Terminal App:

DnD Spell slot tracker

Logic Walkthrough

This application focuses on different things you can do to the variable “your_character”. Option 1 lets you create the variable.

Option 2 allows you to view the remaining spell slots they have.

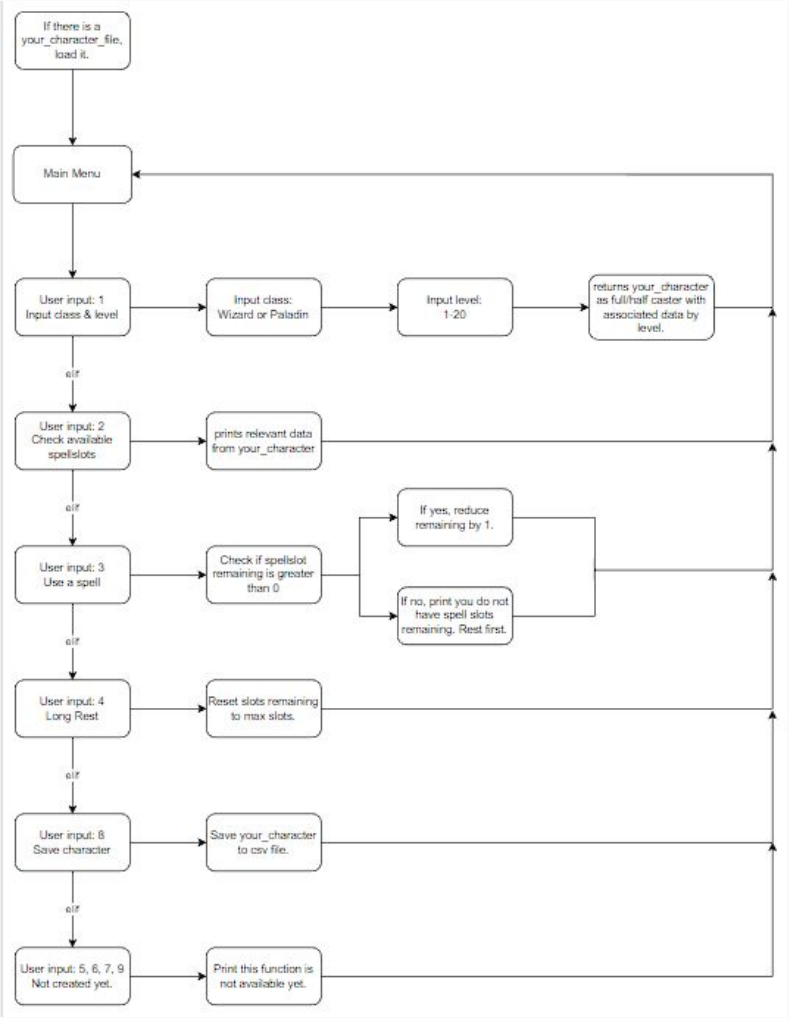
Option 3 allows you to use a spell slot, if you have any remaining.

Option 4 lets you replenish the spell slots.

Option 8 commits the current variable to a saved document for later use.

And it all starts from this lil’ guy.

```
Type quit to close the program.
1: Input class selection, and class level.
2: Check available spellslots
3: Use a spell
4: Long rest
5: View spell database
6: Update character spell-list.
7: Recommend my next learned spell.
8: Save your character.
9: Settings.
Enter your option:
```



Feature 1: Character creation

There were three main logical steps to take when choosing a class through this program.

The first is getting the user to choose what class they'd like to play (in this example, Wizard).

The second step is choosing their level, and final step is returning the values associated with that class & level.

This code was creating the (coding) class with all the available information, and selecting a (dnd) class they would like to play.

