Завдання 3. Гра в слова

Це завдання стосуватиметься теми роботи зі словниками та іншими структурами даних, а також тестуванню у Python.

Про співпрацю:
Ви можете працювати над цим завданням разом з іншими студентами. Однак, кожен має написати і здати власну версію виконаного завдання. *Переконайтеся, що ви вказали з ким виконували це завдання у коментарях до вашого коду.*

Вступ

У цьому завданні ви реалізуєте наш власний варіант гри у слова!

Нехай вас не лякає об'єм цього завдання. Воно займає велику кількість тексту, але дуже просте.

Почнімо з того, що опишемо гру в слова: вона схожа на Скрабл. Гравцям роздають букви, з яких вони потім мають скласти одне або більше слів. Кожне **правильне** слово заробляє гравцю очки, на основі довжини слова та літер, з яких воно складається.

Правила гри описано далі. **Не починайте писати код зараз — як і у попередньому завданні ми розіб'ємо його на кроки нижче!**

Роздача

- Гравцю роздають руку з `HAND_SIZE` літер алфавіту, вибраних випадково. Це може включати декілька повторень однієї літери.
- Гравці складають літери з руки у стільки слів, скільки захочуть, але використовуючи кожну літеру лише один раз.
- Деякі літери можуть залишитися невикористані, хоча розмір руки, коли грається слово, впливає на його вартість.

Підрахунок очків

- Очки за роздачу є сумою очків за кожне сформоване слово.
- Очки за слово є **добутком** з двох компонент:
    - Перша компонента: сума очків за букви у слові.
    - Друга компонента: більша з двох величин — 1 чи `[7 * word_length - 3 * (n - word_length)]`, де:
        - `word_length` — це кількість букв, використаних у слові;
        - `n` — це кількість літер доступних у поточній руці.
- Букви оцінюються як у Скрабл: A коштує 1 очко, B коштує 3, C коштує 3, D коштує 2, E коштує 1 і т. д. Ми визначили словник `SCRABBLE_LETTER_VALUES`, що співставляє кожну маленьку літеру її вартості у Скрабл.

- Приклад:

Adapted from MIT OpenCourseWare 6.0001 Introduction to Computer Science and Programming in Python
Translation by Gasper
CC BY-NC-SA 4.0, for more information see https://ocw.mit.edu/terms

- Якщо *n=6* і рука складається з 1 “w”, 2 “e” і 1 “d” (та двох інших літер), зіграене слово “weed” буде коштувати 176 очків: (4+1+1+2) * (7*4 - 3*(6-4)) = 176. Перший множник є сумою вартостей кожної використаної літери; другий множник є результатом обчислення спеціального виразу, що заохочує гравця грати довгі слова та відбирає очки за літери, що залишилися у руці.
- У якості іншого прикладу, якщо *n=7*, зігране слово “it” буде коштувати 2 очки: (1+1) * (1) = 2. Другий множник дорівнює 1, бо 7*2 - 3*(7 - 2) = -1, що менше за 1.

Початок роботи

1. Завантажте та збережіть файли в одну папку. Це включає пайтон-файл `ps3.py`, який має містити увесь ваш майбутній код та провадить набір початкових функцій та шаблони нових. Також, це включає файл для тестуавння вашого коду `test_ps3.py`, і файл валідних слів `words.txt`. **Не змінюйте та не видаляйте нічого у файлах, якщо це не вказано.**
2. Запустіть `ps3.py` не роблячи жодних змін у ньому для того, аби перевірити, що все працює правильно. Цей код завантажує список валідних слів із файлу, а потім викликає функцію `play_game`. Ви реалізуєте функції, необхідні для її роботи. Якщо все добре, після невеликої затримки ви маєте побачити наступний вивід:

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

   Якщо ви натомість бачите `IOError` (наприклад, *No such file or directory*), перевірте чи зберегли ви файл `words.txt` у ту саму папку, що й `ps3.py`!

