# CSC121 Lab 13: ITS Python Cert Prep

## Goals

In this lab assignment, students will demonstrate the ability to:

* Understand how the legacy string method format works
* Use the os, os.path, and sys modules
* Read and write date and time information with the datetime module
* Create and run an automated test using the unittest module

NOTE: This lab document is a copyrighted work of Wake Tech Community College and the course instructor. Any posting of this document outside of Wake Tech is considered a copyright violation. Students who post these documents outside of Wake Tech are subject to academic and possible legal actions.

## Instructions

In this lab, you will create programs which cover topics on the Certiport ITS Python Certification Exam we have not yet covered in class.

Follow the instructions in each problem and submit the specified files.

Problem 1 has a starter file you will modify according to the problem specification. Problems 2-5 are all programs that you create from scratch that meets the problem specification. Problem 5 will use an additional Python file that is provided on Blackboard.

## Problems

### Problem 1 – Using the string format method

In this problem, you will explore how you can produce similar output that f-strings produce using the legacy format method provided by the string data type.

Download the file **Lab13P1-starter.py** from Blackboard and rename it **Lab13P1.py**. Run the program before changing it so that you can see the output. The output should not change from the updates that you make.

There are two TODO sections in the main:

* In the first TODO section, change the 4 output statements so they are using f-strings instead of the string format method.
* In the second TODO section, change the 4 output statements so they are using the string format method instead of f-strings.

Submit the program file **Lab13P1.py** to Blackboard for credit.

### Problem 2 – The sys Module

In this problem, you will write a program which will use the sys module to get parameters from the command line and use them in a program that gets random numbers.

Create a file called **Lab13P2.py**. Make sure that file has a main function that is called at the end of the file.

The user will run this program from the Terminal and will provide 3 additional parameters after the program name. These parameters indicate: The start of a range of numbers, the end of a range of numbers, and how many of those numbers to randomly sample from the range. So if the user runs this on the command line:

python Lab12P1.py 1 10 3

…the program will generate a list of numbers from 1 to 10 (including the 10), and then it will use the random module’s function sample to randomly choose 3 numbers and put in another list. The results will then be printed.

Here’s a sample execution of the program:

Sample output:

C:\PycharmProjects\WakeTech\Lab12> **python Lab13P1.py 1 10 3**

Grabbing 3 random numbers between 1 and 10:

[8, 2, 10]

C:\PycharmProjects\WakeTech\Lab12> **python Lab13P1.py 5 20 6**

Grabbing 6 random numbers between 5 and 20:

[15, 9, 18, 14, 16, 19]

You must print the message **and** the sample list in your output.

To get the command line parameters, you must import the sys module and access the argv list that is available through the sys module. You will need to run this program from the Terminal in order to provide user parameters on the command line.

Submit the program file **Lab13P2.py** to Blackboard for credit.

### Problem 3 – The os and os.path Modules

In this problem, you will show that you can access the file system using the os and os.path modules.

Create a file called **Lab13P3.py**. Inside that file create two functions:

* show\_files\_and\_dirs – This function takes no parameters and returns no values.
* main – This function also takes no parameters and returns no values.

The file should import the os module. Then implement show\_files\_and\_dirs in this way:

* Call the listdir function inside the os module to get a list of files and directories in the current directory.
* Print this message: Files and directories in current directory:
* Use a for loop to go through the list of files and directories and print each name on a separate line.

The main function will then do the following:

* Call show\_files\_and\_dirs to show what is currently in the directory.
* Use the isfile function inside of os.path to determine if ‘info.txt’ is present.
  + If it is, use the rename function in the os module to change the name to ‘information.txt’.
  + If it is not, print this message: File info.txt does not exist. Cannot rename.
* Use the exists function inside of os.path to determine if ‘information’ is present.
  + If it is present, print this message: Directory information already exists. Cannot create.
  + If it is NOT, use the mkdir function in the os module to create a directory named ‘information’.
* Call show\_files\_and\_dirs to show what is in the directory after the previous actions.

If there is an info.txt in the directory, and there is not an information directory in the current directory, here’s a possible sample output of running this program:

Sample output:

Files and directories in current directory:

display\_utils.py

info.txt

Lab13P1.py

Lab13P2.py

Lab13P3.py

Lab13P4.py

Files and directories in current directory:

display\_utils.py

information

information.txt

Lab13P1.py

Lab13P2.py

Lab13P3.py

Lab13P4.py

Submit the program file **Lab13P3.py** to Blackboard for credit.

### Problem 4 – The datetime Module

In this problem, you will be using the datetime module to manipulate date and time information.

Create a file called **Lab13P4.py**. Make sure that file has a main function that is called at the end of the file.

