COMP 202

Winter 2022

Assignment 1

Due: Sunday, February 6^{th}, 11:59 p.m.

Please read the entire PDF before starting. You must do this assignment individually.

Question 1:	15 points
Question 2:	85 points
	100 points total

It is very important that you follow the directions as closely as possible. The directions, while perhaps tedious, are designed to make it as easy as possible for the TAs to mark the assignments by letting them run your assignment, in some cases through automated tests. While these tests will never be used to determine your entire grade, they speed up the process significantly, allowing the TAs to provide better feedback and not waste time on administrative details. Plus, if the TA is in a good mood while grading, then that increases the chance of them giving out partial marks. :)

To get full marks, you must follow all directions below:

- Make sure that all file names and function names are **spelled exactly** as described in this document. Otherwise, a 50% penalty per question will be applied.
- Make sure that your code **runs without errors**. Code with errors will receive a very low mark.
- Write your name and student ID in a comment at the top of all `.py` files you hand in.
- Name your variables appropriately. The purpose of each variable should be obvious from the name.
- **Comment your code.** A comment every line is not needed, but there should be enough comments to fully understand your program.
- Avoid writing repetitive code, but rather call helper functions! You are welcome to add additional functions if you think this can increase the readability of your code.
- Lines of code should NOT require the TA to scroll horizontally to read the whole thing.
- Vertical spacing is also important when writing code. Separate each block of code (also within a function) with an empty line.
- **Up to 30% can be removed for bad indentation of your code, omission of comments, and/or poor coding style (as discussed in class).**

Hints & tips

- **Start early.** Programming projects always take more time than you estimate!
- Do not wait until the last minute to submit your code. **Submit early and often**—a good rule of thumb is to submit every time you finish writing and testing a function.
- Write your code **incrementally**. Don't try to write everything at once. That never works well. Start off with something small and make sure that it works, then add to it gradually, making sure that it works every step of the way.
- Read these instructions and make sure you understand them thoroughly before you start. Ask questions if anything is unclear!

- Seek help when you get stuck! Check our discussion board first to see if your question has already been asked and answered. Ask your question on the discussion board if it hasn't been asked already. Talk to your TA during office hours if you are having difficulties with programming. Go to an instructor's office hours if you need extra help with understanding a part of the course content.
  - At the same time, beware not to post anything on the discussion board that might give away any part of your solution—this would constitute plagiarism, and the consequences would be unpleasant for everyone involved. If you cannot think of a way to ask your question without giving away part of your solution, then please drop by our office hours.
- If you come to see us in office hours, please do not ask "Here is my program. What's wrong with it?" We expect you to at least make an effort to start to debug your own code, a skill which you are meant to learn as part of this course. And as you will discover for yourself, reading through someone else's code is a difficult process—we just don't have the time to read through and understand even a fraction of everyone's code in detail.
  - However, if you show us the work that you've done to narrow down the problem to a specific section of the code, why you think it doesn't work, and what you've tried to fix it, it will be much easier to provide you with the specific help you require and we will be happy to do so.

Revisions

Jan. 29

- Question 1: Added **Note 2** about docstrings.
- Question 2: Corrected the name of `ask_question` function in its examples.
- Question 2: Added a clarification of the types of minutes and hours in `minutes_to_hour_string`, also added an example to demonstrate it. In particular, the number of hours should be an integer whereas the number of minutes should be float.
- Question 2: Corrected the examples for `get_trip_time_with_refill`. Also added an extra description for the function parameter `required_gallons`.
- Question 2: Corrected the amount of time shown in the example for `menu` function, and added a second example.

Jan. 30

- Question 2: The string returned by `minutes_to_hour_string` should have the minutes rounded to two decimal places.
- Question 2: Changed the examples for `get_cheapest_trip_cost` as they were confusing.

Part 1 (0 points): Warm-up

*Do **N**OT submit this part, as it will not be graded. However, doing these exercises might help you to do the second part of the assignment, which will be graded. If you have difficulties with the questions of Part 1, then we suggest that you consult the TAs during their office hours; they can help you and work with you through the warm-up questions. You are responsible for knowing all of the material in these questions.*

Warm-up Question 1 (0 points)

Practice with Number Bases:

We usually use base 10 in our daily lives, because we have ten fingers. When operating in base 10, numbers have a `ones` column, a `tens` column, a `hundreds` column, etc. These are all the powers of 10.