3. Файл `ps3.py` містить набір уже реалізованих функцій, які ви можете використовувати при написанні вашого вирішення. Ви можете ігнорувати код, що знаходиться між цими коментарями, але ви маєте прочитати та зрозуміти решту.

   ```
   #                    -------------------------------------
   #                       Helper code
   #  (you don't need to understand this helper code)
   .
   .
   .
   #          (end of helper code)
   # -------------------------------------
   ```

4. Це завдання структуровано таким чином, що ви напишете послідовність модулярних функцій, а потім склеїти їх докупи, щоб сформувати закінчену гру. Замість того, аби чекати доки гра готова, ви маєте протестувати кожну написану функцію окремо, перш ніж рухатися далі. Такий підхід називається юніт-тестуванням, і він допоможе вам відлагодити вашу програму.

Adapted from MIT OpenCourseWare 6.0001 Introduction to Computer Science and Programming in Python
Translation by Gasper
CC BY-NC-SA 4.0, for more information see https://ocw.mit.edu/terms

5. Ми додали підказки щодо того як ви можете захотіти реалізувати деякі із необхідних функцій у включених файлах. Вам не потрібно видаляти їх у вашій фінальній версії.

Для початку ми також підготували кілька тестових функцій. У ході вашої роботи над завданням, запускайте `test_ps3.py` аби перевіряти вашу роботу.

Якщо ваш код пройде юніт-тест ви побачите повідомлення SUCCESS; інакше ви побачите повідомлення FAILURE. **Ці тести не є вичерпними. Ви також можете захотіти протестувати ваш код іншими способами** (наприклад, з іншими тестовими значеннями).

Якщо ви запустите `test_ps3.py` використовуючи початкову версію `ps3.py`, ви маєте побачити, що усі тести провалюються.

Ось тестові функції, які ми надаємо:

`test_get_word_score`

Перевіряє реалізацію `get_word_score`.

`test_update_hand`

Перевіряє реалізацію `update_hand`.

`test_is_valid_word`

Перевіряє реалізацію `is_valid_word`.

`test_wildcard`

Перевіряє модифікації зроблені для підтримки вайлдкард. (Детальніше далі.)

Задача 1: Очки за слова

Першим кроком є реалізація функції, що підраховує очки за одне слово. Заповніть код функції `get_word_score` у файлі `ps3.py` відповідно до специфікації функції.

У якості нагадування, ось правила обрахунку вартості слова:

- Очки за слово є **добутком** з двох компонент:
  - Перша компонента: сума очків за букви у слові.
  - Друга компонента: більша з двох величин — 1 чи `[7 * word_length - 3 * (n - word_length)]`, де:
    - `word_length` — це кількість букв, використаних у слові;
    - `n` — це кількість літер доступних у поточній руці.

Ви маєте використовувати словник `SCRABBLE_LETTER_VALUES` визначений на початку файлу `ps3.py`. **Не** розраховуйте, що у руці завжди 7 букв! Параметр `n` — це загальна кількість букв у руці, коли було введено слово.

Нарешті, функція `str.lower` може бути корисною:

Adapted from MIT OpenCourseWare 6.0001 Introduction to Computer Science and Programming in Python
Translation by Gasper
CC BY-NC-SA 4.0, for more information see https://ocw.mit.edu/terms

```
s = “My string”
print(s.lower())
>>>> “my string”
```

Якщо ви не знаєте, що вона робить, ви можете набрати `help(str.lower)` у інтерактивній консолі Пайтона, щоб побачити документацію до неї.

Тестування: якщо цю функцію реалізовано вірно і ви запустите `test_ps3.py`, то тест `test_get_word_score()` пройде. Ви також маєте протестувати вашу реалізацію `get_word_score` самі, використовуючи адекватні англійські слова. Помітьте, що тест із вайлдкардами зупиниться із помилкою `KeyError`. На даний момент це нормально — ви виправите це у Задачі 4.

