문제집 4

배포일: 2016년 10월 5일

제출 마감: 2016년 10월 12일 오후 11시 59분

이 문제집은 두 부분으로 구성됩니다. 첫 번째 부분은 문제를 재귀적인 방식으로 생각하는 연습을 할 수 있도록 하며, 문제를 동일한 문제의 더 간단한 형태로 환원할 수 있다는 아이디어를 활용합니다. `ps4a.py` 파일에서 문자열을 입력으로 받아 해당 문자열 내 모든 가능한 문자 재배열을 계산하는 재귀 함수를 작성하게 됩니다.

두 번째 부분에서는 각 인스턴스가 특정 속성과 이를 조작하는 메서드를 포함하는 클래스 관점에서 문제를 사고하는 경험을 제공합니다. `ps4b.py`에서는 객체 지향 프로그래밍을 사용하여 시저/시프트 암호(Caesar/shift cipher)를 작성하게 됩니다. `ps4c.py`에서는 객체 지향 프로그래밍을 사용하여 매우 간단한 치환 암호(substitution cipher)를 작성하게 됩니다.

항상 그렇듯이, 제공된 파일의 이름을 변경하거나, 제공된 헬퍼 함수를 수정하거나, 함수/메서드 이름을 변경하거나, 제공된 문서 문자열(docstring)을 삭제하지 마십시오. `words.txt`와 `story.txt` 파일은 `ps4a.py`, `ps4b.py`, `ps4c.py` 파일을 저장하는 폴더와 동일한 위치에 보관해야 합니다.

마지막으로, 스타일 가이드를 참고하시기 바랍니다. 위반 사항(예: 설명이 부족한 변수명, 주석 처리되지 않은 코드)에 대해서는 감점을 적용할 예정입니다. 본 과제 스타일 가이드 중 6, 7, 8번 항목이 특히 관련성이 높으니 시작
(예: 설명이 부족한 변수명, 주석 처리되지 않은 코드 등)에 대해 감점을 적용할 예정입니다. 본 과제 스타일 가이드 중 6, 7, 8번 항목이 특히 중요하니, 과제를 시작하기 전과 제출 전에 반드시 확인하시기 바랍니다!

Part A: 문자열의 순열

순열은 단순히 재배열을 가리키는 이름입니다. 따라서 문자열 'abc'의 순열은 'abc', 'acb', 'bac', 'bca', 'cab', 'cba'입니다. 주목할 점은 순열 자체가 자신의 순열(자명 순열)이라는 사실입니다. 이 과제 부분에서는 문자열을 입력으로 받아 그 모든 순열의 목록을 반환하는 **재귀** 함수 `get_permutations`를 작성해야 합니다. 이 함수는 과제 C 부분에서 유용하게 사용될 것입니다.

요구사항에 대한 몇 가지 참고 사항: **재귀를 반드시 사용해야** 하며, 전역 변수는 사용할 수 없습니다. 또한 루프를 사용하여 솔루션을 구현해도 무방합니다. 반환되는 순열의 순서는 중요하지 않습니다. 최종 리스트에 중복을 포함하지 않도록 주의하십시오.

권장 접근법

재귀 문제를 해결하려면 최소한 하나의 기본 사례와 재귀 사례(들)이 있어야 합니다. 기본 사례는 이 문제에 대한 가장 단순한 입력으로 생각할 수 있습니다(해결이 명백하고 재귀가 필요하지 않은 경우). 이 접근법에서 기본 사례는 시퀀스가 단일 문자일 때입니다(단일 문자를 순서대로 배열하는 방법은 단 하나뿐입니다).

시퀀스가 한 문자보다 길다면, 시퀀스의 모든 순열을 쉽게 찾을 수 있도록 도와줄 더 간단한 문제 형태를 찾아야 합니다. 아래 의사 코드는 이 문제를 재귀적으로 해결하는 한 가지 접근법을 제시합니다.

