BÁO CÁO MÔN HỌC:

**Nhập môn ngành (Nhật)**

Tên đề tài:

**SPELLING CORRECTOR**

1. **Đơn vị thực hiện:**

* Đơn vị thực hiện: Nhóm sinh viên
* Trần Đức Duy
* Nguyễn Thành Hiếu
* Trần Ngọc Minh Hoàng.
* Giáo viên hướng dẫn: Đặng Hoài Phương

1. **Các căn cứ:**
2. **Đặt vấn đề:**

* Hiện nay có rất nhiều ứng dụng giao tiếp, trao đổi công việc được ra đời và phát triển. Việc giao tiếp trên mạng đã trở thành xu hướng tương lai bởi nó giải quyết được hai vấn đề cốt lõi trong giao tiếp là vị trí và thời gian. Nhờ vào các ứng dụng này việc thực hiện trao đổi thông tin trong công việc trở nên dễ dàng hơn bao giờ hết. Tuy nhiên việc giao tiếp trên mạng vẫn có những vấn đề còn tồn động. Một trong số đó chính là việc người dùng sử dụng từ ngữ để bộc lộ suy nghĩ thay vì lời nói như trước nên việc sai lỗi chính tả diễn ra thường xuyên và khiến cho người dùng khó chịu bởi sai một từ ngữ có thể khiến việc diễn đạt bị lệch đi và việc truyền ý tưởng tới người khác bị sai lệch. Và chẳng có ai muốn viết một tin nhắn hay mail thật dài mà lại bị sai chính tả cả, chính vì vậy mà cần phải có phần mềm sửa lỗi chính tả được tạo ra.

1. **Tính khả thi:**

* Hiện nay đã có rất nhiều trang web, ứng dụng và các project có sẵn nên rất dễ nghiên cứu.
* Chương trình chạy bằng Python nên có nhiều thư viện hỗ trợ cho xử lí ngôn ngữ tự nhiên.
* Đã có sẵn các file dữ liệu training uy tín được lấy từ các cuốn sách và từ điển nhằm phục vụ cho xử lí ngôn ngữ tự nhiên.

1. **Mục tiêu:**

* Tạo ra phần mềm chỉnh sửa lỗi chính tả ngôn ngữ Tiếng Anh đạt độ chính xác 80% và thời gian chạy phần mềm 80 từ/s.

1. ***Nội dung:***
2. ***Phương hướng xây dựng:***

* Nghiên cứu cơ sở lý thuyết xử lí ngôn ngữ tự nhiên.
* Tìm ra giải pháp xử lí văn bản.
* Xây dựng chương trình chỉnh sửa lỗi chính tả từ các thư viện xử lí ngôn ngữ tự trên trên Python.
* Xây dựng giao diện tương tác cho người dùng.

1. ***Tính năng cần có:***

* Chỉnh sửa lỗi chính tả của từ đơn.
* Có thể chỉnh sửa nhiều lần qua giao diện thao tác của người dùng.

1. ***Quá trình nghiên cứu và tạo ra phần mềm:***
2. ***Xử lí ngôn ngữ tự nhiên***
3. ***Giới thiệu xử lí ngôn ngữ tự nhiên:***

* **Xử lý ngôn ngữ tự nhiên (Natural Language Processing)** là một nhánh của Trí tuệ nhân tạo, tập trung vào việc nghiên cứu sự tương tác giữa máy tính và ngôn ngữ tự nhiên của con người, dưới dạng tiếng nói (speech) hoặc văn bản (text).
* **Mục tiêu của lĩnh vực này** là giúp máy tính hiểu và thực hiện hiệu quả những nhiệm vụ liên quan đến ngôn ngữ của con người như:
* Tương tác giữa người và máy.
* Cải thiện hiệu quả giao tiếp giữa người với người.
* Nâng cao hiệu quả xử lý văn bản.
* …
* **NPL** có thể chia thành 2 nhánh lớn không hoàn toàn độc lập với nhau: **Xử lý tiếng nói** và **Xử lý văn bản.**
* **Xử lý tiếng nói:** Tập trung nghiên cứu, phát triển các thuật toán, chương trình máy tính xử lý ngôn ngữ của con người ở dạng tiếng nói (dữ liệu âm thanh). Các ứng dụng quan trọng của xử lý tiếng nói bao gồm **nhận dạng tiếng nói** và **tổng hợp tiếng nói.**
* **Nhận dạng tiếng nói:** Chuyển ngôn ngữ từ dạng tiếng nói sang dạng văn bản.
* **Tổng hợp tiếng nói:** Chuyển ngôn ngữ từ dạng văn bản thành tiếng nói.
* **Xử lý văn bản:** Tập trung phân tích dữ liệu văn bản. Các ứng dụng quan trọng của xử lý văn bản bao gồm tìm kiếm và truy xuất thông tin, dịch máy, tóm tắt văn bản tự động, hay kiểm lỗi chính tả tự động.

1. ***Các bước xử lý văn bản***

* **Phân tích hình vị:** là sự nhận biết, phân tích, và miêu tả cấu trúc của hình vị trong một ngôn ngữ cho trước và các đơn vị ngôn ngữ khác, như từ gốc, biên từ, phụ tố, từ loại, v.v. Trong xử lý tiếng Việt, hai bài toán điển hình trong phần này là tách từ (word segmentation) và gán nhãn từ loại (part-of-speech tagging).
* **Phân tích cú pháp:** là quy trình phân tích một chuỗi các biểu tượng, ở dạng ngôn ngữ tự nhiên hoặc ngôn ngữ máy tính, tuân theo văn phạm hình thức. Văn phạm hình thức thường dùng trong phân tích cú pháp của ngôn ngữ tự nhiên bao gồm Văn phạm phi ngữ cảnh (Context-free grammar – CFG), Văn phạm danh mục kết nối (Combinatory categorial grammar – CCG), và Văn phạm phụ thuộc (Dependency grammar – DG). Đầu vào của quá trình phân tích là một câu gồm một chuỗi từ và nhãn từ loại của chúng, và đầu ra là một cây phân tích thể hiện cấu trúc cú pháp của câu đó.
* **Phân tích ngữ nghĩa:** là quá trình liên hệ cấu trúc ngữ nghĩa, từ cấp độ cụm từ, mệnh đề, câu và đoạn đến cấp độ toàn bài viết, với ý nghĩa độc lập của chúng. Nói cách khác, việc này nhằm tìm ra ngữ nghĩa của đầu vào ngôn từ. Phân tích ngữ nghĩa bao gồm hai mức độ: Ngữ nghĩa từ vựng biểu hiện các ý nghĩa của những từ thành phần, và phân biệt nghĩa của từ; Ngữ nghĩa thành phần liên quan đến cách thức các từ liên kết để hình thành những nghĩa rộng hơn.
* **Phân tích diễn ngôn:** là phân tích văn bản có xét tới mối quan hệ giữa ngôn ngữ và ngữ cảnh sử dụng (context-of-use). Phân tích diễn ngôn, do đó, được thực hiện ở mức độ đoạn văn hoặc toàn bộ văn bản thay vì chỉ phân tích riêng ở mức câu.

