THE UNIVERSITY OF WESTERN AUSTRALIA

Lecture 20
Lists to Dictionaries

Objectives

- To revise operations related to sequences including lists
- To learn about new sequences: Tuple and Dictionary

Revision - Lists

- String and lists are subclasses of sequence
  - *Lists are mutable, but strings are not*
- Items in a list or string are obtained by indexing, with list (and string) items numbered from 0
- Lists can contain items of different types, e.g.

```
[1, 2.0, "three"]
```

- Lists are dynamic (they grow and shrink as required).

Revision: Sequence Operations

Operator	Meaning
<seq> + <seq>	Concatenation
<seq> * <int-expr>	Repetition
<seq>[]	Indexing
len(<seq>)	Length
<seq>[:]	Slicing
for <var> in <seq>:	Iteration
<expr> in <seq>	Membership (Boolean)

Revision: Sequence Operations

- Except for the membership check, we've used these operations before on strings.
- The membership operation can be used to see if a certain value appears anywhere in a sequence.

```
>>> lst = [1,2,3,4]
>>> 3 in lst
True
>>> month = 1
>>> year = 2000
>>> if month in [1,2] :  # month == 1 or month == 2
      year -= 1
```

Revision: List Operations

Method	Meaning
<list>.append(x)	Add element x to end of list.
<list>.sort()	Sort the list. A comparison function may be passed as a parameter. By default, sorted in ascending order
<list>.reverse()	Reverse the list.
<list>.index(x)	Returns index of first occurrence of x.
<list>.insert(i, x)	Insert x into list at index i.
<list>.count(x)	Returns the number of occurrences of x in list.
<list>.remove(x)	Deletes the first occurrence of x in list.
<list>.pop(i)	Deletes the i^{th} element of the list and returns its value.

Revision: List Operations

```
>>> lst = [3, 1, 4, 1, 5, 9]
>>> lst.append(2)
>>> lst
[3, 1, 4, 1, 5, 9, 2]
>>> lst.sort()
>>> lst
[1, 1, 2, 3, 4, 5, 9]
>>> lst.reverse()
>>> lst
[9, 5, 4, 3, 2, 1, 1]
>>> lst.index(4)
2
```

```
>>> lst.insert(4, "Hello")
>>> lst
[9, 5, 4, 3, 'Hello', 2, 1, 1]
>>> lst.count(1)
2
>>> lst.remove(1)
>>> lst
[9, 5, 4, 3, 'Hello', 2, 1]
>>> lst.pop(3)
3
>>> lst
[9, 5, 4, 'Hello', 2, 1]
```

Revision: List Operations

- Most of these methods don't return a value** – they change the contents of the list in some way.

```
>>> lst.sort()

>>> lst.sort(reverse=True)
```

- Lists can grow by appending new items and shrink when items are deleted. Individual items or entire slices can be removed from a list using the `del` operator.

** They return None

Revision: List Operations

- ```
  >>> myList=[34, 26, 0, 10]
  >>> del myList[1]
  >>> myList
  [34, 0, 10]
  >>> del myList[1:3]
  >>> myList
  [34]
  ```

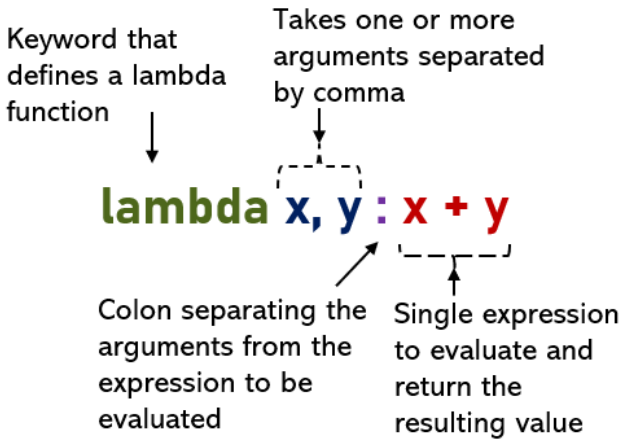
- `del` isn't a list method, but a built-in operation that can also be used on list items.
  ```

Map and filter

- map() and filter() are common functions that could transform, and filter sequences respectively, often with the help of ***lambda*** *functions*.

- What is a lambda function?

A **lambda** function is a small anonymous function which can take any number of arguments, but can only have one expression/return statement



We often use lambda functions when passing functions as arguments to other functions.

Map and filter

- **map()** is a function that can transform a sequence with the help of a **lambda** function.
- E.g., create a new list that has strings instead of the integers [8, 16, 19, 6]
  - *Pass the sequence and the lambda function as arguments the map() function*

