Assignment 2

Objectives: Practice using while loops to run algorithms and to encode sounds in a WAV file.

Note: Include DocStrings in each script you submit analogous to the following:

```
"""This script asks for two numbers and prints the greatest common divisor


Submitted by Mauricio Arias, NetID ma6918
[Include a short explanation here of how the script does that.]
"""
```

Part 1. The Greatest Common Divisor. Write a program that asks for two numbers and calculates the greatest common divisor using the Euclidean algorithm. Use the following example taken from Wikipedia to guide your coding:

```
Calculate the gcd for 105 and 28:

105 % 28 = 21

28 % 21 = 7

21 % 7 = 0 → process is over.

The last divisor (or previous remainder) is the answer.

GCD(105, 28) = 7
```

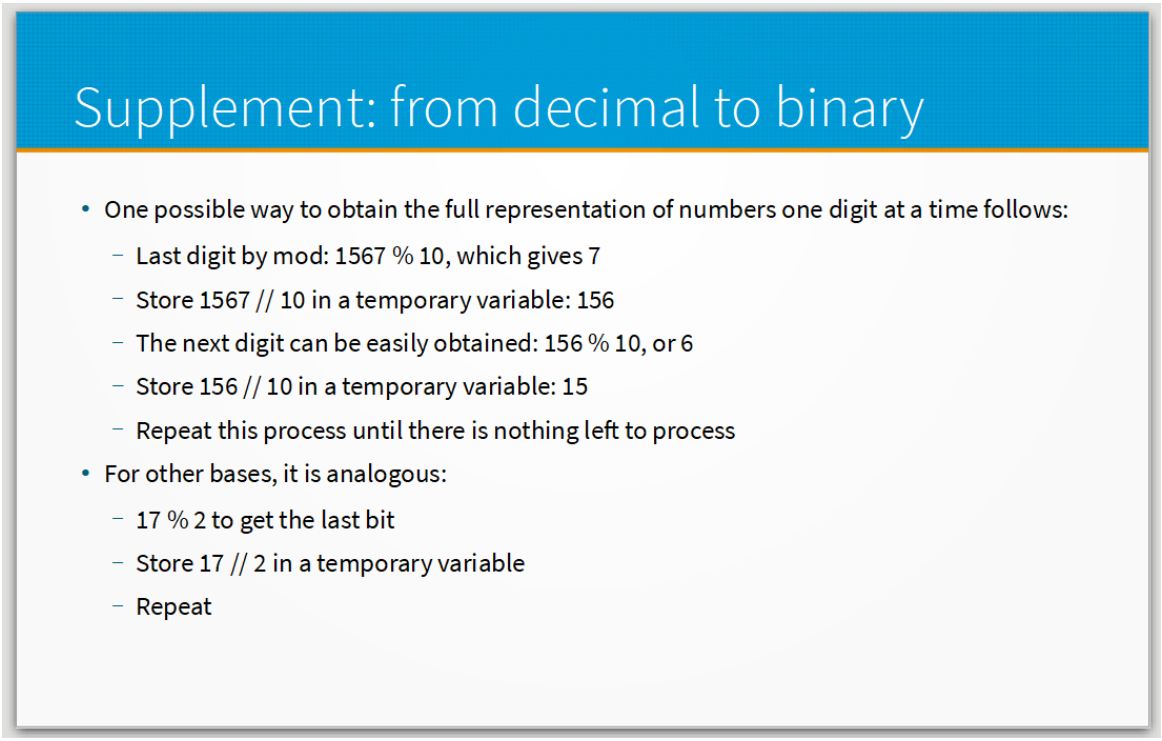
Notice that we don't use the dividend after each step is done. This allows us to reuse the variables as we move through the steps. Do that in your code. Test your program at least as shown in the example below. Submit your script as `[NetID]_gcd.py. (2.5 pts)`

```
C:\Users\Mauricio\Documents\tests>python gcd.py
What is the first number? 156
What is the second number? 39
gcd(156, 39) = 39

C:\Users\Mauricio\Documents\tests>python gcd.py
What is the first number? 12827291
What is the second number? 15485807
gcd(12827291, 15485807) = 1

C:\Users\Mauricio\Documents\tests>python gcd.py
What is the first number? 16
What is the second number? 24
gcd(16, 24) = 8
```

Part 2. Obtaining number representations in different bases. Write a program that asks for a natural number and a base and prints a representation of the number in the base provided: use a string for this purpose. (This script assumes that the natural number is given in base 10 and that the base is an integer.)

We will limit the base to the set {2, 3, …, 9} to avoid the need to create new symbols. Check this condition using **chained comparisons** and if the base provided is not in that set indicate that fact to the user and request the base again. For this step use a "`while True:`" loop and exit out of it when the condition is satisfied.

