A09: Error Detection

- Assignment A09 should be completed 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 should seek help completing this assignment from the #csc226 Slack channel.

Learning Objectives

- More practice breaking a larger problem down into smaller pieces using functions
- Gain practice using strings
- Introduce concepts about binary and parity

How to Start

- To begin, make a copy of this document by going to File >> Make a Copy...
- Share the copied document with all members of your team, if you are working with someone. You can share this document by hitting the blue button in the top right of the document, then entering the email address of all members in the bottom input field.
- Change the file name of this document to ***username1*****,** ***username2*** **- A09: Error Detection** (for example, **pearcej, shepherdp - A09: Error Detection**). To do this, click the label in the top left corner of your browser.
- Next, go to the **Github classroom for A09: Error Detection**.
- You will be asked to create a team:
  - One of you will create the team.
  - Then, the other will join the team created by the first partner.
  - If you are working alone, create a team anyways, with just you in it.
- Paste the link to your Github repo here:

Github Repo Link:	

- After you've opened the repository in PyCharm, start by creating a new branch.

Binary and Error Detection

Most modern human cultures use the decimal, or base 10, number system. Computers use a different system called **binary** for more than just processing numbers. Binary is a base 2 number system in which values are represented by different combinations of 0 and 1, also known as OFF or ON. It is the primary "alphabet" of all digital information (text, music, photos, videos, etc).

Binary may seem weird to us, but it is not new to humans. Morse code also uses two digits (long and short) to represent the entire alphabet. In addition, Australia's aboriginal peoples counted by two, and numerous tribes of the African bush sent complex messages using drum signals at high and low pitches. Even smoke signals are a form of binary communication.

When data is transmitted, it is subject to noise which may introduce errors into the data. Error detection techniques such as the **parity bit** allow detecting such errors with the addition of a single bit. There are two types of parity (even and odd), but we only need one of the them, so we will choose to use even parity.

An **even parity bit** can be added to the end of a string of binary code as a simple form of error detection. The number of 1 bits in the bit string are counted. If that total is odd, the parity bit value is set to 1, making the total count of 1's in the set an even number. If the count of 1's in the bit string is already even, the parity bit's value is 0.

Here is a video which explains this notion in a fun way. Think of the dark sides of the cards as 1 and the light sides as 0.

What the magician did was to add a parity row and column which allowed him not only to detect, but to correct the "error." Correction was possible because he added along both axes. Cool, huh? (Don’t tell I told the secret)

In serial data transmission, a common format utilizes chunks which consist of 7 data bits and an even parity bit. This was chosen because the American Standard Code for Information Interchange (ASCII), a character-encoding scheme based on the English alphabet, encodes 128 of the most important alphabetic, numeric, and other characters into 7-bit binary integers.

7-Bit ASCII Character Codes

	---0000	---0001	---0010	---0011	---0100	---0101	---0110	---0111	---1000	---1001	---1010	---1011	---1100	---1101	---1110	---1111
000----	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
001----	DLE	DCL	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
010----	space	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
011----	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
100----	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
101----	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
110----	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
111----	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

For example: Using the chart, one can see that the character 'J' is 1001010 (100+1010) in ASCII, and the character 'S' is 1010011 (101+0011).

In modem-type communications, a parity bit is added to the beginning of the 7-bit ASCII character, making it 8 bits, or 1 byte, long.

So, when even parity is used and a parity bit is added to the front, the letter 'J' would be encoded as 11001010 (1+100+1010) because without the extra one, the sum of the bits is odd (3).

When even parity is used, 'S' becomes 01010011 (0+101+0011) because summing the bits gives 1+0+1+0+0+1+1 is already even, so 0 is used for the parity bit.

Test Driven Development (TDD)

Test-driven development (TDD) (also called test-driven design) is a method of software development in which unit tests are written prior to the actual code. Functions are written as "stubs." The main idea is to get something working and keep making it better until all the unit tests are satisfied. The process is iterated as many times as necessary until each function (or unit) is functioning according to the desired specifications.

