COMP 202

Fall 2022

Assignment 1

Due: Thursday, October 6th, 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.

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.
- 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. If the line is getting way too long, then it's best to split it up into two different statements.
- 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 docstrings, and/or poor coding style as discussed in our lectures.**

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 a TA during office hours if you are having difficulties with programming. Go to the instructor's office hours if you need extra help with understanding a part of the course material.

  – 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. 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 in order 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

None yet!

Part 1 (0 points): Warm-up

*Do **NOT** 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:

You can try these exercises on paper, and then confirm if your answers are correct by running them in the shell.

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:

- Create and use variables.
- Learn how to build expressions containing different type of operators.
- Get familiar with string concatenation.
- Use `print` to display information.
- Understand the difference between user input and input to a function.
- Use and manipulate inputs received by the program from the function `input`.
- Use simple conditional statements.
- Define and use simple functions.
- Understand the difference between void functions and functions that return a value.
- 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 our lectures up to and including Lecture 9 (`while` loops). For this reason, you are NOT allowed to use anything seen after that lecture 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

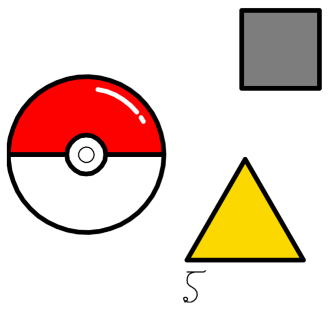
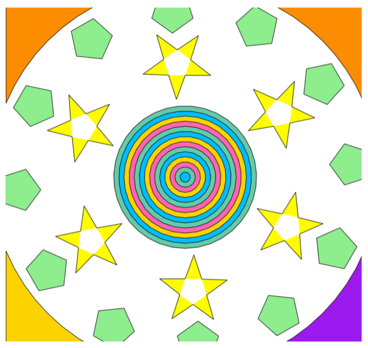
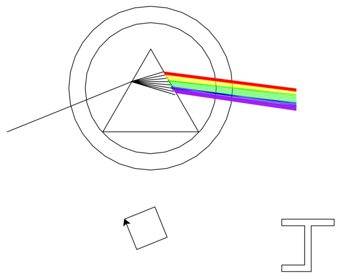
For each question, we provide several **examples** of how your code should behave. All examples are given as if you were to call the functions from the shell.

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 to it during testing. The same goes for user input. 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 will then call that function inside your `my_artwork()` function with appropriate arguments.
  - You must write your own function that is dissimilar to the ones seen in class. Do not copy one of the functions we have written or discussed in our lectures (e.g., `square`, `polygon`, `circle`, etc.) You can include those in your code and use them, but they will not count for this requirement.
- the drawing must include at least two different colors (note: since white is the default background color, it does not count as one of the two colors)
- everything must fit into the Turtle window; do not draw outside of the window
- the first letter of your first name must appear somewhere (you must sign your artwork!) (Note: The letter does not count as a shape.)
- there can be **no** calls to the `input` function
- **Do not call any functions (including `turtle.Turtle`) in the main body. This rule is for all assignments and assignment questions, not just this one. Please pay it careful attention as you will lose many marks otherwise.**

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 the `speed` method from Turtle can speed up the drawing (e.g., `speed("fastest")`).

Note 2: The docstrings for the functions you write in this question do not need examples, but must still have the type contract and description.

Also, a reminder to only use the methods from the Turtle module that we have seen in class, or a penalty will be applied. There is one exception: you can use the `circle` method. `t.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., `t.circle(r, 90)`, which will draw only a quarter of a circle (90 degrees).

Some submissions from students of previous years (scaled down) are shown above.

Question 2: Pizza Shop (85 points)

In this question we will do some calculations for a pizza shop: figuring out the cost of certain pizzas and pizza-related items, and letting the user interact with our program through a menu of different program options.

There will be a few different calculations, and each will be placed in its own function. Whenever you are faced with a complex problem, which will most often be the case when programming, it is best to break it down into smaller sub-parts. Each different computation should be put 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 **`pizza.py`**.

First, define the following global variables at the top of your file. You will use these variables in certain functions of your code.

- `PIZZA_CAKE_COST_PER_CENTIMETRE_CUBED = 4.0`
- `PIZZA_POUTINE_COST_PER_CENTIMETRE_CUBED = 3.0`
- `SPECIAL_INGREDIENT = "guacamole"`
- `SPECIAL_INGREDIENT_COST = 19.99`
- `FAIR = True`

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' for short. 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, and we never want to update it). 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 changing its value in the code directly.

