Step 1.

```
import csv
file = open('Pokemon.csv', 'rt', encoding='UTF8')
csvreader = csv.reader(file)
header = next(csvreader)
data = []
for row in csvreader:
    if row[2] == 'Fire' or row[2] == 'Water' or row[3] == 'Fire' or row[3] == 'Water':
        row_edited = [row[1], row[2], row[3],
                      int(row[5]), int(row[6]), int(row[7]), row[12]]
        if row_edited[3] == '': row_edited[3] = None
        data.append(row_edited)
file.close()
```

csv 모듈의 함수인 csvreader 는 Pokemon.csv 를 한 줄씩 format 된 형태로 읽어올 수 있도록 해준다. 첫 줄을 제외하고 모든 줄을 읽어 나간다. 이때 row[2]와 row[3]이 Type 을 가리키며 이 둘 중에 'Water' 혹은 'Fire'가 없다면 그 줄은 무시한다. 이후 데이터의 사용을 더 편리하게 하기 위하여 필요한 정보만 남기고 숫자인 부분들은 int type 으로 변환해주어 data 에 append 해준다.

```
import numpy as np
rand = np.random.choice(len(data), 8)
```

numpy 를 사용하여 이후 step 에 필요한 8 개의 pokemon 을 뽑는다.

```
import pandas as pd
df = pd.DataFrame((data[i] for i in rand), columns =
                  ['Name', 'Type 1', 'Type 2', 'HP', 'Attack', 'Defense', 'Legendary'])
print(df)
```

뽑은 pokemon 들의 데이터로 dataframe 을 생성하고 출력한다.

	Name	Type 1	Type 2	HP	Attack	Defense	Legendary
0	Combusken	Fire	Fighting	60	85	60	False
1	Sealeo	Ice	Water	90	60	70	False
2	Camerupt	Fire	Ground	70	100	70	False
3	Entei	Fire		115	115	85	True
4	Flareon	Fire		65	130	60	False
5	Wailmer	Water		130	70	35	False
6	GyaradosMega Gyarados	Water	Dark	95	155	109	False
7	Fletchinder	Fire	Flying	62	73	55	False

선택한 Pokemon 의 데이터프레임 print() 출력 결과

Step 2.

Pokemon Class 내에서 사용되는 Class Variable 은 9 개가 있다. 변수는 다음과 같다.

self.name : 포켓몬의 이름을 저장한다.

self.type1,type2 : 종류(타입)이 어떤 것인지를 나타낸다.

self.is_knocked_out : 기절상태인지를 저장하는 bool 이다.

self.max_health : 최대 체력을 나타낸다. 초기에는 최대 체력과 현재 체력이 동일하기 때문에 정의할때는 health 로 정의한다.

self.health : 현재 체력을 나타낸다.

self.is_legendary : 전설의 포켓몬인지를 나타낸다.

self.Attk : 공격력을 나타낸다.

self. Def : 방어력을 나타낸다.

이렇게 정의된 변수들을 이용하여 Pokemon Class 를 구성하였다. 이 Pokemon Class 내에서 Representation Method 를 이용하여 포켓몬의 정보를 출력하는 부분은 다음과 같이 구현하였다.

```
 def __repr__(self):
    if self.is_legendary == 'True':
      return "\n== Pokemon Information ==\n\n**Congratulations! Your pokemon is a legendary
pokemon!**\nName: {}\nType 1: {}\nType 2: {} \nMaximun health: {}\nCurrent health: {}\nAttack: {}
\nDefense: {}\n".format(self.name, self.type1, self.type2, self.max_health, self.health, self.Attk,
self.Def)
    else:
      return "\n== Pokemon Information ==\n\nName: {}\nType 1: {}\nType 2: {}\nMaximun health:
{}\nCurrent health: {}\nAttack: {} \nDefense: {}\n".format(self.name, self.type1, self.type2,
self.max_health, self.health, self.Attk, self.Def)
```

이 method 는 if 문을 통해 2 부분으로 나누어져있는데, 포켓몬이 전설의 포켓몬인지를 나타내는 `is_legendary` 가 'True'면, `**Congratulations! Your pokemon is a legendary pokemon!**` 이라는 줄을 추가로 출력한다. 그 외에는 포켓몬에 대한 정보를 String 의 형태로 return 한다는 점은 동일하다.

포켓몬의 repr 출력은 다음과 같다.

```
== Pokemon Information ==

Name: Wailord
Type 1: Water
Type 2: None
Maximun health: 170
Current health: 170
Attack: 90
Defense: 45
```

```
== Pokemon Information ==

**Congratulations! Your pokemon is a legendary pokemon!**
Name: Kyogre
Type 1: Water
Type 2: None
Maximun health: 100
Current health: 100
Attack: 100
Defense: 90
```

일반 포켓몬과 전설의 포켓몬의 print() 출력 결과

Step 3.

