# WikiScraper

Stwórz narzędzie do pobierania i analizy danych z wybranego przez Ciebie wiki z podanymi poniżej poleceniami. Wybierz dowolne wiki, którego licencja/regulamin pozwala na zbieranie z niej danych (np. licencja BY-NC-SA plus brak obostrzeń przeciwko scrapowaniu w regulaminie). Na przykład Bulbapedia, Explain XKCD, proteopedia.org. Najlepiej nie używaj strony z subdomen wikipedia.org, ponieważ ostatnio wprowadzane są tam zabezpieczenia przeciwko scrapowaniu.

Ponadto korzystając z doświadczeń uzyskanych przy tworzeniu programu zaproponuj i zanalizuj metodę wykrywania języka na podstawie częstotliwości słów w badanym tekście (niekoniecznie wiki).

## Funkcjonalności programu

Program powinien być uruchamiany (po zainicjalizowaniu środowiska, np. virtualenv) za pomocą

```
python wiki_scraper.py <argumenty>
```

i w zależności od argumentów wykonywać poniższe polecenia. Oto możliwe argumenty (gdzie [--coś] oznacza, że argument jest opcjonalny, a wszystko w cudzysłowach to jakieś dane od użytkownika):

```
--summary "szukana fraza"
```

Na początku program powinien pobrać kod źródłowy strony wiki dla szukanej frazy. Na przykład dla frazy "Team Rocket" na bulbapedii byłaby to strona o adresie https://bulbapedia.bulbagarden.net/wiki/Team_Rocket. Na wielu wiki adres dla różnych fraz składa się z stałego prefiksu i szukanej frazy ze spacjami zamienionymi na podkreślniki. Obsłuż przypadki, w których artykuł dla podanej frazy nie jest dostępny na wiki. Możesz założyć, że fraza nie będzie zawierać znaków innych niż litery alfabetu łacińskiego i spacje.

Program powinien znaleźć pierwszy paragraf artykułu dla wybranej frazy (pomijając wszelkie elementy stałe strony jak menu) i wypisać jego tekstową zawartość bez żadnych znaczników HTML. Przykładowo dla powyższego artykułu powinno wypisać zawartość pierwszego <p> w  pomijając wszelkie znaczniki wewnątrz, tj. "Team Rocket (Japanese: ロケット団だん Rocket-dan, literally Rocket Gang) is a villainous team (...) with a small outpost in the Sevii Islands.", w szczególności pomijając <b>, , itp. Może przydać się funkcja `get_text` w Beautiful Soup.

## `--table "szukana fraza" --number n [--first-row-is-header]`

Program powinien pobrać stronę dla szukanej frazy, jak w "--summary", a następnie znaleźć n-tą tabelę na stronie (numerując od 1, gdzie tabela to konkretnie <table>), wypisać z niej dane tekstowe i zapisać do pliku "szukana fraza.csv" (z odpowiednio podmienioną frazą z argumentów linii polecenia). Wystarczą te dane, które jest w stanie obsłużyć pandas po przekazaniu mu odpowiednich danych z Beautiful Soup. Można zignorować nietypowe dane, których pandas nie będzie w stanie obsłużyć, np. obrazki. Interesuje nas też tylko tekst w komórkach, bez znaczników HTML, tak, jak w `--summary`. Na przykład dla `--table Type --number 2` na przykładzie bulbapedii (https://bulbapedia.bulbagarden.net/wiki/Type) program powinien pokazać tabelę z podrozdziału "Type Chart" z różnymi mnożnikami.

Jeżeli opcja `--first-row-is-header` jest podana, to pierwszy wiersz tabeli powinien być traktowany jak nagłówki kolumn. W przeciwnym razie powinno się zakładać brak nagłówków kolumn. Powinno się zakładać, że pierwsza kolumna to nagłówki wierszy. Jeżeli wiki ma na ogół dobrze zaznaczone, co jest nagłówkiem za pomocą np. `<thead>` i `<th>` i jesteś w stanie odpowiednio to użyć do automatycznego wykrycia i użycia obecności nagłówków zamiast potraktowania wszystkiego jako dane w tabelce, to możesz nie implementować tego argumentu linii polecenia i zamiast tego polegać na automatycznym wykrywaniu.

Ponadto program powinien wypisać w formie tabeli (tj. z wyraźnie widocznymi kolumnami; wystarczy tak, jak to robi pandas), ile razy dana wartość wystąpiła w tabeli z wyłączeniem nagłówków.

