CS1026B Assignment 2: Games

Due: **Tuesday, February 10th, 2026 @ 11:59 PM ET**

Change Log

- 2026-01-14 Initial post, no changes yet.

Table of Contents

Late Penalty

Late submissions will be given a **zero grade** if you have insufficient late coupons remaining.

> ⚠ Any resubmissions past the due date will be considered late and will require late coupons.

Assignments will be accepted up to 3 days late, but **only if you have at least one late coupon remaining for each day late** (see the course syllabus for full details on how late coupons work).

If you submit late and do not have enough late coupons remaining, a **zero grade** will be given for the assignment (there is no partial late mark).

No action is required to submit late. Simply submit late and your late coupons will be applied by the TA during marking.

▲ Back to Top

⚖ Academic Integrity

By submitting this assignment, you agree that you have read and understood the academic dishonesty policy for this course covered in both the course syllabus and the academic dishonesty document. This includes:

- Assignments must be completed individually, without any help from others.
- No copying solutions, formulas, files, or written answers from other students.
- No sharing of any part of your work with other students.
- No unauthorized use of generative AI tools (e.g. ChatGPT, Copilot, etc.). You can not use generative AI for this assignment.

> ℹ Suspected violations will be reported to the integrity committee and receive a zero grade.

▲ Back to Top

🎯 Learning Objectives

1. Use conditional statements (i.e., `if`, `elif`, and `else`).
2. Implement loops, both `for` and `while`.

3. Use control statements (e.g., `break` and `continue`).
4. Use both relational (e.g., ==, `!=`) and logical operators (e.g., `and`, `or`).
5. Read and validate user input.
6. Use string formatting to produce clean output.
7. Manipulate strings using concatenation and slicing.
8. Generate random numbers, using seeding for reproducibility.
9. Submit to Gradescope successfully: upload the correct `.py` files, interpret autograder feedback, and resubmit files after addressing any issues.
10. Demonstrate basic code quality: meaningful variable names, basic whitespace/indentation, and comments.

▲ Back to Top

Assignment Instructions

> Make sure that you read over the Non-Functional Specification before you start coding. It contains requirements for comments, variable names, and style you need to follow.

For this assignment you are to submit ***two*** separate Python programs:

- `character.py` (based on Part 1)
- `moo.py` (based on Part 2)

Each part of this assignment will provide a specification for what these programs will do, any input they will take, and the output they will provide. To obtain full marks from the autograder, you are required to follow the specification and example input/output exactly.

Keep in mind that **you can resubmit your code to the Gradescope autograder an unlimited number of times** leading up to the due date. Don't be afraid to resubmit your code often to test it, but we also highly recommend testing your code locally. Only the most recent resubmission will count for marks. **You must ensure you submit both programs in your final submission**. Any missing programs will receive a zero grade.

Introduction

In this assignment you'll build a program that can be used with a tabletop roleplaying game (TTRPG) (e.g., *Dungeons & Dragons*) and another that is based on an old game called "Bulls and Cows." The latter may be more familiar to you as being similar to *Wordle* or *Mastermind*. Both of these programs will build upon the skills you learned in the first assignment, adding in some new skills like loops, conditions, and randomness. As a result of these additional skills, you should expect these programs to take a little longer to write.

These programs will require you to think through your code logically in order to plan the conditions and loops necessary to get the programs working. It is a good idea to start thinking through the logic prior to writing the code, as it is easy to write yourself into a corner; a little work ahead of time can save a lot of work later on.

Once you're done, you'll be able to roll up several characters for a tabletop roleplaying game, and you'll also have a game that you can play against your computer.

▲ Back to Top

Part 1: Tabletop Roleplaying Game Character Generator

In honour of the ending of *Stranger Things*, for Part 1 you will build a basic **TTRPG Character Generator**, in a file named **`character.py`**.

In a TTRPG, players create characters that have **ability scores**. These ability scores determine a character's ability to succeed at the actions that players wish to perform during the game. The scores usually range from 3 to 18, with higher numbers being better. The higher the number, the better the bonus a player gets to any of the rolls that they make that require that ability. Low numbers can provide negative "bonuses." Bonuses impact the likelihood that the player will succeed at what they are attempting to accomplish in the game. Each character also has a **class** that identifies what skills that character has (e.g., fighters are good at combat, wizards are good at using magic). The character's class is usually chosen based on the highest ability score.

