CS 2316 Learning Objectives

This document lists the CS 2316 Learning Objectives and tries to give you an idea of what each learning objective encompases. Each learning objective will have a list of concepts that are related to or contained by the learning objective. We also list some sample questions that you should be able to answer if you have completed the learning objective. The example scoring metrics should give you an idea of what a 1,2,3, or 4 score mean, but are not absolute guidelines.

Learning Objective: Template First Page

Phase:

Concepts students should understand:

Example Task Requests or Questions TA's might have during a demo:

Example Scoring Metrics:

1 - Understands what they are supposed to be doing, but can't get the code working.
2 - Has working code, but can't modify it to make changes or explain how / why it works.
3 - Has working demo code, can make modifications to change behavior.
4 - Clearly demonstrated mastery of the subject. Demo code does EVERYTHING you can do with this topic, and the student can create new code on the fly to perform a similar task.

Learning Objective: Variable Assignment

Phase: 1

Concepts students should understand:
Students should understand how to create a variable and make it point to a value using the assignment statement (=). They should understand that the entire expression on the right side of the assignment statement is evaluated to determine the value before the assignment occurs. (allowing statements such as x = x+1) They should understand that variable names can contain letters, underscores, and numbers, but may NOT begin with a number, and that names are case sensitive.

Example Task Requests or Questions TA's might have during a demo:
Can you create a new variable that points at 5? Can you change it to now point at "test"? You used "var" as your variable name, what happens if you say "Var=3" now? What are the rules for what is a valid variable name?

a = "aStr"
b = a
a = "New String"
print(a) = ?
print(b) = ?

Example Scoring Metrics

1 Unable to create a new variable, or does not understand the direction of assignment. ("test" = variable).
2 Can create single variables, but believes that the = statement sets up a mathematical equivalence.
3 Successfully demonstrates ability to initialize variable and reassign value.
4 Understands using the same variable on both sides of the assignment statement.

Learning Objective: Simple Expressions

Phase: 1

Concepts students should understand:
Students should understand how to do basic calculations using numbers and variables.

Example Task Requests or Questions TA's might have during a demo:
What does 5 * 10 + 2 evaluate to in python? How many times does 3 divide 10 evenly? What is the remainder left over? How do you calculate 8 raised to the 3rd power?

Example Scoring Metrics:

- **1 -** Can not use basic mathematical expressions.
- **2 -** Can use basic math, but not with variables.
- **3 -** Can perform basic mathematical expressions using operators such as (+, -, *, /).
- **4 -** Understands operator precedence, and advanced operators such as **, //, and %

Learning Objective: Calling Functions

Phase: 1

Concepts students should understand:

Students should understand that when you call a function, the flow of execution passes to the code inside the function until it returns. Arguments you give to a function are passed into the function's code as parameters, which are local to the function. When a function that is called returns a value, the function call evaluates to the value, allowing you to incorporate the function into an expression, and save the result with an assignment statement.

Example Task Requests or Questions TA's might have during a demo:

Can you call a function for me? Where have you passed in an argument to a function? If you put a variable inside the argument list, what gets passed into the function? What happens if you put one function call inside the argument list of another function? How do you save the value your function call returned?

Example Scoring Metrics:

1 - Can not call functions.

2 - Can call functions without parameters.

3 - Understands how to call functions, passing in 0 or more parameters.

4 - Understands concepts such as function compositing, and saving the return value of a function.

Learning Objective: Defining Functions

Phase: 1

Concepts students should understand:

Students should be able to create (define) their own functions that take in parameters and return values. They should understand how to use multiple parameters as local variables (or create a function with no parameters) and how to return a result from the function.

Example Task Requests or Questions TA's might have during a demo:

What happens if you left the return statement out of this function? How can you know the type of an input parameter? Write a function for me that accepts two numbers, adds them together, and returns the result. What is a default parameter? How would you use one?

Example Scoring Metrics:

1 - Understands what they are supposed to be doing, but can't get the code working.

2 - Has working code, but can't modify it to make changes or explain how / why it works.

3 - Can define functions that take in multiple arguments and return values.

4 - Clearly demonstrated mastery of the subject. Functions work with multiple parameters and return appropriate values. The student can create new code on the fly to perform a similar task. Understands default parameters.

