

## Programming Project #2

### Assignment Overview:

This assignment involves the coding and testing of a program that uses arithmetic expressions.

The assignment is worth 20 points (2% of the course grades), and must be completed before 11:59 pm on Monday, September 10<sup>th</sup>.

### Background:

One of the oldest problems of music is how to map the notes of a musical piece to a set of audio frequencies. There are various "tuning approaches" that state slightly different ways of assigning notes to a particular frequency. This project will require that you write a program that does one particular kind of this mapping.

First, we must define some form of musical note notation. The one common representation is the *octpch* note notation. This notation represents each note as a number pair, where the first number indicates which *octave* the note belongs to and the second number the note within the octave, termed the *pitch class*. There are 12 *semitones* within each octave on the keyboard, namely:

Note	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
Pc	0	1	2	3	4	5	6	7	8	9	10	11

Octpch representations are often written in decimal format. For example 5.9 is the 5<sup>th</sup> octave, 9<sup>th</sup> semitone. We must map sound frequencies to this representation. We start by choosing a reference note. The reference note frequency mapping is that 4.9 (A in the fourth octave) is 440 Hz. Every tuning mapping must insure that each note in the next higher octave has a frequency that is twice the previous octave, therefore  $5.9 \Rightarrow 880$  Hz and  $3.9 \Rightarrow 220$  Hz. Our tuning system will assume that each of the semitones within an octave is equally spaced, that is the distance from one semitone to the next is the same within the octave. This is called a *Tempered Scale*. The formula we use is:

$$x * 2^{(o + \frac{m}{12})}$$

where  $x$  is a reference Hz value for a known note,  $o$  is the *difference* (sign matters) in octave between the reference note and the note in question and  $m$  is the *difference* (sign matters) between the reference semitone and the note semitone in question. Consider the value for 0.0, C in the first octave, which is 4 octaves and 9 semitones below 4.9 (A of the 4th octave).

$$C_0 = 440 * 2^{(-4 + \frac{-9}{12})} = 440 * 2^{-4.75} = 16.35159 \text{ Hz}$$

### Problem Statement

Your program will prompt for three octpch pairs, convert each of them to Hertz (Hz), print out the values and then play them using the simple Windows Beep function (Windows only).

### Program Specifications:

1. Your program will output a brief, descriptive message when it first starts, indicating the purpose of the program and the user-required input and output that will be provided.
2. The program will then prompt for the three octpch pairs, each pair having two integers: the octave and the pitch class. It will prompt for the octave and pitchclass separately, so you do not have to convert 5.9 into the appropriate octave and pitch.
3. The program will convert each octpch pair (octave and pitch class), yielding a floating point Hz result.
4. The program will print the Hz value of three such pairs, along with the original octave and pitchclass values.
5. **OPTIONAL**. The program can then play the three notes using the Windows Beep function as described at the end of this document. It is not required and there are some restrictions.

### Program notes

1. Note that the way to do exponentiation in Python is via the `pow` (short for power) function. The `pow` function is part of the `math` module which you must import at the beginning of your program. You import that function as follows:

```
from math import pow
```

2. Once imported, the `pow` function has the following form:

`pow(x, y)` where x is raised to the power y

Try it. Open the Python Shell window, import **pow**, and try it. Begin with some values you know, e.g.  $2^3$  is **pow(2, 3)** which should be 8. Simply type **pow(2, 3)**, hit Return, and see what you get. Try some other values to get a feel for how **pow** works.

3. Once converted, you can print out the oct and pch values as well as the hz value for the pair. See the sample output below for an example.
4. There is no error checking required. Erroneous input can produce erroneous outp
5. Conditional statements (“if then else” statements) and loop statements (“for” or “while” statements) are not required for this program and should not be used.

