CS 106AX Programming Methodologies in JavaScript and Python Autumn 2026

IG Problem 5A: Reassemble Pokémon Evolutions

Written by Ben Yan

Part I: File Reading and Creation of a Pokedex Dictionary

```
def createPokedex(filename):
    """
    Given a file structured like the one above, returns a dictionary with each Pokémon species as keys,
    and the corresponding value as the species they evolve into, or None if the Pokémon doesn't evolve,
    e.g., the fully evolved Pokemon of Raichu, Blastoise, Ninetales, Lugia, etc.
    """
    pokedex = {}
    with open(filename, "r") as file:
        for line in file:
            line = line.strip()
            colon = line.find(":")

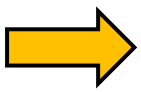
            if colon == -1: # Pokemon is fully evolved
                pokedex[line] = None

            else: # Pokemon evolves to another species
                pokemon = line[:colon]
                evolved = line[colon + 1:]
                pokedex[pokemon] = evolved

    return pokedex
```

Here's a sample input file on the left, and its corresponding **pokedex** dictionary ☺.

```
Squirtle:Wartotle
Vulpix:Ninetales
Wartotle:Blastoise
Blastoise
Pichu:Pikachu
Ninetales
Lugia
Raichu
Pikachu:Raichu
```



```
{
    "Squirtle": "Wartortle",
    "Vulpix": "Ninetales",
    "Wartortle": "Blastoise",
    "Blastoise": None,
    "Pichu": "Pikachu",
    "Ninetales": None,
    "Lugia": None,
    "Raichu": None,
    "Pikachu": "Raichu",
}
```

Part II: Reassembling Evolutionary Sequences

```
def reassembleEvolutions(pokedex):
    """
    Given a dictionary mapping Pokémon to the Pokémon they directly evolve into,
    returns a list of all distinct evolution sequences, start to end.
    """
    sequences = {} # key is final, fully evolved Pokémon in each sequence

    for pokemon in pokedex:
        # trace sequence of evolutionary descendants until fully evolved
        evolution = [pokemon]
        nextPokemon = pokedex[pokemon]
        while nextPokemon != None:
            evolution.append(nextPokemon)
            nextPokemon = pokedex[nextPokemon]

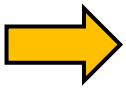
        # track the longest sequence evolving up to each final Pokémon
        finalPokemon = evolution[-1]
        if finalPokemon not in sequence:
            sequence[finalPokemon] = evolution

        # or update that Pokémon's sequence when a longer chain is found
        else if len(evolution) > len(sequence[finalPokemon])
            sequence[finalPokemon] = evolution

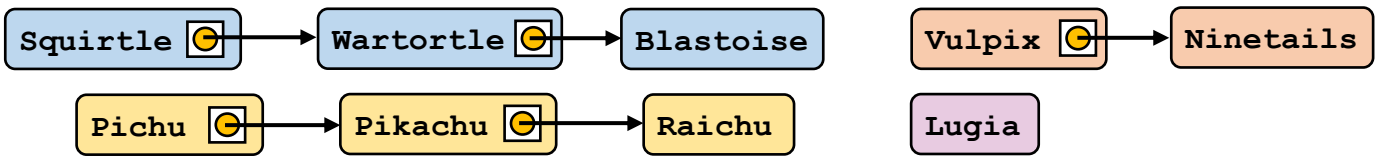
    # unpack the dictionary values (full evolution sequences) into a list
    evolutions = []
    for key in sequences: # each key is a final Pokemon
        evolutions.append(sequences[key])
    return evolutions
```

Here's a sample input **pokedex**, and its corresponding output list of evolutionary sequences ☺.

```
{
    "Squirtle": "Wartortle",
    "Vulpix": "Ninetales",
    "Wartortle": "Blastoise",
    "Blastoise": None,
    "Pichu": "Pikachu",
    "Ninetales": None,
    "Lugia": None,
    "Raichu": None,
    "Pikachu": "Raichu",
}
```



```
[["Squirtle", "Wartortle", "Blastoise"],
 ["Vulpix", "Ninetales"],
 ["Pichu", "Pikachu", "Raichu"],
 ["Lugia"]]
```



