**Chapter 2 – If statements and loops**

Now that we have a basic understanding of how to represent data on a computer and how to communicate with our interface we need to start working on the logic of the game.

To new programmers, it may be difficult to understand what is meant by the logic behind a program. Using combinations of if statements and loops, you can make the program carry out different actions based on the information it is given. For example, our game will have at least two classes and we need to make sure that different classes will have different traits or skill moves. You will be using a lot of these throughout the program so it important to understand these constructs well.

If statements

If statements are used to execute(run) a portion of code based on whether a condition is met. When the condition is false none of the statements in that if statement will be executed. Here is the basic syntax of an if statement:

if (condition):

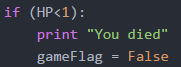
statement 1

statement 2

Since we are writing in python, it is important that indentation is done correctly so that what you want is in the if statement and not outside it. Indentation is done by pressing the *TAB* button and every statement we want inside our if statement must have 1 more indentation than the start of the if statement. It is also important to note that you can put many statements in an if statement and have nested if statements. This is when you place an if statement within an if statement.

Example 2.1

Going back to the health variable that we used in **chapter 1** we learnt how to add and subtract values from our health but didn’t think about what happens when our health reaches zero. If we have 30 health and take 40 damage, our new health would be -10 but nothing really happens after that. This is where we create an if statement to tell the player when their health reaches zero or below that they have died and can no longer continue in the game. Here is an example implementation from our game:



Here we have the condition (HP<1) which means the indented code will only execute if our health reach zero or below. If it does, we use the print function we learnt about in **chapter 1** to tell the player they have died. We also have a “gameFlag” variable which will be useful later when we want to check if a new game session needs to be started.

The if statement can be expanded upon further through the use of an if else statement. Essentially what this is does is similar to a regular if statement but adds an extra part to the end. If the condition of the if statement is not true (i.e. false) the block of code inside the else will be executed. The syntax looks like the following:

if (condition):

statement 1

statement 2

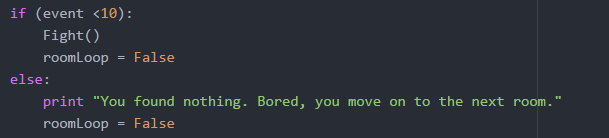
else:

statement 3

Once again it is important to be careful with indentation for the if statement to work correctly. In the above code block, if the condition is true then statements 1 and 2 will be executed. If it is false then statement 3 will be executed.

Example 2.2

A way this is used in our game would be for events that can happen when in a room. We want to have different outcomes based on what our random number generator produces:



In this example, the value of “event” is randomly generated between 1 and 100. If the number is less than 10, a fight is initiated which we will create later. If the value is not less than 10, we just simply have a print statement that says nothing interesting has happened. We also have a variable called “roomLoop” which is used as a condition for a loop which is what we will explain in the next topic of this chapter.

Loops

Loops in programming simply do what the name suggests. They repeat a set of instructions on a loop while a condition is met. There are a few different types of loops but the most common types a ‘for’ and ‘while’ loops.

For loops have many different ways of being expressed but they all work on the principle of iterating through a condition. This simply means cycling through a set of data whether it is simply a range of numbers of if it is iterating through elements in a list (lists will be explained in more depth later). For now we will simply show you how to use a for loop on numbers:

For i in range(0,9)

Statement 1

Statement 2

The above example will simply execute statements 1 and 2 repeatedly 10 times. Another thing with for loops is we can use the variable declared in them. For example, every time the loop executes the value stored in ‘i’ will change. In the first loop ‘i’ will be equal to 0, then 1, then 2 etc until it reaches 9. This will be very useful when we use for loops with lists.

The other main type of loop is the ‘while’ loop which continuously repeats while a condition is met. If the condition is not immediately met none of the statements within the loop will be executed. They also follow a simple structure which is similar to an if statement:

While(condition):

Statement 1

Statement 2

Example 2.3

In this example we will look at how a while loop is used but there will definitely be uses of for loops later on as we develop the game further. Going back to the ‘roomLoop’ variable in the previous example. In programming this is commonly referred to as a Boolean which either takes a true or false value. This makes it particularly useful in conditions as we can check the value of the variable and it will either be true and satisfy the condition or be false and not satisfy the condition.

The reason why we have a loop here is because we want to make sure that while the player is in the same room, we want to ask the player what they want to do (which makes sense why it is called a ‘while loop’). Once the user completes an action in the room or there is nothing else to do, the ‘roomLoop’ value will be set to false. This allows our program to know that the user can move onto the next room in our dungeon. Since the user has entered a new room, we will then set the value of ‘roomLoop’ to true once again.

Another widespread use of the while loop in a game like this is to ensure valid inputs are given to the player. If we are expected the player to give a command but they give a command that’s is not recognised by our game, we can keep waiting until an input that the game does recognise is given. This minimises the chances of our game crashing during a playthrough.

MINI EXERCISE

1. Write a small program that takes two numbers. Using a loop of your choice, print every number between those two numbers.
2. Write a small program that takes in a number. The program must print a triangle which has the height of the input number e.g input = 3 output=

\*

\*\*

\*\*\*

1. Write a shopping list calculator. The user can enter a price. If they wish to end the list they must type ‘stop’. Once the list has been complete, you should print the total.