### Assignment Deliverables:

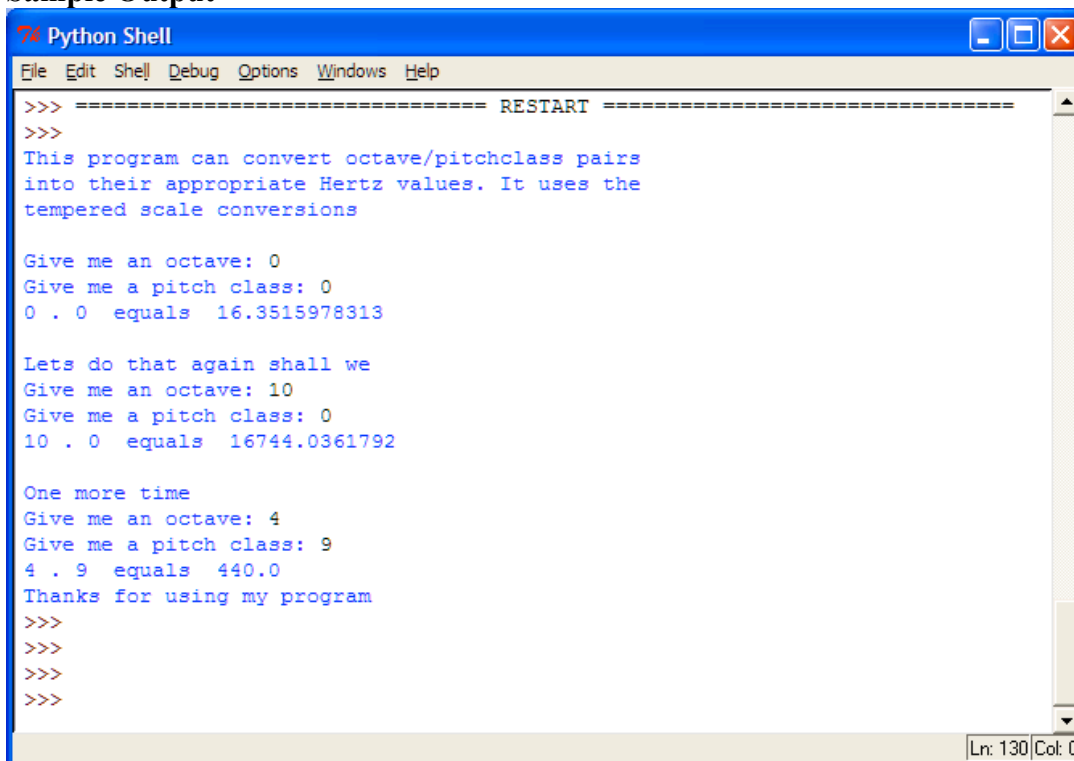
proj02.py – your source code solution

Please be sure to use the specified file name, and to preserve the file in your account until your assignment has been graded. You will electronically submit a copy of the file using the "handin" program.

### Assignment notes:

1. Before you start solving the problem, experiment with **pow** as described above to get a feel for how they work. That is, understand the tools before you begin trying to solve the problem.
2. Begin problem solving by using paper and pencil (and calculator) to solve the problem of calculating Hz values. You need to figure out how to work out a solution by hand before trying to develop a Python program.
3. Next try out some calculations with the Python shell: calculate a Hz We suggest you write your program incrementally. Begin by opening IDLE and creating a new Python program. Write comments at the top to clarify your goals. Save it as proj02.py
4. Next, add the code to prompt the user for the octave and pitch class number. Put in a print statement to print out the input values. Run the program and test it.
5. Continue in this manner to add functionality to your program, running and testing it at each step. For example, try including a calculation of a Hz value and print out the Hz result. Run the program and test it.
6. Hand in the program each time you reach an intermediate milestone to be sure that you will have submitted **something** that works before the deadline, even if some unforeseen glitch prevents you from handing in the completed solution.

### Sample Output



```
Python Shell
File Edit Shell Debug Options Windows Help
>>> ===== RESTART =====
>>>
This program can convert octave/pitchclass pairs
into their appropriate Hertz values. It uses the
tempered scale conversions

Give me an octave: 0
Give me a pitch class: 0
0 . 0 equals 16.3515978313

Lets do that again shall we
Give me an octave: 10
Give me a pitch class: 0
10 . 0 equals 16744.0361792

One more time
Give me an octave: 4
Give me a pitch class: 9
4 . 9 equals 440.0
Thanks for using my program
>>>
>>>
>>>
>>>
Ln: 130 | Col: 0
```

### Optional, Playing music

There is a sound function for Windows that allows you to play a frequency for a fixed period of time. It is in the winsound module. To access it, you need to import it at the beginning of your program as follows:

```
from winsound import Beep
```

Once imported, the Beep function is used in the following way:

```
Beep(hertzValue, time)
```

The hertzValue is the value calculated using the formula above. The time value is measure in milliseconds (1000 milliseconds = 1 second). A good time value is probably 500. Thus `Beep(440,500)` would play the equivalent of A in the fourth octave for half a second. Try `Beep` in the Python shell window. Import `Beep`, enter `Beep(440 ,500)` and see (hear) what you get. Try some other values to get a feel for how `Beep` works.

There are some restrictions on the use of the `Beep` function in Python. First, all values submitted should be integers. This, unfortunately, restricts the accuracy of the played notes. The program will run if floating point values are used, but the decimal part of any floating point number will be deleted. Second, the Python version can only play values in the range of 37 to 32,767. For example, it cannot play the example value of 0.0. Third, it only works on Windows, not the Mac or Linux.