# A7: UPC Bar Codes

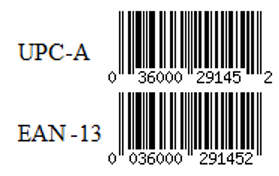
* Assignment A7 should be complete individually or with one partner.
* If you work with a partner, be sure to follow good pair programming practices.
* Be sure to read the entire prompt and understand the problem before beginning coding.
* You have two weeks to work on this assignment. However, it is large, and you should not wait until the Tuesday before it’s due to start on it. You won’t finish it in time.

## Learning Objectives

* Learn about an important application of computer science, the UPC code
* Work on the design of a larger problem
* Use lists in an application problem

## UPC CodeUPC code Kleenex

The Universal Product Code (UPC-A) is a bar code system that was first used in 1974 in Ohio designed to automate checkout at the grocery store. It has since then been much more widely adopted, and now it is seen on packages in lot of different stores in the United States. You’ve likely scanned these things yourself at the self-check lines at Wal-Mart. (The UPC code you see shown to the right is for a box of tissues made by a US company called Kleenex.)

The UPC code has now expanded into the EAN-13 (European Article Number) to be able to be used around the world. As an expansion of UPC, an agreement was made that all scanners which could scan EAN-13 codes would also be able to scan UPC codes. Thus, the UPC-A codes in the US did not need to change.

Compare the UPC-A code for that box of Kleenex to the EAN-13 code -- the bars are all identical.

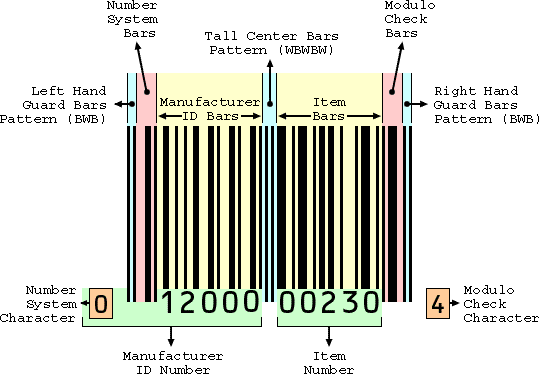
There are some significant differences between the UPC-A and the EAN-13 codes though:

1. The UPC-A has 12 human-readable digits, but the EAN code has an extra digit at the very beginning, making it 13 digits. For products made in the US, this extra digit is always a 0, indicated by the zero that starts the number in the EAN-13 code of the example.
2. The placement of the human readable digits beneath the code also differs. Note in the example to the right, the entire EAN-13 for a box of Kleenex tissues (036000291452) is contained under the bars, with the exception of the leading zero.

## The UPC-A Code Decoded

One of the most interesting sites on UPC codes is by an artist named Scott Blake, who is really into bar codes. You have got to see it to believe it. Really. Check it out: <http://www.barcodeart.com/artwork/index.html>

In addition to the art he created with bar codes, he created a diagram decoding the UPC-A Code for a 2 liter bottle of Pepsi, that is very informative and explains to some extent what each of the parts means.



There are a lot of parts to this picture, but you will be working on two specifics, the "Modulo Check Character" and how the bars are constructed.

## Modulo Check Character

In the UPC-A system, the modulo check character is a digit that is used to verify that the bar code has scanned correctly. This number is calculated as follows:

1. Add the digits in the odd-numbered positions, starting with the "Number System Character", the number in the bottom left of the picture above.
2. Because there are 12 digits in a UPC label, the sum includes the first, third, fifth, seventh, ninth and eleventh digits from left to right.
3. Multiply the resulting by three.
4. Add the digits in the even-numbered positions, starting again from the left to the right. We exclude the "Modulo Check Character", which means that the sum only has the second, fourth, sixth, eighth and tenth digits.
5. Add these two result together and find the remainder when divided by 10. (i.e we can use the modulo operator)
6. If the result is not zero, subtract the result from ten to yield your check digit; otherwise your check digit is zero.

### Modulo Check Examples



Consider the Kleenex tissue UPC-A we saw above (the image is shown to the right). The first 11 digits of code are as follows, where we enlarged and colored the digits that are in even positions for clarity:

0 3 6 0 0 0 2 9 1 4 5

Calculating the Module Check goes as follows:

1. Add the odd-numbered digits: 0 + 6 + 0 + 2 + 1 + 5 = 14 and multiply the sum by three: 14 × 3 = 42
2. Add the even-numbered digits: 3 + 0 + 0 + 9 + 4 = 16
3. Add these two results: 42 + 16 = 58, and calculate the remainder when divided by 10:   
   58 % 10 = 8
4. 8 is not 0, so subtract eight from ten: 10 − 8 = 2. According to this process, the check digit is thus 2, which is exactly the last number of the UPC code (the modulo check character)!



