Step 1.

주어진 Pokemon.csv에서 데이터를 읽어오는 방법을 모르겠다.

Step 2.

Pokemon Class 내에서 사용되는 Class Variable은 9개가 있다. 변수는 다음과 같다.

self.name : 포켓몬의 이름을 저장한다.

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

self.is\_knocked\_out : 기절상태인지를 저장하는 bool이다.

self.max\_health : 최대 체력을 나타낸다. 초기에는 최대 체력과 현재 체력이 동일하기 때문에 정의할때는 health로 정의한다.

self.health : 현재 체력을 나타낸다.

self.is\_legendary : 전설의 포켓몬인지를 저장하는 bool이다.

self.Attk : 공격력을 나타낸다.

self. Def : 방어력을 나타낸다.

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

 def \_\_repr\_\_(self):

    if self.is\_legendary == True:

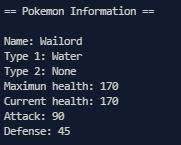
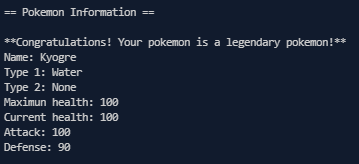
      return "\n== Pokemon Information ==\n\n\*\*Congratulations! Your pokemon is a legendary pokemon!\*\*\nName: {}\nType 1: {}\nType 2: {} \nMaximun health: {}\nCurrent health: {}\nAttack: {} \nDefense: {}\n".format(self.name, self.type1, self.type2, self.max\_health, self.health, self.Attk, self.Def)

    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 method는 if문을 통해 2 부분으로 나누어져있는데, 포켓몬이 전설의 포켓몬인지를 나타내는 is\_legendary bool이 True면, \*\*Congratulations! Your pokemon is a legendary pokemon!\*\* 이라는 줄을 추가로 출력한다. 그 외에는 포켓몬에 대한 정보를 String의 형태로 return한다는 점은 동일하다.

포켓몬의 repr 출력은 다음과 같다.

일반 포켓몬과 전설의 포켓몬의 print() 출력 결과

Step 3.

Trainer Class 내에서 사용되는 Class Variable은 5개가 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 Class를 구성하였다. 이 Trainer Class 내에서 Representation Method를 이용하여 트레이너의 정보를 출력하는 부분은 다음과 같이 구현하였다.

def \_\_repr\_\_(self):

    namelist = []

    i = 0

    while i < len(self.pokemon\_list):

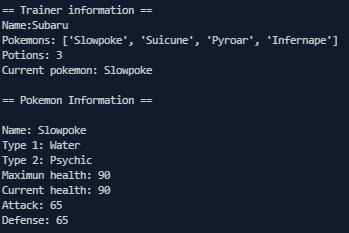
      namelist.append(self.pokemon\_list[i].name)

      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 loop을 통해서 각 포켓몬의 이름만을 namelist에 저장하였고, 나중에 return을 할 때에는 이 이름들이 저장된 namelist를 출력하는 방법을 사용하였다.

트레이너의 repr 출력은 다음과 같다.



트레이너의 print()의 출력

Step4.

포켓몬을 기절시키는 method는 다음과 같다.

  def knock\_out(self):

    if self.health != 0:

      self.health = 0

    if self.is\_knocked\_out:

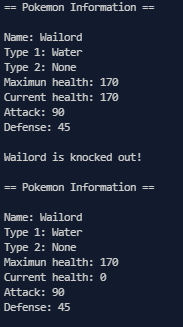
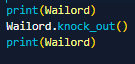
      print("{} is already knocked out.".format(self.name))

    else:

      self.is\_knocked\_out = True

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

이 method가 호출이 되면, 우선 체력이 0이 아닌 경우, 체력을 0으로 설정한다. 그 다음, 포켓몬이 이미 기절해있는지를 확인한다. 이는 self.is\_knocked\_out가 True 인지를 확인하는 방법으로 진행한다. 만약 이미 포켓몬이 기절해있다면, 포켓몬이 이미 기절해있다는 사실을 알려준다. 만약 포켓몬이 기절해있지 않다면, is\_knocked\_out를 1로 설정하고, 포켓몬이 기절하였다는 상태메시지를 띄운다. 이 출력은 다음과 같다.



