CS1026: Assignment 2
Codes and Check Digits

Due: October 20th, 2021, 9pm
Weight: 8%

Learning Outcomes:
By completing this assignment, you will gain skills relating to

- Python programming constructs loops and functions
- Using lists in Python
- Creating and using Python modules
- Algorithm development and testing; designing test cases
- Following program specifications.

Background:

Many products that we use have an identification number that may or may not have a bar-code. Some examples are books, electronics, grocery items, credit cards, money orders, driver's license, etc. The identification number helps encode the information about the product. These numbers are usually separated by a space or a hyphen and each part holds specific information about the product; UPC codes (Universal Product Codes) are examples of such codes that are formed as bar codes and used for identifying merchandise.

A check digit is added to the identification number (usually the last digit). This digit is used to verify the identification number for its legitimacy. Check digit is added to the number to detect any errors made while typing the number into the system or when reading the number electronically. The check digit is calculated with an algorithm. The following describes some common identification codes and algorithms used for identification numbers and check digits.

Basic Code. Check digit algorithms are typically based on doing a computation on the digits in the identification number and then computing a check digit which is added to the identification number. A simple code is the *Basic Code* and an algorithm for doing this is as follows:

- Sum the individual digits in the base identification number.
- Compute the result of the sum modulo 10; this produces a number between 0 and 9.
- Add the resulting digit to the base identification number to get the completed identification number.

For example, if the base identification number is 2452834, the sum is 28; 28 modulo 10 is 8. The complete identification number is then 24528348. To check that a number is a valid identification number, repeat the process excluding the last digit – the result should be the same. Assume that the identification number is 1223344558. Compute the sum of

122334455, which is 29; modulo 10 results in 9. Since 9 ≠ 8, 1223344558 is **not** a valid identification number.

Positional Code. A similar approach to the *Basic Code* is the *Positional Code* which makes use of the position of the digit in the identification number as part of the computation. A simple Positional Algorithm is as follows:

- Compute the sum of multiplying each digit by its position in the base identification number.
- Compute the result of the sum modulo 10.
- Add the resulting digit to the base identification number to get the completed identification number.

For example, if the base identification number is 2452834, then the sum = 1x2 + 2x4 + 3x5 + 4x2 + 5x8 + 6x3 + 7x4 = 2+8+15+8+40+18+28 = 119; 119 modulo 10 is 9. The complete identification number is then 24528349. To check, one uses the same process on the identification number without the check digit – the result should be the same as the check digit.

Universal Product Code (UPC): The UPC is a barcode symbol and is used to track trade items in stores. The most common form of UPC is the UPC-A which consists of 12 digits which is unique for a trade item. The general UPC algorithm is as follows:

- Compute the sum of multiplying each ***odd*** digit by 3 and each **even** digit by 1 in the base identification number.
- Compute the result of the sum modulo 10; again, this results in a number between 0 and 9.
- If the resulting digit **is between 1 and 9,** then subtract the resulting digit from 10 to get the check digit and add that to the base identification number to get the completed identification number. If the resulting digit **is 0, then add 0** to the base identification number to get the completed identification number.

For example, if the base identification number is 2452834, then the sum = 3x2 + 1x4 + 3x5 + 1x2 + 3x8 + 1x3 + 3x4 = 6+4+15+2+24+3+12 = 66; 66 modulo 10 is 6 and the check digit is 4 = 10-6. The complete identification number is then 24528344. If the base identification number is 2452874, then the sum = 3x2 + 1x4 + 3x5 + 1x2 + 3x8 + 1x7 + 3x4 = 6+4+15+2+24+7+12 = 70; 70 modulo 10 is 0, so the check digit is just 0. The complete identification number is then 24528370. To check, one uses the same process on the identification number without the check digit – the result should be the same as the check digit.

Tasks:

In this assignment, you will write a **complete** program in Python that will repeatedly ask the user for a code (integer) and determine whether it is a valid *Basic Code*, a *Positional Code* or a *UPC*

Code. It is possible that some integers may be valid for more than one of these codes; even all three!

Your program should consist of two Python files: **Assign2.py** which handles all the input and output and **code_check.py**. **Assign2.py** should repeatedly prompt the user for input, a string of digits, and then call functions in **code_check.py** to determine whether the digits in the string correspond to one of the above codes, *Basic*, *Position*, *UPC*. The program should have a list for each type of code and when it finds that the string is a certain type of code, it should add the string to the list for that type of code. If a string is not one of the three types of codes, it should add it to a separate list. The program should repeatedly prompt the user for a string until the user enters a string that is zero, i.e., "0". The program should then output each list, as described below, when the user has finished.

