**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN, ĐHQG-HCM**

**KHOA CÔNG NGHỆ THÔNG TIN**

---o0o---

Logo

Description automatically generated with medium confidence

**BÁO CÁO**

**ĐỒ ÁN LẬP TRÌNH SOCKET:**

**ĐIỀU KHIỂN TỪ XA BẰNG EMAIL**

**Thành viên nhóm MSSV**

Đỗ Phan Tuấn Đạt 22127057

Phạm Thành Đạt 22127064

Lê Hồ Phi Hoàng 22127123

Trần Nguyễn Minh Hoàng 22127131

**Lớp:** 22CLC02

**Môn học:** Mạng máy tính

**Học kỳ:** 1

**Năm học:** 2023-2024

# Mục lục

[I. Mục lục 2](#_Toc151382801)

[II. Môi trường làm việc, thư viện hỗ trợ 3](#_Toc151382802)

[1. Môi trường làm việc: 3](#_Toc151382803)

[2. Ngôn ngữ: 3](#_Toc151382804)

[3. Thư viện hỗ trợ 3](#_Toc151382805)

[III. Danh sách file trong source code 3](#_Toc151382806)

[IV. Phân tích, giải thích source code 4](#_Toc151382807)

[1. readMail.py 4](#_Toc151382808)

[2. sendMail.py 12](#_Toc151382809)

[3. keyLog.py 21](#_Toc151382810)

[4. appController.py 23](#_Toc151382811)

[5. processController.py 28](#_Toc151382812)

[6. powerController.py 30](#_Toc151382813)

[V. Hướng dẫn sử dụng 31](#_Toc151382814)

[1. Thiết lập 31](#_Toc151382815)

[a) Thiết bị được điều khiển 31](#_Toc151382816)

[b) Thiết bị điều khiển 31](#_Toc151382817)

[2. Ghi bàn phím 32](#_Toc151382818)

[3. Chụp màn hình 33](#_Toc151382819)

[VI. Đóng góp 33](#_Toc151382820)

[VII. Tài liệu tham khảo 33](#_Toc151382821)

# Môi trường làm việc, thư viện hỗ trợ

## Môi trường làm việc:

Visual Studio Code

## Ngôn ngữ:

Python

## Thư viện hỗ trợ

* **os**: Cho chức năng hệ điều hành
* **signal**: Gửi tín hiệu đến các tiến trình
* **psutil**: Quản lí tiến trình
* **subprocess**: Chạy lệnh bên ngoài
* **tabulate**: Định dạng dữ liệu thành bảng
* **bs4**: Phân tích cú pháp HTML và XML
* **imaplib**: Truy cập và thao tác với email trên server IMAP
* **email**: Làm việc với email
* **email.header**: Giải mã email có thể chứa ký tự không phải ASCII
* **threading:** Tạo và quản lý luồng
* **time:** Làm việc với thời gian
* **re:** Tìm kiếm và thao tác chuỗi dựa theo mẫu
* **smtplib:** Gửi email
* **PIL:** Chụp màn hình
* **windll:** import từ module **ctypes** để sử dụng các hàm của thư viện động Windows.

# Danh sách file trong source code

* readMail.py: Chứa các hàm để đọc email và thực hiện lệnh theo từ khóa trong nội dung mail
* sendMail.py: Chứa hàm để gửi email reply tương ứng với lệnh đã thực hiện
* keyLog.py: Chứa class keyLog dùng cho chưc năng ghi bàn phím
* appController.py: Chứa class AppController cùng các hàm dùng trong các hoạt động liên quan đến app
* processController.py: Chứa class ProcessController cùng các hàm dùng trong các hoạt động liên quan đến process
* powerController.py: Chứa các hàm để đăng xuất khỏi Window và tắt thiết bị

