Project 2 (Programs with Control Flow)

Problem 1. (*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. Your program should report the message "Value of t must be $\leq$ 50 F" if $t > 50$, and the message "Value of v must be $>$ 3 mph" if $v \leq 3$.

```
>_ ~/workspace/project2

$ python3 wind_chill.py 55 15
Value of t must be <= 50 F
$ python3 wind_chill.py 32 15
21.588988890532022
```

Directions:

- Use an if statement to decide when to report the error messages and when to compute and write the wind chill w.

Problem 2. (*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, computes the day of the week (0 for Sunday, 1 for Monday, and so on) *dow* using the formulae below, and writes the day as a string ("Sunday", "Monday", and so on) to standard output.

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

```
>_ ~/workspace/project2

$ python3 day_of_week.py 3 14 1879
Friday
$ python3 day_of_week.py 4 12 1882
Wednesday
```

Directions:

- After computing *dow*, use an if statement to write the correct output based on the value of *dow*.

Problem 3. (*Playing Card*) Write a program called `card.py` that simulates the selection of a random card from a standard deck of 52 playing cards, and writes it to standard output.

```
>_ ~/workspace/project2

$ python3 card.py
3 of Clubs
$ python3 card.py
Ace of Spades
```

Directions:

- Set *rank* to a random integer from $[2, 14]$.
- Use an if statement to set *rankStr* to a string corresponding to *rank* — the ranks are 2, 3, ..., *Jack*, *Queen*, *King*, and *Ace*.
- Set *suit* to a random integer from $[1, 4]$.

- Use an if statement to set *suitStr* to a string corresponding to *suit* — the suits are *Clubs*, *Diamonds*, *Hearts*, and *Spades*.
- Write the desired output.

Problem 4. (*Root Finding*) Write a program called `root.py` (a variant of the `sqrt.py` program we dicussed in class) that accepts k (int), c (float), and *epsilon* (float) as command-line arguments, and writes to standard output the kth root of c, up to *epsilon* decimal places.

```
>_ ~/workspace/project2

$ python3 root.py 3 2 1e-15
1.2599210498948732
$ python3 root.py 3 27 1e-15
3.0
```

Directions:

- Set t (our guess) to c.
- Repeat as long as $|1 - c/t^k| > \epsilon$:
  - Replace t by $t - f(t)/f'(t)$, where $f(t) = t^k - c$ and $f'(t) = kt^{k-1}$.
- Write t (the kth root of c).

Problem 5. (*Greatest Common Divisor*) Write a program called `gcd.py` that accepts p (int) and q (int) as command-line arguments, and writes to standard output the greatest common divisor (gcd) of p and q.

```
>_ ~/workspace/project2

$ python3 gcd.py 408 1440
24
$ python3 gcd.py 21 22
1
```

Directions:

- Repeat as long as $p \bmod q \neq 0$:
  - Exchange p and q with q and $p \bmod q$.
- Write q (the gcd).

Problem 6. (*Fibonacci Number*) Write a program called `fibonacci.py` that accepts n (int) as command-line argument, and writes to standard output the nth number from the Fibonacci sequence $(1, 1, 2, 3, 5, 8, 13, \dots)$.

```
>_ ~/workspace/project2

$ python3 fibonacci.py 10
55
$ python3 fibonacci.py 15
610
```

Directions:

- Set a, b (the first two Fibonacci numbers) to 1, and i to 3.
- Repeat as long as $i \leq n$:
  - Exchange a and b with b and $a + b$.

    – Increment i by 1.

- Write b (nth Fibonacci number).

Problem 7. (*Sum of Powers*) Write a program called `sum_of_powers.py` that accepts n (int) and k (int) as command-line arguments, and writes to standard output the sum $1^k + 2^k + \cdots + n^k$.

```
>_ ~/workspace/project2

$ python3 sum_of_powers.py 15 1
120
$ python3 sum_of_powers.py 10 3
3025
```

Directions:

- Set *total* to 0.
- Repeat for each $i \in [1, n]$:
    - Increment *total* by i^k.
- Write *total* (sum of powers).

