# COS 125 – HMWK5 – Dice Game Scoring Engine

You can reference the textbook (in BrightSpace), class materials, and tutoring. **Do not use AI or concepts outside of what have been taught in the course so far**. Include the comment header in all homework assignment files, specify any collaboration you had.

## I. General Rules & Academic Integrity

- **Topics:** Practicing elements and concepts covered in class: Getting user input, data type conversion, manipulating lists/strings, implementing decision/control structures (if/else), loops, complex data types, and functions. Do not use breaks, global values, modules other than random.
- **Concepts Used: Only use concepts presented in the course to-date.** Using advanced concepts or concepts from outside the course does not demonstrate understanding of the required material and may result in a zero for the assignment. Asking AI, online searches, or people not associated with the course is not allowed and many times leads you to incorrect solutions.
- **Resources:** You may reference the textbook, class materials, and tutoring (if you use a tutor, list their name in collaboration).
- **Academic Integrity: Do not use AI.** Specify any collaboration you had within the comment header, if you went to Boardman 138, state that.
- **Start early. Planning out the program helps.**

## II. Submission Requirements

- **File Name:** yourName_dice.py (replace "yourName" with your actual name, e.g., lauraGurney_dice.py).
- **Comment Header:** Include the required comment header in your file and ensure data within the header is updated.
- **Folder Name:** Place the .py file into a folder named yourName_diceGame.
- **Submission:** Keep python files inside your assignment folder. Compress the folder (.zip) and submit the compressed file.

This assignment requires you to build the core scoring logic for a turn in a simplified dice game using Python **lists** and **dictionaries** and **functions**. The dynamic nature of the scoring rules will require mastery of loops, conditional logic, and dictionary manipulation.

## Core Data Structures:

Begin by planning what you will need, how things are processed, and passed around. Read the entire assignment first. Draw a flowchart of the processes in the program, how values will be passed, and what values will be modified.

Also indicate all logic and inputs/outputs required. The program will create a new points value assignment for different dice "face values" each time the game is run. The points for the dice face values will be based on the frequency of each number as it appears during 50 rolls.

1. **Submit Planning:** Submit a flowchart, outline, or other planning document showing how you planned out the program. Images of whiteboard drawings, etc. will be accepted. Place in the submission folder with the .py file.
2. **Determination frequency:** A process to determine how many times each dice face shows when rolled 50 times. This should count the number of times each face (number) appears in a turn of 50 rolls of one dice. Use this to develop the Count (Frequency) of dice numbers. This should be processed inside a function and the results passed out as a data type that can be referenced (e.g. When checking the count of the number 6 a value should be accessible from a dictionary of face value and frequencies).
3. **Determine point values**: Create a process that will take the frequencies of each dice face value and evaluate it to calculate points for each dice face value. This is dynamic as it is calculated from the 50 rolls. It is NOT calculated at the same time as the 50 rolls. Create a function that takes the frequencies in and calculates points based on frequencies. Return a points valuation to the rest of the program.
4. **Dice Hand:** Represents 5 randomly generated whole number representing 5 dice rolled.
5. **Scoring function:** Create a function that will take the values rolled for a round and reference the points values assigned to determine the valuation of the "hand" of dice. Return the total points scored for this round of dice rolls. Create the function so the number of dice can be any number of values and will work for a dice hand of 5 values or 100 values, etc. (abstraction).
6. **Game play:** In the main() function ask the user how many dice they're using in the game. Roll that many randomly generated values between 1 and 6 (representing dice faces). Store the values in a list. Use the main() to pass values to and receive values from functions to determine the score of the round.
7. **Winning:** If the randomly rolled dice game total scores a total value that is a multiple of 7 then they win, otherwise the computer wins.
8. **Repeat:** Allow the user to play again if they'd like to. You do not need to keep score between games.

