Objectives

- A new data type: Lists

Administrivia

- My office hours: today, 2-3:30; Thurs, 2:30-4:30
- Kidus's evening office hours: tonight, 6-9 p.m.

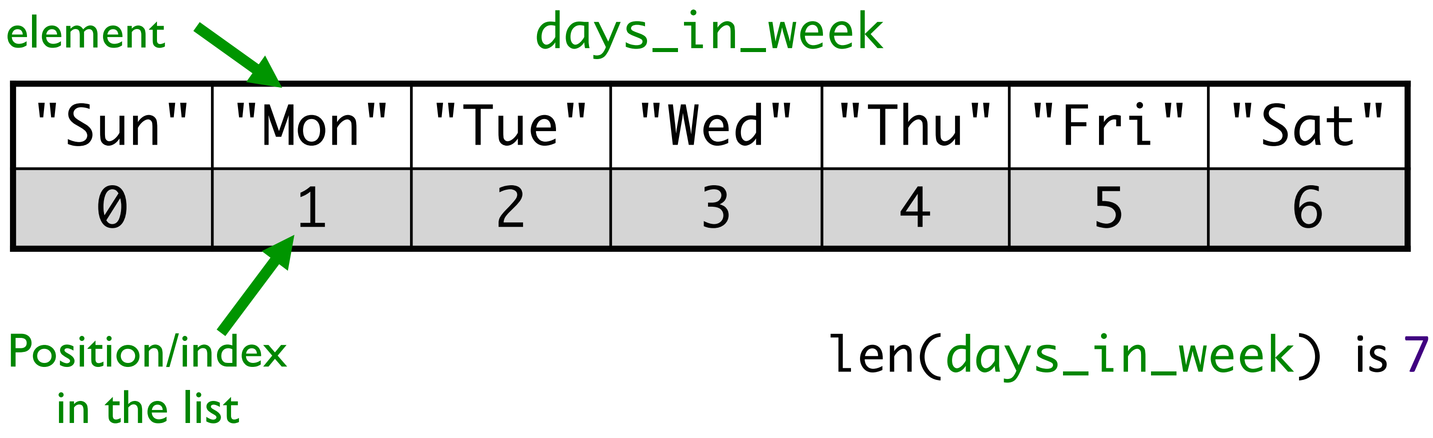
Lab 7 Retrospective

- Things we learned in the past keep coming back!
  - Combining with the new things!
  - That's the power of computing/programming!
- You probably noticed: more thinking than coding!
  - (just as I predicted!)
- Labs are the learning space—may be dark and stormy but you are learning!
  - Look at your Lab 3 programs—they likely seemed difficult at the time

Sequences of Data

- Data types model various information
  - Numbers, strings, rectangles, …
- Sequences so far …
  - `str`: sequence of characters
  - `range`: generator (sequence of numbers)
- We commonly group a sequence of data together and refer to them by one name
  - Days of the week: Sunday, Monday, Tuesday, …
  - Months of the year: Jan, Feb, Mar, …
  - Shopping list
- Can represent this data as a `list` in Python
  - Similar to **arrays** in other languages

Lists: A *Sequence* of Data Elements


element
days_in_week
"Sun" "Mon" "Tue" "Wed" "Thu" "Fri" "Sat"
0 1 2 3 4 5 6
Position/index in the list

`len(days_in_week)` is 7

- Elements in lists can be *any* data type

What does the structure look like similar to?