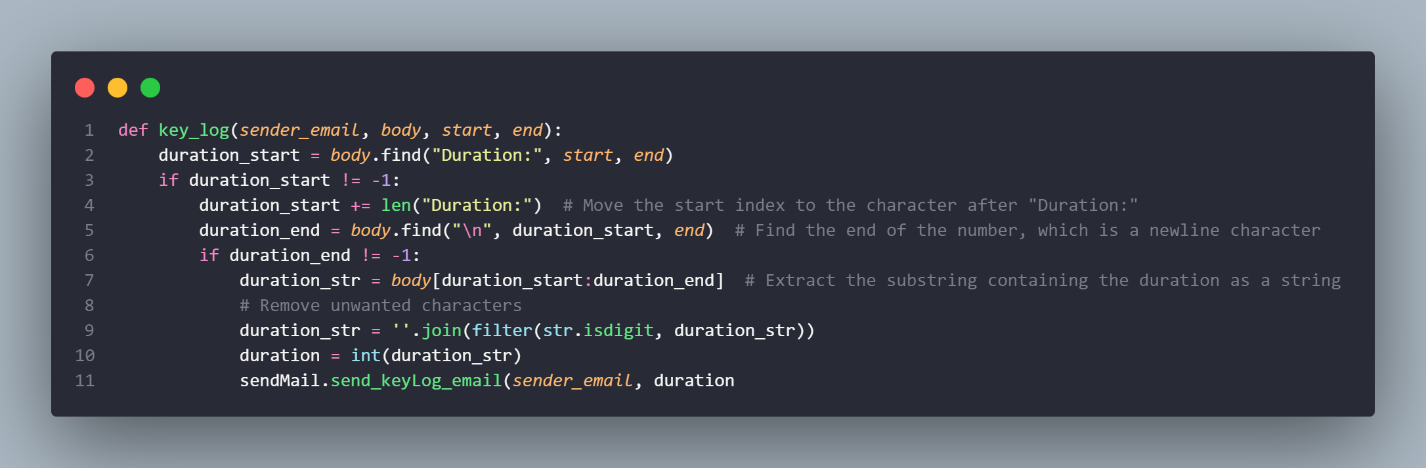
# Phân tích, giải thích source code

## readMail.py

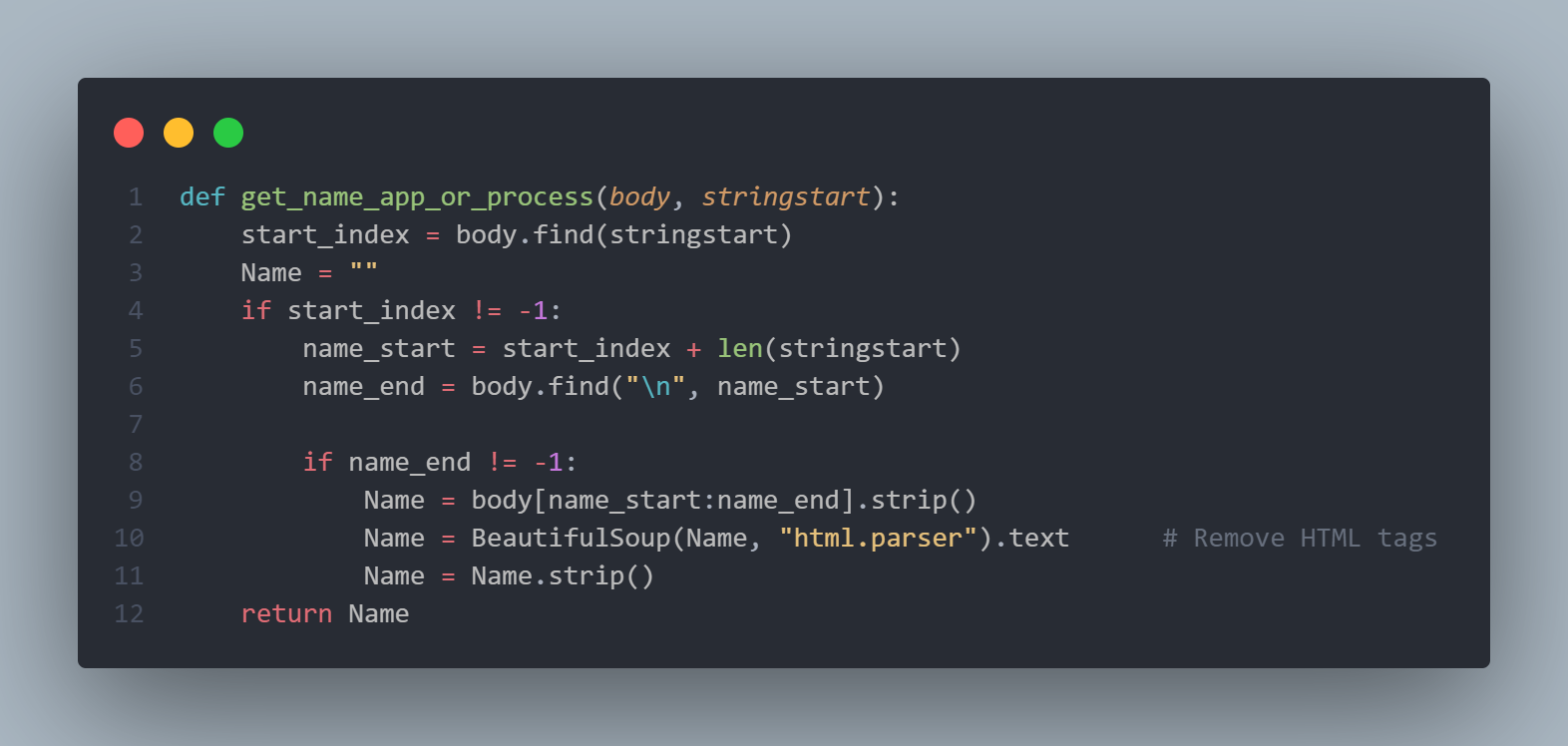
* ***extract\_email\_information:***



* **email\_message = email.message\_from\_bytes(raw\_email)**: Chuyển đổi dữ liệu email từ dạng byte sang đối tượng email bằng cách sử dụng **email.message\_from\_bytes**. Điều này cho phép ta tương tác với thông tin trong email như tiêu đề, người gửi và nội dung.
* **subject, encoding = decode\_header(email\_message["Subject"])[0]**: Trích xuất thông tin về tiêu đề của email. Hàm **decode\_header** được sử dụng để giải mã tiêu đề vì có thể có mã hóa đặc biệt trong tiêu đề. Nếu tiêu đề là dạng bytes, nó sẽ được giải mã bằng cách sử dụng encoding tương ứng.
* **from\_, encoding = decode\_header(email\_message["From"])[0]**: Trích xuất thông tin về người gửi của email. Tương tự như tiêu đề, thông tin người gửi cũng có thể được mã hóa và cần giải mã nếu là dạng bytes.
* **sender\_email = email\_message["From"]**: Lấy địa chỉ email của người gửi từ email.
* **match = re.search(r'<([^>]+)>', sender\_email)**: Sử dụng biểu thức chính quy để tìm kiếm địa chỉ email trong dấu ngoặc nhọn (**<...>**). Nếu tìm thấy, **sender\_email** sẽ được cập nhật với địa chỉ email được tìm thấy.
* **print("Subject:", subject)**, **print("From:", from\_)**: Hiển thị thông tin về tiêu đề và người gửi email. Đây là dữ liệu được in ra để kiểm tra khi phân tích email.
* **if email\_message.is\_multipart():**: Kiểm tra xem email có nhiều phần không. Nếu có, đoạn mã sẽ lặp qua từng phần để trích xuất nội dung.
* **for part in email\_message.walk():**: Lặp qua từng phần trong email.
* **content\_type = part.get\_content\_type()**: Lấy loại nội dung của phần email hiện tại.
* **content\_disposition = str(part.get("Content-Disposition"))**: Lấy thông tin về việc bố trí nội dung của phần email hiện tại.
* **try: ... except: ...**: Thử giải mã nội dung phần email hiện tại. Nếu không thể giải mã, nó sẽ bỏ qua lỗi và không làm gì cả.
* **if content\_type == "text/plain" and "attachment" not in content\_disposition: ...**: Kiểm tra xem nội dung của phần email hiện tại có phải là văn bản thuần túy không và không phải là file đính kèm. Nếu đúng, in ra nội dung văn bản này.
* **return sender\_email, body**: Trả về địa chỉ email của người gửi và nội dung của email (body) cho việc xử lý tiếp theo.
* ***key\_log(sender\_email, body, start, end):***

****

* **duration\_start = body.find("Duration:", start, end)**: Tìm vị trí xuất hiện đầu tiên của chuỗi "Duration:" trong phần của văn bản được chỉ định.
* **if duration\_start != -1: ...**: Kiểm tra xem chuỗi "Duration:" có được tìm thấy không.
* **duration\_start += len("Duration:")**: Di chuyển chỉ số bắt đầu đến ký tự sau "Duration:".
* **duration\_end = body.find("\n", duration\_start, end)**: Tìm vị trí kết thúc của số liệu, thường là ký tự xuống dòng "\n".
* **if duration\_end != -1: ...**: Kiểm tra xem kết thúc của số liệu có được tìm thấy không.
* **duration\_str = body[duration\_start:duration\_end]**: Trích xuất chuỗi chứa thời lượng như một chuỗi ký tự.
* **duration\_str = ''.join(filter(str.isdigit, duration\_str))**: Loại bỏ các ký tự không phải là số từ chuỗi số liệu.
* **duration = int(duration\_str)**: Chuyển chuỗi số liệu thành một số nguyên.
* **sendMail.send\_keyLog\_email(sender\_email, duration)**: Gửi email với thông tin về thời lượng và người gửi email (**sender\_email**).
* ***get\_name\_app\_or\_process(body, stringstart):***

****

Thực hiện việc trích xuất tên của ứng dụng hoặc quá trình từ văn bản dựa trên chuỗi bắt đầu được chỉ định (**stringstart**).

* **start\_index = body.find(stringstart)**: Tìm vị trí xuất hiện đầu tiên của chuỗi bắt đầu được chỉ định (**stringstart**) trong văn bản.
* **Name = ""**: Khởi tạo biến **Name** rỗng.
* **if start\_index != -1: ...**: Kiểm tra xem chuỗi bắt đầu được tìm thấy không.
* **name\_start = start\_index + len(stringstart)**: Xác định vị trí bắt đầu của tên ứng dụng hoặc quá trình.
* **name\_end = body.find("\n", name\_start)**: Tìm vị trí kết thúc của tên, thường là ký tự xuống dòng "\n".
* **if name\_end != -1: ...**: Kiểm tra xem kết thúc của tên có được tìm thấy không.
* **Name = body[name\_start:name\_end].strip()**: Trích xuất tên ứng dụng hoặc quá trình và loại bỏ khoảng trắng xung quanh nó.
* **Name = BeautifulSoup(Name, "html.parser").text**: Loại bỏ các thẻ HTML từ tên (nếu có).
* **return Name**: Trả về tên đã được xử lý.
* ***Các hàm end\_process(sender\_email, body), start\_process(sender\_email, body), start\_app(sender\_email, body), end\_app(sender\_email, body):***



Các hàm này sử dụng hàm **get\_name\_app\_or\_process()** để trích xuất tên app hoặc process từ văn bản, sau đó gửi email với thông tin tương ứng nếu tên được tìm thấy.

* ***process\_email:***



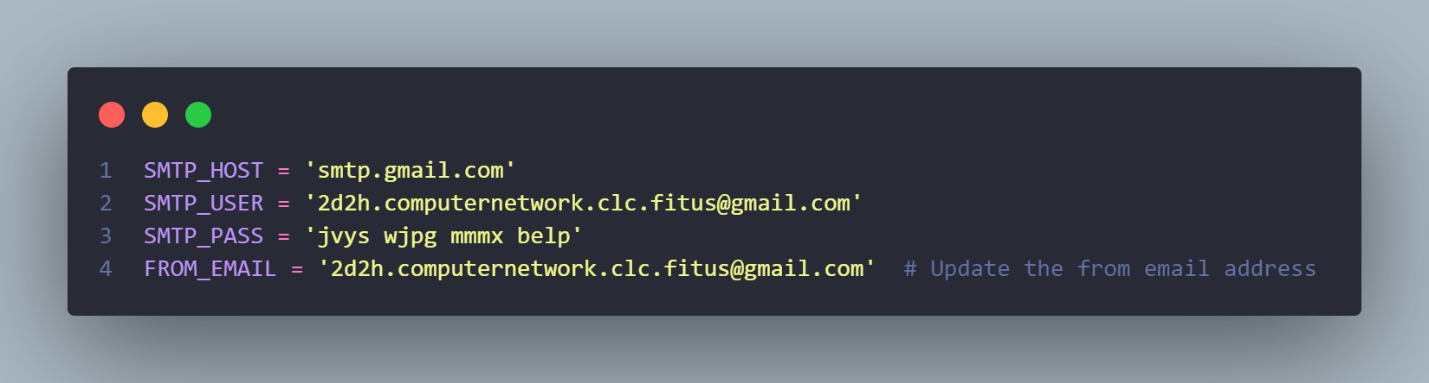
* **start = 0** và **end = len(body)**: Xác định vị trí bắt đầu và kết thúc trong chuỗi **body**.
* **if body.find("KeyLog", start, end) != -1: ...**: Kiểm tra xem chuỗi "KeyLog" có xuất hiện trong văn bản không. Nếu có, gọi hàm **key\_log()** để xử lý thông tin về Keylog và trả về chuỗi “KeyLog".
* **if body.find("ListApp", start, end) != -1: ...**: Tương tự, kiểm tra chuỗi "ListApp" có xuất hiện trong văn bản không. Nếu có, gửi email liệt kê các app đang hoạt động và trả về chuỗi “ListApp”.
* **if body.find("Screenshot", start, end) != -1: ...**: Kiểm tra chuỗi "Screenshot" có xuất hiện trong văn bản không. Nếu có, gửi email chứa ảnh chụp màn hình và trả về chuỗi “Screenshot”.
* **if body.find("Shut Down", start, end) != -1: ...**: Kiểm tra chuỗi "Shut Down" có xuất hiện trong văn bản không. Nếu có, thực hiện thao tác tắt máy.
* **if body.find("Log out", start, end) != -1: ...**: Kiểm tra chuỗi "Log out" có xuất hiện trong văn bản không. Nếu có, thực hiện thao tác đăng xuất.
* **if body.find("ListProcess", start, end) != -1: ...**: Kiểm tra chuỗi "ListProcess" có xuất hiện trong văn bản không. Nếu có, gửi email liệt kê các process đang hoạt động và trả về chuỗi “ListProcess”.
* **if body.find("StartProcess", start, end) != -1: ...**: Kiểm tra chuỗi "StartProcess" có xuất hiện trong văn bản không. Nếu có, gọi hàm **start\_process()** để xử lý thông tin khởi động một process và trả về chuỗi “StartProcess”.
* **if body.find("EndProcess", start, end) != -1: ...**: Kiểm tra chuỗi "EndProcess" có xuất hiện trong văn bản không. Nếu có, gọi hàm **end\_process()** để xử lý thông tin kết thúc một process và trả về chuỗi “EndProcess”.
* **if body.find("StartApp", start, end) != -1: ...**: Kiểm tra chuỗi "StartApp" có xuất hiện trong văn bản không. Nếu có, gọi hàm **start\_app()** để xử lý thông tin khởi động một app và trả về chuỗi “StartApp”.
* **if body.find("EndApp", start, end) != -1: ...**: Kiểm tra chuỗi "EndApp" có xuất hiện trong văn bản không. Nếu có, gọi hàm **end\_app()** để xử lý thông tin kết thúc một app và trả về chuỗi “EndApp”.
* **if body.find("Die", start, end) != -1: …**: Kiểm tra chuỗi “Die” có xuất hiện trong văn bản không. Nếu có, trả về chuỗi “Die”.
* **else: …**: Trường hợp mặc định, Nếu không có từ khóa nào được tìm thấy, hàm trả về chuỗi "Invalid".
* ***read\_email****:*

