Objectives

- Lists, continued
  - Making copies
  - Passing as parameters
- Introduction to Files

Review

- What is a list?
- What is the syntax for a list?
- How can we make a list of numbers with a fixed step quickly?
- How are lists and strings similar?
  - What similar things can we do to lists and strings?
- How are they different?
  - What are the implications of those differences?
- What does None mean? When does it come up?

Review: Lists and Strings in Common

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>`

Review: Lists vs. Strings

- Strings are **immutable**
  - Can't be mutated?
  - Err, can't be modified/changed
    - A change requires recreation
- 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"]
```

Review: Lists vs. Strings

Strings

- Methods that are meant to change a string return a *changed copy* of the String
- Consequence: Call the 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 method but don't assign it to a variable
- Example use:
  - `myList.sort()`

5

Review: 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

6

Copies of Lists

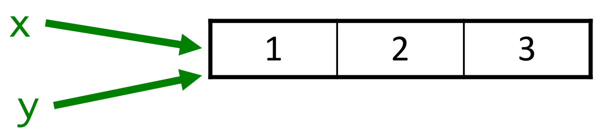
- What does the following code display?

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

- Run in Python interpreter

List Identifiers are **Aliases**

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



- y is **not** a copy of x
- y is another alias to that list/object
  - y points to what x points to

Copies of Lists

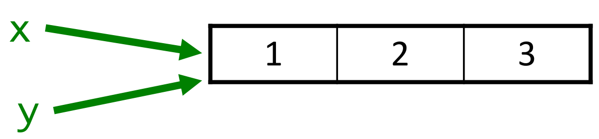
- What does the following code display?

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

- View in Python visualizer

List Identifiers are **Aliases**

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



- y is **not** a copy of x
- y is another alias to that list/object
  - ➢ y points to what x points to
- How to make a copy of x?

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

Empty list

OR

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

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

```
x = 5        x = 7
y = 7   ->   y = 5
```

Recall: Immutable Data is Passed by Value

```
def main():
    x = 5
    y = 7

    swap(x, y)