More on next slide.

```
class Full_Caster:
    def __init__(self, dndclass = "Wizard", level = 0, cantrips_known = 0, level_1_spellslots = 0, level_1_remaining = 0, level_2_spellslots = 0, level_2_remaining = 0, level_3_spellslots = 0, level_3_remaining = 0,
    level_4_spellslots = 0, level_4_remaining = 0, level_5_spellslots = 0, level_5_remaining = 0, level_6_spellslots = 0, level_6_remaining = 0, level_7_spellslots = 0, level_7_remaining = 0,
    level_8_spellslots = 0, level_8_remaining = 0, level_9_spellslots = 0, level_9_remaining = 0):
        self.dndclass = dndclass
        self.level = level
        self.cantrips_known = cantrips_known
        self.level_1_spellslots = level_1_spellslots
        self.level_1_remaining = level_1_remaining
        self.level_2_spellslots = level_2_spellslots
        self.level_2_remaining = level_2_remaining
        self.level_3_spellslots = level_3_spellslots
        self.level_3_remaining = level_3_remaining
        self.level_4_spellslots = level_4_spellslots
        self.level_4_remaining = level_4_remaining
        self.level_5_spellslots = level_5_spellslots
        self.level_5_remaining = level_5_remaining
        self.level_6_spellslots = level_6_spellslots
        self.level_6_remaining = level_6_remaining
        self.level_7_spellslots = level_7_spellslots
        self.level_7_remaining = level_7_remaining
        self.level_8_spellslots = level_8_spellslots
        self.level_8_remaining = level_8_remaining
        self.level_9_spellslots = level_9_spellslots
        self.level_9_remaining = level_9_remaining
```

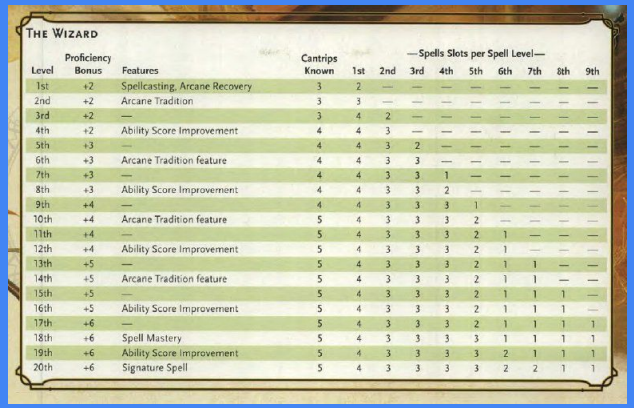
```
def class_input():
    print("Please enter in the class from the following choices: ")
    class_selection = input("Paladin \nWizard\n\n-------- \n\n")
    system('clear')
    return class_selection


def class_selector(class_selection):
    if class_selection == "quit":
        quit()
    elif class_selection == "Paladin":
        your_character = get_paladin_level()
        input("You successfully created a level " + str(your_character.level) + " " + your_character.dndclass + ".\nPlease press enter to return to main menu.\n\n-------- \n\n")
        return your_character
    elif class_selection == "Wizard":
        your_character = get_wizard_level()
        input("You successfully created a level " + str(your_character.level) + " " + your_character.dndclass + ".\nPlease press enter to return to main menu.\n\n-------- \n\n")
        return your_character
```

Feature 1 cont

The easiest way I know to get all the information required from the wizard table (for example) into this program is assigning a class and essentially making this table.

For any level chosen, the application assigns the correct data. While there might be an automatic way somewhere, leveling up is not consistent in its returns.



