

Lab 8

- Pair programming
- Feedback on Lab 7
- Review
 - Lists
 - Files
 - Modules

1

Lab 8: Pair Programming

Every lab,
pairs will change

Name (alphabetically)	Partnered with		Name (alphabetically)	Partnered with
Aiden	Nick		Jenna	Amanda
Amanda	Jenna		Lakpa	Cassandra
Cassandra	Lakpa		Mac	Declan
Cole	Elle		Mary	Ford
Declan	Mac		Matt	Renan
Elle	Cole		Nick	Aiden
Ford	Mary		Patrick	Han
Han	Patrick		Renan	Matt
Jack	Shelby		Shelby	Jack

Sit with your
teammate.

You can each
log in, but one
computer will
be the
machine you
[both]
program on.

2

Pair Programming

- **Two people work together at a single computer**
- **Driver** and **Navigator** work together on one task
- Roles change often
- Collective responsibility for outcome
- One approach used in “real world”

3

Pair Programming Tradeoffs

Pros

- Bring together multiple perspectives, experiences, abilities, and expertise
- Higher quality code
 - Catch bugs earlier
- Knowledge transfer
- Enhanced learning, communication
- Requires 100% engagement

Cons

- Slows down lines per minute (~15%)
- Loss of autonomy
- Scheduling
- Overconfidence
- Concentration
- Requires 100% engagement

4

Pair Programming Roles

Driver

- The role *I* play when we write programs in class
- Uses keyboard and mouse to execute all actions on the computer
- Ask questions wherever there is a lack of clarity
- Offer alternative solutions if you disagree with the navigator
 - When there is disagreement, defer to the navigator. If idea fails, get to failure quickly and move on
- Make sure code is “clean”
- Explains actions taken
- Brainstorms

Navigator

- The role *you* play when we write programs in class
- Directs driver’s actions
 - Dictates the code that is to be written - the “what”
 - Clearly communicates what code to write
- Explains *why* chose particular solution to this problem
- Checks for errors and typos
- Plans the problem solving or debugging actions
- Asks questions

Your team will create your own workflow, within these guidelines

March 16, 2022

5

5

Expectations

- Take collective ownership of the code you and your partner are writing
 - No “my part” and “your part.”
- Be an active, engaged, respectful team player
 - Goal: 50/50 division of labor (brainstorming, typing, testing, problem-solving, debugging, ...)
 - Speak up when you don’t understand, think there is an error, or wonder if there is a better way
 - Don’t be too proud to admit a mistake
 - Apologize if you hurt your partner’s feelings

March 16, 2022

Sprengle - CSCI111

6

6

Expectations

- Be open-minded
 - Pair programming is an opportunity to learn
 - One of the most important predictors of success in pair programming is buy-in: if you are determined to make the practice fail, it will.
- Coordinate breaks (e.g., for bathroom)
- Seek advice when you need it
 - We're still here to help
 - We'll ask even more questions to guide your approach

Expectations

- Don't be bound to the keyboard/mouse/monitor
 - Read the instructions, pause and think on your own
 - Draw pictures
 - Refer to handouts
- Break down the problem into manageable pieces
 - Not always in order of problem
- Comments – include both authors

Submissions

See lab for more information

- At the end of the lab period, run `pairturnin` to copy your files to a shared location
- If you finish the lab in class, `pairturnin` also serves as your electronic submission
- Otherwise (if you didn't finish),
 - If you want to continue working together after lab, you can, **BUT** you must always work together in pair
 - **No** working partly on your own/partly together
 - Run `pairturnin` when you complete the lab
 - Use `indiv_startup` to copy the shared code into your own lab directory
 - If you want to finish up on your own
 - Run `indiv_startup` to copy the shared code into your own lab directory
 - Submit using `turnin` as usual

Advice from Previous Students

- I would advise them to really study the material and make sure they **understand the vocabulary** so that they can have **useful conversations** with their partners and the **navigator can actually navigate**.
- I would advise them to **prepare for the labs beforehand** so the person and their partner would be on **even footing** before the lab.
- **Try talking through** the syntax, semantic, or EOF **error** before asking for help. By doing so, if the pair does not figure out the error on their own, they are able to **explain their error better** after talking it through as a pair.

Advice from Previous Students

- If your partner seems to be taking a lead and you are **falling behind, voice** that to them and ask if they can **help you to understand** the material better. Try to be conscious of your role and if you are taking too much of a lead.
- I would also recommend, even though the labs are finished in pairs, to make sure **practice is done on your own before taking the tests.**
- It often felt as if one partner was far more skilled than the other and thus that only one should have a say. As we began to **practice this more**, I think that the class as a whole became far more relaxed and were able to **begin to truly collaborate effectively.**

March 16, 2022

Sprenkle - CSCI111

11

11

Advice from Previous Students

- **Trust your partner** more than you think you should; be willing to take an **objective step back** and start over; **test often** and **test well**; and **experiment to prove to yourself how something works.**
- I recommend that they learn to listen to their partner. By actually hearing out their ideas, you can learn a lot about how **problems can be solved in different fashions** and be able to **apply these skills later on in their lives in and out of programming.**
- My best advice for pair programming and programming is **to not be afraid to fail.** It is okay to mess up, your partner will not judge you for a mistake, but you have to **not be afraid to fail in order to succeed in computer science**

March 16, 2022

Sprenkle - CSCI111

12

12

LAB 7 FEEDBACK

March 16, 2022

Sprenkle - CSCI111

13

13

Review Caesar Cipher

- Consider the following (partial) solutions

```
for char in message:  
    asciiVal = ord(char)  
    if asciiVal == 32:  
        ...  
    else:  
        ...
```

```
for char in message:  
    if char == " "  
        ...  
    else:  
        ...
```

