**Problem 1.** (*Day of the Week*) Write a program called `day_of_week.py` that accepts $m$ (int), $d$ (int), and $y$ (int) as command-line arguments, representing a date, and writes the day of the week (0 for Sunday, 1 for Monday, and so on) *dow* to standard output, computed as

$$\begin{aligned} y_0 &= y - (14 - m)/12, \\ x_0 &= y_0 + y_0/4 - y_0/100 + y_0/400, \\ m_0 &= m + 12 \times ((14 - m)/12) - 2, \\ dow &= (d + x_0 + 31 \times m_0/12) \bmod 7. \end{aligned}$$

```
>_ ~/workspace/project1
$ python3 day_of_week.py 3 14 1879
5
$ python3 day_of_week.py 2 12 1809
0
```

Directions:

- Accept three integers $m$, $d$, and $y$ as command-line arguments.
- Compute and write the value of day of the week *dow*.
- Use `//` (floored division) for `/` and `%` for mod.

**Problem 2.** (*Mercator Projection*) The Mercator projection is a conformal (angle preserving) projection that maps latitude $\varphi$ and longitude $\lambda$, expressed in degrees, to rectangular coordinates $(x, y)$. It is widely used — for example, in nautical charts and in the maps that you print from the web. The projection, with the center of map set to the prime meridian (0 degrees) and radius of Earth set to 1, is defined by the equations

$$x = \lambda \text{ and } y = \frac{\ln\left(\frac{1+\sin\varphi}{1-\sin\varphi}\right)}{2}.$$

Write a program called `mercator.py` that accepts latitude $\varphi$ (float) and longitude $\lambda$ (float) as command-line arguments and writes the corresponding $x$ and $y$ values to standard output, separated by a space.

```
>_ ~/workspace/project1
$ python3 mercator.py 42.36 -71.06
-71.06 0.8176461519422712
$ python3 mercator.py 51.18 -1.83
-1.83 1.0431252282097918
```

Directions:

- Accept two floats $\varphi$ and $\lambda$ as command-line arguments.
- Compute and write the values of $x$ and $y$, separated by a space.
- Use `math.radians()` to convert degrees to radians.
- Use `math.log()` for natural logarithm.

**Problem 3.** (*Great Circle Distance*) Write a program called `great_circle.py` that accepts $x_1$ (float), $y_1$ (float), $x_2$ (float), and $y_2$ (float) as command-line arguments, representing the latitude and longitude in degrees of two points on Earth, and writes to standard output the great-circle distance $d$ (in km) between them, computed as

$$d = 6359.83 \arccos(\sin(x_1)\sin(x_2) + \cos(x_1)\cos(x_2)\cos(y_1 - y_2)).$$

```
>_ ~/workspace/project1
$ python3 great_circle.py 48.87 -2.33 37.8 -122.4
8701.387455462233
$ python3 great_circle.py 46.36 -71.06 39.90 116.41
10376.503884802196
```

Directions:

- Accept four floats $x_1$, $y_1$, $x_2$, and $y_2$ as command-line arguments.
- Compute and write the value of great-circle distance $d$.
- Use `math.radians()` to convert degrees to radians.
- To calculate the arccosine of a number, use `math.acos()`.

**Problem 4.** (*Wind Chill*) Given the temperature $t$ (in Fahrenheit) and the wind speed $v$ (in miles per hour), the National Weather Service defines the effective temperature (the wind chill) to be

$$w = 35.74 + 0.6215t + (0.4275t - 35.75)v^{0.16}.$$

Write a program called `wind_chill.py` that accepts $t$ (float) and $v$ (float) as command-line arguments, and writes the wind chill $w$ to standard output.

```
>_ ~/workspace/project1
$ python3 wind_chill.py 32 15
21.588988890532022
$ python3 wind_chill.py 10 10
-3.5402167842280647
```

Directions:

- Accept two floats $t$ and $v$ as command-line arguments.
- Compute and write the value of wind chill $w$.
- Use `**` for exponentiation, ie, use `x ** y` for $x^y$.

**Problem 5.** (*Gravitational Force*) Write a program called `gravitational_force.py` that accepts $m_1$ (float), $m_2$ (float), and $r$ (float) as command-line arguments, representing the masses (in kg) of two objects and the distance (in m) between their centers, and writes to standard output the gravitational force $f$ (in N) acting between the objects, computed as

$$f = G\frac{m_1 m_2}{r^2},$$

where $G = 6.674 \times 10^{-11}$ (in $\mathrm{m}^3\ \mathrm{kg}^{-1}\ \mathrm{s}^{-2}$) is the gravitational constant.

```
>_ ~/workspace/project1
$ python3 gravitational_force.py 2e30 6e24 1.5e11
3.5594666666666664e+22
$ python3 gravitational_force.py 6e24 7.35e22 3.84e8
1.9960083007812498e+20
```

Directions:

- Accept three floats $m_1$, $m_2$, and $r$ as command-line arguments.
- Compute and write the value of gravitational force $f$.
- Use scientific notation for the value of $G$ (eg, `6.022e23` for $6.022 \times 10^{23}$).

