Objectives

- A new data type: Lists

Review

- The data type of the loop variable depends on what's after `in`

```
string = “some string”

for x in range(len(string)):
     # loop body …

for x in string:
     # loop body …
```

What is the data type of the loop variable **x**?

Review

- The data type of the loop variable depends on what's after `in`

```
string = “some string”

for x in range(len(string)):
     # loop body …

for x in string:
     # loop body …
```

Integer

String

Review

- 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

element

`daysInWeek`

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

Position in the list

`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

- 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”]

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

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”

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

- Equivalent to

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

Iterates through *positions* in list

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 problem
  - **`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”)
```

Problem: If have a list **`attendingstudents`**, check if each attending student is an enrolled student

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 …

Fibonacci Sequence

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

Fibonacci Sequence

- Create a list of the 1st 15 Fibonacci numbers
  - $F_0=F_1=1;\ F_n=F_{n-1}+F_{n-2}$

Grow list as we go

```
fibs = []          # create an empty list
fibs.append(1)     # append the first two Fib numbers
fibs.append(1)
for x in range(2, 16): # compute the next 13 nums
      newfib = fibs[x-1]+fibs[x-2]
      fibs.append(newfib)

print(fibs)        # print out the list
```

Fibonacci Sequence

- Create a list of the 1st 15 Fibonacci numbers
  - $F_0=F_1=1;\ F_n=F_{n-1}+F_{n-2}$

- Create list
- Update values

```
fibs = list(range(15))  # creates a list of size 15,
                        # containing nums 0 to 14
fibs[0] = 1
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 on sep line
      print(num)
```

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 **static** 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

Lists vs. Strings

- Strings are **immutable**
  - Can't be mutated?
  - Er, can't be modified/ changed
- Lists are **mutable**
  - Can be changed
  - 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`

Special Value: None

- Special value we can use
  - E.g., Return value from function when there is an error
- Similar to `null` in Java
- If you execute

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

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

Practice: Wheel of Fortune

- Modify to keep track of previous guesses
  - If user made that guess before, print message
- What are the data types of the data we're modeling?

Practice: Wheel of Fortune

- Model the wheel
  - Money
  - Bankruptcy, lose a turn, free spin
- Simulate spinning the wheel

Practice: Wheel of Fortune

- Big set of puzzles
  - How do we represent?
  - How do we pick a puzzle?
  - How do we ensure no repeating of puzzle?

Practice: Wheel of Fortune

- Big set of puzzles
  - How do we represent?
    - List of strings; each string is a puzzle
  - How do we pick a puzzle?
  - How do we ensure no repeating of puzzle?
- Alt 1:
  - Start at beginning of list, move to the next one until reach the end; repeat
- Alt 2:
  - Randomly pick a puzzle; remove puzzle from list (either using `pop` or `remove`); Repeat

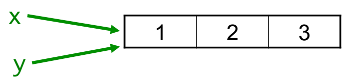
Copies of Lists

- What does the following code output?

```
x = [1, 2, 3]
y = x
y[0] = -1
print(y)
print(x)
```

List Identifiers are **Pointers**

```
x = [1, 2, 3]
y = x
```



- y is **not** a copy of x
  - Points to what x points to
- How to make a copy of y?

```
y = x + []    OR    y = []
                    y.extend(x)
```

Empty list

Wrap Up

- Similarity and differences between lists and strings

Assignments

- For Friday
  - Lab 5 due
  - A Comparison of bugs due (Broader Issue)
- Extra credit opportunities
  - Lab problems
  - Review articles (similar to broader issues)