Thực hiện việc đăng nhập vào tài khoản Gmail, tìm và lấy email chưa đọc, rồi xử lý thông tin từ email cuối cùng được tìm thấy.



* **username = "2d2h.computernetwork.clc.fitus@gmail.com" và password = "jvys wjpg mmmx belp"**: Tên người dùng và mật khẩu của tài khoản Gmail được sử dụng để đăng nhập và truy cập email.
* **imap = imaplib.IMAP4\_SSL("imap.gmail.com")**: Tạo một kết nối SSL với máy chủ IMAP của Gmail để lấy email.
* **imap.login(username, password)**: Xác thực với tên người dùng và mật khẩu đã cung cấp.
* **imap.select("inbox")**: Chọn hộp thư "inbox" để thao tác.
* **status, email\_ids = imap.search(None, 'UNSEEN')**: Tìm kiếm các email chưa đọc trong hộp thư.
* **if not email\_ids[0]: ...**: Kiểm tra xem có email chưa đọc nào không. Nếu không tìm thấy, in ra thông báo "No unread emails found." và trả về chuỗi “Error”.
* **latest\_email\_id = email\_ids[0].split()[-1]**: Chọn ID của email cuối cùng trong danh sách các email chưa đọc.
* **status, email\_data = imap.fetch(latest\_email\_id, "(RFC822)")**: Lấy dữ liệu của email cuối cùng theo ID.
* **sender\_email, body = extract\_email\_information(email\_data[0][1])**: Trích xuất thông tin về người gửi và nội dung email từ dữ liệu email đã lấy.
* **command = process\_email(sender\_email, body)**: Gọi hàm **process\_email()** để xử lý thông tin từ email như các yêu cầu được đưa ra trong nội dung email và trả về lệnh cần thực hiện.
* **imap.logout()**: Đóng kết nối với máy chủ IMAP, đồng thời thoát khỏi tài khoản email.
* **return command**: Trả về lệnh đã nhận được sau khi xử lý email
* ***if \_\_name\_\_ == "\_\_main\_\_": …***: Được sử dụng để kiểm tra xem mã Python đang được thực thi trực tiếp từ chương trình chính hay không. Nếu đoạn mã được chạy trực tiếp (không phải là một module được nhập vào từ một chương trình khác), thì các dòng code bên dưới **if \_\_name\_\_ == "\_\_main\_\_":** sẽ được thực thi. Trong trường hợp này, hàm **read\_mail()** sẽ được gọi.

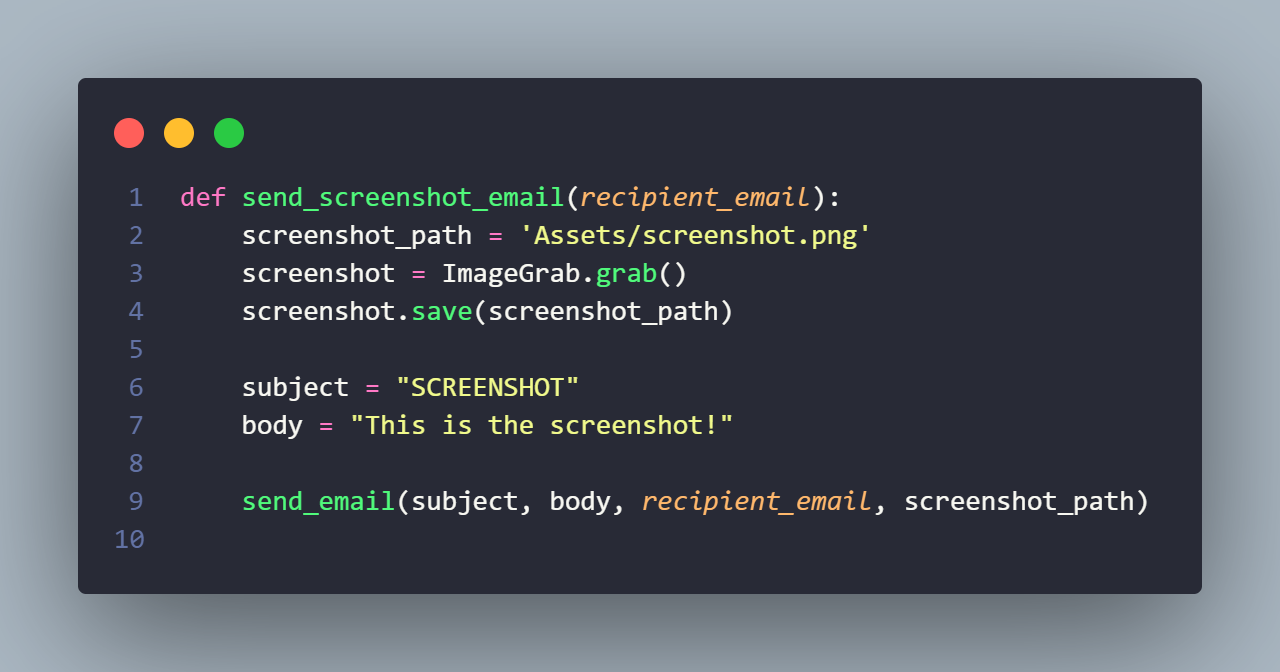
## sendMail.py



* **SMTP\_HOST = 'smtp.gmail.com'**: Đây là địa chỉ máy chủ SMTP của Gmail. Khi gửi email, phải kết nối đến máy chủ này để chuyển thư đi.
* **SMTP\_USER = '2d2h.computernetwork.clc.fitus@gmail.com'**: Đây là địa chỉ email của người gửi, sẽ được sử dụng để đăng nhập vào máy chủ SMTP.
* **SMTP\_PASS = 'jvys wjpg mmmx belp'**: Đây là mật khẩu của địa chỉ email người gửi, được sử dụng để xác thực khi đăng nhập vào máy chủ SMTP. Lưu ý rằng việc lưu mật khẩu trực tiếp trong mã nguồn không được khuyến khích do có thể tạo ra rủi ro bảo mật. Trong môi trường thực tế, nên sử dụng biện pháp bảo mật cao hơn, chẳng hạn như lưu mật khẩu trong biến môi trường hoặc sử dụng các dịch vụ quản lý mật khẩu.
* **FROM\_EMAIL = '2d2h.computernetwork.clc.fitus@gmail.com'**: Đây là địa chỉ email người gửi. Nếu cần thay đổi địa chỉ email người gửi, bạn có thể chỉnh sửa giá trị của biến này.
* ***sendMail:***



