Mullumbimby High School
Assessment Task

Student Name:	

Task Name:	D&D Character Creator		
Year:	11	**Faculty:**	TAS
Subject:	Software Engineering	**Teacher:**	Steedman
Date Assigned:	13/06/2025	**Date Due:**	25/07/2025
Weighting:	35%	**Total Mark:**	___ / 100

Outcomes to be Assessed:

SE-11-01 - describes methods used to plan, develop and engineer software solutions
SE-11-02 - explains how structural elements are used to develop programming code
SE-11-04 - applies safe and secure practices to collect, use and store data
SE-11-06 - applies tools and resources to design, develop, manage and evaluate software
SE-11-08 - applies language structures to refine code
SE-11-09 - manages and documents the development of a software project

Task Description:

As a software engineer, you are tasked to design and develop a D&D Character Creator program in Python. You will work through each stage of the software development cycle while demonstrating your skills with concepts such as object oriented programming, data types and structures, input validation, functions, loops, and lists.

Submission Instructions:

Hand in:
1. This completed workbook
2. Your project via GitHub Classroom

Absence / Misadventure Instructions:	Academic Integrity:	Acknowledgement:
All Stage 6 students have been issued with an Assessment Policy Handbook, which outlines the procedures relating to absence/illness/misadventure. This handbook can also be found on the school website. Students must submit an illness/misadventure form to the Head Teacher Senior Studies (Mrs Elliott) for foreseeable absences/misadventures before the task due date and immediately on return to school for unforeseeable absences / misadventures. The following penalties for late submission without an acceptable reason will apply: one day late - 20% deducted, two days late - 40% deducted. Three or more days late will be awarded zero.	Any instances of academic dishonesty may be interpreted as a non-attempt (failing grade). Please refer to the assessment policy booklet for more information. To ensure academic integrity, teachers may require students to submit their work using Google Docs or other platforms with version history tracking (or some other means of evidence of authorship). This enables teachers to verify that the content has been created by the student and not generated by an AI tool.	CLICK HERE OR SCAN

D&D Character Creator

Overview:

You are required to design and develop an object-oriented program that creates and manages Dungeons and Dragons (D&D) characters. Your program should demonstrate your understanding of object-oriented programming concepts and your ability to apply them in a practical context.'

Requirements:

1. Create a base **Character** class with attributes such as name, race, class, level, hit points, armour class and ability scores (strength, dexterity, constitution, intelligence, wisdom, charisma).

2. Create a **Dice** class that can be used to roll dice throughout the program

3. Create methods for:
    a. Rolling random stats for a character
    b. Leveling up a character
    c. Displaying character information

4. Create a user interface that allows users to:
    a. Create a new character
    b. View existing characters
    c. Manage an existing character

5. Implement error handling and input validation to ensure the program runs smoothly.

6. Use appropriate naming conventions and comment your code effectively.

7. Keep your code modular and simple by organising it into separate files and classes.

8. Make your program unique. Once you have the basics in, add in some of your own ideas. For example:
    a. Save and load characters
    b. Add an inventory system
    c. Add a weapon system to roll attacks and damage
    d. Roll initiative order for all of your characters

To further understand Dungeons and Dragons you may like to explore these links:

- DND 5th Edition - Community Wiki
- Roll20 - D&D 5th Edition

Specifications (15):

Requirements / Specifications Table (10)

From the requirements above, five have been chosen for you to research and detail further.
These are essential for understanding the workings of a D&D Character Creator program.

Requirements	Specifications
Ability Score Generation: Research the standard method for generating ability scores in D&D 5e (4d6 drop lowest). Explain the process in detail and/or provide pseudocode for how you would implement this in your program.	
Ability Score Modifiers: Research how ability score modifiers are calculated in D&D 5e. Explain the formula for determining modifiers from ability scores and/or write pseudocode for a function that calculates a modifier from a given ability score.	
Hit Point Calculation: Investigate how hit points are calculated in D&D 5e. Explain the process for determining hit points at 1st level and for subsequent levels. How would you calculate a character's hit points, taking into account their level, Constitution modifier, and class hit die (refer to the CLASS_INFO dictionary provided).	
Racial Bonus Application: Examine the RACE_BONUSES dictionary provided. Explain how you would use this information to apply racial bonuses to a character's ability scores. Pay particular attention to races like Human that give bonuses to all abilities, and Half-Elf that allows choice in ability score increases. Write pseudocode for a function that would apply these bonuses when a character is created.	
Armor Class Calculation: Explain how Armor Class (AC) is calculated in D&D 5e. Describe how Dexterity affects AC and/or write pseudocode for a function that would calculate a character's AC.	