There is nothing special about 10 though. This can in fact be done with any number. In base 2, we have each column representing (from right to left) 1,2,4,8,16, etc. In base 3, it would be 1,3,9,27, etc.

Answer the following short questions about number representation and counting.

1. In base 10, what is the largest digit that you can put in each column? What about base 2? Base 3? Base n?
2. Represent the number thirteen in base 5.
3. Represent the number thirteen in base 2.
4. What is the number 11010010 in base 10?

Warm-up Question 2 (0 points)

Logic:

1. What does the following logical expression evaluate to?

```
(False or False) and (True and (not False))
```

2. Let a and b be boolean variables. Is it possible to set values for a and b to have the following expression evaluate as `False`?

```
b or (((not a) or (not a)) or (a or (not b)))
```

Warm-up Question 3 (0 points)

Expressions: Write a program `even_and_positive.py` that takes an integer as input from the user and displays on your screen whether it is true or false that such integer is even, positive, or both.

An example of what you could see in the shell when you run the program is:

```
>>> %Run even_and_positive.py
Please enter a number: -2
-2 is an even number: True
-2 is a positive number: False
-2 is a positive even number: False

>>> %Run even_and_positive.py
Please enter a number: 7
7 is an even number: False
7 is a positive number: True
7 is a positive even number: False
```

Warm-up Question 4 (0 points)

Conditional statements: Write a program `hello_bye.py` that takes an integer as input from the user. If the integer is equal to 1, then the program displays `Hello everyone!`, otherwise it displays `Bye bye!`.

An example of what you could see in the shell when you run the program is:

```
>>> %Run hello_bye.py
Choose a number: 0
Bye bye!

>>> %Run hello_bye.py
Choose a number: 1
Hello everyone!

>>> %Run hello_bye.py
Choose a number: 329
Bye bye!
```

Warm-up Question 5 (0 points)

Void Functions: Create a file called `greetings.py`, and in this file, define a function called `hello`. This function should take one input argument and display a string obtained by concatenating *Hello* with the input received. For instance, if you call `hello("world!")` in your program you should see the following displayed on your screen:

```
Hello world!
```

- Think about three different ways of writing this function.
- What is the return value of the function?

Warm-up Question 6 (0 points)

Void Functions: Create a file called `drawing_numbers.py`. In this file create a function called `display_two`. The function should not take any input argument and should display the following pattern:

```
  22
2   2
   2
 2
22222
```

Use strings composed out of the space character and the character '2'.

- Think about two different ways of writing this function.
- Try to write a similar function `display_twenty` which displays the pattern '20'

Warm-up Question 7 (0 points)

Fruitful Functions: Write a function that takes three integers `x`, `y`, and `z` as input. This function returns `True` if `z` is equal to 3 or if `z` is equal to the sum of `x` and `y`, and `False` otherwise.

Warm-up Question 8 (0 points)

Fruitful Functions: Write a function that takes two integers `x`, `y` and a string `op`. This function returns the sum of `x` and `y` if `op` is equal to `+`, the product of `x` and `y` if `op` is equal to `*`, and zero in all other cases.

Part 2

The questions in this part of the assignment will be graded.

The main learning objectives for this assignment are:

- Correctly create and use variables.
- Learn how to build expressions containing different type of operators.
- Get familiar with string concatenation.
- Correctly use `print` to display information.
- Understand the difference between inputs to a function and inputs to a program (received from the user through the function `input`).
- Correctly use and manipulate inputs received by the program from the function `input`.
- Correctly use simple conditional statements.
- Correctly define and use simple functions.
- Solidify your understanding of how executing instructions from the shell differs from running a program.

Note that this assignment is designed for you to be practicing what you have learned in the videos up to and including Lecture 11 (Functions 3). For this reason, you are NOT allowed to use anything seen after Lecture 11 or not seen in class at all. You will be heavily penalized if you do so.