Problem 8. (*Dragon Curve*) The instructions for drawing a dragon curve are strings of the characters F, L, and R, where F means "draw a line while moving 1 unit forward", L means "turn left", and R means "turn right". The key to solving this problem is to note that a curve of order n is a curve of order $n-1$ followed by an L followed by a curve of order $n-1$ traversed in reverse order, replacing L with R and R with L. Write a program called `dragon_curve.py` that accepts n (int) as command-line argument, and writes to standard output the instructions for drawing a dragon curve of order n.

```
>_ ~/workspace/project2

$ python3 dragon_curve.py 0
F
$ python3 dragon_curve.py 1
FLF
$ python3 dragon_curve.py 2
FLFLFRF
$ python3 dragon_curve.py 3
FLFLFRFLFLFRFRF
```

Directions:

- Set *dragon* and *nogard* to the string "F".
- Repeat for each $i \in [1, n]$:
    - Exchange *dragon* and *nogard* with *dragon* "L" *nogard* and *dragon* "R" *nogard*.
- Write *dragon* (dragon curve of order n).

Problem 9. (*Perfect Numbers*) A perfect number is a positive integer whose proper divisors add up to the number. For example, 6 is a perfect number since its proper divisors 1, 2, and 3 add up to 6. Write a program called `perfect_numbers.py` that accepts n (int) as command-line argument, and writes to standard output the perfect numbers that are less than or equal to n.

```
>_ ~/workspace/project2

$ python3 perfect_numbers.py 10
6
$ python3 perfect_numbers.py 1000
```

```
6
28
496
```

Directions:

- Repeat for each $i \in [2, n]$:
  - Set *total* (sum of divisors of i) to 0.
  - Repeat for each $j \in [1, i/2]$:
    - If $i \bmod j = 0$, increment *total* by j.
  - If $total = i$, write i (perfect number).

Problem 10. (*Ramanujan Numbers*) Srinivasa Ramanujan was an Indian mathematician who became famous for his intuition for numbers. When the English mathematician G. H. Hardy came to visit him one day, Hardy remarked that the number of his taxi was 1729, a rather dull number. Ramanujan replied, "No, Hardy! It is a very interesting number. It is the smallest number expressible as the sum of two cubes in two different ways." Verify this claim by writing a program `ramanujan_numbers.py` that accepts n (int) as command-line argument, and writes to standard output all integers less than or equal to n that can be expressed as the sum of two cubes in two different ways. In other words, find distinct positive integers a, b, c, and d such that $a^3 + b^3 = c^3 + d^3 \leq n$.

```
>_ ~/workspace/project2

$ python3 ramanujan_numbers.py 10000
1729 = 1^3 + 12^3 = 9^3 + 10^3
4104 = 2^3 + 16^3 = 9^3 + 15^3
$ python3 ramanujan_numbers.py 40000
1729 = 1^3 + 12^3 = 9^3 + 10^3
4104 = 2^3 + 16^3 = 9^3 + 15^3
13832 = 2^3 + 24^3 = 18^3 + 20^3
39312 = 2^3 + 34^3 = 15^3 + 33^3
32832 = 4^3 + 32^3 = 18^3 + 30^3
20683 = 10^3 + 27^3 = 19^3 + 24^3
```

Directions:

- Use four nested while loops, with the following bounds on the loop variables:

$$1 \leq a \leq \sqrt[3]{n}$$
$$a + 1 \leq b \leq \sqrt[3]{n - a^3}$$
$$a + 1 \leq c \leq \sqrt[3]{n}$$
$$c + 1 \leq d \leq \sqrt[3]{n - c^3}$$

- Inside the innermost loop, if $a^3 + b^3 = c^3 + d^3$, write the desired output.

Note: use $x \times x \times x$ instead of x^3 and $x \times x \times x \leq y$ in place of $x \leq \sqrt[3]{y}$.

Files to Submit

1. `wind_chill.py`
2. `day_of_week.py`
3. `card.py`
4. `root.py`
5. `gcd.py`

6. `fibonacci.py`
7. `sum_of_powers.py`
8. `dragon_curve.py`
9. `perfect_numbers.py`
10. `ramanujan_numbers.py`
11. `notes.txt`

Before you submit your files, make sure:

- You do not use concepts from sections beyond "Control Flow".
- Your code is adequately commented, follows good programming principles, and meets any specific requirements such as corner cases and running times.
- 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 the project in no more than 200 words.