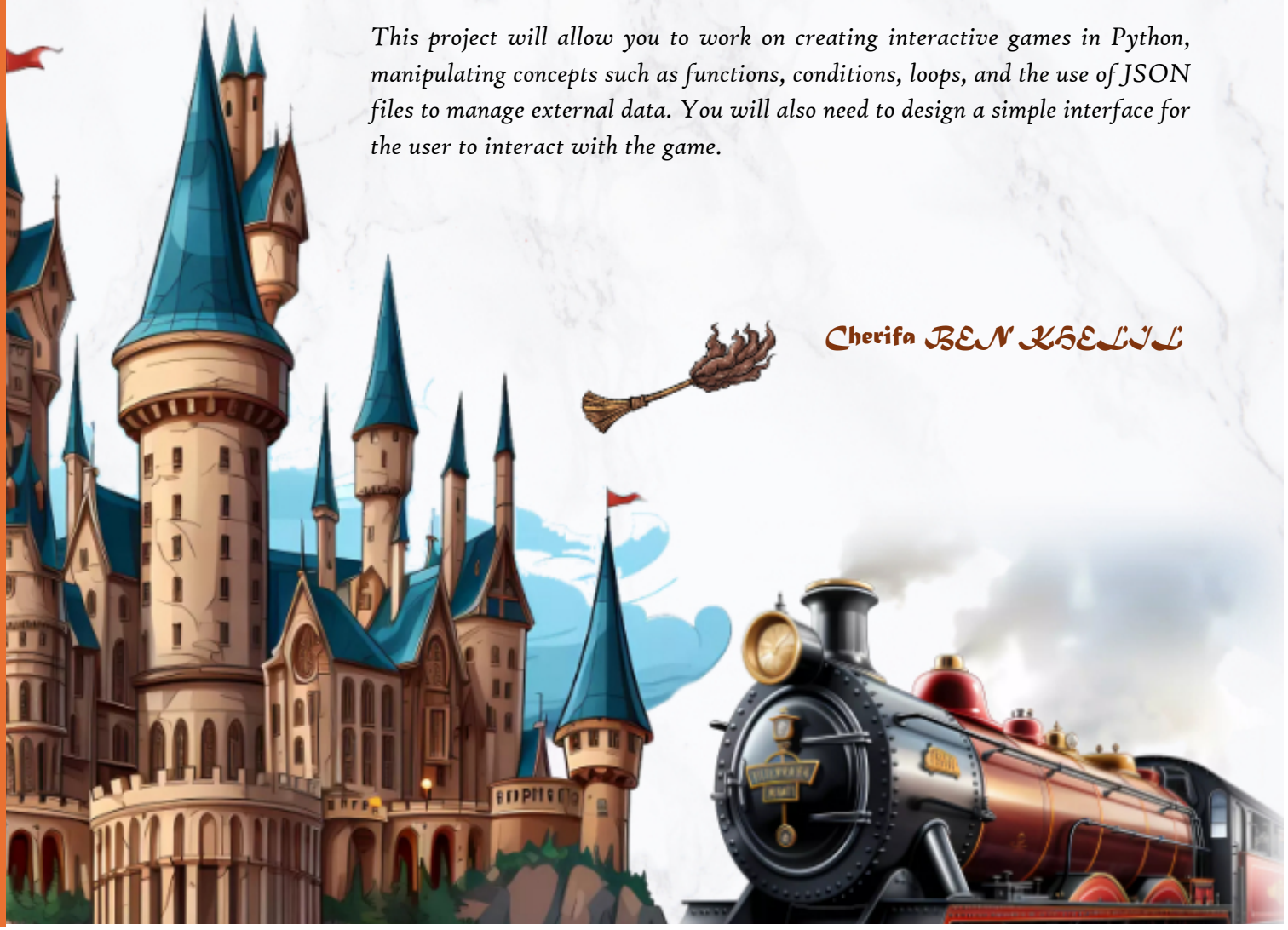
efrei
PARIS PANTHÉON-ASSAS UNIVERSITÉ

TI101I – Programming in Python

Hogwarts Project: The Art of Coding like a Wizard

This project will allow you to work on creating interactive games in Python, manipulating concepts such as functions, conditions, loops, and the use of JSON files to manage external data. You will also need to design a simple interface for the user to interact with the game.

Cherifa BEN KHELIL



The Project Grimoire

You must create an interactive adventure game in Python, inspired by **the world of Harry Potter**. It will be implemented in **two parts**.
The **first part** will consist of **three chapters** that the player must complete in sequence. Narratively, this section will cover most of the events of **the first film/book**.

The **second part**, corresponding to **chapter 4**, will consist of simulating a ***Quidditch match*** **and/or** recreating a significant event from **a movie** of your choice, from among those in the saga **starting with the second installment**.

You will use **Python data structures** (dictionaries, lists, sets, etc.) to manage characters, houses, spells, and trials.

This project will allow you to explore and consolidate your skills in:

- Functions and modularization.
- Manipulating nested dictionaries and lists.
- Managing random events and mini-games.
- Reading and, if needed, writing JSON files for saving, importing, and retrieving data.

The project should be developed in stages: first you will develop the first part, then the second, and then you can add optional features to enhance your project.

Project Timeline

1. **Intermediate submission:** Sunday, December 21, 2025
2. **Final submission:** Saturday, January 3, 2026
3. **Defense:** week of January 5, 2026

Table of Contents

1. Authorized libraries and native functions

The use of native modules and functions is limited to those specifically authorized and listed in this section. It is **prohibited** to use functions other than those that allow *adding*, *inserting*, and *deleting* from a *list* or *collection*, *cast* operations, and the *open* function for reading or writing to files, including specific functions for writing to a file.

1.1. The random library

The **random** module in Python provides functions for generating random numbers and performing random operations. In this project, you will have the opportunity to use two functions:

The random.randint(a, b) function: generates a random integer between a and b, including both extremes. This means that if you call random.randint(1, 6), it will return a random integer between 1 and 6.

Example:

```
import random
random_number = random.randint(1, 6)
print(random_number)  # Displays a number between 1 and 6
```

The random.choice(sequence) function: randomly selects an element from a sequence (such as a list, tuple, or string). For example, if you have a list of names, using random.choice(names) will return a random name from that list. This is particularly useful in games or simulations where a random choice is required.

Example:

```
import random
options = ['A', 'B', 'C']
random_choice = random.choice(options)
print(random_choice)  # Displays A, B, or C at random
```

1.2. The json library

The **json** library in Python allows you to work with data in JSON (JavaScript Object Notation) format, a lightweight and readable format for data exchange. To load data from a JSON file, the **json.load()** function is used. This function reads the JSON file and converts it into a Python object, such as a dictionary or a list, making it easier to access the information it contains.

Example:

```
import json
"""
Load data from the following data.json file:
{
    "name": "Pierre",
    "age": 30,
    "city": "Paris,"
    "interests": ["reading", "sports", "music"]
}
"""
with open('data.json', 'r', encoding='utf-8') as f:
    data = json.load(f)
# Display loaded data
print(f"Name: {data['name']}")
print(f"Age: {data['age']}")
print(f"City: {data['city']}")
print("Interests: ")
for interest in data['interests']:
print(interest, end=" ") # Displays each interest on a new line
```

For more details, see the following documentation[1] : JSON Load in Python - GeeksforGeeks.

