Assignment 1 (Straight-line Programs)

Goal: Implement simple programs *without* using control-flow (ie, branch and loop) statements.

Problem 1. (*Name and Age*) Write a program called `name_age.py` that accepts *name* (str) and *age* (str) as command-line arguments, and writes the string "*name* is *age* years old." to standard output.

```
>_ ~/workspace/straightline_programs
$ python3 name_age.py Alice 19
Alice is 19 years old.
$ python3 name_age.py Bob 23
Bob is 23 years old.
```

Problem 2. (*Greet Three*) Write a program called `greet_three.py` that accepts $name_1$ (str), $name_2$ (str), and $name_3$ (str) as command-line arguments, and writes the string "Hi $name_3$, $name_2$, and $name_1$." to standard output.

```
>_ ~/workspace/straightline_programs
$ python3 greet_three.py Alice Bob Carol
Hi Carol, Bob, and Alice.
$ python3 greet_three.py Dan Eve Fred
Hi Fred, Eve, and Dan.
```

Problem 3. (*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/straightline_programs
$ python3 day_of_week.py 3 14 1879
5
$ python3 day_of_week.py 2 12 1809
0
```

Problem 4. (*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/straightline_programs
$ python3 three_sort.py 1 3 2
1 2 3
$ python3 three_sort.py 3 2 1
1 2 3
```

Problem 5. (*Body Mass Index*) The body mass index (BMI) is the ratio of the weight w of a person (in kg) to the square of the height h (in m). Write a program called `bmi.py` that accepts w (float) and h (float) as command-line arguments, and writes the BMI value to standard output.

```
>_ ~/workspace/straightline_programs
$ python3 bmi.py 75 1.83
22.395413419331717
$ python3 bmi.py 97 1.75
```

```
31.6734693877551
```

Problem 6. (*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/straightline_programs
$ python3 wind_chill.py 32 15
21.588988890532022
$ python3 wind_chill.py 10 10
-3.5402167842280647
```

Problem 7. (*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 m^3 kg^{-1} s^{-2}) is the gravitational constant.

```
>_ ~/workspace/straightline_programs
$ python3 gravitational_force.py 2e30 6e24 1.5e11
3.559466666666664e+22
$ python3 gravitational_force.py 6e24 7.35e22 3.84e8
1.9960083007812498e+20
```

Problem 8. (*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/straightline_programs
$ python3 gambler.py 10 100 0.51
0.6661883734200654 0.3338116265799349
$ python3 gambler.py 100 10 0.51
0.006110712510580903 0.9938892874894192
```

Problem 9. (*Waiting Time*) If λ 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 λ (float) and t (float) as command-line arguments, and writes the probability p to standard output.

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

Problem 10. (*Cartesian Coordinates*) Write a program called `cartesian.py` that accepts r (float) and θ (float) representing the coordinates of a point in polar form, converts the coordinates into Cartesian form x and y using formulae $x = r\cos(\theta)$ and $y = r\sin(\theta)$, and writes those values to standard output, separated by a space.

```
>_ ~/workspace/straightline_programs
$ python3 cartesian.py 1 45
0.7071067811865476 0.7071067811865475
$ python3 cartesian.py 1 60
0.5000000000000001 0.8660254037844386
```

Problem 11. (*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/straightline_programs
$ 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
```

Problem 12. (*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 θ_1 (float), n_1 (float), and n_2 (float) as command-line arguments, and writes to standard output the corresponding angle of refraction θ_2 in degrees.

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

Problem 13. (*Uniform Random Numbers*) Write a program called `stats.py` that accepts a (int) and b (int) as command-line arguments, generates three random floats (x_1, x_2, and x_3), each from the interval $[a, b)$, computes their mean $\mu = (x_1 + x_2 + x_3)/3$, variance $var = ((x_1 - \mu)^2 + (x_2 - \mu)^2 + (x_3 - \mu)^2)/3$, and standard deviation $\sigma = \sqrt{var}$, and writes those values to standard output, separated by a space.

```
>_ ~/workspace/straightline_programs
$ python3 stats.py 0 1
0.5731084550427492 0.04897843881307027 0.22131072909615176
$ python3 stats.py 50 100
91.3736830296877 25.288830238538182 5.028800079396494
```

Problem 14. (*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/straightline_programs

$ python3 die_roll.py 6
12
$ python3 die_roll.py 6
10
```

Problem 15. (*Triangle Inequality*) Write a program called `triangle.py` that accepts x (int), y (int), and z (int) as command-line arguments, and writes `True` to standard output if each one of them is less than or equal to the sum of the other two, and `False` otherwise.

```
>_ ~/workspace/straightline_programs

$ python3 triangle.py 3 3 3
True
$ python3 triangle.py 2 4 7
False
```

Files to Submit:

1. `name_age.py`
2. `greet_three.py`
3. `day_of_week.py`
4. `three_sort.py`
5. `bmi.py`
6. `wind_chill.py`
7. `gravitational_force.py`
8. `gambler.py`
9. `waiting_time.py`
10. `cartesian.py`
11. `great_circle.py`
12. `snell.py`
13. `stats.py`
14. `die_roll.py`
15. `triangle.py`
16. `notes.txt`

Before you submit your files, make sure:

- You do not use concepts from sections beyond *Basic Data Types*.
- Your code is adequately commented, follows good programming principles, and meets any problem-specific requirements.
- You edit the sections (`#1` mandatory, `#2` if applicable, and `#3` optional) in the given `notes.txt` file as appropriate. Section `#1` must provide a clear high-level description of each problem in no more than 100 words.