# Question: Description: In 1787, Wolfgang Amadeus Mozart created a dic...

Python code

**Description:**[1]

In 1787, Wolfgang Amadeus Mozart created a dice game (Mozart's Musikalisches Würfelspiel). In the game, you compose a two-part waltz by pasting together 32 of 272 pre-composed musical elements at random.

**The waltz.** The waltz consists of two parts - the minuet and the trio. Each is comprised of 16 measures, which are generated at random according to a fixed set of rules, as described below.

- *Minuet.* The minuet consists of 16 measures. There are 176 possible Minuet measures, named `M1.wav` through `M176.wav`. To determine which one to play, roll two fair dice for each column of the following table.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 96 | 22 | 141 | 41 | 105 | 122 | 11 | 30 | 70 | 121 | 26 | 9 | 112 | 49 | 109 | 14 |
| 3 | 32 | 6 | 128 | 63 | 146 | 46 | 134 | 81 | 117 | 39 | 126 | 56 | 174 | 18 | 116 | 83 |
| 4 | 69 | 95 | 158 | 13 | 153 | 55 | 110 | 24 | 66 | 139 | 15 | 132 | 73 | 58 | 145 | 79 |
| 5 | 40 | 17 | 113 | 85 | 161 | 2 | 159 | 100 | 90 | 176 | 7 | 34 | 67 | 160 | 52 | 170 |
| 6 | 148 | 74 | 163 | 45 | 80 | 97 | 36 | 107 | 25 | 143 | 64 | 125 | 76 | 136 | 1 | 93 |
| 7 | 104 | 157 | 27 | 167 | 154 | 68 | 118 | 91 | 138 | 71 | 150 | 29 | 101 | 162 | 23 | 151 |
| 8 | 152 | 60 | 171 | 53 | 99 | 133 | 21 | 127 | 16 | 155 | 57 | 175 | 43 | 168 | 89 | 172 |
| 9 | 119 | 84 | 114 | 50 | 140 | 86 | 169 | 94 | 120 | 88 | 48 | 166 | 51 | 115 | 72 | 111 |
| 10 | 98 | 142 | 42 | 156 | 75 | 129 | 62 | 123 | 65 | 77 | 19 | 82 | 137 | 38 | 149 | 8 |
| 11 | 3 | 87 | 165 | 61 | 135 | 47 | 147 | 33 | 102 | 4 | 31 | 164 | 144 | 59 | 173 | 78 |
| 12 | 54 | 130 | 10 | 103 | 28 | 37 | 106 | 5 | 35 | 20 | 108 | 92 | 12 | 124 | 44 | 131 |

For example, if you roll an 11 for measure 3, then play measure 165.

- *Trio.* The trio consists of 16 measures. There are 96 possible Trio measures named `T1.wav` through `T96.wav`. To determine which one to play, roll one fair die for each column, and use the following table.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 72 | 6 | 59 | 25 | 81 | 41 | 89 | 13 | 36 | 5 | 46 | 79 | 30 | 95 | 19 | 66 |
| 2 | 56 | 82 | 42 | 74 | 14 | 7 | 26 | 71 | 76 | 20 | 64 | 84 | 8 | 35 | 47 | 88 |
| 3 | 75 | 39 | 54 | 1 | 65 | 43 | 15 | 80 | 9 | 34 | 93 | 48 | 69 | 58 | 90 | 21 |
| 4 | 40 | 73 | 16 | 68 | 29 | 55 | 2 | 61 | 22 | 67 | 49 | 77 | 57 | 87 | 33 | 10 |
| 5 | 83 | 3 | 28 | 53 | 37 | 17 | 44 | 70 | 63 | 85 | 32 | 96 | 12 | 23 | 50 | 91 |
| 6 | 18 | 45 | 62 | 38 | 4 | 27 | 52 | 94 | 11 | 92 | 24 | 86 | 51 | 60 | 78 | 31 |

[1] This description is stolen, wholesale, from https://introcs.cs.princeton.edu/java/assignments/mozart.html

There are 11^16 * 6^16 different possible results, some of which are more likely than others. Since this is over 10^23 different possibilities, each time you play the game you are likely to compose a piece of music that has never been heard before! Mozart carefully constructed the measures to obey a rigid harmonic structure, so each waltz reflects Mozart's distinct style. Unfortunately, due to the rigidity, the process never results in anything truly extraordinary.

