ICS-33: In-Lab Programming Exam #1

Name (printed: Last, First): _____ Lab # (1- 15): _____

Name (signed: First Last): _____ Seat # (1-46): _____

This In-Lab programming exam is worth a total of 100 points. It requires you to define several functions in a module. I will supply a script for calling these functions and printing their results, so that you can check them visually for correctness; we will use a batch self-check file with similar tests for grading purposes only: you will not use **bsc** files. **Although these functions are related, you can write and test these functions in any order. None of the functions rely on each other to work correctly.**

You will have approximately 105 minutes to work on the exam, after logging in and setting up your computer (I estimate that taking about 5 minutes). Make sure that you read the instructions carefully and understand the problems before writing/debugging their code: don't rush through this part of the exam. You may write-on/annotate these pages. Finally, you will write, test, and debug the functions in the **exam.py** module.

We will test your functions only for correctness; each test you pass will improve your grade, and functions that produce no correct results will earn no credit. This means that your functions must define exactly the parameters specified in the descriptions below and must work on all the example arguments; your functions should also work on any other similar arguments. To aid you writing and debugging these functions

1. Write clear, concise, and simple Python code.
2. Choose good names for local variables.

You do not need to include any comments in your code; but, feel free to add them to aid yourself. We will **not** grade you on appropriate names, comments, or Python idioms: only on correctness. You may also call extra **print** functions in your code or use the **Eclipse Debugger** to help you understand/debug your functions.

You may import and use any functions in the standard Python library and the **goody** and **prompt** modules (which will be included in the project file that you will download). Documentation for Python's standard library and these modules will be available during the exam. I have written all the standard import statements that I needed to write my solution; feel free to include other imports, or change the form of the included imports to be simpler to use.

If you are having problems with the operating system (logging on, accessing the correct folder/files, accessing the Python documentation, submitting your work) or Eclipse (starting it, setting it up for Python, running Python scripts, running the Eclipse debugger, displaying line numbers) please talk to the staff as soon as possible. We are trying to test only your programming ability, so we will help you with these other activities. But, we cannot help you understand, test, or debug your programming errors. I have asked the TAs, Readers and Tutors to NOT ANSWER ANY QUESTIONS about the exam itself: read it carefully and look at the test cases for clarification.

You should have a good understanding of the solutions to the problems in Programming Assignment #1 (especially the first 3); you should also have a good understanding of the material on Quiz #1. You should know how to read files; create, manipulate, and iterate over **list**s, **tuple**s, **set**s, and dictionaries (**dict** and **defaultdict**) in 3 ways: by **keys**, **values**, and **items**; call the functions or methods **all**, **any**; **min**, **max**, **sum**; **join**, **split**; **sort**, **sorted**; **enumerate**, **zip**. Note that you can call **min**, **max**, **sort**, and **sorted** with a **key** function (often a simple **lambda**) that determines how to order the values in the iterable argument on which these functions compute. You are free to use or avoid various Python language features (e.g., lambdas and comprehensions): do what is easiest for you, because we are grading only on whether your functions work correctly.

1 of 5

Before submitting your program, ensure that it runs (has no syntax errors) and ensure there are no strange/unneeded **import** statements at the top.

Summary:

This exam is worth 100 points. It requires you to write five functions according to the specifications below. Each problem is graded on correctness only (how many tests it passes); each problem will be worth 20 points.

The module you will download

1. Defines each method with reasonable annotated parameter names (which you can change) and a body that is **pass** (thus, it returns **None** and will pass no tests).
2. Includes a script that tests these functions individually on different legal inputs, printing the results, and printing whether or not they are correct; if they are not correct, further information is printed.

The next five sections explain the details of these functions. You should probably try writing/testing each function as you read these details. Spend about 15-20 minutes on each function; at that point, if you are not making good progress toward a solution, move on to work on later functions (which you might find easier) and return if you have time. Each problem is graded based on the percentage of tests it passes.

A Data Structure for Voter Registration

Suppose that political parties wanted to store a database containing basic information about voters by their zipcode. We will represent this information in a **dictionary** whose keys specify a **zipcode** (**int**). Associated with each **zipcode** key is another **dictionary** whose keys specify a **party** (a political party, a **str**); associated with each **party** key is the number of registered **voters** of that **party** in the **zipcode** (**int** that is non-negative).

For example a simple/small database might be.

```
db = {1: {'d': 10, 'i': 15,          'r': 15},
      2: {'d': 12,                   'r':  8},
      3: {'d': 10, 'i': 30, 'l': 20, 'r': 22},
      4: {'d': 30,          'l': 20, 'r': 30},
      5: {         'i': 15, 'l': 15, 'r': 15}}
```

I have printed this dictionary in an easy to read form. Python prints dictionaries on one line, and can print their key/values in any order, because dictionaries are not ordered. Assume that any distinct **str** can represent a political **party**: in the example above, **'d'** could be **democrat**, **'i'** could be **independent**, **'l'** could be **libertarian**, **'r'** could be **republican** (and there might be others).

