Simple Word Game

Introduction

이 게임은 ‘Scrabble’이나 ‘Words With Friends’ 같은 게임과 비슷합니다 (보드를 이용하지 않는 더 단순한 버전입니다). 플레이어는 무작위로 주어진 글자들을 사용해 단어를 만들고, 만들어진 단어에 따라 점수를 받습니다.

게임 규칙은 아래와 같습니다:

Dealing

- 플레이어는 무작위로 선택된 글자 n개를 받습니다. (지금은 7개로 가정합니다)
- 각 글자는 최대 한 번씩만 사용할 수 있으며, 받은 글자(알파벳)들을 조합해서 여러 개의 단어를 만들 수 있습니다.
- 글자는 남아있을 수 있으며, 사용하지 않은 글자는 점수에 포함되지 않습니다.

Scoring

- 최종 점수는 만든 모든 단어의 점수를 더한 값입니다.
- 한 단어의 점수는 다음과 같이 계산됩니다:
  (해당 단어의 각 글자 점수 합계) × (단어의 길이)
  여기에 n개의 글자를 모두 사용하는 단어를 찾으면 추가로 50점을 보너스로 받습니다.
- 각 글자의 점수는 스크래블 게임과 동일하며, 예를 들어 A는 1점, B는 3점, C는 3점, D는 2점, E는 1점 등입니다. 각 소문자 글자에 점수를 매핑한 `LETTER_VALUES` 딕셔너리가 주어집니다.

계산 예시:

- `"weed"`라는 단어는 (w:4, e:1, e:1, d:2) → 4+1+1+2 = 8점, 여기에 길이 4를 곱해서 8×4 = **32**점
- n=7일 때 `"waybill"`을 찾으면, (w:4, a:1, y:4, b:3, i:1, l:1, l:1) → 총합 15, 곱하기 길이 7 = 105점, 여기에 50점 보너스를 더해 총 **155**점

Sample Output

게임은 다음과 같은 방식으로 진행됩니다:

```
Loading word list from file...
   83667 words loaded.
Enter n to deal a new hand, r to replay the last hand, or e to end game: r
You have not played a hand yet. Please play a new hand first!

Enter n to deal a new hand, r to replay the last hand, or e to end game: n
Current Hand: i e m u b c r
Enter word, or a "." to end this hand: rice
"rice" earned 24 points. Total: 24 points
Current Hand: m u b
Enter word, or a "." to end this hand: bum
"bum" earned 21 points. Total: 45 points
Run out of letters. Total score: 45 points.

Enter n to deal a new hand, r to replay the last hand, or e to end game: a
Invalid command.

Enter n to deal a new hand, r to replay the last hand, or e to end game: r
Current Hand: i e m u b c r
Enter word, or a "." to end this hand: cube
"cube" earned 32 points. Total: 32 points
Current Hand: i m r
Enter word, or a "." to end this hand: ir
Invalid word, please try again.
Current Hand: i m r
Enter word, or a "." to end this hand: .
Total score: 32 points.

Enter n to deal a new hand, r to replay the last hand, or e to end game: n
Current Hand: a i y b w l l
Enter word, or a "." to end this hand: birth
Invalid word, please try again.
Current Hand: a i y b w l l
Enter word, or a "." to end this hand: waybill
"waybill" earned 155 points. Total: 155 points
Run out of letters. Total score: 155 points

Enter n to deal a new hand, r to replay the last hand, or e to end game: e
```

실행 환경 확인하기

data file: words.txt

83,667개의 영단어를 포함하고 있으며, 각 단어는 줄바꿈 문자로 구분되어 있습니다.