1. ***Sự phức tạp của NPL***
2. **Tri thức nền (background knowledge):**

* Việc hình thành các tri thức cần thiết để có đủ khả năng phân tích, phán đoán, xử lý tình huống cần trải qua một khoảng thời gian rất dài.
* Việc đưa các tri thức nền này vào máy tính là thách thức lớn, đến nay vẫn chưa có giải pháp tốt.
* Ví dụ: Một đứa trẻ, từ khi sinh ra cho tới khi tập nói, tập đọc trải qua một giai đoạn dài phát triển. Trong giai đoạn này đứa trẻ không ngừng tiếp xúc với thế giới bên ngoài, có các trải nghiệm và tiếp thu kiến thức từ thế giới xung quanh. Những tri thức cơ bản dần dần được hình thành như lửa thì nóng, nước đá thì lạnh, đi đường đèn đỏ thì dừng, đèn xanh thì đi, nhưng không có đèn tím. Những tri thức này giúp ích rất nhiều cho con người trong việc hiểu ngôn ngữ.

1. **Tính nhập nhằng của ngôn ngữ (ambiguity):**

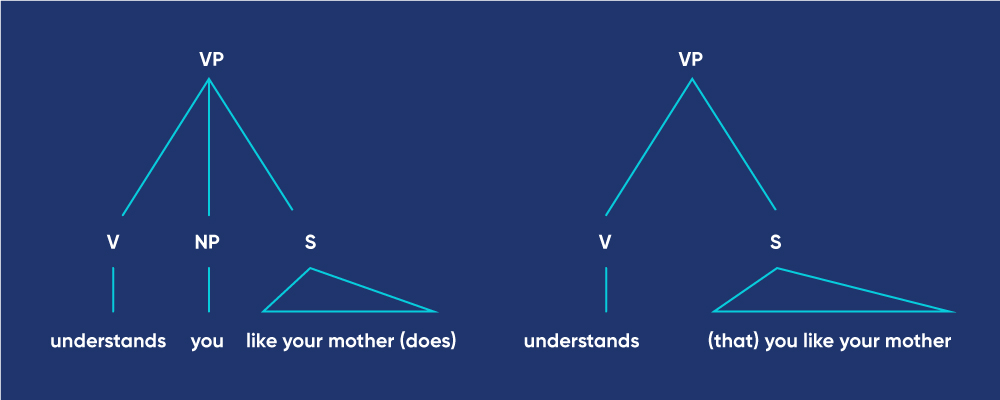
* **Nhập nhằng** là hiện tượng xảy ra khi ngôn ngữ có thể được hiểu theo nhiều cách khác nhau, tùy thuộc vào ngữ cảnh mà nó xuất hiện.
* Trong xử lý ngôn ngữ tự nhiên, **nhập nhằng** có thể xuất hiện ở nhiều cấp độ, từ vựng, ngữ pháp, ngữ nghĩa, dẫn tới khó khăn trong việc xử lý trên máy tính.
* Ví dụ 1:

(S1) They book that hotel.

(S2) They read that book.

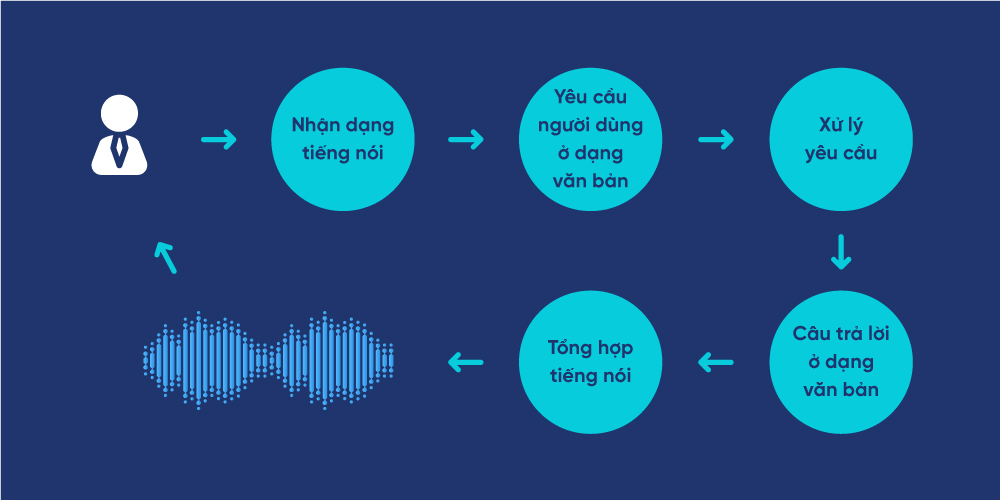
Đầu tiên, từ book là **nhập nhằng** về mặt từ loại. Book có thể là một động từ (trong câu S1) hoặc một danh từ (trong câu S2) tùy thuộc vào ngữ cảnh xuất hiện của nó. Hiện tượng này gây khó khăn cho bài toán gán nhãn từ loại. Không chỉ vậy, book cũng **nhập nhằng** về mặt ngữ nghĩa. Book có thể là một hành động đặt hàng thứ gì đó (trong câu S1) hoặc có thể là một văn bản viết được xuất bản dưới dạng in ấn hay điện tử (trong câu S2). Hiện tượng này gây khó khăn cho bài toán xác định nghĩa của từ.

* Ví dụ 2:



Ở góc độ ngữ pháp, câu này có thể được giải thích theo hai cây cú pháp như trên hình. Những cấu trúc khác nhau dẫn đến những cách hiểu khác nhau: **“a computer understands you like your mother does”** hoặc **“a computer understands that you like your mother”.** Hiện tượng này gây khó khăn cho cả hai bài toán là **phân tích cú pháp** và **phân tích ngữ nghĩa.**

1. ***Giao tiếp giữa người và máy dựa trên NLP***



* Hình 2 mô tả kiến trúc tiêu biểu của một chương trình máy tính giao tiếp với con người qua tiếng nói. Chương trình sẽ bao gồm các bước cơ bản sau:

1. **Nhận dạng tiếng nói:** ở bước này, máy tính sẽ nhận dạng yêu cầu của người dùng ở dạng tiếng nói và chuyển yêu cầu này về dạng văn bản.
2. **Xử lý yêu cầu:** máy tính sẽ phân tích yêu cầu ở dạng văn bản, xử lý, đưa ra câu trả lời sử dụng các kỹ thuật trong xử lý văn bản.
3. **Tổng hợp tiếng nói:** ở bước này, câu trả lời sẽ được chuyển từ dạng văn bản sang tiếng nói và gửi tới người dùng.
4. ***Các bài toán cơ bản trong NPL***

* **Mô hình hóa ngôn ngữ (Language modelling)**

**Mô hình hóa ngôn ngữ (LM)** gán một xác suất cho bất kỳ chuỗi từ nào. Về cơ bản, trong bài toán này, ta cần dự đoán từ tiếp theo xuất hiện theo trình tự, dựa trên lịch sử của các từ đã xuất hiện trước đó. **LM** rất quan trọng trong các ứng dụng khác nhau của **NLP**, và là lý do tại sao máy móc có thể hiểu được thông tin định tính. Một số ứng dụng của **Mô hình hóa ngôn ngữ** bao gồm: nhận dạng giọng nói, nhận dạng ký tự quang học, nhận dạng chữ viết tay, dịch máy và sửa lỗi chính tả.