1.3. Other native functions allowed

The abs() function is a native Python function, part of the standard library. It returns the absolute value of a number.

Example:

```
number = -10
print(abs(number)) # Displays 10
```

The lower() function in Python converts all alphabetical characters in a string to lowercase. It is part of the standard library and is used on a string to return a new string where all characters are lowercase. This method does not modify the original string, but returns a new modified string.

Example:

```
text = "Hello World"
```

[1] https://www.geeksforgeeks.org/json-load-in-python/

```
lowercase_text = text.lower()
print(lowercase_text)
```

The strip() function is a native method of strings in Python. It is used on a string to remove spaces (or other characters) at the beginning and end.

Example:

```
text = "   Harry Potter   "
cleaned_text = text.strip()
print(cleaned_text)  # Displays "Harry Potter" without the surrounding spaces
```

The split() function in Python is used to split a string into several sub-strings based on a specified delimiter. By default, it uses a space as the delimiter, but you can also specify another character as the separator. It returns a list containing the resulting sub-strings.

Example 1:

```
text = "Python is a powerful language"
words = text.split()  # Default separation, i.e., by spaces
print(words)
```

Output:

```
['Python', 'is', 'a', 'powerful', 'language']
```

If you want to split the string using a different character, you can specify it as an argument:

Example 2:

```
text = "1,2,3,4,5"
elements = text.split(",")  # Separation by comma
print(elements)
```

Output:

```
['1', '2', '3', '4', '5']
```

The **join() function** in Python is used to concatenate the elements of a list of strings into a single string using a specified separator.

Example:

```
words = ['Python', 'is', 'a', 'powerful', 'language']
text = ",".join(words)  # Default separation by a space
print(text)
```

Output:

```
Python,is,a,powerful,language
```

2. Creating the Project

2.1. First Steps to Start and Manage Your Project in Pairs

This project must be carried out in pairs (***only one trio is allowed for odd-numbered groups***). It is necessary to collaborate effectively with your partner throughout the project. You will use ***Git*** and ***GitHub*** to manage the project.

Normally, you have already created your ***GitHub*** account and completed the initial practical assignment allowing you to create your project, add collaborators, and make your first pushes. As a reminder, you can consult the document **"Using Git with PyCharm"** available on Moodle, which will guide you through these steps.

Reminder:

- The repository name must follow this convention: "***hogwarts-student1name-student2name-groupname***." Example: For a pair named **Durant** and **Dupont** from group **INT3**, the repository should be named "***hogwarts-durant-dupont-int3***."
- **Add the pair and teachers as collaborators:** The person who created the main repository must add the other member of the pair (or trio) and the two lab teachers as **collaborators** on the GitHub repository.
- Throughout the project, remember to **pull** regularly to retrieve the latest changes from your partner and push your own changes using **Git push**.

2.2. Project Structure

The structure described below must be followed:

```
hogwarts/
│
├── main.py
├── menu.py
├── universe/
│   ├── character.py
│   └── house.py
│
├── chapters/
│   ├── chapter_1.py
│   ├── chapter_2.py
│   ├── chapter_3.py
│   ├── chapter_4.py
│   └── chapter_5_extension.py
├── utils/
│   └── input_utils.py
└── data/
    ├── houses.json
    ├── teams_quidditch.json
```

```
    ├── spells.json
    ├── magic_quiz.json
    └── inventory.json
```

The tree structure is designed to separate the different parts of the project:

- **main.py**: game entry point, coordinates execution.
- **menu.py**: manages the main menu.
- The **universe/** folder contains the fundamental elements of the magical world**:**
    - **character.py**: creation and management of the player.
    - **house.py**: choice and management of the house.
- The chapters of the game are located in **chapters/** with one module per chapter to follow the narrative progression step by step.
- Utility functions are centralized in **utils/**, with **input_utils.py** responsible for checking user input and reading files.
- Place the downloaded data from Moodle in the data/ directory:
    - **inventory.json**: items available and their prices on Diagon Alley.
    - **houses.json**: information about the houses at Hogwarts.
    - **spells.json**: list of spells and their descriptions.
    - **quiz_magie.json**: set of questions and answers for the quiz section of chapter 3.
    - **equipes_quidditch.json**: data on the four Quidditch teams corresponding to the houses.

Create these files as you code the project, following this organization.

Note: the ***chapter_5_extension.py*** module could correspond to an additional free chapter, inspired by another episode of the saga.

3. Implementation

This section outlines the functions to be implemented in each file for the first part of the project, covering the initial **three chapters**, as well as the **utility** modules and those related to **the** game **universe**. Chapter 4 will be presented later in the second part of the project.

To structure the development effectively, it is recommended to proceed in the following order:

1. Implementation of utility functions **utils/input_utils.py**
2. Implementation of the **universe/character.py** module in parallel with **chapters/chapter_1.py**
3. Implementation of the **universe/house.py** module in parallel with **chapters/chapter_2.py**

4. Implementation of the following chapters (**chapter_3.py** and **chapter_4.py**)

The **main.py** and **menu.py** modules can be implemented from the outset. However, the **menu.py** module will need to be updated after each chapter is finalized to integrate the newly implemented features.

3.1. Input and file management: utils/input_utils.py

The **input_utils.py** file contains utility functions for properly managing user input in your game. **Before coding the rest of the project,** it is recommended that you implement this file, as all interactions with the player will go through these functions.

You must code the following three functions:

3.1.1. ask_text(message)

This function should be called whenever the user is prompted to enter text. It ensures that the input is not empty or only whitespace. If the user provides an invalid entry, the function will repeatedly prompt until valid text is entered, then returns the user's input.

Grimoire hint: to manage entries correctly, don't forget to remove spaces at the beginning and end of the string (for example, using the **strip()** method).

Example:

```
name = ask_text("Enter your character's name: ")
```

Output:

```
Enter your character's name: Potter
```

3.1.2. ask_number(message, min_val=None, max_val=None)

This function asks the user to enter an integer. The ***min_val*** and ***max_val*** limits are optional: if specified, the function ensures that the number entered is within the defined range. If the user enters an invalid value or one outside the limits, the function asks for the entry again until a correct number is obtained. It then returns the validated integer.

Grimoire hint: To check that a string of characters represents an integer, remember to **go through each character** and make sure it corresponds to a valid digit. Don't forget that a number can start with a `'-'` sign to indicate that it is negative. You can then manually convert each digit to an integer using `ord(c) - ord('0')` and recompose the number.

Example:

```
choice = ask_number("Courage level (1-10): ", 1, 10)
```

Output:

```
Courage level (1-10): 20
Please enter a number between 1 and 10.
Courage level (1-10): 8
```

3.1.3. ask_choice(message, options)

This function displays a numbered list of options and prompts the user to make a selection by entering the corresponding number. If the input is invalid, i.e., non-numeric or outside the proposed range, the function will repeatedly prompt the user until a valid choice is made. It then returns the value chosen from the options.

Grimoire hint: To create a choice menu, **number your options** and prompt the user to enter the number corresponding to their selection. You can reuse the **ask_number()** function to ensure the input is valid and within the allowed range.

Example:

```
choice = ask_choice("Do you want to continue?", ["Yes", "No"])
```

Output:

```
Do you want to continue?

1. Yes

2. No

Your choice:
```

3.1.4. load_file (file_path)

This function takes a file path as a parameter, opens the file in read mode (using UTF-8 encoding to correctly handle special characters), reads its contents as JSON using **json.load(),** and returns the data as a Python dictionary or list, depending on the JSON file's structure.

