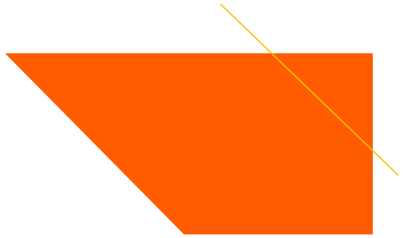
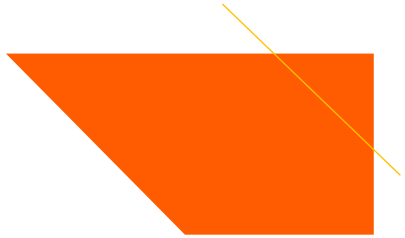
Introduction to Problem Solving in Python

COSI 10A



Class objectives

- Tuples (last subsection of 7.3)
- File Processing (6.1)

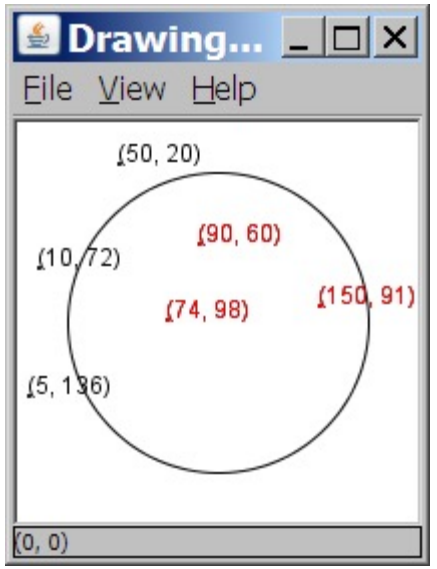


Tuples

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 window (`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

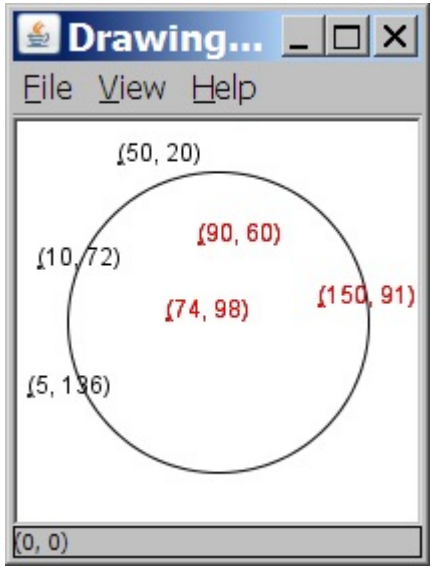
```
lines = open("cities.txt").readlines()
names = [0] * len(lines)
x_coords = [0] * len(lines)
y_coords = [0] * len(lines)

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 (immutable)**
- 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

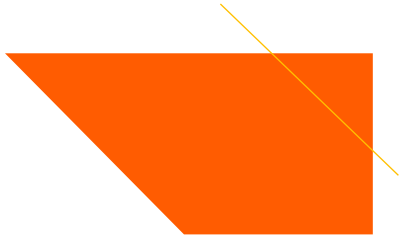
- You can access elements using [] notation, just like lists and strings

```
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

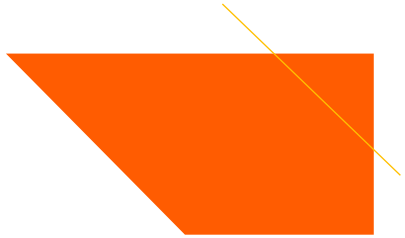
- Write a function called `days_till` that accepts a start month and day and a stop month and day and returns the number of days between them

