Lesson 5: Dragon Battle

In this lesson, you will use loops to create a battle against a fierce dragon. You will have multiple turns to attack. Each turn, you must choose between two different attacks: a light activity or heavy attack. If you miss the dragon will hit you.

Part 1 – Choosing an Attack

Create a new file like before and save it as **YourName LastInitial Lesson 5 – Dragon Battle.py** in your folder.

Let's start the program by importing the random module and creating variables for our player's health and the dragon's health. Remember anytime you want any sort of randomness in your game, you should import random.

```
1  import random
2
3  playerHealth = 100
4  dragonHealth = 100
5
6  print("The dragon has appeared, can you defeat it?")
```

Next, let's set up the while loop that controls our game. Our game will end when either the player or dragon has run of out of health so the game keeps going while both of their health is greater than 0.

```
6  print("The dragon has appeared, can you defeat it?")
7  while (playerHealth > 0 and dragonHealth > 0):
```

Next, we need to think about what will happen inside the loop. Think of the actions that we want to repeat each turn. Each turn you're going to want to print out the player's health and the dragon's health and ask the player for their choice.

*__ are blanks for you to fill in

```
7  while (playerHealth > 0 and dragonHealth > 0):
8      print("YOUR HEALTH:",_____________)
9      print("DRAGON HEALTH", _____________)
10     move = input("Type 1 for light attack or 2 for heavy attack")
```


©2020 TechKnowHow Inc.

If you don't know what to put in the blanks, here is a hint: think about what you're trying to print out, then use the corresponding variable.

Part 2 – Creating Probabilities

Let's create a variable called number and set it equal to a random number between 1-100 and then use it to determine whether or not our light attack will land. The blank here is the amount of damage your light attack will do. You can put whatever number you like.

*… represents 2 or more lines of code that you've already written

```
7   while (playerHealth > 0 and dragonHealth > 0):
…       …
10      move = input("Type 1 for light attack or 2 for heavy attack")
11      number = random.randint(1, 100) #used to set up probability
12      if (number <= 75 and move == "1"): #75% chance to land light attack
13          print("You've landed your light attack.")
14          dragonHealth -= ___
```

Setting up probabilities with number = random.randint(1, 100) is very simple. All you need to do is check that the random number is less than or equal to whatever accuracy you want. Up above, we wanted the light attack to have a 75% chance of landing, so we checked that number was less than or equal to 75.

Why does this work? Think about how many numbers exist between 1-100 (including 1 and 100). That's right, 100 numbers. Now, how many of those numbers are lower than or equal to 75? Only 75 numbers (1-75) are lower than or equal to 75. So, we have 75/100 numbers that make number < 75 true and 75/100 is 75%.

Now that we've set up the probability for our light attack, let's set up the probability for our heavy attack. Try filling in these blanks. The first is the percent chance of your heavy attack hitting. It should be lower than your light attack. The second is what the player must input to use the heavy attack. Remember, the heavy attack is used when the player types 2. The last blank is the amount of damage you want your heavy attack to do. You should do more damage with your heavy attack than your light attack.

```
7   while (playerHealth > 0 and dragonHealth > 0):
…       …
12      if (number <= 75 and move == "1"): #75% chance to land light attack
```


©2020 TechKnowHow Inc.

```
13          print("You've landed your light attack.")
14          dragonHealth -= ___
15      elif (number <= __ and move == __):
16          print("You've landed your heavy attack!")
17          dragonHealth -= __
```

Lastly, we'll add an else to our conditional which will handle the case when the player has missed their attack. The blank here is the amount of damage that the dragon does on you. Feel free to put what you'd like.

```
7   while (playerHealth > 0 and dragonHealth > 0):
…       …
15      elif (number <= __ and move == __):
16          print("You've landed your heavy attack!")
17          dragonHealth -= __
18      else:
19          print("You've missed your attack and the dragon has struck you")
20          playerHealth -= __
```

Part 3 – Checking Who Won

When the loop ends, we want to congratulate the player for defeating the dragon. Why can't we just add this print statement at the end of our code?

```
while (playerHealth > 0 and dragonHealth > 0):
      #Game code
print("You have defeated the dragon")
```

When the loop ends, it's not necessarily because the player has won. The loop ends when either the player or the dragon has run out of health, not just when the dragon has run out of health. So, we need an if statement to check if the player still has health. If they have, we'll congratulate them, otherwise we will tell them that they have lost. Make sure this code is outside the while loop.

```
7   while (playerHealth > 0 and dragonHealth > 0):
…       …
21  if (playerHealth > 0):
22      print("You have defeated the dragon")
23  else:
24      print("You have been defeated")
```


©2020 TechKnowHow Inc.

It is *very* important that you understand what circumstances cause your loop to end for 2 reasons. First, it will prevent from making incorrect assumptions at the end of the loop (like in the example we just went over). Second, it will prevent you from creating an infinite loop that can overload your computer and cause the program to crash.

Part 4 – Fine Tuning the Game

Play your game and see how easy or hard it is to win. If it's too easy, try lowering the percentages, decreasing the damage done by your attacks, and/or increasing the damage done by the dragon. If it's too hard, try increasing the percentages, increasing the damage done by your attacks, and/or decreasing the damage that the dragon does.

This sort of fine-tuning is an important part in game design. Once you've got the logic (code) of a game done, you need to actually play it and change the numbers around to find a balance of fun and difficulty.

As always, feel free to change any print statements.

BONUS: Try adding an ultimate attack to your game. Something that does an insanely high amount of damage, but also has a very low chance of hitting.

©2020 TechKnowHow Inc.