IPO Diagram (5)

Create an Input-Process-Output (IPO) diagram for each method below:

Ability Score Generation		
Inputs	**Processing**	**Outputs**

Ability Score Modifiers		
Inputs	**Processing**	**Outputs**
ability_score		ability_score_modifier

Hints for IPO Diagrams:

- Start by listing all Inputs and Outputs clearly before detailing the Processing steps.
- In the Processing section, describe step-by-step actions that transform Inputs into Outputs.
- Use clear, concise language and focus on key operations without over-complicating.

Design (10)

Screen Design (5)

Create a screen design for the program using text. This should visually represent how the menu system will appear to the users as they go through the process of creating and managing a character. **Bold** any sample user input. Your screen design should include:

- Main menu
- Character creation process
- Character management menu
- Display of character information
- Any submenus (e.g., for managing weapons or inventory)

UML Class Diagram (5)

Create a UML Class Diagram for the Character class. This should include:

- Class name
- Attributes (including data types)
- Methods
- Any relationships with other classes (if applicable)

Use proper UML notation, including:

- '+' for public elements
- '-' for private elements
- Underline for static (class) variables/methods

Character

.

Additional Hints:

- Use proper data types for attributes (e.g., str for name, int for level)
- Include key methods like init, generate_abilities, calculate_hit_points
- Don't forget to include getters and setters if applicable

Development (60)

Your program should reflect a solid understanding of object-oriented programming concepts and demonstrate good coding practices. Focus on writing clean, efficient, and well-documented code. This project is an opportunity to showcase your skills in Python programming and problem-solving.

Object-Oriented Design (10):

- Implement a well-structured Character class with appropriate attributes and methods.
- Use encapsulation effectively (private attributes with public methods as needed).
- Implement additional classes as necessary (e.g., Dice, Weapon) with clear relationships.

Game Logic Implementation (10):

- Accurately calculate ability scores, modifiers, hit points, and armor class.
- Correctly apply racial bonuses to ability scores.
- Implement leveling up mechanism with appropriate stat increases.

Data Structure Usage (8):

- Effectively use the provided dictionaries (RACE_BONUSES, CLASS_INFO, etc.).
- Use appropriate data structures for managing character attributes and optional features.
- Store and manage multiple characters efficiently.

Control Structures and Error Handling (8):

- Use loops and conditional statements for character creation and management flow.
- Implement robust error handling for invalid inputs or unexpected scenarios.
- Ensure the program doesn't crash due to user errors.

User Interface (5):

- Create a clear and intuitive menu system for character creation and management.
- Display character information in a readable format.
- Provide clear prompts and feedback for user actions.

Code Quality and Style (5):

- Follow Python's PEP 8 style guide for coding conventions.
- Use meaningful variable and function names.
- Organize code into logical functions or methods for readability.

Documentation and Comments (5):

- Add comments explaining the purpose of functions and key sections of code.
- Include inline comments for complex logic or calculations.
- Provide a brief program header with authorship, purpose, and date.

Version Control (5):

- Use GitHub to manage and document the development process.
- Commit changes regularly with clear, descriptive commit messages.
- Demonstrate progression of project through commit history.

Testing and Debugging (10)

Testing Table (7)

Create a comprehensive testing table for your Character Creator. This table should document various test cases, including normal, boundary, and erroneous conditions, to ensure the game functions as intended.

ID	Test Case Description	Expected Output	Actual Output	Observations
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				

Debugging Process (3)

Identify and describe at least two bugs encountered during development or testing. For each bug, briefly explain the issue, its cause, and the steps taken to fix it. Include how you verified that the fix resolved the problem without introducing new issues.

