Lecture 7 Objectives:

- Dictionaries
- Graphical user interface (GUI)
DICTIONARIES
Nonsequential Data Collections

- Lists allows us to store and retrieve items from sequential data collections.
- When we need to access an item of the collection, we call it using its index (position of the item) which is an integer.
- In Python, there are more flexible ways of storing data using key-value pair.
- These collections are called dictionary.
- Instead of using index→value as in lists, key→value pairs are used in dictionaries.
- Keys can be of any immutable type.
Nonsequential Data Collections

- A dictionary in Python can be created using key-value pairs with curly brackets `{}`
  `{<key>:<value>,<key>:<value>,...}`
- The function `dict` creates a new dictionary with no items.

```python
>>> x=dict()
```
Dictionaries

```python
>>> myPasswords={"sis":"pass1","bankA":"pass2","bankB":"pass3"}
```

```python
>>> myPasswords
{'sis': 'pass1', 'bankB': 'pass3', 'bankA': 'pass2'}
```

- Once the dictionary is created, we can use the keys to access the corresponding value

```python
>>> myPasswords["bankA"]
'pass2'
```
Dictionaries

• Dictionaries are **mutable** collections that implement mapping from keys to values
• Keys can be of any **immutable** type
  – strings
  – tuples
• Values can be of any type
• Dictionaries are very efficient and can store large amounts of items
• Python provides built-in functions for dictionaries
Dictionaries

• Dictionaries can be extended dynamically
  >>> myPasswords
  {'sis': 'pass1', 'bankB': 'pass3', 'bankA': 'pass2'}
  >>> myPasswords['bankC'] = 'pass3'
  >>> myPasswords
  {'sis': 'pass1', 'bankB': 'pass3', 'bankA': 'pass2', 'bankC': 'pass3'}

• There is no order (sequence) in the dictionary. Mapping is based on the key values
• The len function works on dictionaries; it returns the number of key-value pairs:
  >>> len(myPasswords)
  4
## Dictionaries

<table>
<thead>
<tr>
<th>Method</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;dict&gt;.has_key(&lt;key&gt;)</code></td>
<td>Returns true if dictionary contains the specified key, false if it does not</td>
</tr>
<tr>
<td><code>&lt;key&gt; in &lt;dict&gt;</code></td>
<td>Same as has_key function</td>
</tr>
<tr>
<td><code>&lt;dict&gt;.keys()</code></td>
<td>Returns a list of keys</td>
</tr>
<tr>
<td><code>&lt;dict&gt;.values()</code></td>
<td>Returns a list of the values</td>
</tr>
<tr>
<td><code>&lt;dict&gt;.items()</code></td>
<td>Returns a list of tuples (key, value) representing the key-value pairs</td>
</tr>
<tr>
<td><code>&lt;dict&gt;.get(&lt;key&gt;,&lt;default&gt;)</code></td>
<td>If dictionary has key returns its value; otherwise returns default</td>
</tr>
<tr>
<td>del <code>&lt;dict&gt;[&lt;key&gt;]</code></td>
<td>Deletes the specified entry</td>
</tr>
<tr>
<td><code>&lt;dict&gt;.clear()</code></td>
<td>Deletes all entries</td>
</tr>
</tbody>
</table>
Dictionaries

```python
>>> myPasswords
{'sis': 'pass1', 'bankB': 'pass3', 'bankA': 'pass2', 'bankC': 'pass3'}
>>> myPasswords.keys()
['sis', 'bankB', 'bankA', 'bankC']
>>> myPasswords.values()
['pass1', 'pass3', 'pass2', 'pass3']
>>> "BankA" in myPasswords
False
>>> "sis" in myPasswords
True
>>> del myPasswords["sis"]
>>> myPasswords
{'bankB': 'pass3', 'bankA': 'pass2', 'bankC': 'pass3'}
>>> myPasswords.get("bankA","nothing")
'pass2'
>>> myPasswords.get("bank","nothing")
'nothing'
```
Example

• Write a function `countWords(filename)` that takes a filename as argument, counts the occurrence of each word in that file and displays them.
def countWords(filename):
    try:
        fp=open(filename)
        fc=fp.read()
    except:
        print('File IO problem')
    return fp.close()

    #empty dictionary to store word counts
    word_count=dict()

    #split into words
    word_list=fc.split()
    for word in word_list:
        if word in word_count:
            word_count[word]=word_count[word]+1
        else:
            word_count[word]=1

    for key in word_count:
        print(key+'-->'+str(word_count[key]))
Dictionaries and Tuples

• Dictionaries have a method called items that returns a list of tuples, where each tuple is a key-value pair.
  >>> d = {'a':0, 'b':1, 'c':2}
  >>> t = d.items()
  >>> print t
  [('a', 0), ('c', 2), ('b', 1)]

• As you should expect from a dictionary, the items are in no particular order.