* **Phân loại văn bản (Text classification)**

**Phân loại văn bản** gán các danh mục được xác định trước cho văn bản dựa trên nội dung của nó. Cho đến nay, phân loại văn bản là ứng dụng phổ biến nhất của **NLP**, được sử dụng để xây dựng các công cụ khác nhau như trình phát hiện thư rác và chương trình phân tích cảm xúc.

* **Trích xuất thông tin (Information extraction)**

**Trích xuất thông tin (IE)** tự động trích xuất thông tin có liên quan từ các tài liệu văn bản không có cấu trúc và / hoặc bán cấu trúc. Ví dụ về các loại tài liệu này bao gồm lịch sự kiện từ email hoặc tên của những người được đề cập trong một bài đăng trên mạng xã hội.

* **Truy xuất thông tin (Information retrieval)**

Google là một loại hệ thống **Truy xuất Thông tin (IR)** phổ biến nhất mà chúng ta thường sử dụng. **IR** làm nhiệm vụ tìm kiếm các tài liệu có liên quan từ một bộ dữ liệu lớn các tài liệu liên quan đến truy vấn do người dùng thực hiện.

* **Tác tử phần mềm hội thoại (Conversational agent)**

**Tác tử phần mềm hội thoại thuộc AI hội thoại**, liên quan đến việc xây dựng các hệ thống đối thoại mô phỏng các tương tác của con người. Các ví dụ phổ biến về AI hội thoại bao gồm Alexa, Siri, Google Home, Cortana, hay trợ lý ảo ViVi. Các công nghệ như chatbot cũng được hỗ trợ bởi  tác tử phần mềm hội thoại và ngày càng phổ biến trong các doanh nghiệp.

* **Tóm tắt văn bản (Text summarization)**

**Tự động tóm tắt** là quá trình rút ngắn một tập hợp dữ liệu để tạo một tập hợp con đại diện cho thông tin quan trọng nhất hoặc có liên quan trong nội dung gốc

* **Hỏi đáp (Question answering)**

**Hỏi đáp** là bài toán xây dựng các hệ thống có thể tự động trả lời cho các câu hỏi do con người đặt ra bằng ngôn ngữ tự nhiên.

* **Dịch máy (Machine translation)**

**Dịch máy (MT)** là một nhánh con của ngôn ngữ học tính toán liên quan đến việc chuyển đổi một đoạn văn bản từ ngôn ngữ này sang ngôn ngữ khác. Một ứng dụng phổ biến của loại này là Google Dịch.

* **Mô hình hóa chủ đề (Topic modelling)**

**Mô hình hóa chủ đề** là một kỹ thuật Học máy không giám sát giúp khám phá cấu trúc chủ đề của một bộ tài liệu lớn. Ứng dụng NLP này là một công cụ khá phổ biến, được sử dụng trên nhiều lĩnh vực khác nhau – như Văn học, và Tin sinh học.

1. ***NPL trong sửa lỗi chính tả tiếng Anh***

**Tự động phát hiện và sửa lỗi chính tả (Auto Correction)** là một trong những bài toán cơ bản nhất trong xử lý ngôn ngữ tự nhiên. Tính năng này có trong các ứng dụng soạn thảo văn bản, nhập liệu, nhận dạng.

**Phương pháp cơ bản:**

Các phương pháp truyền thống trong sửa lỗi chính tả dựa vào hai thành phần cơ bản là **Language Model** và **Error Model**. **Language Model** là mô hình xác xuất, cho biết xác suất xuất hiện của một từ trong một ngữ cảnh (context). Context thường là các từ đứng ngay trước từ đang được xem xét.

Gọi w là từ người dùng gõ. Gọi c là từ cần xem xét để tự động sửa w thành c. Yêu cầu của việc sửa lỗi chính tả là xác định c sao cho xác suất có điều kiện P(c|w) là lớn nhất.

Theo công thức Bayes:

P(c|w) = P(c). P(w|c) / P(w)

Trong đó:

– **P(c)**: xác xuất của từ **c**

– **P(w|c)**: xác xuất người dùng nhập nhầm **w** thay vì **c** (như mong muốn).

– **P(w)**: xác xuất nhập từ **w**.

Vì **P(w)** không đổi với các c khác nhau, để tìm P(c|w) lớn nhất, ta cần tìm csao cho P(c).P(w|c) lớn nhất.

Một cách trực quan, từ sửa đúng nhất là từ có xác suất lớn (thể hiện qua P(c)), và gần giống với từ được gõ (thể hiện qua P(w|c)). P(c) được xác định từ mô hình **Language Model**, trong khi P(w|c) được xác định trong **Error Model** sẽ được được mô tả dưới đây.

**Language Model**

**Language Model** là mô hình xác suất cho biết xác suất xuất hiện của một từ. Mô hình xác suất của các từ đơn lẻ được gọi là unigram language model. Trên thực tế, language model không chỉ cân nhắc đến các từ đơn lẻ, mà còn tính đến ngữ cảnh, bao gồm các từ đứng trước hoặc sau từ đang được xem xét. Với bài toán nhập văn bản, chúng ta xét đến các từ đi liền trước. Xác suất một từ được cân nhắc là P(c|s), trong đó s là chuỗi các từ đi liền trước:

– **Bigram language model:** P(c|x) là mô hình xác xuất tính đến 1 từ ngay phía trước, cho biết xác xuất của c khi từ đi trước là x.

– **Trigram language model:** P(c | x1x2), xác xuất c đi sau 2 từ x1 x2.

– **N-gram language model:** P(c | x1…xn), xác xuất c đi sau n từ: x1…xn.

Để xây dựng language model, chúng ta dựa vào kho dữ liệu văn bản lớn (corpus) đếm tần suất xuất hiện của các từ, cụm từ. Mô hình thường được dùng nhất là bigram. Từ trigram trở lên, language model đòi hỏi kích thước rất lớn nên ít được dùng trên thực tế.

**Error Model**

**Error Model** là mô hình đánh giá khả năng một từ mong muốn c bị viết/gõ nhầm sang từ khác w: P(w|c). Để xây dựng error model, có hai cách tiếp cận:

– Dựa trên thống kê lỗi trên dữ liệu văn bản thu thập được, xây dựng mô hình xác xuất. Phương pháp này rất khó thực hiện, vì tần suất lỗi chính tả trong dữ liệu thu thập thường rất thấp. Để có mô hình tốt, cần phải có dữ liệu rất lớn.

– Định nghĩa các loại lỗi thường xảy ra và xây dựng mô hình đánh giá khả năng xảy ra lỗi (rules based). Đây là phương pháp đơn giản và cũng thường được dùng nhất, sẽ được mô tả chi tiết dưới đây.

Các lỗi nhập liệu sai có thể xếp thành các loại:

– **Replace**: something -> simething

– **Transpose**: something -> smoething

– **Delete**: something -> smething

– **Insert**: something -> sometthing

Để đo sự khác nhau giữa 2 chuỗi {c, w}, chúng ta dùng edit distance là số phép biến đổi được định nghĩa ở trên để biến c thành w, hoặc ngược lại.

**Ví dụ:** với w = somthing

– Tập các từ với edit distance = 1 là: {something, soothing}

– Tập các từ edit distance = 2 là : {loathing, nothing, scathing, seething, smoothing, something, soothing, sorting}

Dùng **error mode**l này, kết hợp với language model, thuật toán auto-correction đơn giản có thể thực hiện như sau theo thứ tự sau:

– Nếu w là từ đúng chính tả, chọn w.