You will implement a program that constructs and plays a random waltz according to the rules of Mozart's game.

**Directions:**
Two setup steps must be completed before beginning the assignment proper.

First, please install the simpleaudio Python package. This can be done with Python's Package Installer, **pip**. On Windows open the command prompt CMD, on MAC start the terminal, and enter the following commands:

Windows: `python -m pip install simpleaudio --user`
(You may need to use 'py -3' in place of 'python')

Mac: `python -m pip install simpleaudio --user`
(You may need to use 'python3' in place of python)

Please do the installation as soon as possible (this weekend) so that any installation problems can be resolved in a timely fashion.

Second, the audio files are provided to you as a ZIP archive. You must unzip the archive in order to access the files. Download the zip archive. Decompress it--your computer should know how to do this--**but**, just double-clicking the file won't be enough. I have placed a copy of the Python source file, **`musical_dice_game.py`**, inside this folder with all the audio files. Do not move it out of this folder (unless you understand how file paths work).

Now you are ready to begin the assignment. For this assignment, you will complete the implementation of **`musical_dice_game.py`**.

This file contains two tables represented as nested lists. The first table contains Mozart's table for randomly choosing 16 measures for the minuet portion of the waltz. The second table contains Mozart's table for randomly choosing 16 measures for the trio portion of the waltz. The values in each table are numbers of measures of music represented as strings. Each of these numbers uniquely identifies an audio file distributed with the assignment—the number is a part of the audio file's name.

You are responsible for implementing four functions. The first function, **`minuet_filename()`**, takes one argument, which is a string. That string will be the number of one of the entries randomly selected from Mozart's minuet table. The function will return a string that is the filename for the corresponding audio file. Each minuet musical file begins with a capital "M", and then the number from the table, and then a period, ".", and then the file extension for the audio file, "wav".

For example, if you select "119" by rolling a 9 for the first minuet measure, then the corresponding filename will be "M119.wav".

The second function, **`trio_filename()`**, is very similar. It also takes one argument, a numeral represented as a string that corresponds to the name of one of the audio files that contains a measure of trio music. Again, the function must return the complete filename of the audio file as a string. The filename begins with a capital "T", followed by the number from the table, then a period, ".", and then the file extension for the audio file, "wav".

For example, if you select "49" by rolling a 4 for the 11th trio measure, then the corresponding filename will be "T49.wav".

The third function, **`roll_dice()`**, is used to simulate the dice rolling. The function takes one integer argument that specifies how many 6-sided dice will be rolled. You will use functions from Python's **`random`** library to simulate rolling the specified number of 6-sided dice. The function will then return the sum of all the dice rolls performed by the function.

For example, if the first "die roll" produced a 5 and the second "die roll" produces a 2, then the result returned by the **`roll_dice()`** function should be 7.

The final function is called **`construct_waltz()`**. It does not take any arguments, and it does not return a value. Instead, you will loop over the columns of the minuet table and use the function call **`roll_dice(2)`** to simulate a roll of two 6-sided dice to randomly select a number identifying a musical measure from each column. You must record your selections. I recommend appending each selection to a list.

The function will then perform the same loop over the trio table, using the function call **`roll_dice(1)`** to simulate rolling one 6-sided die to randomly selecting a number identifying a musical measure from each column and appending the result to your list of selections.

After you have randomly selected all the measures that will make up your waltz, you then need to play your waltz you need to access the audio files that will compose your waltz for playback.

Loop over your list of selections and use the **`minuet_filename()`** and **`trio_filename()`** functions to construct filenames for the corresponding .wav audio files. Then pass each filename to the **`simpleaudio.WaveObject.from_wave_file()`** in order to construct an object of the **`simpleaudio WaveObject`** class. You should append each of these objects to a new list as you create them.