Grimoire hint: Use this function whenever you need to retrieve data from a JSON file.

3.2. The Kingdom of Wizards: universe

3.2.1. Character and attribute management: universe/character.py

This module contains functions for creating, managing, and displaying the player's character.

3.2.1.1. init_character(last_name, first_name, attributes)

This function creates a **dictionary** to represent the player's character, using the following structure:

```
{
    "Last Name": string_value_of_last_name,
    "First Name": string_value_of_first_name,
    "Money": integer_value_of_money,
    "Inventory": list,
    "Spells": list,
    "Attributes": dictionary
}
```

The fields must be filled in and initialized as follows:

- Personal information is initialized with the *first* and *last names* provided as parameters.
- The starting amount of money is set at *100 galleons*.
- The inventory is created as an *empty list*.
- Since the player is a novice wizard, their '*Spells'* field is initialized as an empty list.
- The attributes field is initialized with the dictionary passed as the attributes parameter.

The function then returns this complete dictionary, which serves as the foundation for all interactions with the character in the game.

3.2.1.2. display_character(character)

This function displays the player's character information: full name, amount of money available, attributes, inventory contents, and spells mastered, **as well as all new information that may be added to the character throughout the game** (other fields will be introduced later). It should be used to help the player track their character's progress and possessions throughout the adventure.

Grimoire hint: Consider **iterating directly over the keys of a dictionary**. If a key's value is:

- a dictionary: iterate through its subkeys to display the nested information.
- a list: convert its elements to strings and join them using **join().**
- otherwise, simply display the value.

Display example:

```
Character profile:
Last name: Potter
First name: Harry
Money: 100
Inventory:
Spells:
```

```
Attributes:
 - Courage: 8
 - Intelligence: 8
 - Loyalty: 8
 - Ambition: 8
```

3.2.1.3. modify_money(character, amount)

This function receives a positive or negative integer (the *amount* parameter) and adds it to the player's character's current amount of money.

Grimoire hint: Use this function to manage the player's financial transactions throughout the game.

3.2.1.4. add_item(character, key, item)

This function adds an *item* to the list corresponding to the field specified by the *key* parameter in the *character* dictionary. The *key* can be either 'Inventory' or 'Spells'. The item, passed as a string, is appended to the specified list.

Grimoire hint: Call this function whenever the player obtains or recovers a new item or spell during their journey.

3.2.2. House data and score management: universe/house.py

The **house.py** module is dedicated to managing Hogwarts houses, including player distribution, updating points, and displaying the winning house. It contains three main functions.

3.2.2.1. update_house_points(houses, house_name, points)

This function is used to update the score of a specific house. It takes as a parameter a dictionary where the keys are the house names and the values are their current points :

```
houses = {
    "Gryffindor": 0,
    "Slytherin": 0,
    "Hufflepuff": 0,
    "Ravenclaw": 0
}
```

The *house_name* parameter specifies which house's score should be adjusted, and *points* is the number of points to add (or subtract if negative). If the house exists, the function updates its score and displays a message showing the change and the new total. If the house cannot be found, a

warning message is displayed. This function does not return a value but modifies the dictionary passed as a parameter directly.

3.2.2.2. display_winning_house(houses)

This function displays the house with the highest score at a given moment. It receives a dictionary *of houses* with the current scores as a parameter. If only one house is in the lead, it is displayed as the winner. In the event of a tie, all the houses that are tied are listed.

Grimoire hint: Start by looking for the highest score among all houses, then browse the dictionary to identify all the houses that have that score.

3.2.2.3. assign_house(character, questions)

This function determines a player's house by combining the character's personal attributes with their answers to the Sorting Hat's personality test during the sorting ceremony. It takes the following parameters:

- character: a dictionary representing the player's character, including their attributes (courage, intelligence, loyalty, ambition).
- questions: a list of tuples, each containing: (1) the question text, (2) a list of possible choices, and (3) the corresponding houses for each answer.

```
questions = [
    (
        "You see a friend in danger. What do you do?",
        ["Rush to help", "Think of a plan", "Seek help", "Stay calm
and observe"],
        ["Gryffindor", "Slytherin", "Hufflepuff", "Ravenclaw"]
    ),
    (
        "Which trait describes you best?",
        ["Brave and loyal", "Cunning and ambitious", "Patient and
hardworking", "Intelligent and curious"],
        ["Gryffindor", "Slytherin", "Hufflepuff", "Ravenclaw"]
    ),
    (
        "When faced with a difficult challenge, you...",
        ["Charge in without hesitation", "Look for the best
strategy", "Rely on your friends",
         "Analyze the problem"],
        ["Gryffindor", "Slytherin", "Hufflepuff", "Ravenclaw"]
    )
]
```

The function calculates a ***score*** for each house: first by taking into account the player's character attributes (*each trait influences a specific house*), then by adding points based on the answers to the test.

Grimoire hint: The function's logic follows these steps:

1. Initialize a dictionary assigning a score of 0 to each house.
2. For each character attribute, retrieve its value (points) and multiply this value by 2 to determine its influence on the corresponding house:
    - **Courage** → courage points × 2 → add to **Gryffindor**
    - **Ambition** → ambition points × 2 → add to **Slytherin**
    - **Loyalty** → loyalty points × 2 → add to **Hufflepuff**
    - **Intelligence** → intelligence points × 2 → added to **Ravenclaw**
3. For each answer to the test, add 3 points to the house corresponding to the option chosen by the player.
4. The house with the highest score becomes the player's final house, and the function returns the name of that house.

This function is used by **sorting_ceremony(character)** in the **chapter_2** module to ensure that the assignment runs smoothly. A complete example of execution is available in the dedicated section.

Execution example:

```
You see a friend in danger. What do you do?
1. Rush to help
2. Think of a plan
3. Seek help
4. Stay calm and observe
Your choice: 2

Which trait describes you best?
1. Brave and loyal
2. Cunning and ambitious
3. Patient and hardworking
4. Intelligent and curious
Your choice: 1
```

```
When faced with a difficult challenge, you...
1. Charge in without hesitation
2. Look for the best strategy
3. Rely on your friends
4. Analyze the problem
Your choice: 2

Summary of scores:
Gryffindor: 21 points
Slytherin: 22 points
Hufflepuff: 20 points
Ravenclaw: 16 points
```

3.3. The Manuscript of the First Four Chapters

In this section, you will model the first three chapters of the adventure. Each chapter is represented by a module that groups events, which will be implemented as functions ready to be integrated into the overall gameplay. Details about the functions and their constraints will be provided in the following sections. First, here is an overview of the chronology of events in these chapters:

Chapter 1 – Arrival in the magical world:

- Player's character creation: last name, first name, and starting attributes.
- Receipt of the letter from Hogwarts.
- Meeting with Hagrid.
- Purchase of supplies on Diagon Alley (budget and inventory management).

Chapter 2 – The journey to Hogwarts:

- Meeting Ron, Hermione, and Draco.
- House selection via direct choice or personality test (minimum of 3 questions).
- Settling into the common room.

Chapter 3 – Classes and discovering Hogwarts:

- Learning basic spells: 1 defensive spell, 1 offensive spell, 3 utility spells.
- Mini-games related to Snape's classes (logic, memory, spell recognition, etc.).

