Objectives

At the end of this unit you should be able to

• Store, access and modify data in a list
• Identify and explain the difference between an alias and a copy of a list
• Explain the difference between mutable and immutable objects
• Read, trace, summarize and write Python code which manipulates data in lists
• List and recreate some common design patterns for manipulating data in lists
More Python Built-In Types

• In addition to simple types we have seen, Python has data types whose values are *collections* of other values instead of a single value (like the files):
  • **Lists** and a **tuples** are ordered sequences of data
  • **Sets** are collections of data without any ordering
  • A **dictionary** provides a mapping from keys to data
    – i.e. from student number to student name
  • A file is also an ordered sequence of data, but files
    – exist independently of the program
    – as we have seen, the protocol to access file data is slightly more complex (e.g. open, close, readline, write, etc)
  • We now focus on lists
    – Dictionaries, Sets and Tuples will follow
Why We Need Lists

• Suppose we set an experiment to count the number of salmon that cross a point in the Fraser river between 3:00pm and 4:00pm for 10 days and got:

  35, 40, 22, 36, 20, 25, 35, 40, 40, 32

• How can we store this result?
  – use 10 variables? .... what if we continue counting for another month ... year?

• How can I keep a shopping list with my groceries I need to buy?
  – I'd like to start with: bread, milk, tomatoes, cheese, caviar
  – and keep removing or adding as I go through my freezer

• The best way to do these is to use lists.
Lists

• Are created by comma separated entries within "[ ]"

\[
\text{salmon} = [35, 40, 22, 36, 20, 25, 35, 40, 40, 32]
\text{groceries} = ['bread', 'milk', 'cheese', 'tomatoes', 'caviar']
\]

• A list can contain data of multiple types:

\[
\text{new_list} = [ 'a', 5, "CPSC 301", 1.414, [ 42, 54 ] ]
\]

• Each item in a list has a position or an index
  – positions go from 0 to \text{len(list)} – 1
  – for instance:

\[
\text{salmon} \rightarrow [35, 40, 22, 36, 20, 25, 35, 40, 40, 32]
\text{index:} \quad 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9
\]

• A list is one object but contains references to other objects. A better memory model for the salmon list would be:

\[
\begin{array}{cccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
35 & 40 & 22 & 36 & 20 & 25 & 35 & 40 & 40 & 32 \\
\end{array}
\]
Lists as Sequences

• Lists are sequences (like a string)
• To access an item we use: `list_name[index]`
  For instance, if `salmon` ➔ `[35, 40, 22, 36, 20, 25, 35, 40, 40, 32]`
  – `salmon[0]` ➔ `35`,  `salmon[9]` ➔ `32`,
  – `groceries[1]` ➔ "milk"

• Indexes are in the range from 0 to `len(list)-1`
  – `salmon[10]` produces an "index out of range" error

• Negative indices may be used to index backwards from the end of the list
  – `salmon[-1]` ➔ `32`  -1 represents the index of the last element
  – `salmon[-2]` ➔ `40`  - 2\textsuperscript{nd} from the end
  – `salmon[-10]` ➔ `35`
Iterating over a List

- We can iterate over the elements of a list using a for-loop or using the index.
  I.e. to print the day and the salmon count for that day we can do:

  ```python
day = 1
for count in salmon:
    print("day", day, ":", count)
    day += 1
```

  or

  ```python
for i in range(len(salmon)):
    print("day", i+1, ":", salmon[i])
```

- The first method is better and easier to understand.
Creating and Modifying Lists

• A list is created either from an existing sequence (like a string) using
  \textbf{list(sequence)}
or it starts as an empty list and is built by adding an element at a time.

• An empty list is defined as \texttt{[]} (like the empty string \texttt{""})
  \texttt{my\_list = []} creates an empty list ( \texttt{len(my\_list)} is 0 )

• Can use \textbf{append(object)} to add a new element at the end of the list:
  \texttt{my\_list.append(30)} then \texttt{my\_list \rightarrow [30]}
  \texttt{my\_list.append(40)} then \texttt{my\_list \rightarrow [30, 40]}

• Can use \textbf{insert(index, object)} to insert a new element at a specific index (location) in the list:
  \texttt{my\_list.insert(1, 50)} then \texttt{my\_list \rightarrow [30, 50, 40]}
  \texttt{my\_list.insert(0, 60)} then \texttt{my\_list \rightarrow [60, 30, 50, 40]}
Creating and Modifying Lists (cont'd)

• Can change the element at a given index:
  
  if my_list → [60, 30, 50, 40]
  
  my_list[2] = 10    =>    my_list → [60, 30, 10, 40]

• Can remove an element from a list:
  
  my_list.remove(30)  =>  my_list → [60, 10, 40]

• More methods that can be applied to a list:
  
  – L.pop()  ---  removes the last element from the list L
  – L.reverse()  ---  reverses the order of the elements in the list L
    (last becomes first, ...., first becomes last)
  – L.sort()  ---  sorts the elements in the list from smaller to larger
  – L.count(v)  ---  counts the occurrences of v in list L
  – L.index(v)  ---  returns the index of the first occurrence of v in list L

• Try help(list) to see all methods of list. Most methods change the list and return None.

• These operations show that lists are mutable objects unlike strings which are immutable
Python has a number of built-in functions on lists (which are not methods of list and don't change their argument list):

- `len(list)` - returns the length of the list
- `max(list)` - returns the maximum value in the list
- `min(list)` - returns the minimum value in the list
- `sum(list)` - returns the sum of the values in the list
- `sorted(list)` – returns a sorted copy of the list

Example:
```python
if salmon \[35, 40, 22, 36, 20, 25, 35, 40, 40, 32\]

to calculate the average count we can write:
```average = sum(salmon)/len(salmon)``` 

- We can concatenate two lists by using the + operator:
  ```python
dairy = [ "milk", "cheese"]
veggies = ["tomatoes", "broccoli"]
groceries = dairy + veggies
```
Slicing

• Given a list, we can extract an entire sublist, called a **slice** of the list, using
  \[
  \text{list}[i:j]
  \]
  which returns a new list containing the elements from the old list that are in positions \(i, i+1, \ldots, j-1\)

• For instance:
  \[
  \text{groceries} = [\text{'bread'}, \text{'milk'}, \text{'cheese'}, \text{'tomatoes'}]
  \text{dairy} = \text{groceries}[1:3] \quad \text{then} \quad \text{dairy} \rightarrow [\text{'milk'}, \text{'cheese'}]
  \]

• \text{list}[:j] is the same as \text{list}[0:j]
  \text{list}[i:] is the same as \text{list}[i:len(list)]
  \text{list}[:] creates a new copy of \text{list}
  \text{list}[:\text{:-1}] creates a new copy of the original list without the last element