Now loop over your list of **`WaveObjects`** and call the **`play()`** method on each one. Make sure that you wait until each measure has finished playing before playing the next one.

**Bonus (5 points):** You can earn 5 bonus points for using the **`play_buffer()`** function to play your waltz rather than the **`play()`** function. (You can only earn the bonus credit if it works—you may want to implement your solution using the **`play()`** function first before then trying to use **`play_buffer()`** instead.)

And that is it. Your random waltz generator is now complete.

There are 11^16 * 6^16 different possible results, some of which are more likely than others. Since this is over 10^23 different possibilities, each time you play the game you are likely to compose a piece of music that has never been heard before! Mozart carefully constructed the measures to obey a rigid harmonic structure, so each waltz reflects Mozart's distinct style. Unfortunately, due to the rigidity, the process never results in anything truly extraordinary.

You will implement a program that constructs and plays a random waltz according to the rules of Mozart's game.

**Directions:**
Two setup steps must be completed before beginning the assignment proper.

First, please install the simpleaudio Python package. This can be done with Python's Package Installer, **pip**. On Windows open the command prompt CMD, on MAC start the terminal, and enter the following commands:

```
Windows:   python -m pip install simpleaudio --user
     (You may need to use 'py -3' in place of 'python')

Mac:       python -m pip install simpleaudio --user
     (You may need to use 'python3' in place of python)
```

Please do the installation as soon as possible (this weekend) so that any installation problems can be resolved in a timely fashion.

Second, the audio files are provided to you as a ZIP archive. You must unzip the archive in order to access the files. Download the zip archive. Decompress it--your computer should know how to do this--**but**, just double-clicking the file won't be enough. I have placed a copy of the Python source file, **`musical_dice_game.py`**, inside this folder with all the audio files. Do not move it out of this folder (unless you understand how file paths work).

Now you are ready to begin the assignment. For this assignment, you will complete the implementation of **`musical_dice_game.py`**.

This file contains two tables represented as nested lists. The first table contains Mozart's table for randomly choosing 16 measures for the minuet portion of the waltz. The second table contains Mozart's table for randomly choosing 16 measures for the trio portion of the waltz. The values in each table are numbers of measures of music represented as strings. Each of these numbers uniquely identifies an audio file distributed with the assignment—the number is a part of the audio file's name.

You are responsible for implementing four functions. The first function, **`minuet_filename()`**, takes one argument, which is a string. That string will be the number of one of the entries randomly selected from Mozart's minuet table. The function will return a string that is the filename for the corresponding audio file. Each minuet musical file begins with a capital "M", and then the number from the table, and then a period, ".", and then the file extension for the audio file, "wav".

For example, if you select "119" by rolling a 9 for the first minuet measure, then the corresponding filename will be "M119.wav".

The second function, **`trio_filename()`**, is very similar. It also takes one argument, a numeral represented as a string that corresponds to the name of one of the audio files that contains a measure of trio music. Again, the function must return the complete filename of the audio file as a string. The filename begins with a capital "T", followed by the number from the table, then a period, ".", and then the file extension for the audio file, "wav".

For example, if you select "49" by rolling a 4 for the 11th trio measure, then the corresponding filename will be "T49.wav".

The third function, **`roll_dice()`**, is used to simulate the dice rolling. The function takes one integer argument that specifies how many 6-sided dice will be rolled. You will use functions from Python's **`random`** library to simulate rolling the specified number of 6-sided dice. The function will then return the sum of all the dice rolls performed by the function.

For example, if the first "die roll" produced a 5 and the second "die roll" produces a 2, then the result returned by the **`roll_dice()`** function should be 7.

The final function is called **`construct_waltz()`**. It does not take any arguments, and it does not return a value. Instead, you will loop over the columns of the minuet table and use the function call **`roll_dice(2)`** to simulate a roll of two 6-sided dice to randomly select a number identifying a musical measure from each column. You must record your selections. I recommend appending each selection to a list.

The function will then perform the same loop over the trio table, using the function call **`roll_dice(1)`** to simulate rolling one 6-sided die to randomly selecting a number identifying a musical measure from each column and appending the result to your list of selections.

