Final project briefs

PA Consulting: Intro to Programming with Python by Women in Tech

Content by Michelle Yip

Your final project allows you to showcase everything you've learnt during this course!

There are three briefs to choose from which are each text based games, but you are also welcome/encouraged to come up with your own idea - feel free to discuss this with one of the instructors for guidance.

The text based games should be played via the Python command line output, using the `print()` function to output information to the user and the `input()` function to get user input at different stages of your game.

Each brief is split into different requirements:

- Must haves
- Could haves
- Further ideas for extension

Choose your favourite brief and get going! If you're a bit stuck with making a start then there is some basic demo code for each brief at the bottom of this document; you can also contact the instructors for support. There will be some drop in sessions organised with the instructors where you can get help on your projects as well.

We encourage you to extend the game in a unique way and add as much functionality as you can!

How to structure your work

Create a new folder named `final-project` inside your `python-course` folder where all your project files will be located

You should start by choosing your project brief and deciding which could haves and further extensions you would like to try and incorporate - make a bit of a step by step plan.

Then, start by working solely on creating the most basic functionality that you can, such as one or two of the must-haves. Once you have this working, you should commit your work on GitHub desktop.

You can then add more functionality, step by step. Make sure you run and test your work at each step, and keep committing at each working checkpoint as well. If you go badly wrong and your code *stops* working and you're not sure why, we can easily roll back to your latest commit which should have working code! (Ask an instructor for help if this happens)

Structuring your work in this way means you should always have at least *some* working version of your project to show at the end of the course. If you try to code a huge project all at once without testing and checking it at each stage, it will be nearly impossible to debug and you might end up with no working code whatsoever!

Option 1 - Quiz game

Create a quiz program which will ask users a list of questions and then produces a result at the end

Must haves

- When the quiz starts, ask for (and store) the player's name
- Ask the player at least one question e.g.
  - "What colour is the sky?" A:Blue
  - "What is 1+1?" A:2
- Compare the player's answer to the correct answer and print out whether their answer was correct or not with their name, e.g.

  `Congrats Michelle, you got the answer right!`

  `Better luck next time Michelle, your answer was incorrect`

- Accept all answers in any form e.g. uppercase or lowercase
- Ask if the player wants to play again

Could haves

- Ask the player multiple questions
- Retrieve a random question from your list and ask the player so that the game is different each time
- Keep track of the player's score and print it out at the end

Further ideas for extension

- Allow player to select a difficulty or theme
- Take your questions from an API or file
- Create a scoreboard file which can save multiple player scores
- Use the scoreboard file to tell the player their ranking at the end of the quiz
- Add a timer - the faster the time the higher the score
- Add a time limit - 0 marks if the time limit is reached
- Add a hint option or skip option
- If the answer is close - have a message to prompt the player

- Have a mixture of open end questions and multiple choice questions

Option 2 - Adventure game

Create an adventure game where your character will complete a level by fighting monsters etc

Must haves

- Ask the player if they want to start a game
- Your character will begin with a fixed statistic (stored in a dictionary, example in code block below)
  - Their attack strength is 10
  - Their health is 50
- There is also a monster which they will fight (statistics for the monster also stored in a dictionary, example in code block below)
  - The monster's attack strength is 10
  - The monster's health is 60
- Continuously ask the player whether they want to select one of the following options (add an error message if option is not valid)
  1. Attack
  2. View stats
- If 'View stats' is selected, all statistics for the player and monster should be printed
- If 'Attack' is selected, the player should **at random** either hit, miss, or be attacked themselves by the monster
  - If they hit the monster, the monster's health should decrease by the player's attack strength
  - If they miss the monster, no change in health occurs
  - If the monster attacks the player, the player's health should decrease by the monsters's attack strength
- Whoever has 0 health first loses and the other wins

In [ ]:

```
# Option 3 guidance
character_stats = {
    'attack': 10,
    'health': 50
}

monster_stats = {
    'attack': 10,
    'health': 60
}
```