Задача 2: Карти в руки

Будь ласка, прочитайте задачу 2 повністю, перш ніж ми переходити до написання коду. Більшість описаних нижче функцій уже реалізовано.

Представлення руки

Рукою називається набір букв протягом гри. На початку гравці отримують набір випадкових літер. Наприклад, гравець може почати з наступною рукою: **a**, **q**, **l**, **m**, **u**, **i**, **l**. У нашій програмі рука буде представлена у вигляді словника: ключами будуть літери (у нижньому регістрі), а значеннями — кількість разів ця буква зустрічається у руці. Наприклад, рука із попереднього прикладу буде представлена як:

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

Зверніть увагу, як представлено повторювану літеру `‘l’`. У словниковому представленні, звичним способом звертатися до значення є `hand[‘a’]`, де `‘a’` є ключем, який ми хочемо знайти. Однак це буде працювати тільки якщо ключ уже є у словнику, інакше ми отримаємо `KeyError`. Аби уникнути цього, ми можемо натомість використовувати функцію `hand.get(‘a’, 0)`. Це “безпечний” спосіб отримувати значення, якщо ми не впевнені, що ключ є у словнику. `d.get(key, default)` повертає значення для `key`, якщо `key` є у словнику, і `default` інакше. Якщо `default` не задано, вона повертає `None`, тому цей метод ніколи не викличе `KeyError`.

Приведення слів до словникового вигляду

Однією із корисних функцій, які ми уже реалізували, є `get_frequency_dict`, визначена на початку `ps3.py`. Якщо передати їй рядок із літер, вона поверне словник, де ключами є літери із рядка, а значеннями — кількість разів ця літера зустрічається у рядку. Наприклад,

```
>> get_frequency_dict(“hello”)
{'h': 1, 'e': 1, 'l': 2, 'o': 1}
```

Adapted from MIT OpenCourseWare 6.0001 Introduction to Computer Science and Programming in Python
Translation by Gasper
CC BY-NC-SA 4.0, for more information see https://ocw.mit.edu/terms

Як бачите, це такий самий тип словника, що й той, який ми використовуємо для представлення руки.

Відображення руки

Маючи руку, представлену словником, ми хочемо відобразити її у вигляді, доступному для користувача. Ми реалізували це у функції `display_hand`. Витратьте зараз кілька хвилин аби ретельно прочитати її і зрозуміти, що вона робить і як вона працює.

Генерація випадкової руки

Рука, яку отримує гравець, складається із випадково вибраних літер. Ми реалізували функцію `deal_hand`, що генерує випадкову руку. Ця функція приймає в якості аргумента додатне ціле число `n`, і повертає новий словник, що представляє руку з `n` літер у нижньому регістрі. Знову, витратьте кілька хвилин, аби ретельно прочитати її та зрозуміти, що вона робить і як вона працює.

Видалення літер з руки (ви реалізуєте це!)

Гравець починає з повною рукою з `n` літер. Протягом того, як гравець називає слова, літери із набору стають використаними. Наприклад, гравець може стартувати із такою рукою: **a**, **q**, **l**, **m**, **u**, **i**, **l**. Гравець може зіграти слово **quail**. Після цього у його руці залишаться наступні літери: **l**, **m**.

Зараз ви маєте написати функцію, що приймає в якості аргументів руку і слово, використовує літери з руки, аби скласти це слово, та повертає **нову** руку, що містить лише літери, що залишилися. Ваша функція мусить **не** модифікувати вхідну руку. Наприклад:

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

(**ПРИМІТКА:** Як варіант, у прикладі вище після виклику `update_hand` значення змінної `new_hand` може бути `{'a':0, 'q':0, 'l':1, 'm':1, 'u':0, 'i':0}`. Конкретне значення залежить від вашої реалізації. Але вивід від `display_hand` має бути однаковий у будь-якому випадку.)