The examples shown for the functions below will assume that the constants are set to the above values. However, during testing, we will 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 when appropriate and that you do not instead 'hard-code' the values (meaning writing the values directly into the code instead of using the global variables).

You must write the following functions for this assignment:

- `get_pizza_area(diameter)`: Takes a positive float for the diameter of a pizza, and returns its area as a float. The float should **not** be rounded. We round it only in the examples below by calling the `round` function.

```
>>> round(get_pizza_area(1.5), 2)
1.77
```

```
>>> round(get_pizza_area(2.0), 5)
3.14159
```

- `get_fair_quantity(diameter1, diameter2)`: Takes two positive floats, for the diameter of two pizzas. Returns as an integer the minimum number of small pizzas that must be ordered to get at least the same amount of pizza (by area) as one large pizza. (Note that either input to the function may be the smaller or larger one.)

  If the `FAIR` variable is set to `False`, then return 1.5 times the true amount (rounded down).

  In the first example below, given a pizza of diameter 14.0, it will take 4 pizzas of diameter 8.0 to get at least the same amount of pizza by area. We can calculate this by finding the area of each pizza size ($\pi(\frac{14}{2})^2$, $\pi(\frac{8}{2})^2$) and checking how many multiples of the smaller quantity are needed to get equal to or more than the bigger quantity.

```
>>> get_fair_quantity(14.0, 8.0)
4
>>> get_fair_quantity(3.0, 14.0)
22
```

- `pizza_formula(d_large, d_small, c_large, c_small, n_small)`: Takes four positive floats (diameter of large and small pizza, and cost of large and small pizza) and one positive integer (number of small pizzas) as input. Exactly one of the five inputs will be the integer `-1`, which we call the 'missing' input value. The function must use the other four values to calculate and return the missing input value as a float, rounded to two decimal places. For example, if the first input (diameter of large pizza) is `-1`, then your function must use the other four inputs to calculate and return the diameter of the large pizza. (Assume that the number of large pizzas is 1.)

  Your formula should be based on the following relationship: the amount of small pizza (by area) per dollar should be equal to the amount of large pizza (by area) per dollar.

  Note: For this function, there will be some repetitive code. Try to reduce the amount of repetition where you can.

```
>>> pizza_formula(12.0, 6.0, 10.0, -1, 2)
5.0
>>> pizza_formula(14.0, 8.0, 9.55, -1, 4)
12.47
```

- `get_pizza_cake_cost(base_diameter, height_per_level)`: Takes one positive integer (diameter of the cake base) and one positive float (height in centimetres per level) as inputs. Returns the cost of the pizza cake as a float rounded to two decimal places.

  A pizza cake is a stack of pizzas all on top of each other, with the largest at the bottom (with the given integer base diameter). Each successive pizza on top will be 1cm smaller in diameter than the one below. The pizza on the very top will have diameter 1cm. The total cost of the cake will be the sum of all individual pizzas. A pizza cost can be calculated by multiplying its area by the given height and by the global variable that gives the cost of pizza cake by centimetre cubed.

  If the `FAIR` variable is set to `False`, then return 1.5 times the true amount (rounded to two decimal places).

  Note: You may need to use a loop for this function.

```
>>> get_pizza_cake_cost(2, 1.0)
15.71
>>> get_pizza_cake_cost(10, 2.0)
2419.03
```

- `get_pizza_poutine_cost(diameter, height)`: Takes one positive integer (diameter of the poutine cylindrical cup) and one positive float (height in centimetres) as inputs. Returns the cost of the pizza poutine as a float rounded to two decimal places.

  A pizza poutine is a cylinder containing pizza, fries and gravy. The total cost of the poutine can be calculated by multiplying its volume by the global variable indicating the cost of a pizza poutine per centimetre cubed. Recall that the volume of a cylinder is $\pi * r^2 * h$, where r is the cylinder's radius and h is its height.

  If the `FAIR` variable is set to `False`, then return 1.5 times the true amount (rounded to two decimal places).

```
>>> get_pizza_poutine_cost(2, 1.0)
9.42
>>> get_pizza_poutine_cost(10, 2.0)
471.24
```

- `display_welcome_menu()`: Takes no inputs and returns nothing. Displays to the screen a menu with a welcome message and the three options available to the user. You can choose your own welcome message, but your menu must include the three options in the given order (and with A. B. and C. as the options). You can modify the option titles if you like, though they should still have the same meaning as the original (e.g., you could write 'Choose a special order' instead of 'Special Orders').

