Final Project for Course 3 - OMDb and Dad Jokes Mashup

This project will take you through the process of mashing up data from two different APIs.

The OMDb API lets you provide a movie title as a query input and get back data about the movie, including scores from various review sites (Rotten Tomatoes, IMDb, etc.).

icanhazdadjokes.com returns random dad jokes containing a search string that you specify in your query.

The end result of this project will be a function that takes in a movie title as input and produces a formatted text string that includes a couple dad jokes related to a word from the movie's plot.

For example, here are a couple of sample outputs:

```
Baby Mama
Rotten Tomatoes rating: 63%
A successful, single businesswoman who dreams of having a baby
discovers she is infertile and hires a **working** class woman to be
her unlikely surrogate.
Speaking of **working**, that reminds me of some jokes.
Hope they're better than the movie!

Want to hear a joke about construction? Nah, I'm still **working** on
it.
Why doesn't the Chimney-Sweep call out sick from work? Because he's
used to **working** with a flue.
```

```
Back to the Future
Rotten Tomatoes rating: 93%
Marty McFly, a 17-year-old high school student, is **accidentally**
sent 30 years into the past in a time-traveling DeLorean invented by
his close friend, the maverick scientist Doc Brown.
Speaking of **accidentally**, that reminds me of some jokes.
Hope they're as good as the movie!

I **accidentally** drank a bottle of invisible ink. Now I’m in
hospital, waiting to be seen.
A butcher **accidentally** backed into his meat grinder and got a
little behind in his work that day.
```

To avoid problems with rate limits and site accessibility, we have provided a cache file with results for all the queries you need to make to both OMDb and icanhazdadjokes. Just use `requests_with_caching.get()` rather than `requests.get()`. If you're having trouble, you may not be formatting your queries properly, or you may not be asking for data that exists in our cache. We will try to provide as much information as we can to help guide you to form queries for which data exists in the cache.

ALERT: All Query Results Should Be Found in the Cache File

If you ever run `requests_with_caching.get()` and it says either of the following, then **something was wrong** with your query:

- new; adding to cache
- found in page-specific cache

```
import requests_with_caching

apikey = "abcd1234"  # you may *optionally* replace this with your API key.
# Note: you do *not* need an API key to complete this assignment. Every request should be in the cache
requests_with_caching.clear_cache()
# print(list(requests_with_caching.perm_cache().keys()))
```

Fetching movie info from OMDb

Your first task will be to fetch data from OMDb. The documentation for the API is at https://www.omdbapi.com/

Define a function called `get_movie_data`. It takes in one parameter which is a string that should represent the title of a movie you want to search. The function should return a dictionary with information about that movie.

Again, use `requests_with_caching.get()`. If you were to use the live OMDP API, you would need to get an API key, as described in the documentation. However **you do not need an API key** to complete this assignment. For the queries on movies that are already in the permanent cache, you won't need an API key.

HINT: Be sure to include **only** keys `t` and `r` as query parameters in order to extract data from the cache. If any other parameters are included, the function will not be able to recognize the data that you're attempting to pull from the cache.

The following movie titles are in the cache:

- Black Panther
- Venom
- Baby Mama

- Sherlock Holmes
- Return of the Jedi
- Back to the Future

```
import requests_with_caching

def get_movie_data(movie_title):
    base_url = "http://www.omdbapi.com/"

    # Set up parameters as required
    params = {
        "t": movie_title,
        "r": "json"
    }

    # Make the request using caching
    response = requests_with_caching.get(base_url, params=params)

    # Return the resulting JSON as a dictionary
    return response.json()

results = get_movie_data("Black Panther")
assert type(results) == type({})
assert results["Year"] == "2018"

# some other invocations that we use in the automated tests; uncomment
these if you are getting errors and want better error messages
# print(get_movie_data("Venom"))
# print(get_movie_data("Baby Mama"))
```

```
found in permanent_cache
```

Extract the Rotten Tomatoes Rating for a Movie

Next, you will write a function that extracts the Rotten Tomatoes rating for a movie from the results dictionary as an *integer*. If the movie does not have a Rotten Tomatoes rating, return `-1`.

Fill in the template for the function below

```
def rt_rating(rated):
    ratings = rated.get("Ratings", [])
    for rating in ratings:
        if rating.get("Source") == "Rotten Tomatoes":
            return int(rating["Value"].strip("%"))
    return -1
```

```
results = get_movie_data("Black Panther")
assert rt_rating(results) == 96
# We suggest that you write an assert statement to check the output of
your function for the movie Black Panther. The autograder will check
results for some other movies.
# results = get_movie_data("Black Panther")
# assert rt_rating(results) == 96

found in permanent_cache
```

Fetching Jokes

Now you will use another API to fetch a couple of dad jokes related to a movie's plot.