knock\_out() 전과 후의 출력

포켓몬을 부활시키는 method는 다음과 같다.

 def revive(self):

    if self.is\_knocked\_out:

      self.is\_knocked\_out = False

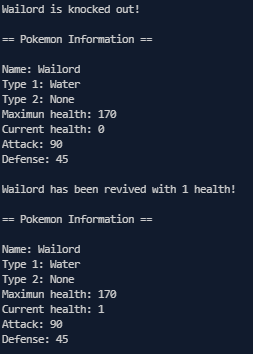
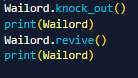
      self.health = 1

      print("{} has been revived with {} health!".format(self.name, self.health))

    else:

      print("{} is not knocked out.".format(self.name))

이 method가 호출이 되면, 우선 포켓몬이 기절상태인지를 확인한다. 기절상태가 아니면, 기절상태가 아니라고 문구를 띄운다. 만약 포켓몬이 기절상태라면, is\_knocked\_out를 0으로 설정하고, 체력을 1로 설정함으로서 포켓몬을 부활시킨다. 이 출력은 다음과 같다. 시연을 위해서 미리 knock\_out()를 호출하였다.



revive() 전후의 출력

체력이 닳는 상태는 다음과 같이 구현하였다.

  def lose\_health(self, dmg):

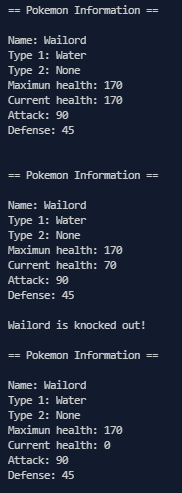
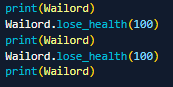
    self.health -= dmg

    if self.health <= 0:

      self.health = 0

      self.knock\_out()

이는 dmg의 값을 받아, 포켓몬의 체력을 나타내는 self.health에서 dmg 만큼을 빼서 체력을 줄인다. 이 때, 만약 체력이 0 이하로 떨어지면 체력을 0으로 설정하고, knock\_out()를 호출하여 포켓몬을 기절시킨다. 이 함수의 출력은 다음과 같다.



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:

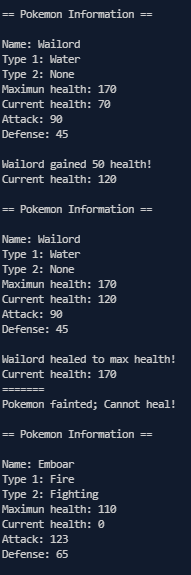
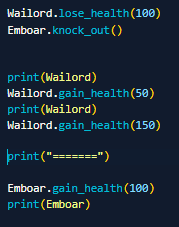
        print("{} gained {} health!".format(self.name, heal))

      print("Current health: {}".format(self.health))

    else:

      print("Pokemon fainted; Cannot heal!")

이 method가 호출이 되면, 우선 포켓몬이 기절상태인지 확인한다. 포켓몬이 기절상태이면, 체력회복을 하지 않는다. 만약 포켓몬이 살아있다면, 포켓몬의 self.health에 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.



gain\_health()의 실행 결과

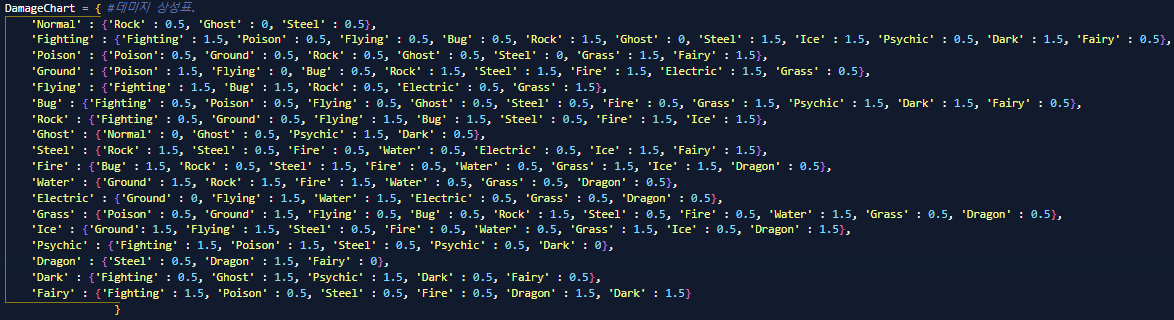
모든 method가 의도한 대로 동작하는 것을 알 수 있다.