## `--count-words "szukana fraza"`

Program powinien pobrać stronę dla szukanej frazy, jak w `--summary`, następnie znaleźć wszystkie słowa w tekście artykułu i je policzyć. W słowa artykułu nie powinny wchodzić elementy stałe strony jak menu, ale mogą (nie muszą) wchodzić wszelkie komunikaty (jak "wymaga uzupełnienia"). Program następnie powinien utworzyć lub zaktualizować zserializowany do formatu JSON słownik z tymi słowami jako kluczami i całkowitą liczbą z wszystkich uruchomień programu jako wartościami. Ten słownik powinien znajdować się w `./word-counts.json`.

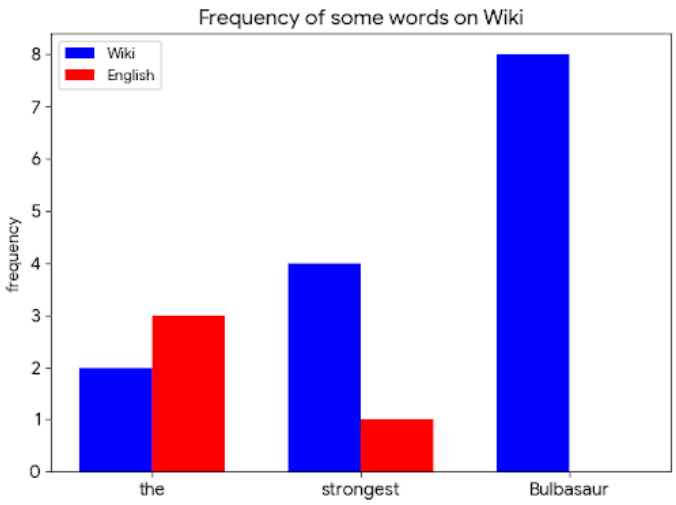
## `--analyze-relative-word-frequency --mode "tryb" --count n [--chart "ścieżka/do/pliku.png"]`

Program powinien porównać i zwizualizować częstotliwość używanych słów zebranych za pomocą `--count-words` w porównaniu do częstotliwości tych słów w języku, w którym napisana jest wybrana wiki. W tym celu zdobądź informacje o pewnej liczbie najczęstszych słów używanych w języku wiki (zobacz komentarz *) i za pomocą `pandas` stwórz i wypisz tabelę z

kolumnami: word, frequency in the article, frequency in wiki language (słowo, częstotliwość w artykule, częstotliwość w języku wiki). Znormalizuj częstotliwość tak, by obie wartości były w jakiś sposób porównywalne, np. podziel je przez częstotliwość najczęstszego słowa. Tabela powinna zawierać n wierszy i w zależności od parametru `--mode` równego:

- `article` - być posortowana po częstotliwości zebranych słów na wiki i ma zawierać luki w drugiej kolumnie dla słów, które nie występują w wybranym słowniku języka wiki (np. nazwa własna Bulbasaur z bulbapedii);
- `language` - być posortowana po częstotliwości słów w języku wiki i ma zawierać luki w pierwszej kolumnie dla słów, które nie występują na wiki.

Jeżeli podano argument `--chart` to program powinien również stworzyć plik z wykresem słupkowym i z podaną ścieżką. Wykres powinien zawierać 2n słupków z częstotliwościami obu słów w języku wiki i w artykułach, po 2 na każde słowo. Zadbaj o estetykę wykresu, by miał tytuł, by każdy z typów częstotliwości miał inny kolor i aby wykres zawierał legendę do tych kolorów. Możesz się posłużyć poniższym wykresem jako inspiracją:




**(*)** Może być bardzo przydatna paczka wordfreq, ale możesz wybrać dowolne źródło danych statystycznych o języku wiki (uwaga na licencje). Źródło to powinno zawierać informacje o co najmniej 1000 najczęstszych słów języka wiki.

`--auto-count-words "początkowa szukana fraza" --depth n --wait t`

Program powinien wykonywać to, co `--count-words`, zaczynając od podanej frazy i podążając za wszystkimi linkami do innych fraz, do głębokości n. Przez głębokość rozumiemy liczbę krawędzi od początku w grafie linków między stronami albo inaczej podążanie do n razy linkiem do innej strony. Dla każdej przetwarzanej frazy, program powinien:

- wypisać przetwarzaną frazę,
- pobrać treść artykułu (bez elementów stałych strony),

- jeżeli do obecnej frazy prowadzi sekwencja mniej niż n linków od początkowej frazy, wybrać z niego wszystkie linki prowadzące do innych fraz, których jeszcze nie odwiedził,
- wykonać `--count-words` dla przetwarzanej frazy,
- poczekać `t` sekund,
- zrobić to samo dla kolejnej, jeszcze nie przetworzonej frazy.

Możesz sam wybrać w jakiej kolejności chodzisz po grafie linków na wiki, np. BFS lub DFS. Wykrywanie, które linki prowadzą do innych fraz, może być zrealizowane np. za pomocą sprawdzenia, czy atrybut href w linku ma odpowiedni prefiks. Przykładowo dla bulbapedii jest to `/wiki/`. Program nie powinien wychodzić poza wiki. Podczas testowania pamiętaj, że `t=0` może spowodować, że wiki zablokuje Cię na jakiś czas za zbyt dużą liczbę żądań na sekundę.