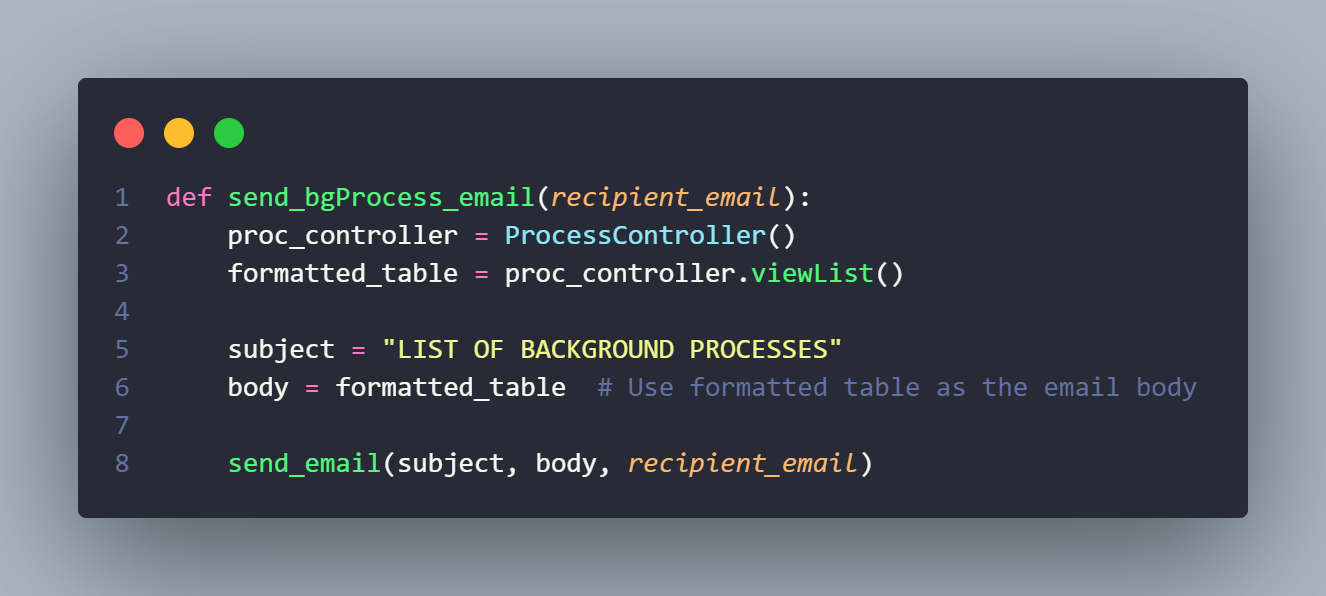
* **msg = MIMEMultipart()**: Tạo một đối tượng MIMEMultipart, là một đối tượng MIME sử dụng để xây dựng các thông điệp email với nhiều phần khác nhau.
* **msg['From'] = FROM\_EMAIL**: Thiết lập trường "From" của thông điệp email bằng địa chỉ email người gửi (**FROM\_EMAIL**).
* **msg['To'] = recipient\_email**: Thiết lập trường "To" của thông điệp email bằng địa chỉ email của người nhận (**recipient\_email**).
* **msg['Subject'] = subject**: Thiết lập trường "Subject" của thông điệp email bằng tiêu đề được truyền vào hàm (**subject**).
* **text = MIMEText(body, 'plain')**: Tạo một đối tượng MIMEText chứa nội dung văn bản của email (**body**) với định dạng 'plain'.
* **msg.attach(text)**: Đính kèm đối tượng MIMEText vào đối tượng MIMEMultipart để thêm nội dung văn bản vào thông điệp email.
* **if attachment\_path:**: Kiểm tra xem có đường dẫn đính kèm (**attachment\_path**) không.
* **with open(attachment\_path, 'rb') as attachment\_file:**: Mở tệp đính kèm ở chế độ đọc dưới dạng nhị phân (**'rb'**).
* **attachment = MIMEImage(attachment\_file.read(), name='screenshot.png')**: Tạo một đối tượng MIMEImage từ dữ liệu của tệp đính kèm, và đặt tên tệp là 'screenshot.png'.
* **msg.attach(attachment)**: Đính kèm đối tượng MIMEImage vào đối tượng MIMEMultipart để thêm hình ảnh hoặc bất kỳ đính kèm nào khác vào thông điệp email.
* **smtp\_server = SMTP\_SSL(SMTP\_HOST, port=SMTP\_SSL\_PORT)**: Tạo một đối tượng SMTP\_SSL để kết nối đến máy chủ SMTP của Gmail qua SSL.
* **smtp\_server.set\_debuglevel(1)**: Thiết lập chế độ debug cho đối tượng SMTP để hiển thị tất cả tương tác với máy chủ SMTP.
* **smtp\_server.login(SMTP\_USER, SMTP\_PASS)**: Đăng nhập vào máy chủ SMTP bằng tên người dùng và mật khẩu.
* **smtp\_server.sendmail(FROM\_EMAIL, recipient\_email, msg.as\_string())**: Gửi thông điệp email đến người nhận.
* **smtp\_server.quit()**: Ngắt kết nối từ máy chủ SMTP sau khi đã gửi email xong.
* ***send\_screenshot\_email:***



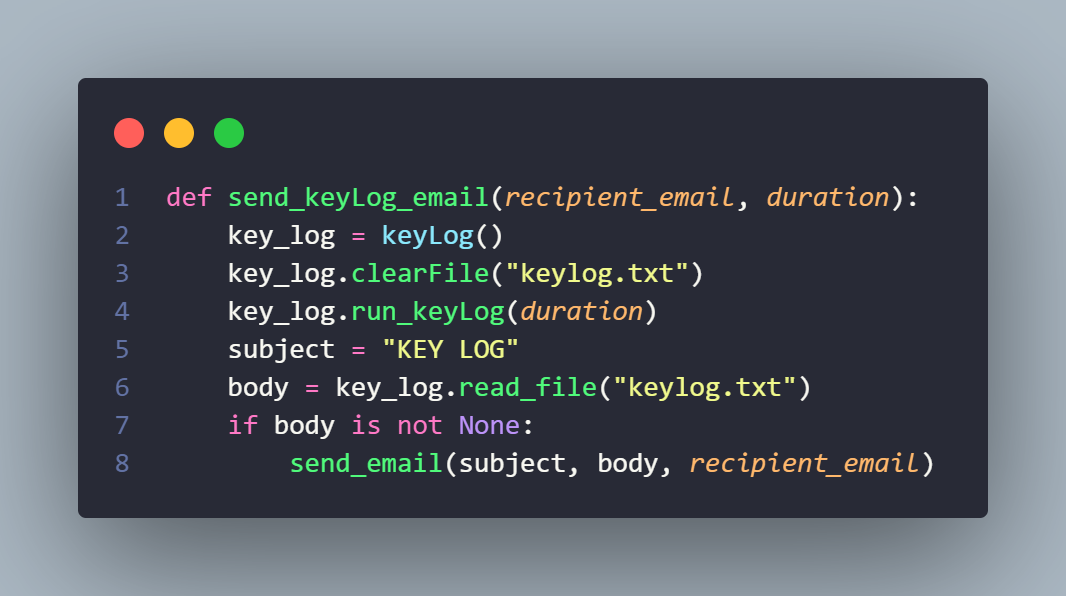
* **screenshot\_path = 'Assets/screenshot.png'**: Xác định đường dẫn nơi mà ảnh chụp màn hình sẽ được lưu trữ.
* **screenshot = ImageGrab.grab()**: Sử dụng hàm **grab()** từ thư viện PIL để chụp màn hình và lưu vào biến **screenshot**.
* **screenshot.save(screenshot\_path)**: Lưu ảnh chụp màn hình vào đường dẫn đã xác định ở bước 1.
* **subject = "SCREENSHOT"**: Đặt chủ đề của email là "SCREENSHOT".
* **body = "This is the screenshot!"**: Thiết lập nội dung văn bản của email là "This is the screenshot!".
* **send\_email(subject, body, recipient\_email, screenshot\_path)**: Gọi hàm **send\_email** để gửi email với chủ đề, nội dung và ảnh chụp màn hình đã được xác định.
* ***send\_process\_email:***



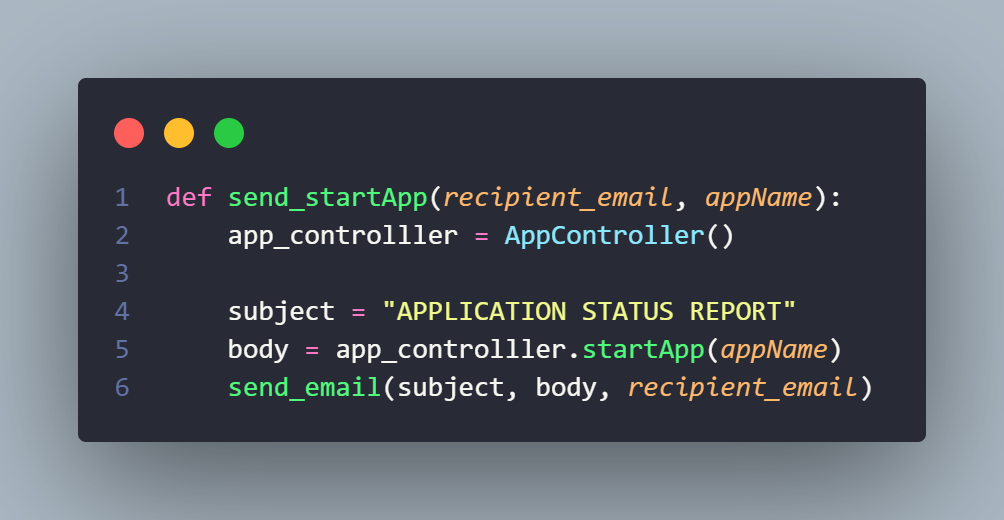
* **app\_controller = AppController()**: Tạo một đối tượng của lớp **AppController**. Lớp này chứa các phương thức để quản lý ứng dụng trên hệ thống.
* **formatted\_table = app\_controller.viewList()**: Gọi phương thức **viewList** từ đối tượng **app\_controller** để lấy danh sách các ứng dụng và định dạng nó thành một bảng.
* **subject = "LIST OF APPLICATIONS"**: Đặt chủ đề của email là "LIST OF APPLICATIONS".
* **body = formatted\_table**: Sử dụng bảng đã được định dạng như là nội dung email.
* **send\_email(subject, body, recipient\_email)**: Gọi hàm **send\_email** để gửi email với chủ đề, nội dung và không có tệp đính kèm (ảnh chụp màn hình, v.v.) cho địa chỉ email đã chỉ định.
* ***send\_bgProcess\_email:***