ВАЖЛИВО: Якщо гравець вводить слово, що є невірним, через те що це несправжнє слово чи через те, що у нього немає літер, аби скласти його, він або вона все одно втрачає літери, які вони використали, з руки в якості пенальті. Переконайтеся, що ваша

Adapted from MIT OpenCourseWare 6.0001 Introduction to Computer Science and Programming in Python
Translation by Gasper
CC BY-NC-SA 4.0, for more information see https://ocw.mit.edu/terms

реалізація враховує це! Не припускайте, що слова, які ви отримуєте, використовують лише літери, що дійсно існують в руці гравця. Наприклад:

```
>> hand = {'j':2, 'o':1, 'l':1, 'w':1, 'n':2}
>> display_hand(hand)
j j o l w n n
>> hand = update_hand(hand, 'jolly')
>> hand
{'j':1, 'w':1, 'n':2}
>> display_hand(hand)
j w n n
```

Помітьте, що були використані одна 'j', одна 'o' і одна 'l' (хоча гравець спробував використати дві, лише одна була у нього в руці). Літера 'y' не має жодного ефекту на руку, оскільки її там і так не було. Крім того, примітка вище про альтернативне представлення руки застосовна і до цього прикладу.

Реалізуйте функцію `update_hand` відповідно до специфікації у шаблоні коду.

ПІДКАЗКА: Ви можете захотіти перечитати документацію до методу `.copy` у словників.

Тестування: Переконайтеся, що тести `test_update_hand` виконуються без помилок. Ви також можете захотіти протестувати вашу реалізацію з деякими адекватними вхідними даними.

Задача 3: Правильні слова

До цього моменту ми не написали жодного коду, який би перевіряв, чи відповідають введені користувачем слова правилам гри. *Правильним* словом є слово, що міститься у списку слів (тут ми ігноруємо регістр слів) **і** складене повністю з літер, що є в поточній руці гравця.

Реалізуйте функцію `is_valid_word` відповідно до специфікації.

Тестування: Переконайтеся, що тести виконуються без помилок. Ви також маєте протестувати вашу реалізацію на деяких адекватних вхідних даних. Серед іншого, ви можете захотіти протестувати вашу реалізацію викликаючи її кілька разів на тій самій руці — якою має бути коректна поведінка у цьому випадку?

Задача 4: Вайлдкарди

Ми хочемо дозволити, аби руки гравців могли містити підстановочні літери (для простоти, в цьому документі ми називаємо їх англійським словом *wildcard* — вайлдкарди), які ми позначатимемо астериском (`*`). **Вайлдкарди можуть заміняти лише голосні.** Кожна початкова рука, яку ми генеруємо, мусить містити в точності один вайлдкард, як одну з літер. Гравець **не отримує** жодних очків за використання вайлдкард (на відміну від інших літер), але **рахується** за використану чи невикористану літеру при підрахунку очків.

Adapted from MIT OpenCourseWare 6.0001 Introduction to Computer Science and Programming in Python
Translation by Gasper
CC BY-NC-SA 4.0, for more information see https://ocw.mit.edu/terms

Протягом гри гравець, який хоче використати вайлдкард, має ввести “*” (без лапок) замість певної літери. Код, що перевіряє правильність слів, не має робити припущень, відносно того, якою мала бути замінена літера, але має перевіряти, що принаймні одне правильне слово може бути зроблено з вайлдкардом на місці голосної у бажаній позиції.

Приклади нижче показують як вайлдкарди мають поводитися в контексті розіграшу руки, який ви реалізуєте у Задачі 5 нижче. Не переймайтеся поки про цю частину — лише зверніть увагу на те, як обробляються вайлдкарди.

(Синім кольором тут і далі позначено ввід користувача, аби відділити його від виводу комп’ютера.)