THE WIZARD

Level	Proficiency Bonus	Features	Cantrips Known	—Spells Slots per Spell Level— 1st	2nd	3rd	4th	5th	6th	7th	8th	9th
1st	+2	Spellcasting, Arcane Recovery	3	2	—	—	—	—	—	—	—	—
2nd	+2	Arcane Tradition	3	3	—	—	—	—	—	—	—	—
3rd	+2	—	3	4	2	—	—	—	—	—	—	—
4th	+2	Ability Score Improvement	4	4	3	—	—	—	—	—	—	—
5th	+3	—	4	4	3	2	—	—	—	—	—	—
6th	+3	Arcane Tradition feature	4	4	3	3	—	—	—	—	—	—
7th	+3	—	4	4	3	3	1	—	—	—	—	—
8th	+3	Ability Score Improvement	4	4	3	3	2	—	—	—	—	—
9th	+4	—	4	4	3	3	3	1	—	—	—	—
10th	+4	Arcane Tradition feature	5	4	3	3	3	2	—	—	—	—
11th	+4	—	5	4	3	3	3	2	1	—	—	—
12th	+4	Ability Score Improvement	5	4	3	3	3	2	1	—	—	—
13th	+5	—	5	4	3	3	3	2	1	1	—	—
14th	+5	Arcane Tradition feature	5	4	3	3	3	2	1	1	—	—
15th	+5	—	5	4	3	3	3	2	1	1	1	—
16th	+5	Ability Score Improvement	5	4	3	3	3	2	1	1	1	—
17th	+6	—	5	4	3	3	3	2	1	1	1	1
18th	+6	Spell Mastery	5	4	3	3	3	3	1	1	1	1
19th	+6	Ability Score Improvement	5	4	3	3	3	3	2	1	1	1
20th	+6	Signature Spell	5	4	3	3	3	3	2	2	1	1

```
def get_wizard_level():
    input_level = 0
    your_character = None
    while input_level != "quit":
        system('clear')
        input_level = input("What level is your Wizard?\n\n-------- \n\n")
        system('clear')
        if input_level == "quit":
            quit()
        elif input_level == "1":
            your_character = Full_Caster("Wizard", 1, 3, 2, 2)
            return your_character
        elif input_level == "2":
            your_character = Full_Caster("Wizard", 2, 3, 3, 3)
            return your_character
        elif input_level == "3":
            your_character = Full_Caster("Wizard", 3, 3, 4, 4, 2, 2)
            return your_character
        elif input_level == "4":
            your_character = Full_Caster("Wizard", 4, 4, 4, 4, 3, 3)
            return your_character
        elif input_level == "5":
elif input_level == "20":
    your_character = Full_Caster("Wizard", 20, 5, 4, 4, 3, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 1, 1, 1, 1)
    return your_character
else:
    print("That's not a valid option, please try again.")
    input("Enter to continue.\n\n-------- \n\n")
```

Feature 2:
Get spell slots:

This one returns a pretty table of all the information the user might want to know for their character to do with spell slots.

For each spell slot level, shows how many remaining and how many they have maximum.

```
def get_spells(your_character):
    try:
        x = PrettyTable()
        x.field_names = ["Spellslot level", "remaining", "maximum"]
        x.add_row(["1", your_character.level_1_remaining, your_character.level_1_spellslots])
        x.add_row(["2", your_character.level_2_remaining, your_character.level_2_spellslots])
        x.add_row(["3", your_character.level_3_remaining, your_character.level_3_spellslots])
        x.add_row(["4", your_character.level_4_remaining, your_character.level_4_spellslots])
        x.add_row(["5", your_character.level_5_remaining, your_character.level_5_spellslots])
        x.add_row(["6", your_character.level_6_remaining, your_character.level_6_spellslots])
        x.add_row(["7", your_character.level_7_remaining, your_character.level_7_spellslots])
        x.add_row(["8", your_character.level_8_remaining, your_character.level_8_spellslots])
        x.add_row(["9", your_character.level_9_remaining, your_character.level_9_spellslots])
        print(x)
        input("\nPress enter to return to the main menu.\n\n--------\n\n")
    except AttributeError:
        input("You need to create a charcter first.\nPress enter to return to main menu.\n\n-------- \n\n")
```

Feature 3: Use a spell

This function will ask what spell slot level you'd like to use, then if you have at least one remaining will subtract a spell slot remaining.

If you do not have any spell slots remaining, it'll tell you that you need to rest before you can use more of that level spell.

```
def use_spell(your_character):
    try:
        spell_selection = input("What level spellslot are you using?\n\n--------\n\n")
        system('clear')
        if spell_selection == "1":
            if your_character.level_1_remaining >= 1:
                your_character.level_1_remaining -= 1
                system('clear')
                input("You have " + str(your_character.level_1_remaining) + " level 1 spellslots remaining.\nPress Enter to return to the Main Menu.\n\n-------- \n\n")
                return your_character
            else:
                input("You do not have enough spell slots remaining. You need to rest first!\n Press enter to continue.\n\n-------- \n\n")
        elif spell_selection == "2":
            if your_character.level_2_remaining >= 1:
                your_character.level_2_remaining -= 1
                system('clear')
                input("You have " + str(your_character.level_2_remaining) + " level 2 spellslots remaining.\nPress Enter to return to the Main Menu.\n\n-------- \n\n")
                return your_character
            else:
                system('clear')
                input("You do not have enough spell slots remaining. You need to rest first!\n Press enter to continue.\n\n-------- \n\n")
        elif spell_selection == "3":
            if your_character.level_3_remaining >= 1:
                your_character.level_3_remaining -= 1
                system('clear')
                input("You have " + str(your_character.level_3_remaining) + " level 3 spellslots remaining.\nPress Enter to return to the Main Menu.\n\n-------- \n\n")
                return your_character
```