Here's an alternative implementation, which cleverly uses the fact that all evolutionary fragments end in the same final Pokemon—and so starts with each final Pokemon, working backwards until we reach a starting Pokemon, e.g., *Squirtle*. We then reverse each backwards sequence to get the original one!

```
def getEvolutionSequences(pokedex):
      """
      Given a dictionary mapping Pokémon to the Pokémon they directly evolve into,
      return a list of all distinct evolution sequences, start to end.
      """

      invertedPokedex = {} # maps each Pokemon to the previous one it evolves from
      finalForms = [] # list of all fully evolved Pokemon (i.e. evolves to no one)

      for pokemon in pokedex:
            evolved = pokedex[pokemon]
            if evolved == None: # pokemon is fully evolved
                  finalForms.append(pokemon)

            else: # reverses the key-value pair from original Pokemon-to-evolved map
                  invertedPokedex[evolved] = pokemon

   sequences = []
   for pokemon in finalForms:
       # for all fully evolved Pokemon, trace their evolutionary
       # sequence in reverse, until we reach the starting Pokemon
       inverseSequence = []
       while pokemon != None:
           inverseSequence.append(pokemon)
           pokemon = invertedPokedex.get(pokemon, None)

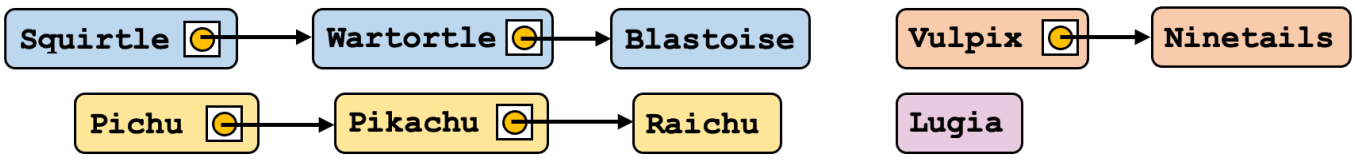
       # include this sequence, with the correct, un-reversed evolutionary ordering
       sequences.append(inverseSequence[::-1])

   return sequences
```

For the provided example, here's what the inverted **pokedex** and **finalForms** list would look like.

```
invertedPokedex = {
      "Wartortle": "Squirtle",
      "Ninetales": "Vulpix",
      "Blastoise": " Wartortle",
      "Pikachu": "Pichu",
      "Raichu": "Pikachu",
}
```

```
finalForms = [
      "Blastoise",
      "Ninetales",
      "Lugia",
      "Raichu",
]
```



🫶 IG Problem 3B: Ship Names Generator 🫶

Written by Ben Yan

```
""" This function takes two one-word names, and smashes them together into
    a fanfic-ready ship name, according to the vowel / consonant rules above.
"""
def shipThem(nameOne, nameTwo):
    oneFirstVowel = findFirstVowel(nameOne)
    twoFirstVowel = findFirstVowel(nameTwo)

    if (oneFirstVowel > 0): # starts with a consonant
        return nameOne[:oneFirstVowel] + nameTwo[twoFirstVowel:]

    else: # starts with a vowel
        oneSecondVowel = findFirstVowel(nameOne, 1);
        return nameOne[:oneSecondVowel] + nameTwo[twoFirstVowel + 1:]


""" Returns index of first vowel at or after the start position, and -1 if
    no vowel could be found. If the start is omitted, it's 0 by default.
"""
def findFirstVowel(word, start = 0):
    for index in range(start, len(word)):
        ch = word[index]
        if isEnglishVowel(ch):
            return index

    return index # no vowel found!


""" Returns true if the string is of length one, and a lowercase vowel
"""
def isEnglishVowel(ch):
    return len(ch) == 1 and "aeiou".find(ch) >= 0
```

Python Interpreter

```
>>> shipThem("scott", "jerry")
"scerry"
>>> shipThem("megumi", "yuji")
"muji"
>>> shipThem("wednesday", "enid")
"wenid"
>>> shipThem("armin", "eren")
"armren"
```