For full marks, in addition to the points listed on page 1, make sure to add the appropriate documentation string (docstring) to *all* the functions you write. The docstring must contain the following:

- The type contract of the function.
- A description of what the function is expected to do.
- At least three (3) examples of calls to the function. You are allowed to use *at most one* example per function from this PDF.

Examples

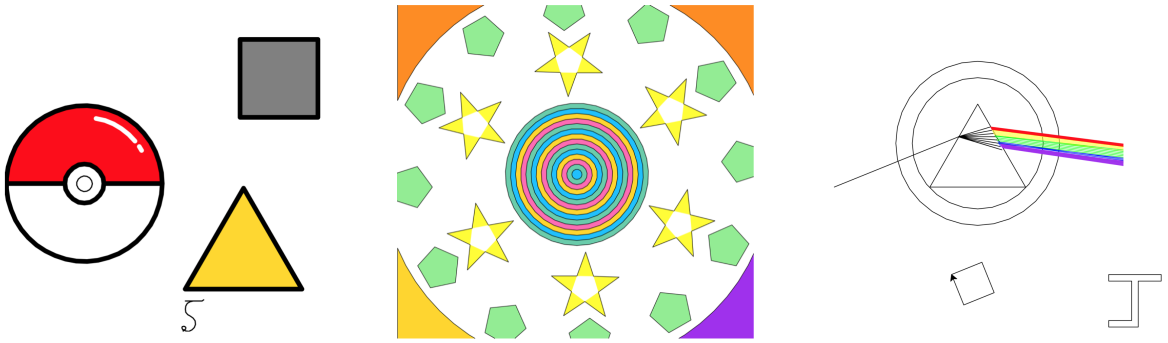
For each question, we provide several **examples** of how your code should behave. All examples are given as if you were to call the functions from the shell.

When you upload your code to codePost, some of these examples will be run automatically to test that your code outputs the same as given in the example. However, **it is your responsibility to make sure your code/functions work for any inputs, not just the ones shown in the examples.** When the time comes to grade your assignment, we will run additional, private tests that will use inputs not seen in the examples. You should make sure that your functions work for all the different possible scenarios. Also, when testing your code, know that mindlessly plugging in various different inputs is not enough—it's not the quantity of tests that matters, it's having tests that cover all of the possible scenarios, and that requires thinking about possible scenarios.

Furthermore, please note that your code files **should not contain any function calls in the main body of the program** (i.e., outside of any functions). Code that contains function calls in the main body **will automatically fail the tests on codePost and thus be heavily penalized**. It is OK to place function calls in the main body of your code for testing purposes, but if you do so, make certain that you remove them before submitting. You can also test your functions by calling them from the shell.

Safe Assumptions

For all questions on this assignment, you can safely assume that the **type** of the inputs (both to the functions and those provided to the program by the user) will always be correct. For example, if a function takes as input a string, you can assume that a string will always be provided. At times you will be required to do some input validation, but this requirement will always be clearly stated. Otherwise, your functions should work with any possible input that respect the function's description. For example, if the description says that the function takes as input a positive integer, then it should work with all integers greater than 0. If it mentions an integer, then it should work for *any* integer. Make sure to test your functions for edge cases!



Question 1: Turtle Art (15 points)

This question asks you to write a function `my_artwork()` in a file called `artwork.py`. The function should take **no inputs** and return nothing. It should draw a picture using the Turtle module. You are free to draw what you like, but your code/drawing must satisfy at least the following requirements:

- the drawing must include at least three shapes
- at least one shape must be drawn using a for loop
- at least one shape must be drawn using a function, with that function having at least two parameters that modify the shape being drawn in some way. (You cannot simply copy one of the functions we have written in our lectures - you must write your own function that is dissimilar to the ones seen in class.)
- the drawing must include at least two different colors
- everything must fit into the Turtle window; do not draw outside of the window or make the window larger. Assume a standard HD (1920x1080) resolution.
- the first letter of your first name must appear somewhere (you must sign your artwork!)
- there should be **no** calls to the `input` function
- **As always, do not call any functions (including `turtle.Turtle`) in the main body.**

Any submission meeting these requirements will obtain full marks, but you are encouraged to go beyond them. Our TAs will be showcasing their favorite submissions in the Slack. (Due to McGill policy, we cannot share students' names without permission, so the chosen artworks will be posted in the Slack without names. However, if you would like your name to appear with your artwork if the TA decides to post it in the Slack, then please write a sentence to that effect in your `README.txt` file.)