• Conversely, you can use a list of tuples to initialize a new dictionary:
  >>> t = [('a', 0), ('c', 2), ('b', 1)]
  >>> d = dict(t)
  >>> print d
  {'a': 0, 'c': 2, 'b': 1}
Dictionaries and Tuples

• Combining this feature with zip yields a concise way to create a dictionary:
  >>> d = dict(zip('abc', range(3)))
  >>> print d
  {'a': 0, 'c': 2, 'b': 1}
• The dictionary method update also takes a list of tuples and adds them, as key-value pairs, to an existing dictionary.
• Combining items, tuple assignment and for, you get the idiom for traversing the keys and values of a dictionary:
  for key, value in d.items():
    print(value, key)
  The output of this loop is:
  0 a
  2 c
  1 b
Dictionaries and Tuples

• It is common to use tuples as keys in dictionaries
• For example, a telephone directory might map from last-name, firstname pairs to telephone numbers.
• Assuming that we have defined last, first and number, we could write:
  directory[last,first] = number
• The expression in brackets is a tuple. We could use tuple assignment to traverse this dictionary.
  for last, first in directory:
    print (first, last, directory[last,first])
Example

- Write a Python function named `countPairs` that takes a numeric list and count the number of pair occurrences.
- Consider the list `[1,2,3,2,1,3,2,1,2]`
- In this list
  - `(1, 2) → 2`
  - `(3, 2) → 2`
  - `(1, 3) → 1`
  - `(2, 3) → 1`
  - `(2, 1) → 2`
Example

def countPairs(nlist):
    pair_counts = dict()

    for i in range(0, len(nlist)-1):
        if (nlist[i], nlist[i+1]) in pair_counts:
            pair_counts[nlist[i], nlist[i+1]] =
                pair_counts[nlist[i], nlist[i+1]] + 1
        else:
            pair_counts[nlist[i], nlist[i+1]] = 1

    for pair in pair_counts:
        print(pair, ' --> ', str(pair_counts[pair]))
GRAPHICAL USER INTERFACE (GUI)
Graphical User Interface (GUI) design

• Modern software uses GUI to interface users
• GUI includes
  – Windows, dialogs
  – UI controls (button, textbox etc)
  – Events (mouse, keyboard, time etc.)
• Frameworks are used for GUI design
  – Tkinter
  – wxWidgets
  – Qt
  – GTK+
  – ...

GUI Design

- Although there are many frameworks, their concepts are very similar
  - Standard controls (ie. Button, textbox, menu etc.) exist in all frameworks
  - Events associated with each control (ie. Mouse click, resize, content changed etc.) exist in all frameworks
  - (Nearly) all frameworks come with a layout design utility (ie. QtDesigner). These utilities make it easier to design the screen layouts.
  - Some of the frameworks are OS dependent (ie. .NET), works only with a specific OS.
  - Works as event-driven
GUI Design

• OS independent frameworks can work on any OS → more portable code
• Once the concept of GUI is grasped, learning different frameworks does not take too much time
• In this course, we are going to learn a very simple GUI framework (graphics.py by Zelle)
• This framework supplies thin wrapper classes to Tkinter framework which is quite complicated.
Event-based Programming

- Programming techniques we have seen in this class so far
  - Procedural
  - Object oriented
- In these techniques, commands are executed in a sequential manner. Functions are called whenever they are in the execution flow.
- Difference of procedural and object oriented techniques lies in how we arrange data and functions.
- When UI is employed, the code executes depending of events happening.
- Events may be coming from a user or the system. Eg. when the user clicks on a button, or when the duration of a timer ends an event is created.
Event Handler Functions

• A function (called event-handler) should be written to handle each possible event. When an event occurs, the corresponding handler function is executed by the system.