You should use test-driven development in this assignment.

Your Task

In this assignment, we will take a string of 0's and 1's and do the following:

- Ensure the string consists entirely of 0's and 1's (i.e., binary only).
- Check the string to see if its **length** is divisible by 7. (It should be because it should consist of 7-bit ASCII characters.)
- If the string passes both of the above tests, break it into chunks of 7. (These should each be an ASCII character.)
- For each chunk, prepend (i.e. concatenate at the front end) an even parity bit (Prepend a 1 if the sum of the 7 bits is odd, and prepend a 0 otherwise). This will make each chunk an 8-bit chunk, typically called a **byte**.

Hint: Included in your repository is some example code, **peel_digits.py**, that illustrates appending to a list which you will find very useful in this assignment.

Geek note: Because we are using strings to represent bits, and each character in a string uses 7 bits, we are actually using 7 times more space than we need to. In other words, these same ideas can be done with bits rather than bytes. The ideas are all the same.

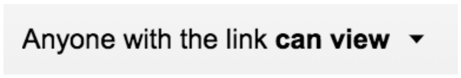
Requirements:

- Create each of the following functions:
    - `unittest()` - use the same function from previous assignments. **No modifications are needed to this function**; it's only used to make the test suite more usable.
    - `parity_test_suite()` - tests each fruitful function below. Be thorough in your testing; test expected behavior, as well as unexpected behavior (e.g., a non-binary string as input). I highly suggest you write the test suite before you write all of the function's functionality. Instead, use the stubs provided in the code, write the test suite, then write each function.
    - `main()` - the main function, which either calls the test suite (which tests each function individually) or the code that solves the problem above (i.e., normal mode).
    - `is_binary()` - takes a bit string as input, returning `True` if it is at least 7-bits and consists solely of 0s and 1s, and returning `False` otherwise.
    - `is_div_by_sevens()` – takes a bit string as input, returning `True` if the length of the string is evenly divisible by 7, and returning `False` otherwise.
    - `split_into_sevens()` – takes a bit string whose length is evenly divisible by 7 as input, and returns the resultant list of **7-bit long strings** as output.
- Be sure to include good documentation, including good comments, docstring for each function, and the standard header at the top of your program.

For the curious (not for credit): Instead of using strings, use lists to achieve the same results.

Submission Instructions

1. Review the requirements above to ensure you have completed everything that was required of you.
2. Edit the **README.md** file in your repository. Replace the lines with the correct information for your name(s), the link the repository, and the link to this document, which you can get via the blue Share button SHARE in the top right of this window.
3. Check the **Share** settings for this document (top right). Set them to **"Anyone with the link can view"**. That is how we will be able to access this document to grade you:



Anyone with the link **can view**

4. **Merge** all branches to the master branch:
    a. On your branch, **Add** and **Commit**
        i. Right click any new files (they'll be green) in the Project pane of PyCharm (far left), and click **Git** >> **Add.**
        ii. Right click the project directory in the Project pane of PyCharm, and click **Commit Directory**...:
        iii. Add a meaningful commit message, and click "**Commit**":
    b. **Pull** and **Merge** master into your branch:
        i. Go to Git >> Pull
        ii. Check the box to pull origin/master (which will also merge it into your branch)
        iii. Resolve any merge conflicts using the "Merge..." button, if there are any.
    c. **Push** your branch to GitHub.
    d. Issue a **pull request** in GitHub. If you worked with a partner, review each other's code to ensure no conflicts were created.
5. Accept the pull request. If you worked with a partner, be sure to do the above steps for each partner, and not at the same time (one partner does all steps before the other starts).
6. **Check your repository in GitHub to ensure everything was submitted to the `master` branch.** You can view the updated repository at the link you pasted in the "How to Start" section.