Приклад №1: Правильне слово, складене без вайлдкард

```
Current Hand: c o w s * z
Enter word, or "!!" to indicate that you are finished: cows
"cows" earned 198 points. Total: 198 points

Current Hand: * z
Enter word, or "!!" to indicate that you are finished: !!
Total score: 198 points
```

Приклад №2: Правильне слово, складене з використанням вайлдкард

```
Current Hand: c o w s * z
Enter word, or "!!" to indicate that you are finished: c*ws
"cows" earned 176 points. Total: 176 points

Current Hand: o z
Enter word, or "!!" to indicate that you are finished: !!
Total score: 176 points
```

Приклад №3: Невірне слово з вайлдкардом

```
Current Hand: c o w s * z
Enter word, or "!!" to indicate that you are finished: c*wz
That is not a valid word. Please choose another word.

Current Hand: o s
Enter word, or "!!" to indicate that you are finished: !!
Total score: 0 points
```

Приклад №4: Інше невірне слово з вайлдкардом

```
Current Hand: c o w s * z
Enter word, or "!!" to indicate that you are finished: *ows
That is not a valid word. Please choose another word.

Current Hand: c z
Enter word, or "!!" to indicate that you are finished: !!
Total score: 0 points
```

Adapted from MIT OpenCourseWare 6.0001 Introduction to Computer Science and Programming in Python
Translation by Gasper
CC BY-NC-SA 4.0, for more information see https://ocw.mit.edu/terms

Модифікуйте функцію `deal_hand` аби вона завжди повертала один вайлдкард у кожній руці. Помітьте, що зараз `deal_hand` гарантує, що третина літер буде голосними, а решта — приголосними. Залиште кількість приголосних без змін, і замініть одну із потенційних голосних вайлдакрдом. Ви також матимете відредагувати одну чи більше констант, визначених на початку файлу, аби врахувати вайлдкарди.

Потім, модифікуйте функцію `is_valid_word` аби підтримувати вайлдкарди. **Підказка:** Перевірте, які слова можна скласти, замінюючи вайлдкард нан інші голосні. Ви можете захотіти переглянути документацію до функції `find` з модуля роботи з рядками і помітьте як вона реагує, коли символ не знайдено. Також може бути корисною константа `VOWELS` визначена на початку файлу.

Тестування: Переконайтеся, що тести `test_wildcards` виконуються без помилок. Ви також можете захотіти перевірити вашу реалізацію на деяких адекватних вхідних даних.

Задача 5: Розігрування руки

Тепер ми готові почати писати код, який взаємодіє з гравцем.

Реалізуйте функцію `play_hand`. Ця функція дозволяє гравцю зіграти одну руку. Перед цим вам необхідно буде реалізувати допоміжну функцію `calculate_handlen`, що може бути зроблено у менш ніж 5 рядків коду.

Аби закінчити руку достроково, гравець **має** ввести "`!!`" (два знаки оклику).

Помітьте, що після рядка `# BEGIN PSEUDOCODE` знаходиться багато, власне, псевдокоду. Це зроблено для того, аби допомогти вам у написанні вашої функції. Перегляньте Чому псевдокод (англ.), аби дізнатися більше про те, навіщо і як використовувати псевдокод, перш ніж ви почнете розбиратися з цією проблемою.

Тестування: Перевірте вашу реалізацію так, ніби ви граєте у гру: запустіть вашу програму і викличте функцію `play_hand` з вашого командного рядка з аргументами руки і `word_list`.