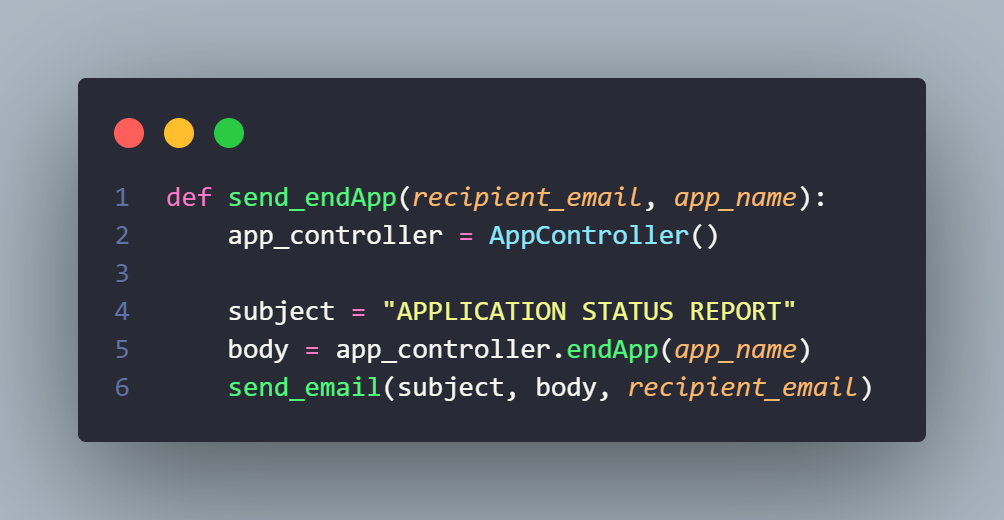
* **proc\_controller = ProcessController()**: Tạo một đối tượng của lớp **ProcessController**. Lớp này chứa các phương thức để quản lý các tiến trình nền trên hệ thống.
* **formatted\_table = proc\_controller.viewList()**: Gọi phương thức **viewList** từ đối tượng **proc\_controller** để lấy danh sách các tiến trình nền và định dạng nó thành một bảng.
* **subject = "LIST OF BACKGROUND PROCESSES"**: Đặt chủ đề của email là "LIST OF BACKGROUND PROCESSES".
* **body = formatted\_table**: Sử dụng bảng đã được định dạng như là nội dung email.
* **send\_email(subject, body, recipient\_email)**: Gọi hàm **send\_email** để gửi email với chủ đề, nội dung và không có tệp đính kèm (ảnh chụp màn hình, v.v.) cho địa chỉ email đã chỉ định.Top of Form
* ***send\_keyLog\_email:***



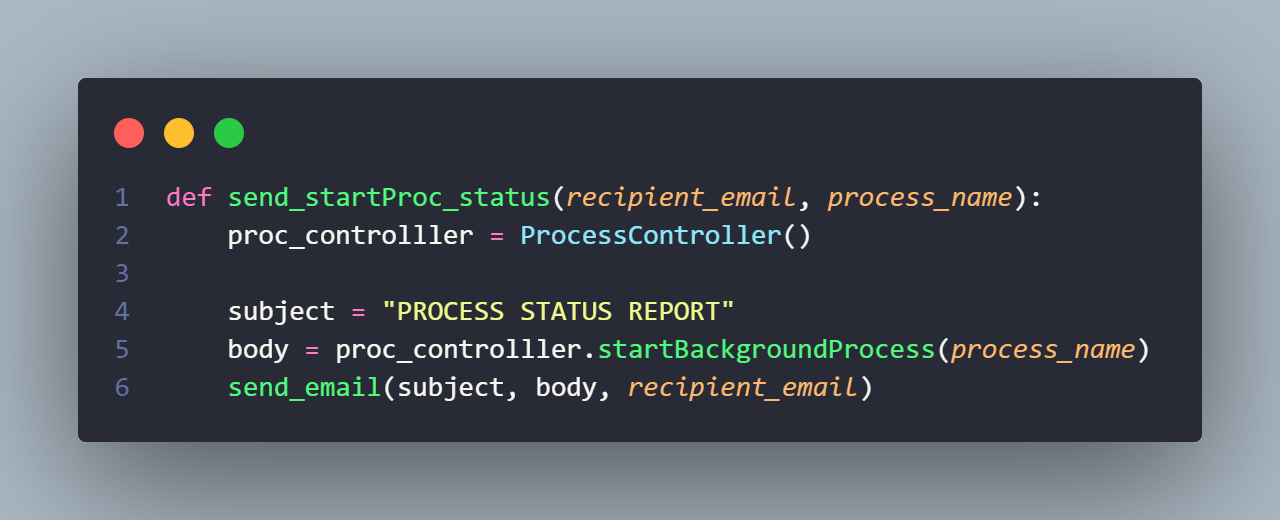
* **key\_log = keyLog()**: Tạo một đối tượng của lớp **keyLog**. Lớp này chứa các phương thức để ghi và đọc dữ liệu.
* **key\_log.clearFile("keylog.txt")**: Gọi phương thức **clearFile** của đối tượng **key\_log** để xóa nội dung của tệp "keylog.txt". Điều này giúp đảm bảo rằng chỉ có thông tin mới nhất được lưu trữ.
* **key\_log.run\_keyLog(duration)**: Gọi phương thức **run\_keyLog** để bắt đầu ghi lại các sự kiện từ bàn phím trong một khoảng thời gian **duration**.
* **subject = "KEY LOG"**: Đặt chủ đề của email là "KEY LOG".
* **body = key\_log.read\_file("keylog.txt")**: Gọi phương thức **read\_file** của đối tượng **key\_log** để đọc nội dung từ tệp "keylog.txt" và lưu trữ nó trong biến **body**.
* **if body is not None:**: Kiểm tra xem **body** có giá trị hay không.
* **send\_email(subject, body, recipient\_email)**: Gọi hàm **send\_email** để gửi email với chủ đề "KEY LOG" và nội dung là nội dung đã đọc từ tệp "keylog.txt" tới địa chỉ email đã chỉ định.
* ***send\_startApp:***



* **app\_controller = AppController()**: Tạo một đối tượng của lớp **AppController**. Lớp này chứa các phương thức để quản lý và kiểm soát các ứng dụng trên hệ thống.
* **subject = "APPLICATION STATUS REPORT"**: Đặt chủ đề của email là "APPLICATION STATUS REPORT".
* **body = app\_controller.startApp(appName)**: Gọi phương thức **startApp** của đối tượng **app\_controller** với **appName** làm đối số để lấy thông tin về việc khởi động một ứng dụng và lưu trữ thông tin này trong biến **body**.
* **send\_email(subject, body, recipient\_email)**: Gọi hàm **send\_email** để gửi email với chủ đề "APPLICATION STATUS REPORT" và nội dung là thông tin về việc khởi động một ứng dụng tới địa chỉ email đã chỉ định.
* ***send\_endApp:***



* **app\_controller = AppController()**: Tạo một đối tượng của lớp **AppController**. Lớp này chứa các phương thức để quản lý và kiểm soát các ứng dụng trên hệ thống.
* **subject = "APPLICATION STATUS REPORT"**: Đặt chủ đề của email là "APPLICATION STATUS REPORT".
* **body = app\_controller.endApp(app\_name)**: Gọi phương thức **endApp** của đối tượng **app\_controller** với **app\_name** làm đối số để lấy thông tin về việc kết thúc một ứng dụng và lưu trữ thông tin này trong biến **body**.
* **send\_email(subject, body, recipient\_email)**: Gọi hàm **send\_email** để gửi email với chủ đề "APPLICATION STATUS REPORT" và nội dung là thông tin về việc kết thúc một ứng dụng tới địa chỉ email đã chỉ định.
* ***send\_startProc\_status:***



* **proc\_controller = ProcessController()**: Tạo một đối tượng của lớp **ProcessController**. Lớp này chứa các phương thức để quản lý và kiểm soát các tiến trình trên hệ thống.
* **subject = "PROCESS STATUS REPORT"**: Đặt chủ đề của email là "PROCESS STATUS REPORT".
* **body = proc\_controller.startBackgroundProcess(process\_name)**: Gọi phương thức **startBackgroundProcess** của đối tượng **proc\_controller** với **process\_name** làm đối số để lấy thông tin về việc bắt đầu một tiến trình nền và lưu trữ thông tin này trong biến **body**.
* **send\_email(subject, body, recipient\_email)**: Gọi hàm **send\_email** để gửi email với chủ đề "PROCESS STATUS REPORT" và nội dung là thông tin về việc bắt đầu một tiến trình nền tới địa chỉ email đã chỉ định.
* ***send\_endProc\_status:***

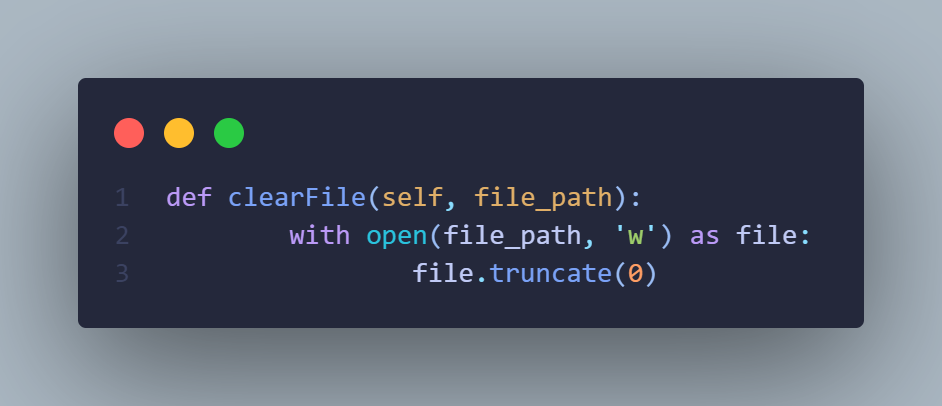


* **proc\_controller = ProcessController()**: Tạo một đối tượng của lớp **ProcessController**. Lớp này chứa các phương thức để quản lý và kiểm soát các tiến trình trên hệ thống.
* **subject = "PROCESS STATUS REPORT"**: Đặt chủ đề của email là "PROCESS STATUS REPORT".
* **body = proc\_controller.endProcess(process\_name)**: Gọi phương thức **endProcess** của đối tượng **proc\_controller** với **process\_name** làm đối số để lấy thông tin về việc kết thúc một tiến trình và lưu trữ thông tin này trong biến **body**.
* **send\_email(subject, body, recipient\_email)**: Gọi hàm **send\_email** để gửi email với chủ đề "PROCESS STATUS REPORT" và nội dung là thông tin về việc kết thúc một tiến trình tới địa chỉ email đã chỉ định.