3.3.1. Character creation and start of the adventure: chapters/chapter_1.py

The **chapter_1.py** module represents the first chapter of the adventure: ***arrival in the magical world***. This module integrates a series of functions that sequentially handle character creation, receiving the Hogwarts letter, meeting Hagrid, and purchasing supplies prior to the journey to school.

3.3.1.1. introduction()

This function displays the introductory text for the chapter. It should welcome the player and announce the beginning of the story.

Grimoire hint: A simple **input()** allows you to pause so that the player can read the message before continuing.

3.3.1.2. create_character()

This function creates the main character (the player). It prompts the player to enter their first and last names and assign a value from 1 to 10 to four qualities: *courage*, *intelligence*, *loyalty*, and *ambition*. The values entered are grouped in an attributes dictionary, then given to the ***init_character()*** function to create the complete character. Finally, the character's profile is displayed on the screen.

Example of execution:

```
Enter your character's last name: Potter
Enter your character's first name: Harry

Choose your attributes:
Courage level (1-10): 8
Intelligence level (1-10): 8
Loyalty level (1-10): 8
Ambition level (1-10): 8

Character profile:
Last name: Potter
First name: Harry
Money: 100
Inventory:
Spells:
Attributes:
 - Courage: 8
 - Intelligence: 8
```

```
- Loyalty: 8
- Ambition: 8
```

3.3.1.3. receive_letter()

This function simulates receiving the Hogwarts acceptance letter. It displays the letter's message and gives the player two options: accept or decline. If the player declines, the game ends immediately with a humorous message. If they accept, the adventure continues.

Grimoire hint: The **exit()** function immediately terminates program execution. Once called, no further instructions are executed, and the program stops at that point. You can pass an exit code to indicate the reason for termination: 0 for normal exit, or any other value to signal an error.

Execution example:

```
An owl flies through the window, delivering a letter sealed with the
Hogwarts crest...

"Dear Student,

We are pleased to inform you that you have been accepted to Hogwarts
School of Witchcraft and Wizardry!"

Do you accept this invitation and go to Hogwarts?

1. Yes, of course!

2. No, I'd rather stay with Uncle Vernon...

Your choice: 2

You tear up the letter, and Uncle Vernon cheers:

"EXCELLENT! Finally, someone NORMAL in this house!"

The magical world will never know you existed... Game over.
```

3.3.1.4. meet_hagrid(character)

This function introduces the character Hagrid. It displays a short dialogue and asks the player if they want to follow him. Regardless of the choice, Hagrid ends up leading the player to Diagon Alley.

Example of execution:

```
Hagrid: 'Hello Harry! I'm here to help you with your shopping on Diagon
Alley.'

Do you want to follow Hagrid?
```

```
1. Yes

2. No

Your choice: 2

Hagrid gently insists and takes you along anyway!
```

3.3.1.5. buy_supplies(character)

This function allows to buy the required school supplies on Diagon Alley. The complete catalog is loaded from the **data/inventory.json** file. The player must buy the **three essential items**: *Magic wand*, *Wizard robe*, and *Potions book*.

Once these purchases have been made, the player must choose a pet from among the authorized species:

- Owl - 20 galleons
- Cat - 15 galleons
- Rat - 10 galleons
- Toad - 5 galleons

Please note: The budget is checked before each purchase. If the player does not have enough money or forgets a mandatory item, they lose the game.

Finally, the function displays the character's final inventory.

Example of execution:

```
Welcome to Diagon Alley!

Catalog of available items:
1. Magic Wand - 35 Galleons (required)
2. Wizard Robe - 20 Galleons (required)
3. Tin Cauldron - 15 Galleons
4. Potions Book - 25 Galleons (required)
5. Magic Quill - 5 Galleons
6. Enchanted Book - 30 Galleons
7. Copper Scale - 10 Galleons
8. Invisibility Cloak - 100 Galleons

You have 100 Galleons.
Remaining required items: Magic Wand, Wizard Robe, Potions Book
Enter the number of the item to buy: 1
You bought: Magic Wand (-35 Galleons).

You have 65 Galleons.
Remaining required items: Wizard Robe, Potions Book
Enter the number of the item to buy: 2
```

```
You bought: Wizard Robe (-20 Galleons).

You have 45 Galleons.
Remaining required items: Potions Book
Enter the number of the item to buy: 4
You bought: Potions Book (-25 Galleons).

All required items have been purchased!

It's time to choose your Hogwarts pet!

You have 20 Galleons.

Available pets:
1. Owl - 20 Galleons
2. Cat - 15 Galleons
3. Rat - 10 Galleons
4. Toad - 5 Galleons
Which pet do you want?
1. Owl
2. Cat
3. Rat
4. Toad
Your choice: 1
You chose: Owl (-20 Galleons).

All required items have been successfully purchased! Here is your
final inventory:

Character Profile:
Last Name : Potter
First Name : Harry
Money : 0
Inventory : Magic Wand, Wizard Robe, Potions Book, Owl
Spells :
Attributes :
- courage : 8
- intelligence : 8
- loyalty : 8
- ambition : 8
```

3.3.1.6. start_chapter_1()

This function orchestrates the entire course of Chapter 1. It calls the other functions in order:

1. Display introduction
2. Character creation
3. Receiving the letter
4. Meeting Hagrid

5. Purchasing supplies
6. Display a message confirming the end of the chapter and announcing the start of the adventure at Hogwarts, for example: "*End of Chapter 1! Your adventure begins at Hogwarts...*"
7. Return of the dictionary representing the player's created character

3.3.2. Journey to Hogwarts and selection of the house: chapters/chapter_2.py

The **chapter_2.py** module manages the second chapter of the adventure, ***"The Journey to Hogwarts*****,"** which covers the trip to Hogwarts, the first encounters, and the assignment to houses. It must contain the following functions:

3.3.2.1. meet_friends(character)

This function depicts the journey on the Hogwarts Express and the first meeting with *Ron*, *Hermione*, and *Draco*. It takes the **character** dictionary as a parameter and presents the player with a series of interactive encounters. At each encounter, the player chooses from several options. **Each choice affects one of the character's attributes**, such as courage, intelligence, loyalty, or ambition, reflecting the player's personality and decisions during these interactions.

Meeting with Ron:

The player will see the following message appear.

```
Hi! I'm Ron Weasley. Mind if I sit with you?

How do you respond?

1. Sure, have a seat!

2. Sorry, I prefer to travel alone.

Your choice:
```

- If the player agrees to sit with Ron, this reflects loyalty and sociability. The character's **loyalty** attribute is therefore increased by 1.
- If the player refuses, it shows greater independence or personal ambition. The **ambition** attribute is then increased by 1.

Meeting Hermione:

The following content will be displayed on the screen.

```
Hello, I'm Hermione Granger. Have you ever read 'A History of Magic'?

How do you respond?

1. Yes, I love learning new things!

2. Uh… no, I prefer adventures over books.
```

- If the player shows an interest in learning, the **intelligence** attribute is increased by 1.
- If the player prefers adventures to books, they demonstrate courage in action rather than in study. The **courage** attribute is then increased by 1.