## Niefunkcjonalne wymagania programu

Program powinien być zaprojektowany zgodnie z zasadami programowania obiektowego oraz rozbity na moduły umożliwiające ich ponowne wykorzystywanie w odosobnieniu od innych (na tyle, na ile to możliwe). W szczególności powinno być możliwe zaimportowanie modułów w środowisku REPL lub notatniku Jupyter i ich użycie po podaniu odpowiednich parametrów. Nie wszystko musi być obiektowe - funkcje są mile widziane tam, gdzie nie pracujemy z jakimś stanem (jak "obecnie przetwarzana fraza"). Niech jednak istnieje klasa kontrolująca działanie programu, do której przekazywane są już sparsowane parametry linii polecenia oraz klasa dla obiektu przetwarzającego wybrany artykuł.

Napisz kod tak, aby był łatwo testowalny, tj. aby dało się wiele funkcjonalności przetestować odrębnie z REPLa lub Jupytera tworząc odpowiednie obiekty i przekazując im bezpośrednio utworzone parametry. Przykładowo można tak zaprojektować klasę scrapera, by można było w REPLu utworzyć go dla pojedynczej frazy:

```
scraper = Scraper(BULBAPEDIA_URL, “Team Rocket”,
                  use_local_html_file_instead=True)
```

i wywołać jego metody (np. `scraper.scrape()`), ale np. należy nie projektować powyższego tak, by brało szukaną frazę czy tryb działania bezpośrednio od `argparse` czy jakiegoś globalnego obiektu. Zapewnij, by scraper zamiast pobierania kodu strony z wiki mógł przeczytać plik na dysku po podaniu odpowiednich argumentów do jego konstruktora bądź jakiejś metody. Zwróć uwagę, że ze względu na optymalizowanie ponownego importowania modułów w Pythonie, mimo tych ułatwień, będzie konieczne restartowanie REPLa/Jupytera po zmianach lub manualne odświeżanie modułu i utworzonych obiektów.

Wykorzystaj zaimplementowane ułatwienia do testowania kawałków kodu do stworzenia 4 testów jednostkowych testujących różne pojedyncze metody lub funkcje nie nawiązując żadnego połączenia internetowego. Mogą w nich być wywoływane inne metody i funkcje w ramach inicjalizacji, ale testowana powinna być jedna konkretna metoda czy funkcja. Mogą to być naprawdę małe funkcje, np. metoda dająca informację, czy dany link jest artykułem z wiki.

Stwórz test integracyjny, który będzie programem z innym wykonywanym plikiem ("mainem") uruchamianym za pomocą (po inicjalizacji środowiska):

```
python wiki_scraper_integration_test.py
```

który ładuje z pliku na dysku (zamiast ze strony wiki) wybraną frazę i testuje jedną z głównych funkcjonalności (np. `--summary` dla wybranej frazy). Przykładowo, w przypadku Bulbapedii można zapisać HTML strony o Team Rocket i sprawdzić, czy wykryty tekst zaczyna się od "Team Rocket" i kończy na "outpost in the Sevil Islands.". Program powinien się zakończyć z niezerowym kodem błędu w przypadku niepowodzenia testu (np. w wyniku zgłoszenia jakiegoś wyjątku lub niespełnienia asserta).

Podczas pisania programu stosuj się do stylu z PEP8. Tolerowana będzie niezgodność (przypadkowa bądź specjalna) z co najwyżej trzema regułami (np. jak preferujesz nieco dłuższe linie).

Pisząc program użyj jakiegoś repozytorium (np. git) i zapisuj do niego postępy pracy na bieżąco. W szczególności powinno ono zawierać co najmniej kilka commitów oddalonych od siebie o rozsądną ilość czasu. Repozytorium nie musi być publiczne, wystarczy pokazać jego historię przy prezentacji zadania.

## Dodatkowe ustalenia do programu

- Aby być bezpiecznym w kwestii licencji używanych danych możesz program i jego wyjście udostępnić na dowolnej licencji pozwalającej na sprawdzenie programu i możesz dodać do wyjścia odpowiednie informacje. Przykładowo, aby spełnić warunki licencji BY-NC-SA bulbapedii, możesz wypisać "Wyjście programu na licencji BY-NC-SA stworzone na podstawie artykułu dostępnego na stronie https://bulbapedia.bulbagarden.net/wiki/Team_Rocket.").
- Możesz pracować na słowach w takiej formie, w jakiej są w tekście, tj. nie musisz przejmować się ich odmianą. Dość dobry do analizy jest język angielski, bo łatwo go podzielić na słowa i stosunkowo często są one w niezmienionej formie. Niemniej, możesz wykorzystać wiki w dowolnym języku, nawet z tak bogatą odmianą słów jak polski. Nie będzie oceniana sensowność analizy, tj. jest akceptowalne, że wiki będzie zawierać w dużej części rozłączne słowa ze zbiorem słów z wybranego języka w wersji słownikowej.