Use the algorithm we discussed in class for obtaining the string to report. See an example of the output in the next page. Submit your script as `[NetID]_base_change.py. (2.5 pts)`




```
What number are you interested in converting? 134
What base shall I use? 11
The base has to be between 2 and 9. Please try again.

What base shall I use? 10
The base has to be between 2 and 9. Please try again.

What base shall I use? 1
The base has to be between 2 and 9. Please try again.

What base shall I use? 0
The base has to be between 2 and 9. Please try again.

What base shall I use? 8
The number 134 is represented by 206 in base 8.
```

Part 3. Making an audio file. For the following tasks we will make WAV files that include simple sounds. The theory of sound is extensive and we won't discuss it in detail here. However, it is grounded in understanding sounds as distortions in air pressure that follow sinusoidal wave behavior. Appropriately chosen sinusoidal waves can be combined to make any repetitive sound. To deal with non-repetitive sounds a more complex extension of this theory is available.

One important aspect of computers is that they work in discrete time. Their time is in general not continuous but instead has moments when things change followed by periods of time when they don't: let's call the moments when things change t_0, t_1, t_2 and so on; t_i in general. Therefore, to mimic a continuously changing sound a computer will excite the speaker with a specific intensity at t_0 and then wait until t_1 to excite the speaker again with a different intensity. The process is repeated for t_2 and the following time points. This happens extremely fast. For reproducing a sound with CD quality, the speakers are excited 44100 times every second! Therefore to reproduce a sound we will take "snapshots" of its intensity at many time points. This process is called Pulse Coded Modulation or PCM. We will only take 8000 snapshots of the target intensity of the sound per second though. (See template .py file attached for details.) These snapshots will be added to a bytearray that already contains the appropriate header and it in turn will saved in a WAV file.

A template script is provided with 2 `TODO` labels. Additionally a `WAV_utilities.py` file is provided. Make a "lib" folder/directory for it in your working folder/directory.

Task 3.1. For this task we will produce a sound that mimics two alternating pure tones (sinusoidal waves). Submit your script as `[NetID]_alternating_tones.py`. Submit also your wav file as `[NetID]_alternating_tones.wav`. (2.5 pts)

a) Using the template .py file attached, add code to ask how many seconds of sounds should be generated: First TODO section in the file. Keep asking until the user provides a number between 1 and 20 seconds, inclusive.

b) Define two variables to keep track of all the necessary data in the second TODO section: a time variable that keeps track of the time for which we are generating the intensity and a variable that keeps track of the sample number for which we are generating this intensity. They are both related. Remember that every second we generate however many samples are specified by the parameter `samples_per_s`. Use a while loop for this, updating the sample number and the time at the end of each cycle.

c) Use the ternary operator to choose between freq1 and freq2 as the frequency of the sound wave. Write code so that it alternates between these frequencies every .25 seconds in the timeline of the generated sounds: see time variable above.

d) Generate the intensity for each sample by using the formula

$$\text{sample\_intensity} = \text{max\_amplitude} / 10 * \sin(2\pi f t)$$

where t is the time point for which we are generating the sample and f is the frequency of the sound wave: see frequency above. The math module defines pi and sin(). The utilities library defines max_amplitude as the maximum amplitude that can be encoded.

e) Encode this sample into the available bytes for each sample using the function `bytes_pack_signed()`, which takes the value to be encoded and the space available for that: `sample_depth//8` in this case. It returns a bytearray with the appropriately encoded data. This function is provided in the utilities library.

f) Concatenate that result to the bytearray `sound_binary`.

g) Repeat until all the necessary samples have been generated.

Task 3.2. For this task we will make a variation of the previous script to generate a stereo sound that has two channels: left and right. The information for these channels is added to the data stream by writing the encoded intensity for the left channel as was done before, immediately followed by the encoded intensity for the right channel. These two contiguous additions are done for each and every sample/time point.

The same alternating sound as before will be used but the intensity will change with time. The idea is to make the sound move from one channel to the other. (Earphones might be needed to appreciate this effect. Please test the sound without earphones first to avoid exposure to loud sounds.) **Make sure you change the channels parameter to 2**. The total duration will be divided into three equal periods:

Period 1. For the left channel, the sample intensity (see part d in previous task) will be multiplied by an additional factor that goes from 0 at the beginning of the period to 1 at the end: see Figure 1 below for a detailed description. For the right channel, this factor will be 0.

Period 2. For the left channel, that factor goes from 1 at the beginning to 0 at the end. For the right channel, the additional factor goes from 0 at the beginning of the period to 1 at the end: see Figure 2 below for a detailed description.

Period 3. For the left channel, the factor stays at 0. For the right channel, the additional factor goes from 1 at the beginning of the period to 0 at the end.

Submit your script as `[NetID]_moving_tones.py`. Submit also your wav file as `[NetID]_moving_tones.wav`. (2.5 pts)

The figures below exemplify the multiplicative factors to be added to the formulae derived from the one in part d of Task 3.1. Notice that the total duration of this particular sound is 12 seconds and 3 periods of 4 seconds each are shown for each channel. The total length of the sound will be determined by what the user specifies, the length of each period will vary accordingly.