Learning Objective: Conditionals and Booleans

Phase: 1

Concepts students should understand:

Students should be able to conditionally execute blocks of code based upon a value in a variable (whose value is not known until run time, possibly because it came from a user, a function parameter, or data loaded from the internet or a file.) They should understand how to nest conditionals within each other and chain multiple elif clauses together. They should know how to use comparison operators such as (==, !=, <, <=, in) and use boolean operators such as and/or/not to make more complicated decision functions or simplify the nesting of conditionals.

Example Task Requests or Questions TA's might have during a demo:

Write a conditional statement that tells the user what type of vehicle you are driving based upon the number of wheels it has. Now modify it to use only < (less than) conditional operators (or == operators, or > operators). What does this <boolean expression> evaluate to? Explain why.

Example Scoring Metrics:

1 - Understands what they are supposed to be doing, but can't get the code working.

2 - Can implement basic conditionals, use basic conditional expressions, and evaluate basic conditional expressions (<, ==, !=, etc) on paper.

3 - Can successfully implement an if, elif, else chained conditional in code. Can use compound boolean expressions (single and/or/not) can evaluate compound boolean expressions on paper

4 - Clearly demonstrated mastery of the subject. Can implement nested conditionals. Demonstrates ability to implement complex nested conditionals. Can use compound complex (more than one conjunction) boolean expressions. Can evaluate compound complex (more than one conjunction) boolean expressions on paper.

Learning Objective: While Loops

Phase: 1

Concepts students should understand:

Students should be able to use while loops to execute a block of code until some condition is met. They should be able to use a counter to execute a block of code a specific number of times. They should understand what (does not) happen if the boolean conditional in the while expression evaluates to False the first time it is executed.

Example Task Requests or Questions TA's might have during a demo:

Write a while loop that counts from 3 to 10 skipping by 2's. Now make it count from 20 to 3 skipping by threes. Why does this while loop go forever? How would you make it work correctly?

Example Scoring Metrics:

1 - Understands what a while loop does, but can't get the code working.

2 - Can use a basic while loop.

3 - Knows how to avoid an infinite while loop.

4 - Can manipulate while loop to accomplish different tasks. Demonstrates conceptual understanding of while loops. Knows when they are appropriate to use, and when a for loop would be a better choice. Clearly demonstrated mastery of the subject. Demo code does EVERYTHING you can do with this topic, and the student can create new code on the fly to perform a similar task.

Learning Objective: For loops

Phase: 1

Concepts students should understand:

Students should understand the syntax of a for loop, what the looping variable does, and how it iterates through a sequence. In addition, they should be able to use the range() function to cause a for loop to execute a block of code a particular number of times, and be able to specify different step sizes when moving through numbers.

Example Task Requests or Questions TA's might have during a demo:

Write a for loop that counts from 1 to 10. From 1 to 10 counting by 2's. From 10 to 1 counting by 3's. Write a for loop that prints every item in this sequence (list/string). Write a for loop that prints every OTHER item in this list/sequence. Here is a while loop, convert it to a for loop (or explain why it can not be converted). What does this looping variable point at once the for loop is finished running?

Example Scoring Metrics:

1 - Knows what a for loop is, but can't get the code working.

2 - Can use basic for loop, has working code, but can't modify it to make changes or explain how / why it works.

3 - Can use for loop to iterate through a collection both by generating indices (*for i in range(len(list)*) and foreach (*for item in aList*). Any iterable collection is acceptable.

4 - Can translate while loop code to for loop code. Can explain why some while loops cannot be converted to for loops. Clearly demonstrated mastery of the subject. Demo code does EVERYTHING you can do with this topic, and the student can create new code on the fly to perform a similar task.

Learning Objective: Using & Creating Strings

Phase: 1

Concepts students should understand:

Students should be able to create strings that include quote marks, double quote marks, and special characters such as newline characters and tabs. They should understand how the backslash (\ ) character is special, how to use it to escape quotes and create special characters, and how to insert a literal backslash into a string. They should know what a raw string is and how to use one. They should understand how to make multi-line strings (or multi-line string comments). They should be able to tell the actual length of a string (output of len() function) given a string with escaped characters in it. They should be able to extract a single character from a string using indexing. They should understand how to test if a character is a letter or a digit.