– Tìm từ c có P(c|x) cao nhất trong các từ sai 1 vị trí so với w (trong đó x là từ đứng trước .

– Tìm từ có P(c|x) cao nhất trong các từ sai 2 vị trí so với w.

Chỉ với language model và error model đơn giản như trên, các bộ auto correction cho tiếng Anh có thể đạt được độ chính xác trên 90% đối với các từ sai chính tả.

1. ***Thư viện TextBlob:***
2. **Giới thiệu thư viện:**

* **TextBlob** là thư viện của Python dùng để xử lý dữ liệu văn bản. Nó cung cấp một API đơn giản để đi sâu vào các tác vụ xử lý ngôn ngữ tự nhiên (NLP) phổ biến như gắn thẻ một phần lời nói, trích xuất cụm từ danh từ, phân tích cảm xúc, phân loại, dịch thuật, v.v
* **Những khả năng của** **TextBlob:**
* **TextBlob** cung cấp phương thức mã hóa khá đơn giản, chính là Tokenization. Tokenization là quá trình chia một đoạn văn bản lớn thành nhiều từ hoặc nhiều câu.
* Với **TextBlob**, việc chuyển đổi các từ về dạng gốc dễ dàng hơn bao giờ hết. Quá trình này được gọi là Lemmatization.
* Thư viện này cung cấp cho bạn giải pháp dễ dàng để gắn thẻ **Từ loại** **(PoS).** Tuy nhiên, tính năng này cũng đáng chú ý trong các thư viện NLP khác.
* Với **TextBlob**, bằng cách sử dụng các thủ tục đa phân hóa hoặc thủ tục đơn, bạn có thể chuyển văn bản của mình thành đơn hoặc nhiều văn bản.
* Ngoài ra, bạn có thể dễ dàng trích xuất các cụm danh từ khác nhau trong **TextBlob** bằng cách sử dụng thuộc tính cụm danh từ đơn giản.
* **TextBlob** cũng cung cấp cho bạn các chức năng đếm số lượng từ / cụm từ, chuyển đổi chữ hoa và chữ thường, sửa lỗi cú pháp, dịch, phát hiện N-gram và hơn thế nữa.
* Cách tải thư viện **TextBlob** phiên bản V0.8.0:
* Dành cho Python 3: pip3 install textblob.
* Dành cho Python 2: pip3 install textblob.

1. **Giới thiệu về Spelling Correction của TextBlob:**
2. Sử dụng **correct()** để chỉnh sửa chính tả cho đối tượng **TextBlob** có kiểu dữ liệu **BaseBlob** và trả về giá trị có kiểu dữ liệu **BaseBlob.**

>>> b = TextBlob("I havv goood speling!")

>>> print(b.correct())

I have good spelling!

1. Đối với đối tượng là Word(không phải đối tượng **TextBlob**) thì sửa dụng hàm **spellcheck().**

>>> from textblob import Word

>>> w = Word('falibility')

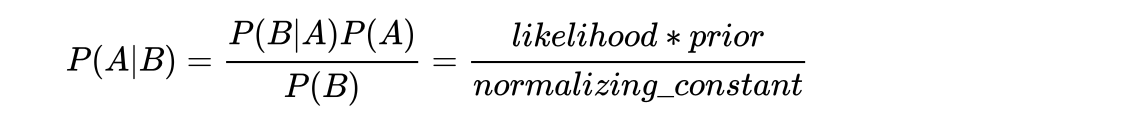
>>> w.spellcheck()

[('fallibility', 1.0)

1. **Spelling Correction** dựa trên “How to Write a Spellng Corrector” của Peter Norvig như một thư viện mẫu và có độ chính xác khoảng 70%.

1. **Những kết luận sau khi nghiên cứu bài “How to Write a Spellng Corrector” của Peter Norvig:**
2. **Xử lí ngôn ngữ theo theo lý thuyết xác suất:**

* Xử lí chính tả theo lý thuyết xác suất tức là chương trình sau khi có được chuỗi kí tự được nhập vào thì sẽ tính toán xác suất của tất cả cá từ có khả năng là từ chính xác và cho ra kết quả là chuỗi kí tự có xác suất là từ từ chính xác cao nhất.
* Để tính toán được xác suất thì ta sử dụng **Định lý Bayes**. **Định lý Bayes** là một kết quả của lý thuyết xác suất. Nó đề cập đến phân bố xác suất có điều kiện của biến ngẫu nhiên A với giả thiết thông tin về một biến khác B: phân bố xác suất có điều kiện của B khi biết A và phân bố xác suất của một mình A.
* **Định lý Bayes** cho phép tính xác suất xảy ra của một sự kiện ngẫu nhiên A khi biết được sự kiện liên quan B đã xảy ra. Xác suất được ký hiệu là P(A|B), đọc là “xác suất của A nếu có B”. Đại lượng này được gọi là [xác suất có điều kiện](https://vi.wikipedia.org/wiki/X%C3%A1c_su%E1%BA%A5t_c%C3%B3_%C4%91i%E1%BB%81u_ki%E1%BB%87n) hay sản xuất hậu nghiệm vì nó được rút ra từ giá trị cho của B học phụ thuộc vào giá trị đó.
* **Theo định lý Bayes**, xác suất xảy ra *A* khi biết *B* sẽ phụ thuộc vào 3 yếu tố:
* **Xác suất xảy ra *A*** của riêng nó, không quan tâm đến *B*. Ký hiệu là *P(A)* và đọc là xác suất của A. Đây được gọi là [xác suất biên duyên](https://vi.wikipedia.org/wiki/X%C3%A1c_su%E1%BA%A5t_bi%C3%AAn_duy%C3%AAn) hay [xác suất tiên nghiệm](https://vi.wikipedia.org/w/index.php?title=X%C3%A1c_su%E1%BA%A5t_ti%C3%AAn_nghi%E1%BB%87m&action=edit&redlink=1), nó là "[tiên nghiệm](https://vi.wikipedia.org/wiki/Ti%C3%AAn_nghi%E1%BB%87m)" theo nghĩa rằng nó không quan tâm đến bất kỳ thông tin nào về *B*.
* **Xác suất xảy ra *B*** của riêng nó, không quan tâm đến *A*. Ký hiệu là *P(B)* và đọc là "xác suất của B". Đại lượng này còn gọi là [hằng số chuẩn hóa](https://vi.wikipedia.org/w/index.php?title=H%E1%BA%B1ng_s%E1%BB%91_chu%E1%BA%A9n_h%C3%B3a&action=edit&redlink=1) (*normalising constant*), vì nó luôn giống nhau, không phụ thuộc vào sự kiện *A* đang muốn biết.
* **Xác suất xảy ra *B*** khi biết *A* xảy ra. Ký hiệu là *P(B|A)* và đọc là "xác suất của B nếu có A". Đại lượng này gọi là [khả năng](https://vi.wikipedia.org/w/index.php?title=H%C3%A0m_kh%E1%BA%A3_n%C4%83ng&action=edit&redlink=1) (*likelihood*) xảy ra *B* khi biết *A* đã xảy ra. Chú ý không nhầm lẫn giữa khả năng xảy ra *B* khi biết *A* và xác suất xảy ra *A* khi biết *B*.
* Khi biết ba đại lượng này xác suất của A khi biết B được cho bởi công thức:

