Lists and Dictionaries again

zyBooks
"Programming in Python 3"
chapter 8

List

- A *collection* of objects
  - Written using **[** square brackets **]**
- Each object can be changed
- Lists can get longer, shorter

- Create two lists:
  - `list_1 = [ 'hello', 7, 'Goodbye', 0 ]`
  -
```
list_2 = [ ]
for x in range(10):
    list_2.append( x**2 )
```

Access List Elements

- *Index* notation specifies individual elements
  - list_1[2]
- *Slice* notation specifies groups of elements
  - Similar to index notation
  - list_2[3 : 7]
  - Slices are also lists, just smaller ones
- Indexes start at 0
  - So list_1[2] is actually the third element of the list
- Index -1 is always the *last* element
  - -2 is next-to-last, etc.

Some Things to Do With a List

Action	Example
Create a list	v = [1, 2, 3,] v2 = [ ]
Convert something into a list	lines = list(h.readlines())
Add to the end of a list	v.append(4)
Add anywhere in a list	v.insert(2, 9)
Concatenate (join) two lists	v2 = lines + v v2.extend([11, 12, 13])
Access one element of a list	x = v2[1]
Slice a list	sl = v2[2 : 3]

practice

- read lines from a file
  - v = handle.readlines()
  - type of v?
- How many lines?

More Things to Do With a List

Action	Example
Sort a list "in place"	v.sort()
Make a sorted version of a list	s = sorted(v2)
Flip a list end-for-end	v.reverse()
Add anywhere in a list	v.insert(2, 9)
Extract an element from a list	y = v2.pop(0)
Remove an element from a list	v2.remove(9)
Count # of occurrences of an item in a list	c = v2.count(' the ')
Find an item in a list	v2.index(3)

practice – "concordance"

- read words from a file into a list
- make sorted copy of list
- reverse sorted list (two ways)
- find a word's location in list
- count # of occurrences of a word in a list

Iterating Through a List

- Use the list's length
  - ```
    for i in range( len( mylist ) ):
        # do something with mylist[i]
    ```
- Get the items directly
  - ```
    for item in mylist:
        # do something with item
    ```
- Use the enumerate() function
  - ```
    for i, item in enumerate( mylist ):
        # do something with item and its index i
    ```
    ```

practice – "concordance 2"

- Iterate through list of words
- Build dictionary
  - Each value is list of indexes

Nested Lists

- List elements can be anything
  - Can be another list
- Example:
  - ```
    family = [ ]
    family.append( ['parent', 'Esther'] )
    family.append( ['parent', 'Pedro'] )
    family.append( ['daughter', 'Faith'] )
    family.append( ['son', 'Robert'] )
    family.append( ['pet', 'Ginger'] )

    import pprint
    pprint.pprint( family )
    ```
    ```

Multiplication Table, Stored In a List

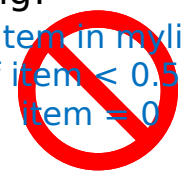
```
mtable = [ ]
for m1 in range(1, 12+1):
    row = [ ]
    for m2 in range(1, 12+1):
        row.append( m1 * m2 )
    mtable.append(row)

import pprint
pprint.pprint( mtable )

print()

for row in mtable:
    for col in row:
        print(' {:3d}'.format( col ), end='')
    print( )
```

Slices and Strides

- List slice – a sublist, ranging from starting index to (but not including) ending index
  - midrows = mtable[5 : 9]
- Default slices includes *every* element in range
  - Selected elements spaced 1 element apart
- Slice *stride* – optional distance between elements of slice
  - Selected elements space by the stride size
  - oddrows = mtable[1 : 12 : 2]
    evenrows = mtable[0 : 13 : 2]

Modifying a List in a Loop

- Cannot just modify elements directly, must work on indexed positions within list
- Wrong!
  - ```
    for item in mylist:
        if item < 0.5:
            item = 0
    ```



- Right…
  - ```
    for i, item in enumerate(mylist):
        if item < 0.5:
            mylist[ i ] = 0
    ```

List Comprehensions

- Fast way to build a list
- Example:
  - ```
    list_2 = [ ]
    for x in range(10):
        list_2.append( x**2 )
    ```
  - becomes
  - ```
    list_2 = [ x**2  for x in range(10) ]
    ```
- List of random numbers:
  - ```
    list_3 = [ random.random()  for x in range(500) ]
    ```
- Get inputs:
  - ```
    list_vals = [ float(input('value?'))  for x in range(8) ]
    ```

