

# Lecture 23

**Topics:** *Chapter 11. Data Collections*

**11.4** Designing with lists and classes

**11.7** Non-sequential collections

## 11.4 Designing with lists and classes

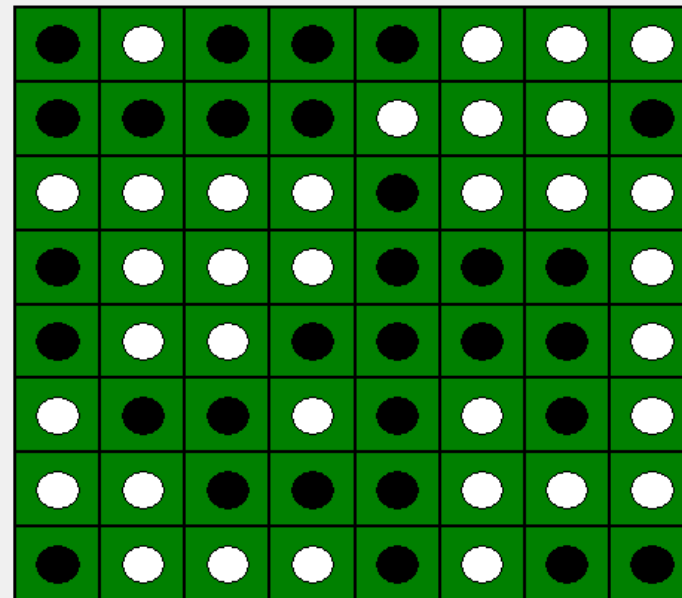
Lists and classes taken together give us powerful tools for structuring the data in our programs.

## 11.4 Designing with lists and classes

### 8×8 board with bi-color disks

Let's write a program that will display an 8 × 8 board of disks that are randomly colored into black and white. Upon a click on a disk, it will change the color of the disk to opposite.

This is a first draft of the **Othello** (**Reversi**) game.



EXIT

## 11.4 Designing with lists and classes

Think of two classes:

### Board

```
self.window  
self.board = []  
self.color = []
```

```
__init__(self,window)  
changeDisk(self,point)
```

**self.board**

*is a list of disks (circles)*

**self.color**

*is a corresponding list of  
disk colors*

### Button

```
self.window  
...
```

```
__init__(self,window,point,width,height,  
message)  
draw(self)  
undraw(self)  
move(self,dx,dy)  
setText(self,newMessage)
```

run **othello.py**

## 11.7 Non-sequential collections

Python provides a number of built-in data types for collections.

After lists, a collection type *dictionary* is probably the most widely used.

Lists allow us to store and retrieve items from *sequential collections* (recall indexing).

Nevertheless sometimes we need to retrieve the information by a *key* (say student's ID or a person's SSN). A record (*value*) is accessed by the key.

In programming terminology it is called a *key-value* pair.

## 11.7 Non-sequential collections

A collection that allows us to look up information associated with arbitrary key is called *mapping*.

Python *dictionaries* are mappings.

Some other programming languages provide similar structures called *hashes* or *associative arrays*.

## 11.7 Non-sequential collections

### Python dictionaries

**Examples** (Python interpreter):

```
>>> records = {123: ["Kevin", 24],  
234: ["Andrew", 78], 756: ["Janine", 56]}
```


```
>>> records  
{234: ['Andrew', 78], 123: ['Kevin', 24], 756:  
['Janine', 56]}
```

## 11.7 Non-sequential collections

### Python dictionaries

**Examples** (Python interpreter):

```
>>> records = {123: ["Kevin", 24],  
234: ["Andrew", 78], 756: ["Janine", 56]}
```



```
>>> records  
{234: ['Andrew', 78], 123: ['Kevin', 24], 756:  
['Janine', 56]}
```



## 11.7 Non-sequential collections

### Python dictionaries

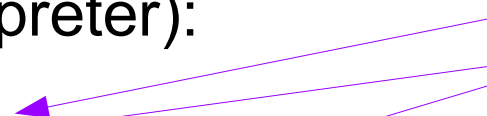
**Examples** (Python interpreter):

```
>>> records = {123: ["Kevin", 24],  
234: ["Andrew", 78], 756: ["Janine", 56]}
```

```
>>> records
```

```
{234: ['Andrew', 78], 123: ['Kevin', 24], 756:  
['Janine', 56]}
```

*keys (not mutable, i.e.  
cannot be modified!)*



## 11.7 Non-sequential collections

### Python dictionaries

**Examples** (Python interpreter):

```
>>> records = {123: ["Kevin", 24],  
234: ["Andrew", 78], 756: ["Janine", 56]}  
>>> records  
{234: ['Andrew', 78], 123: ['Kevin', 24], 756:  
['Janine', 56]}
```

*values*

*(mutable, i.e. can be modified)*

## 11.7 Non-sequential collections

### Python dictionaries

**Examples** (Python interpreter):

```
>>> records = {123: ["Kevin", 24],  
234: ["Andrew", 78], 756: ["Janine", 56]}
```

```
>>> records  
{234: ['Andrew', 78], 123: ['Kevin', 24], 756:  
['Janine', 56]}
```

```
>>> records[123]  
['Kevin', 24]
```

```
>>> records[236]
```

```
Traceback (most recent call last):  
  File "<pyshell#4>", line 1, in <module>  
    records[236]  
KeyError: 236
```

## 11.7 Non-sequential collections

### Python dictionaries

**Examples** (Python interpreter):

```
>>> records = {123: ["Kevin", 24],  
234: ["Andrew", 78], 756: ["Janine", 56]}
```

```
>>> records  
{234: ['Andrew', 78], 123: ['Kevin', 24], 756:  
['Janine', 56]}
```

```
>>> records[456]=["Alba", 27]
```

```
>>> records  
{456: ['Alba', 27], 234: ['Andrew', 78], 123:  
['Kevin', 24], 756: ['Janine', 56]}
```

## 11.7 Non-sequential collections

method	meaning
<code>&lt;key&gt; in &lt;dict&gt;</code>	returns True if dictionary contains the key, and false otherwise
<code>&lt;dict&gt;[key]=value</code>	adds tuple <code>&lt;key&gt;:&lt;value&gt;</code> to the dictionary
<code>&lt;dict&gt;.keys()</code>	returns a sequence of keys
<code>&lt;dict&gt;.values()</code>	returns a sequence of values
<code>&lt;dict&gt;.items()</code>	returns a sequence of tuples (key,value)
<code>&lt;dict&gt;.get(&lt;key&gt;,&lt;default&gt;)</code>	if dictionary has key returns its value; otherwise returns default
<code>del &lt;dict&gt;[&lt;key&gt;]</code>	deletes the specified by the key entry
<code>&lt;dict&gt;.clear()</code>	deletes all entries
<code>for &lt;var&gt; in &lt;dict&gt;</code>	loops over the keys

## 11.7 Non-sequential collections

### Python dictionaries

#### Example:

Let's write a program that will read students' records from a file and print a list of their names. Students have ids. A dictionary will be used to store students records by their id number.

```
184758 Adams, Samantha 56 222.32
365853 Cole, Amanda 100 390
634649 Jack, Adam 140 490
747284 Katz, Mery 28 86.8
104755 Zenith, Kevin 135 459
```

see [studentsRecordsAsDictionary.py](#)