Feature 4: Rest

This feature sets all the remaining spell slots to your maximum spell slots (for that level) effectively resetting your character allowing you to go ahead and use more spells.

```
def rest(your_character):
    try:
        your_character.level_1_remaining = your_character.level_1_spellslots
        your_character.level_2_remaining = your_character.level_2_spellslots
        your_character.level_3_remaining = your_character.level_3_spellslots
        your_character.level_4_remaining = your_character.level_4_spellslots
        your_character.level_5_remaining = your_character.level_5_spellslots
        your_character.level_6_remaining = your_character.level_6_spellslots
        your_character.level_7_remaining = your_character.level_7_spellslots
        your_character.level_8_remaining = your_character.level_8_spellslots
        your_character.level_9_remaining = your_character.level_9_spellslots
        input("You are all rested, and spellslots have been restored.\nPress enter to continue.\n\n-------- \n\n")
        return your_character
    except AttributeError:
        input("It looks like your character creation was interupted. Please continue with character creation. \nPress enter to return to the main menu.\n\n-------- \n\n")
```

Feature 5:
Saving your character

First the “data” needs the made by assigning all your character data to dictionary key:values. Then it writes these as header:row into a CSV file using DictWriter.

Once done it’ll give you a confirmation message.

```
def get_data(your_character):
    data = {
        "dndclass": your_character.dndclass,
        "level": your_character.level,
        "cantrips_known": your_character.cantrips_known,
        "level_1_spellslots": your_character.level_1_spellslots,
        "level_1_remaining": your_character.level_1_remaining,
        "level_2_spellslots": your_character.level_2_spellslots,
        "level_2_remaining": your_character.level_2_remaining,
        "level_3_spellslots": your_character.level_3_spellslots,
        "level_3_remaining": your_character.level_3_remaining,
        "level_4_spellslots": your_character.level_4_spellslots,
        "level_4_remaining": your_character.level_4_remaining,
        "level_5_spellslots": your_character.level_5_spellslots,
        "level_5_remaining": your_character.level_5_remaining,
        "level_6_spellslots": your_character.level_6_spellslots,
        "level_6_remaining": your_character.level_6_remaining,
        "level_7_spellslots": your_character.level_7_spellslots,
        "level_7_remaining": your_character.level_7_remaining,
        "level_8_spellslots": your_character.level_8_spellslots,
        "level_8_remaining": your_character.level_8_remaining,
        "level_9_spellslots": your_character.level_9_spellslots,
        "level_9_remaining": your_character.level_9_remaining,
    }
    return data

loaded_data = []

your_character = Full_Caster("Nothing1", 0)

def save_character(your_character):
    try:
        data = get_data(your_character)
        with open("your_character_file.csv", 'w') as your_character_file:
            writer = csv.DictWriter(your_character_file, fieldnames=data.keys())
            writer.writeheader()
            writer.writerow(data)
        input("Your character has saved successfully.\nPress Enter to return to the main menu.\n\n-------- \n\n")
    except AttributeError:
        input("It looks like your character creation was interupted. Please continue with character creation. \nPress enter to return to the main menu.\n\n-------- \n\n")
```