Note 1: Recall that you can import the `speed` function from turtle and then call `speed("fastest")` to speed up the drawing routines, so that you don't waste time when testing your code.

Note 2: While writing docstrings for the functions in this question there is no need to include examples but type contract and description are still needed.

Also, a reminder to please only use the functions from the Turtle module that we have seen in class, or you will lose marks. There is one exception: you can use the `circle` function from Turtle module. `circle(r)` takes a radius `r` as argument and draws a circle of the given radius. You can also specify a second integer argument for the extent of the circle to draw, e.g., `circle(r, 90)`, which will draw only a quarter of a circle (90 degrees).

Some submissions from students of previous years (with their size scaled down to fit) can be found at the top of this page.

Question 2: Road Trip (85 points)

For eons, or at least the eons where forms of transportation have existed, humankind has traveled from one place to another, making plans and logistical decisions along the way.

In this question, we will do some planning for a road trip of our own. We will focus primarily on the fuel required to travel over a certain distance, and see if we have enough money to refuel mid-way through the trip, and how much time and money the trip will cost.

This planning will require a lot of different, inter-connected calculations. Whenever you are faced with this kind of complex problem, which will most often be the case when programming, it is best to break it down into smaller sub-parts, or functions. Put each different computation into its own function. This organizational paradigm serves three purposes. First, it makes your code file much easier to read through and navigate. Secondly, and most importantly, it lets you focus on one thing at a time. Finally, it helps with testing. Each time you write a function, you should always **test it thoroughly**, giving it different inputs and checking that the output is what you expect it to be. It is easier to test functions when they are small and do not do too much on their own. Only once you are certain that a function is working properly, then you can move onto the next function.

Functions also help in code re-use. By putting some computation into a function, if we then needed to perform the same computation again, we can just call that same function instead of copy-and-pasting the code to a different location (as long as the function returns a value). Copy-and-pasting is bad in programming! One of the keys of programming is **don't repeat yourself** ('DRY'). Whenever you have repeated several lines of code, ask yourself if there is a way to simplify the code by putting it into a function instead (and then just calling it when needed). We will also be checking your code to make sure that not too much code is repeated throughout your file.

With the above in mind, we will begin writing our code. Write all your code for this question in a file called `road_trip.py`.

First, define the following global variables at the top of your file:

- `MILES_PER_HOUR = 60.0` (represents how many miles per hour your vehicle will be travelling)
- `MILES_PER_GALLON = 30.0` (how many miles can be travelled by expending one gallon of fuel)
- `MINUTES_PER_HOUR = 60.0` (the number of minutes in an hour)
- `KILOMETRES_PER_MILE = 25146 / 15625` (the ratio of kilometres to miles)

Note that the names of these global variables are in all-caps. Variables in all-caps are by convention known to be 'constant' variables, or 'constants.' A constant is a variable whose value will never change throughout the execution of the program (e.g., it does not depend on user input nor on the result of any calculation). It is set once, typically at the top of your code file, and not modified afterwards in the code. (It is, however, always possible for you to modify the value of a constant yourself, by replacing the initial value on the line where it is defined.)

The examples shown for the functions below will assume that the constants are set to the above values. However, during testing, we can modify the values of these global variables and check that your code still works properly. You should make sure that your code uses these global variables and that you do not instead 'hard-code' the values (meaning writing the values above directly into the code instead of using the global variables).

Write the following functions in the file `road_trip.py`.

- `ask_question(question, option1, option2, option3, option4)`: Takes five strings: the first corresponding to a question, and the others corresponding to possible answers to the question. Prints out the question and the four choices, placing a number from 1 to 4 adjacent to each choice. Asks the user to enter their answer, and returns their input. If a choice is equal to the empty string, then do not print that choice nor the adjacent number to the screen.

