# CS2520 Programming Assignment #3 (50 points)

**Goal:** practice Python data structures

**Task 1: Password Management System (25 points)**

The following describes the required logistics and basic requirements. However, feel free to add your own design features.

**task1() function:**

1. Create a tuple of usernames and a list of passwords (you choose how to initialize them), e.g. usernames = (‘lyang’, ‘kSimon’, ‘danny’, …), passwords = [‘sheCodes#123’, ‘catchAllGood1%’, ‘@my2Choices’, …]. The usernames and passwords must meet the following requirements: Usernames must contain letters (lowercase as well as uppercases can be used). Usernames are unique (i.e. no two usernames can be the same.) Passwords must meet the following requirements: (1) at least 8 characters, (2) contains at least one lowercase letter, one uppercase letter, one digit, and one special character (non-letter or digit). We may assume the usernames and passwords in the initialization have met the requirements (your responsibility to ensure that.) or you could use function(s) to validate them. Either way okay, your choice.
2. Zip the usernames and passwords to form a dictionary of usernames/passwords with the username as the key.
3. Call functions (described see below, these functions must be defined inside task1() function) to perform: (1) login users; acknowledge if login successful or not. (2) change passwords; acknowledge if password change successful or not. (3) create new user; if successful, welcome the new user. If not, acknowledge it.

The following functions are needed:

1. A function to authenticate/login a user, i.e. it takes the dictionary as a parameter, prompts for a username, if username does not exist, issue error message, but do allow 3 attempts before the function returns False or None (unsuccessful). If the username is found, asks for a password. Login the user if password matches, allow 3 tries. If successful, the function returns the entered username.
2. A function to create a new user. Again, the function should take the dictionary as parameter, prompts for a username (verify the username meets the above requirements, i.e. in valid format and not duplicate with existing one) and then a password (verify that the password meets the above requirements). If one of them not valid (okay for one attempt only, but you may design multiple attempts if you wish – optional) the function returns a None value, if both the username and password are valid, the function returns the username (so that in task1() function, you could acknowledge, welcome ….).
3. A function to update the password. It takes the dictionary as well as a username as the parameter. The function asks the user to enter the current password and then a new password and verifies that the current password matches the one stored in the dictionary and the new password meets the format requirement before updating. The function terminates by returning a True value if update successful, otherwise returning a False value without updating the dictionary.

**Required test runs:** task1() function should include two test runs that covers all (primary) features described above. Test run 1 is specified below while test run2 of your choice. See below for details.

**Test run1**: Initialize the username and password lists as follows, then try to login a user, add a new user, and update password. In each action (login/add/update) try both scenarios (i.e. successful or unsuccessful).

usernames = (‘lyang’, ‘kSimon’, ‘danny’, ‘tomatcpp’, ‘csDept’, ‘CoScpp’, ‘broncoWins’, ‘ponyExp’, ‘BldgAndRooms’, ‘helloKitty’ )

passwords = [‘sheCodes#123’, ‘catchAllGood1%’, ‘@my2Choices’, ‘123abc;;;’, ‘Hello2Monday$’, ‘GoodFriday@Cpp2’, ‘CS2520@Python’, ‘JavaIsHot2!’, ‘2ManyRainingDays!’, ‘1Startup@Starbucks’]

**Test run2:** your choice of initialization list, but at least with 12 users.

**Task 2: Sets and Lists (12 points)**

Create a task2() function that includes the following actions as well as two test runs.

Let L1 be the first 50 prime numbers, L2 50 be randomly generated integers in the range [0, 100].

1. Use ***list comprehension*** to generate L1, L2
2. Form sets S1 and S2 from L1 and L2 respectively. Print out the number of elements in each set. Then, create the following two sets:
3. R1: those elements are in S1 and in S2. Print out how many elements in R1.
4. R2: those elements are in S1 or in S2 but not in both. Print out how many elements in R2.
5. **Test runs:** two test runs must be coded in task2() function.

**Task 3: Tuples (10 points)**

Create a task3() function that performs:

1. Define a function tryTuple() that takes a variable length of parameters, return min, max, and total number of arguments.
2. In task3() call the tryTuple() function 3 times (see test runs below), for each test run, capture the return values with a tuple, then print out the tuple.
3. Three test runs: (1) tryTuple(3, 5, 12, 9, 1, 2) (2) tryTuple(“hi”, “bye”, “happy”) (3) tryTuple([“cat”, 3], [“dog”, 2])

**main() function** (3 points)

Define a main() function that calls task1(), task2(), task3(). Note: all test runs are built into each task function. Call main to run.

**Submission requirement:** Past test run results to the end of the code (or to the end of each task respectively), save into .py file. Submit the .py file.

**Grading criteria:**

Program quality (use of Python features, program organization, readability).

Correctness (correct outputs for all required test runs).

Submission package: code and output properly assembled.