• This way of programming is called event-based programming

• When the program is started, it is initialized (show the dialogs etc.) and starts to wait for events

• Most of the programs with UI works like this.
Graphics Library

- Graphics library is a simple UI framework designed (by John Zelle) for teaching purpose
- It can be downloaded from http://mcsp.wartburg.edu/zelle/python/graphics.py
- A reference can also be downloaded from http://mcsp.wartburg.edu/zelle/python/graphics/graphics.pdf
Graphics Library

• Copy graphics.py into C:\Python32\Lib

• To use this library, it should be imported using >>> from graphics import *

• Then you can start using functions within this library

• There are two kinds of objects in the library.
  – The GraphWin class implements a window where drawing can be done,
  – GraphicsObjects are provided that can be drawn into a GraphWin.
from graphics import *

def main():
    win = GraphWin("My Circle",100,100)
    c=Circle(Point(50,50),10)
    c.draw(win)
    win.getMouse()
    win.close()

main()
Graphics Library

- Graphic library includes the following objects:
  - Point,
  - Line,
  - Circle,
  - Oval,
  - Rectangle,
  - Polygon,
  - Text,
  - Entry (for text-based input),
  - Image
GraphWin Object

- A GraphWin object represents a window on the screen where graphical images may be drawn. A program may define any number of GraphWins.
- A GraphWin understands the following methods:
  - GraphWin(title, width, height, autoflush) Constructs a new graphics window for drawing on the screen. The parameters are optional, the default title is “Graphics Window,” and the default size is 200 x 200. The autoflush parameter, if True causes the window to be immediately updated after every drawing operation. The default value is False, allowing operations to “batch up” for better efficiency.
  - plot(x, y, color) Draws the pixel at (x, y) in the window. Color is optional, black is the default. Note: pixel-level operations are very inefficient and this method should be avoided.
GraphWin Object

- A GraphWin understands the following methods:
  - `plotPixel(x, y, Color)` Draws the pixel at the “raw” position \((x, y)\) ignoring any coordinate transformations set up by `setCoords`. Note: pixel-level operations are very inefficient and this method should be avoided.
  - `setBackground(color)` Sets the window background to the given color. The initial background is gray.
  - `close()` Closes the on-screen window. Once a window is closed, further operations on the window will raise a `GraphicsError` exception.
GraphWin Object

- `isClosed()` Returns a Boolean indicating if the window has been closed either by an explicit call to close or a click on its close box.
- `getMouse()` Pauses for the user to click in the window and returns where the mouse was clicked as a `Point` object. Raises `GraphicsError` if the window is closed while `getMouse` is in progress.
- `setCoords(xll, yll, xur, yur)` Sets the coordinate system of the window. The lower left corner is `(xll, yll)` and the upper right corner is `(xur, yur)`. All subsequent drawing will be done with respect to the altered coordinate system (except for `plotPixel`).
- `update()` Causes any pending window operations to be performed. Normally, this will happen automatically during idle periods. Explicit `update()` calls may be useful for animations.
GraphWin Examples

>>> win=GraphWin("This title will appear on the top of the window",30,100)

>>> win.setBackground('red')

>>> p=win.getMouse()

>>> p.x
662

>>> p.y
9
GraphWin Examples

```python
>>> win=GraphWin("This title will appear on the top of the window",300,300)
>>> win.plotPixel(150,150,'red')
```
Graphic Objects

• The module provides the following classes of drawable objects:
  – Point,
  – Line,
  – Circle,
  – Oval,
  – Rectangle,
  – Polygon,
  – Text.

• All objects are initially created unfilled with a black outline. All graphics objects support the following generic set of methods:
  – setFill(color) Sets the interior of the object to the given color.
  – setOutline(color) Sets the outline of the object to the given color.
Graphic Objects

- `setWidth(pixels)` Sets the width of the outline of the object to this many pixels. (Does not work for `Point`.)
- `draw(aGraphWin)` Draws the object into the given `GraphWin`. An object may only be drawn in one window at a time.
- `undraw()` Undraws the object from a graphics window. Returns silently if object is not drawn.
- `move(dx,dy)` Moves the object `dx` units in the x direction and `dy` units in the y direction. If the object is currently drawn, its image is adjusted to the new position.
- `clone()` Returns a duplicate of the object. Clones are always created in an undrawn state. Other than that, they are identical to the cloned object.
Point Object

- Point(x,y) Constructs a point having the given coordinates.
- getX() Returns the x coordinate of a point.
- getY() Returns the y coordinate of a point.
Line Object

- Line(point1, point2) Constructs a line segment from point1 to point2.
- setArrow(string) Sets the arrowhead status of a line. Arrows may be drawn at either the first point, the last point, or both. Possible values of string are 'first', 'last', 'both', and 'none'. The default setting is 'none'.

