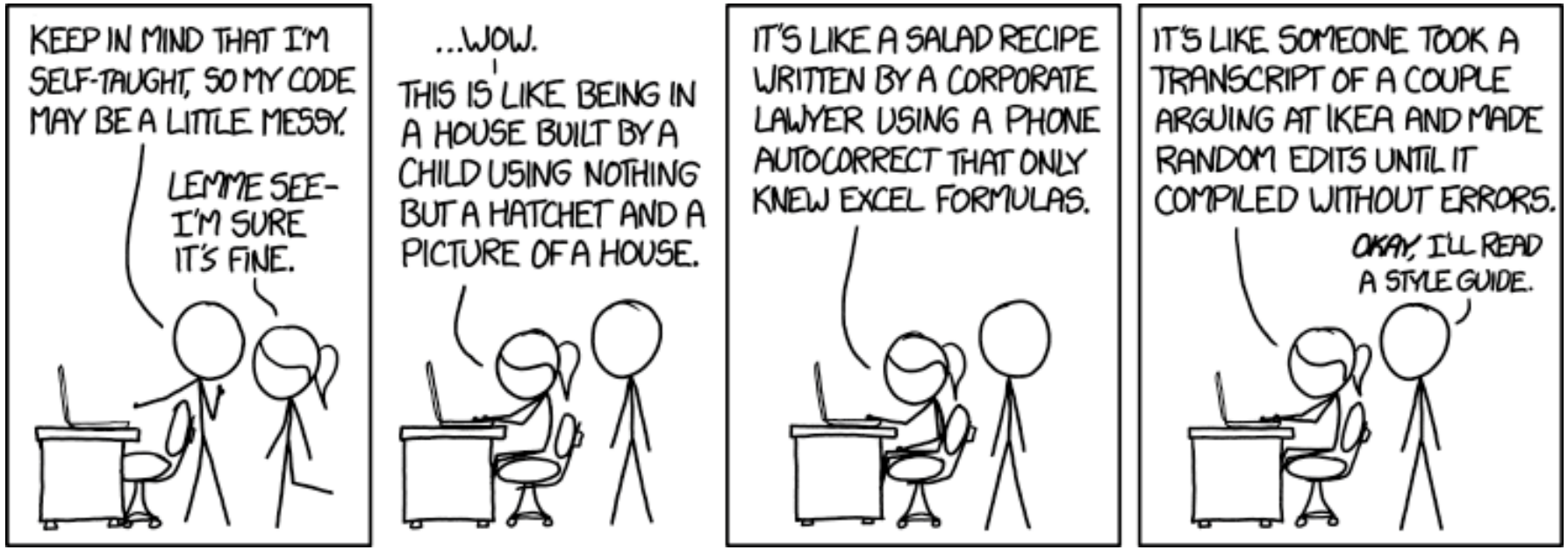
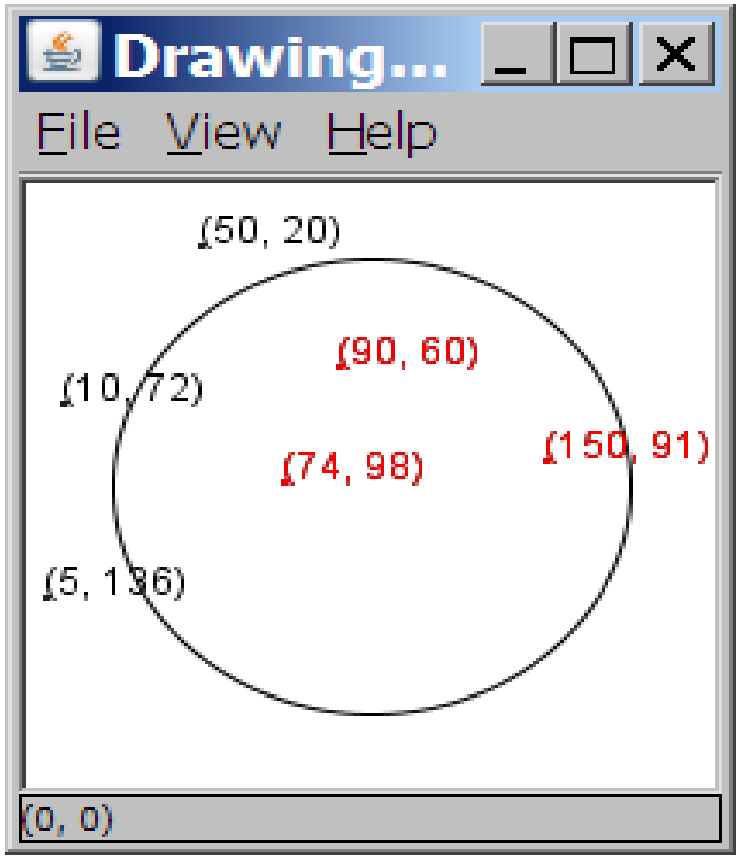
CSc 110, Spring 2017