This program takes no input. Instead, it uses randomness to simulate rolling dice, applies those roles to character ability scores, and then it suggests a character class that benefits from the highest ability score that was rolled. It then identifies the bonus generated from the highest ability score.

> **VERY IMPORTANT:** In order for the autograder to function correctly, the only function from the `random` module you should use in this program is `random.randint()`. It is also very important that you follow the instructions related to randomness very carefully, or you may end up running into issues with the autograder.

1.1 Generate Ability Scores

Your program will begin by rolling dice (i.e., generating random numbers) to **generate ability scores** for a character.

1. **Roll the Dice**:
    - Roll four six-sided dice (i.e., generate a number between 1 and 6 four separate times).
    - Determine which of the rolls is the lowest roll.
    - Each ability score will be based on the three best dice rolled. Sum the dice and drop the lowest roll. For example, if your program rolls `4, 2, 6, 3`, then the sum will be `4 + 6 + 3 = 13`, with the 2 being dropped. If the lowest roll appears in more than one roll, the program should still only drop one roll.
    - Add the ability score (i.e., the sum that was just calculated) to a string (see below) that will store all six ability scores. Each ability score should be two digits to make the string easy to parse (i.e., read). This means that if the ability score only contains one digit, then add a `0` to it. This means that the ability score `7` should become `07`.
    - Repeat the above steps for all six ability scores.
    - Once six ability scores have been generated, there should be a 12-character string that holds all of the scores. For example, if the generated ability scores were `7, 12, 9, 14, 17, 11`, they should be represented in the following string: `"071209141711"` (because single-digit scores should have had a `0` added to them).
    - **IMPORTANT NOTE:** You must use a string. You are not allowed to use a list, a tuple, nor any data structure other than a string, as one of the skills you are expected to practice in this assignment is string manipulation.
2. **Store the Ability Scores**:
    - Now your program will store the values in six variables, one for each ability score. The ability scores are, in alphabetical order, **charisma, constitution, dexterity, intelligence, strength**, and **wisdom**.
    - The program should read the string, separating each of the two-digit numbers and converting them to integers. (The `0` will be dropped when the strings are converted to integers, and that's what we want.)
    - Each of those integers should be stored in an ability score. For example, the first two digits from the string become the charisma score. The next two represent constitution, etc. until all of the ability scores have a value. If the string were `"071209141711"`, then these are the ability scores for the character:
        - Charisma → 7.
        - Constitution → 12.
        - Dexterity → 9.
        - Intelligence → 14.
        - Strength → 17.

- Wisdom → 11.

Hints

1. String slicing will help when it comes time to unpack the string that you have created. It will allow you to access substrings of the ability score string. You can read more about string slicing here.
2. You may also find f-string formatting will help you out. There is a nice little cheat sheet here. There is a link on that page that will take you to more information on f-strings, too.

1.2 Determine the Character's Class and Bonus

Now that the ability scores have been generated, we want to decide which class this character should belong to and identify their key ability.

1. **Determine the Highest Ability Score**:
   - Compare the ability scores to determine which is the highest score. If there is a tie, the one that comes first alphabetically is considered the highest ability score.
2. **Determine the Character's Class**:
   - Using the character's highest ability score, determine their class using the following list:
     - Charisma → Sorcerer
     - Constitution → Barbarian
     - Dexterity → Rogue
     - Intelligence → Wizard
     - Strength → Fighter
     - Wisdom → Druid
3. **Determine the Character's Ability Bonus**:
   - Determine the bonus that should apply to the highest ability score. The number on the left is the ability score, and the number on the right is the bonus that score generates:
     - 18 → +4
     - 16 or 17 → +3
     - 14 or 15 → +2
     - 12 or 13 → +1
     - 10 or 11 → +0
     - 8 or 9 → -1
     - 6 or 7 → -2
     - 4 or 5 → -3

- 3 → -4

1.3 Output

In its output, the program should highlight **the character's highest ability score**, **the recommended class** (the class should be in uppercase letters), **all of the ability scores**, and **the bonus that the character receives**. Several examples of the program being run can be seen below.

Make sure you match the formatting and spelling of the output shown in the following examples. All spacing, spellings, line breaks, capitalization, and punctuation should be the same.