구름 환경에서 파일은 현재 폴더를 기준으로 data 폴더 안에 있습니다. 코드창의 탭을 눌러 파일을 확인할 수 있습니다.

```
WORDLIST_FILENAME = “data/words.txt"
```

실행 확인

기본 제공 코드(skeleton code)가 제대로 실행되면 다음과 같은 출력이 보입니다:

```
Loading word list from file...
   83667 words loaded.
playGame not yet implemented.
```

---

helper code(도움 코드**)** 확인하기

기본 코드(skeleton code)에는 여러 함수가 이미 구현되어 있습니다.

```
# -----------------------------------
# Helper code
# 이미 구현된 코드이며, 독스트링을 읽어서 사용 방법을 알아두세요.
```

위 주석으로 감싸진 부분은 내부 동작을 이해할 필요는 없지만, 제공되는 함수들의 사용법(docstring)은 꼭 읽고 이해하세요.

단계별로 함수 작성하기

이 문제 세트는 여러 개의 모듈화된 함수를 작성한 뒤, 이를 조합하여 게임을 완성하도록 구성되어 있습니다.

전체 게임이 완성될 때까지 기다리지 말고**,** 함수 하나씩 작성한 뒤 바로 테스트해보세요**.**
이런 방식을 “단위 테스트(unit testing)”라고 하며, 코드 디버깅에 큰 도움이 됩니다.

문법 오류(syntax error)와 실행 오류(runtime error)를 먼저 디버깅하고, 테스트 케이스(프로그램이 예상대로 동작하는지 확인하기 위한 입력값과 그에 따른 기대 결과 세트)를 잘 설계해서 논리 오류(logic error)도 충분히 확인하세요.