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
```



File Processing

File input/output (I/O)

- A **file** is a collection of information that is stored on a computer and assigned a particular name
  - A file name ends with a suffix that indicates the kind of data it contains or the format which it has been stored
- Files can be classified into **text files** and **binary files** depending on the format that is used
  - Text files are stored using the `.txt` extension (`.py`, `.html`, etc. are text format)
  - Binary files are stored using an internal format that requires special software to process

File paths

- Files are grouped into directories (or folders)
- Directories are organized in a hierarchy, starting from a root directory at the top
  - Windows machines: the root directory is indicated with `C:\`.
  - Linux and Mac machines: the root directory is indicated with `/`.
- A file path is a description of a file's location on a computer, starting with a drive and including the path from the root directory to the directory where the file is stored

File paths

- **Absolute path**: specifies a drive or a top "/" folder

  `C:/Documents/smith/hw6/input/data.csv`

- **Relative path**: does not specify any top-level folder

  `names.dat`

  `input/kinglear.txt`

- Assumed to be relative to the current directory: `file = open("data/readme.txt")`
- If our program is in `H:/hw6`, `open` will look for `H:/hw6/data/readme.txt`

Reading a file

- To access a file from inside a Python program you need to use the function `open`

Syntax:

```
name = open("filename")
```

Open a file for reading

Roses are red,
Violets are blue.
All my base
Are belong to you.

poem.txt

```
file = open("poem.txt")
```

When this line of code is executed Python construct a special file object linked to the file poem.txt

- The file object has several functions and features that help you to view and manipulate the data in the file

Reading a file

Roses are red,
Violets are blue.
All my base
Are belong to you.

poem.txt

Example:

```
file = open("poem.txt") # open file

# read and returns the entire file as a string
filetext = file.read()

print(filetext)

# print the number of characters in the string
print(len(filetext))

file.close() # close file
```

Reading a file

- The function `close` tells Python that your program is done using the file
  - The program will still run if you don't call `close`
  - Forgetting to close files can lead to bugs or lost of data if you are writing to a file

- The `with` statement is used to open a file and later close it automatically

```
with open("poem.txt") as file: # open file
     filetext = file.read()
     print(filetext)
     print(len(filetext))
```

- Recommended way to read and write files

Methods of file object

Function name	Description
`file.close()`	Indicates that you are done reading/writing the file
`file.read()`	Reads and returns the entire file as a string
`file.readable()`	Returns True if the file can be read
`file.readline()`	Reads and returns the entire line as a string
`file.readlines()`	Reads and returns the entire file as a list of line strings
`file.write("text")`	Sends text to an output file
`file.writelines(lines)`	Sends a list of lines to an output file

Reading a file

Roses are red,
Violets are blue.
All my base
Are belong to you.

poem.txt

file.readlines()

```
['Roses are red,\n', 'Violets are blue.\n', 'All my base\n', 'Are belong to you.']
```

file.readline()

```
'Roses are red,\n'
```

Line-Based File Processing

- Line-based processing is the practice of processing input line by line

Syntax:

```
for line in file:
        statement
        statement
        …
        statement
```

Reading a file line by line

Example:

```
with open("poem.txt") as file:
        line_count = 0
        for line in file:
                print("next line:", line)
                line_count += 1
        print("Line count:", line_count)
```

```
next line: Roses are red,

next line: Violets are blue.

next line: All my base

next line: Are belong to you.
Line count: 4
```

Line-Based File Processing

- Line-based processing is the practice of processing input line by line

Syntax:

```
for line in file:
        statement
        statement
        …
        statement
```

Reading a file line by line

Example:

```
with open("poem.txt") as file:
        line_count = 0
        for line in file:
                print("next line:", line.rstrip())
                line_count += 1
        print("Line count:", line_count)
```

```
next line: Roses are red,
next line: Violets are blue.
next line: All my base
next line: Are belong to you.
Line count: 4
```

Reading a file

Roses are red,
Violets are blue.
All my base
Are belong to you.

poem.txt

```
Roses are red,\nViolets are blue.\nAll my base\nAre belong to you.
```

input cursor

Each iteration of the `for` loop causes the cursor to move forward (consuming input)

```
Roses are red,\nViolets are blue.\nAll my base\nAre belong to you.
```

input cursor

First iteration

```
Roses are red,\nViolets are blue.\nAll my base\nAre belong to you.
```

input cursor

When the input cursor reaches the end of the file, the `for` loop stops