## Dynamic Point Assignment Logic

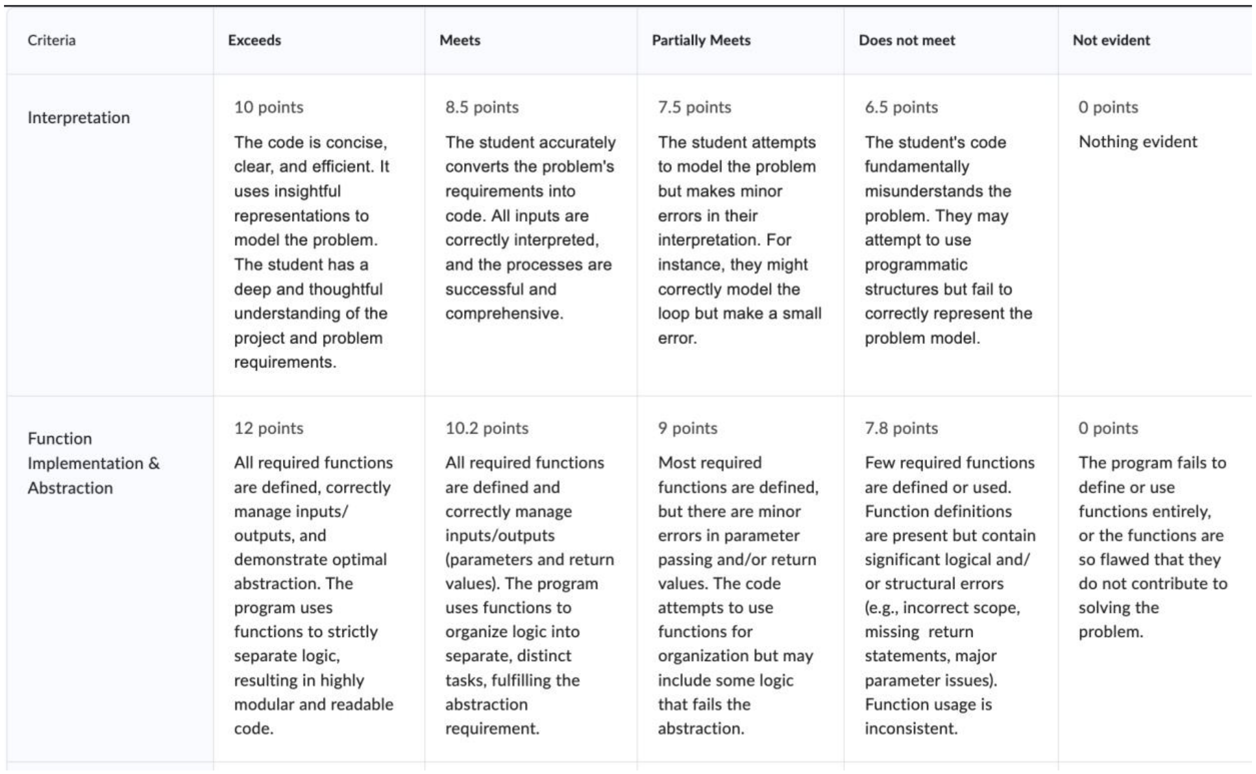
Dynamically generate numbers between 1 and 6 for 50 times. Store the values generated. Evaluate how many of each number 1 to 6 were generated and determine frequency.

Frequency is assessed by the count of the number of times the number is generated during the 50 times.

| **Rank** | **Frequency** | **Dice Value** | **Bonus Multiplier** |
|---|---|---|---|
| **1st** | Highest Count | (e.g., 3) | 1x (No Bonus) |
| **2nd** | 2nd Highest Count | (e.g., 4) | 2x |
| **3rd** | 3rd Highest Count | (e.g., 6) | 3x |
| **4th** | 4th Highest Count | (e.g., 3) | 4x |
| **5th** | 5th Highest Count | (e.g., 1) | 5x |
| **6th** | Lowest Count | (e.g., 2) | **6x (Highest Bonus)** |

---

**Grading will be based on:**
Implementing course content correctly.
Implementing logic, loops, and data types properly.



| Criteria | **Exceeds** | **Meets** | **Partially Meets** | **Does not meet** | **Not evident** |
|---|---|---|---|---|---|
| Interpretation | 10 points<br>The code is concise, clear, and efficient. It uses insightful representations to model the problem. The student has a deep and thoughtful understanding of the project and problem requirements. | 8.5 points<br>The student accurately converts the problem's requirements into code. All inputs are correctly interpreted, and the processes are successful and comprehensive. | 7.5 points<br>The student attempts to model the problem but makes minor errors in their interpretation. For instance, they might correctly model the loop but make a small error. | 6.5 points<br>The student's code fundamentally misunderstands the problem. They may attempt to use programmatic structures but fail to correctly represent the problem model. | 0 points<br>Nothing evident |
| Function Implementation & Abstraction | 12 points<br>All required functions are defined, correctly manage inputs/ outputs, and demonstrate optimal abstraction. The program uses functions to strictly separate logic, resulting in highly modular and readable code. | 10.2 points<br>All required functions are defined and correctly manage inputs/outputs (parameters and return values). The program uses functions to organize logic into separate, distinct tasks, fulfilling the abstraction requirement. | 9 points<br>Most required functions are defined, but there are minor errors in parameter passing and/or return values. The code attempts to use functions for organization but may include some logic that fails the abstraction. | 7.8 points<br>Few required functions are defined or used. Function definitions are present but contain significant logical and/ or structural errors (e.g., incorrect scope, missing return statements, major parameter issues). Function usage is inconsistent. | 0 points<br>The program fails to define or use functions entirely, or the functions are so flawed that they do not contribute to solving the problem. |

