Objectives

- A new data type: Lists

1

Lab 7 Retrospective

- Things we learned in the past keep coming back!
  - ➢ Combining with the new things!

- That's the power of computing/programming!

2

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
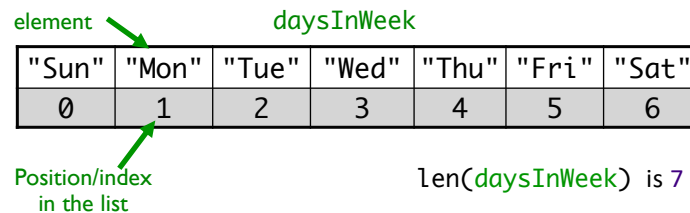
March 8, 2023   Sprenkle - CSCI111   4

4

Lists: A *Sequence* of Data Elements

element   `daysInWeek`

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

Position/index in the list   `len(daysInWeek)` is 7

- Elements in lists can be *any* data type

What does this look similar to, in structure?

March 8, 2023   Sprenkle - CSCI111   5

5

2

Example Lists in Python

- Empty List: []

- List of `strs`:
  - ➢ daysInWeek=["Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"]

- List of `floats`
  - ➢ 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?

March 8, 2023

6

6

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

March 8, 2023   Sprenkle - CSCI111   7

7

3

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>`

8

Lists: A Sequence of Data Elements

element   *daysInWeek*

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

Position in the list

`len(daysInWeek)` is 7

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

9

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

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.")
```

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.")
```

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

12

Complete Old MacDonald

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

for i in range(len(animals)):



    printVerse(
```

Doc String (as seen using help function):

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

13

6

Practice

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

Practice

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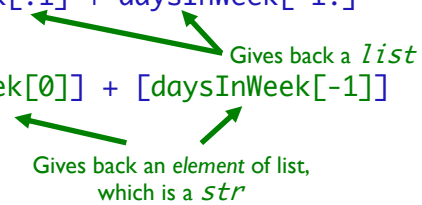 - ➢ `weekDays = daysInWeek[1:6]`

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:]`

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

  Gives back a *list*

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

16

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")
```

17

8

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

18

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?

19

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"]
```

22

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`

23

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

```
list = list.sort()
print(list)
```

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

What should we code instead?

24

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, …

25

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)
```

26

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
```

27

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]   Alternative
    fibs.append(newfib)  # add next number to the list

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

Lists vs. Arrays

- Briefly, lists are similar to arrays in other languages
  - More similar to *Vectors* in C++ and *ArrayLists* in Java
- Typically, arrays have **fixed** lengths
  - Can't insert and remove elements from arrays so that the length of the array changes
  - 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
```

30

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)
```

31

15

Looking Ahead

- Lab 7 – due Friday