Trainer Class 내에서 사용되는 Class Variable 은 5 개가 있다.

self.name : 트레이너의 이름

self.pokemon_list : 트레이너의 포켓몬의 이름

self.potions : 보유 포션의 개수

self.current_pokemon : 현재 포켓몬

self.Lost 트레이너가 패배하였는가를 나타내는 Bool

이렇게 정의된 변수들을 이용하여 Trainer Class 를 구성하였다. 이 Trainer Class 내에서 Representation Method 를 이용하여 트레이너의 정보를 출력하는 부분은 다음과 같이 구현하였다.

```
def __repr__(self):
    namelist = []
    i = 0
    while i < len(self.pokemon_list):
      namelist.append(self.pokemon_list[i].name)
      i += 1

    return "== Trainer information ==\nName:{} \nPokemons: {}\nPotions: {} \nCurrent pokemon: {}
\n{}".format(self.name, namelist, self.potions, self.current_pokemon.name, self.current_pokemon)
```

트레이너의 포켓몬의 이름만을 출력하기 위해서는, while loop 을 통해서 각 포켓몬의 이름만을 namelist 에 저장하였고, 나중에 return 을 할 때에는 이 이름들이 저장된 namelist 를 출력하는 방법을 사용하였다.

트레이너의 repr 출력은 다음과 같다.

```
== Trainer information ==
Name:Subaru
Pokemons: ['Slowpoke', 'Suicune', 'Pyroar', 'Infernape']
Potions: 3
Current pokemon: Slowpoke

== Pokemon Information ==

Name: Slowpoke
Type 1: Water
Type 2: Psychic
Maximun health: 90
Current health: 90
Attack: 65
Defense: 65
```

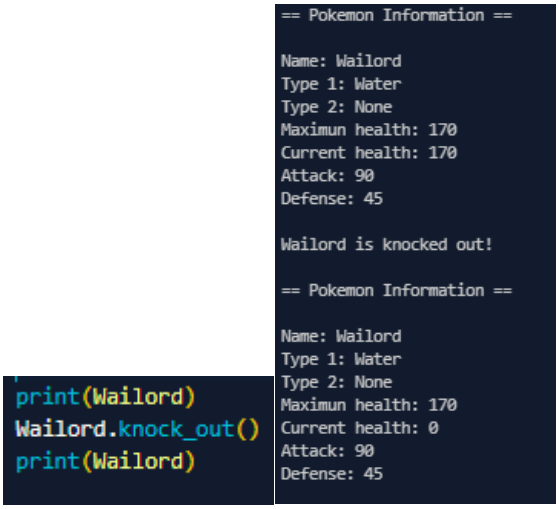
트레이너의 print()의 출력

Step4.

포켓몬을 기절시키는 method 는 다음과 같다.

```
  def knock_out(self):
    if self.health != 0:
      self.health = 0
    if self.is_knocked_out:
      print("{} is already knocked out.".format(self.name))
    else:
      self.is_knocked_out = True
      print("{} is knocked out!".format(self.name))
```

이 method 가 호출이 되면, 우선 체력이 0 이 아닌 경우, 체력을 0 으로 설정한다. 그 다음, 포켓몬이 이미 기절해있는지를 확인한다. 이는 `self.is_knocked_out` 가 True 인지를 확인하는 방법으로 진행한다. 만약 이미 포켓몬이 기절해있다면, 포켓몬이 이미 기절해있다는 사실을 알려준다. 만약 포켓몬이 기절해있지 않다면, `is_knocked_out` 를 1 로 설정하고, 포켓몬이 기절하였다는 상태메시지를 띄운다. 이 출력은 다음과 같다.



knock_out() 전과 후의 출력

포켓몬을 부활시키는 method 는 다음과 같다.

```
def revive(self):
    if self.is_knocked_out:
      self.is_knocked_out = False
      self.health = 1
      print("{} has been revived with {} health!".format(self.name, self.health))
    else:
      print("{} is not knocked out.".format(self.name))
```

이 method 가 호출이 되면, 우선 포켓몬이 기절상태인지를 확인한다. 기절상태가 아니면, 기절상태가 아니라고 문구를 띄운다. 만약 포켓몬이 기절상태라면, `is_knocked_out` 를 0 으로 설정하고, 체력을 1 로 설정함으로서 포켓몬을 부활시킨다. 이 출력은 다음과 같다. 시연을 위해서 미리 `knock_out()`를 호출하였다.

```
Wailord.knock_out()
print(Wailord)
Wailord.revive()
print(Wailord)
```

```
Wailord is knocked out!

== Pokemon Information ==

Name: Wailord
Type 1: Water
Type 2: None
Maximun health: 170
Current health: 0
Attack: 90
Defense: 45

Wailord has been revived with 1 health!

== Pokemon Information ==

Name: Wailord
Type 1: Water
Type 2: None
Maximun health: 170
Current health: 1
Attack: 90
Defense: 45
```

revive() 전후의 출력

체력이 닳는 상태는 다음과 같이 구현하였다.

```
def lose_health(self, dmg):
  self.health -= dmg
  if self.health <= 0:
    self.health = 0
    self.knock_out()
```