Could haves

- Add a few more monsters before the player finishes the journey
- Randomise the monster's stats - some are harder to defeat

- You could have a finish line for your journey (e.g. at 30 steps) and every round you roll a (5 sided) dice and move forward so many steps. Every step can have an event e.g.
    - Found a health potion (+20 health)
    - A monster appeared
    - You found a sword (+20 attack)
    - Found a revive potion (if you reach 0 health, you will automatically regenerate 20 health)
- Add more complex moves and options for fighting monsters, such as attack, defend and dodge
- Adjust the game so that it is a different kind of adventure (not fighting monsters)
- Increase the player's attack strength when they beat a monster

Further ideas for extension

- Get the monster/other character data from an API, such as the Pokemon API
- Add multiple levels (e.g. 5 levels) at each level the monster stats increase and the game gets harder
- Add some files which will store your character stats and monster stats, or keep track of scores
- Add an option to select a character you want to play (each with different starting stats)
- Have more statistics which affect the game other than attack strength

Option 3 - Guess the word

Create a word guessing game, where the player guesses letter by letter

Must haves

- Have a word for the player to guess
- Display how many letters the word has to the player, represented by blank spaces e.g. `_ _ _ _`
- Allow the player to input a letter
- Add an error message if input is more than one letter
- Check if the letter exists in the word
- If it exists - replace the blank spaces with the letter e.g. `_ o _ _`
- If it does not exist - leave blank
- Continuously ask for another letter until the word is complete
- Print number of attempts player needed

Could haves

- Store a list of different words in a list and randomly select a word for each game
- Add a max limit of times the player can make an attempt to guess a letter
- Allow players to guess the whole word, adding an error message if the inputted word length does not match the guessing word length

Further ideas for extension

- Create a scoreboard stored in a file
- Use an API or file to generate words
- Create different lists of words and assign them a theme or difficulty level which the player can choose from
- Have different rounds

Starting code examples

In [ ]:

```
# Option 1 - Quiz game

questions = [
    {
        "question": "What is the capital of Germany? ",
        "answer": "Berlin"
    }
]

score = 0

for question in questions:
    player_ans = input(question['question'])
    if player_ans == question['answer']:
        print('Correct!')
        score += 1
    else:
        print('Incorrect')

print(f"Your score was {score}")
```

In [ ]:

```
# Option 2 - Adventure game

from random import randint

character_stats = {
    'attack':10,
    'health':50
}

monster_stats={
    'attack':10,
    'health':60
}

while (character_stats['health'] != 0 and monster_stats['health'] != 0):
    action = input("There is a monster in front of you! \n 1. Attack \n 2. Vie
    if (action == "2" or action.lower == "view_stats"):
        print (f"Your stats {character_stats}")
        print (f"Monster stats {monster_stats}")
    else:
        print(f"You decided to attack!")
        move = randint(0, 2)
        if move == 0:
            print(f"You hit the monster!")
            monster_stats['health'] = monster_stats['health'] - character_stat
            print (f"Monster took {character_stats['attack']} damage")
        elif move == 1:
            print(f"You missed the monster!")
        else:
            print(f"The monster attacked you!")
            character_stats['health'] = character_stats['health'] - monster_st
            print (f"You lost {monster_stats['attack']} health")

if (character_stats['health'] == 0):
    print("You lost the fight!")
elif(monster_stats['health'] == 0):
    print("You won the fight!")
```

In [ ]:

```
# Option 3 - Guess the word

word = list("hello")
ans = []
length = len(word)

for i in range(length):
    ans.append("_")

print(' '.join(ans))

while ans!= word:
    letter = input("Enter a letter: ")
    if letter in word:
        # Using the enumerate() function in a for loop lets you also access th
        # So we can access the index of the letter in the word with the 'posi
        # and the actual letter with the 'l' variable (e.g. 'e')
        for position, l in enumerate(word):
            if l == letter:
                ans[position] = str(letter)
        print(' '.join(ans))
    else:
        print(' '.join(ans))
```