Goal: In 1787, Wolfgang Amadeus Mozart created a dice game (Mozart's Musikalisches Würfelspiel), in which you compose a two-part waltz by pasting together 32 of 272 pre-composed musical elements at random. The goal of this assignment is to implement Mozart's game by writing a program to generate a two-part waltz and another program to play the waltz.

Part I: Warmup Problems

The problems in this part of the assignment are intended to give you solid practice on concepts (working with one- and two-dimensional lists) needed to solve the problems in Part II.

Problem 1. (*Reverse*) Write a program called `reverse.py` that accepts strings from standard input, and writes them in reverse order to standard output.

```
≻_ ~/workspace/mozart_waltz_generator
$ python3 reverse.py
b o l t o n
<ctrl-d>
n o t l o b
$ python3 reverse.py
m a d a m
<ctrl-d>
m a d a m
```

Problem 2. (*Euclidean Distance*) Write a program called `distance.py` that accepts n (int) as command-line argument, 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.

```
≻_ ~/workspace/mozart_waltz_generator
$ python3 distance.py 2
1 0 <enter>
0 1 <enter>
1.4142135623730951
$ python3 distance.py 5
-9 1 10 -1 1 <enter>
-5 9 6 7 4 <enter>
13.0
```

Problem 3. (*Birthday Problem*) Suppose that people enter an empty room until a pair of people share a birthday. On average, how many people will have to enter before there is a match? Write a program called `birthday.py` that accepts *trials* (int) as command-line argument, runs *trials* experiments to estimate this quantity — each experiment involves sampling individuals until a pair of them share a birthday, and writes the value to standard output.

```
≻_ ~/workspace/mozart_waltz_generator
$ python3 birthday.py 1000
24
$ python3 birthday.py 1000
25
```

Problem 4. (*Transpose*) Write a program called `transpose.py` that accepts m (int) and n (int) as command-line arguments, $m \times n$ floats from standard input representing the elements of an $m \times n$ matrix a, and writes to standard output the transpose of a. Recall that the transpose of an m-by-n matrix A is an n-by-m matrix B such that $B_{ij} = A_{ji}$, where $0 \leq i < n$ and $0 \leq j < m$.

```
≻_ ~/workspace/mozart_waltz_generator
$ python3 transpose.py 2 2
1 2 <enter>
3 4 <enter>
1.0 3.0
2.0 4.0
```

```
$ python3 transpose.py 2 3
1 2 3 <enter>
4 5 6 <enter>
1.0 4.0
2.0 5.0
3.0 6.0
```

Problem 5. (*Pascal's Triangle*) Pascal's triangle $\mathcal{P}_n$ is a triangular array with $n+1$ rows, each listing the coefficients of the binomial expansion $(x+y)^i$, where $0 \leq i \leq n$. For example, $\mathcal{P}_4$ is the triangular array:

$$\begin{array}{lllll} 1 \\ 1 & 1 \\ 1 & 2 & 1 \\ 1 & 3 & 3 & 1 \\ 1 & 4 & 6 & 4 & 1 \end{array}$$

The term $\mathcal{P}_n(i,j)$ is calculated as $\mathcal{P}_n(i-1,j-1) + \mathcal{P}_n(i-1,j)$, where $0 \leq i \leq n$ and $1 \leq j < i$, with $\mathcal{P}_n(i,0) = \mathcal{P}_n(i,i) = 1$ for all i. Write a program called `pascal.py` that accepts n (int) as command-line argument, and writes $\mathcal{P}_n$ to standard output.

```
>_ ~/workspace/mozart_waltz_generator

$ python3 pascal.py 3
1
1 1
1 2 1
1 3 3 1
$ python3 pascal.py 10
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
1 10 45 120 210 252 210 120 45 10 1
```

Part II: Mozart Waltz Generator

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` in the `data` directory. To determine which one to play, roll *two* fair dice, and use 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 a 4 and 6 for measure 8, then play measure 123 (ie, `data/M123.wav`).

- *Trio* The trio also consists of 16 measures. There are 96 possible trio measures named `T1.wav` through `T96.wav` in the `data` directory. To determine which one to play, roll *one* fair die, 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

For example, if you roll a 4 for measure 13, then play measure 57 (ie, `data/T57.wav`).

Composition: There are $11^{16} \times 6^{16} = 129,629,238,163,050,258,624,287,932,416$ possible compositions, some of which are more likely than others. Since this is a *huge* number of 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.

Problem 6. (*Generating the Waltz*) Write a program called `generatewaltz.py` that accepts the minuet and trio tables from standard input, generates a random sequence of 32 measures according to the rules described above, and writes the sequence to standard output.

```
>_ ~/workspace/mozart_waltz_generator
$ python3 generatewaltz.py < data/mozart.txt
69 95 27 103 105 129 21 24 66 155 48 34 43 18 89 78 72 39 59 68 29 7 15 94 76 34 93 77 12 95 47 10
$ python3 generatewaltz.py < data/mozart.txt
32 84 27 50 153 97 36 100 16 4 150 34 51 115 1 78 18 3 59 74 37 43 52 71 9 20 32 79 57 35 90 10
```

Problem 7. (*Playing the Waltz*) Write a program called `playwaltz.py` that accepts from standard input, a sequence of 32 integers representing the 32 measures of a waltz, and plays the waltz to standard audio. Before playing any audio, your program must check if the inputs are erroneous, and if so, must call `sys.exit(message)` to exit the program with an appropriate error message. Your program must check for the following errors:

- If the number of measures is not 32, exit with the message "A waltz must contain exactly 32 measures".
- If a minuet measure is not from $[1, 176]$, exit with the message "A minuet measure must be from [1, 176]".
- If a trio measure is not from $[1, 96]$, exit with the message "A trio measure must be from [1, 96]".

Note: No audio must be played in the event of an error.

```
>_ ~/workspace/mozart_waltz_generator
$ python3 generatewaltz.py < data/mozart.txt | python3 playwaltz.py
```

Data: The `data` directory contains:

- The 272 minuet and trio measures as `.wav` files.
- The values of the above minuet and trio tables in `mozart.txt`.
- A sample waltz, `mozart.wav`, generated using the process described above.

Files to Submit:

1. `reverse.py`
2. `distance.py`

3. `birthday.py`
4. `transpose.py`
5. `pascal.py`
6. `generatewaltz.py`
7. `playwaltz.py`
8. `notes.txt`

Before you submit your files, make sure:

- You do not use concepts from sections beyond *Input and Output*.
- Your code follows good programming principles (ie, it is clean and well-organized; uses meaningful variable names; and includes useful comments).
- You edit the sections (`#1` mandatory, `#2` if applicable, and `#3` optional) in the given `notes.txt` file as appropriate. In section `#1`, for each problem, you must include in nore more than 100 words: a short, high-level description of the problem; your approach to solve it; and any issues you encountered and if/how you managed to solve them.

Acknowledgement: Part II of this assignment is an adaptation of the Mozart Waltz Generator assignment developed at Princeton University by David Costanzo and Kevin Wayne.