Lecture 23: 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 poor 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]          # city name
    x_coords[i] = parts[1]
    y_coords[i] = parts[2]
...
```

- What's bad about this solution?

A poor solution

```
    names[i] = parts[0]      # city name
    x_coords[i] = parts[1]
    y_coords[i] = parts[2]
...
```

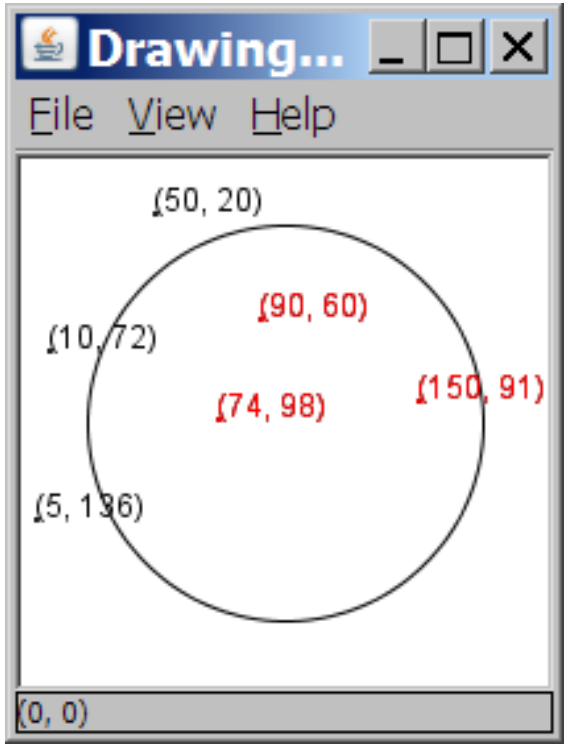
- **Parallel lists**: two or more lists with related data at the same indices.
- Parallel lists can easily lead to bugs:
  - may get "out of sync" if you add an x-coordinate but not a y-coordinate
- Would have to pass all three lists as parameters to a function.
- Is there a better representation?

Observations

- Each item in the data set is a name, an x-coordinate and y-coordinate for a given city

```
Winslow 50 20
```

- It would be better to associate these values



Tuples

Good for associating a fixed number of items

Syntax for creating a tuple:

(**value0, value1, … ,valueN**)

Example:

```
("Tucson", 90, 60)
```

Tuples can be subscripted just like lists and strings:

```
>>> t = ("Tucson", 90, 60)
>>> t
('Tucson', 90, 60)
>>> t[0]
'Tucson'
```

Tuples vs. lists

- Tuples
  - tuples hold a fixed number of items
  - the items in a tuple cannot be assigned to

```
>>> t = ("Tucson", 90, 60)
>>> t
('Tucson', 90, 60)
>>> t[0] = "OldPueblo"
…
TypeError: 'tuple' object does not support item assignment
```

- Lists
  - lists may grow or shrink
  - the items in a list can be assigned to
  - typically a list holds values of the same type (e.g., all integers or all strings)

Using tuples

- As mentioned, tuples are subscripted just like lists and strings

```
t = ("Tucson", 90, 60)
x_coord = t[1]
```

- You can loop through tuples the same as lists and strings

operation	call	result
len()	`len((1, 2, 3))`	`3`
+	`(1, 2, 3) + (4, 5, 6)`	`(1, 2, 3, 4, 5, 6)`
*****	`('Hi', 1) * 2`	`('Hi', 1, 'Hi', 1)`
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`

Using tuples

```
>>> book = ("Pride and Prejudice", "Austin", 1813, "Fiction")
>>> book
('Pride and Prejudice', 'Austin', 1813, 'Fiction')
>>> len(book)
4
>>> "Fiction" in  book
True
>>> for item in book:
      print(item)