| | | | | | |
|---|---|---|---|---|---|
| Logical Structures | 12 points<br>The program correctly uses logical decision structures (e.g., if/elif/else) to handle all possible scenarios, including edge cases. The use of these structures is efficient and well-placed. | 10.2 points<br>The program correctly uses if/elif/else to handle the core logic and ensure the program doesn't crash on invalid input, if applicable. | 9 points<br>The program attempts to use conditional logic but makes minor errors that may cause the program to fail or produce incorrect logical outputs under certain conditions. | 7.8 points<br>The program fails to use conditional logic. | 0 points<br>Nothing evident |
| Looping and Iterations | 12 points<br>The student implements looping structures correctly to complete required processes perfectly. Loop(s) terminate correctly and efficiently, and the variables are updated within and nest, if used is correctly implemented. | 10.2 points<br>The student correctly implements loop(s), functions as intended, terminating when the initial cost is covered. Minor issues may be present. | 9 points<br>Loop(s) not implemented or contain logical errors (e.g., infinite loop, incorrect termination condition) that prevent from functioning properly. Not implemented correctly to solve specified challenge. | 7.8 points<br>The student fails to implement a while loop or uses an incorrect loop type and or structure. | 0 points<br>Nothing evident |
| Data Handling | 10 points<br>All inputs are stored in self-documenting variables, and the program demonstrates robust data conversion and handling of program data and user inputs. | 8.5 points<br>The program correctly prompts the user for all inputs, stores them in descriptive variables, and converts data types correctly for calculations. Handles program data properly is minimal errors | 7.5 points<br>The program prompts for some but not all inputs, or the variable names are not clear. There may be errors in data type conversion that led to incorrect output. | 6.5 points<br>The program does not prompt for all inputs or fails to store them correctly. Data type conversion is either not attempted or is consistently incorrect. | 0 points<br>Nothing evident |
| Communication & Formatting | 8 points<br>The program's output is highly effective and easy to read. It uses excellent formatting (\t, \n) to display results of program clearly and concisely. | 6.8 points<br>The program successfully adheres to all output specification standards. The readability and user experience output could be more clearly formatted and effective. | 6 points<br>The program's output does not effectively meet the overall purpose of the program / problem. The output is presented without a clear explanation of what is meant, lacking formatting and clarity. | 5.2 points<br>The program fails to provide adequately formatted output. It may use vague, hard to read, or the output is completely absent or incomprehensible. | 0 points<br>Nothing evident |

| | | | | | |
|---|---|---|---|---|---|
| File & Folder Organization | 8 points<br>The file and folder naming and organization are professional, clear, and logical. | 6.8 points<br>The file and folder naming and organization are correct. | 6 points<br>The file or folder naming is slightly off, but the file is still recognizable. | 5.2 points<br>The file and folder naming and organization are incorrect, making the file difficult to find or identify. | 0 points<br>Nothing evident |
| Header Comments Zero grade if not included | 8 points<br>The student includes a comment header that is well-formatted and includes all of the required information along with a brief description of the program's functionality. | 6.8 points | 6 points<br>Comment header in present without required information and is significantly lacking relevant information | 5.2 points | 0 points<br>Nothing evident - Zero for assignment |
| Planning Documentation | 10 points<br>The documentation (flowchart/ pseudocode) is fully comprehensive, accurately mapping all functions, data flow, variable modifications, and complex logic. It demonstrates an exceptional understanding of the problem structure *before* coding. | 8.5 points<br>The documentation is complete, clearly outlining the main functions, the passing of primary data structures, and the general sequence of the program. It accurately addresses the required inputs and outputs for core functions. | 7.5 points<br>The documentation is present but incomplete or lacks clarity. It might skip key functions or fail to clearly indicate handling of data and processes between steps. | 6.5 points<br>The documentation is minimal or confusing. It fails to map the core dependencies or uses unclear symbols/ formats, making it difficult to follow the intended program flow. | 0 points<br>No planning documentation (flowchart, pseudocode, or design diagram) is submitted or provided. |
| Program Comprehensiveness & Adherence to Instructions / Problem Model | 10 points<br>The submission is complete, meeting all functional, structural, and abstract requirements (e.g., all prompts answered, all code functions, all parts of the program model addressed). All submission rules are followed perfectly. | 8.5 points<br>The submission meets all core functional requirements and follows all explicit submission and academic integrity rules (e.g., correct file type, required components present, no plagiarism). Minor, non-critical aspects may be missing or slightly flawed. | 7.5 points<br>The submission achieves most core functional requirements but fails one or more critical components (e.g., a major section is missing, a required function doesn't work, or the core logic/process is flawed). | 6.5 points<br>The submission is missing multiple critical functional components and/or shows a clear violation of a major instruction(s). The submission is incomplete or unusable. | 0 points<br>The submission fails to open/run, or it violates a core academic rule. Most functionality is absent. |