Bootcamp Group Project

Table of Contents

we think code_

Introduction

Congratulations! WeThinkCode_ has selected you as one of its interns. Your first task is to help WeThinkCode_ manage bootcamp administration by creating a WeThinkCode_ Bootcamp Helper.

The Bootcamp Helper is a programme with a number of features that assist WTC with automating internal bootcamp processes and managing bootcamps. **Each team member will be responsible for building a feature.**

You are allowed to build as many features as you are able to however you will need to complete at least **one** feature per team member.

Each candidate will create ONE of the following features:

1. **Username generator**

A feature that generates a unique bootcamp username based on a format.

2. **Team Allocator**

A feature that allocates users into different teams for the group project. Teams are allocated based on their track (experienced or non experienced), campus and bootcamp type (virtual or physical).

3. **Registration station**

A feature that displays the information of candidates who have registered to a bootcamp, their booking information and gives them an opportunity to confirm or change their booking.

4. **Bootcamp exercise marker**

A program that administers and grades a multiple-choice exam. It starts off with 5 questions and repeats the questions that were incorrectly answered until all questions have been correctly answered.

5. Bootcamp final selector

A program that determines if a candidate has passed or failed bootcamp based on their results across different assessments.

Features

Each feature has a skeleton code and unit test. The skeleton code is the code you will build from.

In order to do the project you will need to do the following:

- Download and unzip the Bootcamp Helper Resources folder
- Each feature has a skeleton code that you will start coding from. The skeleton code is named after the feature.
- Each Feature has a set of unit tests to run. The unit tests are named also named test_[Name of feature]
- The process of running the unit tests is detailed at the end of the document.
- Start coding your feature using the existing structure of the skeleton code.
- **Do Not change the name of the file or any of the function names.**

Username Generator

A feature that generates a unique bootcamp username based on a format and personal information.

The program should be structured in the following way:

1. Your program should prompt a user to input Their First Name, Last Name, Campus and the cohort year they are entering. - It is your choice how you will expect this input, one by one or in a single string
2. Your program should validate user input in the following ways:
    a. First name and last name name should not contain digits
    b. Campus should be a valid campus
    c. Cohort year should be a valid cohort year - a candidate can't join a cohort in the past
3. You will have a function that produces the username from the input provided.
4. The user will then be asked if the final username is correct. Let them know what the format of the username is and if the final username is correct.

See below for an example of the final bootcamp username based on personal information:

First Name: Lungelo

Last Name: Mkhize

Cohort Year: 2022

Final Campus: Durban

Final username:

elomkhDBN2022

ELO - Last 3 letters of first name (if their name is less than 3 letters you should add the letter O at the end)

MKH - First 3 letters of their last name (if their name is less than 3 letters you should add the letter O at the end)

DBN - Final Campus selection - Johannesburg is JHB, Cape Town is CPT, Durban is DBN, Phokeng is PHO

2022 - The cohort year they are entering

Team Allocator

A program that allocates users into different teams for the group project. A list (student_list()) with dummy data has been provided to you.

The program should be structured in the following way:

1. Use the student_list() function to get a list students with the following structure:
   elomkhDBN2022 - 4 April - Johannesburg Physical - seat 3

   ddhaalJHB2022 - 2 May - Cape Town Virtual

2. Group users according to their **campus** (Johannesburg or CapeTown or Durban or Phokeng)

   NB: Use the dbn_campus_students(student_list) function to group Durban students according to their campus, cpt_campus_students(student_list) function to group Cape Town students according to their campus, jhb_campus_students(student_list) function to group Johannesburg students according to their campus, and the nw_campus_students(student_list) function to group Phokeng students according to their campus.

3. Group users according to their **mode** of participation (Physical or Virtual)

   NB: Use the dbn_physical_students(dbn_students) function to get the Durban students mode of participation,cpt_physical_students(cpt_students) function to get the Cape Town students mode of participation, jhb_physical_students(jhb_students) function to get the Johannesburg students mode of participation, and nw_physical_students(nw_students) function to get the Phokeng students mode of participation. Use the get_virtual_students(student_list) function to get the Virtual students mode of participation.

4. Teams should have at least **4 candidates**

a. **If** there is a surplus of users, add the remaining candidates in a team of their own

NB: Use the dbn_physical_teams(dbn_physical_students) function to create the teams of 4 for the Durban campus, cpt_physical_teams(cpt_physical_students) function to create the teams of 4 for the Cape Town campus, jhb_physical_teams(jhb_physical_students) function to create the teams of 4 for the Johannesburg campus, nw_physical_teams(nw_physical_students) function to create the teams of 4 for the Phokeng campus. Use the virtual_teams(virtual_students) function to create the teams of 4 for the Virtual campus.