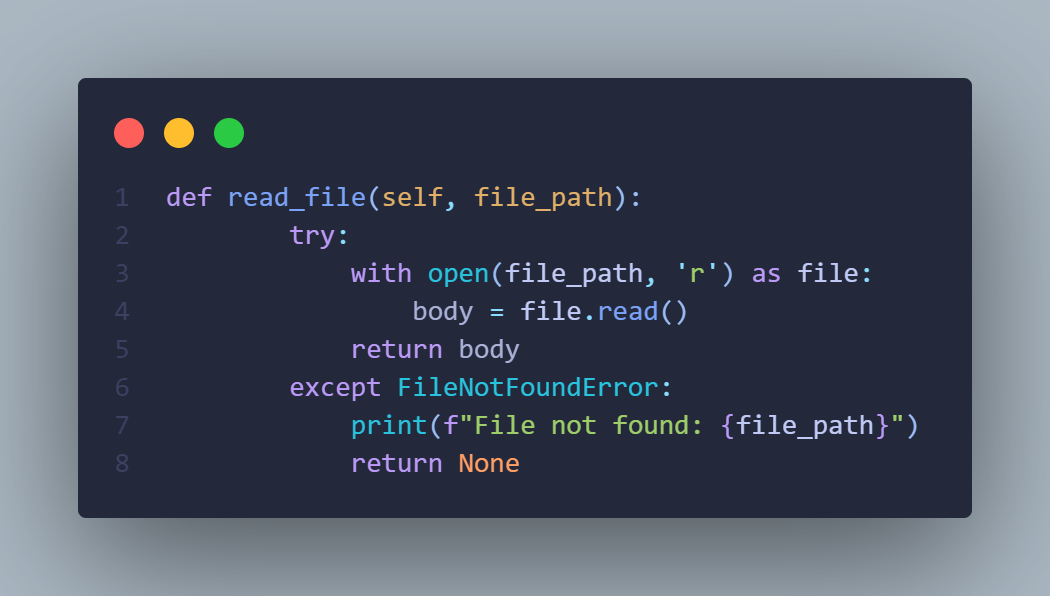
## keyLog.py

Class **keyLog()**: bao gồm các hàm liên quan đến ghi bàn phím:

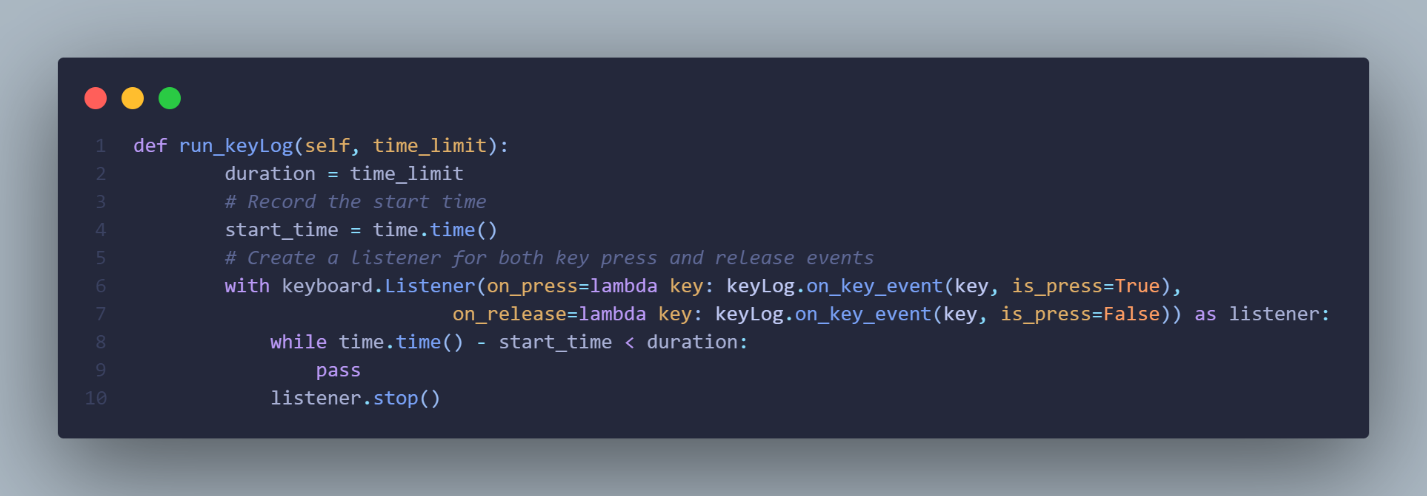
* ***clearFile***:



* **with open(file\_path, 'w') as file**: Mở tệp đường dẫn được chỉ định trong chế độ ghi ('w'). Sử dụng từ khóa with giúp đảm bảo tệp sẽ được đóng lại sau khi hoàn thành, ngay cả khi có lỗi xảy ra.
* **file.truncate(0):** Gọi phương thức truncate với tham số là 0. Phương thức này cắt độ dài của tệp xuống còn 0, tức là xóa tất cả nội dung của tệp.
* ***on\_key\_event***:
* **def on\_key\_event():** Định nghĩa một hàm tên là on\_key\_event nhận vào hai tham số: key và is\_press.
* **try:** Bắt đầu một khối try để bắt các ngoại lệ có thể xảy ra.
* **if is\_press:** Kiểm tra nếu sự kiện là nhấn phím.
* **with open("keylog.txt", "a", encoding='utf-8') as f:** Mở tệp "keylog.txt" trong chế độ ghi tiếp ('a') với mã hóa UTF-8. Tệp này sẽ được tạo nếu nó không tồn tại.
* **f.write('Key pressed: {0}\n'.format(key))**: Ghi chuỗi 'Key pressed: {0}\n' vào tệp, với {0} được thay thế bằng giá trị của key. Điều này ghi lại phím đã được nhấn.
* **except AttributeError:** Bắt ngoại lệ AttributeError, có thể xảy ra khi cố gắng truy cập một thuộc tính không tồn tại của đối tượng key.
* **pass:** Nếu có AttributeError, hàm sẽ không làm gì và tiếp tục thực thi. Điều này giúp xử lý các phím đặc biệt mà không gây ra lỗi.
* ***read\_file:***



* **def read\_file(self, file\_path):** Định nghĩa một hàm read\_file nhận vào một tham số file\_path.
* **try**: Bắt đầu một khối try để bắt các ngoại lệ có thể xảy ra.
* **with open(file\_path, 'r') as file:** Mở tệp đường dẫn được chỉ định trong chế độ đọc ('r'). Sử dụng từ khóa with giúp đảm bảo tệp sẽ được đóng lại sau khi hoàn thành, ngay cả khi có lỗi xảy ra.
* **body = file.read():** Đọc toàn bộ nội dung của tệp và gán nó vào biến body.
* **return body**: Trả về nội dung của tệp.
* **except FileNotFoundError**: Bắt ngoại lệ FileNotFoundError, có thể xảy ra khi tệp không tồn tại.
* **print(f"File not found: {file\_path}"):** In ra thông báo lỗi với đường dẫn tệp không tìm thấy.
* **return None:** Trả về None khi không tìm thấy tệp.
* ***run\_keyLog:***



* **def run\_keyLog(self, time\_limit):** Định nghĩa một hàm run\_keyLog nhận vào tham số time\_limit.
* **duration = time\_limit:** Đặt thời gian chạy cho keylogger.
* **start\_time = time.time():** Ghi lại thời gian bắt đầu.
* **with keyboard.Listener(...) as listener:** Tạo một listener để theo dõi cả sự kiện nhấn phím và thả phím.
* **on\_press=lambda key: keyLog.on\_key\_event(key, is\_press=True):** Khi một phím được nhấn, hàm on\_key\_event sẽ được gọi với tham số is\_press được đặt là True.
* **on\_release=lambda key: keyLog.on\_key\_event(key, is\_press=False):** Khi một phím được thả, hàm on\_key\_event sẽ được gọi với tham số is\_press được đặt là False.
* **while time.time() - start\_time < duration:** Vòng lặp sẽ tiếp tục cho đến khi thời gian hiện tại trừ đi thời gian bắt đầu vượt quá thời gian chạy đã định.
* **listener.stop():** Sau khi vòng lặp kết thúc, listener sẽ được dừng lại.