Evaluation (5)

Critically assess your D&D Character Creator, focusing on its alignment with object-oriented principles, your success in implementing complex D&D mechanics, and the overall quality of your solution. Reflect on challenges faced, lessons learned, and potential future enhancements to showcase your growth as a developer. For example:

- **Requirements Alignment:** Assess how well your final program meets the initial requirements and specifications. Discuss any features that were implemented successfully, partially, or not at all.
- **Object-Oriented Design Reflection:** Evaluate the effectiveness of your object-oriented approach. Discuss the strengths and weaknesses of your class structure and how well it represents the D&D character creation system.
- **Challenges and Solutions:** Identify the most significant challenges faced during development, particularly related to implementing D&D rules or OOP concepts. Explain how you overcame these challenges and what you learned from the process.
- **Code Quality and Efficiency:** Reflect on the quality and efficiency of your code. Discuss any areas where you believe your implementation is particularly strong or could be improved.
- **Future Improvements:** Suggest potential enhancements or extensions to your program. Consider how you could expand the system to include more D&D features or improve its usability.

Marking Rubric

Requirements & Specifications					
Criteria	Basic	Limited	Effective	Highly Effective	**Mark**
Requirements/ Specifications Table	The table includes only basic information for 1-2 D&D mechanics. Explanations lack detail and pseudocode is missing or incorrect. 1-3	The table covers 3-4 D&D mechanics with some detail. Explanations are present but may lack depth. Pseudocode is attempted but may have errors. 4-6	The table covers all 5 D&D mechanics with clear explanations. Pseudocode is provided for most items and is generally correct. 7-8	The table provides comprehensive details for all 5 D&D mechanics. Explanations are thorough and insightful. Pseudocode is correct, efficient, and well-commented. 9-10	/10
IPO Diagrams	Only one diagram is attempted. Inputs, processing, and outputs are unclear or incorrect. 1	Both diagrams are attempted but may have errors or lack detail in one or more sections. 2	Both diagrams are complete with correct inputs, processing steps, and outputs. Minor details may be missing. 3-4	Both diagrams are comprehensive and accurate. Inputs, processing steps, and outputs are clearly defined and show a deep understanding of the mechanics. 5	/5

Design					
Criteria	Basic	Limited	Effective	Highly Effective	**Mark**
Screen Design	Basic text representation of menu system. Missing several key elements. User input not highlighted. 1	Includes most required elements but may lack clarity or detail. Some user inputs highlighted. 2	Clear representation of all required elements. User inputs consistently highlighted. Good flow between menus. 3-4	Comprehensive and intuitive design covering all required elements plus additional helpful features. All user inputs clearly highlighted. Excellent menu flow and organization. 5	/5
UML Class Diagram	Basic attempt at UML diagram. Missing several key elements or using incorrect notation. 1	Includes most required elements but may have errors in notation or missing some attributes/methods. 2	Correct UML diagram with all required elements. Proper notation used. May have minor omissions. 3-4	Comprehensive and accurate UML diagram. Includes all required elements plus additional relevant attributes/methods. Perfect use of UML notation. 5	/5

Development					
Criteria	Basic	Limited	Effective	Highly Effective	**Mark**
Object Oriented Design	Basic Character class implemented with minimal attributes and methods. Little to no encapsulation. No additional classes. 1-3	Character class with some appropriate attributes and methods. Basic encapsulation. Additional classes may be present but poorly integrated. 4-6	Well-structured Character class with appropriate attributes and methods. Good use of encapsulation. Additional classes (e.g., Dice, Weapon) implemented with clear relationships. 7-8	Exceptional Character class design and encapsulation. Additional classes expertly implemented and integrated. Advanced OOP concepts (e.g., inheritance, polymorphism) used effectively where appropriate. 9-10	/10
Program Logic	Basic calculations attempted but with significant errors. Racial bonuses and leveling up not implemented or incorrectly implemented. 1-3	Most calculations implemented with some errors. Racial bonuses applied inconsistently. Basic leveling up implemented. 4-6	All required calculations correctly implemented. Racial bonuses applied correctly. Leveling up mechanism functions as expected. 7-8	All calculations implemented perfectly. Racial bonuses and leveling up mechanism are robust and handle all edge cases. Additional game logic features implemented beyond requirements. 9-10	/10
Data Structures	Minimal use of provided dictionaries. Inefficient data structures for character management. 1-2	Some use of provided dictionaries. Basic data structures used for character management. 3-4	Effective use of all provided dictionaries. Appropriate data structures used for character management and optional features. 5-6	Optimal use of provided dictionaries and additional custom data structures. Highly efficient storage and management of multiple characters and features. 7-8	/8