5. Write all the information into a **text file name the file campus_teams.txt, and virtual_teams.txt**
   NB: Use the dbn_teams_file(durban_physical_teams) function to write the Durban teams into a file, cpt_teams_file(cpt_physical_teams) function to write the Cape Town teams into a file, jhb_teams_file(jhb_physical_teams) function to write theJohannesburg teams into a file, nw_teams_file(nw_physical_teams) function to write the Phokeng teams into a file and the virtual_teams_file(virtual_teams) function to write the Virtual teams into a file.

Registration Station

A program that displays the information of candidates who have registered to a bootcamp, their booking information and gives them an opportunity to confirm or change their booking.

The program should be structured in the following way:

1. You should have a text file with dummy bootcamp registrations (bootcampers.txt). The bootcamp information should be structured in the following way:

elomkhDBN2022 - 4 April - Johannesburg Physical - No prior experience

ddhaalJHB2022 - 2 May - Cape Town Virtual - Prior Experience

elomkhDBN2022 - username

4 April - Bootcamp date

Johannesburg Physical - Bootcamp venue

No prior experience - Experience or no experience

2. Your program should request a username, and read the text file for a username that matches one that has been input.
3. Your program should then print out the information contained after the username (from the date), ask the user if it is correct and prompt them to confirm or deny.
4. If the user denies any information, you should prompt them to input the correct information and change their details.
5. Once you have gone through all of the information and confirmed the correct information you should print out the details.

Functions used for testing:

correct_or_incorrect() - Prompt to ask if details are correct or not, must return “correct” or “incorrect”.

correct_details() - Prompt to correct and write user details to text file, prompt field includes:

Username, Date, Location, Experience.

find_username(file_name) - Main function for running Registration Station, which include:

- get username input from user
- check if username exists and print corresponding detail
- return corresponding information for specific username from text file as required

Bootcamp Exercise Marker

The program should be structured in the following way:

1. You should have a text file with a list of multiple choice questions and answers structured in the following way:
    a. 1. What are variables used for? A - Variables are used to store information to be referenced and manipulated in a computer program and can change throughout the life cycle of a program. B - Variables are used to store information that will never change C - sdjhdsjhfdhjsdfdbfhsd.
    2. How many leaves are in a tree? A - 23 B - 45 C - 78

2. Your program should administer a multiple choice exam out of 5. The first round will print 5 random questions from the bank of questions, accept an answer (A, B or C) and print out a final score at the end of the round.
3. The next round will consist of the questions that were answered incorrectly, once that round is complete it will print out the answer and continue until all questions have been answered correctly.

Bootcamp Final Selector

A program that determines if a candidate has passed or failed bootcamp based on their results across different assessments.

The program should be structured in the following way:

1. A student's results should be stored in text files that contain a list of usernames and scores.

Example:

Format (username - score)

Formula:

Final_results = ((exam_results / 100) * 60) + group_project + ((daily_exercises / 30) * 20)

Average = (final_results / 100) * 10

Exam_results:
StudentA - 50
StudentB - 30 (30%) (18/60)
StudentC - 40

group_project:

StudentA - 8
StudentB - 10 (50%) (10/20)
StudentC - 12

Daily_exercises:

StudentA - 18
StudentB - 20 (66%) (13/20)
StudentC - 22
41/100

2. You should have 3 results. One for the exam (out of 100), one for the group project (out of 20) and one for daily exercises (out of 30). The exam will weigh 60%, the group project 20% and the daily exercises 20%.
3. A user should be able to enter a username via the command line and your program should fetch the user's results from each file and calculate a final average mark out of 100.
4. A candidate has passed if they have an overall average above 80%.
5. There are two different types of passes. A first class pass which is over 90% average and a normal pass which is over 80%.
6. Your program should print out the final average score out of 100, whether they passed or failed, whether they passed first class or second class, and the score for each category.

Testing process

As part of your group project submissions, we want to introduce you to **test driven development**, which is a programming practice where tests are written for your code solutions and you are required to write code that passes those tests.

To run a unit tests to ensure that your code works as required, do the following:

1. Open the terminal
2. Ask your group mentor to help you navigate to **Home/Desktop/Bootcamp Helper Resources**
3. NB: Make sure you copied your unzipped project folder and files to the desktop before you start

4. Once you're in the directory in step 2, type in **python -m unittest test_[feature name].py**

Replace [feature name] with the name of the feature you're working on to run the correct unit tests. Also check the Bootcamp Helper Resources directory for a unit test that corresponds with the feature you're working on, i.e. for the team allocator feature the unit test *test_team_allocator.py* is the unit test you should run. For this feature, in the terminal you'd have *Home/Desktop/Bootcamp Helper Resources/ python test_team_allocator.py*

Use the test results and the comments in the skeleton code provided to you to guide you on what is missing in your code.

Make sure that your code compiles before submitting. Your code has to compile before a project mentor can mark it. Try to make sure that your project feature passes as many unit tests as possible.

Your project mentor will use the number of unit tests passed to give feedback on your performance.