Objectives

- Continuing lists
- Introduction to Files

Review

- What is a list?
- What is the syntax for a list?
- How are lists and strings similar?
- How are they different?
  - What are the implications of those differences?

Review: Lists vs. Strings

- Strings are **immutable**
  - Can't be mutated?
  - Err, 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"]
```

One effect: list methods modify the list on which the method was called
→ Don't return a copy of the object, modified

Special Value: None

- 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

(Similar to `null` in Java)

- If you execute

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

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

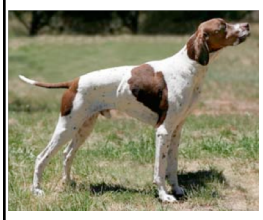
Copies of Lists

- What does the following code output?

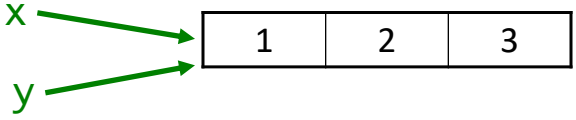
```
x = [1, 2, 3]
y = x
y[0] = -1
print(y)
print(x)
```

- Run in Python interpreter
- View in Python visualizer

List Identifiers are **Pointers**



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



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

```
y = x + []
```

OR

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

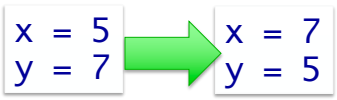
Empty list

Immutable vs Mutable Parameters

PASSING PARAMETERS

Passing Parameters

- Only ***copies*** of the actual parameters are given to the function
  - For **immutable** data types Which are?
- The ***actual*** parameters in the calling code do not change
- **Swap example:**
  - Swap two values in script
  - Then, put into a function



Immutable Data is Passed by Value

```
def swap(a, b):
    tmp = a
    a = b
    b = tmp
    print(a, b)

x = 5
y = 7

swap(x, y)

print("x =", x)
print("y =", y)
```

This code does not have the desired effect in that x and y are not swapped.

Since integers are passed ***by value***, the values of x and y are not changed by the call to the swap function.

Lists as Parameters to Functions

If a list that is passed as a parameter into a function is **modified *in* the function**, the list **is modified *outside* the function**

- Lists are **not** passed-by-value/copied
- Different from immutable types (e.g., numbers, strings)

- Parameter is actually a **pointer** to the list in memory

Problem: Sort a list of 3 numbers, in descending order

```
# order list such that list3[0] >= list3[1] >= list3[2]
def descendSort3Nums( list3 ):
```

Called as:

```
list = …
descendSort3Nums(list)
print(list)
```

How implemented with list methods?
Can we do this using only 3 comparisons?

Descend Sort a List w/ 3 elements

```
def descendSort3Nums(list3):
    if list3[1] > list3[0]:
        # swap 'em
        tmp = list3[0]
        list3[0] = list3[1]
        list3[1] = tmp

    if list3[2] > list3[1]:
        tmp = list3[1]
        list3[1] = list3[2]
        list3[2] = tmp

    if list3[1] > list3[0]:
        tmp = list3[0]
        list3[0] = list3[1]
        list3[1] = tmp
```