이는 dmg 의 값을 받아, 포켓몬의 체력을 나타내는 `self.health` 에서 dmg 만큼을 빼서 체력을 줄인다. 이 때, 만약 체력이 0 이하로 떨어지면 체력을 0 으로 설정하고, `knock_out()`를 호출하여 포켓몬을 기절시킨다. 이 함수의 출력은 다음과 같다.

```
print(Wailord)
Wailord.lose_health(100)
print(Wailord)
Wailord.lose_health(100)
print(Wailord)
```

```
== Pokemon Information ==

Name: Wailord
Type 1: Water
Type 2: None
Maximun health: 170
Current health: 170
Attack: 90
Defense: 45


== Pokemon Information ==

Name: Wailord
Type 1: Water
Type 2: None
Maximun health: 170
Current health: 70
Attack: 90
Defense: 45

Wailord is knocked out!

== Pokemon Information ==

Name: Wailord
Type 1: Water
Type 2: None
Maximun health: 170
Current health: 0
Attack: 90
Defense: 45
```

lose_health()의 결과

체력을 회복하는 기능은 다음과 같이 구현하였다.

```
def gain_health(self, heal):
  if self.is_knocked_out == False:
    self.health += heal
    if self.health >= self.max_health:
      self.health = self.max_health
      print("{} healed to max health!".format(self.name))
    else:
      print("{} gained {} health!".format(self.name, heal))
    print("Current health: {}".format(self.health))
  else:
    print("Pokemon fainted; Cannot heal!")
```

이 method 가 호출이 되면, 우선 포켓몬이 기절상태인지 확인한다. 포켓몬이 기절상태이면, 체력회복을 하지 않는다. 만약 포켓몬이 살아있다면, 포켓몬의 `self.health` 에 회복할 체력을 더해서 체력을 회복한다. 만약 회복한 체력이 최대 체력보다 높아지면, 현재 체력을 최대 체력으로 설정하여, 과치료 현상을 방지한다. 시행 결과는 다음과 같다. 시연을 위해서 미리 `lose_health()`와 `knock_out()`를 앞에 추가하였다.

```
Wailord.lose_health(100)
Emboar.knock_out()


print(Wailord)
Wailord.gain_health(50)
print(Wailord)
Wailord.gain_health(150)

print("=======")

Emboar.gain_health(100)
print(Emboar)
```

```
== Pokemon Information ==

Name: Wailord
Type 1: Water
Type 2: None
Maximun health: 170
Current health: 70
Attack: 90
Defense: 45

Wailord gained 50 health!
Current health: 120

== Pokemon Information ==

Name: Wailord
Type 1: Water
Type 2: None
Maximun health: 170
Current health: 120
Attack: 90
Defense: 45

Wailord healed to max health!
Current health: 170
=======
Pokemon fainted; Cannot heal!

== Pokemon Information ==

Name: Emboar
Type 1: Fire
Type 2: Fighting
Maximun health: 110
Current health: 0
Attack: 123
Defense: 65
```

gain_health()의 실행 결과

모든 method 가 의도한 대로 동작하는 것을 알 수 있다.

Step 5.

포켓몬 타입에 따른 데미지는 다음과 같다.

