

BUILDING AND MODIFYING A DICTIONARY

```
>>> d = {}
>>> d['bob'] = 35
>>> d['betty'] = 29
>>> d['mel'] = 24
>>> d
{'bob': 35, 'mel': 24, 'betty': 29}
>>> del d['betty']
>>> d
{'bob': 35, 'mel': 24}
>>> d.pop(‘mel’)
24
>>> d
{‘bob’: 35}
```

CREATE

DELETE

DELETE

LOOPING

```
>>> dict = {}
>>> dict = {“bob”:35, “betty”:29, “mel”:24}
>>> for key in dict:
...     print(key + ” -> ” + str(dict[key]))
...
bob -> 35
betty -> 29
mel -> 24
>>>
```

- A `for` loop runs through the ***keys*** of the dictionary.
  - You can then look up the associated value.

Compare to looping through a list:

```
>>> lst = {“alice”, “bob”, “carl”}
>>> for e in lst:
...     print(e)
```

PYTHON DICTIONARY SUMMARY

- List creation via enumeration of some associations:
  ```
  {'a':89,'b':4}      {}
  ```
- Accessing contents by key; dictionary size:
  ```
  d['a']              len(d)
  ```
- Updating an entry's associated value with key re-assignment:
  ```
  d['a'] = 88
  ```
- Modifying/mutating a dictionary to add/remove entries:
  ```
    d['c'] = 111
    del d['b']
  ```
- Checking key inclusion, content equality, object identity:
  ```
    'a' in d       d == {'e':78}      d1 is d2
  ```
- Loop through the keys using a `for` loop.

LIST VS. DICTIONARY

LIST

- ordered sequence of elements
- look up an element by an integer index
- indices have an order
- index is an integer

DICTIONARY

- matches “keys” to “values
- look up an item by another item
- no order is guaranteed
- key can be any immutable type

Source: MIT 6.001 Open Course Ware