>>> l1=Line(Point(10,10),Point(80,80))
>>> l1.draw(win)
Line Object

```python
>>> l1.setArrow('both')
```

```python
>>> l1.setOutline('blue')
```

```python
>>> l1.setWidth(5)
```
Circle Objects

• Circle(centerPoint, radius) Constructs a circle with given center point and radius.
• getCenter() Returns a clone of the center point of the circle.
• getRadius() Returns the radius of the circle.

>>> c1=Circle(Point(40,40),80)
>>> c1.draw(win)
>>> c1.setWidth(5)
Rectangle Object

- Rectangle(point1, point2) Constructs a rectangle having opposite corners at point1 and point2.
- getCenter() Returns a clone of the center point of the rectangle.

```python
>>> r1=Rectangle(Point(80,100),Point(150,150))
>>> r1.setWidth(5)
>>> r1.draw(win)
>>> r1.setOutline('red')
```
Oval Object

- Oval(point1, point2) Constructs an oval in the bounding box determined by point1 and point2.
- getCenter() Returns a clone of the point at the center of the oval.

```python
>>> o1=Oval(Point(80,100),Point(150,150))
>>> o1.setOutline('green')
>>> o1.draw(win)
>>> o1.setWidth(8)
```
Text Object

- Text(anchPoint, string) Constructs a text object that displays the given string centered at anchPoint. The text is displayed horizontally.
- setText(string) Sets the text of the object to string.
- getText() Returns the current string.
- getAnchor() Returns a clone of the anchor point.
- setFace(family) Changes the font face to the given family. Possible values are: ’helvetica’, ’courier’, ’times roman’, and ’arial’.
- setSize(point) Changes the font size to the given point size. Sizes from 5 to 36 points are legal.
- setStyle(style) Changes font to the given style. Possible values are ’normal’, ’bold’, ’italic’, and ’bold italic’.
- setTextColor(color) Sets the color of the text to color. Note: setFill has the same effect.
Text Example

```python
>>> t1=Text(Point(100,100),'String on screen')
>>> t1.setTextColor('red')
>>> t1.setStyle('italic')
>>> t1.draw(win)
```
### Pixmap Object

- Simple image manipulation is done through the Pixmap class. A Pixmap object allows pixel-level access to an image. Pixmaps allow for saving to a file and may be displayed using an Image object.
- `Pixmap(filename)` Constructs a Pixmap from the image file, filename.
- `Pixmap(width, height)` Constructs a Pixmap of the given height and width. Initially, all pixels will be transparent.
- `getWidth()` Returns the width of the image in pixels.
- `getHeight()` Returns the height of the image in pixels.
- `getPixel(x,y)` Returns a triple (r,g,b) of the red, green, and blue intensities of the pixel at (x,y). Intensity values are in range(256).
- `setPixel(x,y,color)` Color is a triple (r,g,b) representing a color for the pixel. Sets pixel at (x,y) to the given color.
- `save(filename)` Saves the image in a file having the given name. The format for the file is determined by the extension on the filename (e.g. .ppm or .gif).
- `clone()` Returns a copy of the Pixmap.
Images

- The graphics module also provides minimal support for displaying certain image formats into a GraphWin.
- Most platforms will support bitmap, PPM, and GIF images. Display is done with an Image object.
- Images support the generic methods
  - `move(dx,dy)`,
  - `draw(graphwin)`,
  - `undraw()`, and `clone()`.
- Image specific methods are:
  - `Image(centerPoint, image)` image is either the name of an image file, or a Pixmap object. Constructs an image from contents of the given file or pixmap, centered at the given center point.
Reversi with GUI

• Write reversi or othello on a 8x8 board
• For game rules see
  http://en.wikipedia.org/wiki/Reversi

• Game will be played between 2 players, NOT against computer
• For a simple GUI see the simpleGUI.py