Step 5.

포켓몬 타입에 따른 데미지는 다음과 같다.



이 표를 바탕으로, 포켓몬의 공격에 곱해지는 가중치를 설정하였다. 이 때, 효과가 굉장한 공격은 가중치를 1.5로, 효과가 일반적인 공격은 가중치를 1로, 효과가 별로인 공격은 가중치를 0.5로 설정하였다. 이렇게 한 이유는, 문제에서 주어진것처럼 효과가 굉장한 공격의 가중치를 0.5, 별로인 공격의 가중치를 0.25로 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 Dictionary를 이용해서 만들면 다음과 같다.



이 상성표는 Nested Dictionary를 이용하여 제작하였다. 첫번째(바깥의/큰)Dictionary는 공격하는 포켓몬의 타입을 index로 사용한다. 각 index에 지정된, 두번째(안의/작은) Dictionary는 이전에 선택한 Index(타입)의 공격이 강한/약한 타입을 Index로 갖고, 각 항목의 값은 곱해지는 가중치이다.

예를 들어, 고스트 타입의 Dictionary는 다음과 같다.

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

이 때, Ghost 인덱스를 갖는 Dictionary는 Normal 일때 0, Ghost일 때 0.5, Psychic일 때 1.5, Dark일 때 0.5의 값을 갖는다. 이는 Ghost타입이 공격할 때, Normal타입에게는 공격이 효과가 없고, Ghost, Dark타입에게는 별로 효과가 없으며, Psychic 타입에게는 공격효과가 굉장하다는 것을 의미한다.

이 상성표를 이용하여 가중치의 값을 설정하는 법은 다음과 같다.

    Modifier = 1

try:

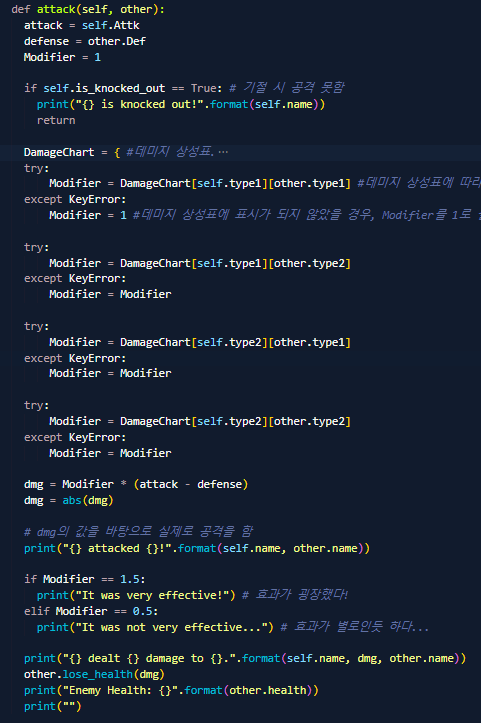
        Modifier = DamageChart[self.type][other.type]

    except KeyError:

        Modifier = Modifier

Modifier의 기본값은 1이고, 이전에 작성한 상성표에서 특수효과가 있는 상성인지를 try를 통해 확인한다. 만약 상성이 존재하지 않으면, DamageChart에 항목이 존재하지 않기 때문에 KeyError가 뜨게 된다. 이 경우, 상성 효과가 존재하지 않는 것이기 때문에, Modifier에 변화를 주지 않는다.

이렇게 구현한 데미지 가중치 시스템을 이용하여 Pokemon Class 내에서 공격하는 method를 만들면 다음과 같다. (상성표 Dictionary는 너무 길어서 사진에서는 숨김처리하였다.)



