Lesson 10: Pokemon Battle

In this lesson, we will practice using classes by creating a pokemon battling game. Actual pokemon battling can actually be quite complex, but we will simplify some of the logic here. Don't' worry though, your game will have all of the basics and will still be fun to play!

PART 1: Creating the Move Class

Let's create a class for our pokemon moves. Moves actually have a lot of stats involved with them, so it is best to create a class for them.

```
class Move:
```

Moves have a name, type (e.g. water, fire, grass, etc), damage and accuracy. All these stats will differ for every attack, so let's create an initialization function that sets up all these stats for our Move as instance variables.

```
class Move:

    def __init__(self, name, moveType, damage, accuracy):
        self.name = name
        self.moveType = ________
        self.damage = ______
        self.accuracy = ________
```

Now that we have the move class, we can make a couple of moves for our game. Remember that to make an object from a class, you use the initialization function by calling the name of the class. Also, take a look above at the order of all the parameters. That is the order you must give your arguments when you make the move objects.

```
class Move:
    ...

tackle = Move("Tackle", "Normal", 10, 95)
waterGun = ___("Water Gun", "Water", 20, 75)
bubbleBeam = ___("Bubble Beam", "Water", 15, 85)
surf = ____("Surf", "Water", 30, 50)
ember = ____("Ember", "Fire", 15, 85)
```

```
flamethrower = ____("Flamethrower", "Fire", 20, 75)
slash = _____("Slash", "Normal", 15, 90)
```

Try writing this one line of code to test out your move class:

print("Move:", tackle.name, "Type:", tackle.moveType, "Damage:", tackle.damage,

"Accuracy:", tackle.accuracy)

Delete this line when you are done.

Part 2: Making the Pokemon Class

Let's make the pokemon class so we can make some pokemon objects. All pokemon in our battling game will start with 100 health, so we can declare that as a default value outside the initialization function.

```
class Pokemon:
    health = 100

tackle = Move("Tackle", "Normal", 10, 95)
```

Apart from health, pokemon have a name, type (e.g. fire, water, etc.), and a list of moves that they know. The moveList will be a list of Move objects. These are unique for each pokemon. Let's set them up in the initialization function for the class.

```
class Pokemon:
    health = 100

    def __init__(self, name, pokemonType, moveList):
        self.name = ____
        self.type = ____
        self.moveList = ____
```

For now, we will create one small function in the pokemon class that prints out our move list. To do that, we need to create a for loop that iterates through our move list and prints out the name of the move. Note that because this function is inside a class and will be called from an object, it requires us to put self as a parameter to change the object's instance variables. However, when we call this function, we will not need to add any arguments.

```
class Pokemon:
    health = 100
```

```
    def __init__(self, name, pokemonType, moveList):
        ...
    def printMoveList(self):
        for i in range(len(self.moveList)):
            print(str(i + 1) + ") " + self.moveList[i].name)
```

Previously, we created a lot of Move objects that will be the moves in our game. Let's organize them into lists for each pokemon.

```
slash = Move("Slash", "Normal", 15, 90)

squirtleMoves = [tackle, waterGun, bubbleBeam, surf]
charmanderMoves = [tackle, ember, flamethrower, slash]
```

We can finally make some pokemon! Make one for the player and another for the computer.

```
squirtleMoves = [tackle, waterGun, bubbleBeam, surf]
charmanderMoves = [tackle, ember, flamethrower, slash]
playerPokemon = Pokemon("Squirtle", "Water", ____________)
cpuPokemon = Pokemon("Charmander", "Fire", _______________)
```

Let's tell the player what pokemon they have and what pokemon the computer has. We should also tell them their pokemon's moves.

```
cpuPokemon = Pokemon("Charmander", "Fire", charmanderMoves)

print("Welcome to Pokemon Rumble")
print("The computer chooses", cpuPokemon.name)
print("Your Pokemon is", playerPokemon.name)
playersPokemon.printMoveList() # delete this line when you are done
```