이 표를 바탕으로, 포켓몬의 공격에 곱해지는 가중치를 설정하였다. 이 때, 효과가 굉장한 공격은 가중치를 1.5 로, 효과가 일반적인 공격은 가중치를 1 로, 효과가 별로인 공격은 가중치를 0.5 로 설정하였다. 이렇게 한 이유는, 문제에서 주어진것처럼 효과가 굉장한 공격의 가중치를 0.5, 별로인 공격의 가중치를 0.25 로 설정하면, 가중치값이 없는 일반공격보다 효과가 굉장한 공격이 약해지는 경우가 발생하기 때문이다. 문제에서는 물타입과 불타입 포켓몬만 사용하였기 때문에, 일반 공격에 대해서는 고려할 필요가 없지만, 전체 상성표를 작성할 때에는 이 상황도 고려해야 하기 때문에, 주어진 문제에서 가중치를 약간 바꾸었다. 이렇게 설정한 가중치를 Python Dictionary 를 이용해서 만들면 다음과 같다.

```
DamageChart = { #데미지 상성표.
    'Normal' : {'Rock' : 0.5, 'Ghost' : 0, 'Steel' : 0.5},
    'Fighting' : {'Fighting' : 1.5, 'Poison' : 0.5, 'Flying' : 0.5, 'Bug' : 0.5, 'Rock' : 1.5, 'Ghost' : 0, 'Steel' : 1.5, 'Ice' : 1.5, 'Psychic' : 0.5, 'Dark' : 1.5, 'Fairy' : 0.5},
    'Poison' : {'Poison': 0.5, 'Ground' : 0.5, 'Rock' : 0.5, 'Ghost' : 0.5, 'Steel' : 0, 'Grass' : 1.5, 'Fairy' : 1.5},
    'Ground' : {'Poison' : 1.5, 'Flying' : 0, 'Bug' : 0.5, 'Rock' : 1.5, 'Steel' : 1.5, 'Fire' : 1.5, 'Electric' : 1.5, 'Grass' : 0.5},
    'Flying' : {'Fighting' : 1.5, 'Bug' : 1.5, 'Rock' : 0.5, 'Electric' : 0.5, 'Grass' : 1.5},
    'Bug' : {'Fighting' : 0.5, 'Poison' : 0.5, 'Flying' : 0.5, 'Ghost' : 0.5, 'Steel' : 0.5, 'Fire' : 0.5, 'Grass' : 1.5, 'Psychic' : 1.5, 'Dark' : 1.5, 'Fairy' : 0.5},
    'Rock' : {'Fighting' : 0.5, 'Ground' : 0.5, 'Flying' : 1.5, 'Bug' : 1.5, 'Steel' : 0.5, 'Fire' : 1.5, 'Ice' : 1.5},
    'Ghost' : {'Normal' : 0, 'Ghost' : 0.5, 'Psychic' : 1.5, 'Dark' : 0.5},
    'Steel' : {'Rock' : 1.5, 'Steel' : 0.5, 'Fire' : 0.5, 'Water' : 0.5, 'Electric' : 0.5, 'Ice' : 1.5, 'Fairy' : 1.5},
    'Fire' : {'Bug' : 1.5, 'Rock' : 0.5, 'Steel' : 1.5, 'Fire' : 0.5, 'Water' : 0.5, 'Grass' : 1.5, 'Ice' : 1.5, 'Dragon' : 0.5},
    'Water' : {'Ground' : 1.5, 'Rock' : 1.5, 'Fire' : 1.5, 'Water' : 0.5, 'Grass' : 0.5, 'Dragon' : 0.5},
    'Electric' : {'Ground' : 0, 'Flying' : 1.5, 'Water' : 1.5, 'Electric' : 0.5, 'Grass' : 0.5, 'Dragon' : 0.5},
    'Grass' : {'Poison' : 0.5, 'Ground' : 1.5, 'Flying' : 0.5, 'Bug' : 0.5, 'Rock' : 1.5, 'Steel' : 0.5, 'Fire' : 0.5, 'Water' : 1.5, 'Grass' : 0.5, 'Dragon' : 0.5},
    'Ice' : {'Ground': 1.5, 'Flying' : 1.5, 'Steel' : 0.5, 'Fire' : 0.5, 'Water' : 0.5, 'Grass' : 1.5, 'Ice' : 0.5, 'Dragon' : 1.5},
    'Psychic' : {'Fighting' : 1.5, 'Poison' : 1.5, 'Steel' : 0.5, 'Psychic' : 0.5, 'Dark' : 0},
    'Dragon' : {'Steel' : 0.5, 'Dragon' : 1.5, 'Fairy' : 0},
    'Dark' : {'Fighting' : 0.5, 'Ghost' : 1.5, 'Psychic' : 1.5, 'Dark' : 0.5, 'Fairy' : 0.5},
    'Fairy' : {'Fighting' : 1.5, 'Poison' : 0.5, 'Steel' : 0.5, 'Fire' : 0.5, 'Dragon' : 1.5, 'Dark' : 1.5}
            }
```

이 상성표는 Nested Dictionary 를 이용하여 제작하였다. 첫번째(바깥의/큰)Dictionary 는 공격하는 포켓몬의 타입을 index 로 사용한다. 각 index 에 지정된, 두번째(안의/작은) Dictionary 는 이전에 선택한 Index(타입)의 공격이 강한/약한 타입을 Index 로 갖고, 각 항목의 값은 곱해지는 가중치이다.