You will do this in two stages. First you'll implement a helper function that calls the API to get jokes, asking for jokes related to a single word.

Then you'll use that helper function, calling it with the longest words from the plot summary until it finds one that there are jokes for.

Search for Jokes Containing a Word

To search for dad jokes, you'll be using the API for icanhazdadjokes. The documentation for the API is at https://icanhazdadjoke.com/api

Define a function called `get_joke_data`. It takes in one parameter which is a string. The function should return a dictionary with information about **up to two** jokes that contain that string.

Again, use `requests_with_caching.get()`. All the query results you need are already in the permanent cache.

- Note 1: Be sure to include **only** keys `term` and `limit` as query parameters in order to extract data from the cache. If any other parameters are included, the function will not be able to recognize the data that you're attempting to pull from the cache.
- Note 2: Use the `limit` parameter in the icanhazdadjokes API to limit it to two results (instead of slicing)

```
def get_joke_data(data):
    url = "https://icanhazdadjoke.com/search"
    params = {
        "term": data,
        "limit": 2  # limit results to 2 jokes
    }
    headers = {"Accept": "application/json"}

    response = requests_with_caching.get(url, params=params,
```

```
headers=headers)
    return response.json()

assert (
    len(get_joke_data("magic")["results"]) == 2
), "The correct number of jokes for 'magic' should be 2"
assert (
    get_joke_data("magic")["results"][0]["joke"]
    == "What do you call a magician who has lost their magic? Ian."
)
```

```
found in permanent_cache
found in permanent_cache
```

Get Jokes for a Long Word from the Plot Description

Now you'll define a function called `get_jokes`. It will extract the longest word from the plot and try to find jokes for it. If there aren't any, it will proceed to the next longest word, and so on, until it finds a word for which there are jokes. If there is more than one word of the same length, try words that are earlier in the description first (which `sorted` does by default, since it's "stable" and will only move things around the minimum necessary).

We provide code that removes punctuation from the words in `plot` and assigns the result to the variable `words`. Your code should extend this by sorting `words` from longest to shortest and use the sorted list (and the `get_joke_data` function that you defined above) to find the longest word with associated jokes. If there are no words with associated jokes, your function should return the tuple `(None, None)`. If there is a word with associated jokes, your function should return a tuple with (1) the longest word with a joke and (2) a list of jokes associated with that word (as a list of strings).

```
def get_jokes(plot, verbosity=0):
    import string

    # Split plot into words and strip common punctuation
    words = plot.split()
    words = [w.strip(string.punctuation) for w in words]

    # Track unique words (case-insensitive)
    seen = set()
    unique_words = []
    for w in words:
        lw = w.lower()
        if lw not in seen:
            seen.add(lw)
            unique_words.append(w)

    # Sort words from longest to shortest
```

```
    unique_words.sort(key=lambda w: -len(w))

    # Try to get jokes for each word
    for word in unique_words:
        joke_data = get_joke_data(word)
        if joke_data and "results" in joke_data and
len(joke_data["results"]) > 0:
            if verbosity:
                print("Found jokes for:", word)
            jokes = [j["joke"] for j in joke_data["results"]]

            # Pad the joke list to exactly 2 items
            if len(jokes) == 1:
                jokes.append(jokes[0])
            elif len(jokes) == 0:
                continue

            return word, jokes

    # No jokes found
    return None, None

# Test case: Plot with a word that has jokes
plot = "I had dreams of a cat."
result = get_jokes(plot, 1)
print(result)

assert result[0] == "dreams", "Expected the word 'dreams'"
assert result[1][0] == "I'm tired of following my dreams. I'm just
going to ask them where they are going and meet up with them later."
assert result[1][1] == result[1][0], "Expected joke duplication if
only one exists"
```

```
found in permanent_cache
Found jokes for: dreams
('dreams', ["I'm tired of following my dreams. I'm just going to ask
them where they are going and meet up with them later.", "I'm tired of
following my dreams. I'm just going to ask them where they are going
and meet up with them later."])
```

BONUS CHALLENGE (ungraded). If we had specified that ties should be broken by taking words in the alphabetic order rather than later, how could you have done that? Try writing a test and then implementing it!

Put it All Together

Now put it all together to make the full app. Define a function, `haha_me`. It takes a movie name as input and verbosity and returns a text string that is meant to entertain the reader.