Example 1.1 - A Sorcerer

```
The character's highest ability score is Charisma: 17.
The recommended character class is: SORCERER.

The Sorcerer has the following stats:
Charisma:        17
Constitution:    15
Dexterity:       10
Intelligence:    14
Strength:        12
Wisdom:          15
They get a +3 bonus to Charisma.
```

Green text *is user input,* ***white text*** *is program output.*

Example 1.2 - A Wizard

```
The character's highest ability score is Intelligence: 16.
The recommended character class is: WIZARD.

The Wizard has the following stats:
Charisma:        15
Constitution:    15
Dexterity:       12
Intelligence:    16
Strength:         8
Wisdom:          15
They get a +3 bonus to Intelligence.
```

Green text *is user input,* ***white text*** *is program output.*

Example 1.3 - A Druid

```
The character's highest ability score is Wisdom: 18.
The recommended character class is: DRUID.

The Druid has the following stats:
Charisma:         7
Constitution:     7
Dexterity:       13
Intelligence:    13
Strength:        10
Wisdom:          18
They get a +4 bonus to Wisdom.
```

Green text is user input, white text is program output.

Hint

The output of this program should be random, so the output you see may not (in fact, it almost certainly won't) match the output from the examples above. Test your code multiple times to ensure that there are different outcomes and that those outcomes make sense.

▲ Back to Top

Part 2: Bulls and Cows

Many of you are familiar with games like Wordle, but that style of game has existed for much longer than you might think. Mastermind is a similar game, but instead of guessing words, like in Wordle, you guess coloured pegs on a board. In both games, you are trying to guess a hidden value. When you identify one of the letters or coloured pegs, you are told whether or not that element is in the puzzle and whether or not your guess has any value it in the exact right position.

An earlier version of this game called "Bulls and Cows" has you guessing a series of numbers. When you guess a number correctly, it's a cow. If you guess its position correctly, it's a bull. In this part, you're going to build a working Bulls and Cows game that you can play based on a random, five-digit value generated by the computer. The goal is to solve the puzzle in six or fewer guesses.

> **VERY IMPORTANT:** In order for the autograder to function correctly, the only function from the `random` module you should use in this program is `random.randint()`. It is also very important that you follow the instructions related to randomness very carefully, or you may end up running into issues with the autograder.

2.1 Generate the Puzzle

Your program will generate a puzzle composed of five unique digits.

1. **Generate Digits:**
    - The puzzle generated must be five digits long. Each of the digits must be generated randomly, in the range 1 to 9, and each digit must be unique, so you will need to generate each digit individually.
    - The digits should be stored in a string as you go. If the digit already appears in the string, a different digit should be generated.
    - Once the program has generated a five-digit string composed of five unique digits (e.g., `"23561"`), the puzzle is ready.

2.2 Playing the Game

The program will now allow a player (usually you) to attempt to solve the puzzle.

1. **Validating User Input**
    - The user should begin the game with six guesses.
    - The program will prompt the user to enter a five-digit number. This prompt will indicate, in parentheses, how many guesses the player has remaining. For example, if the user is making their first guess, then they have six guesses remaining. In this case, they should see the following prompt:

```
Guess a five-digit number (6 guesses remaining):
```

    - The user will now guess the value.
        - If the user enters a value that contains any character that is not a digit from 1 to 9, then they should see the following:

```
Guess a five-digit number (6 guesses remaining): hello
Please only include the digits 1 through 9 in your guess.
This will not count against your number of guesses.
```

- The user should then be prompted again, and a guess shouldn't be counted against them.
- If the user enters a value that contains more (or fewer) than five characters, then they should see the following:

```
Guess a five-digit number (6 guesses remaining): 123456
Please enter a five-digit number.
This will not count against your number of guesses.
```

- The user should be prompted to enter a valid number. Their guess should not count against them.
- If the user enters a value that contains the same digit more than once, then they should see the following:

```
Guess a five-digit number (6 guesses remaining): 12341
Any digit can only appear once in your guess.
This will not count against your number of guesses.
```

- If at any point in their guessing, the user enters invalid input, they should be prompted with the above messages (depending on the error), and they should not have their guess counted against them.

Hint

When validating whether or not a string character is a number, you can use the `isdecimal()` string method. You can read more about this method here.

2. **Playing a Round**
   - A **round** of the game consists of the player making a guess and the program telling them what is correct about their guess.
   - While playing the game, a string that matches the current state of the user's knowledge should be visible after each guess. At the beginning of the game, this string is just a series of five hyphens (i.e., `-----`) representing potential positions. We will call this string of hyphens the **tracking string**.
   - If the user has entered a valid five-digit number, then that value should be compared to the puzzle value that was generated in Part 2.1. If any of the digits in the user's guess matches the number and position of that number in the puzzle, then **the user has found a bull.**
     - For example, if the puzzle solution is 23561 and the user enters 24578, then 2 and 5 are both bulls because those digits are in the same position in both the puzzle solution 23561 and the player's guess 24578.