Example Task Requests or Questions TA's might have during a demo:

Make a string in python that contains: He said "Don't". Make a string that contains tabs and newlines. Make a string which contains the backslash character. Demonstrate the use of a raw string. How can you make a string span multiple lines? What is the length of this string? Place the 3rd character of this string into a new variable. Now write python code that will tell me if that letter is a letter or a digit.

Example Scoring Metrics:

1 - Understands what they are supposed to be doing, but can't get the code working.

2 - Can concatenate multiple strings together. Can create strings that contain quotes or double quotes. Has working code, but can't modify it to make changes or explain how / why it works.

3 - Can create strings which contain both quotes & double quotes. Can make multi-line strings. Understands the difference between the true length of a string and it's apparent length when displayed in IDLE. Has working demo code, can make modifications to change behavior. Knows and can use built in string functions (upper, lower, strip, split, etc..). Can extract a single character from a string using indexing.

4 - Clearly demonstrated mastery of the subject. Can include any type of special character in a string. Knows what a raw string is and how to use it. Knows how to test if a character is a letter or a digit or something else.

Learning Objective: Printing & String Formatting

Phase: 1

Concepts students should understand:

How to print text (strings) to the screen. How to print the contents of variables to the screen. How to combine strings and variables into statements that get printed to the screen. The difference between printing an item inside a function and returning the item from the function. How to use string formatting to insert variable data into a string, as well as format it. Students must use Python3 style string formatting (" {}" as opposed to python2 string formatting: "%s")

Example Task Requests or Questions TA's might have during a demo:

Print a message that includes the contents of a variable. What is the return value of the print function?

Example Scoring Metrics:

1 - Can print one string or variable.

2 - Can print multiple items using one print statement, can use string formatting (.format)

3 - Can specify number of decimal places to display from a float in output. Can identify the type of print function (e.g. type(print("hi")) and explain why it evaluates to that type. Has working demo code, can make modifications to change behavior.

4 - Clearly demonstrated mastery of the subject. Describes and demonstrates difference between print and return from a function. Can use new-line, tab characters, etc. to display data in formatted columns. Demo code does EVERYTHING you can do with this topic, and the student can create new code on the fly to perform a similar task.

Learning Objective: Indexing & Slices

Phase: 1

Concepts students should understand:

Students should understand how to access a single item or a subset of a sequence (such as a List, Tuple, or String). They should know how to index from the back of the sequence with negative index numbers. They should know how to use indexing with a while loop to access each item in the sequence. They should know how to use a slice operator to select a subset of the sequence, or a selected set of items stepping by numbers other than 1. Students should understand how to do item assignment to data types that support them (lists). They should also understand how to "modify" a string or other immutable data type such as a tuple using concatenation and re-assignment.

Example Task Requests or Questions TA's might have during a demo:

Extract the second item from this list and make a variable point at it. Extract every other item from this list. Extract every third item from this list, starting at the end and working towards the beginning. "Modify" a string so that the first letter is different using slices.

Example Scoring Metrics:

1 - Understands what they are supposed to be doing, but can't get the code working.

2 - Can extract single item for sequence using indexing. Can slice forward using default step. Has working code, but can't modify it to make changes or explain how / why it works.

3 - Understands and can use negative indexing to grab a single item counting from the back of a sequence. Can slice backward using default step. Can modify an item in a list. Has working demo code, can make modifications to change behavior.

4 - Can slice forward or backward using non-default step. Can "modify" an immutable data type by re-assigning the variable appropriately. Clearly demonstrated mastery of the subject. Demo code does EVERYTHING you can do with this topic, and the student can create new code on the fly to perform a similar task.

Learning Objective: Tuples

Phase: 2

Concepts students should understand:
Students should understand how to create tuples (including length zero and length one tuples). Students should understand how to "modify" a tuple (by re-assignment, much like a string). Students must be able to use tuple-assignment to pack and unpack a set of variables. Students should understand what advantages tuples have over lists.

Example Task Requests or Questions TA's might have during a demo:
Make a tuple. Access a particular item in it. Make a tuple with only one item in it. Concatenate that tuple with another one. Modify this tuple by removing an item from the middle and re-assigning to the same variable. Make a tuple out of variables. Assign some variables from a tuple. Use a tuple to swap the contents of 2 or 3 variables around.
Explain why you would ever want to use a tuple instead of a list.