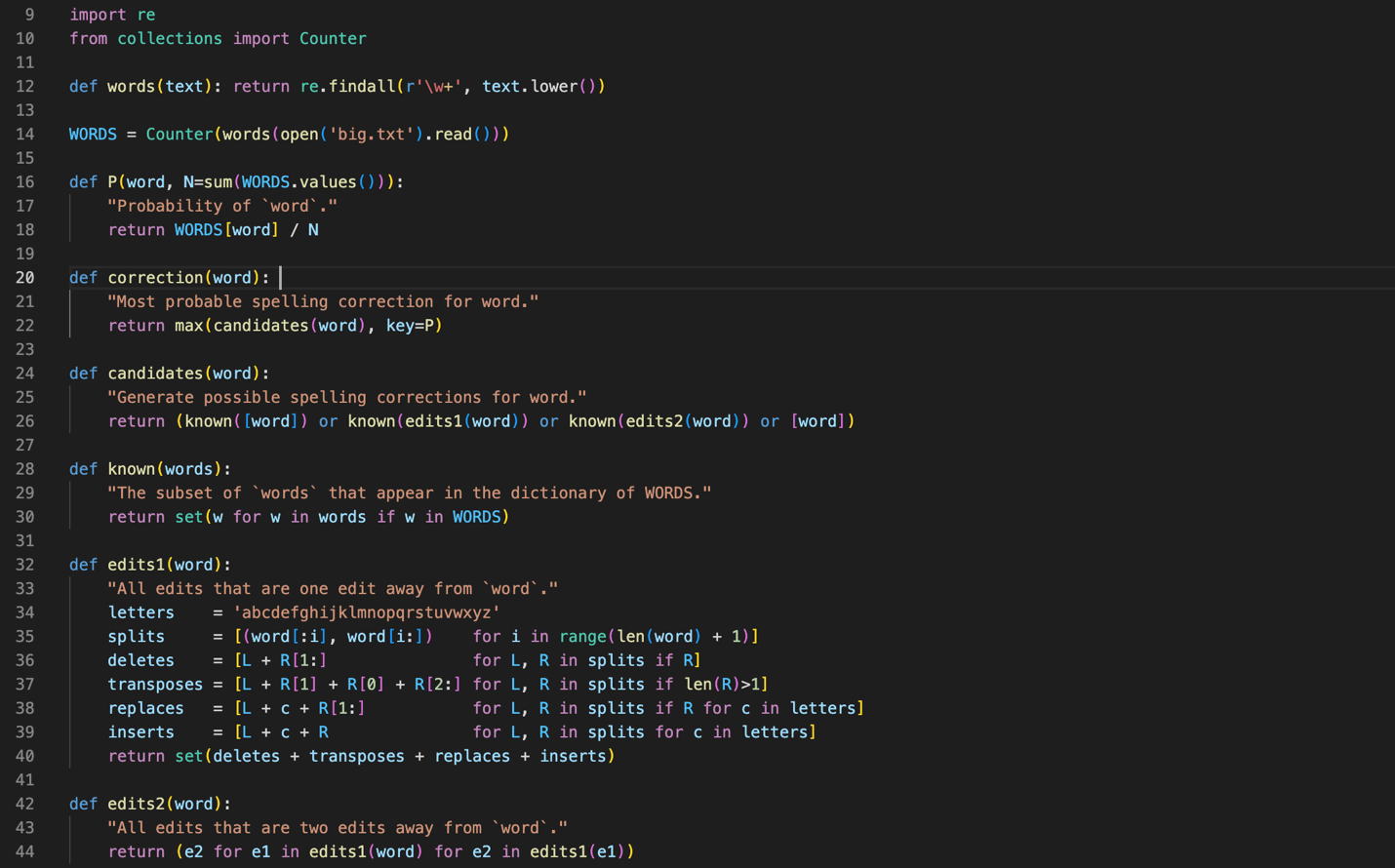


* Từ **Định lý Bayes** ta rút được công thức tính toán xác suất của từ chính xác khi có được từ nhập vào như sau:

**argmax*c ∈ candidates* P(*c*).P(*w*|*c*) / P(*w*)**

* **Hằng số chuẩn hoá P(w)** khi từ nhập từ bàn phím là không đổi nên khi xác suất của từ chính xác khi biết được từ nhập vài từ bàn phím phụ thuộc vào tích P(c).P(w|c).
* **4 phần của công thức này như sau:**
* **Selection mechanism**: [argmax](https://machinelearningmastery.com/argmax-in-machine-learning/): chọn từ có xác suất là từ chính xác cao nhất trong số các từ có khả năng là từ chính xác.
* **Candidate model**: *c ∈ candidates*:Tập hợp các từ chính xác ***c*** cần được xem xét.
* **Language model**: P(c):Xác xuất tiên nghiệm của từ chính xác ***c*** từ điển Tiếng Anh được dùng. VD: tần số xuất hiện của từ “the” trong file [biteg.txt](https://norvig.com/big.txt) khoảng 7% thì P(the)=0,07.
* **Error model:** P(c|w): xác suất lỗi chính tả có thể có khi biết từ chính xác. VD: P(the|the) sẽ rất cao nhưng P(theeexyz|the) rất thấp.

1. **Cách áp dụng lý thuyết xác suất vào chương trình kiểm tra lỗi chính tả của Peter Norvig:**

****

1. **Selection Mechanism:** Chọn biến max với đối số key là P(hàm P thực hiện tính xác suất)
2. **Candidate model:** Tạo hàm **edits1(word)** bao gồm các phép biến đổi cơ bản với một từ gồm xoá kí tự, chuyển vị, thay thế kí tự, thêm kí tự. Hàm trả về một set(có thể là một set lớn) gồm tất cả các chuỗi được chỉnh sửa sau một phép biến đổi cơ bản từ từ được cho. Để thu gọn set này lại một set nhỏ hơn nhờ vào việc lựa chọn những ta mà ta biết đó là gì(trong file big.txt), ta sẽ tạo hàm **known.** Ta cũng cần xem xét các chuỗi có đưuocj sau 2 phép biến đỏi cơ bản từ từ được cho nên sẽ tạo thêm hàm **edits2(word).**
3. **Language model:**Chúng ta có thể ước lượng P(c) bằng cách đếm số lần xuất hiện của từ đó trong file [big.txt](https://norvig.com/big.txt) (Đây là file chứa khoảng 1 triệu từ được lấy từ trích đoạn của cuốn sách “[Project Gutenberg](http://www.gutenberg.org/wiki/Main_Page)” và một danh sách các từ xuát hiện nhiều trong [Wiktionary](http://en.wiktionary.org/wiki/Wiktionary:Frequency_lists) và [British National Corpus](http://www.kilgarriff.co.uk/bnc-readme.html)). Hàm **words (text)** biến văn bản thành danh sách từ và **words(text)** giữ giá trị đếm số lần xuất hiện của từ chính xác, trong khi hàm **P**() tính xác suất.
4. **Error model:** Chúng ta chưa thể xác định xác suất các lỗi sẽ xuất hiện khi có từ chính xác vì chưa có dữ liệu nên sẽ làm cách gọn hơn là sẽ tạo ra danh sách không rỗng bao gồm các từ đã biết trong set edit1 **(known(edits1(..))** sẽ có khả năng xảy ra cao hơn vô hạn các từ đã biết trongset edits2(**known(edits2(…))** và ít khả năng xảy ra hơn các từ không cần chỉnh sửa(tức là từ đã chính xác sẵn). Như vậy khi tạo ra hàm này thì không cần nhân P(c) cho P(w|c) nữa vì các từ chính xác đưuocj chọn theo thứ tự ưu tiên sẽ có cùng xác suất xảy ra và chỉ cần so sánh P(c). Đó là công dụng của hàm **candidates(word).**
5. **Tỉ lệ chính xác của thuật toán:**