Skeleton code:

```
# The Word Game

import random

VOWELS = 'aeiou'
ALPHABETS = 'abcdefghijklmnopqrstuvwxyz'
HAND_SIZE = 7

LETTER_VALUES = {
      'a': 1, 'b': 3, 'c': 3, 'd': 2, 'e': 1, 'f': 4, 'g': 2, 'h': 4, 'i':
1, 'j': 8, 'k': 5, 'l': 1, 'm': 3, 'n': 1, 'o': 1, 'p': 3, 'q': 10, 'r': 1,
's': 1, 't': 1, 'u': 1, 'v': 4, 'w': 4, 'x': 8, 'y': 4, 'z': 10
}

# -----------------------------------
# Helper code
# 이미 구현된 코드이며, 독스트링을 읽어서 사용 방법을 알아두세요.

WORDLIST_FILENAME = "data/words.txt"

def loadWords() -> list[str]:
      """
      유효한 단어 목록을 반환합니다. 단어는 소문자로 구성된 문자열입니다.
      이 함수가 완료되는 데 시간이 걸릴 수 있습니다.
      """
      print("Loading word list from file...")
      inFile = open(WORDLIST_FILENAME, 'r')
      wordList = []
      for line in inFile:
            wordList.append(line.strip().lower())
      print("  ", len(wordList), "words loaded.")
      return wordList


def getFrequencyDict(sequence: str) -> dict[str, int]:
      """
      키가 sequence의 각 문자이고 값이 문자가 반복되는 횟수를 나타내는 정수인 사전을
반환합니다.
      """
      freq = {}
      for x in sequence:
            freq[x] = freq.get(x,0) + 1
      return freq

# (end of helper code)
# -----------------------------------


#
```

```
# Problem #1: Word Scores
#
def getWordScore(word: str, n: int) -> int:
      """
      단어의 점수를 반환합니다. 해당 단어가 유효한 단어라고 가정합니다.
      [점수 규칙]에 따라 점수를 계산합니다.
      """
      # TO DO ...



# ------------------------------------
# Helper code
# 이미 구현된 코드이며, 독스트링을 읽어서 사용 방법을 알아두세요.

def displayHand(hand: dict[str, int]) -> None:
      """
      hand에 있는 글자를 출력합니다.

      Example:
      >>> displayHand({'a':1, 'x':2, 'l':3, 'e':1})
         a x x l l l e
      출력 순서는 중요하지 않습니다.

      """
      for letter in hand.keys():
            for j in range(hand[letter]):
                  print(letter,end=" ")
      print()


def dealHand(n: int) -> dict[str, int]:
      """
      n개의 소문자를 포함하는 임의의 hand(패)를 반환합니다.
      hand 중 최소 n/3(내림)개 문자는 모음입니다.

      hand는 딕셔너리로 표현됩니다.
      키(key)는 문자이고 값(value)은 해당 hand에 포함된 특정 문자의 개수입니다.
      """
      hand={}
      numVowels = n // 3

      for i in range(numVowels):
            x = VOWELS[random.randint(0, len(VOWELS)-1)]
            hand[x] = hand.get(x, 0) + 1

      for i in range(numVowels, n):
            x = ALPHABETS[random.randint(0, len(ALPHABETS)-1)]
            hand[x] = hand.get(x, 0) + 1

      return hand
```

```
# (end of helper code)
# -----------------------------------


#
# Problem #2: Dealing with Hands
#
def updateHand(hand: dict[str, int], word: str) -> dict[str, int]:
    """
    'hand'에 word의 모든 글자가 있다고 가정합니다.
    즉, 'word'에 글자가 몇 번 나타나든 'hand'에는 최소한 'word'에 있는 글자만큼의
글자가 있다고 가정합니다.

    hand를 업데이트합니다. 주어진 단어의 글자를 모두 사용한 상태의 새 딕셔너리를
반환합니다.

    주의: 매개변수 hand를 수정하지 않습니다.
    """
    # TO DO ...



#
# Problem #3: Valid Words
#
def isValidWord(word: str, hand: dict[str, int], wordList: list[str]) ->
bool:
    """
    word가 wordList에 있고, word가 hand에 있는 문자로 구성된 경우 True를
반환합니다.
    그렇지 않으면 False를 반환합니다.

    주의: hand 또는 wordList를 변경하지 않습니다.
    """
    # TO DO ...


#
# Problem #4: Playing a hand
#

def calculateHandLen(hand: dict[str, int]) -> int:
    """
    현재 hand(패)의 길이(문자 수)를 반환합니다.
    """
    # TO DO ...


#
# Problem #5: Playing a Hand
#
```

```
def playHand(hand: dict[str, int], wordList: list[str], n: int) -> None:
      """
      사용자가 다음과 같이 주어진 패(hand)를 플레이할 수 있도록 합니다.

      * 패(hand)가 표시됩니다.
      * 사용자는 단어나 마침표(문자열 ".")를 입력합니다.
      * 잘못된 단어를 입력하면 유효하지 않음이 표시되고, 다시 단어를 선택하라는 메시지가
표시됩니다.
      * 유효한 단어를 입력하면 패(hand)의 글자가 사용(소모)됩니다.
      * 유효한 단어가 입력되면 해당 단어의 점수와 총점수가 표시되고, 패에 남은 글자가
표시되며, 사용자에게 다시 단어를 입력하라고 요청합니다.
      * 사용되지 않은 글자가 더 이상 없거나 사용자가 "."을 입력하면 패가 종료됩니다.
      * 사용되지 않은 글자가 더 이상 없어서 종료된 경우 문자들을 모두 사용했다는 것을
표시합니다.
      * 패(hand)가 끝나면 단어 점수의 합계가 표시됩니다.

      출력할 메시지는 과제 안내문 2쪽의 Sample Output을 참고하세요.
      """
      # TO DO ...


#
# Problem #6: Playing a game
#

def playGame(wordList: list[str]) -> None:
      """
      사용자가 임의의 수의 패(hand)를 플레이하도록 허용합니다.

      1) 사용자에게 'n', 'r' 또는 'e'를 입력하도록 요청합니다.
            * 사용자가 'n'을 입력하면 새로운 (무작위) 패를 플레이합니다.
            * 사용자가 'r'을 입력하면 마지막 패를 다시 플레이합니다.
            * 사용자가 'e'를 입력하면 게임(프로그램)을 종료합니다.
            * 사용자가 다른 것을 입력하면 입력이 잘못되었다("Invalid command.")고
알립니다.

      2) 플레이를 마치면 1단계부터 반복합니다.
      """
      # TO DO ...
      print("playGame not yet implemented.") # <-- 함수 구현 후 지워주세요.



#
# Main Code
#
if __name__ == '__main__':
      random.seed(7)  # 재현 가능한 테스트를 위한 코드입니다. 지우지 마세요.
      wordList = loadWords()
      playGame(wordList)
```