입력 문자열 시퀀스 가 주어졌을 때:

- 기본 사례:
    - 시퀀스가 단일 문자일 경우, 순서를 정할 수 있는 방법은 단 하나뿐입니다
        - 시퀀스를 포함하는 단일 요소 리스트를 반환합니다
- 재귀적 경우:
    - 순서대로 나열된 문자열에서 첫 번째 문자를 제외한 모든 문자의 순열 목록을 반환하는 메서드가 있다고 가정합시다 (힌트: 재귀를 생각해보세요)
    - 그러면 문자열의 모든 문자의 순열은 첫 번째 문자를 나머지 문자 순열 각각에 삽입할 수 있는 **모든 다른 방법들이** 됩니다
        - *예시*: 단어 'bust'에서 'b'를 제외한 나머지 글자의 순열 목록은 ['ust', 'sut', 'stu', 'uts', 'tus', 'tsu']입니다.
            - 그 다음 'ust'는 다음과 같은 조합을 줍니다: 'bust', 'ubst', 'usbt', 'ustb'
            - ‘sut’는 다음과 같은 변형을 생성합니다: ‘bsut’, ‘sbut’, ‘subt’, ‘sutb’
            - 등등 ...

`ps4a.py`에 있는 `get_permutations(sequence)` 함수를 docstring의 사양에 따라 구현하세요. 함수에 대한 세 가지 테스트 케이스를 `if __name__ == '__main__'` 아래에 함수에 대한 세 가지 테스트 케이스를 작성하세요. 각 테스트 케이스는 입력, 예상 출력, 실제 출력을 표시해야 합니다. `if __name__ == '__main__'`을 참조하십시오.

파트 B: 카이사르 암호 방식

젊은 시절의 영광을 되살리고 친구들에게 비밀 메시지를 전하고 싶었던 적이 있나요? 자, 지금이 바로 그 기회입니다! 하지만 먼저 몇 가지 용어를 알아보겠습니다:

- 암호화 - 메시지를 알아볼 수 없도록 흐리거나 인코딩하는 과정
- 복호화 - 암호화된 메시지를 해독하여 다시 읽을 수 있게 만드는 과정
- 암호 - 암호화와 복호화를 수행하는 알고리즘
- 명문 - 원래의 메시지
- 암호문 - 암호화된 메시지. 참고: 암호문은 비록 난해해 보일지라도 여전히 원본 메시지의 모든 정보를 포함하고 있음.

카이사르 암호

시저 암호는 정수를 선택하여 메시지의 모든 문자를 그 정수만큼 이동시키는 방식입니다. 즉, 이동량 k를 가정하면, 평문에서 나타나는 알파벳의 i번째문자는 모두 암호문에서 알파벳의 $(i + k)$번째문자로 변환됩니다. $i + k > 26$(알파벳 길이)인 경우에 주의해야 합니다.

대문자와 소문자를 개별적으로 처리하여 대문자는 항상 대문자로, 소문자는 항상 소문자로 매핑합니다. 대문자가 "A"로 매핑되면 동일한 소문자는 "a"로 매핑되어야 합니다. 구두점과 공백은 변경 없이 유지해야 합니다. 예를 들어, 쉼표가 포함된 평문 메시지는 동일한 위치에 쉼표가 있는 암호문을 가져야 합니다.

예시:

평문	shift	암호문
‘abcdef’	2	‘cdefgh’
‘Hello, World!’	4	‘립스, 아스프!’
‘’	어떤 값	‘’

클래스와 상속

이것은 클래스로 코딩하는 첫 경험입니다! 기대하세요! Message 클래스와 그 하위 클래스인 `PlaintextMessage` 및 `CiphertextMessage`를 만들 것입니다. `Message`에는 문자열에 암호화를 적용하는 데 사용할 수 있는 메서드들이 포함되어 있습니다.

메시지를 암호화하거나 복호화합니다(시저 암호에서는 동일한 작업입니다). `PlaintextMessage`는 지정된 시프트 값을 사용하여 문자열을 인코딩하는 메서드를 제공합니다. 본 클래스는 항상 메시지의 인코딩된 버전을 생성하며, 인코딩 방식을 변경하는 메서드를 갖습니다. `CiphertextMessage`는 문자열을 디코딩하는 데 사용되는 메서드를 포함합니다.