Control Structures and Error Handling	Basic control structures used. Little to no error handling, program crashes on unexpected inputs. 1-2	Adequate use of control structures. Some error handling implemented, but may not cover all cases. 3-4	Good use of control structures for program flow. Robust error handling for most invalid inputs and unexpected scenarios. 5-6	Excellent use of control structures, optimising program flow. Comprehensive error handling covering all possible scenarios with informative user feedback. 7-8	/8
Input Validation	Input validation is incomplete or frequently fails, leading to crashes or bugs. 1	Basic input validation implemented, handling common errors but missing edge cases. 2	Robust input validation, preventing invalid inputs and handling errors gracefully. 3-4	Comprehensive and foolproof input validation, enhancing user experience and game stability. 5	/5
User Interface	Basic menu system with minimal guidance. Character information displayed poorly. 1	Functional menu system with some user guidance. Character information displayed adequately. 2	Clear and intuitive menu system. Character information well-displayed. Good user prompts and feedback. 3-4	Exceptional menu system with excellent user guidance. Character information displayed comprehensively and attractively. Clear, informative prompts and feedback for all user actions. 5	/5
Code Quality and Style	Minimal adherence to PEP 8. Poor variable / function naming. Code organisation needs improvement. 1	Some adherence to PEP 8. Variable / function names are mostly clear. Code organisation is adequate. 2	Good adherence to PEP 8. Clear variable/function names. Code is well-organised and readable. 3-4	Strict adherence to PEP 8. Excellent variable/function naming. Code is exceptionally well-organised, readable, and efficient. 5	/5
Docs and Comments	Minimal comments. No program header. Purpose of code sections unclear. 1	Some comments present. Basic program header. Purpose of some code sections explained. 2	Good comments throughout. Clear program header. Purpose of most functions and complex logic explained. 3-4	Comprehensive, clear comments. Detailed program header. All functions and complex logic well-explained. Additional documentation provided where necessary. 5	/5
Version Control	Minimal use of GitHub. Few commits with unclear messages. 1	Basic use of GitHub. Regular commits but messages may lack clarity. 2	Good use of GitHub. Regular commits with clear messages. Project progression visible. 3	Excellent use of GitHub. Frequent, well-documented commits. Clear project progression. 4	/4

Testing and Debugging					
Criteria	Basic	Limited	Effective	Highly Effective	**Mark**
Testing Table	Few test cases (1-3) documented. Limited coverage of functionality. Missing key information in some fields. 1	Some test cases (4-6) documented. Covers basic functionality but may miss edge cases. Most fields filled but lack detail. 2-3	Good range of test cases (7-8) covering normal, boundary, and some erroneous conditions. All fields filled with clear information. 4-5	Comprehensive set of test cases (9-10) covering all aspects of the program, including normal, boundary, and erroneous conditions. Detailed and clear information in all fields. 6-7	/7
Debugging Process	Little or no explanation of bugs or the fixing process. 0	One or two bugs described but with limited detail. Basic explanation of fixing process provided. 1	Two bugs clearly described. Good explanation of the cause, fixing process, and verification for each. 2	Two or more bugs described in detail. Comprehensive explanation of the cause, fixing process, and thorough verification for each. Includes reflection on lessons learned. 3	/3

Evaluation					
Criteria	Basic	Limited	Effective	Highly Effective	**Mark**
Evaluation	Superficial assessment of the project. Minimal reflection on OOP principles or D&D mechanics. Few challenges or improvements discussed. 1	Basic assessment of the project. Some reflection on OOP principles and D&D mechanics. Limited discussion of challenges and potential improvements. 2	Thorough assessment of the project. Good reflection on OOP principles and D&D mechanics. Clear discussion of challenges faced and potential improvements. 3-4	Comprehensive and insightful assessment. Deep reflection on OOP principles and D&D mechanics. Detailed analysis of challenges, solutions, and future improvements. 5	/5