Example Scoring Metrics:

1 - Can't create a tuple.
2 - Can create tuples of arbitrary length (zero, one, more)
3 - Can manipulate tuples (add items, delete items, access items) like strings.
4 - Can explain the advantages of a tuple over a list. Demonstrates tuple-assignment (tuple packing and unpacking).

Learning Objective: Lists

Phase: 2

Concepts students should understand:

Students should be able to understand how to construct lists of varying lengths via declaration, how to append new values to lists, how to remove values from a list, how to sort a list (.sort() vs sorted()), quickly find minimum and maximum values (min() and max()) and how to concatenate lists.

Example Task Requests or Questions TA's might have during a demo:

Make a new list with several values in it. Append some more values to this list. Sort this list. Create a new list. Concatenate the new list with the old list. Sort this list again using a different sort method from the one you used before. Find the minimum value of the list. Find the maximum value of the list.

Example Scoring Metrics:

1 - Can create a list with one value in it

2 - Can create a list with multiple values and access values of the list

3 - Can append values and concatenate lists, can also delete values from lists

4 - Student understands the differences between .sort() and sorted() and can use min()/max() effectively

Learning Objective: Dictionaries

Phase: 2

Concepts students should understand:
Students should understand how a dictionary works, the relationship between keys and values, and how to add new key/value pairs. They should understand how to detect if a key exists in a dictionary, how to retrieve a value given a key, and how to get a list of keys, values, or key/value pairs. Students should know what data types can and cannot be used as a key or a value.

Example Task Requests or Questions TA's might have during a demo:
Make a dictionary (with a few items already in it). Add a key/value pair to it. Look up the value associated with a particular key. Test to see if this key is in the dictionary (without using a try/except).

Example Scoring Metrics:

1 - Can initialize and create a dictionary with one key/value pair in it
2 - Can add more values to a dictionary; Student understands that keys are immutable
3 - Can change a value in the dictionary and can delete a key/value pair
4 - Student can test to see if a key is in the dictionary

Learning Objective: Using Objects

Phase: 2

Concepts students should understand:

Students should know the difference between an object and a class. They should know how to create and use a object. They should know the difference between an object variable and a class variable.

Example Task Requests or Questions TA's might have during a demo:

What is an object? What is a class? What is an instance? What is the difference between an object variable and a class variable? Be able to call an object variable and a class variable.

Example Scoring Metrics:

1 - Understands what they are supposed to be doing, but can't get the code working.

2 - Has working code, but can't modify it to make changes or explain how / why it works.

3 - Has working demo code, can make modifications to change behavior.

4 - Clearly demonstrated mastery of the subject. Demo code does EVERYTHING you can do with this topic, and the student can create new code on the fly to perform a similar task.

Learning Objective: Simple Recursion

Phase: 2

Concepts students should understand:

Students should understand what recursion is. They should know how to write a recursive function. They should know the 3 requirements for “good” recursion. They should understand how each version of the function gets it’s own local variables. They should understand how returned values are handled.

Example Task Requests or Questions TA’s might have during a demo:

Write a Fibonacci sequence using a recursion. Write the factorial function using recursion. Get input from the user (force good input) using recursion. What are the 3 required components in writing a good recursive function? When a recursive function calls itself again, how are return values handled? Draw a stack diagram of how each version of this function calls itself.

Example Scoring Metrics:

1 - Understands what they are supposed to be doing, but can’t get the code working.

2 - Has code with a recursive function call, but something doesn’t work (lack of termination condition, not returning data correctly up the stack, etc...)

3 - Code solves a problem using recursion, but student doesn’t fully understand how it works.

4 - Clearly demonstrated mastery of the subject. Can draw a stack diagram explaining the recursion and the copies of local variables that each version of the function call has. Demo code does EVERYTHING you can do with this topic, and the student can create new code on the fly to perform a similar task.

Learning Objective: GUI Widgets

Phase: 2

Concepts students should understand:

Students should understand what each widget is used for, and know the difference among them. They should be able to create widgets on the spot, and know what possible parameters are accepted for each widget.

Example Task Requests or Questions TA's might have during a demo:

Know how to create a button, a label, an entry box, radio buttons, and checkboxes. Know what kind of parameters can be passed on inside the widget definition. Understand the difference between widgets, and know how to use them for proper use. Know how to use .config for each widget.