## appController.py

Class **AppController()**: bao gồm các hàm liên quan đến điều khiển ứng dụng:

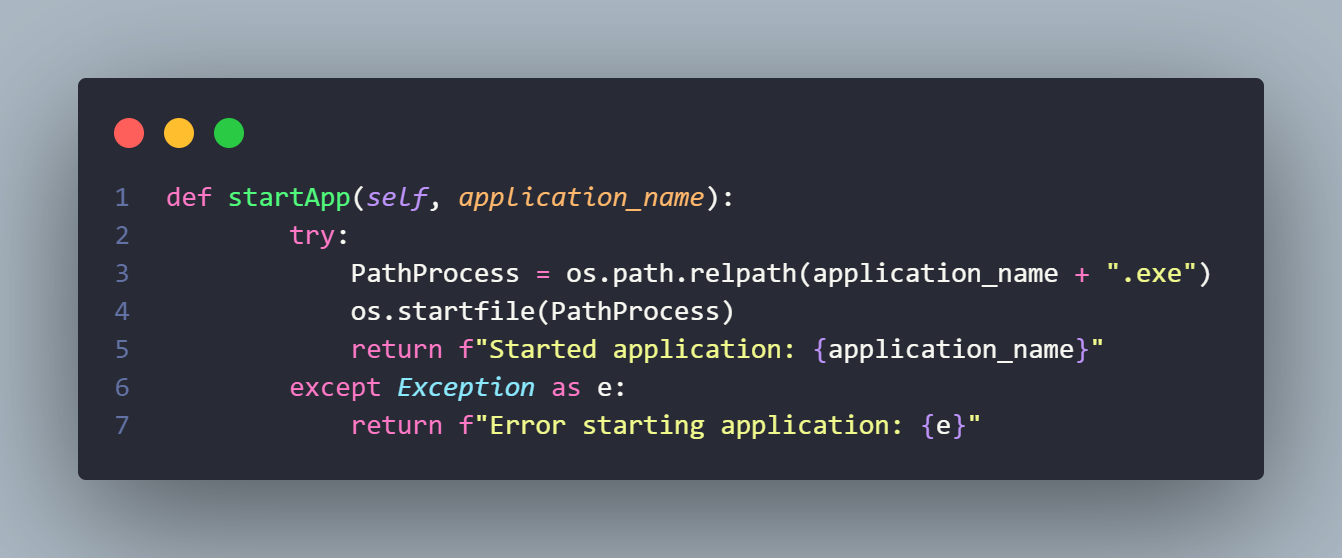
* ***process2List*:**



* **a = processes.decode().strip()**: Chuyển đổi chuỗi byte thành chuỗi văn bản thông qua phương thức **decode()**, sau đó loại bỏ các khoảng trắng ở đầu và cuối chuỗi bằng phương thức **strip()**. Kết quả là chuỗi văn bản đã được chuẩn hóa.Giải mã chuỗi byte thành chuỗi thông thường, loại bỏ khoảng trắng dẫn/dẫn cuối.
* **b = a.split("\r\n")**: Tách chuỗi thành một danh sách các dòng, sử dụng ký tự xuống dòng (**"\r\n"**) làm điểm tách.Tách mỗi dòng thành một danh sách các từ.
* **b = [" ".join(x.split()) for x in b]**: Cho mỗi dòng trong danh sách, loại bỏ các khoảng trắng thừa ở giữa bằng cách sử dụng **split()** để tách từng từ trong dòng, sau đó **join()** để kết hợp lại thành một chuỗi mới. Kết quả là danh sách các dòng đã được chuẩn hóa.
* **c = [x.split() for x in b][2:]**: Cho mỗi dòng trong danh sách sau bước trước, tách chuỗi thành các phần tử và lưu thành danh sách con. Bỏ qua hai phần tử đầu tiên của danh sách (hàng tiêu đề không cần thiết).
* **return c**: Trả về danh sách cuối cùng, đại diện cho thông tin chi tiết về các tiến trình, được biểu diễn dưới dạng danh sách các danh sách.
* ***viewList*:**



* **subprocess.check\_output**: Là một hàm trong thư viện **subprocess** của Python, được sử dụng để thực thi một lệnh hệ thống và trả về kết quả của lệnh đó dưới dạng chuỗi byte.
* "**powershell gps | where {$\_.MainWindowTitle} | select Name,Id,@{Name='ThreadCount';Expression={$\_.Threads.Count}}**"
  + **powershell gps**: Chạy cmdlet gps trong PowerShell để lấy thông tin về các tiến trình (Get-Process).
  + **where {$\_.MainWindowTitle}**: Lọc các tiến trình dựa trên có cửa sổ chính hay không.
  + **select Name,Id,@{Name='ThreadCount';Expression={$\_.Threads.Count}}**: Chọn các thuộc tính Name, Id và tạo một thuộc tính mới là ThreadCount, thể hiện số lượng luồng cho mỗi tiến trình.
* Kết quả của lệnh PowerShell được trả về dưới dạng chuỗi byte và được lưu vào biến **app**
* **self.appList = self.process2List(app)**: Kết quả được truyền vào hàm process2List để chuyển đổi từ chuỗi byte sang danh sách các tiến trình.
* **table\_headers = ["Process", "PID", "Thread Count"]**: Định nghĩa tiêu đề cho bảng dữ liệu.
* **formatted\_table = tabulate(self.appList, headers=table\_headers, tablefmt="pretty")**: Sử dụng thư viện **tabulate** để định dạng danh sách các tiến trình thành một bảng dễ đọc. Kết quả được trả về dưới dạng một chuỗi chứa bảng được định dạng.
* **return formatted\_table**: Trả về bảng dữ liệu đã được định dạng.
* ***startApp*:**



* **PathProcess = os.path.relpath(application\_name + ".exe")**: Xây dựng đường dẫn tương đối đến tập tin thực thi của ứng dụng. Chuỗi **".exe"** được thêm vào tên ứng dụng để đảm bảo rằng extension của tập tin thực thi đã được xác định.
* **os.startfile(PathProcess)**: Sử dụng hàm **startfile** từ module **os** để mở tập tin hoặc ứng dụng với chương trình mặc định được liên kết với loại tập tin đó trên hệ điều hành. Trong trường hợp này, nó sẽ mở ứng dụng có đường dẫn là **PathProcess**.
* **return f"Started application: {application\_name}"**: Trả về một chuỗi thông báo thành công khi ứng dụng được khởi động. Chuỗi này chứa tên ứng dụng để người dùng biết ứng dụng nào đã được bắt đầu.
* **except Exception as e: return f"Error starting application: {e}"**: Nếu có bất kỳ lỗi nào xảy ra trong quá trình khởi động ứng dụng, chương trình sẽ rơi vào khối except. Nó sẽ trả về một chuỗi thông báo lỗi chứa thông tin chi tiết về lỗi đó, giúp người dùng xác định vấn đề
* ***endApp*:**