이 코드를 각 부분별로 나누어 분석하면 다음과 같다.

    attack = self.Attk

    defense = other.Def

    Modifier = 1

인스턴스 변수를 생성하는 부분이다. attack는 현재 포켓몬의 공격력, defense는 당대 포켓몬의 방어력이고, Modifier는 가중치값으로, 기본값은 1로 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\_out를 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 4번 사용하였는데, 이는 각 포켓몬은 최대 타입이 2개까지 있을 수 있기 때문에, 총 가능한 상성의 경우의 수는 4가지가 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))

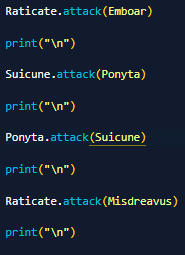
    other.lose\_health(dmg)

    print("Enemy Health: {}".format(other.health))

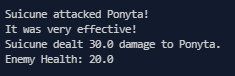
    print("")

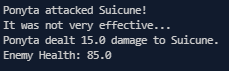
우선 공격을 하였다는 메시지를 띄운다. 그 다음 가중치에 따라 특수효과가 있다면 공격이 효과적이었는지, 별로였는지, 효과가 없는지를 출력한 뒤, 공격에 대한 정보를 출력한다. 그 다음에는 other.lose\_health() 를 이용하여 데미지의 크기만큼 상대 포켓몬의 체력을 깎는다. 마지막으로, 상대방의 체력을 표시한다.

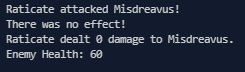
이렇게 해서 상대 포켓몬을 공격하는 method를 만들 수 있고, 이 결과는 다음과 같다.



 특수효과가 없는 일반 공격

 효과가 굉장한 공격

 효과가 별로인 공격

 효과가 없는 공격

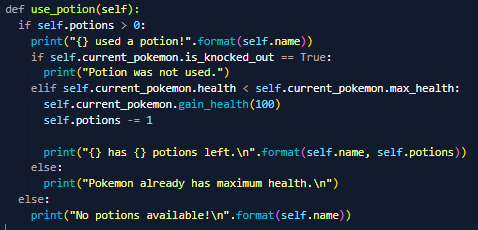
이를 이용하여, Trainer Class 내에서 상대 트레이너를 공격하는 method를 만들면 다음과 같다.

  def attack(self, other):

    self.current\_pokemon.attack(other.current\_pokemon)

상대 트레이너의 현재 포켓몬인 other.current\_pokemon을 받아, 자신의 현 포켓몬인 self.current\_pokemon과 함께 위의 pokemon 클래스의 attack 메소드를 사용한다.

Trainer Class 내에서 회복약인 Potion을 사용하는 method는 다음과 같다.



이 method는 우선 self.potions를 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\_out를 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)

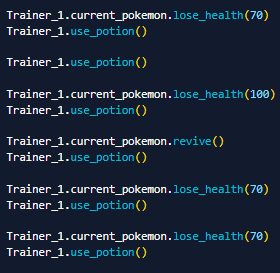
        self.potions -= 1

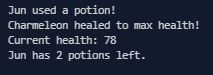
        print("{} has {} potions left.\n".format(self.name, self.potions))

      else:

        print("Pokemon already has maximum health.\n")

이후, 포켓몬의 체력이 최대 체력인지를 확인한다. 만약 포켓몬의 체력이 max\_health과 동일하다면, 체력이 최대라는 뜻이므로, 포션을 사용하지 않는다. 그 외의 경우에는 포켓몬의 체력을 100 회복한 뒤, 트레이너가 보유한 포션의 수를 1 만큼 감소한다. 이 method의 출력은 다음과 같다.