Example Lists in Python: [ ]

- Empty List: `[]`
- List of `str`s:
  - `days_in_week = ["Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"]`
- List of `floats`
  - `high_temperatures = [60.4, 70.2, 63.8, 55.7,  54.2]`
- Lists can contain >1 type
  - `wheel_of_fortune = [250, 1000, "Bankrupt", "Free Play"]`

Benefits of Lists

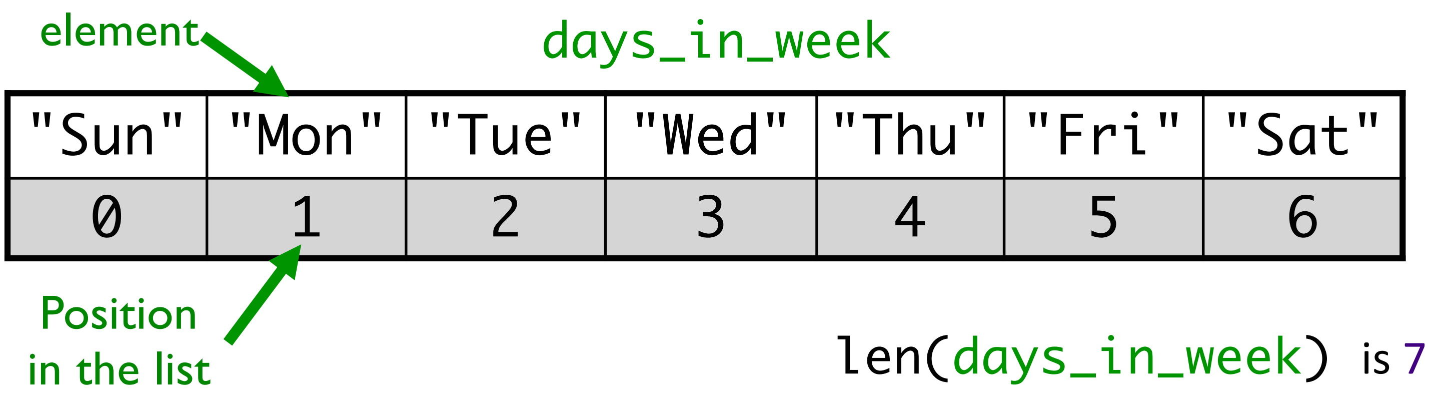
- Group related items together
  - Instead of creating separate variables
    - `sunday  =  "Sun"`
    - `monday  =  "Mon"`
- Convenient for dealing with large amounts of data
  - Example: could keep all the temperature data in a list if needed to reuse later
- Functions and methods for handling, manipulating lists

List Operations

Similar to operations for strings

Concatenation	`<seq> + <seq>`
Repetition	`<seq> * <int-expr>`
Indexing	`<seq>[<int-expr>]`
Length	`len(<seq>)`
Slicing	`<seq>[:]`
Iteration	`for <var> in <seq>:`
Membership	`<expr> in <seq>`

Lists: A Sequence of Data Elements



"Sun"	"Mon"	"Tue"	"Wed"	"Thu"	"Fri"	"Sat"
0	1	2	3	4	5	6

`len(days_in_week)` is 7

- `<listname>[<int_expr>]`
  - Similar to accessing characters in a string
  - `days_in_week[-1]` is "Sat"
  - `days_in_week[0]` is "Sun"

Iterating through a List

- Read as
  - For every element in the list …

An item in the list

list object

```
for item in list:
      print(item)
```

Iterates through *items* in list

- Output equivalent to

```
for x in range(len(list)):
      print(list[x])
```

Iterates through *positions* in list

`daysOfWeek.py`

Example Code

```python
friends = ["Alice", "Bjorn", "Casey", "Duane", "Elsa", "Farrah"]

for name in friends:
    print("I know " + name + "!")
    print(name, "is a friend of mine.")

print("Those are the people I know.")
```

`friends.py`

Example Code

```python
friends = ["Alice", "Bjorn", "Casey", "Duane", "Elsa", "Farrah"]

for name in friends:
    print("I know " + name + ".")
    print(name, "is a friend of mine.")

print("Those are the people I know.")
```

Practice on your own: Rewrite as an “iterate over positions in list” loop

`friends.py`

Complete Old MacDonald

```python
animals = ["cow", "pig", "duck"]
sounds = ["moo", "oink", "quack"]

for i in range(len(animals)):



    print_verse(
```