Example Scoring Metrics:

1 - Understands what they are supposed to be doing, but can't get the code working.

2 - Has working code, but can't modify it to make changes or explain how / why it works.

3 - Has working demo code, can make modifications to change behavior.

4 - Clearly demonstrated mastery of the subject. Demo code does EVERYTHING you can do with this topic, and the student can create new code on the fly to perform a similar task.

Learning Objective: GUI Pop-Up

Phase: 2

Concepts students should understand:

Students should understand how to use file dialogs. They should know the difference between a file dialog and a message box. They should know how to open or save a file using a file dialog. They should also know how to display an error message when appropriate.

Example Task Requests or Questions TA's might have during a demo:

What is the difference between a file dialog and a message box? Create a file dialog that asks the user to open a file. Create a file dialog that asks the user where to save the file. Display an error message using a message box. Change the title of the message box.

Example Scoring Metrics:

1 - Understands what they are supposed to be doing, but can't get the code working.

2 - Has working code, but can't modify it to make changes or explain how / why it works.

3 - Can pop up one or two dialogs, but can not change the messages they display..

4 - Clearly demonstrated mastery of the subject. Knows how to use several different types of dialog (file open, file save, ask Yes No, Error Message, etc...).

Learning Objective: GUI Layout

Phase: 2

Concepts students should understand:

Students should understand the difference between using .pack() and .grid(). They should be able to manipulate widgets around in a GUI using either method. They should know how to use the possible parameters for .pack(), especially SIDE, and understand the default settings. They should know the possible parameters for .grid() (including but not limited to row, column, and sticky).

Example Task Requests or Questions TA's might have during a demo:

Are there any widgets you can't use .pack() on? What about .grid()?
What happens to the variable *widget* when you say *widget = Label(text = "Hello World").pack()*?
What happens to the layout when you change the dimensions of a widget?
Using your code, show me why this widget is located where it is on the GUI.
Add a widget to the bottom right (or top right, middle middle, etc...) of a GUI you already have created.
Create this same GUI using .pack() now, instead of .grid().

Example Scoring Metrics:

1 - Understands what they are supposed to be doing, but can't get the code working.

2 - Has working code, but can't modify it to make changes to their GUI or explain how / why it works.

3 - Has working demo code, can make modifications to change behavior using .grid(), but not .pack() or vise-versa.

4 - Clearly demonstrated mastery of both .pack() and grid(). Demo code does EVERYTHING you can do with both methods, and the student can create new code on the fly to manipulate the layout of their GUI.

Learning Objective: File (text) Input

Phase: 2

Concepts students should understand:
Students should be able to open a file, and read in the file using regular file IO methods. (Not using CSV Module.)They should understand the difference between .read, .readline, .readlines.

Example Task Requests or Questions TA's might have during a demo:
Open a file, and read in first 10 characters. Now read in 1 line. Now read in all remaining lines. Print what's stored when you read in 1 line, and explain the result.
Open this file, read all the lines, and then print every other line (or every line that contains a "#" mark).

Example Scoring Metrics:

1 - Understands what they are supposed to be doing, but can't get the code working.
2 - Has working code, but can't modify it to make changes or explain how / why it works.
3 - Can open and read from a file. Can select certain lines to print from the file.
4 - Clearly understands the difference between read(), readline(), and readlines(). Understands what the functions return when you hit the end of the file.

Learning Objective: GUI Input

Phase: 2

Concepts students should understand:

Can use entry boxes, radio buttons, check boxes, etc. to get input from the user. Knows how to use StringVar and get/set to manipulate text in entry box from user. Use the data collected from user for a purpose.

Example Task Requests or Questions TA's might have during a demo:

Change entry box from using get/set to StringVar and vice versa. If user enters a number, can perform a mathematical expression on number

Example Scoring Metrics:

1 - Understands what they are supposed to be doing, but can't get the code working.

2 - Has working code, but can't modify it to make changes or explain how / why it works.

3 - Code can read text from the user. Student can modify code to add entry fields and read the new entry fields.

4 - Can get both text and numerical data (converted from user entered text) as well as read the state of checkboxes (and/or) radio buttons.