After you have randomly selected all the measures that will make up your waltz, you then need to play your waltz you need to access the audio files that will compose your waltz for playback.

Loop over your list of selections and use the **`minuet_filename()`** and **`trio_filename()`** functions to construct filenames for the corresponding .wav audio files. Then pass each filename to the **`simpleaudio.WaveObject.from_wave_file()`** in order to construct an object of the **`simpleaudio WaveObject`** class. You should append each of these objects to a new list as you create them.

Now loop over your list of **`WaveObjects`** and call the **`play()`** method on each one. Make sure that you wait until each measure has finished playing before playing the next one.

**Bonus (5 points):** You can earn 5 bonus points for using the **`play_buffer()`** function to play your waltz rather than the **`play()`** function. (You can only earn the bonus credit if it works—you may want to implement your solution using the **`play()`** function first before then trying to use **`play_buffer()`** instead.)

And that is it. Your random waltz generator is now complete.

Show transcribed image text

View comments (1)

## Expert Answer

Contact us

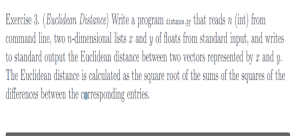
```
import simpleaudio

# Each of these lists corresponds to one column of the table used for the minuet
# portion Mozart's Musical Dice Game. The first two elements are set to None
# because, there is no way for a roll of two dice to return a 0 or a 1. Only the
# indices that can be the result of the roll of two dice have a value.
# Each value is the number labelling a possible musical choice for that measure.
mm01 = [None, None, "96", "32", "69", "40", "148", "104", "152", "119", "98", "3", "54"]
mm02 = [None, None, "22", "6", "95", "17", "74", "157", "60", "84", "142", "87", "130"]
mm03 = [None, None, "141", "128", "158", "113", "163", "27", "171", "114", "42", "165", "10"]
mm04 = [None, None, "41", "63", "13", "85", "45", "167", "53", "50", "156", "61", "103"]
mm05 = [None, None, "105", "46", "153", "161", "80", "154", "99", "140", "75", "135", "28"]
mm06 = [None, None, "122", "46", "55", "2", "97", "68", "133", "86", "129", "47", "37"]
mm07 = [None, None, "11", "134", "110", "159", "36", "118", "21", "169", "62", "147", "37"]
mm08 = [None, None, "30", "81", "24", "100", "107", "91", "127", "94", "123", "33", "5"]
mm09 = [None, None, "70", "117", "66", "90", "25", "138", "16", "120", "65", "102", "35"]
mm10 = [None, None, "121", "39", "139", "176", "143", "71", "155", "88", "77", "4", "20"]
mm11 = [None, None, "26", "126", "15", "7", "64", "150", "57", "48", "19", "31", "108"]
mm12 = [None, None, "9", "56", "132", "34", "125", "29", "175", "166", "82", "164", "92"]
mm13 = [None, None, "112", "174", "73", "67", "76", "101", "43", "51", "137", "144", "12"]
mm14 = [None, None, "49", "18", "58", "160", "136", "162", "168", "115", "38", "59", "124"]
mm15 = [None, None, "109", "116", "145", "52", "1", "23", "89", "72", "149", "173", "44"]
mm16 = [None, None, "14", "83", "79", "170", "93", "151", "172", "111", "8", "78", "131"]

# This table contains all of the columns of the minuet portion of Mozart's
# Musical Dice Game in order.
minuet_table = [mm01, mm02, mm03, mm04, mm05, mm06, mm07, mm08,
                mm09, mm10, mm11, mm12, mm13, mm14, mm15, mm16]

# Each of these lists corresponds to one column of the table used for the trio
# portion Mozart's Musical Dice Game. The first two elements are set to None
# because, there is no way for a roll of two dice to return a 0 or a 1. Only the
# indices that can be the result of the roll of one die have a value.
# Each value is the number labelling a possible musical choice for that measure.
tm01 = [None, "72", "56", "75", "40", "83", "18"]