```
def main():
    list = [1,2,3]
    descendSort3Nums(list)
    print(list)
```

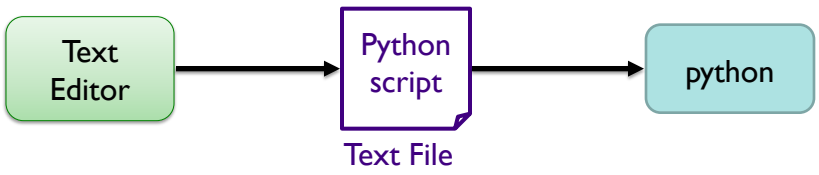
Function does **not** *return* anything. Simply modifies the `list3` parameter.

Sources of Input to Program

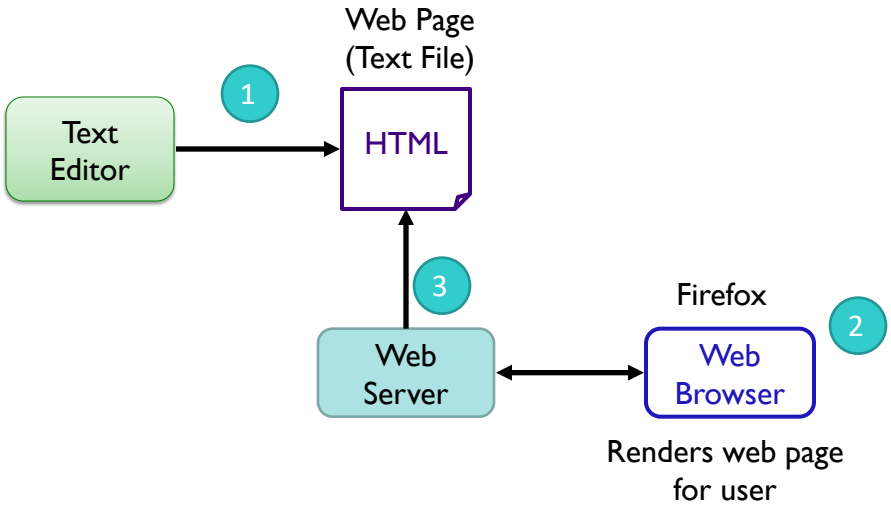
- User input
  - Slow if need to enter a lot of data
  - Error-prone
    - User enters the wrong value!
  - What if want to run again after program gets modified?

Sources of Input to Program

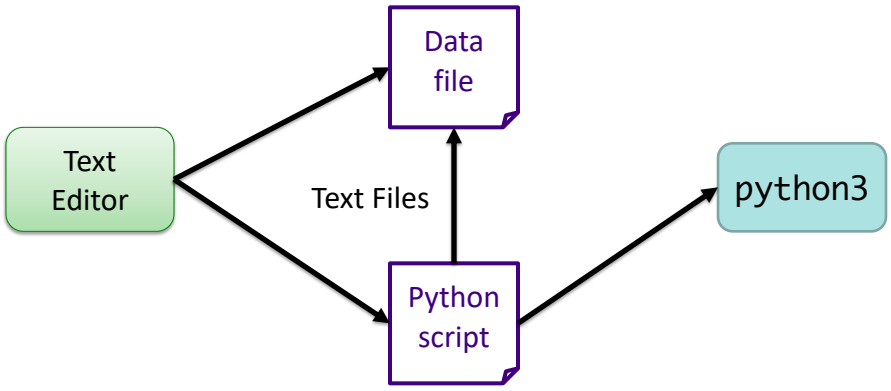
- Text files
  - Enter data once into a file, save it, and reuse it
  - Good for large amounts of data
  - Programs can use files to *communicate*
  - Need to be able to *read from* and *write to* files



Example Use of Files



Example Use of Text File as Input



Wheel of Fortune

- Uses a file of puzzles
  - Can modify puzzle file to get different puzzles

Files

- Conceptually, a file is a ***sequence*** of data stored in memory
- To use a file in a Python script, create an object of type `file`
  - `file` is a *data type*
  - `<varname> = open(<filename>,<mode>)`
    - `<filename>`: `string`
    - `<mode>`: `string`, "r" for read, "w" for write, "a" for append (and others)
  - Ex: `dataFile = open( "years.dat", "r" )`

Built-in function
"constructs" a `file` object

Common File Methods

Method Name	Functionality
`read()`	Read all the content from the file, returned as a string object
`readline()`	Read next line from file, returned as a string object (which includes the "\n"). If it returns "", then you've reached the end of the file
`write(string)`	Write a string to the file
`close()`	Close the file. Must close the file after done reading from/writing to a file

Reading from a File

- Examples of reading from a file using file methods
  - Show file: `data/famous_pairs.txt`

Typically use `.dat` or `.txt` file extension to name files containing data or text

- `file_read.py` (using `read()`)
  - How is what Python printed different than the file's content?
  - How to fix?
- Using `readline()`

Reading from a File

- Recall that a file is a ***sequence*** of data
- Can use a `for` loop to iterate through a file

A *line* (of type `str`) from the file (includes \n)

`file` object

```
for line in dataFile:
    print(line)
```

➢ Read as: for each line in the file, do something

`for_file_read.py`

Data Types of Loop Variables

What are the data types of the loop variable **x**?

```
myString = "some string"
dataFile = open("datafile.dat", "r")

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

for x in myString:
    # loop body …

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

Data Types of Loop Variables

What are the data types of the loop variable **x**?

```
myString = "some string"
dataFile = open("datafile.dat", "r")

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

for x in myString:
     # loop body …

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

integer

string → single characters

string → line (include \n)

Looking Ahead

- Lab 7 due Friday
- BI: Cryptography due Friday