Review the program according to instructions and add detailed content of missing items in the Job7 document

## 1.8. Hướng dẫn phát triển hệ thống

Chúng ta đã hoàn thành việc thiết lập phần cứng cơ bản và chuẩn bị ban đầu cho dự án Raspberry Pi. Bây giờ, hãy khám phá cách tạo, biên dịch và thực thi các chương trình để mở rộng hệ thống của chúng ta. Raspberry Pi hỗ trợ nhiều ngôn ngữ lập trình, nhưng trong bài viết này, chúng ta sẽ tập trung vào việc phát triển chương trình bằng Python.

### 1.8.1. Viết chương trình

Khi lập trình cho Raspberry Pi, một trình soạn thảo văn bản là công cụ không thể thiếu để tạo ra các chương trình dựa trên văn bản. Trình soạn thảo văn bản cho phép bạn tạo mới và chỉnh sửa các tập tin dựa trên ký tự. Trên hệ điều hành Raspbian, bạn có thể sử dụng các trình soạn thảo văn bản dạng Unix-like để lập trình. Hai trình soạn thảo phổ biến nhất là vi và GNU nano. Vi mạnh mẽ nhưng khó học, trong khi nano lại đơn giản, dễ sử dụng và tiện lợi.

#### 1.8.1.1. Trình soạn thảo Nano

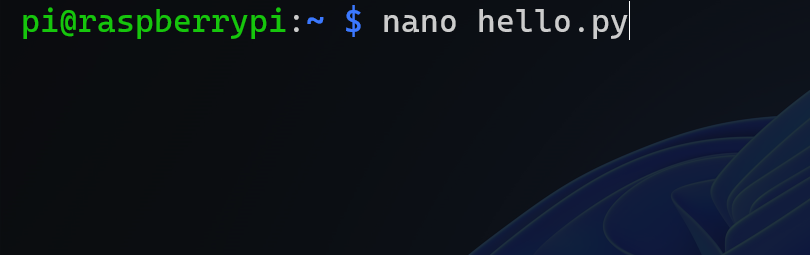
Trình soạn thảo Nano là một lựa chọn nhẹ nhàng và dễ sử dụng hơn so với các trình soạn thảo văn bản khác, tuy nhiên, nó không hỗ trợ nhiều tính năng như các trình soạn thảo khác. Nano cung cấp các màu sắc tích hợp giúp bạn dễ dàng nhận biết nội dung tài liệu. Để chỉnh sửa một tệp, bạn chỉ cần nhập “nano [tên tệp]” để chỉ định tệp cần mở.

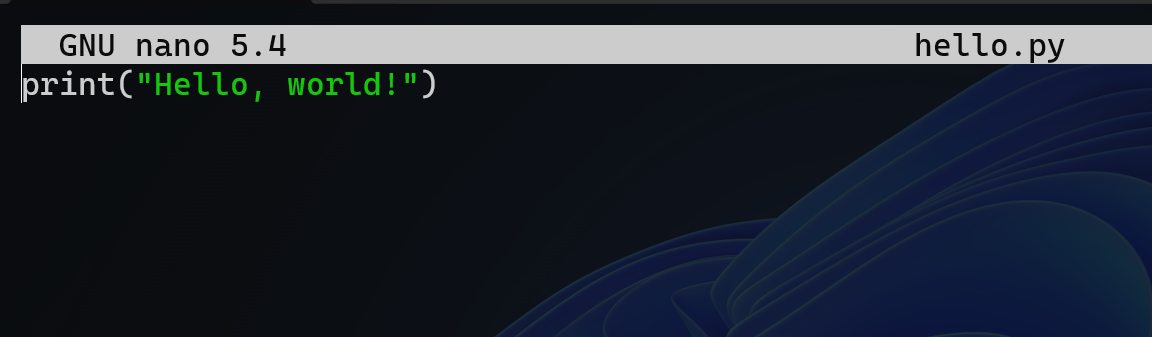
Nếu bạn nhập lệnh tương tự để tạo một tệp, tệp sẽ được tự động tạo với tên đã chỉ định nếu tệp đó chưa tồn tại. Bạn có thể tiếp tục lập trình trong tệp mới này, và sau khi hoàn thành, nhấn Ctrl + X để thoát khỏi trình soạn thảo. Nếu có bất kỳ thay đổi nào trong nội dung, Nano sẽ hỏi bạn có muốn lưu các thay đổi hay không bằng cách yêu cầu bạn nhập chữ cái đầu tiên của “Có” hoặc “Không”.

Nano còn có nhiều tính năng khác mà bạn có thể khám phá. Để biết thêm về các chức năng bổ sung, hãy nhấn phím Ctrl + G để mở menu trợ giúp.

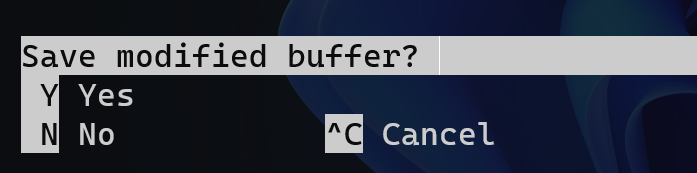
#### 1.8.1.2. Lập trình Python

Nhập lệnh sau để tạo tệp hello.py bằng trình chỉnh sửa nano và tạo mã nguồn sau.