        - When a bull is found, the tracking string should be updated to identify its position. For example, after identifying 2 and 5, the tracking string should be changed from `-----` to `2-5--` so that the player can see their bulls.
    - If the player's guess identifies a digit in the puzzle solution but that digit is in a different position, then **the user has identified a cow.**
        - For example, if the puzzle solution is `23561` and the user enters `37892`, then 3 and 2 are both cows.
3. **A Round Ends**
    - A round ends when the player is informed of how many bulls and cows they have identified.
    - If the player does not find any bulls, they should see the following message: `You did not find any bulls.`
    - If the player found any bulls, they should be told how many they found: `You found 2 bull(s): 2 and 5.`
    - If the player does not find any cows, they should see the following message: `You did not find any cows.`
    - If the player found any cows, they should be told how many they found: `You found 3 cow(s): 3 and 4 and 6.`
    - Following the bulls and cows messages, the updated tracking string should be displayed: `2-5---`.
    - Examples of all of these conditions can be found in the examples provided in Part 2.3.
    - After the round ends, the user will be prompted to enter another five-digit value, but the number of guesses remaining should be decreased by 1.
4. **The Game Ends**
    - The game ends under **two possible conditions:**
        1. The player solves the puzzle.
        2. The player uses six guesses (i.e., they have played six rounds), and the puzzle remains unsolved.
    - If the player solves the puzzle, they should be congratulated, told how many guesses it took, and the answer should be shown. (See Example 2.2.)
    - If the player fails to solve the puzzle, they should be told that they did not solve the puzzle, and the answer should be shown. (See Example 2.3.)

2.3 Output

Make sure you match the formatting and spelling of the output shown in the following examples. All spacing, spellings, line breaks, capitalization, and punctuation should be the same. There is some output that is mentioned above that is not displayed in these examples, but it is described above.

Example 2.1 - Invalid Input

```
Guess a five-digit number (6 guesses remaining): hello
Please only include the digits 1 through 9 in your guess.
This will not count against your number of guesses.

Guess a five-digit number (6 guesses remaining): 123456
Please enter a five-digit number.
This will not count against your number of guesses.

Guess a five-digit number (6 guesses remaining): 01234
Please only include the digits 1 through 9 in your guess.
This will not count against your number of guesses.

Guess a five-digit number (6 guesses remaining): 12341
Digits can only appear once in your guess.
This will not count against your number of guesses.

Guess a five-digit number (6 guesses remaining):
```

Green text *is user input,* ***white text*** *is program output.*

Example 2.2 - Correct Solution

```
Guess a five-digit number (6 guesses remaining): 12345
You did not find any bulls.
You found 3 cow(s): 1 and 2 and 4.
-----

Guess a five-digit number (5 guesses remaining): 41267
You did not find any bulls.
You found 4 cow(s): 4 and 1 and 2 and 7.
-----

Guess a five-digit number (4 guesses remaining): 74128
You found 1 bull(s): 4.
You found 4 cow(s): 7 and 1 and 2 and 8.
-4---

Guess a five-digit number (3 guesses remaining): 84721
You found 3 bull(s): 4 and 7 and 1.
You found 2 cow(s): 8 and 2.
-47-1
```

```
Guess a five-digit number (2 guesses remaining): 24781
Congratulations! You solved the puzzle in 5 guess(es)!
The answer was 24781.
```

Green text is user input, white text is program output.

Example 2.3 - A Losing Game

```
Guess a five-digit number (6 guesses remaining): 12345
You found 1 bull(s): 1.
You found 3 cow(s): 2 and 3 and 5.
1----

Guess a five-digit number (5 guesses remaining): 12345
You found 1 bull(s): 1.
You found 3 cow(s): 2 and 3 and 5.
1----

Guess a five-digit number (4 guesses remaining): 12345
You found 1 bull(s): 1.
You found 3 cow(s): 2 and 3 and 5.
1----

Guess a five-digit number (3 guesses remaining): 12345
You found 1 bull(s): 1.
You found 3 cow(s): 2 and 3 and 5.
1----

Guess a five-digit number (2 guesses remaining): 12345
You found 1 bull(s): 1.
You found 3 cow(s): 2 and 3 and 5.
1----

Guess a five-digit number (1 guesses remaining): 12345
You found 1 bull(s): 1.
You found 3 cow(s): 2 and 3 and 5.
1----

Sorry, but you did not solve the puzzle.
The answer was 13529.
```

Green text is user input, white text is program output.