Pride and Prejudice
Austin
1813
Fiction
>>>
```

Zipval

- Write a function called `zipval(lst, value)` that take a list and value as parameters and returns a list of tuples consisting of each element of the list and the value

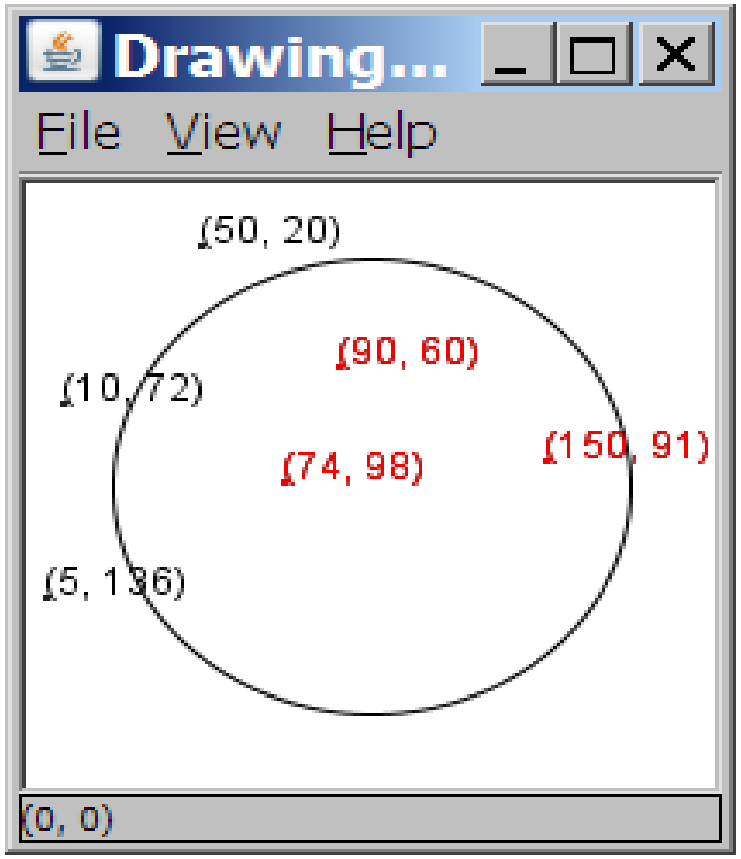
call	return
`zipval([10,20,30], "a")`	`[(10,'a'), (20,'a'), (30, 'a'])`

Zip

- Write a function called `zip(a, b)` that takes two lists as parameters and returns a list of tuples. Each tuple consists of the paired consecutive values of the parameter lists.

call	return
`zip([1,2,3],[4,5,6])`	`[(1,4), (2,5), (3, 6)]`

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
```

Earthquake plot

```
# Draws all of the cities affected by earthquakes
# given the radius input by the user.
from drawingpanel import *

def main():
    cities = get_cities()

    # gets information from the user
    epi_x = int(input("Epicenter x? "))
    epi_y = int(input("Epicenter y? "))
    radius = int(input("Radius? "))

    draw(cities, epi_x, epi_y, radius)
...
```

Earthquake plot – cont.

Let's write `get_cities()`.

First step is the pseudocode.

Open the file `cities.txt`

Read all the lines from the file

For each line of the file

    create a tuple of the city and the x and y coordinates

    add that tuple to a list

return the list

Earthquake plot – cont.

```
# Returns a list of tuples. Each tuple contains a city name, x and y
# coordinates from one line of cities.txt
def get_cities():
    file = open("cities.txt")
    lines = file.readlines()

    cities = []
    for line in lines:    # format: 'Tucson, 60, 90'
        parts = line.split()
        city = (parts[0], parts[1], parts[2])
        cities.append(city)
    return cities
```

Earthquake plot – cont.

```
# Draws all of the cities as dots on a DrawingPanel. If in
# the affected radius, colors them red, otherwise black.
# Draws a circle around the affected region.
def draw(cities, epi_x, epi_y, radius):
    p = DrawingPanel(400, 400)
    p.canvas.create_oval(epi_x - radius, epi_y - radius,
                         epi_x + radius, epi_y + radius)
    for city in cities:       # the variable city is a tuple
        x = int(city[1])      # get x-coordinate
        y = int(city[2])      # get y-coordinate
        color = "black"
        if(x >= epi_x - radius and x <= epi_x + radius and
           y >= epi_y - radius and y <= epi_y + radius):
            color = "red"
        p.canvas.create_oval(x, y, x + 4, y + 4, outline=color)
…
```

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("janUAry", 1, "January", 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
```