예를 들어, 고스트 타입의 Dictionary 는 다음과 같다.

```
'Ghost' : {'Normal' : 0, 'Ghost' : 0.5, 'Psychic' : 1.5, 'Dark' : 0.5},
```

이 때, Ghost 인덱스를 갖는 Dictionary 는 Normal 일때 0, Ghost 일 때 0.5, Psychic 일 때 1.5, Dark 일 때 0.5 의 값을 갖는다. 이는 Ghost 타입이 공격할 때, Normal 타입에게는 공격이 효과가 없고, Ghost, Dark 타입에게는 별로 효과가 없으며, Psychic 타입에게는 공격효과가 굉장하다는 것을 의미한다.

이 상성표를 이용하여 가중치의 값을 설정하는 법은 다음과 같다.

```
    Modifier = 1

    try:
        Modifier = DamageChart[self.type][other.type]
    except KeyError:
        Modifier = Modifier
```

Modifier 의 기본값은 1 이고, 이전에 작성한 상성표에서 특수효과가 있는 상성인지를 try 를 통해 확인한다. 만약 상성이 존재하지 않으면, `DamageChart` 에 항목이 존재하지 않기 때문에 `KeyError` 가 뜨게 된다. 이 경우, 상성 효과가 존재하지 않는 것이기 때문에, Modifier 에 변화를 주지 않는다.

이렇게 구현한 데미지 가중치 시스템을 이용하여 Pokemon Class 내에서 공격하는 method 를 만들면 다음과 같다. (상성표 Dictionary 는 너무 길어서 사진에서는 숨김처리하였다.)

```
def attack(self, other):
  attack = self.Attk
  defense = other.Def
  Modifier = 1

  if self.is_knocked_out == True: # 기절 시 공격 못함
    print("{} is knocked out!".format(self.name))
    return

  DamageChart = { #데미지 상성표. …
  try:
      Modifier = DamageChart[self.type1][other.type1] #데미지 상성표에 따라
  except KeyError:
      Modifier = 1 #데미지 상성표에 표시가 되지 않았을 경우, Modifier를 1로

  try:
      Modifier = DamageChart[self.type1][other.type2]
  except KeyError:
      Modifier = Modifier

  try:
      Modifier = DamageChart[self.type2][other.type1]
  except KeyError:
      Modifier = Modifier

  try:
      Modifier = DamageChart[self.type2][other.type2]
  except KeyError:
      Modifier = Modifier

  dmg = Modifier * (attack - defense)
  dmg = abs(dmg)

  # dmg의 값을 바탕으로 실제로 공격을 함
  print("{} attacked {}!".format(self.name, other.name))

  if Modifier == 1.5:
    print("It was very effective!") # 효과가 굉장했다!
  elif Modifier == 0.5:
    print("It was not very effective...") # 효과가 별로인듯 하다...

  print("{} dealt {} damage to {}.".format(self.name, dmg, other.name))
  other.lose_health(dmg)
  print("Enemy Health: {}".format(other.health))
  print("")
```

이 코드를 각 부분별로 나누어 분석하면 다음과 같다.

```
    attack = self.Attk
    defense = other.Def
    Modifier = 1
```

인스턴스 변수를 생성하는 부분이다. attack 는 현재 포켓몬의 공격력, defense 는 당대 포켓몬의 방어력이고, Modifier 는 가중치값으로, 기본값은 1 로 설정하였다.

```
    if self.is_knocked_out == True:

      print("{} is knocked out!".format(self.name))
      return
```

포켓몬이 기절하였는지를 확인하는 부분이다. 기절한 포켓몬은 공격을 할 수 없기 때문에, `self.is_knocked_out` 를 확인하여 포켓몬이 기절해있는지 아닌지를 확인한다. 만약 포켓몬이 기절해있다면 메시지를 띄우고 그대로 함수를 return 한다.

```
DamageChart = {…}
    try:
        Modifier = DamageChart[self.type1][other.type1]
    except KeyError: …

    try:
```

```
        Modifier = DamageChart[self.type1][other.type2]
    except KeyError: …
        Modifier = Modifier

    try:
        Modifier = DamageChart[self.type2][other.type1]
    except KeyError: …

    try:
        Modifier = DamageChart[self.type2][other.type2]
    except KeyError: …
```

다음 부분은 상성표를 바탕으로 가중치를 설정하는 부분이다. 이 부분은 위에서 설명한대로, 현재 포켓몬과 상대 포켓몬의 타입을 바탕으로 공격의 가중치를 설정한다. 이 때 try 를 4 번 사용하였는데, 이는 각 포켓몬은 최대 타입이 2 개까지 있을 수 있기 때문에, 총 가능한 상성의 경우의 수는 4 가지가 되기 때문이다. 실제 포켓몬스터 게임에서는 두개의 타입과 상성에 따라서 추가 효과가 발생하지만, 이 프로젝트에서는 단순히 공격력의 가중치를 바꾸는 방법으로 계산하였다.

```
    dmg = Modifier * (attack - defense)
    dmg = abs(dmg)
```

실제 공격력을 계산하는 부분이다. 공격력은 다음과 같다.