  Note: You only need one example for the docstring for this function.

```
>>> display_welcome_menu()
Welcome To The Best Pizza Place. Our Pizzas Made With 100% Real Pizza.
Please choose an option:
A. Special Orders
B. Formula Mode
C. Quantity Mode
```

- `special_orders()`: Takes no inputs and returns nothing. Asks the user to choose between cake and poutine, asks them to enter a value for the diameter and for the height, and if they want the special ingredient, then prints out the total cost of the order. The user should be able to type in either `'y'` or `'yes'` to indicate that they want the special ingredient, in which case the cost of the special ingredient is added to the cost of the order. If they type in anything else, then the cost of the special ingredient is not added.

  Note: Make sure to use the two global variables for the special ingredient, as discussed earlier in these instructions. Do not hardcode the name or value of the special ingredient directly.

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

```
>>> special_orders()
Would you like the cake or poutine? cake
Enter diameter: 2
Enter height: 1.0
Do you want the guacamole? yes
The cost is $35.7
>>> special_orders()
Would you like the cake or poutine? poutine
Enter diameter: 2
Enter height: 1.0
Do you want the guacamole? no
The cost is $9.42
```

- `quantity_mode()`: Takes no inputs and returns nothing. Asks the user to enter in the diameters of two pizzas, and then prints out the minimum number of smaller pizzas they would need to buy to get at least the same amount of pizza (by area) as one large pizza.

```
>>> quantity_mode()
Enter diameter 1: 14.0
Enter diameter 2: 8.0
You should buy 4 pizzas.
```

- `formula_mode()`: Takes no inputs and returns nothing. Asks the user to enter in the diameter of a large and small pizza, cost of a large and small pizza, and number of small pizzas (in that order), with one of the values being `-1`, and prints out the actual value of the input that was given as `-1`.

```
>>> formula_mode()
Enter large diameter: 12.0
Enter small diameter: 6.0
Enter large price: 10.0
Enter small price: -1
Enter small number: 2
The missing value is: 5.0
```

- `run_pizza_calculator()`: Takes no inputs and returns nothing. Displays a welcome message to the user and list of program options, asks the user to input an option, and then calls the appropriate function. If the user inputs an invalid option, prints out `"Invalid mode."`.

```
>>> run_pizza_calculator()
Welcome To The Best Pizza Place. Our Pizzas Made With 100% Real Pizza.
Please choose an option:
A. Special Orders
B. Formula Mode
C. Quantity Mode
Your choice: A
Would you like the cake or poutine? cake
Enter diameter: 2
Enter height: 1.0
Do you want the guacamole? yes
The cost is $35.7

>>> run_pizza_calculator()
welcome to pizzas R us. no credit cards accepted, cash only. come to back alley 4 pickup.
choose option:
A. Special Orders
B. Formula Mode
C. Quantity Mode
Your choice: B
Enter large diameter: 12.0
Enter small diameter: 6.0
Enter large price: 10.0
Enter small price: -1
Enter small number: 2
The missing value is: 5.0
```

```
>>> run_pizza_calculator()
Welcome to Flying Pizzas. If you can catch the pizza as it flies out of the oven,
you get your pizza free! Note: Must bring your own protective gear.
Please choose an option:
A. Special Orders
B. Formula Mode
C. Quantity Mode
Your choice: D
Invalid mode.
```

In the example below, the welcome message, menu options and other text are customized (in reference to a popular video game). Note however that the menu options are still labelled A., B., and C., and still have the same meaning as the original text. Note also that the inputs are asked for in the same order as described in the functions above, and the values (in this case, the final cost) are still displayed as required.

(In addition, `FAIR` is set to **`False`** for this example.)

```
>>> run_pizza_calculator()
HEY      EVERY      !! EV3RY  BUDDY  'S FAVORITE [Pizza Shop]
HURRY UP AND BUY! I JUST NEED YOUR [Account Details] AND THE [Number on theB4ck]!
YUM YUM GREAT OPTIONS
A. [Specil Deals]
B. [Prize Winning] F0RMULA
C. HOW MUCH TO BUY?? [You Can Never Buy Enough]!!
TAKE AN OPTION: A
DO YOU WANT THE [[Mouth-watering cake]] or [[Terrifying]] POUTINE? cake
WOAH!! DIAMETER?? 2
AND HEIGHT??? 1.0
DO YOU WANT THE PIZZA INSURANCE? no
DON'T WORRY KIDS I'M AN [HonestMan]! JUST DEPOSIT 23.56 KROMER
```

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`
`pizza.py`
`README.txt` In this file, you can tell the TA about any issues you ran into while doing this assignment.
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(s) following the due date for this assignment, you will be asked to meet with a TA for a 10-20 minute meeting (exact duration TBD). 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 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 in the days following the due date of the assignment.

If you do not attend the meeting, you will receive a 0 for your assignment.