  (Note: Text in examples with a light-gray background corresponds to input entered by the user.)

```
>>> user_response = ask_question('Which color is your favorite?', 'red', \
                                                    'lime', 'blue', 'gray')
Which color is your favorite?
1 red
2 lime
3 blue
4 gray
Your answer: 4
>>> print(user_response)
4

>>> user_response = ask_question('Do you like pineapple pizza', 'yes', '', \
                                                     '', '')
Do you like pineapple pizza
1 yes
Your answer: 1
>>> print(user_response)
1
```

- `km_to_miles(km)`: Takes a non-negative float as input corresponding to an amount in kilometres, and returns that amount converted into miles, as a float rounded to 2 decimals.

```
>>> km_to_miles(2.0)
1.24

>>> km_to_miles(3.5)
2.17
```

- `minutes_to_hour_string(minutes)`: Takes a non-negative float as input corresponding to a number of minutes, and returns a string expressing the number in terms of minutes and hours. *The number of hours should be an integer whereas the number of minutes should be float.* If the number of minutes is smaller than 60, then the function should return `'X minutes'`, where `X` is the number of minutes. If the number of minutes is greater than 60, then the function should return `'Y hours and X minutes'`, where `Y` is the number of hours and `X` is the number of minutes left over (rounded to two decimal places). If the number of minutes can be evenly divided, that is, there are no minutes left over, then the function should return `'Y hours'`, where `Y` is the number of hours. Furthermore, if there is only *exactly* one hour and/or one minute, then the `s` should be omitted from the end of hour and/or minute, respectively.

```
>>> minutes_to_hour_string(140.0)
'2 hours and 20.0 minutes'

>>> minutes_to_hour_string(143.25)
'2 hours and 23.25 minutes'

>>> minutes_to_hour_string(120.0)
'2 hours'
```

- `display_welcome()`: takes nothing as input and returns nothing (void). Prints out a welcome message to the user. You can decide what the message should be; it doesn't have to match the example below. Note: Only one example is needed (instead of three) for this function's docstring.

```
>>> display_welcome()
****************************************
** Welcome to the Road Trip Calculator! **
****************************************
```

- `fuel_consumption(distance_in_miles)`: takes a non-negative float corresponding to a distance in miles and returns a float indicating how many gallons of fuel will be expended by travelling that distance. If the float is 0, then the function should return 0.0.

```
>>> fuel_consumption(60.0)
2.0
>>> fuel_consumption(0.0)
0.0
```

- `get_trip_fuel(distance_from_start_to_end, initial_fuel)`: takes two floats as input and returns a float. The first input represents the distance from the start point to the end point in miles. The second input indicates the amount of initial fuel in gallons of the vehicle upon starting the trip. The function should return the number of extra gallons of fuel required to successfully reach the end point of the trip, given the initial fuel level. (Note that it will be helpful to use here the number of miles that can be travelled by one gallon of fuel (`MILES_PER_GALLON`).) If the initial amount of fuel is enough to complete the trip, i.e., no extra gallons are needed, then the function should return 0.0.

```
>>> get_trip_fuel(600.0, 15.0)
5.0
>>> get_trip_fuel(5.0, 9001.0)
0.0
```

- `get_trip_time_with_refill(required_gallons, num_gallons_refilled_per_minute, distance_from_start_to_end)`: Takes three floats and returns a float indicating the number of minutes needed to get from the start point to the end point of the trip, taking into account both the travel time and time needed to refuel at a fuel station along the route. The first input of the function represents the number of gallons of fuel that will be needed to refill to complete the trip (assuming that the vehicle will stop at a fuel station somewhere along the route) after already taking into account any initial fuel in the vehicle. The second input represents the number of gallons of fuel that can be refilled per minute at the fuel station. The last input represents the total distance of the trip in miles.

```
>>> get_trip_time_with_refill(1000.0, 1.0, 100.0)
1100.0
>>> get_trip_time_with_refill(1.0, 0.01, 1.0)
101.0
```

