CSCI 140 Programming for Data Science Project 2: Part 1

Due Friday September 18, 2020

Random digit dialing is a process commonly used in surveys to try and avoid bias in selecting participants (and it wouldn't be surprising if robo-calls used it too). A lot of random digit dialing is not completely random. For example, if we consider phone numbers in the United States, a subset of area codes would be selected in advance to represent a sampling area, and then random phone numbers would be generated within that area code. In this project, we are going to explore how to generate random phone numbers that are valid residential or cellular numbers, i.e., you should reach someone when you call!

Phone Numbers in the United States

Phone numbers in the United States have 10 digits, for example: 757-221-4000. The first three digits are the **area code** (757 in this example) which indicates a specific geographic area. At the time this project was written, there are 338 unique area codes used in the US for residential and cellular numbers (when we consider only states). The fourth through sixth digits are known as the **prefix** (221 in this example). Prefixes for residential and cellular numbers can be anything except 555, 958, 959 and x11, e.g. 211, 511, 811, 911 (anything with 11 as the last two digits). There are no constraints on the last 4 digits of the phone number (4000 in this example). The last 4 digits are called the **suffix**.

Assignment

You have been given a file called `generate_phone.py` that contains an import line for the random module and a list called AREA_CODES which contains all of the valid area codes in the United States. It also contains two lines that must be the start of your program.

Your task is to write a program that takes no inputs and creates and prints a valid 10 digit phone number. There are MANY ways to do this, but you are required to follow the specifications below. Read them carefully. If you choose to complete this project in some other way, you may receive no credit for your work.

Keep these instructions open and refer to them often while you are working. Write out algorithms on paper before writing code.

Part 1: Generate a Prefix

As explained above, the prefix is the part of the phone number that lies after the area code. For example, in the number, 757-683-5616, the prefix is 683.

The first part of your program will generate a random prefix that is valid. **The rules for what is valid are given in the Phone Numbers in the United States section above** (the prefix is not 555, 958, 959, or x11 (last two digits 11, e.g. 911)).

You have been given the first two lines of code for this task. These lines of code select the first digit for the prefix and the second digit. You must proceed using these two lines UNCHANGED, and the variables these lines create, **first** and **second**, to complete this task. If you choose not to use these variables, you will receive no credit for this part of the assignment.

The variables **first** and **second** will be strings of length 1. They could be for example:

first `'7'` **second** `'3'`

To complete Part 1, you need to select the third digit of the prefix in such a way that the prefix will be valid. This depends on what the first two digits are. The best way to complete this is to write out the logic on paper using the block and arrow diagrams that are in the readings and videos. Here's something to get you started:

If **first** is `'5'` and **second** is '5', what are the possible choices for the third digit of the prefix? What digits should excluded when choosing the third digit?

Once you've worked out the logic for selecting the third digit, the last task is to put the digits together to make the prefix. The prefix should be stored in a variable. In your final product, it should not be printed, but you may want to print it for debugging purposes.

Use good programming style. What do we mean? Don't have extraneous conditions. For example, you shouldn't need a separate condition for the case of 958 and 959, those can be handled in the same place.

Part 2: Generate a Suffix

As explained above, the suffix is the part of the phone number that lies after the prefix. For example, in the number, 757-683-5616, the suffix is 5616.

The second part of your program will generate a random suffix. There are no rules for suffixes, so you just need a string of 4 random digits.

There are several ways to do this, but we will use a **while** loop.

You have been given code for part 2 that does not work correctly. Debug this code.

For debugging, you may not add, delete, or move lines. The structure should stay the same. The changes for the most part are minimal in terms of what you type, but large conceptually.

Write out the logic on paper first. Here's a few things to think about: What is each variable supposed to represent (try printing each variable if you have no idea)? What type is each variable? Are the right variables being used for each operation? When should the loop stop and end? What is each line of the code you were given trying to do?

To complete Part 2, edit the code given to you to create a 4 character string that is a suffix.

Part 3: Select an Area Code

The area code is the first three digits of the phone number. You have been provided with a list of the valid area codes in the United States stored with the variable name AREA_CODES.

For Part 3, select a valid area code randomly from the given list. This should be a single line of code. Be sure to store your area code in a variable. You will need it later.

Part 4: Put it all together

Phew - this is the end! So far, you've made a valid prefix, a suffix, and selected a valid area code. They should all be stored in variables. Now it's time to put these all together to make a phone number.

Use the variables you created in the previous parts to assemble a complete ten digit phone number as a string. The area code should be followed by a hyphen/dash, and the prefix should be followed by a hyphen/dash. For example:

```
'757-221-1212'
```

Store this phone number in a variable called **phone_number**. A line of code to print **phone_number** has already been provided for you.

SUBMISSION EXPECTATIONS

`generate_phone.py` Your implementation of the program including the debugging as described above. **No additional code including input lines should be submitted. Make sure that you have the correct import lines. If your code requires manual intervention, it will affect grading.**

`Project_2_Part_1.pdf` A PDF document containing your reflections on the project. **You must also cite any sources you use. Please be aware that you can consult sources, but all code written must be your own.**

N.B.: You must use the structures we have learned in this class so far to complete this assignment. Implementations using structures we have not covered may receive no credit. Please note that the assignment is not asking you to write functions. If there are def lines in your code anywhere, that is incorrect. This also means NO LIST AND STRING METHODS.

POINT VALUES AND GRADING RUBRIC

This part of the assignment is worth 12.5 of the total 25 points for Project 2.

Part 1: 5 points

Part 2: 5 points

Part 3: 1.25 points

Part 4: 1 point

Write-up: 0.25 points

Grade Level	Execution	Correctness and Algorithms	Formatting	Style and References
A	Code executes without errors, requires no manual intervention	Code correctly implements all algorithms, passes all test cases (with possible exception of extremely rare case as applicable), and meets all required functionality	Inputs and outputs formatted as per the specification both in terms of content and type, may be minimal formatting issues (spacing, punctuation, etc.); correct file formats and names	Code uses current structures and modules discussed in class; meaningful variable names; commented as appropriate; any references cited in write-up
B	Code executes without errors, may require minimal manual intervention such as change of an import line	Meets all required functionality with exception of minor/ rare test cases (as applicable)	Inputs and outputs formatted as per the specification both in terms of content and type, some minor formatting issues may be present (e.g. incorrect string text), correct file formats and names	Code uses current structures and modules discussed in class; meaningful variable names; commented as appropriate; any references cited in write-up
C	Code does not execute without minor manual intervention such as correcting indentation or terminating parentheses or a single syntax error	Code contains logical errors and fails subset of both rare and common test cases, overall algorithmic structure reflects required functionality but implementation does not meet standards	Inputs and outputs have major formatting issues, files incorrectly named but file formats correct	Code uses current structures and modules discussed in class; meaningful variable names; commented as appropriate; references cited in write-up
D	Code does not execute without major manual intervention such as changing variable names, correcting multiple syntax errors, algorithmic changes	Code contains major logical errors and fails majority of test cases, lack of cohesive algorithmic structure, minimal functionality	Inputs and outputs have major formatting issues, files incorrectly named but file formats correct	Code uses some structures and modules from class, but otherwise largely taken from outside sources with citations; variable names and code structure hard to decipher
F	Code requires extensive manual intervention to execute	Code fails all or nearly all test cases and has no functionality to solve the proposed problem	Inputs and outputs have major formatting issues, files in incorrect formats which cannot be graded	Code uses structures and modules from outside sources only with no citations; variable names and code structure hard to decipher; code written in other programming languages or Python 2.7