Encounter with Drago:

The following message will appear on the screen.

```
I'm Draco Malfoy. It's best to choose your friends carefully from the
start, don't you think?

How do you respond?

1. Shake his hand politely.

2. Ignore him completely.

3. Respond with arrogance.
```

- A polite and respectful reaction reflects strategy and social cunning, so **ambition** +1.
- Ignoring Draco shows loyalty to your own choices and friends, so **loyalty** +1.
- Responding arrogantly demonstrates courage in the face of a conflictual situation, so **courage** +1.

At the end, the function displays the modified attributes so that the player can see the impact of their choices.

Example of execution:

```
You board the Hogwarts Express. The train slowly departs northward...

A red-haired boy enters your compartment, looking friendly.

— Hi! I'm Ron Weasley. Mind if I sit with you?

How do you respond?

1. Sure, have a seat!

2. Sorry, I prefer to travel alone.

Your choice: 1

Ron smiles: — Awesome! You'll see, Hogwarts is amazing!



A girl enters next, already carrying a stack of books.

— Hello, I'm Hermione Granger. Have you ever read 'A History of Magic'?

How do you respond?
```

```
1. Yes, I love learning new things!

2. Uh… no, I prefer adventures over books.

Your choice: 1

Hermione smiles, impressed: — Oh, that's rare! You must be very clever!



Then a blonde boy enters, looking arrogant.

— I'm Draco Malfoy. It's best to choose your friends carefully from
the start, don't you think?

How do you respond?

1. Shake his hand politely.

2. Ignore him completely.

3. Respond with arrogance.

Your choice: 2

Draco frowns, annoyed. — You'll regret that!

The train continues its journey. Hogwarts Castle appears on the
horizon...

Your choices already say a lot about your personality!

Your updated attributes: {'courage': 8, 'intelligence': 9, 'loyalty':
10, 'ambition': 8}
```

3.3.2.2. welcome_message()

This function displays a welcome message from Professor *Dumbledore*. It takes no parameters and is used solely for narration.

Grimoire hint: A simple **input()** allows you to pause so that the player can read the message before continuing.

3.3.2.3. sorting_ceremony(character)

This function conducts the Sorting Hat ceremony. It takes the **character** dictionary as a parameter and asks the player to answer a mini personality quiz consisting of three questions.

The questions are represented by the following list:

```
questions = [
    (
        "You see a friend in danger. What do you do?",
```

```
        ["Rush to help", "Think of a plan", "Seek help", "Stay calm
and observe"],
        ["Gryffindor", "Slytherin", "Hufflepuff", "Ravenclaw"]
    ),
    (
        "Which trait describes you best?",
        ["Brave and loyal", "Cunning and ambitious", "Patient and
hardworking", "Intelligent and curious"],
        ["Gryffindor", "Slytherin", "Hufflepuff", "Ravenclaw"]
    ),
    (
        "When faced with a difficult challenge, you...",
        ["Charge in without hesitation", "Look for the best
strategy", "Rely on your friends",
         "Analyze the problem"],
        ["Gryffindor", "Slytherin", "Hufflepuff", "Ravenclaw"]
    )
]
```

Each item in this list is a **tuple** containing three pieces of information:

1. The text of the question.
2. The list of choices offered to the user.
3. The list of houses associated with each choice, in the same order as the options.

Using a **list of tuples** allows these three pieces of related information to be grouped together for each question, while maintaining a fixed order.

The function then calls **assign house(character, questions)** to determine the player's final house and updates the ***"House***" field in the **character** dictionary. It then displays the assigned house with a narrative message.

Example of execution:

```
The sorting ceremony begins in the Great Hall...
The Sorting Hat observes you for a long time before asking its
questions:

You see a friend in danger. What do you do?
1. I rush to help them
2. I think about a plan
3. I look for help
4. I stay calm and observe
Your choice: 2

Which trait best describes you?
1. Brave and loyal
2. Cunning and ambitious
```

```
3. Patient and hardworking
4. Intelligent and curious
Your choice: 1

When faced with a difficult challenge, you...
1. Jump right in without hesitation
2. Look for the best strategy
3. Rely on your friends
4. Analyze the problem
Your choice: 2

Summary of scores:
Gryffindor: 21 points
Slytherin: 22 points
Hufflepuff: 20 points
Ravenclaw: 16 points

The Sorting Hat exclaims: Slytherin!!!
You join the Slytherin students to loud cheers!
```

3.3.2.4. enter_common_room(character)

This function places the player in their house common room. It takes the **character** dictionary as a parameter and retrieves house information from the **houses.json** file.

The function displays:

- A description of the common room,
- The house welcome message, and
- The house colors corresponding to the player's assigned house.

Example of execution:

```
You follow the prefects through the castle corridors...

🐍 You discover a vaulted common room, illuminated by the green glow of the lake. Students watch you with curiosity and respect.

✨ ✨ Cunning and ambition are your allies. Welcome to the noble House of Slytherin.

Your house colors: green, silver
```

3.3.2.5. start_chapter_2(character)

This function coordinates the entire course of **Chapter 2**. It calls the other functions in the following order:

1. Meeting *Ron*, *Hermione*, and *Draco*
2. Welcome message from *Dumbledore*
3. Sorting ceremony in the Great Hall
4. Player's installation in their common room according to their house
5. Display of complete information about the player's character to summarize the end of the chapter
6. Display of a message confirming the end of the chapter and announcing the start of classes at Hogwarts.

3.3.3. Learning spells and magic quiz: chapters/chapter_3.py

This module manages the third chapter, "***Classes and Discovering Hogwarts***," which covers the first magic lessons and a quiz to test the player's knowledge. It must contain the following functions:

3.3.3.1. learn_spells(character, file_path="../data/spells.json")

This function allows the player's character to **learn five spells**, chosen at random from the JSON file containing all available spells. Among these five spells, the player must master:

- **1 offensive spell**
- **1 defensive spell**
- **3 utility spells**

The function takes the **character** dictionary and the path to the spells JSON file as parameters.

Grimoire hint: Spells are selected randomly until the quota for each category is met. For each spell learned:

- The spell is added to the player's **Spells** list.
- A message appears on the screen announcing the **name and type** of the learned spell.

Once all five spells have been acquired, the function displays a complete summary of the spells learned, including their name, type, and description.

Example of execution:

```
You begin your magic lessons at Hogwarts...
You have just learned the spell: Crucio (Offensive)
Press Enter to continue...
You have just learned the spell: Alohomora (Utility)
Press Enter to continue...
```

```
You have just learned the spell: Reparo (Utility)
Press Enter to continue...
You have just learned the spell: Protego (Defensive)
Press Enter to continue...
You have just learned the spell: Obliviate (Utility)
Press Enter to continue...

You have completed your basic spell training at Hogwarts!
Here are the spells you now master:

- Crucio (Offensive): Inflicts unbearable pain on the target.
- Alohomora (Utility): Unlocks doors and locked objects.
- Reparo (Utility): Repairs broken objects.
- Protego (Defensive): Creates a magical shield to block attacks.
- Obliviate (Utility): Erases specific memories from a person's mind.
```

3.3.3.2. magic_quiz(character, file_path="../data/magic_quiz.json")

This function offers an interactive quiz with four random questions to test the player's knowledge. It takes the **character** dictionary and the JSON file containing the list of questions and answers as parameters. For each question, the player enters their answer, and the function compares it to the correct answer. Each correct answer earns 25 points. At the end of the quiz, the total is displayed and added to the player's score.

Grimoire hint: To draw multiple questions at random without duplicates, you can use a loop with **random.choice().** Each time a question is drawn, it is added to the final list only if it is not already there.