Inside your main function, do the following:

* Get the current date and time using the now method that is available through the datetime class inside the datetime module.
* Call the strftime method on the datetime object you got from the now method 3 times and convert the date to the three different formats as shown in the sample output below:
  + 03/16/23 06:19:06 PM
  + Thursday, Mar 16, 2023
  + Thu, March 16, 2023
* While your date and time will be different, the formats should match with the sample output.
* After that, prompt the user to input a date with the format mm/dd/yyyy. Use the prompt as shown in the sample output below.
* Start a try-except block. Inside the try:
  + Convert the string the user entered to a datetime object using the strptime method. Provide the format string that will appropriately convert a two-digit month, dash, two-digit-day, dash, 4-digit year string to a datetime object.
  + Use the strftime method to get the day of the week for that date in a string: Which format code will get the weekday completely spelled out?
* After the try, add an except clause to catch a ValueError which includes a block of code to print the message “Date in wrong format.”

Sample output:

The current date/time is:

03/16/23 06:19:06 PM

Thursday, Mar 16, 2023

Thu, March 16, 2023

Enter a date (e.g. mm/dd/yyyy): 04/01/1989

That date was on a Saturday.

Submit the program file **Lab13P4.py** to Blackboard for credit.

### Problem 5 – The unittest Module

This problem will test your ability to create an automated test using the unittest module.

For this problem, make sure you download the display\_utils.py function module and place it in the same directory where you create a file called **Lab13P5.py**.

Review the display\_util.py code. Note that it has two functions:

* get\_user\_greeting takes one parameter and returns a string that includes the name in a greeting.
* sum\_is\_greater takes three parameters. It returns True if the sum of the first two parameters is greater than the third parameter.

The Lab13P5.py file will implement two automated tests that will test each of these functions. Do the following:

* Import both the unittest and the display\_utils modules.
* Create a class called TestDisplayUtils. Inside the parenthesis, specify unittest.TestCase. That will indicate that the class you created is a subclass of the TestCase class inside of the unittest module.
* Create two methods:
  + test\_get\_user\_greeting has only one parameter, the self parameter.
    - This test method should call get\_user\_greeting inside of display\_utils and pass it the name "Bob". The return value should be assigned to the variable greeting:  
        
      greeting = display\_utils.get\_user\_greeting('Bob')
    - The test method should then use the assertEqual method available via the self parameter to see if greeting is equal to:  
        
      ‘Hello Bob!’
  + test\_sum\_is\_greater has only one parameter, the self parameter.
    - This test method should call the assertTrue method that is available via the self parameter. It should pass the following into that method:  
        
      display\_utils.sum\_is\_greater(5, 8, 12)  
        
      This function call is nested inside the call to assertTrue.
* At the end of Lab13P5.py, you need to add two lines that will enable this test to call the unittest main this Python file is directly called from Python:  
    
  if \_\_name\_\_ == '\_\_main\_\_':  
   unittest.main()  
    
  Note, that there are two underscores before and after name and main on the line with the if statement.

After completing the implementation, run the file from the command line in the Terminal. You should see something similar to this sample output:

Sample output:

PS C:\PycharmProjects\WakeTech\Lab12> python Lab13P5.py

F.

======================================================================

FAIL: test\_get\_user\_greeting (\_\_main\_\_.TestDisplayUtils.test\_get\_user\_greeting)

----------------------------------------------------------------------

Traceback (most recent call last):

File "C:\PycharmProjects\WakeTech\Lab12\Lab13P5.py", line 12, in test\_get\_user\_greeting

self.assertEqual(greeting, 'Hello Bob!')

AssertionError: 'Hello-Bob!' != 'Hello Bob!'

- Hello-Bob!

? ^

+ Hello Bob!

? ^

----------------------------------------------------------------------

Ran 2 tests in 0.001s

FAILED (failures=1)

This exercise has been designed so the first of the two tests will FAIL. The test itself is valid but the code in display\_utils.py has a bug. Make sure that you turn in the Python unittest file you created from these instructions that found this bug and do NOT change the tests so that the bug is not detected.

Submit the program file **Lab13P5.py** to Blackboard for credit.

## Grading Rubric

### Grading rubric for Problem 1 (15 points)

* Program has a well-formatted and correct header [5 points]
* Program does execute correctly and produces correct results [10 points]

### Grading rubric for Problem 2 (15 points)

* Program has a well-formatted and correct header [5 points]
* Program does execute correctly and produces correct results [10 points]

### Grading rubric for Problem 3 (20 points)

* Program has a well-formatted and correct header [5 points]
* Program does execute correctly and produces correct results [15 points]

### Grading rubric for Problem 4 (25 points)

* Program has a well-formatted and correct header [5 points]
* Program does execute correctly and produces correct results [20 points]

### Grading rubric for Problem 5 (25 points)

* Program has a well-formatted and correct header [5 points]
* Program does execute correctly and produces correct results [20 points]