We can also see how the modulo character will work for the UPC-A bar code for the 2-liter bottle of Pepsi that Scott Blake used in his diagram. The first eleven digits for the code is:

0 1 2 0 0 0 0 0 2 3 0

1. Add the odd-numbered digits: 0+2+0+0+2+0 = 4 and multiply the sum by three: 4 × 3 = 12
2. Add the even-numbered digits: 1+0+0+0+3 = 4
3. Add these two results: 12 + 4 = 16, and find the remainder when divided by 10:   
   16 % 10 = 6
4. 6 is not equal to 0, so subtract six from ten: 10 − 6 = 4. According to this process, the check digit should be 4, which it is!

Again, what is the purpose of this check character? As the name suggests, using this process is designed to detect errors and to validate that the UPC code is in fact correct. So, when you are checking out at the grocery store and pass the bar code over the scanner, the computer reads the code and compares the check digit in the UPC with what it should be. If there is a mismatch, a scan error is generated.

The UPC code can detect 100% of single bit read errors (one value is incorrect) and 89% of transposition errors (such as when two numbers get switched by mistake, 21 -> 12). Cool, huh?

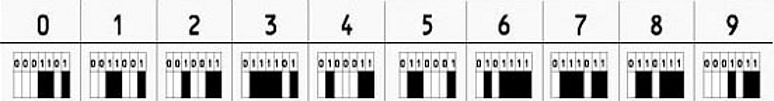
## What do those bars mean?

The bar codes are basically binary numbers, with black representing a 1 and white representing a 0.

You may have noticed that the middle and ends of a UPC always has a bunch of longer bars. Although we will not worry about their lengths, we do need to note that these bars are special. The end bars (called "guard bars") always consist of the bit pattern 101, and the center bars have the bit pattern 01010 (note that both of these patterns are symmetric.)

The UPC bar code is divided into two main areas--the part to the left of center and the part to the right of center.

* **Number System Character + Manufacturer ID Number (to the left of center):**  
    
  The digits between the left hand guard bars and the center bars have the following binary pattern, where the human readable number is on top and the bars below. Note that multiple consecutive bars of the same color have no dividing markers, so the collection will appear "thicker."Also note that white bars encode data (binary 0's) just like the black bar (binary 1's).



* **Item Number + Modulo Check Character (to the right of center):**The pattern for each digit between the center and right hand guard bars is as follows:



Another way to represent this information is in a look-up table:

|  |  |  |
| --- | --- | --- |
| Digit | "Left Side" | "Right Side" |
| 0 | 0001101 | 1110010 |
| 1 | 0011001 | 1100110 |
| 2 | 0010011 | 1101100 |
| 3 | 0111101 | 1000010 |
| 4 | 0100011 | 1011100 |
| 5 | 0110001 | 1001110 |
| 6 | 0101111 | 1010000 |
| 7 | 0111011 | 1000100 |
| 8 | 0110111 | 1001000 |
| 9 | 0001011 | 1110100 |

Why are there separate codes for the left and right sides of the center? Bar codes are frequently read upside down, which basically means backwards, so the patterns are designed so that the scanner can identify which bars it is reading. Thus, the computer can determine that which patterns are the ones for the manufacturer code because they always begin with a 0 and end with a 1. For the patterns on the item number, the opposite is true. Note that these patterns are not mirror reflections of each other; if you read a pattern on the left backwards, there is never a match on the right!

Hint: There are lots of good ways to structure a look-up table in Python. You can use a list of tuples, multiple lists, or even the dictionary like we used in the last homework.

## Using Files and Strings

Here is a Python program which takes a string and reverses the order of each word. It demonstrates some more string methods, but it also shows you how to manipulate lists (which work a lot like strings!). The code is mostly for fun, but also to demonstrate some tools that’ll help you on this assignment:

* [a7\_reverse\_list.py](https://drive.google.com/file/d/0B0J8Yj0B6KRSVExrSkRscWJJcEE/view?usp=sharing)

## Your Tasks

Your task is to create a program which can verify and generate bar codes.