- `is_trip_with_refills_possible(initial_fuel, max_fuel, trip_cost, budget, distance_from_start_to_station1, distance_from_start_to_station2, distance_from_start_to_end)`:
  Takes seven floats as input, described below:
  - `initial_fuel` – the amount of fuel in gallons in the vehicle at the beginning of the trip.
  - `max_fuel` – the maximum capacity of fuel in gallons that the vehicle can carry
  - `trip_cost` – the total cost of the trip
  - `budget` – the budget available for the trip
  - `distance_from_start_to_station1` – the distance in miles from the start point of the trip to the first fuel station on the way to the end point
  - `distance_from_start_to_station2` – the distance in miles from the start point of the trip to the second fuel station on the way to the end point
  - `distance_from_start_to_end` – the distance in miles from the start point to the end point of the trip

  The function returns a boolean value indicating if the trip is possible. Specifically, it returns `False` if *any* of the following conditions are met:
  - if the available budget for the trip is smaller than the trip's total cost;
  - if the initial amount of fuel in the vehicle is not enough to reach the first fuel station;
  - if, after refilling at the first station to the maximum fuel capacity of the vehicle, the vehicle will still not have enough fuel to reach the second fuel station; or,
  - if, after refilling at the second fuel station to the maximum fuel capacity of the vehicle, the vehicle will still not have enough fuel to reach the end point of the trip.

  If none of the above conditions are met, then the trip is possible and the function should return `True`.

```
>>> is_trip_with_refills_possible(5.0, 50.0, 1000.0, 100.0, 1.0, 2.0, 3.0)
False

>>> is_trip_with_refills_possible(1.0, 1000.0, 1.0, 1.0, 1.0, 2.0, 3.0)
True
```

- `get_cheapest_trip_cost(initial_fuel, max_fuel, required_gallons, distance_from_start_to_end, cost_per_gallon1, cost_per_gallon2, distance_to_station1, distance_to_station2)`:
  Takes eight floats as input and returns a float. The function computes the cheapest cost to travel from the start point to the end point, given two fuel stations along the route which each have different fuel costs per gallon. That is, the vehicle has the option of stopping at either or both fuel stations along the way, and has to calculate which of the following options results in the minimum cost, and return that cost:
  1. refuel at neither station (if no extra fuel is needed);
  2. refuel only at the first fuel station;
  3. refuel only at the second fuel station; or,
  4. refuel a certain amount at each fuel station.

  Here are the eight inputs given to the function:
  - `initial_fuel` – the amount of fuel in gallons in the vehicle at the beginning of the trip.
  - `max_fuel` – the maximum capacity of fuel in gallons that the vehicle can carry.

- `required_gallons` – the number of gallons of fuel required to refuel for the trip.
- `distance_from_start_to_end` – the distance in miles from the start point to the end point of the trip
- `cost_per_gallon1` – the cost of fuel per gallon at the first fuel station
- `cost_per_gallon2` – the cost of fuel per gallon at the second fuel station
- `distance_from_start_to_station1` – the distance in miles from the start point of the trip to the first fuel station on the way to the end point
- `distance_from_start_to_station2` – the distance in miles from the start point of the trip to the second fuel station on the way to the end point

For instance, if there are no gallons that need to be refueled for the trip, then the cost will be 0 as there is no need to stop to refuel. Or, if 10 gallons are needed to be refueled, and the first fuel station has a cheaper cost per gallon than the second, then all 10 could be refueled there, and it would not be necessary to stop at the second station. However, assume for example that the maximum fuel capacity of the vehicle is 3 gallons, and the start point, first fuel station, second fuel station and end point are all separated by 3 miles (that is, 0, 3, 6, and 9 miles, respectively, from the start point). Then, although the first fuel station is cheaper, we need to refuel at both stations because of the low maximum capacity of the vehicle. A similar situation can occur when the second fuel station is cheaper than the first.

```
>>> get_cheapest_trip_cost(2.0, 20.0, 1.0, 90.0, 0.0, 1.0, 30.0, 60.0)
0.0

>>> get_cheapest_trip_cost(2.0, 20.0, 1.0, 90.0, 2.0, 1.0, 30.0, 60.0)
1.0
```

