Objectives. Write simple Python programs that involve the following:

1. One-dimensional lists.
2. Two-dimensional (non-ragged and ragged) lists.

Problem 1. (*Basic Stats*) Write a program `stats.py` that writes the mean μ, variance Var, and standard deviation σ of the floats $x_1, x_2, \dots, x_n$ supplied as command-line arguments, computed using the formulae:

$$\mu = \frac{x_1 + x_2 + \cdots + x_n}{n}, Var = \frac{(x_1 - \mu)^2 + (x_2 - \mu)^2 + \cdots + (x_n - \mu)^2}{n}, \text{ and } \sigma = \sqrt{Var}.$$

```
$ python3 stats.py 77 41 3 76 19 48 92 10 26 63
45.5
856.65
29.268583840015218
```

Problem 2. (*Longest Stretch*) Write a program `longest_stretch.py` that writes the longest contiguous sequence of equal integer values, from the sequence of integers read from the command line. If there are multiple stretches of the same length, the program should write the first one.

```
$ python3 longest_stretch.py 1 1 2 3 4 4 4 5 5 6 6 6 7
4 4 4
$ python3 longest_stretch.py 1 1 2 3 4 4 4 5 5 6 6 6 6
6 6 6 6
$ python3 longest_stretch.py 1 1 2 3 4 4 4 5 5 6 6 6 6 7
6 6 6 6
$ python3 longest_stretch.py 1 2 3 4 5
1
```

Problem 3. (*Inverse Permutation*) Write a program `inverse_permutation.py` that accepts a permutation of the integers 0 through $n-1$ as n command-line arguments and writes its inverse. If the permutation is a list `a`, its inverse is the list `b` such that `a[b[i]] = b[a[i]] = i`. Be sure to check that the input is a valid permutation. If not, the program must exit with the message "Not a permutation" — you can do this by calling the function `sys.exit(msg)`, where `msg` is the message you want to write.

```
$ python3 inverse_permutation.py 1 2 3 4 5
Not a permutation
$ python3 inverse_permutation.py 5 3 4 0 1 2
3 4 5 1 2 0
$ python3 inverse_permutation.py 0 1 2 3 4 5
0 1 2 3 4 5
$ python3 inverse_permutation.py 5 4 3 2 1 0
5 4 3 2 1 0
```

Problem 4. (*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 `birthday.py` that takes an integer *trials* from the command line and runs *trials* experiments to estimate this quantity, where each experiment involves sampling individuals until a pair of them share a birthday. Assume birthdays to be uniform random integers from the interval $[0, 365)$.

```
$ python3 birthday.py 1000
24
```

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 $0 \le i \le n$. 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 $0 \le i \le n$ and $1 \le j < i$, with $\mathcal{P}_n(i,0) = \mathcal{P}_n(i,i) = 1$ for all i. Write a program `pascal.py` that takes an integer n as command-line argument and writes $\mathcal{P}_n$.

```
$ 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
```

Files to Submit

1. `stats.py`
2. `longest_stretch.py`
3. `inverse_permutation.py`
4. `birthday.py`
5. `pascal.py`

Before you submit:

- Make sure your programs meet the input and output specifications by running the following command on the terminal:

  ```
  $ python3 run_tests.py -v [<problems>]
  ```

  where the optional argument `<problems>` lists the problems (`Problem1`, `Problem2`, etc.) you want to test, separated by spaces; all the problems are tested if no argument is given.

- Make sure your programs meet the style requirements by running the following command on the terminal:

  ```
  $ pycodestyle <program>
  ```

  where `<program>` is the `.py` file whose style you want to check.