PART 3: Making the Attack Function

Attacks will randomly hit or miss. This means we need to import random at the top of our code so we can have randomness in our game.

```
import random

class Move:
```

Let's make an attack function in our pokemon class. This way all pokemon objects will be able to attack with a move from their move list. It will need three parameters: self (because it will be called from an object), the index of the move, and the pokemon that it is going to hit. First, let's get the move that the pokemon wants to use from our list. Remember that the move we get from the list is a move object. This means we can access all its attributes by using the dot syntax.

```
class Pokemon:
    ...

    def attack(self, moveIndex, targetPokemon):
        move = self.moveList[moveIndex]
```

Next, let's set up our hit probability by creating a random number between 1-100. All we need to do then is check if this number is less than or equal to our move's accuracy stat. For example, say our move has a 75% accuracy. Our randNum can be any number between 1-100, and because there are 75 numbers between 1-75 that is less than or equal to our accuracy, that creates our 75 percent chance.

```
    def attack(self, moveIndex, targetPokemon):
        move = self.moveList[moveIndex]
        randomNumber = random.randint(1, 100)
        if (randomNumber <= move.accuracy):
            targetPokemon.health -= move._____
```

We also need to make sure that the target pokemon's health does not go below zero. After we take away their health, we can check if it's below 0. If it is, then we set it to 0 instead.

```
    def attack(self, moveIndex, targetPokemon):
        move = self.moveList[moveIndex]
        randomNumber = random.randint(1, 100)
        if (randomNumber <= move.accuracy):
            targetPokemon.health -= move._____
            if targetPokemon.health < 0:
                targetPokemon._____ = 0
```

Last, we want this function to tell us whether or not the attack landed. The function will return a Boolean value: True if the attack landed and False if the attack missed. Pay attention to your indentation here. We return true inside the outer if since that is the one that checks if the attack hits. We return false outside the outer conditional.

```
    def attack(self, moveIndex, targetPokemon):
        move = self.moveList[moveIndex]
        randomNumber = random.randint(1, 100)
        if (randomNumber <= move.accuracy):
            targetPokemon.health -= move.damage
            if targetPokemon.health < 0:
                targetPokemon.health = 0
            return True
        return False
```

In Pokemon, moves do not always do the same amount of damage. Moves can do more damage when its type makes it super effective against the target pokemon's type. For example, water moves are super effective against fire pokemon. It is important to note that super effectiveness is based on the move's type, not the pokemon's type. For example, Squirtle's tackle is not super effective against Charmander.

With that explanation out of the way, we should make a function in the Move class that can determine its effectiveness against a target pokemon. Effectiveness will be a multiplier that use to multiply our damage. To start, we set the effectiveness to 1 .

```
class Move:

    def __init__(self, name, moveType, damage, accuracy):
        ...

    def getEffectiveness(self, targetPokemon):
        effectiveness = 1
```

First, we use a conditional to check what type our move is. Then we use a nested conditional to check that type against the target pokemon's type. If it is super effective against the target pokemon, then effectiveness will be 2. If it is weak against the target pokemon, then effectiveness will be 0.5.

```
    def getEffectiveness(self, targetPokemon):
        effectiveness = 1
        if self.moveType == "Water":
            if targetPokemon.type == "Fire":
                effectiveness = 2
        if self.moveType == "Fire":
            if targetPokemon.type == "Water":
                effectiveness = 0.5
        return effectiveness
```

Back in the Pokemon class, we can make our attack function call the getEffectiveness function to get the move's multiplier. Then, we just need to multiply the move's damage by the multiplier.

```
    def attack(self, moveIndex, targetPokemon):
        move = self.moveList[moveIndex]
        randomNumber = random.randint(1, 100)
        if (randomNumber <= move.accuracy):
            multiplier = move.getEffectiveness(targetPokemon)
            targetPokemon.health -= move.damage * multiplier
            if targetPokemon.health < 0:
                targetPokemon.health = 0
            return True
        return False
```