Doc String (as seen using help function):

```
print_verse(animal, sound)
    Prints a verse of Old MacDonald, plugging in the animal
    and sound parameters (which are strings), as appropriate.
```

`oldmac.py`

Practice

- Get a *list* of weekdays and a *list* of weekend days from the days of the week list

```
days_in_week=["Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"]

weekdays =

weekend_days =
```

Practice

- Get a *list* of weekdays
  - `days_in_week=["Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"]`
  - `weekDays = days_in_week[1:6]`

Explain each solution

- Get the *list* of weekend days from the days of the week list
  - `days_in_week=["Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"]`
  - `weekend = days_in_week[:1] + days_in_week[-1:]`

  or

  - `weekend = [days_in_week[0]] + [days_in_week[-1]]`

Practice

- Get the *list* of weekend days from the days of the week list
  - `days_in_week=["Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"]`
  - `weekend = days_in_week[:1] + days_in_week[-1:]`

Evaluates to a `list`

or

  - `weekend = [days_in_week[0]] + [days_in_week[-1]]`

Evaluates to an *element* of the list, which is a `str`

Membership

- ***Check if a list contains an element***
- Example usage
  - `enrolled_students` is a list of students who are enrolled in the class
  - Want to check if a student who attends the class is enrolled in the class

```
if student not in enrolled_students:
    print(student, "is not enrolled")
```

Making Lists of Integers Quickly

- If you want to make a list of integers that are evenly spaced, you can use the `range` generator
- Example: to make a list of the even numbers from 0 to 99:
  - `evenNumList = list(range(0, 99, 2))`

Converts the generated numbers into a list

`str` Method Flashback

- `string.split([sep])`
  - Returns a ***list*** of the words in the string `string`, using `sep` as the delimiter string
  - If `sep` is not specified or is `None`, any *whitespace* (space, new line, tab, etc.) is a separator

`str` Method Flashback

- `string.split([sep])`
  - Returns a ***list*** of the words in the string `string`, using `sep` as the delimiter string
  - If `sep` is not specified or is None, any *whitespace* (space, new line, tab, etc.) is a separator
  - Example:

```
phrase = "Hello, Computational Thinkers!"
x = phrase.split()
```

What is x? What is its data type? What does x contain?

`str` Method Flashback

- `string.join(iterable)`
    - Return a string which is the concatenation of the *strings* in the **`iterable`**/sequence. The separator between elements is `string`.

`str` Method Flashback

- `string.join(iterable)`
  - Return a string which is the concatenation of the *strings* in the **`iterable`**/sequence. The separator between elements is `string`.
  - Example:

```
x = ["1","2","3"]
phrase = " ".join(x)
```

What is `x`'s data type?
What is `phrase`'s data type?
What does `phrase` contain?

List Methods

Method Name	Functionality
`<list>.append(x)`	Add element *x* to the end
`<list>.sort()`	Sort the list
`<list>.reverse()`	Reverse the list
`<list>.index(x)`	Returns the index of the first occurrence of *x*, Error if *x* is not in the list
`<list>.insert(i, x)`	Insert *x* into list at index *i*
`<list>.count(x)`	Returns the number of occurrences of *x* in list
`<list>.remove(x)`	Deletes the first occurrence of *x* in list
`<list>.pop(i)`	Deletes the *i* th element of the list and returns its value

Note: methods do ***not return*** a ***copy*** of the list …

Strings vs. Lists

- Strings are **immutable**
  - Can't be mutated?
  - Err, can't be modified/changed
- Lists are **mutable**
  - Can be changed
    - Called "change in place"
  - Affects how we call/use methods

```
groceryList=["milk", "eggs", "bread", "Doritos", "OJ", "sugar"]

groceryList[0] = "skim milk"
groceryList[3] = "popcorn"

groceryList is now ["skim milk", "eggs", "bread", "popcorn", "OJ", "sugar"]
```

Practice in Interactive Mode

- `myList = [7,8,9]`
- `myString = "abc"`
- `myList[1]`
- `myString[1]`
- `myString.upper()`
- `myList.reverse()`
- `myString`
- `myList`
- `myString = myString.upper()`
- `myList = myList.reverse()`
- `myString`
- `myList`

Special Value: None

(Similar to `null` in Java)

- Special value we can use
  - E.g., Return value from function/method when there is an error
  - Or if function/method does not return anything
- If you execute

```
myList = myList.sort()
print(myList)
```

  - Prints None because `myList.sort()` does **not** *return* anything

What should we write instead?