절댓값(가중치*(자신의 공격력 – 상대방의 방어력))

마지막으로, 실제로 공격을 하는 부분은 다음과 같다.

```
    print("{} attacked {}!".format(self.name, other.name))

    if Modifier == 1.5:
      print("It was very effective!")
    elif Modifier == 0.5:
      print("It was not very effective...")
    elif Modifier == 0:
      print("There was no effect!")

    print("{} dealt {} damage to {}.".format(self.name, dmg, other.name))
    other.lose_health(dmg)
    print("Enemy Health: {}".format(other.health))
    print("")
```

우선 공격을 하였다는 메시지를 띄운다. 그 다음 가중치에 따라 특수효과가 있다면 공격이 효과적이었는지, 별로였는지, 효과가 없는지를 출력한 뒤, 공격에 대한 정보를 출력한다. 그 다음에는 `other.lose_health()` 를 이용하여 데미지의 크기만큼 상대 포켓몬의 체력을 깎는다. 마지막으로, 상대방의 체력을 표시한다.

이렇게 해서 상대 포켓몬을 공격하는 method 를 만들 수 있고, 이 결과는 다음과 같다.

```
Raticate.attack(Emboar)

print("\n")

Suicune.attack(Ponyta)

print("\n")

Ponyta.attack(Suicune)

print("\n")

Raticate.attack(Misdreavus)

print("\n")
```

```
Raticate attacked Emboar!
Raticate dealt 16 damage to Emboar.
Enemy Health: 94
```
특수효과가 없는 일반 공격

```
Suicune attacked Ponyta!
It was very effective!
Suicune dealt 30.0 damage to Ponyta.
Enemy Health: 20.0
```
효과가 굉장한 공격

```
Ponyta attacked Suicune!
It was not very effective...
Ponyta dealt 15.0 damage to Suicune.
Enemy Health: 85.0
```
효과가 별로인 공격

```
Raticate attacked Misdreavus!
There was no effect!
Raticate dealt 0 damage to Misdreavus.
Enemy Health: 60
```
효과가 없는 공격

이를 이용하여, Trainer Class 내에서 상대 트레이너를 공격하는 method 를 만들면 다음과 같다.

```
  def attack(self, other):
    self.current_pokemon.attack(other.current_pokemon)
```

상대 트레이너의 현재 포켓몬인 `other.current_pokemon` 을 받아, 자신의 현 포켓몬인 `self.current_pokemon` 과 함께 위의 pokemon 클래스의 attack 메소드를 사용한다.

Trainer Class 내에서 회복약인 Potion 을 사용하는 method 는 다음과 같다.

```
def use_potion(self):
  if self.potions > 0:
    print("{} used a potion!".format(self.name))
    if self.current_pokemon.is_knocked_out == True:
      print("Potion was not used.")
    elif self.current_pokemon.health < self.current_pokemon.max_health:
      self.current_pokemon.gain_health(100)
      self.potions -= 1

      print("{} has {} potions left.\n".format(self.name, self.potions))
    else:
      print("Pokemon already has maximum health.\n")
  else:
    print("No potions available!\n".format(self.name))
```

이 method 는 우선 `self.potions` 를 확인해서 트레이너에게 포션이 남아있는지 확인한다.

```
if self.potions > 0:

      print("{} used a potion!".format(self.name))
```

만약 사용 가능한 포션이 있으면 포션을 사용하였다고 안내를 한다.

```
  else:
      print("No potions available!\n")
```

만약 사용가능한 포션이 없으면 포션이 없다고 안내를 한다.

포션을 사용하였을 때, 포켓몬이 기절해있으면 포켓몬을 회복할 수 없기 때문에 `current_pokemon.is_knocked_out` 를 확인하여 현재 포켓몬의 상태를 확인한다.

```
       if self.current_pokemon.is_knocked_out == True:

         print("Potion was not used.")
```

만약 포켓몬이 기절해있다면 포션을 사용하지 않는다.

```
      elif self.current_pokemon.health < self.current_pokemon.max_health:
        self.current_pokemon.gain_health(100)
        self.potions -= 1

        print("{} has {} potions left.\n".format(self.name, self.potions))
      else:
        print("Pokemon already has maximum health.\n")
```