구현을 완료한 후에는, 누군가가 제공한 암호화된 문자열을 사용하여 `CiphertextMessage` 인스턴스를 생성하고 이를 복호화해 볼 수 있습니다. 또는 자신의 `PlaintextMessage` 인스턴스를 암호화한 후, `PlaintextMessage` 인스턴스 내의 암호화된 메시지에서 `CiphertextMessage` 인스턴스를 생성하고 이를 복호화하여 원본 평문 메시지와 일치하는지 확인할 수 있습니다.

이 작업은 `ps4b.py` 문서 문자열에 명시된 사양에 따라 이 세 클래스 모두에 대한 메서드를 구현하는 것입니다. 클래스 외부에서 속성에 직접 접근해서는 안 된다는 점을 꼭 기억하세요. 바로 그 때문에 게터와 세터 메서드가 존재합니다. 너무 복잡하게 생각하지 마세요. 게터 메서드는 단순히 속성을 반환해야 하며, 세터 메서드는 전달된 인자와 동일한 값으로 속성을 설정해야 합니다. 간단해 보이지만, 이러한 메서드들은 우리가 조작해서는 안 되는 속성을 다루지 않도록 보장하기 위해 필요합니다. 클래스 자체 외부에서 getter와 setter를 사용하지 않고 직접 클래스 속성을 사용하면 점수가 감점될 뿐만 아니라, 더 중요한 것은 객체 클래스 계층 구조를 설계하고 구현할 때 골치 아픈 문제를 일으킬 수 있습니다.

규칙

파트 B에서는 재귀를 사용하지 않아도 되지만, 사용해도 좋습니다. 여러분을 위해 구현한 몇 가지 보조 함수(`load_words` 및 `is_word`)가 있습니다. 이 함수들을 해결책에 사용할 수 있으며 완전히 이해할 필요는 없지만, 관련 주석은 읽어야 합니다. 파일의 나머지 부분에 있는 보조 코드를 읽고 이해하여 해결책의 지침으로 활용해야 합니다.

Part 1: Message

> `ps4b.py` 파일에 있는 Message 클래스의 메서드를 docstring에 명시된 사양에 따라 완성하세요.

Message 클래스에는 다음 함수들에 대한 스켈레톤 코드가 제공되어 있습니다. 여러분의 임무는 이를 구현하는 것입니다. 각 함수의 사양에 대한 자세한 내용은 해당 함수의 docstring 주석을 참조하십시오.

- `__init__(self, text)`
- 게터 메서드

- `get_message_text(self)`
    - 참고: init에서 이 객체에 추가한 메시지 텍스트의 불변 버전을 반환해야 합니다. 다행히 문자열은 이미 불변 객체이므로 해당 문자열을 그대로 반환하면 됩니다.
- `get_valid_words(self)`
    - 참고: 원본 목록이 실수로 변경되는 것을 방지하기 위해 self.valid_words의 복사본을 반환해야 합니다.

- `build_shift_dict(self, shift)`
    - 참고: 여기서 문자열 모듈의 `ascii_lowercase` 상수가 유용할 수 있습니다.
- `apply_shift(self, shift)`
    - 힌트: `build_shift_dict(self, shift)`를 사용하세요. 공백과 구두점은 암호화 과정에서 변경되지 않아야 합니다.

파트 2: PlaintextMessage

`ps4b.py`에 있는 PlaintextMessage 클래스의 메서드를
문서 문자열의 사양에 따라 메서드를 작성하세요.

작성해야 할 메서드는 다음과 같습니다:

- `___init____(self, text, shift)`
    - 부모 클래스의 생성자를 사용하여 코드를 간결하게 작성하세요. 혼란스러우면 스타일 가이드 #7을 참고하세요.
- getter 메서드
    - `get_shift(self)`
    - `get_encryption_dict(self)`
        - 참고: 원본 사전이 변경되는 것을 방지하기 위해 self.encryption_dict의 복사본을 반환해야 합니다.
    - `get_message_text_encrypted(self)`
- `change_shift(self, shift)`
    - 힌트: 이 작업을 더 쉽게 할 수 있는 다른 메서드가 무엇인지 생각해 보세요. 코드 몇 줄 이상 걸리지 않아야 합니다.

