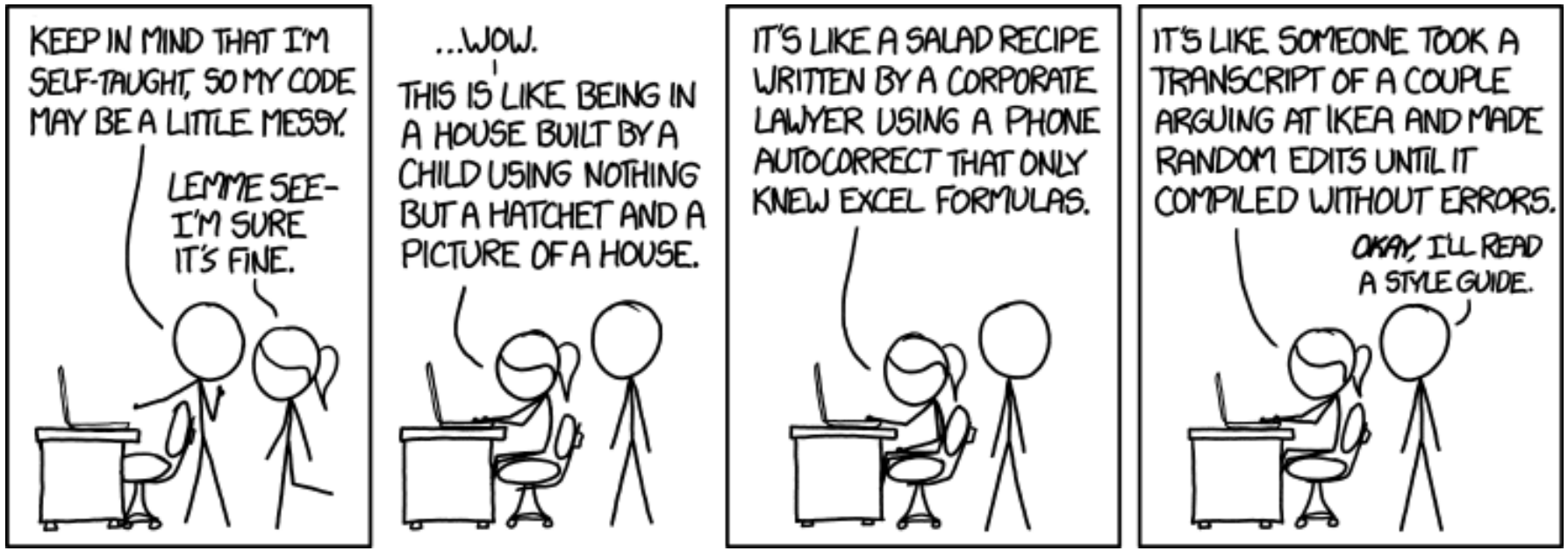
CSc 110, Autumn 2017

Lecture 27: Lists that change size and Tuples

Adapted from slides by Marty Stepp and Stuart Reges



List functions

Function	Description
`append(x)`	Add an item to the end of the list. Equivalent to `a[len(a):] = [x].`
`extend(L)`	Extend the list by appending all the items in the given list. Equivalent to `a[len(a):] = L`
`insert(i, x)`	Inserts an item at a given position. i is the index of the element before which to insert, so `a.insert(0, x)` inserts at the front of the list.
`remove(x)`	Removes the first item from the list whose value is *x*. Errs if there is no such item.
`pop(i)`	Removes the item at the given position in the list, and returns it. `a.pop()` removes and returns the last item in the list.
`clear()`	Remove all items from the list.
`index(x)`	Returns the index in the list of the first item whose value is *x*. Errs if there is no such item.
`count(x)`	Returns the number of times *x* appears in the list.
`sort()`	Sort the items of the list
`reverse()`	Reverses the elements of the list
`copy()`	Return a copy of the list.

Lists that change size

- Sometimes we don't know how big we want our list to be when our program starts
  - It can be useful to create an empty list and fill it up.

```
data = []
data.append("hello")
data.append("world")
print(data)                    # ['hello', 'world']
```

- How would we insert another word in the middle?

Exercise

Write a function called `remove_duplicates` that takes a **sorted** list of numbers and removes any duplicates. For example, if it is called on the following list:

```
data = [-2, 1, 1, 3, 3, 3, 4, 5, 6, 78, 78, 79]
```

after the call the list should be

```
data = [-2, 1, 3, 4, 5, 6, 78, 79]
```

Looping and removing

- When you loop through a list and remove elements you change the length of the list. This means you need to change your upper bound as you are looping.
  - **You must use a while loop when removing items from a list**
  - A `for i in range` loop won't work as it can't adjust when the length of the list changes
  - A `for num in data` loop won't work as it cannot alter the list.