Problem 1 - Word Scores

첫 번째 단계에서는 단어 하나의 점수를 계산하는 코드를 작성합니다. `getWordScore` 함수는 소문자로 이루어진 단어 (문자열)를 입력으로 받아, 게임의 점수 규칙에 따라 해당 단어의 정수 점수를 반환해야 합니다.

점수 규칙

- 각 단어의 점수를 모두 더한 것이 해당 hand의 총 점수입니다.
  *hand: 배분받은 글자(알파벳)들을 포함하는 패
- 단어 점수는 다음과 같이 계산됩니다:

  단어 점수 = (단어 안의 모든 글자 점수 합계) × (단어의 길이)
  n개의 글자를 한 번에 모두 사용한 경우, **50**점 보너스를 추가합니다. (n: 배분받은 글자 개수)
- 글자의 점수는 스크래블(Scrabble) 게임과 동일합니다. 예:
  - A: 1점, B: 3점, C: 3점, D: 2점, E: 1점, ...
  - 이 값들은 이미 `LETTER_VALUES`라는 딕셔너리에 정의되어 있습니다.

예시:

- `"weed"`는 (w:4, e:1, e:1, d:2) → 합계 8, 길이 4 → 8 × 4 = **32**점
- n=7이고 `"waybill"`을 만든 경우 → (w:4, a:1, y:4, b:3, i:1, l:1, l:1) = 15, 길이 7 → 15 × 7 = 105 + 50 보너스 = 총 **155**점

Hint

- 입력되는 단어는 항상 소문자로 구성된 문자열이거나 빈 문자열 `""`이라고 가정할 수 있습니다.
- `LETTER_VALUES` 딕셔너리를 반드시 사용해야 하며, 수정해서는 안 됩니다.
- 패(hand)에 들어오는 글자의 수 n은 항상 7이 아닙니다! n은 보너스 점수를 받을 수 있는 최대 글자 수를 의미합니다. 즉, `HAND_SIZE` 값을 바꾸어도 코드가 작동해야 합니다.
- 여러분이 알고 있는 영어 단어로 충분히 테스트해 보는 것이 좋습니다.

Problem 2 - Dealing with Hands

이 문제의 대부분은 코드를 읽고 이해하는 능력을 키우는 데 있습니다. 다른 사람이 작성한 코드를 읽는 것은 프로그래밍에서 매우 중요하고 유용한 능력입니다. 문제 마지막에는 간단한 함수를 하나 구현하게 됩니다. 얼핏 보면 쉬워 보일 수 있지만, 실제로는 꼼꼼한 읽기와 이해가 필요합니다.

---

hand(패**)**의 표현 방식

플레이어가 게임 중에 갖고 있는 글자들을 hand(패)라고 부릅니다.
예를 들어, 플레이어는 아래와 같은 패를 받을 수 있습니다:

```
a, q, l, m, u, i, l
```

우리 프로그램에서는 패를 다음과 같은 딕셔너리 형태로 표현합니다:

```
hand = {'a':1, 'q':1, 'l':2, 'm':1, 'u':1, 'i':1}
```

여기서 알 수 있듯이, 글자 `'l'`은 2번 등장하기 때문에 값이 2입니다.

- 딕셔너리에서 값을 얻는 기본 방법은 `hand['a']`입니다.
- 하지만 키가 존재하지 않을 경우 `KeyError`가 발생하므로 안전하게 접근하려면 `hand.get('a', 0)`을 사용하는 것이 좋습니다.

