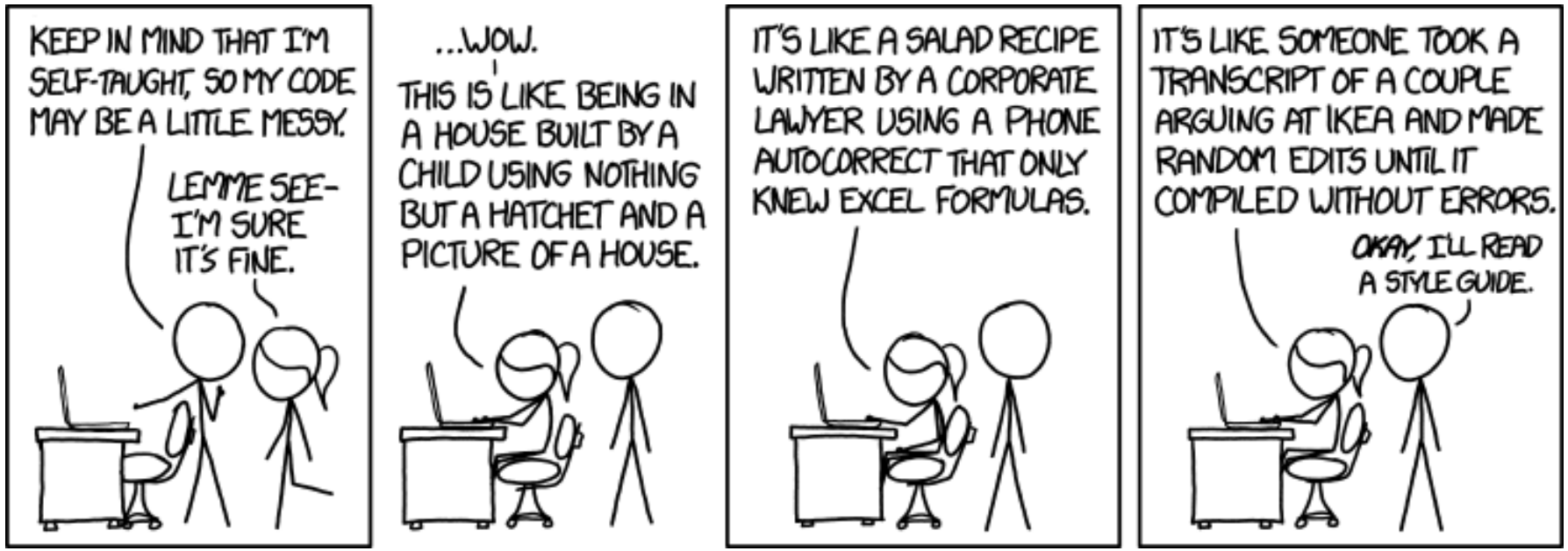
CSc 110, Spring 2018

Lecture 27: Lists Tuples

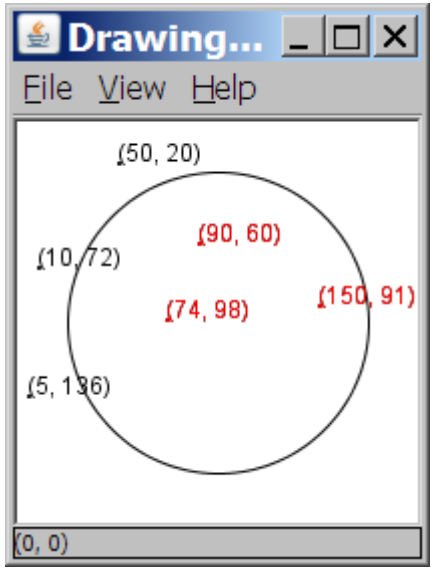
Adapted from slides by Marty Stepp and Stuart Reges



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

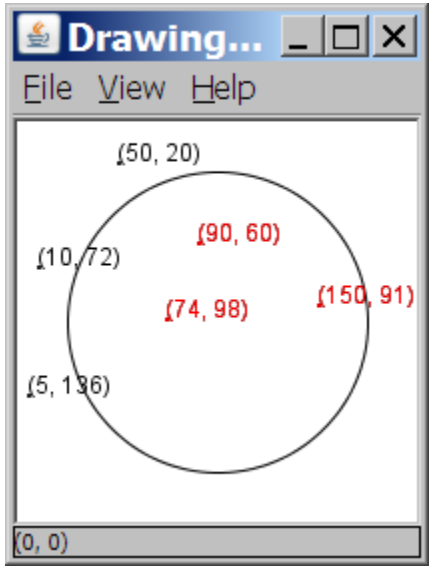
```
lines = open("cities.txt").readlines()
names = [0] * len(lines)
x_coords = [0] * len(lines)
y_coords = [0] * len(lines)

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
```

Output to files

- Open a file in write or append mode
  - 'w' - write mode – replaces everything in the file
  - 'a' – append mode – adds to the bottom of the file preserving what is already in it

```
name = open("filename", "w")      # write
name = open("filename", "a")      # append
```

Output to files

name`.write(`**str**`)` – writes the given string to the file

name`.close()` – closes file once writing is done

Example:

```
out = open("output.txt", "w")
out.write("Hello, world!\n")
out.write("How are you?")
out.close()
text = open("output.txt").read()   # Hello, world!\nHow are you?
```