

# Lab 5

*Learn about the different types of loops and how they work in different situations.*

*In this lab, you will practice with the three different types of loops: for loops, while loops, and do while loops. You will also learn keywords such as pretest and posttest, and how to pick the best loop(s) for different circumstances.*

## In-Lab Activities

Don't forget to answer the **Guided Inquiry Questions** in your **notebooks**. It is also encouraged to use it frequently while working through activities.


### Warm-Ups

Go to the Google Forms link on the board and complete your Warm-Up Questions. You have 10 minutes to complete this. If you finish early, feel free to begin the Lab Activities, but make sure your partner is ready as well.

### Activity 1: Grader

Write a program to help a teacher to calculate the average score of students in the class. This program should do the following things:

1. Allow the teacher to enter the number of students in the class. (What kind of **message** should you include here?)
2. Based on that number of students, run a **loop** to ask for each student's test score. (What kind of **loop** do you think would be most helpful here? You already have a fixed number of times to **iterate** based on the teacher's input.)
3. After the loop ends, you will need to display the class's **average grade** back to the teacher. However, once the loop ends, you may have already lost the data (each student's individual score) – what should you add *inside* the loop so that you will be able to calculate the average at the end? Think about using a **running total**.



In this **for loop**, what is the **control variable**? What is the **loop continuation condition**?




While going through these activities, pay attention to the different types of loops and in what situations you are using them.

### Activity 2: Using a while loop

Write a program similar to that of Activity 1, however, done in a different way:

1. Instead of having the teacher enter the number of students, allow them to keep entering grades until they enter -1. Once a -1 is entered, the loop will **exit**.
2. Once out of the loop, **calculate** and **display** the **average** of all the grades.
3. The **number of students** should also be displayed at the end. (What should you add to the loop to keep track of this?)



What is the **loop continuation condition** in this program? What is the **sentinel value**?  
Draw a **flow chart** of this program.



Try using this with both a **pretest** and a **posttest** loop. Write your observations of each in your **notebook**. Which was easier to use in this situation? Why?



## CHECKPOINT

35 MINUTES



### Brainstorming Individually

(3-5 minutes)

Read the next Activity and start **brainstorming**; think about ways you could start solving this problem and write down any notes. Do this *individually* for 3 to 5 minutes and then come together afterwards to compare, discuss, and code. If you can, try drawing the **flowchart** together as well. 😊

## Activity 3: Validating criteria



You will oftentimes need to **validate** user-input in programs - only allowing the user to enter specific types of input and/or forcing them to reenter if this input is incorrect. For example, let's say you have a menu with choices A, B, and C, but the user decides to enter W. Using loops, you may now tell the user that this is incorrect and force them to make another choice (hopefully a correct one this time).

To practice with this, write a program that asks the user to enter a positive integer. Now here are some rules for the **validation** part:

1. If that number is negative, re ask the same question and allow the user to enter a new number. (How can you use a **loop** to achieve this?)
2. Now add one more criteria: the number must also not be divisible by 10. Once again, if the input does not follow this rule, repeat the original question and allow the user to enter a new number.
3. Once a correct number has been inputted by the user, have the program display their number along with the message "Wise choice."



## CHECKPOINT

25 MINUTES



## Activity 4: The Talking Robot

So now you know how to validate criteria, run a loop, and generate a random number. In this activity, you will make a chatbot that talks to the user – if you would like a more complex example of this, search for Cleverbot and try having a short conversation with it (but don't spend too much time on it!).

In this chatbot program, allow the user to enter a string as a form of dialogue. Next, create some preset statements (at least five) that will be randomly selected as the response to the user's dialogue. Run the program to see how the bot behaves.

What are your opinions about this bot? Is it very believable during conversation? Probably not, but don't worry! We will continue working with this in later Labs, and will eventually create more chatbots with more complexity and realism.



## CHECKPOINT

30 MINUTES



## Challenge (yet fun, hopefully) Activity: Gotta catch 'em all

**Pair Programming:** Since this is a very challenging program and will take time, switch roles every 20-30 minutes.



You will now make a simple Pokémon game, similar to the one your TA showed you at the beginning of Lab. There are many parts that you must include in this game. The steps and requirements are shown below, but don't get overwhelmed! Take it one step at a time and cross sections off as you complete them.

1. Before starting anything, allow the player to pick his or her starter Pokémon from a given list. Your Pokémon name should be saved in a string for use throughout the game. (What kind of **message** and **input** should you include to achieve this?)
2. Once complete, the player is shown a menu in which they can choose to (1) Search for Pokémon, (2) Use a potion, or (3) Quit the game. You must use a loop, so that once they choose and complete one menu option, they are taken back to this menu once again. The player can keep choosing menu items until they decide to quit (or if their Pokémon faints).

Make a list in your **notebook** of all the variables you believe you will need to **declare**. Which of these will need to be **instantiated**?  
Continue to add on to this list as you read through the steps and requirements below.

Here are some brief descriptions of the menu items:

1. **Search for Pokémon:** Have a random number generator to determine whether or not the player's search was successful. Generate a number from 1 to 10. If the number falls below 6, the search was successful. Otherwise, the player failed to find a Pokémon. Display message accordingly.
2. **Use a potion:** The player has a total of 10 potions. Each potions heals the starter Pokémon by 20 HP, and decrements the player's total amount of potions remaining by one.
3. **Quit:** Display a goodbye message and end the game.

Whenever a player successfully finds a Pokémon, there are a few things you must **instantiate** for that new Pokémon:

1. Have their starting HP be a randomly generated amount from 50 to 100.
2. Have the Pokémon type itself be random, and there must be at least three different ones (for example: "Caterpie", "Pichu", and "Munchlax"). This name should be used in the fight dialogue.

The player will also be given another menu with new options when a Pokémon is found. They may either (1) Fight the enemy Pokémon, (2) Use a Pokeball, (3) Use a Potion, or (4) Flee. These options are now described below:

1. **Fight the enemy Pokémon:** Each Pokémon will take turns to do damage to the other one. Have the damage amount be a randomly generated number (you may choose a range that seems reasonable to you), and subtract that amount from the receiving Pokémon's HP.
2. **Use a Pokeball:** The player throws a Pokeball at the enemy Pokémon. They either succeed or fail to catch it. The success rate is determined by the following algorithm:

*double successRate = enemyTotalHP / enemyCurrentHP;  
If a random number from 1 to 10 is less than the successRate, the player caught the Pokémon. Otherwise, they failed to catch it.*

What is a **nested** statement? A **nested loop**?  
How can you apply this knowledge in your program?

3. **Use a Potion:** Like before, each potion heals the main Pokémon by 20 HP, and each use decrements the potion counter by one.
4. **Flee:** The player flees the battle and returns to the main menu.

For every new **block**, write a comment above it to describe its purpose.

There are a few extra things to note:

1. If the enemy Pokémon faints, you can no longer fight it and the player returns to the main menu.
2. If the player's Pokémon faints, the game ends and you can no longer search for Pokémon (or use a potion).
3. The player's Pokémon starts at 100 HP, and cannot use potions to go over this starting amount.

**If you need to see the example program again, ask your TA. They may show you the running program, however, they cannot just give you the code. If you do need additional assistance, do not hesitate to ask!**

## Reflection

Go to the Google Forms link given by your TA to work on your Reflection for Lab 5. This counts as a part of your participation grade, and it is important that you put as much detail as possible.