Feature 6: Loading character

From the CSV file created during saving the character, it reads back the information and assigns all the data back to the class instance relevant to the dnd class, and returns your character ready for use the next time you open the app.

```
def load_character():
    with open ('your_character_file.csv', 'r') as your_character_file:
        heading = next(your_character_file)
        reader = csv.reader(your_character_file)
        for row in reader:
            loaded_data = row
    if loaded_data[0] == "Wizard":
        your_character = Full_Caster("Wizard", 0)
        your_character.level = int(loaded_data[1])
        your_character.cantrips_known = int(loaded_data[2])
        your_character.level_1_spellslots = int(loaded_data[3])
        your_character.level_1_remaining = int(loaded_data[4])
        your_character.level_2_spellslots = int(loaded_data[5])
        your_character.level_2_remaining = int(loaded_data[6])
        your_character.level_3_spellslots = int(loaded_data[7])
        your_character.level_3_remaining = int(loaded_data[8])
        your_character.level_4_spellslots = int(loaded_data[9])
        your_character.level_4_remaining = int(loaded_data[10])
        your_character.level_5_spellslots = int(loaded_data[11])
        your_character.level_5_remaining = int(loaded_data[12])
        your_character.level_6_spellslots = int(loaded_data[13])
        your_character.level_6_remaining = int(loaded_data[14])
        your_character.level_7_spellslots = int(loaded_data[15])
        your_character.level_7_remaining = int(loaded_data[16])
        your_character.level_8_spellslots = int(loaded_data[17])
        your_character.level_8_remaining = int(loaded_data[18])
        your_character.level_9_spellslots = int(loaded_data[19])
        your_character.level_9_remaining = int(loaded_data[20])
        return your_character
    elif loaded_data[0] == "Paladin":
        your_character = Half_Caster("Paladin", 0)
        your_character.level = int(loaded_data[1])
        your_character.cantrips_known = int(loaded_data[2])
        your_character.level_1_spellslots = int(loaded_data[3])
        your_character.level_1_remaining = int(loaded_data[4])
        your_character.level_2_spellslots = int(loaded_data[5])
        your_character.level_2_remaining = int(loaded_data[6])
        your_character.level_3_spellslots = int(loaded_data[7])
        your_character.level_3_remaining = int(loaded_data[8])
        your_character.level_4_spellslots = int(loaded_data[9])
        your_character.level_4_remaining = int(loaded_data[10])
        your_character.level_5_spellslots = int(loaded_data[11])
        your_character.level_5_remaining = int(loaded_data[12])
        your_character.level_6_spellslots = int(loaded_data[13])
        your_character.level_6_remaining = int(loaded_data[14])
        your_character.level_7_spellslots = int(loaded_data[15])
        your_character.level_7_remaining = int(loaded_data[16])
        your_character.level_8_spellslots = int(loaded_data[17])
        your_character.level_8_remaining = int(loaded_data[18])
        your_character.level_9_spellslots = int(loaded_data[19])
        your_character.level_9_remaining = int(loaded_data[20])
        return your_character
```

Why do these classes look so WET?

- Through my code you've probably seen that everything is done separately for Wizard and Paladin. While they use the same information, they use different **amounts**.
- Wizards are known as **full casters**, and therefore get much more than their **half caster** (Paladin) counterparts.
- Since there are a few of each (full & half caster), this is a more future-looking way to be able to add classes in the future will fit right into Full or Half caster Classes.

Error Handling

Value Error

Get_Paladin converts user input to an integer to determine level.

If user inputs anything other than an integer, or higher than 20 it creates a value error.

Except catches the error and asks for a valid result.

```
def get_paladin_level():
    input_level = 0
    your_character = None
    while your_character == None:
        try:
            system('clear')
            input_level = input("What level is your Paladin? Please choose from 1-20.\n\n-------- \n\n")
            system('clear')
            int_input = int(input_level)
            if int_input == 1:
                your_character = Half_Caster("Paladin", 1)
                return your_character
```

```
    elif int_input == 20:
        your_character = Half_Caster("Paladin", 20, 0, 4, 3, 3, 3, 2)
        return your_character
except ValueError:
    input("That's not a valid response, please only enter a number from 1-20.\nPress Enter to continue.\n\n-------- \n\n")
```

Using conditionals to avoid errors.

By using else for anything other than an expected answer there's no need to force an error.

A cleaner example of how to request level than get paladin (done both for error handling).

```
def get_wizard_level():
    input_level = 0
    your_character = None
    while input_level != "quit":
        system('clear')
        input_level = input("What level is your Wizard?\n\n-------- \n\n")
        system('clear')
        if input_level == "quit":
            quit()
        elif input_level == "1":
            your_character = Full_Caster("Wizard", 1, 3, 2, 2)
            return your_character
        elif input_level == "2":
            your_character = Full_Caster("Wizard", 2, 3, 3, 3)
```

```
    elif input_level == "20":
        your_character = Full_Caster("Wizard", 20, 5, 4, 4, 3, 3, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 1, 1, 1, 1)
        return your_character
    else:
        print("That's not a valid option, please try again.")
        input("Enter to continue.\n\n-------- \n\n")
```

Attribute Error

Late into creating this application I realized you could cancel character creation during the process, creating this attribute error if your_character ever needed to be passed.