This data structure means that

1. In **zipcode 1** there are **10 voters** registered as democrats, **15 voters** registered as independents, and **15 voters** registered as republicans.
2. In **zipcode 2** there are **12 voters** registered as democrats and **8 voters** registered as republicans.
3. In **zipcode 3** there are **10 voters** registered as democrats, **30 voters** registered as independents, **20 voters** registered as libertarians, **22 voters** registered as republicans.
4. In **zipcode 4** there are **30 voters** registered as democrats, **20 voters** registered as libertarians, and **30 voters** registered republicans.
5. In **zipcode 5** there are **15 voters** registered a independents, **15 voters** registered as libertarians, and **15 voters** registered as republicans.

1: Details of party_summary:

The **party_summary** function takes a **dict** argument in the form illustrated on the middle of page 2,

2 of 5

each key in the **dict** is an **int** (**zipcode**) whose associated value is another **dict**: this inner **dict** associates a **str** key (political **party**) with an **int** (the number of registered **voters** in that **party** in that **zipcode**).

and returns a dictionary (**dict** or **defaultdict**) whose keys are political **parties**; each key is associated with the total number of registered **voters** for that **party** in the database. The parameter dictionary should not be changed.

Calling this function with **db** bound to the dictionary shown on the middle of page 2: **party_summary(db)**

returns a dictionary with the following associations:

```
{'d': 62, 'i': 60, 'r': 90, 'l': 55}
```

Python prints dictionaries in any order, because they are not ordered.

2: Details of read_db:

The **read_db** function takes an open file as an argument; it returns a dictionary (**dict** or **defaultdict**) in the form illustrated on the middle of page 2.

The input file will consist of some number of lines; each line specifies a **zipcode**, followed by one or more political **party** and count of the registered **voters** for that **party**. All these different items are separated by colons (**:**) and contain no internal spaces.

For example, the **db1.txt** file contains the lines

```
3:d:10:i:30
1:d:10:i:15
4:r:30
1:r:15
2:d:12:r:8
4:l:20
5:i:15
3:l:20:r:22
4:d:30
5:l:15:r:15
```

Note the second line **1:d:10:i:15** followed (two below) by **1:r:15**. It is legal to repeat the same **zipcode** on different lines, but each **party** for that **zipcode** will be on only one line: the first line contains the **d** and **i parties**, so those **parties** will **not** appear on any other line started by **zipcode 1**. Your function should be able to read a file with these same lines in any order and produce the correct result.

For this file, **read_db** returns a dictionary whose contents are:

```
db = {1: {'d': 10, 'i': 15,          'r': 15},
      2: {'d': 12,                   'r':  8},
      3: {'d': 10, 'i': 30, 'l': 20, 'r': 22},
      4: {'d': 30,          'l': 20, 'r': 30},
      5: {         'i': 15, 'l': 15, 'r': 15}}
```

I have printed this dictionary in an easy to read form. Python prints dictionaries on one line, and can print their key/values in any order, because dictionaries are not ordered.

Important: Remember to convert each count of registered **voters** from a **str** into an **int**.

3: Details of registered_ordering:

The **registered_ordering** function takes two arguments: (a) a **dict** in the form illustrated on the middle of page 2,

3 of 5

> each key in the **dict** is an **int** (**zipcode**) whose associated value is another **dict**: this inner **dict** associates a **str** key (political **party**) with an **int** (the number of registered **voters** in that **party** in that **zipcode**).

and (b) a **str** specifying a political **party**. This function returns a **list** of **int** (**zipcodes**). The **zipcode list** should appear in **decreasing order** based on the number of **voters** registered to that political **party** in the **zipcode**: if the number of **voters** for two political **parties** are the same, their **zipcodes** should appear in increasing order by **zipcode**. The parameter dictionary should not be changed.

Calling this function with the **db** dictionary shown on the middle of page 2: **registered_order(db, 'r')**

returns the following :

```
[4, 3, 1, 5, 2]
```

Notice that **zipcode 4** has **30** republicans, **zipcode 3** has **22** republicans, **zipcode 1** and **zipcode 5** both have **15** republicans, and **zipcode 2** has **8** republicans. So generally the **zipcodes** appear in decreasing order of the number of republican **voters**. And, because **zipcode 1** and **zipcode 5** both have **15** republicans, these **zipcodes** appear next to each other in increasing numeric **zipcode** order: **1** before **5**.