```
>>> map_list = map(lambda n: str(n), [8, 16, 19, 6])
 #map returns a map object, which we can get in list form
by passing it to the list() function


>>> list(map_list)
['8','16','19','6']
```

map() applies a function to all the items in an input sequence

https://book.pythontips.com/en/latest/map_filter.html

Map and filter

- **filter()** can help create a list that contains only the values from another list that match some criteria.
- E.g., create a new list that only contains even integers [8, 16, 19, 6]

```
F_list = filter(lambda n: n % 2 == 0, [8, 16, 19, 6])
>>> list(F_list)
[8, 16, 6]
```

The filter() function returns a filter object, which we can pass to the list() function to get in the list form. The lambda function determines which items in the list remain (if the lambda function returns True) or are filtered out (if it returns False).

Tuples

- A *tuple* is a sequence which looks like a list but uses `()` rather than `[]`.
- Tuples are sequences that are **immutable**, so are used to represent sequences that are not supposed to change,
  - *e.g., student–mark pairs*
  - `[('Fred',55), ('Jemima',68), ('James',45)]`
  - *Sorting a list of tuples sorts on first member of each tuple*
  - *Turn a list into a tuple by using the* `tuple()` *function*

List of tuples

- Sorting list of tuples by second element

```
>>> t = [("Fred",55), ("Jemima",68), ("James",45)]
>>> t.sort(key=lambda x:x[1])
>>> t
[('James', 45), ('Fred', 55), ('Jemima', 68)]
```

Dictionaries

unorder, □ cant □ be □ sorted

- After lists, a dictionary is probably the most widely used collection/compound data type.
- Dictionaries are not as common in other languages as lists (arrays).
- Lists are sequential
  - *To find a particular need to search from the start.*
  - *Do you find a book in the library starting from Dewey number (000 is computer science!)*
    - Use catalogue!

Dictionaries

- Dictionaries use key-value pairs
- There are lots of examples!
  - *Names and phone numbers*
  - *Usernames and passwords*
  - *State names and capitals*
- A collection that allows us to look up information associated with arbitrary keys is called a mapping.
- Python dictionaries are *mapping*s. Other languages call them *hashes* or *associative arrays*.

Dictionaries

- Dictionaries can be created in Python by listing key-value pairs inside of curly braces.
- Keys and values are joined by `:` and are separated with commas.

```
>>>passwd = {"guido":"superprogrammer",
"turing":"genius", "bill":"monopoly"}
```

- We use an indexing notation to do lookups

```
>>> passwd["guido"]
'superprogrammer'
```

- Unlike list indexes, which are integers related to position in the list, dictionary indexes can be almost anything

Dictionaries

- `<dictionary>[<key>]` returns the object with the associated key.
- Dictionaries are mutable.

```
>>> passwd["bill"] = "bluescreen"

>>> passwd

{'guido': 'superprogrammer', 'bill':
'bluescreen', 'turing': 'genius'}
```

- Did you notice the dictionary printed out in a different order than it was created?

Initialising Dictionaries

- Dictionaries can be created directly

```
# dayno is in the range 0..6

dow_map = {0:"Sunday", 1:"Monday", 2:"Tuesday",
           3:"Wednesday”, 4:"Thursday", 5:"Friday",
           6:"Saturday"}

daystr = dow_map[dayno]

print(daystr)
```

Initialising Dictionaries

- Dictionaries can also be created incrementally. That is, start with an empty dictionary and add the key-value pairs one at a time.
- For example, assume the file `passwords` contains comma-separated pairs of user IDs and passwords

```
passwd_dir = {}
for line in open('passwords', 'r'):
    user, pw = line.strip().split(‘,’)
    passwd_dir[user] = pw
```

Summary

- We completed looking at Python lists, noting that many of the functions are actually methods that change the input list, esp. append and sort.
- We looked at tuples, as a special sort of list.
- We looked at dictionaries, as a mapping from keys to values which is not restricted to the order of items