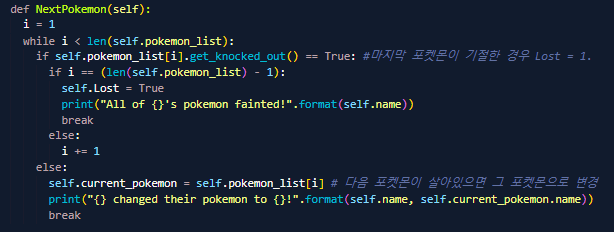
 일반적으로 포션을 사용한 경우

 포켓몬이 최대 체력인 경우

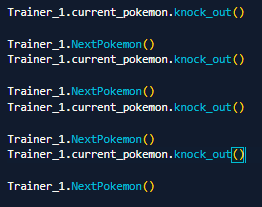
 포켓몬이 기절한 경우

 남은 포션이 없는 경우

위의 method외에도 성공적으로 포켓몬 배틀을 하기 위해서는 포켓몬을 교체하는 method가 필요하다. 이는 다음과 같다.



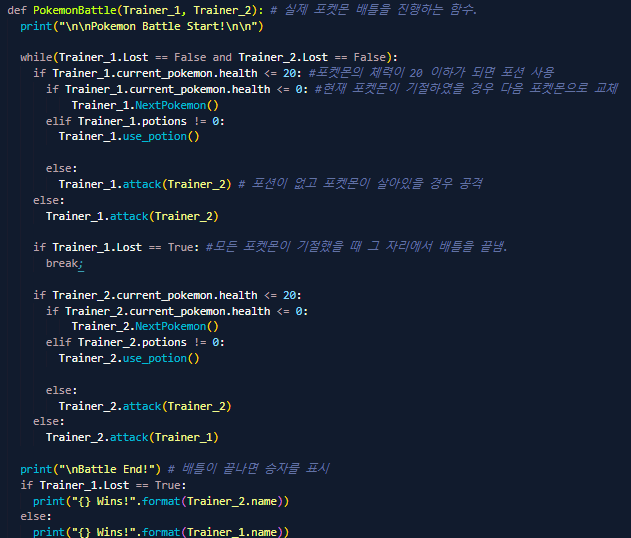
이 method는 트레이너의 보유 포켓몬을 순서대로 확인하여, 기절하지 않은 첫번째 포켓몬은 current\_pokemon으로 설정하여 내보낸다. 만약 모든 포켓몬이 기절하였을 경우, self.Lost를 True로 설정하여, 트레이너의 패배를 표시한다.



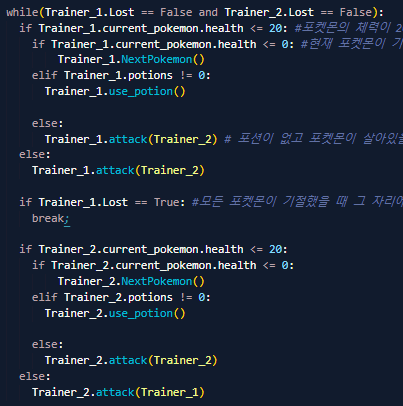
 포켓몬을 바꾸었을 경우

 모든 포켓몬이 기절한 경우

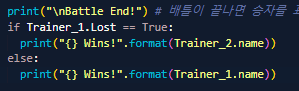
이를 이용하여 포켓몬 배틀을 진행하는 함수를 만들면 다음과 같다.



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



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 loop를 나온다.



배틀이 끝나면 숭자를 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!

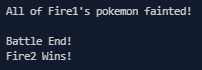
Battle End!

Subaru Wins!

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

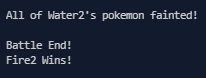
이 결과 역시 위의 경우처럼, 모든 출력을 적기에는 너무 길기 때문에, 결과만 작성하였고, 전체 배틀의 진행은 별도 파일로 첨부하였다.

1) 각 트레이너의 포션 수와 타입이 모두 같은 경우



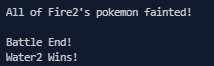
팀 구성이 더 강한 트레이너가 우승하였다.

2) 각 트레이너의 포션 수 및 타입이 모두 다른 경우



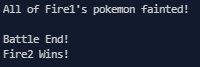
불리한 타입의 트레이너에게 더 많은 포션을 준 결과, 더 많은 포션을 가진 트레이너가 우승하였다.

3) 각 트레이너의 포션 수는 같고 타입은 다른 경우



유리한 타입의 트레이너가 우승하였다.

4) 각 트레이너의 포션 수는 다르고 타입은 같은 경우



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.