Learning Objective: Writing to Files

Phase: 2

Concepts students should understand:

Students should be able to write into a file using conventional File writing methods. They should be able to understand how to use .write and .writelines. They must be able to write variables of various data types to the file.

Example Task Requests or Questions TA's might have during a demo:

Demonstrate how to use write and writelines, explain the difference. How do you maintain a proper formatting of data when writing into a file? (newline characters) How do you write an empty line into a file?

Example Scoring Metrics:

1 - Understands what they are supposed to be doing, but can't get the code working.

2 - Can open file.

3 - Can use either write or writelines in an appropriate way to write readable text out to a file.

4 - Can use both write and writelines, understands why one might be preferable to the other, can write the contents of variables to the file.

Learning Objective: GUI Output

Phase: 2

Concepts students should understand:

Students should know how display textual information using different GUI widgets. Students should know how to change the font size and style. Students should be able to change the relief, width, height of a widget and be able to pad spaces of a text area. They should be able to change the background or foreground color of a widget.

Example Task Requests or Questions TA's might have during a demo:

Display strings, integers, floats using a button, a label, an entry box. How do I change the font style to Arial? How do I change the font size to 14? How do I make the text bold? How do I change the width and height of this widget? How do I add spaces to the front of this widget? How do I change the background color to red?

Example Scoring Metrics:

1 - Understands what they are supposed to be doing, but can't get the code working.

2 - Has working code, but can't modify it to make changes or explain how / why it works.

3 - Has working demo code, GUI displays text/numerical data with different fonts/ colors.

4 - Can modify their GUI to change font/size/color as requested.

Learning Objective: CSV Output

Phase: 2

Concepts students should understand:

Students should be able to write into a file using the csv writer object. They should know how to create a writer module and how to write into a file using writerow or writerows, as well as configure the delimiter and quotechar arguments and specify what each does.

Example Task Requests or Questions TA's might have during a demo:

Create a writer module. Write into a file using writerow. Write into a file using writerows. What does the delimiter and quotechar do? What does the csv file you wrote into looks like? How can we specify a delimiter other than a comma? What happens with items that have the delimiter in them?

Example Scoring Metrics:

1 - Understands what they are supposed to be doing, but can't get the code working.

2 - Has working code, but can't modify it to make changes or explain how / why it works.

3 - Can write data from a list or lists of lists using the csv module.

4 - Clearly understands how quotechars and delimiters are used (and when the CSV Writer does NOT use them...)

Learning Objective: Records with sub-items (2D data)

Phase: 2

Concepts students should understand:

How to combine lists/tuples/etc. for usability purposes. How then to access each “record” and a given subitem. They should know how to use lists nested within a list, and access sub-items. Understands how to make a class that has named attributes to store related data.

Example Task Requests or Questions TA’s might have during a demo:

Show me HW4. How did you create the subrecord, and how did you then use the data in an appropriate way?

Example Scoring Metrics:

1 - Understands what they are supposed to be doing, but can’t get the code working.

2 - Has working code, but can’t modify it to make changes or explain how / why it works.

3 - Can access/modify items within a list/tuple/dictionary that is itself stored in a list.

4 - Clearly demonstrated mastery of the subject. Demo code does EVERYTHING you can do with this topic, and the student can create new code on the fly to perform a similar task.

Project Requirement

One of the “meta” learning objectives for this class is that when you finish it you are able to complete a project for the next required course (CS 4400) and any senior capstone course that you are required to take. The eventual goal is also to prepare you to complete “real world” projects given to you by internships or your first employer. This means that you need to be comfortable building a program that is relatively complicated and is controlled by a GUI. You should be able to write a program that is able to import data from one source, manipulate/filter it, and export the results to a different output format. (e.g. Read from the web and save to an SQL database, Import from an XML file and save to a CSV file, etc...)

You can fulfill your project requirement by doing any 2 of the 3 following homeworks:

- 8 - DataMerge
- 9 a&b - GT Chat
- 10 - IBM XML

Alternatively, you can “roll your own” project (see below). As long as your project is of significant complexity (covers multiple learning objectives) and is approved by your TA/instructor, you may use it to both “check-off” various learning objectives and fulfill your project completion requirement.