Needed to add the error handling for this circumstance, alternatively could change character creation to account for this.

```
def save_character(your_character):
    try:
        data = get_data(your_character)
        with open("your_character_file.csv", 'w') as your_character_file:
            writer = csv.DictWriter(your_character_file, fieldnames=data.keys())
            writer.writeheader()
            writer.writerow(data)
        input("Your character has saved successfully.\nPress Enter to return to the main menu.\n\n-------- \n\n")
    except AttributeError:
        input("It looks like your character creation was interupted. Please continue with character creation. \nPress enter to return to the main menu.\n\n-------- \n\n")
```

File Not Found

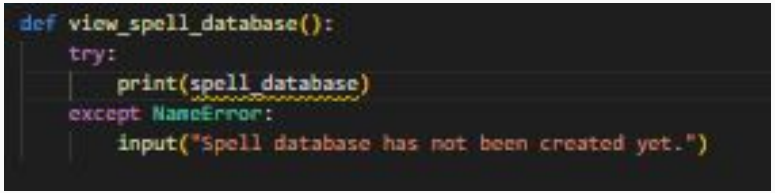
If the character file was renamed, moved, deleted or corrupted then load_character wouldn’t be able to call on this file. Instead of needing to have one as sometimes they will just not have saved one, giving a message is more appropriate.

```
try:
    your_character = load_character()
except FileNotFoundError:
    input("Save file is not found. Please create a character.\nPress Enter to continue.\n\n-------- \n\n")
main_menu(your_character)
```

Name Error

spell_database is planned to be added in the future, and the function was made ‘in advanced’.

Alternatively if spell_database was made but it wasn’t properly passed in, or returned then this error could also occur.



Testing:

	A	B	C	D	E	F	G	
1	Process	Step	Task	Instructions	Status	Expected Result	Actual Result	Co
2		1	Input Class selection	Press 1 in MM. Enter Paladin or Wizard.	Pass	Continue to level input	As intended.	
3		2	Input class level	Enter a number 1-20.	Pass	Character created. RTM. Prints expected class and level.	As intended.	
4	"Rest" your	3	Check available spell slots	Press 2 in MM.	Pass	Prints all available spell slots. RTM. See ReadMe for external validation.	As intended.	
5	Character	4	Use a spell (single or multiple)	Press 3 in MM. Type 1-9 to use level 1-9 slot. Repeat as desired.	Pass	Input spell slot used. RTM.	As intended.	
6		5	Check available spell slots	Press 2 in MM. Confirm amount remaining is reduced by what you used.	Pass	Prints all **remaining** spell slots. RTM.	As intended.	
7		6	Take a Long Rest	Press 4 in MM.	Fail	Prints "You are all rested..." and restores spell slots to max.	Error message "..."	
8		7	Check available spell slots	Press 2 in MM. Confirm amount remaining has been reset to max.	Nil	Same output as step 3.	N/A	
9								
10		1	Input Class selection	Press 1 in MM. Enter Paladin or Wizard.	Pass	Continue to level input	As intended.	
11		2	Input class level	Enter a number 1-20.	Pass	Character created. RTM. Prints expected class and level.	As intended.	
12	Save & Load	3	Check available spell slots	Press 2 in MM.	Pass	Prints all available spell slots. RTM. See ReadMe for external validation.	As intended.	
13	your character	4	Use a spell (single or multiple)	Press 3 in MM. Type 1-9 to use level 1-9 slot. Repeat as desired.	Pass	Input spell slot used. RTM.	As intended.	
14		5	Check available spell slots	Press 2 in MM. Confirm amount remaining is reduced by what you used.	Pass	Prints all **remaining** spell slots. RTM.	As intended.	
15		6	Save your character	Press 8 in MM. Then type quit in MM.	Pass	Character data should be saved in CSV file. RTM	As intended.	
16		7	Quit the application	Type quit in MM.	Pass	Program should close (exit, quit).	As intended.	
17		8	Open app.	Run program.	Pass	View main menu.	As intended.	
18		9	Check available spell slots	Press 2 in MM. Compare to step 5.	Pass	Prints all remaining spell slots from last use.	As intended.	
19								

Main issues:

- The application runs the variable your_character through a lot of functions.
- Any changes can mess around with getting all the correct data for your_character, or its ability to return and be passed into other functions.
- End-to-End testing is most suitable due to the complexity of functions working together.

Thanks for listening!