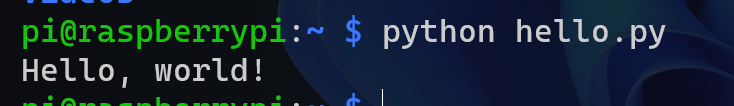
Khi quá trình viết mã nguồn hoàn tất, nhấn Ctrl + x để thoát khỏi trình soạn thảo và nhấn Yes- để lưu nội dung. Nhấn Enter lần nữa để thoát khỏi chương trình nano.



### 1.8.2. Dịch chương trình

Chương trình Python không yêu cầu quá trình xây dựng. Bạn chỉ cần chạy chương trình.

### 1.8.3. Chạy chương trình

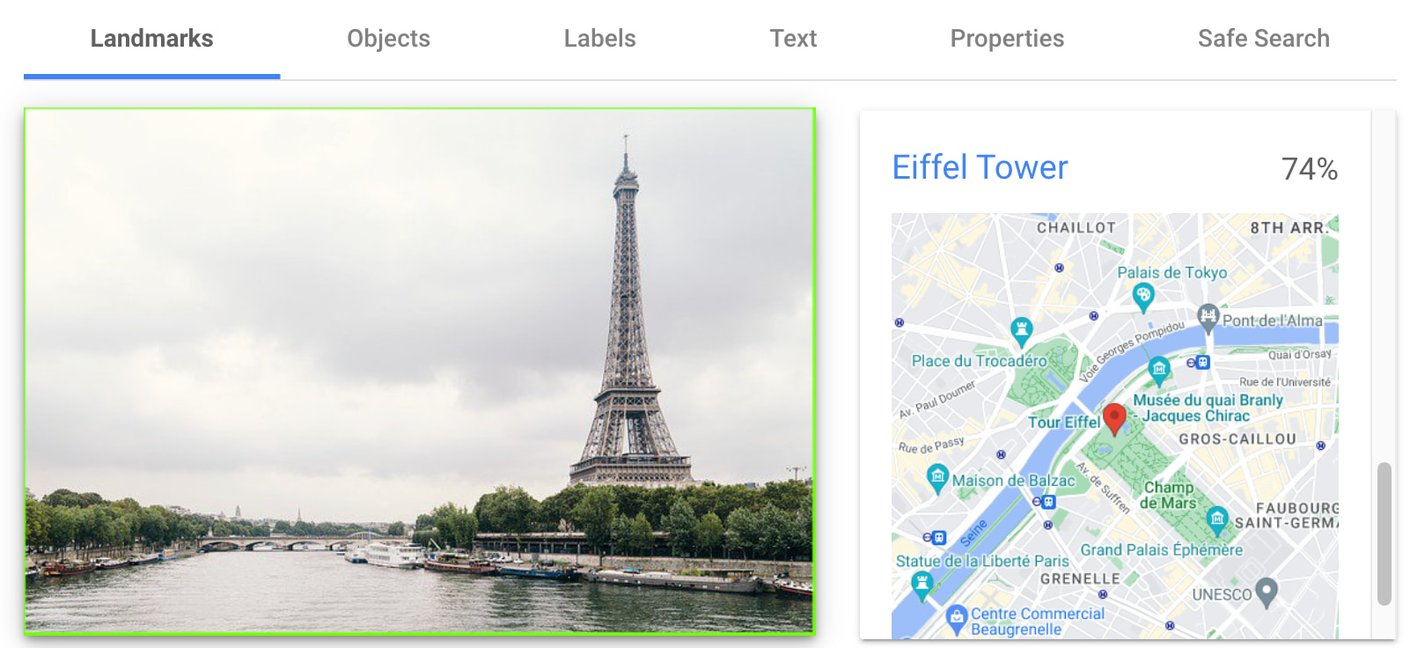
Khi một chương trình Python tạo một tệp chương trình, nó sẽ thực thi chương trình bằng cách nhập thông tin sau vào terminal (shell).

## 4.3. Tìm Landmark

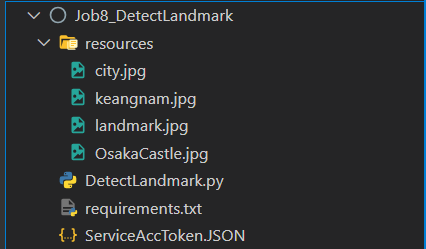
### 4.3.1. Mục đích đoạn mã

Google Vision API cung cấp chức năng phát hiện địa danh (Landmark Detection), cho phép phân tích hình ảnh để xác định các địa danh cụ thể như tòa nhà, đặc điểm tự nhiên và các địa điểm khác có thể nhận biết. API này hỗ trợ phát hiện địa danh trong hình ảnh từ xa và hình ảnh cục bộ.