Примітка: Ваше виведення **має** відповідати прикладам нижче. **Ви не маєте виводити зайві повідомлення "None".**

Приклад №1

```
Current Hand: a j e f * r x
Enter word, or "!!" to indicate that you are finished: jar
"jar" earned 90 points. Total: 90 points

Current Hand: * f x e
Enter word, or "!!" to indicate that you are finished: f*x
"f*x" earned 216 points. Total: 306 points
```

Adapted from MIT OpenCourseWare 6.0001 Introduction to Computer Science and Programming in Python
Translation by Gasper
CC BY-NC-SA 4.0, for more information see https://ocw.mit.edu/terms

```
Current                              Hand:                                 e
Enter word, or "!!" to indicate that you are finished: !!
Total score: 306 points
```

Приклад №2

```
Current Hand: a c f i * t x
Enter word, or "!!" to indicate that you are finished: fix
"fix"     earned     117     points.     Total:     117     points

Current          Hand:          a          c          t          *
Enter word, or "!!" to indicate that you are finished: ac
This is not a valid word. Please choose another word.

Current                  Hand:                  t                  *
Enter word, or "!!" to indicate that you are finished: *t
"*t"     earned     14     points.     Total:     131     points

Ran out of letters. Total score: 131 points
```

Задача 6: Ігровий процес

Гра складається з розіграшу багатьох рук. Аби завершити нашу гру в слова, ми маємо реалізувати дві останні функції.

Реалізуйте функції `substitute_hand` і `play_game` відповідно до їх специфікації. Для гри ви маєте використовувати константу `HAND_SIZE`, щоб визначити кількість літер у руці.

Не припускайте, що у руці завжди буде 7 літер! Наша мета: зберегти модулярність коду — якщо ви хочете грати гру в слова з 10 літерами або з 4 літерами ви можете це зробити просто змінивши значення `HAND_SIZE`.

Реалізуючи підстановку, ви можете захотіти прочитати про методи, пов'язані зі словниками, такі як `.keys`, або ключове слово `del`. Ви також можете захотіти переглянути код `deal_hand`, аби побачити як `random.choice` може бути використаний аби вибрати випадковий елемент із деякої множини (наприклад, рядка).

Помітьте, що ми не наводимо псевдокод для цієї задачі. Тим не менш, в процесі того, як ви як вирішити ці функції, ви можете написати ваш власний.

Тестування: Перевірте вашу реалізацію так, ніби ви граєте у гру. Спробуйте різні значення `HAND_SIZE` у вашій програмі, і переконайтеся, що ви можете грати у слова з різними розмірами руки модифікуючи тільки змінну `HAND_SIZE`.

Приклад

```
Enter total number of hands: 2
Current hand: a c i * p r t
Would you like to substitute a letter? no
```

Adapted from MIT OpenCourseWare 6.0001 Introduction to Computer Science and Programming in Python
Translation by Gasper
CC BY-NC-SA 4.0, for more information see https://ocw.mit.edu/terms

```
Current hand: a c i * p r t
Enter word, or "!!" to indicate that you are finished: part
"part" earned 114 points. Total: 114 points

Current hand: c i *
Enter word, or "!!" to indicate that you are finished: ic*
"ic*" earned 84 point. Total: 198 points

Ran out of letters
Total score for this hand: 198
--------
Would you like to replay the hand? no
Current hand: d d * l o u t

Would you like to substitute a letter? yes
Which letter would you like to replace: l

Current hand: d d * a o u t
Enter word, or "!!" to indicate that you are finished: out
"out" earned 27 points. Total: 27 points

Current hand: d d * a
Enter word, or "!!" to indicate that you are finished: !!
Total score for this hand: 27
--------
Would you like to replay the hand? yes
Current hand: d d * a o u t
Enter word, or "!!" to indicate that you are done: d*d
"d*d" earned 36 points. Total: 36 popointsints

Current hand: a o u t
Enter word, or "!!" to indicate that you are finished: out
"out" earned 54 points. Total: 90 points

Current hand: a
Enter word, or "!!" to indicate that you are finished: !!
Total score for this hand: 90
--------
Total score over all hands: 288
```

На цьому це завдання завершено!

Adapted from MIT OpenCourseWare 6.0001 Introduction to Computer Science and Programming in Python
Translation by Gasper
CC BY-NC-SA 4.0, for more information see https://ocw.mit.edu/terms