예:

```
>>> hand['e']
KeyError: 'e'

>>> hand.get('e', 0)
0
```

문자열을 딕셔너리로 바꾸기

`getFrequencyDict` 함수는 문자열을 입력받아, 글자의 등장 횟수를 딕셔너리 형태로 반환해줍니다.

예:

```
>>> getFrequencyDict("hello")
{'h': 1, 'e': 1, 'l': 2, 'o': 1}
```

패를 화면에 보여주기

패를 딕셔너리로 표현한 후, 이를 사용자에게 보기 좋게 출력하는 함수가 `displayHand`입니다.
이미 구현되어 있으니, 이 함수가 무엇을 하고 어떻게 동작하는지 꼭 읽고 이해하세요.

무작위 패**(hand)** 생성하기

`dealHand(n)` 함수는 `n`개의 소문자를 무작위로 선택하여 새로운 패(hand)를 생성합니다.
이 함수도 이미 구현되어 있으므로 반드시 코드를 읽고 어떻게 작동하는지 이해해보세요.

구현할 함수**: updateHand(hand: dict[str, int], word: str) -> dict[str, int]**

해당 단어(word)의 글자만큼 패(hand)에서 글자가 줄어듭니다.

예:

```
hand = {'a': 1, 'q': 1, 'l': 2, 'm': 1, 'u': 1, 'i': 1}
단어(word)가 'quail'이면 → {'a': 0, 'q': 0, 'l': 1, 'm': 1, 'u': 0, 'i': 0}
```

`updateHand(hand, word)` 함수를 구현하세요.

- Parameters:
    - `hand`: 현재 패 (딕셔너리)
    - `word`: 플레이어가 만든 단어 (문자열)
- Return:
    - `word`에 사용된 글자를 제외한 나머지 패(hand)를 새 딕셔너리로 반환
- 원래의 **hand**는 수정하면 안됩니다 (즉, side effect 없이 동작해야 합니다).

힌트

- 구현은 아주 짧고 간단합니다 (4줄 내외)
- 딕셔너리를 복사할 때 `.copy()` 메서드를 사용할 수 있습니다

예시 실행

```
>>> hand = {'a': 1, 'q': 1, 'l': 2, 'm': 1, 'u': 1, 'i': 1}
>>> updateHand(hand, 'quail')
{'a': 0, 'q': 0, 'l': 1, 'm': 1, 'u': 0, 'i': 0}
```

Problem 3 - Valid Words

지금까지 우리는 무작위 패(hand)를 생성하고, 사용자에게 해당 패를 보여주는 코드를 작성했습니다. 또한 사용자가 입력한 단어에 대해 점수를 계산할 수도 있습니다.

하지만 아직까지는 플레이어가 입력한 단어가 게임 규칙에 맞는지 검사하는 코드는 작성하지 않았습니다.

유효한 단어의 조건

1. 단어(word)가 단어 리스트(**`wordList`**)에 포함되어 있어야 합니다.
2. 단어(word)가 현재 패(**`hand`**)의 글자들만으로 구성되어야 합니다.

이 두 조건을 모두 만족할 때만 해당 단어는 유효한 단어로 간주됩니다.

구현할 함수: **`isValidWord(word, hand, wordList)`**

- Parameters:
  - `word`: 사용자가 입력한 단어 (문자열)
  - `hand`: 현재 패 (딕셔너리)
  - `wordList`: 모든 유효 단어가 담긴 리스트
- Return:
  - 유효한 단어면 `True`, 아니면 `False`
- 주의: `hand`와 `wordList`는 수정하면 안 됩니다.

Problem 4 - Hand Length

이제 본격적으로 사용자와 상호작용하는 코드를 작성할 준비가 되었습니다. 이 문제에서는 `playHand` 함수를 작성하기 위한 준비 단계로, 보조 함수인 `calculateHandLen`을 먼저 구현합니다.

이 함수는 매우 짧은 코드(5줄 이내)로 작성할 수 있습니다.