Example of execution:

```
Welcome to the Hogwarts magic quiz!
Answer the 4 questions correctly to earn points for your house.

1. Which spell is used to levitate objects?
>  Wingardium Leviosssssa
Wrong answer. The correct answer was: Wingardium Leviosa
2. Which spell can be used to disarm an opponent?
>  Expelliarmus
Correct answer! +25 points for your house.
3. What is the basic protective spell?
> Protego
```

```
Correct answer! +25 points for your house.
4. Which spell causes a glowing red light to appear as a warning?
> Lumos
Wrong answer. The correct answer was: Periculum
Score obtained: 50 points
```

3.3.3.3. start_chapter_3(character, houses)

This function orchestrates the entire course of **chapter 3**. It calls, in order:

1. the spell learning function,
2. the magic quiz function to test the player's knowledge,
3. the function **for updating the player's house points,**
4. the function displaying the currently leading house,
5. the function displaying all player information.

3.4. Management of the game's main interface: menu.py

This module manages the main game interface and allows the player to navigate between chapters or exit the game. It contains two main functions:

3.4.1. *display_main_menu()*

This function displays the main game menu:

```
1. Start Chapter 1 – Arrival in the magical world.
2. Exit the game.
```

3.4.2. *launch_menu_choice()*

This function controls the main menu loop and orchestrates the game's progression based on the player's choices. The steps are as follows:

1. Initialize a ***houses*** dictionary to track the points for each house.
2. Displays the main menu in a loop via ***display_main_menu()***.
3. Reads the player's choice:
    a. If the player chooses "1", successively launches the chapters:
        i. The first chapter to create the character and begin the adventure.
        ii. The second chapter for the journey to Hogwarts and house selection.
        iii. The third chapter for learning spells and the magic quiz.
        iv. The fourth chapter for the Quidditch test or the final chapter of the adventure.
    b. If the player chooses "2," display a goodbye message and exit the loop.
    c. Otherwise, display an error message for an invalid choice.

3.5. Game entry point : main.py

This module serves as the main entry point for the game. It launches the main menu loop.

3.6. Part 2: The grand conclusion of the adventure (chapters/chapter_4.py)

In this final chapter, the player will experience **the end of their adventure at Hogwarts** through one of the following two scenarios:

- **A** guided and structured **Quidditch final**, simulating a match between the player's team and one of the other three Hogwarts houses.
- **A free-form scenario**, inspired by a film from the Harry Potter saga, starting with *The Chamber of Secrets*. Scenario ideas will be suggested for each film to help you choose a starting point. However, this scenario remains **open to your creativity**, as long as you respect the technical and narrative constraints defined for the project.

You will therefore have the choice between:

- following a guided module with precise instructions,
- or designing a free module, without detailed guidance but with consistency and quality criteria to be respected.

It is also possible to implement **both options**: in this case, the additional scenario will be valued as **a bonus** during the evaluation.

3.6.1. Guided scenario: Quidditch match

This chapter simulates a ***Quidditch match*** between the player's house and an opposing house. The player's character takes on the role **of Seeker** and directly influences the course of the match, scoring points for their house and influencing the final result. The chapter relies on several functions, each fulfilling a specific role:

- Creation of the two teams and resetting of scores.
- Match play
- End of the match**:**
    - If the Golden Snitch is caught
    - Otherwise, after 20 turns
- Results: display the score, win/loss, and player statistics.

3.3.4.1. create_team(house, team_data, is_player=False, player=None)

This function allows you to build a team ready to play a Quidditch match. It will be used in the **_quidditch_ match()** function to first create the player's team, then the opposing team.

Parameters:

- **house**: the name of the house for which the team is created.
- **team_data**: the data of the players available in the house.
- **is_player**: A Boolean indicating whether the team corresponds to the player's team (*True*) or not (*False*).
- ***player (optional)***: dictionary representing the player's character, used if *is_player=True* to add them as Seeker.

Return:

The function returns a **dictionary representing the team**, containing the following fields:

- **name**: the name of the house.
- **score**: the number of points accumulated, initialized to 0.
- **goals_scored**: the number of goals scored, initialized to 0.
- **goals_blocked**: the number of attacks stopped by the team, initialized to 0.
- **caught_snitch**: Boolean indicating whether the Golden Snitch has been caught (initialized to False).
- **players:** the list of players on the team, possibly including the main player at the top.

Example:

```
team = {
    'name': 'Gryffindor',
    'score': 0,
    'has_scored': 0,
    'has_stopped': 0,
    'caught_snitch': False,
    'players': [
        'Harry Potter (Seeker)',
        'Ginny Weasley (Chaser)',
        'Katie Bell (Chaser)',
        'Demelza Robins (Chaser)',
        'Ron Weasley (Keeper)',
        'Jimmy Peakes (Beater)',
        'Ritchie Coote (Beater)']}
```

Algorithm to follow :

1. Create a team dictionary with all counters initialized (score, goals_scored, goals_blocked, caught_snitch) and the list of players from equipe_data.
2. If is_player=True and the player dictionary is provided:

- Create a **new list of players**.
- Add the player to the top of the list with the role of **Seeker**.
- Then go through the other players and add them to the list, avoiding duplicating the main player.
- Replace the original list of players with this new list.

3. Return the complete team dictionary, ready to be used in the match.

3.3.4.2. attempt_goal(attacking_team, defending_team, player_is_seeker=False)

This function simulates an attempt to score a goal during a Quidditch match. It randomly decides whether the attack is successful or stopped by the defense and updates the scores accordingly.

Parameters:

- **attacking_team**: dictionary representing the team making the attack.
- **defense_team**: dictionary representing the defending team.
- **player_is_seeker**: Boolean indicating whether the main attacker is the player (***True***) or not (***False***).

Return:

The function does not return a value but directly updates the scores in the **attacking_team** and **defending_team** dictionaries.

Algorithm to follow:

1. Generate a random number **chance_goal** between 1 and 10. If **chance_goal** ≥ 6, the shot is successful. Otherwise, it is a failure (attack blocked).
2. If the shot is successful:
    a. Determine which player scores:
        i. if **player_is_seeker=True**, the scorer is the main player,
        ii. otherwise it is a player chosen at random from the attacking team.
    b. Add 10 points to the attacking team's score and increment its goal counter
    c. Display a message indicating the goal scored by the scorer. (Example: "*Harry Potter (Seeker) scores a goal for Gryffindor! (+10 points)*")
3. Otherwise (missed shot):
    a. Increase the defensive team's blocked attack counter.
    b. Display a message indicating that the attack has been blocked. (Example: "*Gryffindor blocks the attack!*")

3.3.4.3. golden_snitch_appears()

This function randomly decides whether the Golden Snitch appears during the match. It takes no parameters and returns a Boolean value: ***True*** if the Golden Snitch appears, ***False*** otherwise. The logic consists of generating a random number between 1 and 6 and considering that the Golden Snitch appears only if this number is equal to 6.

3.6.1.4. catch_golden_snitch(e1, e2)

This function manages the capture of the Golden Snitch during the match by randomly choosing one of the two teams (**e1** and **e2**) as the winner. The team that catches the Golden Snitch sees its score increase by 150 points and its **caught_snitch** indicator change to **True**. The function then returns the dictionary corresponding to the winning team.

