Objectives

- A new data type: Lists

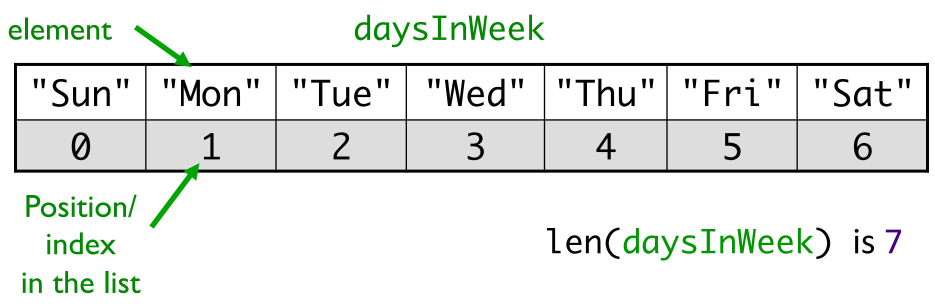
Review

- How can we convert between characters and their numerical representation?
  - How can we convert from the numerical representation to the character?
- What are the various things we can do with strings?

Sequences of Data

- 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



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

`len(daysInWeek)` is 7

- Elements in lists can be *any* data type

What does does this look similar to, in structure?

Example Lists in Python

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

Syntax for list: []
How different from accessing a character in a string?

Benefits of Lists

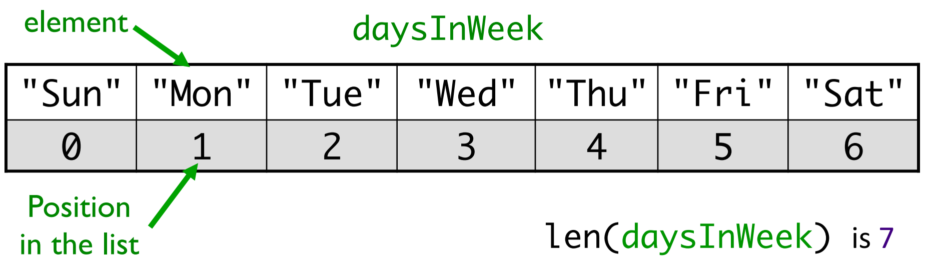
- Group related items together
  - Instead of creating separate variables
    - `sunday = "Sun"`
    - `monday = "Mon"`
- Convenient for dealing with large amounts of data
  - Example: could keep all the temperature data in a list if needed to reuse later
- Functions and methods for handling, manipulating 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(daysInWeek)` is 7

- `<listname>[<int_expr>]`
  - Similar to accessing characters in a string
  - `daysInWeek[-1]` is “Sat”
  - `daysInWeek[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

```
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

Practice

- Get the *list* of weekend days from the days of the week list
  - ➢ `daysInWeek=["Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"]`

Practice

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

    Gives back a `list`

  or
  - ➢ `weekend = [daysInWeek[0]] + [daysInWeek[-1]]`

    Gives back an element of list, which is a `str`

Membership

- ***Check if a list contains an element***
- Example usage
  - `enrolledstudents` 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 enrolledstudents:
     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
  - Example:

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

What is x? 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.
  - 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 …

Lists vs. Strings

- Strings are **immutable**
  - Can't be mutated?
  - Err, can't be modified/ changed
- Lists are **mutable**
  - Can be changed
    - Called "change in place"
  - Changes 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

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

Looking Ahead

- Pre lab for Lab 7 due tomorrow before lab
  - Think about the Caesar Cipher implementation
- Lab 7 pairs
- Broader Issue: Cryptography