구현할 함수: `calculateHandLen(hand)`

- Parameter:
    - `hand`: 현재 패를 나타내는 딕셔너리 (`string -> int`)
- Return:
    - 패(hand)에 남아 있는 글자의 총 개수 (정수)

예시:

```
hand = {'a':1, 'q':1, 'l':2, 'm':1, 'u':1, 'i':1}
calculateHandLen(hand) → 7
```

여기서 'l'은 2번 등장하므로 총 7글자입니다.

Problem 5 - Playing a Hand

주의 사항

- 항상 패(hand)의 글자 수(n)가 7일 거라고 가정하지 마세요.
- HAND_SIZE를 변경해서 배분받은 글자 수를 변경할 수 있습니다.

테스트

코드를 구현한 후, 직접 게임을 실행해 보듯이 테스트해보세요.

예시(1):

```
Current Hand: a q w f f i p
Enter word, or a "." to end this hand: paw
"paw" earned 24 points. Total: 24 points
Current Hand: q f f i
Enter word, or a "." to end this hand: fi
Invalid word, please try again.
Current Hand: q f f i
Enter word, or a "." to end this hand: qi
"qi" earned 22 points. Total: 46 points
Current Hand: f f
Enter word, or a "." to end this hand: .
Total score: 46 points.
```

예시(2):

```
Current Hand: a r e t i i n
Enter word, or a "." to end this hand: inertia
"inertia" earned 99 points. Total: 99 points
Run out of letters. Total score: 99 points.
```

구현할 함수: **`playHand(hand, wordList, n)`**

- Parameters:
    1. `hand`: 현재 패를 딕셔너리로 표현

2. `wordList`: 유효한 단어 리스트
3. `n`: 보너스 점수 기준으로 사용할 패(hand)의 총 글자 수

- 동작:

1. 패(hand)를 출력한다.
2. 사용자는 단어 또는 `'.'`을 입력할 수 있다.
3. 유효하지 않은 단어는 거부되고 재입력을 받는다.
4. 유효한 단어면 점수를 계산하고 패(hand)를 업데이트한다.
5. 유효한 단어면 단어의 점수와 총점수, 남은 글자를 보여주고 다시 단어 입력
6. 모든 글자를 쓰거나 `'.'`를 입력하면 총 점수를 출력하고 종료 (단, 모든 글자를 쓴 경우 총 점수 출력 전에 "`Run out of letters. ` 라고 출력)

Problem 6 - Playing a Game

이제 전체 단어 게임 프로그램을 완성하기 위한 마지막 단계입니다.
여러 패(hand)를 플레이할 수 있도록 하는 `playGame` 함수를 구현하세요.

핵심 개념

- **HAND_SIZE**: 패(hand)에 배분되는 글자의 수. 이 값만 바꿔도 게임 난이도를 조절할 수 있습니다.
- 게임은 여러 패를 플레이할 수 있으며, 각 패는 무작위 또는 직전 패 중 선택할 수 있습니다.

함수 사양

`playGame(wordList)`는 사용자에게 다음 선택지를 줍니다:

안내 메시지: `Enter n to deal a new hand, r to replay the last hand, or e to end game:`

1. `'n'`: 새로운 패(hand)를 무작위로 생성하고 게임을 시작
2. `'r'`: 마지막 패(hand)로 다시 플레이
3. `'e'`: 게임 종료
4. 다른 입력: `"Invalid command."` 출력

   `'r'`을 선택하기 전에 플레이한 패가 없다면 `"You have not played a hand yet. Please play a new hand first!"`를 출력해야 합니다.

게임을 종료하지 않는 한 위 과정을 계속 반복합니다.

힌트

- `playHand()`와 `dealHand()` 함수만 호출하면 됩니다.
- 점수 출력 등은 모두 `playHand()`에 포함되어 있으므로, 이 함수는 비교적 간단합니다.
- 딕셔너리 형식으로 패(hand)를 저장하고, 반복해서 사용할 수 있어야 합니다.