3.6.1.5. display_score(e1, e2)

This function displays the current score of both teams (**e1** and **e2**) during the match. It displays a header "Current score" and shows the score of each team so that the progress of the match can be followed.

Example:

```
Current score:

→ Gryffindor: 30 points

→ Hufflepuff: 10 points
```

3.6.1.6. display_team(house, team)

This function displays the name of the **house** followed by the list of players on **the team** with their roles.

Example:

```
Gryffindor team:

 - Harry Potter (Seeker)

 - Ginny Weasley (Chaser)

 - Katie Bell (Chaser)
```

```
- Demelza Robins (Chaser)
- Ron Weasley (Keeper)
- Jimmy Peakes (Beater)
- Ritchie Coote (Beater)
```

3.6.1.7. quidditch_match(character, houses)

This function manages the entire course of a Quidditch match between the player's house and a randomly selected opposing house, updating the teams' scores and the winning house's points while displaying the match events.

Parameters:

- **character**: dictionary representing the player.
- **houses**: dictionary containing houses informations and scores.

Algorithm to follow:

1. Load team data from ***teams_quidditch.json***.
2. Identify the player's house and randomly choose a different opposing house.
3. Create the player's team and the opposing team (using the ***create_team()*** function).
4. Display the teams (using the ***display_team()*** function).
5. Indicate the player's role (Seeker) with a simple message.
6. Play the match for a maximum of 20 rounds:

   a. Each team tries to score a goal (using the ***attempt_goal()*** function). If the attacking team is the player's team, the main character is involved.

   b. Display the current score (using the ***display_score()*** function).

   c. Check for the appearance of the Golden Snitch (***golden_snitch_appears()*** and ***catch_golden_snitch()***):

      i. If the Golden Snitch appears, the house that catches it receives 150 points and the match ends immediately.

      ii. Otherwise, the match continues as normal.

   d. Wait for user input to move on to the next round.
7. At the end of the match (or after the Golden Snitch is caught), display the final score and determine the outcome of the match (win, loss, or tie). The winning house of the match earns 500 points!
8. Update the winning house's points (either by score or by Golden Snitch) using the ***update_house_points()*** function.

Example:

```
Quidditch Match: Slytherin vs Hufflepuff!

Slytherin Team:
 - Harry Potter(Seeker)
 - Draco Malfoy (Seeker)
 - Cassius Warrington (Chaser)
 - Graham Montague (Chaser)
 - Urquhart (Chaser)
 - Miles Bletchley (Keeper)
 - Vincent Crabbe (Beater)
 - Gregory Goyle (Beater)


Hufflepuff Team:
 - Cedric Diggory (Seeker)
 - Heidi Macavoy (Chaser)
 - Zacharias Smith (Chaser)
 - Jason Samuels (Chaser)
 - Herbert Fleet (Keeper)
 - Anthony Rickett (Beater)
 - Maxine O'Flaherty (Beater)

You are playing for Slytherin as the Seeker

──── Turn 1 ────
Harry Potter (Seeker) scores a goal for Slytherin! (+10 points)
Zacharias Smith (Chaser) scores a goal for Hufflepuff! (+10 points)

Current score:
→ Slytherin: 10 points
→ Hufflepuff: 10 points

Press Enter to continue
──── Turn 2 ────
Harry Potter (Seeker) scores a goal for Slytherin! (+10 points)
Slytherin blocks the attack!

Current score:
→ Slytherin: 20 points
→ Hufflepuff: 10 points

Press Enter to continue
──── Turn 3 ────
Hufflepuff blocks the attack!
Zacharias Smith (Chaser) scores a goal for Hufflepuff! (+10 points)

Current score:
→ Slytherin: 20 points
→ Hufflepuff: 20 points

Press Enter to continue
──── Turn 4 ────
Hufflepuff blocks the attack!
Heidi Macavoy (Chaser) scores a goal for Hufflepuff! (+10 points)
```

```
Current score:
→ Slytherin: 20 points
→ Hufflepuff: 30 points


The Golden Snitch has been caught by Slytherin! (+150 points)

End of the match!

Current score:
Slytherin: 170 points
Hufflepuff: 30 points


Final result:
The Golden Snitch was caught by Slytherin!
+170 points for Slytherin! Total: 170 points.
The winning house is Slytherin with 170 points!
Victory for Slytherin!
+500 points for Slytherin! Total: 670 points.
The winning house is Slytherin with 670 points!
```

3.6.1.8. start_chapter_4_quidditch(character, houses)

This function oversees the entire flow of Chapter 4, which is dedicated to the Quidditch match.

It calls the other functions in order:

- Display the title of Chapter 4.
- Quidditch match progress.
- Display of an end-of-chapter message, for example: "*End of Chapter 4 — What an incredible performance on the field!*".
- Announcement of the house that wins the House Cup.
- Display of complete character information.

3.6.2. Free scenario: the grand conclusion to the adventure

You have chosen to implement a scenario inspired by a film from the Harry Potter saga, starting with The Chamber of Secrets. We offer the following scenario ideas for each film:

Film 2 – The Chamber of Secrets

1. Battle against the spiders in the Forbidden Forest.
2. Final duel against the Basilisk with choice of action (attack, dodge, use Gryffindor's sword) and confrontation with ***Tom Riddle***.

Film 3 – The Prisoner of Azkaban

1. Battle against a ***Dementor***.
2. Use the Marauder's Map to locate an enemy and confront them.
3. Duel against the ***Werewolf***.

Film 4 – The Goblet of Fire

1. Fight against one or more dragons.
2. Rescue friends underwater.
3. Maze: a game involving a series of choices and traps.
4. Final duel against Voldemort with management of dodging and attacking probabilities.

Film 5 – The Order of the Phoenix

1. Organization of Dumbledore's Army (team management and training).
2. Escape game in the Department of Mysteries (solving puzzles in several stages).
3. Duel against the Death Eaters (tactical choices between offensive and defensive spells).

Film 6 – The Half-Blood Prince

1. Potion mini-game (selecting the right ingredients in the right order).
2. Duel against the Death Eaters after the attack on the Burrow.
3. Final duel at the Astronomy Tower (with an interactive narrative sequence).

Film 7 – The Deathly Hallows (parts 1 and 2)

1. Exploration and infiltration of the Ministry of Magic.
2. Escape from Gringotts (interactive journey with the dragon).
3. Journey through the Forest: tactical choices to avoid the Death Eaters.
4. Final duel against Voldemort: strategic combat sequence with spell management and probability calculations.

You must choose **only** one **idea** from those proposed to implement. The other ideas can be added as bonuses if your chapter is completed on time or if you wish to continue developing your Python skills.

Depending on the idea you choose, you will need to construct the chapter's scenario, specifying:

- the key events,
- the characters involved,
- the necessary technical elements (data structures, variables, files, etc.).

The code must be structured into interconnected functions in order to correctly reconstruct the planned scenario.

Please note: adherence to the restrictions on library usage and predefined functions is mandatory.