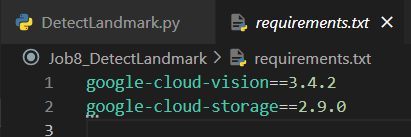
Trong hình ảnh bên trái bên dưới, API Cloud Vision đã phát hiện ra Tháp Eiffel, được hiển thị trong phản hồi trực quan. Không hiển thị trong hình ảnh này ở đây, nhưng cũng được phát hiện, là Pont de Bir-Hakeim (cây cầu) và Champs de Mars (công viên phía trước Tháp Eiffel).



### 4.3.2. Cấu trúc thư mục



### 4.3.3. Cài đặt các gói cần thiết



Sau đó sẽ gõ lệnh sau đây trên terminal: pip install -r requirements.txt

### 4.3.4. Hướng dẫn chi tiết

#### 4.3.4.1. Import các thư viện cần thiết

| from google.cloud import vision  import os, io |
| --- |

* **os** : Thư viện này được sử dụng để làm việc với hệ thống tệp và thư mục.
* **google.cloud.vision**: Thư viện này chứa các công cụ cho việc gửi hình ảnh đến Google Cloud Vision API và nhận kết quả.

#### 4.3.4.2. Thiết lập thông tin xác thực

| os.environ['GOOGLE\_APPLICATION\_CREDENTIALS'] = 'ServiceAccToken.JSON' |
| --- |

* Đoạn mã này sử dụng một tệp JSON chứa thông tin xác thực để gọi API của Google Cloud Vision.

#### 4.3.4.3. Khởi tạo client của Vision API

| client = vision.ImageAnnotatorClient() |
| --- |

* Đoạn mã này tạo một đối tượng client để tương tác với Google Cloud Vision API.

#### 4.3.4.4. Định nghĩa hàm detect\_landmarks(FILE\_PATH)

Hàm nhận vào 1 đường dẫn ảnh và đặt dưới biến FILE\_PATH. Đầu tiên, hàm mở tệp ảnh tại đường dẫn đã cho trong chế độ đọc nhị phân ("rb"). Cho phép đọc nội dung của tệp ảnh và lưu trữ nó trong biến content. Nội dung này sẽ được sử dụng để tạo một đối tượng hình ảnh (image) để gửi đến Vision API.

| with open(path, "rb") as image\_file:  content = image\_file.read() |
| --- |

Tạo một đối tượng hình ảnh (image) sử dụng nội dung của hình ảnh đã đọc. Đây là đối tượng mà bạn sẽ gửi đến API để xác định landmarks.

| image = vision.Image(content=content) |
| --- |

Gọi phương thức landmark\_detection của đối tượng client để gửi yêu cầu xác định landmarks trong hình ảnh. Kết quả sẽ được lưu trữ trong biến response.

| response = client.landmark\_detection(image=image) |
| --- |

Lấy danh sách các annotations của landmarks từ phản hồi của API và lưu trữ chúng trong biến landmarks.

| landmarks = response.landmark\_annotations |
| --- |

Duyệt qua từng landmark trong danh sách landmarks:

Duyệt qua các vị trí (locations) của landmark: Lấy tọa độ vĩ độ và kinh độ của landmark từ vị trí hiện tại và in ra tọa độ vĩ độ và kinh độ của landmark.

| for landmark in landmarks:  print(landmark.description)  for location in landmark.locations:  lat\_lng = location.lat\_lng  print(f"Latitude: {lat\_lng.latitude}")  print(f"Longitude: {lat\_lng.longitude}") |
| --- |

#### 4.3.4.5. Sử dụng hàm

| img\_path = './resources/keangnam.jpg'  detect\_landmarks(img\_path) |
| --- |

img\_path = './resources/keangnam.jpg': Đây là đường dẫn tới tệp hình ảnh bạn muốn xác định các landmarks. Hình ảnh này được đặt tại vị trí ./resources/keangnam.jpg trong cấu trúc thư mục của dự án của bạn.

detect\_landmarks(img\_path): Đoạn mã này gọi hàm detect\_landmarks với đường dẫn của hình ảnh ./resources/keangnam.jpg làm tham số đầu vào. Điều này sẽ thực hiện quá trình xác định landmarks trong hình ảnh và in ra thông tin về chúng.

### 4.3.5. Chi tiết mã nguồn

| import os  from google.cloud import vision  os.environ['GOOGLE\_APPLICATION\_CREDENTIALS'] = 'ServiceAccToken.JSON'  client = vision.ImageAnnotatorClient()  def detect\_landmarks(path):  with open(path, "rb") as image\_file:  content = image\_file.read()  image = vision.Image(content=content)  response = client.landmark\_detection(image=image)  landmarks = response.landmark\_annotations  print("Landmarks:")  for landmark in landmarks:  print(landmark.description)  for location in landmark.locations:  lat\_lng = location.lat\_lng  print(f"Latitude: {lat\_lng.latitude}")  print(f"Longitude: {lat\_lng.longitude}")    img\_path = './resources/keangnam.jpg'  detect\_landmarks(img\_path) |
| --- |

### 4.3.6. Kết quả nhận được