More Examples

- Read an input file with *one* number per line:
  - ```
    with open('number_file1.txt', 'r') as h:
        numlist = [ float( line )  for line in h.readlines() ]
    ```
- Input file with *multiple* numbers per line:
  - ```
    with open('number_file2.txt', 'r') as h:
        numlist =  [ ]
        for line in h.readlines():
            numlist += [ float( t )  for t in line.split() ]
    ```
  - Or, with nested comprehensions:
  - ```
    with open('number_file2.txt', 'r') as h:
        numlist = [ float(t) for line in h.readlines() for t in
    line.split() ]
    ```

Conditional List Comprehensions

- Add test to end of comprehension
  - Useful for *filtering* list comprenhensions
- Make a list of randoms:
  - ```
    rands = [ random.random() for i in range(100) ]
    ```
- Now select only larger values:
  - ```
    bigrands = [ r for r in rands if r > 0.5 ]
    ```
    ```

Comprehension practice

- Build a list of powers-of-2
- Build a list of [powers-of-2, powers-of-16] lists

- Read a text file all at once
- Capitalize each word

- Read a text file, line-by-line
- Capitalize each word on each line

Sorting Lists

- .sort() method – sort a list *in place*
- sorted() function – make a sorted copy of a list
- reverse=True
  - Sort in reversed order
- key= …
  - Specify how sorting is done
  - Value is a function name, that is applied to each list element

Sorting Example

```
family = [ ]
family.append( ['parent', 'Esther'] )
family.append( ['parent', 'Pedro'] )
family.append( ['daughter', 'Faith'] )
family.append( ['son', 'Robert'] )
family.append( ['grandparent', 'Robert'] )
family.append( ['grandparent', 'Annie'] )
family.append( ['grandparent', 'Abdon'] )
family.append( ['grandparent', 'Fortunata'] )

def byname(element):
    return element[1]

family2 = sorted( family, key=byname )
```

Sorting practice

- Read a text file
  - build a list of lines
  - build a list of list-of-words-on-each-line
- Sort list of lines alphabetically
- Sort list of lines by length of line
- Sort list of word-lists by length of word-lists
- Sort list of word-lists alphabetically, according to alphabetically-sorted words

Command-Line Arguments

- Command-line script can include optional arguments
  - Each argument is a separate string
- Arguments available in **sys.argv** list
  - OS-related definitions in **sys** module
- First element of sys.argv is always the script's filename
  - May be the *only* element

Examples: Command-Line Arguments

- Just show all command-line arguments, including script's name:
  - ```
    for arg in sys.argv:
        print( arg )
    ```
- Convert arguments to list of numbers:
  - ```
    numlist = [ ]
    for numstr in sys.argv[1:] :
        numlist.append( float(numstr) )
    ```
  - or, as a comprehension:
  - ```
    numlist = [ float(ns) for ns in sys.argv [1:] ]
    ```
    ```

Small exercise

- Find average of all numbers provided as command-line arguments
  - viz. :

```
myscript  12.05  -13  0.0001  204 8
```

Dictionaries

- Similar to lists:
  - *Collection* of elements
  - *Mutable* – elements can be added, deleted, changed
  - Members can be accessed using an "index" or "key"
- More general than lists:
  - Lists use *numeric* indexes for elements, that identify relative position within list
  - Dictionaries use *keys* that can be of any (non-mutable) type
    - numbers, strings, tuples
  - Dictionaries do not have intrinsic order
    - newer implementations use order-of-insertion

Dictionary Methods

- *my_dict1*.get(key, *default*)
  - Alternative to *dict*[key] – returns key's value
  - Returns *default* value if key isn't found
- *my_dict1*.update(my_*dict2*)
  - Merges *dict2* into *my_dict1*
  - Overwrites shared key values with new values
- *my_dict1*.pop(key, *default*)
  - Returns key's value, and removes key from *my_dict1*
  - Returns *default* value if key isn't found
- *my_dict1*.clear()
  - Removes (discards) all items from *my_dict1*

Dictionary Iterations

- Get keys only: *my_dict1*.keys()
  - for k in *my_dict1*.keys():
    print(*my_dict1*[k])
  - for k in sorted(*my_dict1*.keys()):
    print(*my_dict1*[k])
- Get values only: *my_dict1*.values()
  - for v in *my_dict1*.values():
    print(f)
- Get both keys and values: *my_dict1*.items()
  - for k, v in *my_dict1*.items():
    print('value {} at key {}'.format(v, k))

Dictionary Comprehensions

- Students list:
  - names =
    ['Alice', 'Bob', 'Carol', 'David', 'Emily', 'Frank']
- Grades list:
  - grades = [80, 75, 75, 90, 95, 70]

- Grades dictionary:
  - grades_dict =
    { names[i] : grades[i] for I in range(len(names)) }