Roll your own project option:

You are free to build one or more project(s) that is/are different from our suggested homeworks. If you choose this route, you will be required to submit a one page document prior to your demo that lists what your project does, and tells us which learning objectives you should receive credit for, and how your project illustrates your mastery of each of those learning objective. You are explicitly allowed to “double dip” by completing a project for a different class (or work/internship) and using it to demo learning objectives for CS 2316.

Pair/Group Projects

You are welcome to work with a partner in completing a homework as long as you list both partner’s names at the top of your submission in your collaboration statement. You may work with a different partner on each project/homework. A group size of two people is optimal for scheduling purposes and so that both group members will fully understand the code. Use pair programming practices so that both partners have “driven” and “navigated” their way through programming the project. You may have larger group sizes (up to four people maximum), if the group is working on a “roll your own” project, as long as all group members are listed in the collaboration statement. Each pair/group member will be responsible for demoing the code and successfully convincing the TA that they wrote it (or understand it well enough that they could re-write it if not working with the group).

Phase 3 & 4 Learning Objectives

Phase 3 and 4 learning objectives are somewhat more abstract than the simple Phase 1 and Phase 2 objectives, and can not be easily demonstrated individually. Instead, they will be integrated with your homework requirements for the course. We have listed the learning objectives that each suggested homework project demonstrates.

- **Mini-Homework 7 - Simple Web Scraper**
  Fulfills *HTML Download* and *HTML Data Extract* (also fulfills Phase 2 *GUI Widgets* and *GUI Layout*).

- **Homework 8 - Data Merge**
  Fulfills *HTML Download, HTML Data Extract, CSV Input, CSV Output, Filter Data, Merge Data, Large Number of Records*

- **Homework 9 (A&B) - GT Chat**
  Fulfills GUI Multi-Window, SQL Input, SQL Output, SQL Functions, SQL/GUI program

- **Homework 10 - IBM XML**
  Fulfills *CSV Output, Filter Data, Large Number of Records, XML Read, Large Number of Records*

Explanation of Phase 3 and 4 learning objectives:

You will be learning Phase 3 & 4 learning objectives by building large “project” homeworks (or making your own). We have not specified them with as much detail as the Phase 1 and 2 learning objectives. The following pages list each learning objective and gives some examples, but does not explicitly give sample questions or a rubric for what corresponds to a 1,2,3, or 4 score on the learning objective. It is assumed that if your homework works as intended and that you wrote the code yourself (or understand it well enough to be able to write it yourself, if working with a partner) that you have completed the learning objectives it demonstrates.

Phase 3 learning objectives:

GUI: Multi-Window

You can create a program that has multiple windows and allows the user to switch between them. For example, upon starting your program it presents a login screen, and once the user successfully logs in, the login screen changes to the main program window.

Data Input: CSV Input

You can import a significant amount of data from a CSV file using either the csv module or by doing the text processing yourself (we recommend you use the csv module).

Data Input: HTML Download

You can download data from a website. For this learning objective you are not required to submit a GET or POST form to retrieve the data, but you will need to know how to do that for exams (and real projects in the future.)

Data Input: HTML Data Extract

After you download the HTML of a webpage that contains data of interest, you can use regular expressions and/or plain old string searching to find the data you are interested in and extract it from the rest of the webpage.

Data Processing: Filter Data

Given a large number of records, you are able to make decisions about them (based upon some provided criteria) to decide which records are “of interest” and should be kept, deleted, or manipulated.

Project: GUI/SQL program (may be completed in phase 4)

You are able to make an interactive GUI program that allows the user to see data in an SQL database and add more data to the database. For example, see the GT Chat application of HW 9a&b.

Project: Large numbers of records (May be completed in phase 4)

You have created a program that deals with large numbers of records. For example, see the Data Merge (homework 8) or IBM XML (homework 10) homework projects.

Phase 4 learning objectives:

Data Input: SQL Input
You are able to retrieve data from an SQL database using python.

Data Output: SQL Output
You are able to place data into an SQL database using python.

Data Processing: SQL Functions
You are able to use SQL Functions to have the database server perform calculations about data and give you back the result.

Data Input: XML Read
You are able to read data from an XML file.

Data Processing: Merge Data
Given two different sources of data records (where the records represent the same items, but are each missing some data) you are able to determine which pairs of records “match” and merge the missing data from each record to build a more complete record.

Data Output: XML Output
You can export data to an XML file.