4. The Wizard's Deliverables and Ministry Review

4.1. Scrolls to be Returned

The expected deliverables are as follows:

4.1.1. Interim Submission – December 21, 2025

Each pair (or trio) must push to GitHub with the following commit message: "**interim submission**." This submission must include all parts listed in the following tree structure, which must be finalized and fully functional.

```
├── main.py
├── menu.py
├── universe/
│   ├── character.py
│   └── house.py
│
├── chapters/
│   ├── chapter_1.py
│   ├── chapter_2.py
│   ├── chapter_3.py
├── utils/
│   └── input_utils.py
└── data/
    ├── houses.json
    ├── spells.json
    ├── magic_quiz.json
    └── inventory.json
```

4.1.2. Final submission – December 3, 2025

The pair (or trio) must push to GitHub with the following commit message: "**final submission**." The submission must include **all complete and fully functional files from** the project tree, as well as a **README.md** file**.**

The **README.md** file must be placed at the root of the GitHub repository so that it is automatically visible on the main page and must follow the following structure:

1. **General Presentation:**
    - **Project Title**
    - **Brief Description**
    - **Contributors:** List of members of the pair (or trio).
    - **Installation:**
        - Instructions for cloning the Git repository.
        - Steps to configure the development environment (if installations are necessary).
    - **Usage:**

        - Instructions on how to run the application.
        - Possibly, examples of commands or use cases.
    - **Key Features:** List of key features offered by the application.
2. **Logbook**
    - A logbook can help track project progress and task distribution:
        - **Project Timeline:** Dates and descriptions of key milestones, decisions made, and problems encountered.
        - **Task Distribution:** Who worked on what and how tasks were divided.
3. **Control, Testing, and Validation**
    - **Input and Error Management:**
        - Describe how the code handles values and ranges, as well as the methods implemented to handle potential errors.
        - Provide a list of known bugs.
    - **Testing Strategies:**
        - Specific test cases and results.
        - Screenshots showing the tests in action.

4.1.3. Defense – week of January 5, 2026

During the defense, each pair (or trio) must:

- Arrive with the first three chapters already executed and briefly present them to the instructor before proceeding to the presentation of the final chapter.
- Execute the scenario planned for the final chapter, focusing on the sequence of events and explaining the logic implemented.
- Respond to the teacher's questions clearly, providing well-structured and reasoned explanations.

⚠ Comments in the code (whether explanatory or related to unused sections) are strictly prohibited during the defense. Noncompliance with this rule will result in a score of 0.

4.2. Ministry of Magic Schedule

4.2.1. WEEK 1: Foundations and Chapter 1

Objective: Establish the foundations of the project and implement the first chapter.

Project Configuration

- Set up the project folder structure according to the required directory tree.
- Configure Git and GitHub following the established naming conventions.
- Add collaborators (project partner and teachers).

- Create the initial README file to document the project setup.

Modules to implement

- utils/input_utils.py
- universe/character.py
- chapters/chapter_1.py

Recommended commits and pushes

Each of you should make regular commits and pushes to save your work as you progress. However, be sure to make the following commits at the end of each module or major stage of the project:

1. `Initial project structure + README`
2. `Finalized utils/input_utils.py module: input functions and JSON loading`
3. `Finalized universe/character.py module : character creation and management`
4. `Chapter 1 completed : character creation, letter, meeting Hagrid, and purchases`

Tests to be performed

- User input validation
- Complete walkthrough of Chapter 1
- Management of input errors

4.2.2. WEEK 2: House module + Chapter 2 + Start of intermediate submission preparation

Objective: Develop house management, Chapter 2, and prepare for intermediate submission

Modules to implement

- menu.py
- main.py
- universe/house.py
- chapters/chapter_2.py

Recommended commits and pushes

In addition to regular commits and pushes to save your progress, be sure to **make the following commits** at the end of each stage of development:

1. `Finalized house.py module: point management and house distribution`
2. `Chapter 2 completed: meeting friends, sorting ceremony, and settling in at Hogwarts`

3. menu.py module in progress
4. Entry point main.py finalized

Tests to be performed

- Complete Chapter 2
- Testing house sorting and validating points
- Verification of pushes and commits

4.2.3. WEEK 3: Chapter 3 + Chapter 4 + Intermediate submission

Objective: Implement Chapter 3, finalize the intermediate submission, and begin Chapter 4

Modules to implement

- chapters/chapter_3.py
- Chapter 4: Quidditch match or free scenario

If Chapter 4 is a free scenario: Write a **short description** of the planned functions and submit it to the instructor for approval before implementation.
A **validation push** is then required before continuing development.

Recommended commits and pushes

Make regular commits and pushes throughout development. Be sure to **make the following commits** at the end of each major milestone:

1. Chapter 3 completed
2. menu.py module updated: complete navigation between chapters and game launch
3. Intermediate submission (complete and functional)
4. Chapter 4 in progress: Quidditch match (or scenario validated)

Tests to be performed

- Complete Chapter 3
- Validation of Chapter 4 scenario
- Testing interactions between chapters and integration
- Verification of pushes and commits

4.2.4. WEEK 4: Final Chapter 4 + Final submission

Objective: Finalize Chapter 4 and deliver the final submission

Recommended commits and pushes

Make regular commits and pushes throughout development. Be sure to **make the following commits** at the end of each major milestone:

1. Chapter 4 completed

2. README update
3. Project menu completed
4. Final repository (complete and functional)

Final deliverables

- Complete and functional game
- Complete README
- Finalized Git repository

Tests to be performed

- Complete walkthrough Chapter 4
- Tests and validation of integrated features
- Verification of pushes and commits

4.3. The Secrets of the Final Grade

The project will be assessed **throughout its duration**, during **follow-up sessions**, the **interim submission**, the **final submission**, and the **defense**. It will consider several criteria, each contributing to the final grade like the ingredients of a perfectly balanced potion:

- **Individual work**: As this is a collaborative project, it must demonstrate a **balanced and consistent distribution of tasks** among team members. Each member will be evaluated on their **personal involvement** in the group, the **quality of their contribution**, and their **overall understanding of the project**.
- **Code**: The quality, robustness, and proper functioning of the technical deliverables will be examined.
- **GitHub repository management (README and use of GIT)**: The repository must include a **complete README file structured** according to the expected format. It must also **reflect collaborative work** through **regular** and **relevant** *commits* and *pushes* made by each member. It is therefore **strongly recommended** that you make regular commits and push, **without waiting for a file to be completely finalized, each time a part of your work is advanced.**
- **Defense**: This assesses your ability to **present, explain, and argue** your project in a clear and structured manner, as well as to **respond appropriately to questions** asked by the teacher.

4.4. The Grimoire of Ethics

- Members of **pairs or trios** will have a portion of their grade that is **individual**, some of which will be assigned during **follow-up sessions**. **Repeated unexcused absences** may **lower the grade** for this part.
- Instructors will evaluate based on your **last commit and push made on the day of the deadline**. Any files missing after this date will receive a **score of 0** for the corresponding part. Therefore, it is essential to carefully check that all files have been correctly uploaded to your remote Git repository and that your instructor has been added as a collaborator.
- **Failure to follow instructions**, including the use of **unauthorized predefined functions**, will be **heavily penalized**.
- **Plagiarism in any form is strictly prohibited**, including copying from external sources or generating code using **AI tools**. Group members may consult external sources only for inspiration or to generate ideas**,** but must produce original, personal work**.** Any work deemed non-original will receive a **score of 0/20** and result in a **referral to the disciplinary board**.
- For detailed information on the anti-plagiarism policy and the use of tools such as Compilatio and Studium, please consult the documentation available in the “Direction of Studies” section of the academic portal.

Good luck with your magical adventure at Hogwarts! 🧙‍♂️⚡

"It is our choices that show what we truly are, far more than our abilities." - Albus Dumbledore