---
comparethree.py ->
---
1. Start the program.
2. Receive three integers as inputs. These can be from the command line or any other input method you choose.
3. Compare the first integer with the second and the second with the third.
4. If all comparisons are true (meaning all three integers are the same), then print or return 'equal'.
5. If any of the comparisons are false (meaning at least one integer is different from the others), then print or return 'not equal'.
6. End the program.

---
rangecheck.py ->
---
1. Start the script.
2. Check if exactly one argument is provided (excluding the script name itself). If not, print False and exit the script. This is because the requirement is to test a single float argument.
3. If the correct number of arguments is provided, convert the argument from a string to a float. This is necessary because command-line arguments are passed as strings by default.
4. Check if the converted float value is strictly greater than 0.0 and strictly less than 1.0. This is done using a chained comparison operation.
5. Print True if the value satisfies the condition of being strictly between 0.0 and 1.0, or False otherwise.
6. End the script.

---
windchill.py ->
---
1. Define a function calculate_wind_chill to compute the wind chill index using the temperature and wind speed.
2. Check if the number of command-line arguments is exactly 3 (the script name, temperature, and wind speed).
3. If not, print an error message indicating an invalid number of input arguments and exit the script with an error code.
4. Read the temperature and wind speed from the command-line arguments and convert them to floating-point numbers.
5. Validate the input by checking if the temperature is within the range -58 to 41 degrees Fahrenheit and if the wind speed is greater than or equal to 2 mph.
6. If the input arguments are not valid, print an error message indicating an invalid input argument value.
7. If the input arguments are valid, calculate the wind chill using the calculate_wind_chill function and print the result formatted to two decimal places.

---
powersoftwo.py ->
---
1. It imports the sys module, which allows the script to access command-line arguments.
2. It reads the first command-line argument, which is expected to be the value of n, and converts this argument to an integer.
3. It initializes a variable named power to 1, which represents the first positive power of 2.
4. It enters a while loop that continues to execute as long as power is less than or equal to n.
5. Inside the loop, the script prints the current value of power, followed by a space. This is done using the print function with the end parameter set to a space character to ensure that all powers are printed on the same line.
6. It then doubles the value of power by multiplying it by 2 to get the next power of 2.
7. If power exceeds n, the loop terminates, and the script ends without printing any further values.

---

powersoftwo.py ->
---
1. The script starts by importing the sys module to access command-line arguments. It then parses the first argument provided (after the script name) as an integer n. The script initializes a variable power to 1, which is the first positive power of 2.
2. A while loop is used to iterate as long as power is less than or equal to n. Inside the loop, the script prints the current value of power followed by a space, without adding a newline. This is to ensure all the powers of 2 are printed on the same line.
3. After printing, the script doubles the value of power by multiplying it by 2, which moves to the next power of 2. The loop continues until the value of power exceeds n.
4. If the provided n is negative, the while loop will not execute, and the script will output nothing, satisfying the condition that nothing should be written for negative n.

---
ramanujan.py ->
---
1. Import the sys module to access command-line arguments.
2. Check if a command-line argument (number n) is provided.
3. Initialize an empty list 'results' to store numbers matching Ramanujan's criteria.
4. Use four nested loops to iterate over combinations of a, b, c, and d less than the cube root of n.
5. For each combination, calculate the sum of cubes of a, b and compare it to the sum of cubes of c, d.
6. Ensure a, b, c, and d are distinct numbers.
7. If conditions are met, check if this number is already in 'results'. If not, add it.
8. Finally, print out each number with its two cube sum representations.

---
primecounter.py ->
---
1. Create a list of Boolean values "sieve" of length n+1 and initialize all values to True. The indices of the list will represent numbers, and the value at each index will indicate if that number is prime (True) or not (False).
2. Set sieve[0] and sieve[1] to False since 0 and 1 are not prime.
3. Start with the first prime number, which is 2. Iterate over each number i starting from 2 up to the square root of n.
   - If sieve[i] is True (indicating i is prime):
     a. Iterate over each multiple j of i (starting from i*i and incrementing by i) up to n.
     b. For each j, set sieve[j] to False.
4. Once the iterations are complete, the indices of the list "sieve" that hold the value True are the prime numbers.
5. Return the count of True values in the list "sieve".

---
montyhall.py ->
---
1. Randomly assign the prize to one of the three doors.
2. Simulate the contestant's initial choice.
3. Determine which door the host would open (a door not chosen by the contestant and without the prize).
4. If the contestant sticks with their initial choice, check if they've won.
5. If the contestant switches, they switch to the remaining unopened door, and then check if they've won.