Let's test it. Have Squirtle attack with water gun and have Charmander attack with flame thrower at the bottom of your program. You should see Squirtle's health go down to 90 and Charmander's go down to 60. Note that there is a possibility for the attacks to miss. Keep running the program until they hit since we also want to test out our probability. Delete these lines when you are done.

```
playerPokemon.attack(1, cpuPokemon)

cpuPokemon.attack(2, playerPokemon)

print("squirtle:", playerPokemon.health)

print("charmander", cpuPokemon.health)
```

Part 5 : Let the Battle Begin

The battle will be controlled by a while loop that will keep going while either the player pokemon or computer pokemon has health. Each iteration of the loop will be one round of the battle and at the beginning of each round, we want to tell the player how much health each pokemon has. This code goes at the bottom of your program.

```
while cpuPokemon.health > 0 and playerPokemon.health > 0:
    print("Your Health:", playerPokemon.health)
    print("Computer's Health:", cpuPokemon.health)
    print()
```

Next, we want to tell the player to pick a move by typing 1, 2, 3, or 4. We can print out the moves for them by using the print moves function. To get the move's index, we have to subtract the player's choice by 1 since lists start at 0.

```
while cpuPokemon.health > 0 and playerPokemon.health > 0:
    print("Your Health:", playerPokemon.health)
    print("Computer's Health:", cpuPokemon.health)
    print()
    print("Pick a Move by typing 1, 2, 3, or 4:")
    playerPokemon.printMoveList()
    playerMoveIndex = int(input()) - 1
    playerMove = playerPokemon.moveList[playerMoveIndex]
```

The computer will just pick a random number between 0-3 for their move.

```
while cpuPokemon.health > 0 and playerPokemon.health > 0:
    print("Your Health:", playerPokemon.health)
    print("Computer's Health:", cpuPokemon.health)
    print()
    print("Pick a Move by typing 1, 2, 3, or 4:")
    playerPokemon.printMoveList()
    playerMoveIndex = int(input()) - 1
    playerMove = playerPokemon.moveList[playerMoveIndex]
    cpuMoveIndex = random.randint(0, 3)
    cpuMove = cpuPokemon.moveList[cpuMoveIndex]
```

We will allow the computer to go first since it is at a disadvantage. The attack function takes 2 arguments: the index of the move to use and the pokemon to hit. It also returns true if the attack landed so we can store the result in a variable so that we can tell the player if the move landed.

```
while cpuPokemon.health > 0 and playerPokemon.health > 0:
    ...
    cpuMoveIndex = random.randint(0, 3)
    cpuMove = cpuPokemon.moveList[cpuMoveIndex]
    hit = cpuPokemon.attack(cpuMoveIndex, playerPokemon)
    print(cpuPokemon.name, "used", cpuMove.name)
    if (hit):
        print(cpuPokemon.name, "lands its atttack")
    else:
        print(cpuPokemon.name + "'s attack missed")
```

Next, the player will attack, but before they can, we need to check that they have health left. Then, we let the player attack in the same way that the computer did.

```
while cpuPokemon.health > 0 and playerPokemon.health > 0:
    ...
    if (hit):
        print(cpuPokemon.name, "lands its atttack")
    else:
        print(cpuPokemon.name + "'s attack missed")
    if (playerPokemon._______ > 0):
        hit = playerPokemon.attack(__________, ________)
        print(playerPokemon.name, "used", playerMove.name)
        if (hit):
            print(playerPokemon.name, "lands its attack")
        else:
            print(playerPokemon.name + "'s attack missed")
```

When the loop ends, the battle is over because one of the pokemon is out of health. That means we can declare a winner by checking who has health left outside of the while loop.

```
while cpuPokemon.health > 0 and playerPokemon.health > 0:
    ...

if (playerPokemon._________ > 0):
    print(cpuPokemon.name, "fainted")
    print("You Win")
else:
    print(____________._________, "fainted")
    print("You Lose")
```