이후, 포켓몬의 체력이 최대 체력인지를 확인한다. 만약 포켓몬의 체력이 `max_health` 과 동일하다면, 체력이 최대라는 뜻이므로, 포션을 사용하지 않는다. 그 외의 경우에는 포켓몬의 체력을 100 회복한 뒤, 트레이너가 보유한 포션의 수를 1 만큼 감소한다. 이 method 의 출력은 다음과 같다.

```
Trainer_1.current_pokemon.lose_health(70)
Trainer_1.use_potion()

Trainer_1.use_potion()

Trainer_1.current_pokemon.lose_health(100)
Trainer_1.use_potion()

Trainer_1.current_pokemon.revive()
Trainer_1.use_potion()

Trainer_1.current_pokemon.lose_health(70)
Trainer_1.use_potion()

Trainer_1.current_pokemon.lose_health(70)
Trainer_1.use_potion()
```

```
Jun used a potion!
Charmeleon healed to max health!
Current health: 78
Jun has 2 potions left.
```

일반적으로 포션을 사용한 경우

```
Jun used a potion!
Pokemon already has maximum health.
```

포켓몬이 최대 체력인 경우

```
Charmeleon is knocked out!
Jun used a potion!
Potion was not used.
```

포켓몬이 기절한 경우

```
No potions available!
```

남은 포션이 없는 경우

위의 method 외에도 성공적으로 포켓몬 배틀을 하기 위해서는 포켓몬을 교체하는 method 가 필요하다. 이는 다음과 같다.

```
def NextPokemon(self):
  i = 1
  while i < len(self.pokemon_list):
    if self.pokemon_list[i].get_knocked_out() == True: #마지막 포켓몬이 기절한 경우 Lost = 1.
      if i == (len(self.pokemon_list) - 1):
        self.Lost = True
        print("All of {}'s pokemon fainted!".format(self.name))
        break
      else:
        i += 1
    else:
      self.current_pokemon = self.pokemon_list[i] # 다음 포켓몬이 살아있으면 그 포켓몬으로 변경
      print("{} changed their pokemon to {}!".format(self.name, self.current_pokemon.name))
      break
```

이 method 는 트레이너의 보유 포켓몬을 순서대로 확인하여, 기절하지 않은 첫번째 포켓몬은 current_pokemon 으로 설정하여 내보낸다. 만약 모든 포켓몬이 기절하였을 경우, self.Lost 를 True 로 설정하여, 트레이너의 패배를 표시한다.

```
Trainer_1.current_pokemon.knock_out()

Trainer_1.NextPokemon()
Trainer_1.current_pokemon.knock_out()

Trainer_1.NextPokemon()
Trainer_1.current_pokemon.knock_out()

Trainer_1.NextPokemon()
Trainer_1.current_pokemon.knock_out()

Trainer_1.NextPokemon()
```

`Jun changed their pokemon to Blastoise!` 포켓몬을 바꾸었을 경우

`All of Jun's pokemon fainted!` 모든 포켓몬이 기절한 경우

이를 이용하여 포켓몬 배틀을 진행하는 함수를 만들면 다음과 같다.

```
def PokemonBattle(Trainer_1, Trainer_2): # 실제 포켓몬 배틀을 진행하는 함수.
  print("\n\nPokemon Battle Start!\n\n")

  while(Trainer_1.Lost == False and Trainer_2.Lost == False):
    if Trainer_1.current_pokemon.health <= 20: #포켓몬의 체력이 20 이하가 되면 포션 사용
      if Trainer_1.current_pokemon.health <= 0: #현재 포켓몬이 기절하였을 경우 다음 포켓몬으로 교체
          Trainer_1.NextPokemon()
      elif Trainer_1.potions != 0:
        Trainer_1.use_potion()

      else:
        Trainer_1.attack(Trainer_2) # 포션이 없고 포켓몬이 살아있을 경우 공격
    else:
      Trainer_1.attack(Trainer_2)

    if Trainer_1.Lost == True: #모든 포켓몬이 기절했을 때 그 자리에서 배틀을 끝냄.
      break;

    if Trainer_2.current_pokemon.health <= 20:
      if Trainer_2.current_pokemon.health <= 0:
          Trainer_2.NextPokemon()
      elif Trainer_2.potions != 0:
        Trainer_2.use_potion()

      else:
        Trainer_2.attack(Trainer_2)
    else:
      Trainer_2.attack(Trainer_1)

  print("\nBattle End!") # 배틀이 끝나면 승자를 표시
  if Trainer_1.Lost == True:
    print("{} Wins!".format(Trainer_2.name))
  else:
    print("{} Wins!".format(Trainer_1.name))
```