Sau khi code xong thuật toán thì ta cần kiểm tra chương trình để phát hiện thiếu xót và lỗi gặp phải. Ở đây chúng ta sẽ code một vài đơn vị test và chuẩn bị file [spell-testset1.txt](https://norvig.com/spell-testset1.txt) chứa các testcase để đo độ chính xác

def unit\_tests():

    assert correction('speling') == 'spelling'              # insert

    assert correction('korrectud') == 'corrected'           # replace 2

    assert correction('bycycle') == 'bicycle'               # replace

    assert correction('inconvient') == 'inconvenient'       # insert 2

    assert correction('arrainged') == 'arranged'            # delete

    assert correction('peotry') =='poetry'                  # transpose

    assert correction('peotryy') =='poetry'                 # transpose + delete

    assert correction('word') == 'word'                     # known

    assert correction('quintessential') == 'quintessential' # unknown

    assert words('This is a TEST.') == ['this', 'is', 'a', 'test']

    assert Counter(words('This is a test. 123; A TEST this is.')) == (

           Counter({'123': 1, 'a': 2, 'is': 2, 'test': 2, 'this': 2}))

    assert len(WORDS) == 32198

    assert sum(WORDS.values()) == 1115585

    assert WORDS.most\_common(10) == [

     ('the', 79809),

     ('of', 40024),

     ('and', 38312),

     ('to', 28765),

     ('in', 22023),

     ('a', 21124),

     ('that', 12512),

     ('he', 12401),

     ('was', 11410),

     ('it', 10681)]

    assert WORDS['the'] == 79809

    assert P('quintessential') == 0

    assert 0.07 < P('the') < 0.08

    return 'unit\_tests pass'

def spelltest(tests, verbose=False):

    "Run correction(wrong) on all (right, wrong) pairs; report results."

    import time

    start = time.time()

    good, unknown = 0, 0

    n = len(tests)

    for right, wrong in tests:

        w = correction(wrong)

        good += (w == right)

        if w != right:

            unknown += (right not in WORDS)

            if verbose:

                print('correction({}) => {} ({}); expected {} ({})'

                      .format(wrong, w, WORDS[w], right, WORDS[right]))

    dt = time.time() - start

    print('{:.0%} of {} correct ({:.0%} unknown) at {:.0f} words per second '

          .format(good / n, n, unknown / n, n / dt))

def Testset(lines):

    "Parse 'right: wrong1 wrong2' lines into [('right', 'wrong1'), ('right', 'wrong2')] pairs."

    return [(right, wrong)

            for (right, wrongs) in (line.split(':') for line in lines)

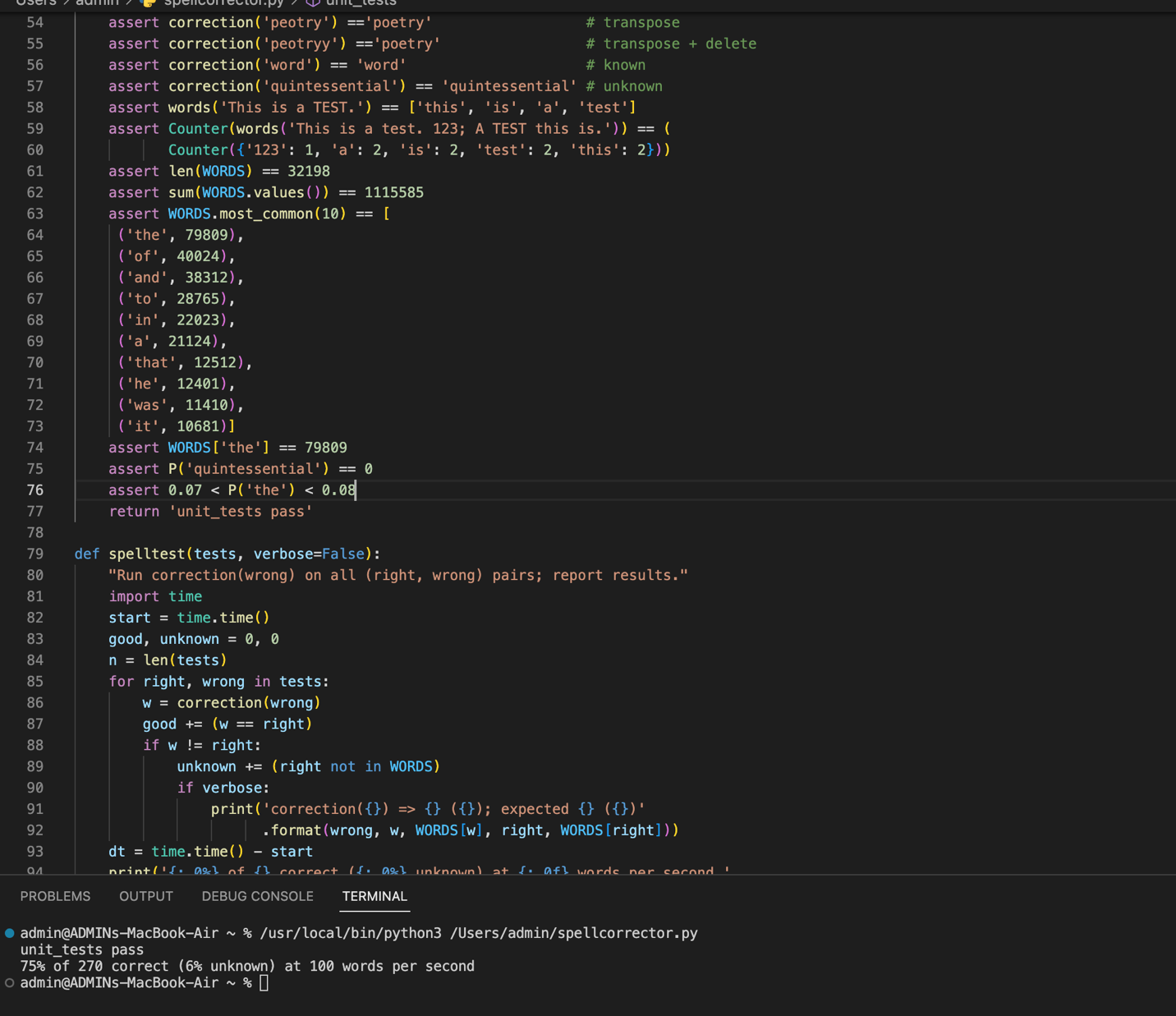
            for wrong in wrongs.split()]

if \_\_name\_\_ == '\_\_main\_\_':

    print(unit\_tests())

    spelltest(Testset(open('spell-testset1.txt')))

Sau khi chạy code thì cho ra kết quả:



Từ kết quả ta thấy được ktỉ lệ chính xác là **75%** và tốc độ kiểm tra chính tả là **100 từ/s**. Kết luận lại, thuật toán này đảm bảo về tốc độ và độ ngắn gọn nhưng vẫn chưa đạt đến độ chính xác mong muốn là **90%**, chậm hơn **15%** so với mong muốn.