We have provided a helper function, `highlight`, that highlights a string inside a larger string by wrapping it in asterisks (`*`). Try invoking it a few times to make sure you understand what it does, then figure out how it should be used based on the sample outputs in the assert statements.

If the movie is not found in the OMDb API (using `get_movie_data`), return `"No movie found: "` followed by the movie title.

If the movie is found, but there are no jokes (`get_jokes` returned `(None, None)`), return `"I've got no jokes about this movie. It's too serious!"`.

If the movie and jokes are found, your function should return a string with each of the following on separate lines:

- The name of the movie
- `"Rotten Tomatoes rating: XX%"` (replacing `"XX"` with the actual Rotten Tomatoes rating)
- The plot of the movie with the joke keyword highlighted (using the provided `highlight` function)
- `"Speaking of **YY**, that reminds me of some jokes."` (replacing `"YY"` with the joke keyword)
- A different phrase about the jokes, depending on the Rotten Tomatoes rating:
  - No Rotten Tomates Rating (meaning the rating is `-1`): `"Hope you like them!"`
  - Rotten Tomatoes Rating below 70%: `"Hope they're better than the movie!"`
  - Rotten Tomates 70%+: `"Hope they're as good as the movie!"`
- *(an empty line)*
- The list of jokes, separated by a newline (using `"\n".join(...)`)

For example, for Venom:

```
Venom
Rotten Tomatoes rating: 30%
A failed reporter is bonded to an alien entity, one of many symbiotes
who have invaded **Earth**. But the being takes a liking to **Earth**
and decides to protect it.
Speaking of **Earth**, that reminds me of some jokes.
Hope they're better than the movie!

Astronomers got tired watching the moon go around the **Earth** for 24
hours. They decided to call it a day.
The rotation of **Earth** really makes my day.
```

```
def highlight(word: str, sentence: str) -> str:
    """
    Highlights a specific word in a sentence by surrounding it with
asterisks (**).
    The highlighting is case-insensitive.
```

```
    Args:
        word (str): The word to be highlighted.
        sentence (str): The sentence in which the word should be
highlighted.

    Returns:
        str: The sentence with the specified word highlighted.
    """
    import re

    # Escape special characters in the word to treat it as a literal
string
    # Use re.sub() to replace the word with the highlighted version
    # Use re.IGNORECASE flag to perform case-insensitive replacement
    return re.sub(re.escape(word), f"**{word}**", sentence,
flags=re.IGNORECASE)

def haha_me(movie_title: str, verbosity=0) -> str:
    import re

    movie_data = get_movie_data(movie_title)
    if movie_data.get("Response") == "False":
        return f"No movie found: {movie_title}"

    plot = movie_data["Plot"]
    rating = rt_rating(movie_data)
    word, jokes = get_jokes(plot, verbosity)

    if word is None:
        return "I've got no jokes about this movie. It's too serious!"

    # Keep original casing for use in output
    match = re.search(re.escape(word), plot, flags=re.IGNORECASE)
    plot_word = plot[match.start():match.end()] if match else word

    lines = []
    lines.append(movie_title)
    lines.append(
        f"Rotten Tomatoes rating: {rating}%"
        if rating != -1 else "Rotten Tomatoes rating: -1%"
    )
    lines.append(highlight(word, plot))
    lines.append(f"Speaking of **{plot_word}**, that reminds me of
some jokes.")

    if rating == -1:
        lines.append("Hope you like them!")
    elif rating < 70:
        lines.append("Hope they're better than the movie!")
    else:
```

```
        lines.append("Hope they're as good as the movie!")

    lines.append("")  # blank line before jokes

    highlighted_jokes = []
    for joke in jokes:
        highlighted_joke = highlight(word, joke)
        if highlighted_joke not in highlighted_jokes:
            highlighted_jokes.append(highlighted_joke)
        if len(highlighted_jokes) == 2:
            break

    lines.extend(highlighted_jokes)
    return "\n".join(lines)

assert (
    haha_me("Black Panther")
    == """Black Panther
Rotten Tomatoes rating: 96%
T'Challa, heir to the hidden but advanced kingdom of Wakanda, must
step **forward** to lead his people into a new future and must
confront a challenger from his country's past.
Speaking of **forward**, that reminds me of some jokes.
Hope they're as good as the movie!

Sometimes I tuck my knees into my chest and lean **forward**.  That’s
just how I roll.
Why do scuba divers fall backwards into the water? Because if they
fell **forward**s they’d still be in the boat."""
)
```

```
found in permanent_cache
found in permanent_cache
found in permanent_cache
found in permanent_cache
found in permanent_cache
found in permanent_cache
found in permanent_cache
found in permanent_cache
found in permanent_cache
```