Minimally, your program should do the following:

1. Your program should ask the user for a 12 digit number, representing a bar code.
2. For every user input, your program should verify the input is a valid number (12 digits, all numbers between 0 and 9). Your program should continue to ask the user for an input until a valid one is given (do not simply end the program if the input is invalid).
3. Your program should then check to see if the 12 digits form a valid UPC code using the Modulo Check Character. Your program should be able to do this check, but if you want to double check your work, just to be certain, you may use <http://www.upcdatabase.com/itemform.asp>
4. If the code is valid, the program should display the UPC code on the screen using the turtle library.
5. If the code is invalid, it should print an error message using the turtle library.
6. Note that each of the lines in a UPC code is either black or white. The right edge of each line exactly meets the left edge of the next line, and each is the same width. Some lines look thicker when lines of the same color follow each other. Note also that guard lines and center lines tend to be longer. How much longer is not important.

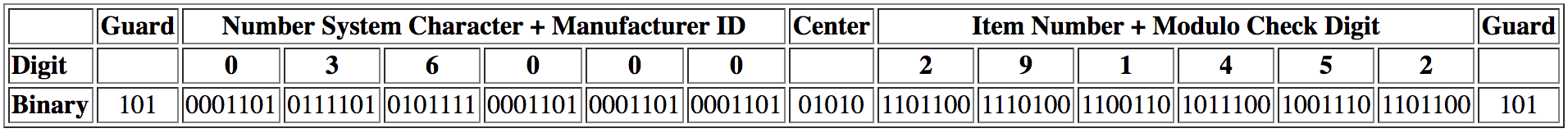
Some pointers worth repeating:

* ***The most important part of this homework is the design.*** If you think about design first, debugging will be easier. For this first part, I highly encourage you to create a [flowchart](https://commons.wikimedia.org/wiki/Flow_chart) that outlines how your program is going to work.
* The second most important part of this homework is to break up the program into smaller functions that you implement one by one, using ***unit tests*** to check its operation on known inputs and outputs.
* The following is a couple of suggestions for the kinds of functions you may want to use, but these functions are ONLY SUGGESTIONS! You are welcome and encouraged to design the program that makes sense to you! Regardless, however, you should still unit testing for anything you design.
  1. Suppose you created a function called convert() that takes in as a parameter a string of 12 digits and it returns a list with each digit of a UPC-A bar code. (Storing the numbers in an list will greatly facilitate computing the check digit.)  
       
     Taking the Kleenex UPC-A bar shown to the right as an example, the input would be the string "036000291452" and the output would be [0, 3, 6, 0, 0, 0, 2, 9, 1, 4, 5, 2]. Thus, you can use the unit test function testit(...) from the text to test a that it works correctly as:

testit(convert("036000291452") == [0, 3, 6, 0, 0, 0, 2, 9, 1, 4, 5, 2])

* 1. Suppose you also created a function called something like check\_code() that checks whether a sequence of digits passed in as input is a valid UPC-A sequence. It can return True if correct and False otherwise.

If you decided to have this function take a string digit sequence as input, you can use the function testit() from the text to test it for both valid and invalid sequences:  
  
testit( check\_code( "036000291452" ) == True)   
testit( check\_code( "036000291455" ) == False )   
# notice that all that changed was the check modulo character

* 1. Suppose that you know that a sequence is a valid UPC-A code. You can then have a function that translates the decimal digits into a sequence of binary numbers that represent the black (1s) and white (0s) bars of the bar code.  
       
     Your translate() function could take as input a single digit, and generate the binary representation of the code using the lookup table scheme outlined above:
  2. Once the function works correctly, test it using test cases!
  3. Latly, you can program the graphical display of the UPC code using the turtle library.

## Additional Guidelines and Best Practices

* Your program must have good structure and style:
  1. It must include a main() function.
  2. The highest level of the program (i.e., no indenting) must only contain the following:
     + the header
     + any import statements
     + function definitions, including def main():
     + a call to the main() function
  3. It must correctly use lists and tuples (you’ll see these officially in Friday’s quiz reading)
  4. It must be designed in a modular fashion, correctly using functions for each task with correct parameter passing and appropriate use of returns.
  5. Use only meaningful variable and function names.
  6. Insert a descriptive docstring for each function you are designing and implementing.
  7. Include a descriptive header as a comment at the top of your source code.

## 

## Submission Instructions

1. Review the requirements above to ensure you completed everything that was expected of you.
2. Save your code as **A7\_upc\_*username*.py**. Replace *username* with your Berea usernames. For example, the TA Bianca Marrero’s file would be **A7\_upc\_marrerob.py.**
3. Upload the file to Moodle by the due date listed on the course website: <https://trello.com/b/w7bIrLoV/>.
4. If you worked with a partner, your partner should upload a file named **A7\_upc\_*usernames*.txt** and include both partner’s name in the document.