Hints

1. The game must keep running until one of the two end conditions is met. Think about how you could keep the game going when you don't know how many guesses will be made.
2. While you're writing the program, it is probably a good idea to have the puzzle solution visible. This will make it easier to see what you are doing.
3. The output of this program should be random, so the output you see may not (in fact, it almost certainly won't) match the output from the examples above. Test your code multiple times to ensure that there are different outcomes and that those outcomes make sense.

▲ Back to Top

Non-Functional Specification and Other Details

In addition to the other requirements given in the previous sections, your program must also fulfill the following requirements:

- **Document each file:** You MUST provide a comment at the top of **all** Python files (files ending in .py) that contains 1) your full name, 2) your student number, 3) your uwo username (the same one you use to log in to OWL), 4) the date the file was created, and 5) a description of what this specific file does. This must be provided at the top of all files, and the description should be updated to describe the specific file it is in (it can not be the same in all files).
- **Other comments throughout your code:** You must also include comments in your code for any lines that are not easily understood. You just must use good judgment in what lines require comments, the marker may remove marks for over or under commenting.
- **Match the example text:** The text contained in your messages and prompts to the user should match the examples given, and all input must be taken in the same order.
- **Restrictions on randomness:** Only use `random.randint()` to generate random numbers. This will simplify the autograder behaviour.
- **Follow good programming practices:** All included code must have a clear purpose and function. Including code that has no purpose, such as variables that are never used, may lead to marks being removed. You should understand each line of your program and

each line should have a purpose. Marks may also be removed for other poor programming practices such as inappropriate recursion, redundant code, not making use of the specified functions when appropriate, or other code quality issues that demonstrate a lack of understanding.

- **Follow good style:** As much as possible, you should follow the PEP8 style guide. Both the autograder and your IDE should provide hints about style violations.
- **Python 3.12:** Your code should work correctly in Python 3.12 and newer. These are the versions of Python the auotgrader will be compatible with.
- **No hardcoding:** You may not hardcode your returns, outputs, or writing to files. Your code should work correctly for any valid input. Attempts to fool or trick the auotgrader will result in the marker manually assigning a zero grade for that test.

▲ Back to Top

🎓 Assessment

The assignment will be marked as a combination of your auto-graded tests and manual grading of your code logic, comments, formatting, style, etc. Marks will be deducted for failing to follow any of the specifications in this document, not documenting your code with comments, using poor formatting or style, hardcoding, or naming your files incorrectly.

Marking Scheme:

- **Autograded Tests:** 80 points
- **Header comment including your name, student ID, username, creation date, and description of file:** 8 points
- **Descriptive in-line comments throughout code:** 5 points
- **Meaningful variable names:** 2 points
- **Good programming practices *(see Non-Functional Specification)*:** 5 points

Total: 100 points

Note: *Marks from the autograded tests may also be manually removed by the marker for violating any of the requirements given in the Non-Functional Specification, hard coding solutions, or failing to implement any features as specified.*

▲ Back to Top

Submission Instructions

You must upload **all files for all parts** (`character.py` and `moo.py`) to the Gradescope autograder. For a tutorial on how to use Gradescope to submit a code assignment, please review this video.

You may resubmit your assignment to Gradescope an unlimited number of times before the submission deadline. Gradescope will give you some feedback from the autograder each time you submit.

Any submissions after the due date will be considered late and subject to the late policy. Do not resubmit after the due date if you do not want to use late coupons.

Your final submission **must** include the files for both parts of this assignment. Any missing files will result in a zero grade for that part of the assignment. Only your most recent submission will count for your assignment grade.

Submissions will not be accepted via e-mail or through any other means. The autograder will only accept files with the correct file extensions. Make sure your files have one file extension and that they are named correctly. Any incorrectly named files will result in a zero grade for that part.

Remember, **you are required to check that the files you submit are correct and all autograder tests are passing**. Teaching assistants will only review your submission for items relating to the Non-Functional Specification when marking; they will not give marks back for failed autograder tests.

▲ Back to Top