1. **Ứng dụng thư viện TextBlob cho Spelling Corrector:**
2. Nhập lớp **TextBlob** từ thư viện **textblob** bằng cú pháp:

from textblob import TextBlob

1. Lập **hàm checkSpelling(**) để sửa chính tả

* **Bao gồm các bước:**
* Lấy từ nhập vào từ bàn phím:

a = text.get()

* Tạo đối đượng **TextBlob:**

b = TextBlob(a)

* Dùng cú pháp **correct()** để chỉnh sửa đối tượng thành đúng chính tả

b.correct()

* Vì lúc này vẫn là đối tượng **TextBlob** có kiểu **BaseBlob** nên cần chuyển về kiểu dữ liệu string để hiển thị kết quả

str(b.correct())

* Hiển thị kết quả sau khi chỉnh sửa

correctedText.set("The corrected word is: "+str(b.correct()))

1. ***Thư viện tkinter :***
2. **Giới thiệu thư viện Tkinter:**

* Tkinter là thư viện GUI tiêu chuẩn cho Python. Tkinter trong Python cung cấp một cách nhanh chóng và dễ dàng để tạo các ứng dụng GUI. Tkinter cung cấp giao diện hướng đối tượng cho bộ công cụ Tk GUI.
* **Cách tải thư viện Tkinter:**
* Dành cho Python 3: pip3 install tkinter.
* Dành cho Python 2: pip install tkinter.
* **Sau đây là các bước để tạo một ứng dụng Tkinter:**

1. Import Tkinter module.
2. Tạo cửa sổ chính của ứng dụng GUI.
3. Thêm một hoặc nhiều widget nói trên vào ứng dụng GUI.
4. Gọi vòng lặp sự kiện chính để các hành động có thể diễn ra t rên màn hình máy tính của người dùng.

* **Các widgets của thư viện tkinter Python:**
* **Button:** Tiện ích Button được sử dụng để hiển thị các nút trong ứng dụng
* **Canvas:** Sử dụng để vẽ các hình dạng, chẳng hạn như đường thẳng, hình bầu dục, đa giác và hình chữ nhật, trong ứng dụng của bạn
* **Checkbutton:** sử dụng để hiển thị một số tùy chọn dưới dạng hộp kiểm. Người dùng có thể chọn nhiều tùy chọn cùng một lúc.
* **Entry:** được sử dụng để hiển thị trường văn bản một dòng để chấp nhận các giá trị từ người dùng.
* **Frame:** được sử dụng như một widget vùng chứa để sắp xếp các widget khác.
* **Label:** được sử dụng để cung cấp chú thích một dòng cho các tiện ích con khác. Nó cũng có thể chứa hình ảnh.
* **Listbox:** được sử dụng để cung cấp danh sách các tùy chọn cho người dùng.
* **Menubutton:** được sử dụng để hiển thị các menu trong ứng dụng của bạn.
* **Menu:**được sử dụng để cung cấp các lệnh khác nhau cho người dùng. Các lệnh này được chứa bên trong Menubutton.
* **Message:**được sử dụng để hiển thị các trường văn bản nhiều dòng để chấp nhận các giá trị từ người dùng.
* **Radiobutton:** được sử dụng để hiển thị một số tùy chọn dưới dạng các nút radio. Người dùng chỉ có thể chọn một tùy chọn tại một thời điểm.
* **Scale:**  được sử dụng để cung cấp tiện ích con trượt.
* **Scrollbar:** được sử dụng để thêm khả năng cuộn vào các tiện ích con khác nhau, chẳng hạn như hộp danh sách.
* **Text:**  được sử dụng để hiển thị văn bản trong nhiều dòng
* **Toplevel:** được sử dụng để cung cấp một vùng chứa cửa sổ riêng biệt.
* **Spinbox:** Tiện ích Spinbox là một biến thể của tiện ích Tkinter Entry tiêu chuẩn, có thể được sử dụng để chọn từ một số giá trị cố định.
* **PanedWindow:** PanedWindow là một widget vùng chứa có thể chứa bất kỳ số lượng ngăn nào, được sắp xếp theo chiều ngang hoặc chiều dọc.
* **LabelFrame:** Labelframe là một tiện ích chứa đơn giản. Mục đích chính của nó là hoạt động như một bộ đệm hoặc vùng chứa cho các bố cục cửa sổ phức tạp
* **tkMessageBox:** Mô-đun này được sử dụng để hiển thị các hộp thông báo trong các ứng dụng của bạn.

1. **Ứng dụng thư viện Tkinter để tạo giao diện phần mềm Spelling Corrector:**
2. Import thư viện Tkinter:

from tkinter import \*

1. Tạo cửa sổ thao tác cho phần mềm:

#Creating the window

wn = Tk()

wn.title("DataFlair Spell Checker")

wn.geometry('500x250')

wn.config(bg='SlateGray1')

* wn = Tk(): Tạo cửa sổ.
* wn.title : Viết tiêu đề của khung phần mềm.
* wn.geometry: Thiết lập kích thước cửa sổ của phần mềm (tính theo đơn vị pixel).
* wn.config(): Thiết lập cấu hình của cửa sổ, trong đó “bg=’SlateGray’” là thuộc tính màu background.

1. Tạo biến kiểu StringVar để nhập và chỉnh sửa trong cửa sổ wn:

text=StringVar(wn)

correctedText =StringVar(wn)

* Ép kiểu biến **text** của hàm checkSpelling() thành kiểu StringVar để hàm hoạt động.
* Ép kiểu biến correctedText của hàm checkSpelling thành kiểu StringVar.

1. Tạo Label hiển thị tên của phần mềm:

Label(wn, text='DataFlair Spell Checker',bg='SlateGray1',

fg='gray30', font=('Times', 20,'bold')).place(x=130, y=10)

* **text:** hiển thị nội dung của văn bản chứa trong dấu ‘ ’ .
* **bg=’SlateGray1’** :chọn màu cho nhãn (slategray = màu xám đá phiến).
* **fg:** Nếu bạn đang hiển thị văn bản hoặc một bitmap trong nhãn này, tùy chọn này chỉ định màu của văn bản. Nếu bạn đang hiển thị một bitmap, đây là màu sẽ xuất hiện tại vị trí của các bit 1 trong bitmap.
* **font=('Times', 20,'bold')** : kiểu chữ hiển thị là Times New Roman, size chữ là 20, in đậm.
* **place(x=130 , y=10):** vị trí của label trong cửa sổ hiển thị(x,y tính theo đơn vị pixel).