4: Details of state_summary:

The **state_summary** function takes two arguments: (a) a **dict** in the form illustrated on the middle of page 2,

> each key in the **dict** is an **int** (**zipcode**) whose associated value is another **dict**: this inner **dict** associates a **str** key (political **party**) with an **int** (the number of registered **voters** in that **party** in that **zipcode**).

and (b) a **dict** whose keys specify a **state** (**str**); associated with each **state** is a **set** of **zipcodes** in that **state**. This function returns a dictionary (**dict** or **defaultdict**); each key is a **state** associated with another dictionary (**dict** or **defaultdict**): this inner dictionary associates a **str** key (political **party**) with an **int** (the number of registered **voters** in that **party** in the entire **state**). The parameter dictionary should not be changed.

Assume we define

```
state_zips = {'CA' : {1,3}, 'WA': {2,4,5}}
```

Calling this function with the **db** dictionary shown on the middle of page 2: **state_summary(db, state_zips)**

returns a dictionary whose associations are:

```
{'CA':  {'d': 20, 'i': 45, 'r': 37, 'l': 20},
 'WA':  {'d': 42, 'r': 53, 'l': 35, 'i': 15}}
```

So, in **CA zipcodes** (**1** and **3**) there are a total of **20** registered **democrat voters**, **45** registered **independent voters**, **37** registered **republican voters**, and **20** registered **libertarian voters**. If there are no registered **voters** in a political **party**, that **party** should not appear as a key in the returned dictionary.

Python prints dictionary associations in any order, because dictionaries are not ordered.

5: Details of predicted_winners:

The **predicted_winners** function takes a **dict** argument in the form illustrated on the middle of page 2,

> each key in the **dict** is an **int** (**zipcode**) whose associated value is another **dict**: this inner **dict** associates a **str** key (political **party**) with an **int** (the number of registered **voters** in that **party** in that

4 of 5

> **zipcode**).

and returns a dictionary (**dict** or **defaultdict**); each key is a political **party** (or the string **'tie'**) associated with a **set** of **zipcodes**. A **zipcode** appears in the **set** associated with a political **party**'s key, if that political **party** has the highest number of registered **voters** in that **zipcode**; if a **zipcode** has two or more **parties** with the highest number of registered **voters**, then that **zipcode** appears in the **set** associated with the key **'tie'**. The parameter dictionary should not be changed.

Calling this function with the **db** dictionary shown on the middle of page 2: **predicted_winners(db)**

returns a dictionary whose associations are:

```
{'tie': {1, 4, 5}, 'd': {2}, 'i': {3}}
```

For example, **zipcode 2** has **12** democrats and **8** republicans, so it is put in the **set** associatied with **'d'**; **zipcode 1** has **10** democrats, **15** independents, and and **15** republicans, so it is put in the **set** associatied with **'tie'**.

Python prints dcitionary associations and **sets** in any order, because these data structurs aren't ordered.

6: Details of biggest_gap (Extra Credit: 1 point; finish all previous problems first):

IMPORTANT: Your function must pass all tests for the extra credit point.

The **biggest_gap** function takes a **dict** argument in the form illustrated on the middle of page 2,

> each key in the **dict** is an **int** (**zipcode**) whose associated value is another **dict**: this inner **dict** associates a **str** key (political **party**) with an **int** (the number of registered **voters** in that **party** in that **zipcode**).

and returns a 2-**tuple**, indicating which pair of "adjacent" **zipcodes** (with no other **zipcodes** in between; see **IMPORTANT** below) has the largest difference in number of total **voters**. The parameter dictionary should not be changed.

Calling this function with the **db** dictionary shown on the middle of page 2: **biggest_gap(db)**

returns the 2-**tuple**

```
(2, 3)
```

Why this 2-**tuple**? **zipcodes 2** and **3** are adjacent (no other **zipcodes** between them); **zipcode 2** has **20 voters** total and **zipcode 3** has **82 voters** total, so the difference is **62**. No other adjacent pairs of **zipcodes** has a bigger difference. For example **zipcode 4** has **80 voters** total, so the difference between **zipcode 3** and **zipcode 4** (which are adjacent) is **2 voters**: a smaller difference than **62**.

IMPORTANT: You may **not** assume **zipcodes** increase by one: the **zipcodes** used as keys in some problem might be **92617**, **92697**, **93201**, and **94375**. The function would look at the three adjacent **zipcode** pairs: (a) **92617** and **92697**, (b) **92697** and **93201**, and (c) **93201** and **94375**, to find which adjacent pair has the largest difference in the total number of registeed **voters** in those **zipcodes**.

5 of 5