## Analiza do przeprowadzenia

Mając do dyspozycji tylko słownik z liczbą poszczególnych słów w danym tekście (jak w `--count-words`) oraz dużą listę najpopularniejszych słów w danym języku z częstotliwościami występowania (posortowaną od najczęstszych), zaproponuj funkcję `lang_confidence_score(word_counts, language_words_with_frequency)` oceniającą czy słowa należą do danego języka z wartościami rosnącymi w miarę lepszego

dopasowania słów z `word_counts` do języka z najczęstszymi słowami `language_words_with_frequency`.

Stwórz notatnik Jupyter, w którym zaimplementowana jest ta funkcja i zbadaj jej skuteczność. W tym celu wybierz co najmniej dwa języki plus język wybranego wiki i zdobądź następujące dane:

- listę co najmniej 1000 najczęściej używanych słów w tych językach (i ich częstotliwości, o ile chcesz jej użyć, np. z paczki wordfreq),
- słownik z `word-counts.json` będący wynikiem `--count-words` dla jednego, dłuższego artykułu z wiki (w miarę możliwości zawierającego co najmniej 5000 słów),
- słownik z `word-counts.json` będący wynikiem `--count-words` dla tak dobranego artykułu z wiki (mającego co najmniej 20 słów), by wynik funkcji był możliwie niski dla wybranego artykułu względem innych języków; w razie problemów ze znalezieniem takiego artykułu możesz przerwać poszukiwania po przejrzeniu 50 artykułów i wybrać najgorszy,
- podobny słownik utworzony na podstawie dłuższego tekstu spoza wiki dla każdego z trzech analizowanych języków.

Dla każdej wartości k = 3, 10, 100, 1000, dla każdego z trzech analizowanych języków, dla każdego z pięciu testowych artykułów (z wiki)/tekstów (spoza wiki): oblicz wartość funkcji na podstawie słownika z liczbą słów w artykule/tekście spoza wiki i najczęstszych k słowach z danego języka wraz z częstotliwością występowania.

Sporządź wykresy przedstawiające w czytelny sposób wyniki dla różnych k.

Zawrzyj w notatniku Jupyter opis słowny wyników empirycznych skuteczności utworzonej metody wykrywania dopasowania słów z danych do najczęstszych k słów danego języka. Ponadto odpowiedz na następujące pytania:

- Czy dobór języków miał duże znaczenie?
- Czy po wartościach `language_words_with_frequency` dla danych i najczęstszych słowach z języka danych widać, że w wybranym języku słowa często są odmieniane?
- Czy trudne było znalezienie takiego artykułu, dla którego wynik `language_words_with_frequency` jest jak najmniejszy w języku wiki? Czy to specyfika tego wiki?

Nie będzie oceniany sam dobór funkcji `language_words_with_frequency` ani jej skuteczność (w granicach rozsądku).

# Kryteria oceniania

Poniżej znajduje się ogólny zarys punktacji, gdzie maksymalna liczba punktów jest za w pełni działające i napisane zgodnie z wszystkimi wytycznymi części zadania:

- `--summary` - 4 pkt.,
- `--table` - 6 pkt.,

- `--count-words` - 4 pkt.,
- `--analyze-relative-word-frequency` - 12 pkt. (w tym 2 pkt. za `--chart`),
- `--auto-count-words` - 8 pkt.,
- testy jednostkowe - 6 pkt.,
- test integracyjny - 6 pkt.,
- analiza języka tekstu - 24 pkt. (w tym 8 pkt. za implementację, 8 pkt. za wykresy i 8 pkt. za klarowne przedstawienie wyników badań).

Oprócz zgodnego z wytycznymi działania programu i notatnika Jupytera bardzo ważną częścią oceny będzie prezentacja programu. W szczególności można stracić punkty za:

- (szczególnie ważne) brak umiejętności opisania, co robią różne części programu i jakie dane przesyłają między sobą,
- brak umiejętności opisania, co robi każda użyta funkcja/klasa z zaimportowanej biblioteki z użytymi w kodzie parametrami,
- brak zgodności stylu formatowania kodu z PEP8,
- brak napisania programu we względnie obiektowy sposób,
- brak poprawnego użycia repozytorium.

Powyższa lista nie jest kompletna. Ostateczne słowo w sprawie kryteriów oceniania ma prowadzący grupę laboratoryjną i mogą je dostosowywać w swoich grupach.