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

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

main()
```

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.

swap.py

Lists as Parameters to Functions

- Lists are not passed-by-value/copied
- Different from immutable types (e.g., numbers, strings)
- Function parameter is actually a pointer to the list in memory

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

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?

descendSort.py

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.

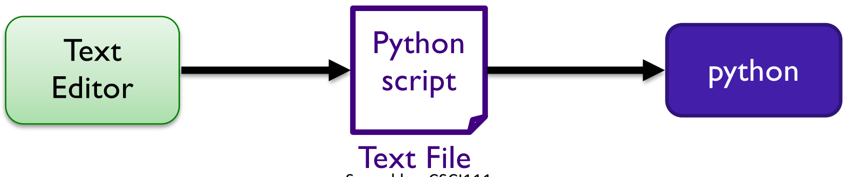
FILES

Sources of Input to Program: User Input

- Pros
  - Easy!
  - Intuitive!
- Cons
  - 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?

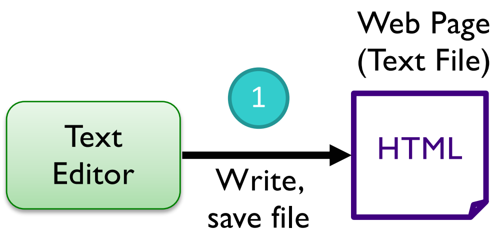
Sources of Input to Program: Text Files

- Pros
  - 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
- Cons
  - Not as intuitive in programming
  - Requires creating a file



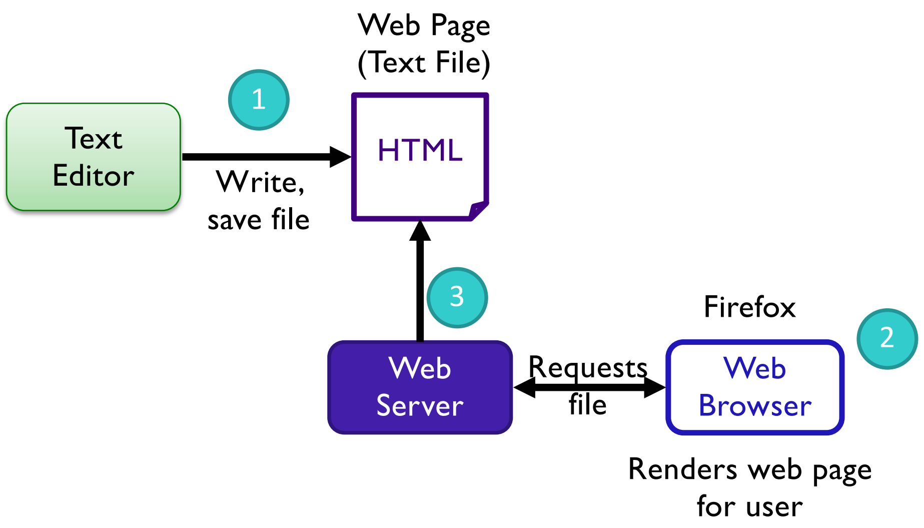
19

Example Use of Files: on the Web

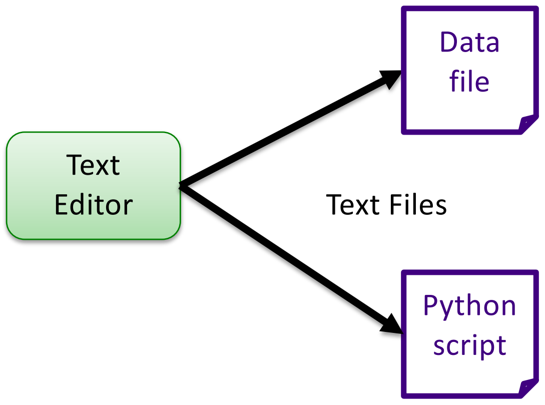


20

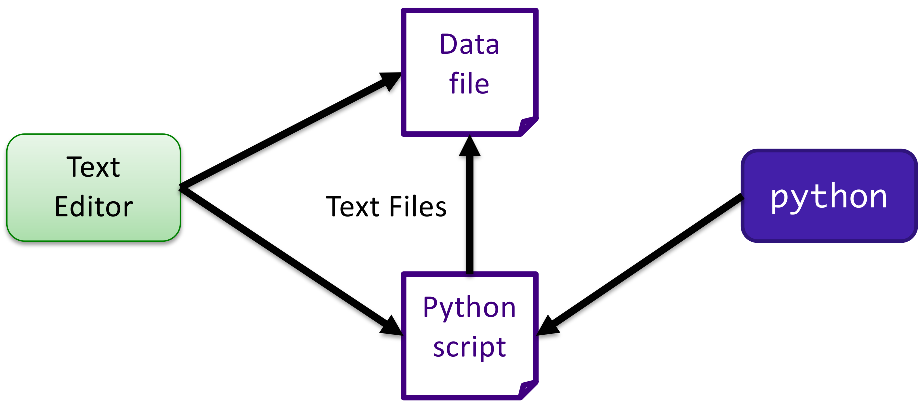
Example Use of Files: on the Web



Example Use of Text File as Input: Data!



Example Use of Text File as Input: Data!



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>)`

    Built-in function "constructs" a file object
    - `<filename>`: `string`
    - `<mode>`: `string`, "r" for read, "w" for write, "a" for append (and others)
  - Ex: `dataFile = open( "temps.dat", "r" )`

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

25

Reading from a File

- Examples of reading from a file using file methods
  - Example: `data/famous_pairs.txt`
- `file_read.py` (using `read()`)
  - How is what Python printed different than the file's content?
  - How to fix?
- Using `readline()`

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

`file_read.py`
`using_readline.py`

26

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 …

`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

- Pre Lab 8 due before lab on Tuesday