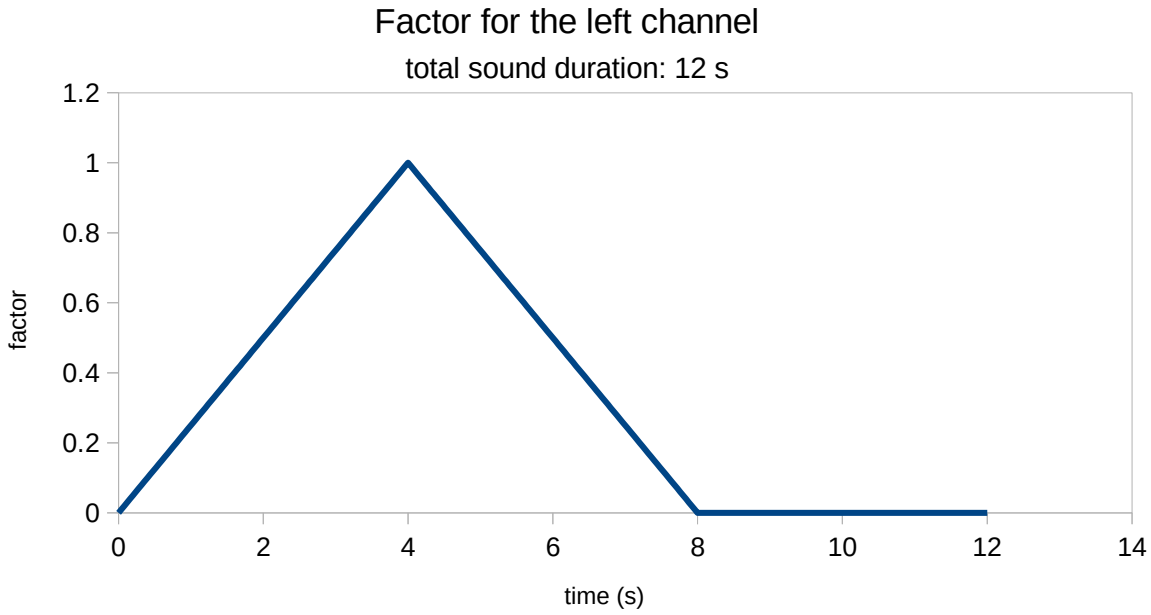


Figure 1. Multiplicative factor for the left channel as time progresses.

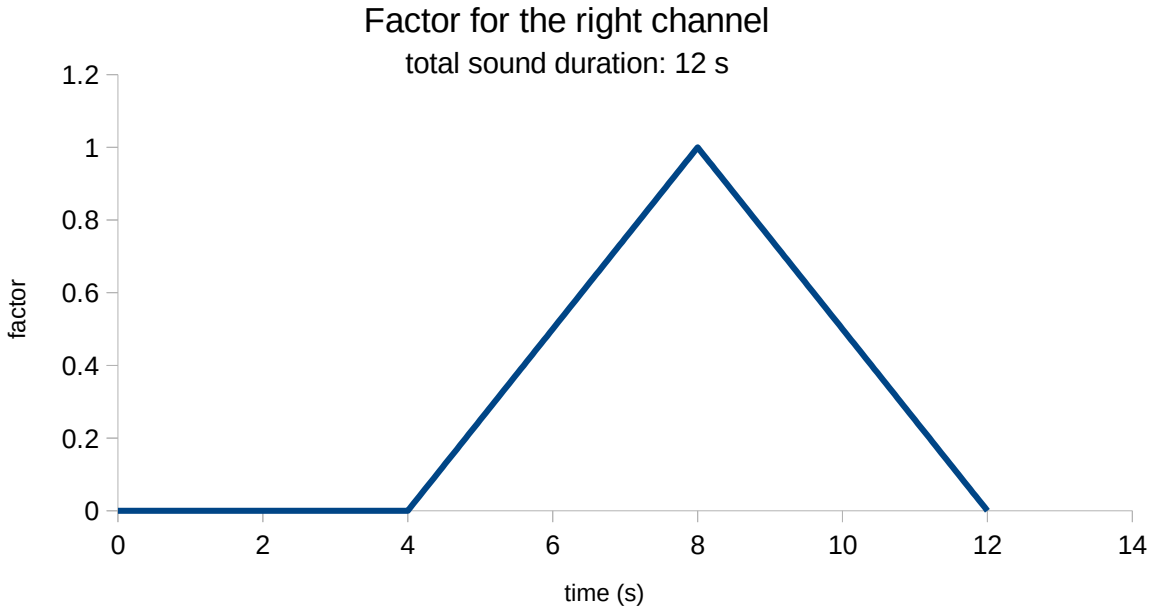


Figure 2. Multiplicative factor for the right channel as time progresses.