**Problem 6.** (*Snell's Law*) Snell's law states that given two mediums, the ratio of the sines of the angles (in degrees) of incidence and refraction is equivalent to the reciprocal of the ratio of the indices of refraction of the two mediums, ie,

$$\frac{\sin(\theta_1)}{\sin(\theta_2)} = \frac{n_2}{n_1}.$$

Write a program called `snell.py` that accepts $\theta_1$ (float), $n_1$ (float), and $n_2$ (float) as command-line arguments, and writes to standard output the corresponding angle of refraction $\theta_2$ in degrees.

```
>_ ~/workspace/project1
$ python3 snell.py 58 1 1.52
33.912513998258994
$ python3 snell.py 30 1 1.2
24.624318352164074
```

Directions:

- Accept three floats $\theta_1$, $n_1$, and $n_2$ as command-line arguments.
- Compute and write the value of the angle of refraction $\theta_2$ in degrees.
- Use `math.radians()` and `math.degrees()` to convert degrees to radians and vice versa.
- To calculate the arcsine of a number, use `math.asin()`.

**Problem 7.** (*Gambler's Ruin*) Consider a coin-flipping game with two players where player one wins each toss with probability $p$, and player two wins with probability $q = 1 - p$. Suppose player one has $n_1$ pennies and player two $n_2$ pennies. Assuming an unfair coin (ie, $p \neq 1/2$), the probabilities $p_1$ and $p_2$ that players one and two, respectively, will end penniless are

$$p_1 = \frac{1 - (\frac{p}{q})^{n_2}}{1 - (\frac{p}{q})^{n_1+n_2}} \text{ and } p_2 = \frac{1 - (\frac{q}{p})^{n_1}}{1 - (\frac{q}{p})^{n_1+n_2}}.$$

Write a program called `gambler.py` that accepts $n_1$ (int), $n_2$ (int), and $p$ (float) as command-line arguments, and writes the probabilities $p_1$ and $p_2$ to standard output, separated by a space.

```
>_ ~/workspace/project1
$ python3 gambler.py 10 100 0.51
0.6661883734200654 0.3338116265799349
$ python3 gambler.py 100 10 0.51
0.006110712510580903 0.9938892874894192
```

Directions:

- Accept integers $n_1, n_2$ and float $p$ as command-line arguments.
- Compute and write the values of $p_1$ and $p_2$, separated by a space.

**Problem 8.** (*Waiting Time*) If $\lambda$ is the average number of events per unit of time, the probability $p$ that one has to wait longer than time $t$ until the next event is given by the exponential distribution

$$p = e^{-\lambda t}.$$

Write a program called `waiting_time.py` that accepts $\lambda$ (float) and $t$ (float) as command-line arguments, and writes the probability $p$ to standard output.

```
>_ ~/workspace/project1
$ python3 waiting_time.py 0.1 5
0.6065306597126334
$ python3 waiting_time.py 0.6 3
0.16529888822158656
```

Directions:

- Accept two floats $\lambda$ and $t$ as command-line arguments.
- Compute and write the value of $p$.
- Use `math.exp(x)` for $e^x$.

**Problem 9.** (*Die Roll*) Write a program called `die_roll.py` that accepts $n$ (int) as command-line argument, representing the number of sides of a fair die, rolls an $n$-sided die twice, and writes to standard output the sum of the numbers rolled.

```
>_ ~/workspace/project1
$ python3 die_roll.py 6
12
$ python3 die_roll.py 6
10
```

Directions:

- Accept an integer $n$ as command-line argument.
- Use `stdrandom.uniformInt()` with suitable arguments to simulate a single roll of an $n$-sided die.
- Write the sum of the numbers rolled.

**Problem 10.** (*Three Sort*) Write a program called `three_sort.py` that accepts $x$ (int), $y$ (int), and $z$ (int) as command-line arguments, and writes them to standard output in ascending order, separated by spaces. Your solution must only use `min()`, `max()`, and basic arithmetic operations to figure out the ordering.

```
>_ ~/workspace/project1
$ python3 three_sort.py 1 3 2
1 2 3
$ python3 three_sort.py 3 2 1
1 2 3
```

Directions:

- Accept three integers $x$, $y$, and $z$ as command-line arguments.
- Find the smallest value $\alpha$ and largest value $\omega$ using `min()` and `max()` functions.
- Find the middle value as an arithmetic combination of $x$, $y$, $z$, $\alpha$, and $\omega$.
- Write the numbers in ascending order, separated by a space.

### Files to Submit

1. `day_of_week.py`
2. `mercator.py`
3. `great_circle.py`

4. `wind_chill.py`
5. `gravitational_force.py`
6. `snell.py`
7. `gambler.py`
8. `waiting_time.py`
9. `die_roll.py`
10. `three_sort.py`
11. `notes.txt`

Before you submit your files, make sure:

- You do not use concepts from sections beyond "Basic Data Types".
- Your programs meet the style requirements by running the following command in the terminal.

```
>_ ~/workspace/project1
$ pycodestyle <program>
```

- Your code is adequately commented, follows good programming principles, and meets any specific requirements such as corner cases and running times.
- You update the `notes.txt` file.