March 16, 2022

Sprenkle - CSCI111

14

14


Review Caesar Cipher

- Consider the following (partial) solutions

```
for char in message:  
    asciiVal = ord(char)  
    if asciiVal == 32:  
        ...  
    else:  
        ...
```

I know what " " means.
I don't immediately know what 32 means.
Lesson: prefer words over numbers.

```
for char in message:  
    if char == " "  
        ...  
    else:  
        ...
```



March 16, 2022

Sprenkle - CSCI111

15

15

Comment Example

```
def encryptLetter(letter, key):  
    """  
    Encrypts a single letter by the given key.  
    Parameters:  
    - letter: a single, lowercase character string  
    - key: an integer (between -25 and 25, inclusive)  
    POST: returns the encrypted character as a str  
    """
```

- Does not say *who* called function, where parameters came from, or where returned to
 - Any code can call the function and pass in input from anywhere (e.g., hardcoded, from user input, test function, ...)
- Does not say variable name returned

March 16, 2022

Sprenkle - CSCI111

16

16

Comment Example 2

```
def encryptLetter(letter, key):  
    """Encrypts a single letter.  
    PRE: Input parameters are a single, lowercase  
    character string (char) and an integer key  
    (between -25 and 25, inclusive)  
    POST: returns the encrypted character as a str"""
```

- Does not say *who* called function, where parameters came from, or where returned to
 - Any code can call the function and pass in input from anywhere (e.g., hardcoded, from user input, test function, ...)
- Does not say variable name returned
- **Format doesn't matter as much as contains required content**

Test Functions

- Designing test functions
 - Pick good test cases
 - Automatically (i.e., program) checks results so it's easy to spot problems
 - Report input/test cases that cause the problems
- Benefits:
 - Quickly and automatically test functions
 - Quickly add new test cases
 - Can rerun test cases quickly if function implementation changes
 - If tested well, you can use the function in other programs with confidence

Review

- What are things we can do to lists?
- How do we work with files?
 - What are things we can do with files?
- What is a module?
 - What are the benefits of modules?
 - How do we create a module?
 - How do we use functions defined in a module?

Review: List Operations

Similar to operations for strings

Concatenation	<code><seq> + <seq></code>
Repetition	<code><seq> * <int-expr></code>
Indexing	<code><seq>[<int-expr>]</code>
Length	<code>len(<seq>)</code>
Slicing	<code><seq>[:]</code>
Iteration	<code>for <var> in <seq>:</code>
Membership	<code><expr> in <seq></code>

Review: List Methods

Method Name	Functionality
<code><list>.append(<i>x</i>)</code>	Add element <i>x</i> to the end
<code><list>.sort()</code>	Sort the list
<code><list>.reverse()</code>	Reverse the list
<code><list>.index(<i>x</i>)</code>	Returns the index of the first occurrence of <i>x</i> , Error if <i>x</i> is not in the list
<code><list>.insert(<i>i</i>, <i>x</i>)</code>	Insert <i>x</i> into list at index <i>i</i>
<code><list>.count(<i>x</i>)</code>	Returns the number of occurrences of <i>x</i> in list
<code><list>.remove(<i>x</i>)</code>	Deletes the first occurrence of <i>x</i> in list
<code><list>.pop(<i>i</i>)</code>	Deletes the <i>i</i> th element of the list and returns its value

Note: methods do **not return a copy** of the list ...

March 16, 2022

Sprenkle - CSCI111

21

21

Review: 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?

March 16, 2022

Sprenkle - CSCI111

22

22

Review: 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?

Review: 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

Review: 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*

Built-in function
"constructs" a file object

➤ `<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")`

Review: Writing to a File

- Create a file object in **write** mode:

➤ `myFile = open("demo.txt", "w")`

- Call write method on file object:

➤ `myFile.write("Write string to file")`

➤ `myFile.write("Also this string")`

What will the output file look like?

- Close the file:

➤ `myFile.close()`

Review: Modules

- Modules group together related functions and constants
- Unlike functions, no special keyword to define a module
 - A module is named by its filename
- You've used modules in the past
 - graphics.py
 - test.py
 - game.py

Just a
Python file!

Problem: Temperature Data

- **Given:** data file that contains the daily high temperatures for last year at one location
 - Data file contains one temperature per line
 - Example: data/florida.dat
- **Problem:** What is the average high temperature for the location?

```
def calculateAvgTemp( datafileName ):
```

Rule of Thumb: Always look at data file before processing it

Problem: Temperature Data

- Implement the algorithm

Problem: Report of Avg Temperature

- **Given:** data files that contains the daily high temperatures for last year at various locations
 - Data file contains one temperature per line
 - Example: `data/florida.dat`
- **Problem:** Write a report of the locations and the average temperature in the form
 - Average temperature `<location1> <avgtemp1>` displayed to two decimal places
`<location2> <avgtemp2>`
...

Problem: Report of Avg Temperature

- Algorithm:
 - Open the file for writing
 - Write out the data to the file
 - Use format
 - Include the `\n`
 - Close the file

Recursive Copy

- Many Unix commands have command-line options
 - Example: `ls -l`
 - `-l`: long form
 - Command run during `turnin` so you can see the dates and other information on your submitted files.
- `cp` has the `-r` option, which means to *recursively* copy
 - Meaning to copy the directory and all of its contents (including subdirectories)
 - Example: to copy the `lab8` directory and all of its contents into your `cs111` directory
 - `cp -r /csci/courses/cs111/handouts/lab8 ~/cs111`

Lab 8 Overview

- Lists
- Modules
- Reading Files
- Writing Files
- Functions, Lists

Focus is on the current week, but we are using tools we learned in the last ~8 weeks.
Remember (or review) all that you can do.