Strings vs. Lists

Strings

- Methods that are meant to change a string return a *changed copy* of the String
  - The string the method was called on does *not* change
- Consequence: Call the modifying method and assign that to a variable
- Example use:
  - `upper = mystr.upper()`

Lists

- Methods that are meant to change a list change the list *in place*
  - Don’t return anything
- Consequence: Call the modifying method but don’t assign it to a variable
- Example use:
  - `mylist.sort()`

Returning to the Fibonacci Sequence

- Goal: Solve using list
- $F_0=0$, $F_1=1$
- $F_n=F_{n-1} + F_{n-2}$
- Example sequence: 1, 1, 2, 3, 5, 8, 13, 21, …

Fibonacci Sequence

- Create a list of the 1st 20 Fibonacci numbers
  - $F_0=0$; $F_1=1$; $F_n=F_{n-1}+F_{n-2}$

Grow list as we go

```
fibs = []                   # create an empty list
fibs.append(0)              # append the first two Fib numbers
fibs.append(1)
```

fibs.py

Fibonacci Sequence

- Create a list of the 1st 20 Fibonacci numbers
  - $F_0=0$; $F_1=1$; $F_n=F_{n-1}+F_{n-2}$

Grow list as we go

```
fibs = []                # create an empty list
fibs.append(0)           # append the first two Fib numbers
fibs.append(1)
for x in range(2, 20): # compute the next 18 numbers
    newfib = fibs[x-1] + fibs[x-2]
    fibs.append(newfib) # add next number to the list

print(fibs)   # print out the list as a list in one line
```

fibs.py

Fibonacci Sequence

- Create a list of the 1st 20 Fibonacci numbers
  - $F_0=0$; $F_1=1$; $F_n=F_{n-1}+F_{n-2}$

```
fibs = []                  # create an empty list
fibs.append(0)             # append the first two Fib numbers
fibs.append(1)
for x in range(2, 20): # compute the next 18 numbers
    newfib = fibs[-1] + fibs[-2]
    fibs.append(newfib) # add next number to the list

print(fibs)   # print out the list as a list in one line
```

Alternative

fibs.py

Lists vs. Arrays

- Briefly, lists are similar to arrays in other languages
  - More similar to *Vectors* in C++ and *ArrayLists* in Java
- However, arrays have ***fixed*** lengths
  - Can't insert and remove elements from arrays to change the length of the array
  - Need to make the array as big as you'll think you'll need

Fibonacci Sequence: Array-like Implementation

- Create a list of the 1st 20 Fibonacci numbers
  - $F_0=F_1=1$; $F_n=F_{n-1}+F_{n-2}$

- Create whole list
- Update values

```
fibs = [0]*20      # creates a list of size 20,
                         # containing all 0s

fibs[0] = 0
fibs[1] = 1
```

fibs2.py

Fibonacci Sequence: Array-like implementation

- Create a list of the 1st 20 Fibonacci numbers
  - $F_0=F_1=1; \; F_n=F_{n-1}+F_{n-2}$

- Create whole list
- Update values

```
fibs = [0]*20     # creates a list of size 20,
                        # containing all 0s

fibs[0] = 0
fibs[1] = 1

for x in range(2, len(fibs)):
    newfib = fibs[x-1] + fibs[x-2]
    fibs[x] = newfib

for num in fibs:   # print each num in list on sep lines
    print(num)
```

fibs2.py

Looking Ahead

- Lab 7 – due Friday
- Broader Issue: Cryptography – due Thursday night