- `main()`: takes nothing as input and returns nothing (void). Prints out a welcome message to the user, then asks them the following questions, converting to float where appropriate:
  - what type of vehicle they will be using for the trip
  - the color of their vehicle
  - the kind of fuel used by their vehicle (gas, electricity, solar, etc.)
  - the distance from the start to end point for their trip, in kilometres
  - their vehicle's initial fuel level on starting the trip
  - their vehicle's maximum fuel level (capacity)
  - their budget for the trip
  - the cost per gallon of fuel at the first fuel station
  - the cost per gallon of fuel at the second fuel station
  - the distance from the start point to the first fuel station, in kilometres
  - the distance from the start point to the second fuel station, in kilometres
  - the number of gallons of fuel refilled per minute at a fuel station

  The function should then convert the distance values from kilometres into miles, and then calculate the amount of fuel, time and money needed for the trip given the entered values. It should then check if the trip is possible given these calculations. If so, it should print `'The trip is possible'`, and also print the amount of time the trip will take (in the format `'Y hours and X minutes'`), and the cost of the trip rounded to two decimal places. If the trip is not possible, then it should print `'The trip is not possible'`.

```
>>> main()
****************************************
** Welcome to the Road Trip Calculator! **
****************************************
What type of vehicle will you be using? spaceship
What color will it be?
1 red
2 green
3 blue
4 vantablack
Your answer: 4
What kind of fuel will your spaceship use? dilithium
Enter distance from starting point to end point (in kilometres): 100
Enter initial fuel level: 1
Enter max fuel level: 10000
Enter your budget: 10000
Enter cost of fuel at station 1: 5
Enter cost of fuel at station 2: 2
Enter distance from starting point to station 1 (in kilometres): 20
Enter distance from starting point to station 2 (in kilometres): 40
Enter amount of fuel refilled per minute: 2
The trip is possible. It will take 1 hour and 2.68 minutes and will cost $2.14.

>>> main()
****************************************
** Welcome to the Road Trip Calculator! **
****************************************
What type of vehicle will you be using? car
What color will it be?
1 red
2 green
3 blue
4 vantablack
Your answer: 3
What kind of fuel will your car use? electric
Enter distance from starting point to end point (in kilometres): 100
Enter initial fuel level: 1
Enter max fuel level: 100
Enter your budget: 1000
Enter cost of fuel at station 1: 2
Enter cost of fuel at station 2: 3
Enter distance from starting point to station 1 (in kilometres): 50
Enter distance from starting point to station 2 (in kilometres): 55
Enter amount of fuel refilled per minute: 3
The trip is not possible.
```

What To Submit

You must submit all your files on codePost (https://codepost.io/). The files you should submit are listed below. Any deviation from these requirements may lead to lost marks.

`artwork.py`
`road_trip.py`
`README.txt` In this file, you can tell the TA about any issues you ran into while doing this assignment. If you point out an error that you know occurs in your program, it may lead the TA to give you more partial credit.
Remember that this assignment like all others is an `individual` assignment and must represent the entirety of your own work. You are permitted to verbally discuss it with your peers, as long as no written notes are taken. If you do discuss it with anyone, please make note of those people in this `README.txt` file. If you didn't talk to anybody nor have anything you want to tell the TA, just say "nothing to report" in the file.

You may make as many submissions as you like, but we will only grade your final submission (all prior ones are automatically deleted).

Note: If you are having trouble, make sure the names of your files are exactly as written above.

Assignment debriefing

In the week following the due date for this assignment, you will be asked to meet with a TA for a 10-15 minute meeting. In this meeting, the TA will grade your submission and discuss with you what you should improve for future assignments.

Only your code will determine your grade. You will not be able to provide any clarifications or extra information in order to improve your grade. However, you will have the opportunity to ask for clarifications regarding your grade.

You may also be asked during the meeting to explain portions of your code. Answers to these questions will again not be used to determine your grade, but inability to explain your code may be used as evidence to support a charge of plagiarism later in the term.

Details on how to schedule a meeting with the TA will be shared with you close to the due date of the assignment.

If you do not attend the meeting, you will not receive a grade for your assignment.