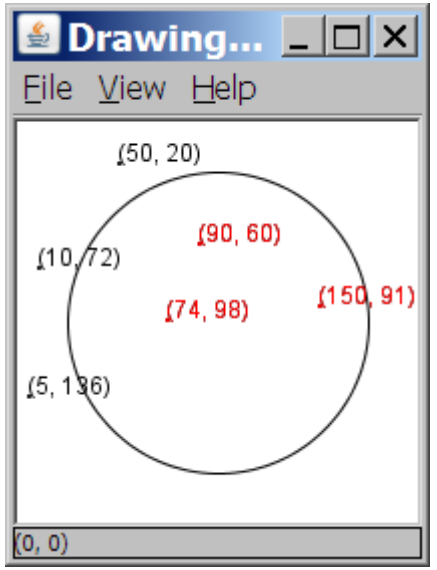
Solution

```
def remove_duplicates(data):
    i = 0
    while i < len(data) - 1:
        if data[i] == data[i + 1]:
            data.pop(i)
        else:                   # we don't want to move on
            i += 1              # to the next element if we
                                # remove as that will me we
                                # will skip the one that
                                # just moved back into the one
                                # we removed's place
```

A programming problem

- Given a file of cities' names and (x, y) coordinates:

```
Winslow 50 20
Tucson 90 60
Phoenix 10 72
Bisbee 74 98
Yuma 5 136
Page 150 91
```



- Write a program to draw the cities on a `DrawingPanel`, then simulates an earthquake that turns all cities red that are within a given radius:

```
Epicenter x? 100
Epicenter y? 100
Affected radius? 75
```

A bad solution

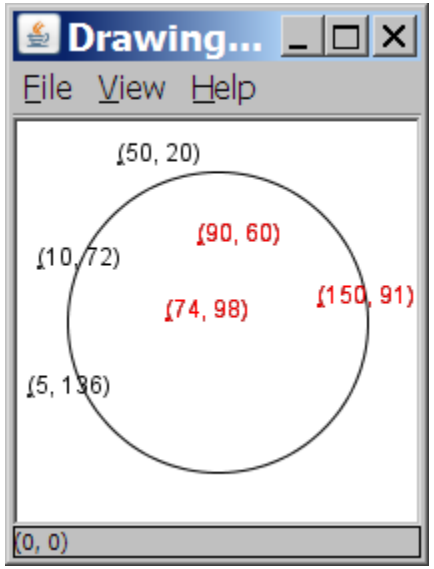
```
lines = open("cities.txt").readlines()
names = [0] * len(lines)
x_coords = [0] * len(lines)
y_coords = [0] * len(lines)

for i in range(0, len(lines)):
    parts = lines[i].split()
    names[i] = parts[0]
    x_coords[i] = parts[1]    # read each city
    y_coords[i] = parts[2]
...
```

- **parallel lists**: 2+ lists with related data at same indexes.
  - Considered poor style.

Observations

- The data in this problem is a set of points.
- It would be better stored together



Tuples

- A sequence similar to a list but it **cannot be altered**
- Good for storing related data
  - We mainly store the same **type** of data in a list
  - We usually store related things in tuples

- Creating tuples

name = (**data**, **other_data**, … , **last_data**)

```
tuple = ("Tucson", 80, 90)
```

Using tuples

- You can access elements using [] notation, just like lists and strings

```
tuple = ("Tucson", 80, 90)
low = tuple[1]
```

- You cannot update a tuple!
  - Tuples are immutable

- You can loop through tuples the same as lists

operation	call	result
len()	`len((1, 2, 3))`	`3`
+	`(1, 2, 3) + (4, 5, 6)`	`(1, 2, 3, 4, 5, 6)`
*****	`('Hi!',) * 4`	`('Hi!', 'Hi!', 'Hi!', 'Hi!')`
in	`3 in (1, 2, 3)`	`True`
for	`for x in (1,2,3): print x,`	`1 2 3`
min()	`min((1, 3))`	`1`
max()	`max((1, 3))`	`3`

Days till

- Write a function called `days_till` that accepts a start month and day and a stop month and day and returns the number of days between them

call	return
`days_till("december", 1, "december", 10)`	9
`days_till("novembeR", 15, "december", 10)`	25
`days_till("OCTober", 6, "december", 17)`	72
`days_till("october", 6, "ocTober", 1)`	360

Days till solution

```
def days_till(start_month, start_day, stop_month, stop_day):
    months = (('january', 31),('february', 28),('march', 31),('april', 30), ('may', 31),('june', 30),
              ('july', 31), ('august', 31),('september', 30), ('october', 31), ('november', 30), ('december', 31))

    if start_month.lower() == stop_month.lower() and stop_day >= start_day:
        return stop_day - start_day
    days = 0
    for i in range(0, len(months)):
        month = months[i]
        if month[0] == start_month.lower():
            days = month[1] - start_day
            i += 1
            while months[i % 12][0] != stop_month.lower():
                days += months[i % 12][1]
                i += 1
            days += stop_day
    return days
```