이 함수는 두 트레이너 모두 Lost 상태가 아닌 경우에 계속 아래의 loop 를 진행한다.

```
while(Trainer_1.Lost == False and Trainer_2.Lost == False):
  if Trainer_1.current_pokemon.health <= 20: #포켓몬의 체력이 2
    if Trainer_1.current_pokemon.health <= 0: #현재 포켓몬이 기
        Trainer_1.NextPokemon()
    elif Trainer_1.potions != 0:
      Trainer_1.use_potion()

    else:
      Trainer_1.attack(Trainer_2) # 포션이 없고 포켓몬이 살아있
  else:
    Trainer_1.attack(Trainer_2)

  if Trainer_1.Lost == True: #모든 포켓몬이 기절했을 때 그 자리
    break;

  if Trainer_2.current_pokemon.health <= 20:
    if Trainer_2.current_pokemon.health <= 0:
        Trainer_2.NextPokemon()
    elif Trainer_2.potions != 0:
      Trainer_2.use_potion()

    else:
      Trainer_2.attack(Trainer_2)
  else:
    Trainer_2.attack(Trainer_1)
```

이 loop 내에서, 각 트레이너는 자신의 포켓몬의 체력이 20 이하로 떨어지는 경우, 포션을 사용한다. 만약 포션을 사용할 수 없으면 상대 트레이너의 포켓몬을 공격한다. 또한, 만약 현재 포켓몬이 기절한 경우, 다음 포켓몬으로 교체를 한다. 이 loop 도중에 한 트레이너의 모든 포켓몬이 기절하면, 그대로 loop 를 나온다.

```
print("\nBattle End!") # 배틀이 끝나면 승자를 출
if Trainer_1.Lost == True:
  print("{} Wins!".format(Trainer_2.name))
else:
  print("{} Wins!".format(Trainer_1.name))
```

배틀이 끝나면 승자를 출력한다.

예시 출력은 다음과 같다. 출력이 길기 때문에 일부만 첨부하였고, 전체 출력은 별도 파일로 첨부하였다.

Pokemon Battle Start!

Charmeleon attacked Slowpoke!

It was not very effective...

Charmeleon dealt 19.5 damage to Slowpoke.

Enemy Health: 70.5

Slowpoke attacked Charmeleon!

It was very effective!

Slowpoke dealt 19.5 damage to Charmeleon.

Enemy Health: 58.5

...

Slowpoke attacked Charmeleon!

It was very effective!

Slowpoke dealt 19.5 damage to Charmeleon.

Enemy Health: 19.5

Jun used a potion!

Charmeleon healed to max health!

Current health: 78

Jun has 2 potions left.

...

Slowpoke attacked Charmeleon!

It was very effective!

Slowpoke dealt 19.5 damage to Charmeleon.

Charmeleon is knocked out!

Enemy Health: 0

Jun changed their pokemon to Blastoise!

Slowpoke attacked Blastoise!

It was not very effective...

Slowpoke dealt 17.5 damage to Blastoise.

Enemy Health: 61.5

...

Pyroar attacked Delphox!

It was not very effective...

Pyroar dealt 2.0 damage to Delphox.

Delphox is knocked out!

Enemy Health: 0

All of Jun's pokemon fainted!

Battle End!

Subaru Wins!

이 경우 외에도, 문제에서 주어진 4 가지 배틀 결과를 정리하면 다음과 같다. 타입이 같아도, 포켓몬의 종류에 따라서 승패는 바뀔 수 있지만, 임의로 정한 포켓몬 배틀의 결과는 다음과 같다.

이 결과 역시 위의 경우처럼, 모든 출력을 적기에는 너무 길기 때문에, 결과만 작성하였고, 전체 배틀의 진행은 별도 파일로 첨부하였다.

1) 각 트레이너의 포션 수와 타입이 모두 같은 경우

```
All of Fire1's pokemon fainted!

Battle End!
Fire2 Wins!
```

팀 구성이 더 강한 트레이너가 우승하였다.

2) 각 트레이너의 포션 수 및 타입이 모두 다른 경우

```
All of Water2's pokemon fainted!

Battle End!
Fire2 Wins!
```

불리한 타입의 트레이너에게 더 많은 포션을 준 결과, 더 많은 포션을 가진 트레이너가 우승하였다.

3) 각 트레이너의 포션 수는 같고 타입은 다른 경우

```
All of Fire2's pokemon fainted!

Battle End!
Water2 Wins!
```

유리한 타입의 트레이너가 우승하였다.

4) 각 트레이너의 포션 수는 다르고 타입은 같은 경우

```
All of Fire1's pokemon fainted!

Battle End!
Fire2 Wins!
```

팀 구성이 강한 트레이너가 포션의 수가 적음에도 불구하고 우승하였다.

이 게임은 각 트레이너의 팀 구성, 그리고 포션의 수에 따라서 결과가 달라지기 때문에, 어떤 경우, 누가 이기는지를 정확하게 말할 수 없다. 그러나 임의의 포켓몬 구성으로 프로그램을 실행한 결과는 다음과 같았다.