파트 3: 암호문 메시지

친구들이 무슨 말을 하는지 알고 싶지 않나요? 암호화된 메시지가 주어졌을 때, 메시지를 인코딩하는 데 사용된 시프트를 안다면 디코딩은 간단합니다. 메시지가 암호화된 메시지이고 `s`가 메시지를 암호화하는 데 사용된 시프트라면, `apply_shift(message, 26-s)`를 적용하면 원래의 평문 메시지를 얻을 수 있습니다. 왜 그런지 아시겠나요?

문제는 물론 시프트 값을 모른다는 점입니다. 하지만 우리의 암호화 방식은

이 메서드는 시프트 값으로 26가지의 서로 다른 값만 가질 수 있습니다! 이 이메일들의 주요 언어는 영어임을 알고 있으므로, 각 시프트 값을 시도하며 복호화된 메시지 내 영어 단어 수를 최대화하는 프로그램을 작성할 수 있다면, 그들의 암호를 해독할 수 있습니다!

`ps4b.py` 파일 내 CiphertextMessage 클래스의 메서드들을
문서 문자열의 사양에 따라 작성하세요.

작성해야 할 메서드는 다음과 같습니다:

- `__init__(self, text)`
  - 힌트: 코드를 간결하게 작성하려면 부모 클래스의 생성자를 사용하세요. 혼란스러우면 스타일 가이드 #7을 참고하세요.
- `decrypt_message(self)`
  - 힌트: 헬퍼 함수 `is_word(wordlist, word)`와 문자열 메서드 `split`이 유용할 수 있습니다(https://docs.python.org/3/library/stdtypes.html#str.split 또는 콘솔에서 `help(str.split)` 참조).
  - 참고: `is_word`는 단어 유효성 판단 시 구두점 및 기타 특수 문자를 무시합니다.

파트 4: 테스트

`if` 문 아래에 `PlaintextMessage`에 대한 테스트 케이스 두 개와 `CiphertextMessage`에 대한 테스트 케이스 두 개를 주석으로 작성하십시오. `__name__ == '__main__'` 아래에 주석으로 작성하세요. 각 테스트 케이스에는 입력값, 예상 출력값, 실제 출력값을 표시해야 합니다. 예시는 `ps4b.py`에서 확인할 수 있습니다. 그런 다음 story.txt 파일을 해독하고, 해독에 사용된 최적의 시프트 값과 복호화된 이야기를 테스트 케이스 아래 주석으로 작성하세요.

힌트: 스켈레톤 코드에는 `get_story_string`이라는 헬퍼 함수가 포함되어 있으며, 이 함수는 암호화된 스토리의 문자열 버전을 반환합니다. 스토리 문자열을 사용하여 `CiphertextMessage` 객체를 생성하고 `decrypt_message`를 사용하여 적절한 시프트 값과 복호화된 스토리를 반환하세요.

파트 C: 치환 암호

메시지를 숨기는 더 나은 방법은 치환 암호를 사용하는 것입니다. 이 방법에서는 숨겨진 암호 체계를 만들어 각 원본 문자를 무작위로 선택한 문자로 치환합니다. 예를 들어, 문자 "a"에 대해서는 다른 26개 문자 중 아무거나(원래 "a"를 유지하는 것도 포함)를 선택할 수 있고, 문자 "b"에 대해서는 남은 25개 문자 중 (원래 "a"에 선택한 문자를 제외한) 아무거나를 선택할 수 있으며, 이와 같은 방식으로 진행됩니다. 대체 암호를 해독하려면 시저 암호보다 훨씬 많은 경우의 수를 시험해야 한다는 점을 알 수 있을 것입니다(26! 대 26). 따라서 이 문제에서는 모음을 암호화하는 치환암호만 고려하겠습니다. 이때 소문자와 대문자는 각각 대응되는 문자로 매핑됩니다. (예: 'A' -> 'O', 그다음 'A' -> 'O'...)
'a'->'o').

클래스와 상속

카이사르 암호와 유사하게, 이 개념을 탐구하기 위해 클래스를 사용할 것입니다. 이러한 종류의 치환 메시지를 처리하기 위한 일반적인 기능을 가진 `SubMessage` 클래스를 만들 것입니다. 또한 더 구체적인 구현과 사양을 가진 `EncryptedSubMessage` 클래스를 작성할 것이며, 이 클래스는 SubMessage 클래스에서 상속받습니다.