1. Tạo label lấy input từ bàn phím:

* ***Tạo label hiển thị nhãn dán “Please enter the word” phía trên thẻ Entry***

Label(wn, text='Please enter the word',

bg='SlateGray1',font=('calibre',13,'normal'),

anchor="e",justify=CENTER).place(x=160, y=70)

* **textvariable:**  bổ sung văn bản được hiển thị trong tiện ích nhãn thành một biến điều khiển của lớp StringVar.
* **anchor:** dùng để căn lề văn bản.
* **justify=CENTER :** quy định canh lề của các dòng ( ở đây là căn giữa).
* **width:** quy định chiều rộng của nhãn (theo đơn vị số kí tự).
* **place(x=160, y=70**): đặt ở vị trí x=160, y=70.
* ***Tạo Entry để nhận từ nhập từ bàn phím:***

Entry(wn,textvariable=text, bg='white',

width=35,font=('calibre',13,'normal')).place(x=70,y=110)

* **Textvariable=text:** biến **text** thành giá trị nhạp vào từ bàn phím để hàm checkSpelling() có thể chỉnh sửa từ mà người dung muốn.
* **place(x=70,y=110)**: giá trị y của Entry phải thấp hơn y của thẻ Label hiển thị “Please enter the word” để đặt ở vị trí thấp hơn.

1. Tạo label hiển thị văn bản đã được sửa chữa:

#Label to show the correct word

opLabel=Label(wn,textvariable=correctedText,fg='white',anchor="e"

,font=('calibre',13,'normal'), justify=LEFT).place(x=132, y=150)

1. Tạo phím để thực hiện lệnh sửa chính tả:

#Button to do the spell check

Button(wn, text="Click Me", bg='SlateGray4',font=('calibre', 13),

command=checkSpelling).place(x=180, y=190)

* **Command=checkSpelling**: Hàm hoặc hành động sẽ được thực hiện khi nhấn vào nút bấm ( ở đây hoạt động được thực hiện là hàm checkSpelling được tạo ra ở trên.

1. Sử dụng hàm mainloop() để duy trì cửa sổ:

#Runs the window till it is closed by the user

wn.mainloop()

* **mainloop():** Gọi vòng lặp sự kiện chính để các hành động có thể diễn ra trên màn hình máy tính của người dùng.

1. ***Triển khai và đánh giá kết quả***

***I. Mô hình triển khai***

**1. Mô hình:**

Phần mềm sửa lỗi chính tả văn bản tiếng Anh DataFlair Spell Checker



**2. Công cụ:**

* Framework: Python
* IDE: Visual Studio Code

***II. Kết quả thực nghiệm***

Các bước thực hiện:

**B1:** Nhập input vào GUI đã bật.

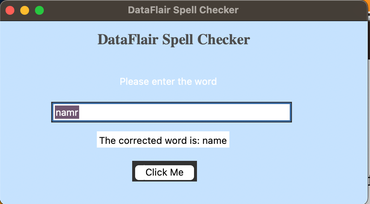
**B2:** Bấm vào nút “Click Me” để hiển thị kết quả.

**1. Kịch bản 1 – Sửa lỗi chính tả một từ**

* Mục tiêu: Kiểm tra tính chính xác của từ đã được sửa theo ý của người dùng.

**a**. **Test 1:**

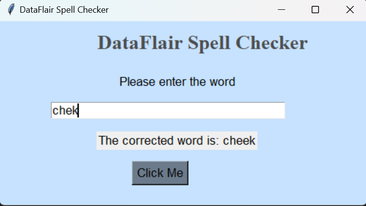
* Input: namr
* Output mong muốn: name
* Kết quả:



* Kết quả đưa ra đúng với mong muốn.

**b. Test 2:**

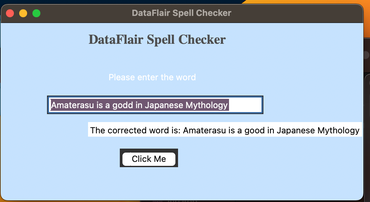
* Input: chek
* Output mong muốn: check
* Kết quả:



* Kết quả đưa ra tuy đúng về mặt chính tả, song chưa phải là kết quả mong muốn.

**2. Kịch bản 2 – Sửa lỗi chính tả văn bản**

* Mục tiêu: Kiểm tra tinh chính xác của từ đã được sửa theo ý của người dùng dựa trên ngữ cảnh.
* Input: Amaterasu is a godd in Japanese Mythology.
* Output mong muốn: Amaterasu is a god in Japanese Mythology.
* Kết quả:



* Kết quả đưa ra đúng về mặt chính tả, song không đúng về mặt ngữ cảnh và từ loại.

***III. Đánh giá***

* Qua kết quả thực nghiệm, có những nhận xét như sau:
* Kết quả nhận được từ chương trình có kết quả tương đối chính xác khoảng 75-80%.
* Ít tiêu tốn tài nguyên CPU nên tốc độ xử lí nhanh, xấp xỉ 100 từ/giây.
* Chỉ đưa ra được một kết quả đối với những từ có thể đưa ra nhiều kết quả. Ví dụ: chek -> check, cheek,…
* Chưa thể đưa ra những kết quả dựa trên ngữ cảnh mà chỉ phân tích theo thứ tự từng từ độc lập.
* Công việc trong tương lai là khắc phục những hạn chế đang có từ kết quả rút ra bao gồm:
* Training vào file big.txt thêm dữ liệu của từ chính xác.
* Tìm thêm dữ liệu về lỗi chính tả sẽ xuất hiện khi có chữ chính xác.
* Khắc phục hạn chế không phân biệt được ngữ cảnh.
  + Mục tiêu trong tương lai: Đạt được độ chính xác từ 90% trở lên.

***G. Kết luận***

Bài báo cáo này trình bày về các cơ sở lý thuyết và cách xây dựng một phần mềm sửa lỗi chính tả, và sau quá trình nghiên cứ, chúng em có những kết luận sau:

* Tính chính xác của kết quả phụ thuộc vào độ chính xác của dữ liệu được đưa vào file train.
* Các bước xử lí còn đơn giản nên chưa xử lí được văn bản dựa trên ngữ cảnh
* Do chưa có những dữ liệu về những lỗi chính tả có thể xuất hiện khi từ những từ chính xác nên hiệu quả chưa tối ưu

Chúng em rất mong nhận được những nhận xét, góp ý từ thầy để có thêm kinh nghiệm và kiến thức để hoàn thiện đề tài này.

* Các nguồn tham khảo

<https://digital.fpt.com.vn/dxarticles/xu-ly-ngon-ngu-tu-nhien.html#:~:text=X%E1%BB%AD%20l%C3%BD%20ng%C3%B4n%20ng%E1%BB%AF%20t%E1%BB%B1%20nhi%C3%AAn%20(natural%20language%20processing)%2C,th%E1%BB%B1c%20h%C3%B3a%20m%E1%BB%A5c%20ti%C3%AAu%20n%C3%A0y>.

<https://vinbigdata.com/xu-ly-ngon-ngu-tu-nhien-bai-toan-cong-cu-ban-nen-biet/>

<https://hoctructuyen123.net/xu-ly-ngon-ngu-tu-nhien-trong-tu-dong-sua-loi-chinh-ta-tieng-viet/>

<https://norvig.com/spell-correct.html?fbclid=IwAR0uLc_yen-Ub3QtU2-p5myRVQm8Ml5-U9K6ssEoXUfkT4JwJI_Yue86RBg>

<https://textblob.readthedocs.io/en/dev/>

<https://norvig.com/big.txt>

<https://norvig.com/spell-testset1.txt>

<https://techvidvan.com/tutorials/python-spelling-checker-corrector/>