The file **code_check.py** should consist of three functions: one function for each type of code. Each function should **take a string as a parameter**, such as "2452834" or "033367258", and determine whether it is valid code; the function should return True or False. For example, the function that checks whether an identification number is a UPC code should take "2452834" as input and return True.

Your program will make use of expressions, decisions, input/output, loops, lists and functions in Python. You should name your program **Assign2.py**. **Your program should strictly adhere to the Functional Specifications (below) and Non-Functional Specifications**.

Functional Specifications:

1. Your program will repeatedly assess codes entered by the user. It will prompt the user for a string of digits and then determine if the digits represent one or more of the *Basic*, *Position* or *UPC* codes. The program should continue prompting until the user enters "0" and then output a list of each type of code, or those strings which were not any of the code types.

2. Your program should maintain a list for each type of code and a list for strings that are none of the codes. The output **MUST** begin with the keywork "Summary" (see below). This should be followed by each type of code on a separate line, with the name of the type of code followed by the codes of that type separated by commas. If a list is empty, then it should output the type of code and "None". See the examples of program execution following. An example of the output is:

```
Summary
Basic :434135, 434191, 00663341272
Position :434191
UPC :80663341344
None :434134
```

3. Assumptions. You may assume the following when writing your program:
   a. The input is a single line prompting the user to enter a string of digits.
   b. You may assume that the strings entered consist of just digits, e.g., 2452834 or 033367258. Notice that leading zeros are allowed and need to be accounted for.

4. Finally, an automated testing program will run a number of test cases against your program. Some examples of output and test cases are presented below; **these are NOT comprehensive - you should create your own test cases and thoroughly test your program.**

Example executions and output:

# Example 1

```
Please enter code (digits only) (enter 0 to quit) 434135
-- code: 434135 valid Basic code.
Please enter code (digits only) (enter 0 to quit) 434134
-- code: 434134 not Basic, Position or UPC code.
Please enter code (digits only) (enter 0 to quit) 434191
-- code: 434191 valid Basic code.
-- code: 434191 valid Position code.
Please enter code (digits only) (enter 0 to quit) 00663341272
-- code: 00663341272 valid Basic code.
Please enter code (digits only) (enter 0 to quit) 80663341344
-- code: 80663341344 valid UPC code.
Please enter code (digits only) (enter 0 to quit) 0

Summary
Basic : 434135, 434191, 00663341272
Position : 434191
UPC : 80663341344
None : 434134
```

# Example 2

```
Please enter code (digits only) (enter 0 to quit) 12345678
-- code: 12345678 valid Basic code.
Please enter code (digits only) (enter 0 to quit) 123456789
-- code: 123456789 not Basic, Position or UPC code.
Please enter code (digits only) (enter 0 to quit) 0

Summary
Basic : 12345678
Position : None
UPC : None
None : 123456789

Process finished with exit code 0
```

Non-Functional Specifications:

1. The program should strictly adhere to the input and output requirements, **particularly the input and the format of the output lists; make sure that your output begins with the word "Summary"**.

2. The program should include brief comments in your code identifying yourself, describing the program, and describing key portions of the code.

3. Assignments are to be done individually and **must be your own work**. Software may be used to detect academic dishonesty (cheating).

4. Use Python coding conventions and good programming techniques. For example:
   a. Meaningful variable names
   b. Conventions for naming variables and constants
   c. Use of constants where appropriate
   d. Readability, indentation, and consistency

5. The name of the files you will submit must be your **Assign2.py** and **code_check.py**. ***Make sure you attach BOTH of your Python source files to your assignment submission; do not put the code inline in the textbox. Asssignment submission can be found under*** **Assignment 2** ***in the*** **Assignments Tab** ***in the OWL course site.***

   Make sure that you develop your code with Python 3.9 as the interpreter and that you have executed it with PyCharm EDU 2021.1; failure to do so may result in the testing program failing (see below).

Marking of the Assignment:

1. **Your program will be executed by an automated testing program.** This testing program assumes that:
   a. The program name is **Assign2.py**; your functions should be in a file **code_check.py**.
   b. That you are using Python 3.9 and that it executes in PyCharm Edu.
   c. That you have submitted it via OWL by uploading it.
   d. That you have adhered to the input/output specifications.

   Failure to adhere to these constraints will likely cause the testing program to fail. This may require a remarking of your program which will include a 10% penalty.

2. Functional specifications:
   a. Does the program behave according to the specifications found in the assignment document?
   b. Does the program handle input and terminate correctly?

   c. Is the output according to specifications?

3. Non-functional specifications as described above.