tm02 = [None, "6", "82", "39", "73", "3", "45"]
tm03 = [None, "59", "42", "54", "16", "28", "62"]
tm04 = [None, "25", "74", "1", "68", "53", "38"]
tm05 = [None, "81", "14", "65", "29", "37", "4"]
tm06 = [None, "41", "7", "43", "55", "17", "27"]
tm07 = [None, "89", "26", "15", "2", "44", "52"]
tm08 = [None, "13", "71", "80", "61", "70", "94"]
tm09 = [None, "36", "76", "9", "22", "63", "11"]
tm10 = [None, "5", "20", "34", "67", "85", "92"]
tm11 = [None, "46", "64", "93", "49", "32", "24"]
tm12 = [None, "79", "84", "48", "77", "96", "86"]
tm13 = [None, "30", "8", "69", "57", "12", "51"]
tm14 = [None, "95", "35", "58", "87", "23", "60"]
tm15 = [None, "19", "47", "90", "33", "50", "78"]
tm16 = [None, "66", "88", "21", "10", "91", "31"]

# This table contains all of the columns of the trio portion of Mozart's
# Musical Dice Game in order.
trio_table = [tm01, tm02, tm03, tm04, tm05, tm06, tm07, tm08,
              tm09, tm10, tm11, tm12, tm13, tm14, tm15, tm16]

# This function takes a string as an argument and constructs a string that names
# one of the wav audio files that contains a measure of music for the minuet
# portion of Mozart's Musical Dice Game.
def minuet_filename(mmid):
    mf = "M" + mmid + ".wav"
    return mf

# This function takes a string as an argument and constructs a string that names
# one of the wav audio files that contains a measure of music for the trio
# portion of Mozart's Musical Dice Game.
def trio_filename(tmid):
    tf = "T" + tmid + ".wav"
    return tf

# This function takes a single interger, named 'num' as its argument. It
# generates the result of rolling 'num' many 6-sided dice.
def roll_dice(num):
    dice = [1, 2, 3, 4, 5, 6]
    sum = 0
    for i in range(0, num):
        sum += random.choice(dice)
    return sum

# Inside this main function you will randomly select 16 measures from the minuet
# table, one from each column of the minuet table, and then randomly select, 16
# measures from the trio table, one from each column of the trio table. Each
# measure is represented by a string numeral, and corresponds to the name of a
```

```
# playing the next measure.
def construct_waltz():
    mt = []
    tt = []
    wo = []
    for column in minuet_table:
        mt.append(column[roll_dice(2)])
    for column in trio_table:
        tt.append(column[roll_dice(1)])
    for item in mt:
        wo.append(minuet_filename(item))
    for item in tt:
        wo.append(trio_filename(item))
    buffer = b""
    for item in wo:

        buffer += simpleaudio.WaveObject.from_wave_file(item).audio_data
    waveObject = simpleaudio.WaveObject.from_wave_file(wo[0])
    nc = waveObject.num_channels
    bps = waveObject.bytes_per_sample
    sr = waveObject.sample_rate
    player = simpleaudio.play_buffer(buffer, nc, bps, sr)
    player.wait_done()

if __name__ == "__main__":
    construct_waltz()
```

Comment

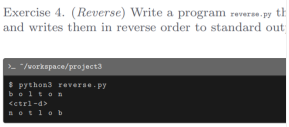
## Up next for you in Computer Science

Exercise 3. (Euclidean Distance) Write a program distance.py that reads n (int)


Exercise 3. (*Euclidean Distance*) Write a program distance.py that reads $n$ (int) from command line, two $n$-dimensional lists $x$ and $y$ of floats from standard input, and writes to standard output the Euclidean distance between two vectors represented by $x$ and $y$. The Euclidean distance is calculated as the square root of the sums of the squares of the differences between the corresponding entries.

See answer

Exercise 4. (Reverse) Write a program reverse.py that reads strings from


Exercise 4. (*Reverse*) Write a program reverse.py th and writes them in reverse order to standard out

```
~/workspace/project3
$ python3 reverse.py
b o l l o n
<ctrl-d>
n o t l o b
```

See answer

See more questions for subjects you study