`ps4c.py` 문서 문자열에 명시된 사양에 따라 두 클래스의 메서드를 구현하는 것이 여러분의 작업입니다. 클래스 외부에서 속성에 직접 접근해서는 안 된다는 점을 꼭 기억하세요. 이를 위해 게터와 세터 메서드가 존재합니다. 다시 말하지만, 너무 복잡하게 생각하지 마세요. get 메서드는 단순히 변수를 반환하고, set 메서드는 전달된 매개변수와 동일한 값으로 속성을 설정하기만 하면 됩니다. 간단해 보이지만, 이러한 메서드들은 우리가 조작해서는 안 되는 속성을 다루지 않도록 보장하기 위해 필요합니다. 클래스 자체 외부에서 getter와 setter를 사용하지 않고 직접 클래스 속성을 사용하면 점수가 감점될 뿐만 아니라, 더 중요한 것은 객체 클래스 계층 구조를 설계하고 구현할 때 골치 아픈 문제를 일으킬 수 있습니다.

파트 1: 서브메시지

> `ps4c.py` 파일에 있는 SubMessage 클래스의 메서드를 docstring에 명시된 사양에 따라 구현하십시오.

SubMessage 클래스의 다음 메서드에 대한 스켈레톤 코드를 제공했습니다.
- 여러분의 임무는 이들을 구현하는 것입니다. 함수 사양에 대한 자세한 내용은 각 함수의 docstring 주석을 참조하십시오.

- `__init__(self, text)`
- 게터 메서드
    - `get_message_text(self)`
    - `get_valid_words(self)`
        - 참고: 원본 목록이 변경되는 것을 방지하기 위해 self.valid_words의 복사본을 반환해야 합니다.
- `build_transpose_dict(self, vowels_permutation)`
- `apply_transpose(self, transpose_dict)`
        - 참고: 테스트 시 입력 사양에 세심한 주의를 기울이십시오.

파트 2: EncryptedSubMessage

`ps4c.py`에 있는 EncryptedSubMessage 클래스의 메서드를
문서 문자열의 사양에 따라 메서드를 작성하십시오.

친구들이 무슨 말을 하는지 알고 싶지 않나요? 암호화된 메시지가 주어졌을 때, 그 메시지를 인코딩하는 데 사용된 치환법을 안다면 디코딩은 아주 간단합니다. 각 문자를 올바른 디코딩된 문자로 바꾸기만 하면 되니까요.

문제는 물론, 대치 규칙을 모른다는 점입니다. 모든 자음을 동일하게 유지하더라도 모음에 대한 다양한 대치 시도를 할 수 있는 선택지가 여전히 많습니다. 시저 암호와 마찬가지로, 제안된 대치 규칙을 테스트하여 해독된 단어가 실제 영어 단어인지 확인하는 방법을 사용할 수 있습니다. 그런 다음 가장 많은 유효한 영어 단어를 생성하는 복호화 방식을 선택합니다. 참고로 대문자 모음, 소문자 모음, 대문자 자음, 소문자 자음의 값을 포함한 상수들을 편의를 위해 제공했습니다.

이 부분에서는 파트 A의 `get_permutations` 메서드를 사용할 수 있습니다(ps4c.py 시작 부분에서 이미 임포트되어 있습니다).

작성해야 할 메서드는 다음과 같습니다:

- `__init__(self, text)`
    - 힌트: 코드를 간결하게 작성하려면 부모 클래스의 생성자를 활용하세요. 혼란스러우면 스타일 가이드 #7을 참고하세요.
- `decrypt_message(self)`

파트 3: 테스트

`SubMessage`에 대한 테스트 케이스 두 개와

if 아래 주석의 EncryptedSubMessage _____name___ == '___main___'
아래 주석에 EncryptedSubMessage를 작성하십시오. 각 테스트 케이스에는 입력값, 예상 출력값, 사용된 함수 호출, 실제 출력값이 포함되어야 합니다.

6.0001 컴퓨터 과학 및 Python 프로그래밍 입문

2016년 가을 학기