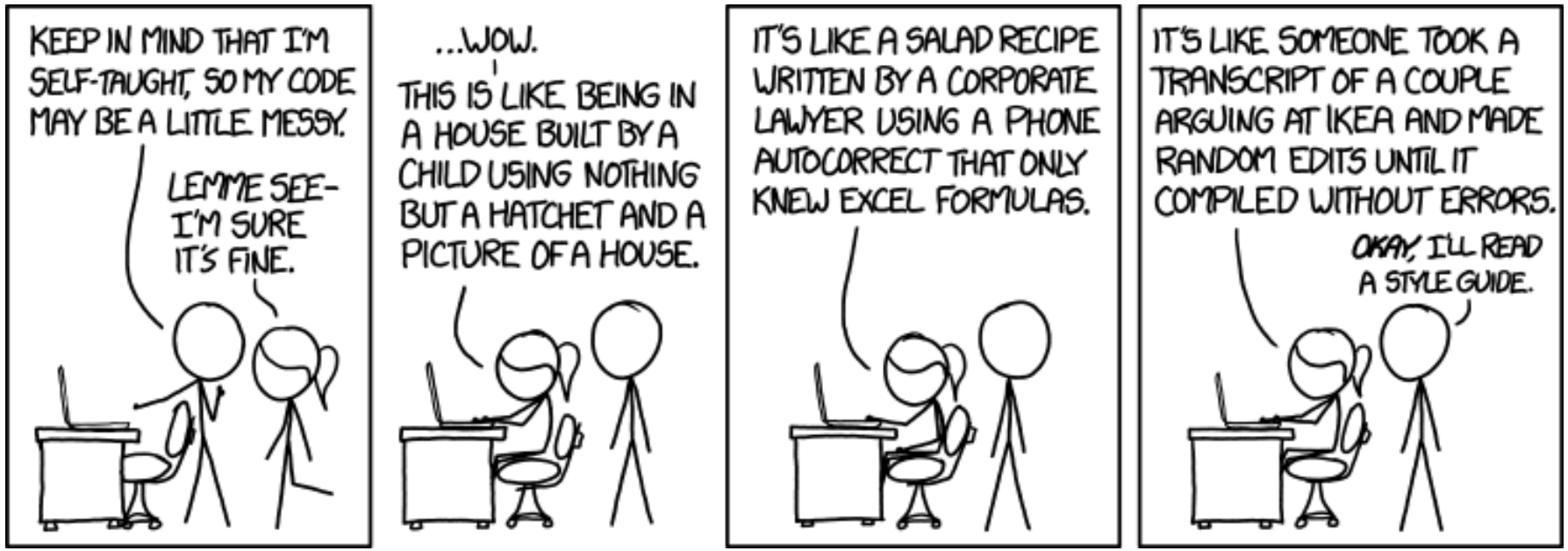
CSc 110, Autumn 2016

Lecture 23: Tuples

Adapted from slides by Marty Stepp and Stuart Reges



Black box testing example

Write a function named `remove_bad_pairs` that accepts as a parameter a list of integers, and removes any adjacent pair of integers in the list if the left element of the pair is larger than the right element of the pair. Every pair's left element is at an even-numbered index in the list, and every pair's right element is at an odd index in the list. For example, suppose a variable named `list` stores the following element values:

```
[3, 7, 9, 2, 5, 5, 8, 5, 6, 3, 4, 7, 3, 1]
```

We can think of this list as a sequence of pairs:

```
[3, 7, 9, 2, 5, 5, 8, 5, 6, 3, 4, 7, 3, 1]
```

The pairs 9-2, 8-5, 6-3, and 3-1 are "bad" because the left element is larger than the right one, so these pairs should be removed. So the call of `remove_bad_pairs(list)` would change the list to store:

```
[3, 7, 5, 5, 4, 7]
```

If the list has an odd length, the last element is not part of a pair and is also considered "bad;" it should therefore be removed by your function.

If an empty list is passed in, the list should still be empty at the end of the call.

What should you test?

- Focus on edge cases
  - Don't just test 1, 2, 3, 4, 5, 6, etc – that won't tell you anything new
- Common edge cases
  - 1, 0, -1
  - Empty list

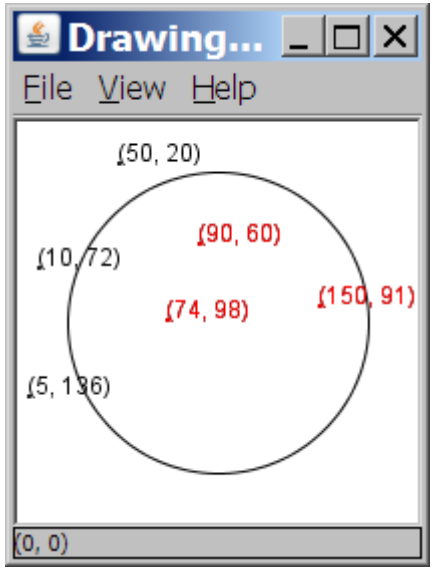
White box testing example

```
remove_bad_pairs(list):
    if (len(list) % 2 != 0):
        v.remove(list[-1])
    i = 0
    while(i < len(list)):
        if (i % 2 != 0 and list[i - 1] > list[i]):
            list.remove(i - 1)
            list.remove(i)
```

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

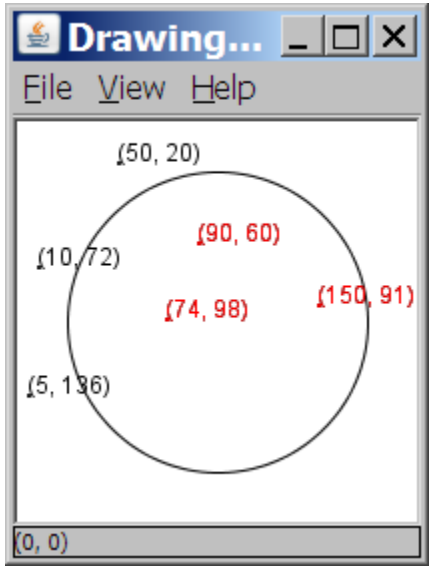
```
lines = open("cities.txt").readlines()
names = [0] * len(lines)
x_coords = [0] * len(lines)
y_coords = [0] * len(lines)

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
```