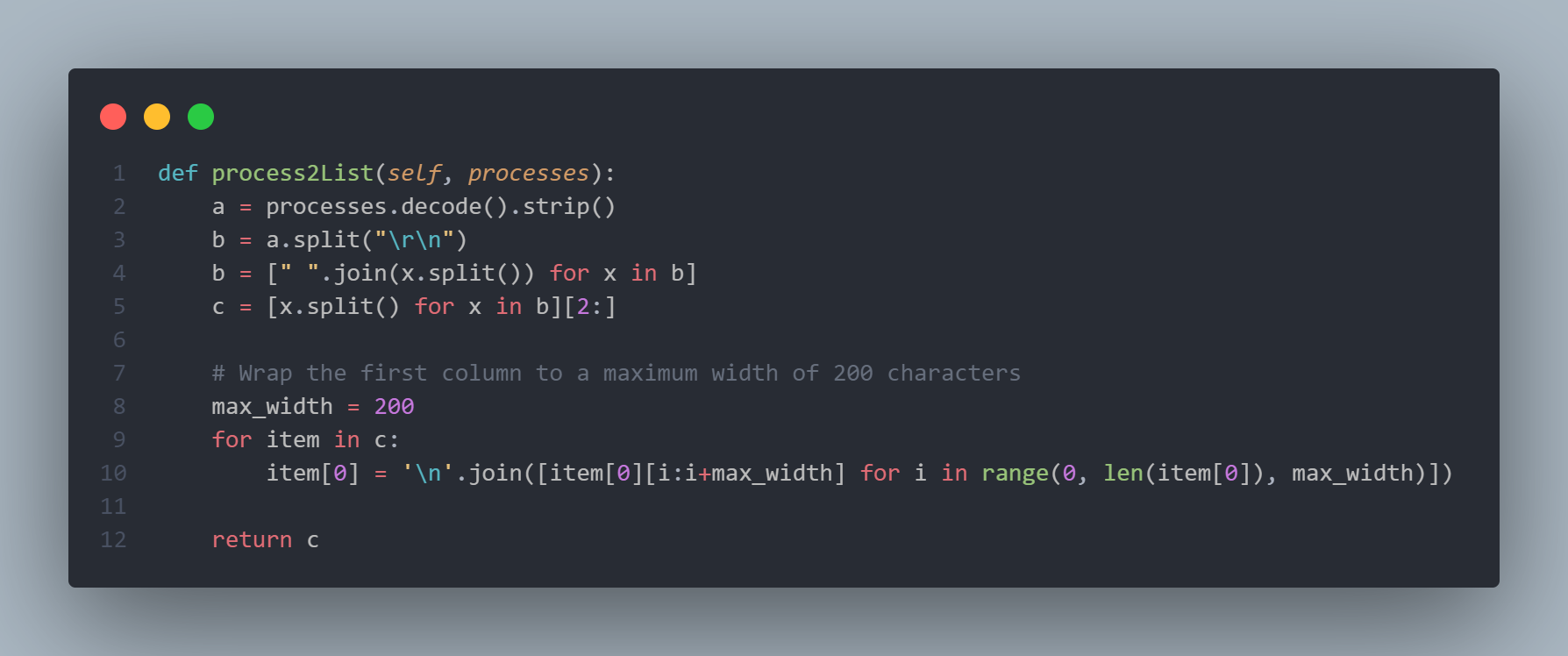


* **check = False**: Tạo một biến **check** và khởi tạo giá trị là **False**. Biến này được sử dụng để kiểm tra xem có tìm thấy và kết thúc ứng dụng hay không.
* **for process in psutil.process\_iter(attrs=['pid', 'name']):**: Sử dụng vòng lặp để duyệt qua tất cả các tiến trình đang chạy trên hệ thống sử dụng **psutil.process\_iter**. Thông tin về tiến trình được thu thập bao gồm **pid** (Process ID) và **name** (tên tiến trình).
* Trong **try**, **process\_info = process.info** lấy thông tin về tiến trình.
* **if process\_info['name'] == application\_name + ".exe":**: Kiểm tra xem tên của tiến trình có khớp với tên ứng dụng được đưa vào hay không.
* **pid = process\_info['pid']**: Lấy Process ID (PID) của tiến trình.
* **process = psutil.Process(pid)**: Tạo một đối tượng **Process** từ PID để thao tác với tiến trình.
* **process.terminate()**: Gọi phương thức **terminate()** để kết thúc tiến trình.
* **check = True**: Đặt giá trị của biến **check** thành **True** để cho biết tiến trình đã được kết thúc thành công.
* **except (psutil.NoSuchProcess, psutil.AccessDenied, psutil.ZombieProcess):** Bắt các ngoại lệ có thể xảy ra trong quá trình lấy thông tin về tiến trình, như khi không tìm thấy tiến trình (**NoSuchProcess**), truy cập bị từ chối (**AccessDenied**), hoặc tiến trình là Zombie (**ZombieProcess**).
* **if(check == True): return f"End Application {application\_name} Successfully!"**: Kiểm tra giá trị của **check**. Nếu **check** là **True**, thì trả về một chuỗi thông báo cho biết ứng dụng đã được kết thúc thành công.
* **else: return f"No App {application\_name} Found."**: Nếu **check** là **False**, thì trả về một chuỗi thông báo cho biết không tìm thấy ứng dụng với tên đã đưa vào.

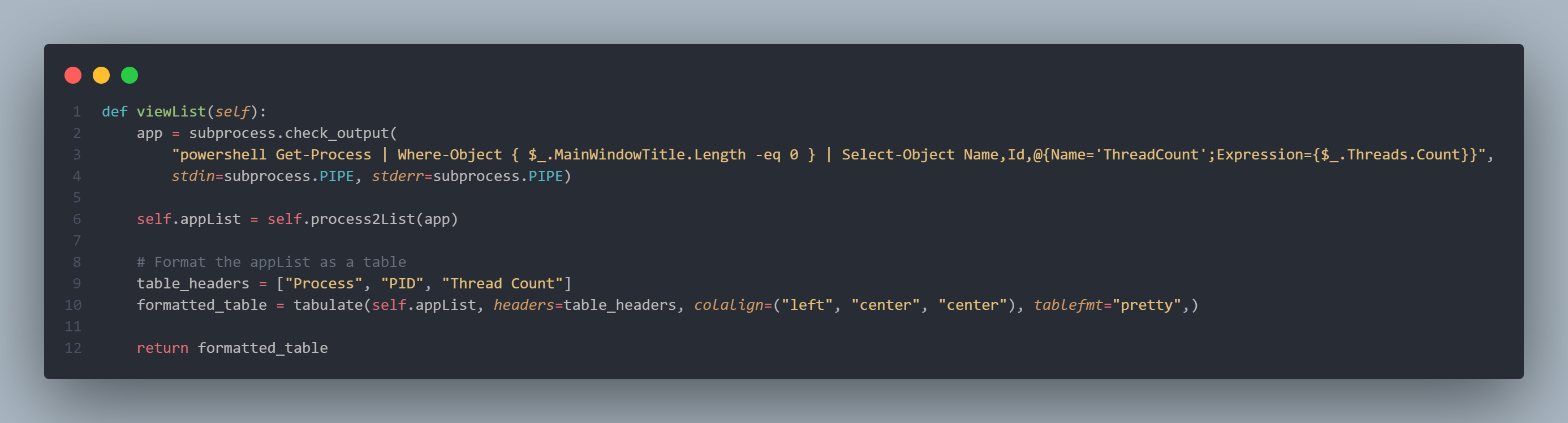
## processController.py

Class **ProcessController()**: chứa các hàm dùng để tương tác với process, bao gồm:

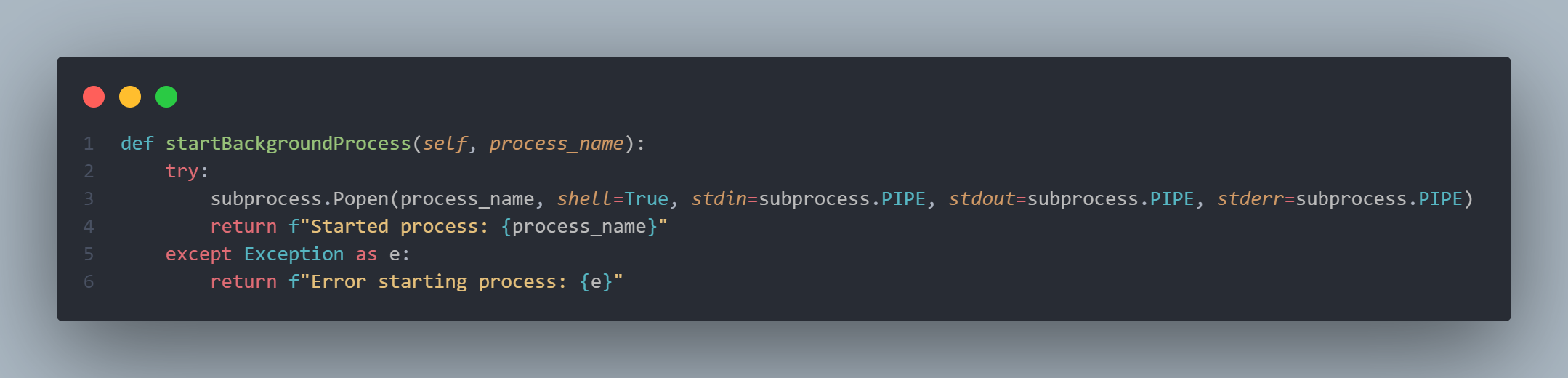
* ***process2List*:**



* Giống như trong **AppController**, hàm **process2List** của **ProcessController** nhận đầu vào là kết quả của lệnh PowerShell được gọi bằng **subprocess.check\_output**.
* Kết quả này được chuyển thành một danh sách các process với thông tin như tên process, ID và số lượng luồng.
* Do tên của process có thể rất dài, đối với mỗi tiến trình, cột đầu tiên (tên process) được chia thành các dòng có chiều dài tối đa là 200 ký tự.
* ***viewList*:**



* **viewList** sử dụng **subprocess.check\_output** để chạy lệnh PowerShell, lấy danh sách các tiến trình không có cửa sổ chính (**Where-Object { $\_.MainWindowTitle.Length -eq 0 }**).
* Sau đó, gọi **process2List** để chuyển đổi kết quả thành danh sách và sử dụng **tabulate** để định dạng danh sách này thành một bảng.
* ***startBackgroundProcess*:**



* Sử dụng **subprocess.Popen** để bắt đầu một process mới dựa trên tên process được truyền vào:
  + **shell=True**: Đặt giá trị này thành True cho phép sử dụng PowerShell để thực hiện câu lệnh.
  + **stdin=subprocess.PIPE**: Thiết lập luồng đầu vào (stdin) của tiến trình. Trong trường hợp này, **subprocess.PIPE** được sử dụng để tạo một ống dẫn đầu vào từ quy trình Python chính đến tiến trình con.
  + **stdout=subprocess.PIPE**: Thiết lập luồng đầu ra (stdout) của tiến trình. Tương tự như stdin, **subprocess.PIPE** tạo một ống dẫn đầu ra từ tiến trình con đến quy trình Python chính.
  + **stderr=subprocess.PIPE**: Thiết lập luồng lỗi (stderr) của tiến trình. Cũng giống như stdin và stdout, subprocess.PIPE tạo một ống dẫn đến quy trình Python chính.
* Nếu có lỗi, hàm sẽ trả về một thông báo lỗi.
* ***endProcess*:**



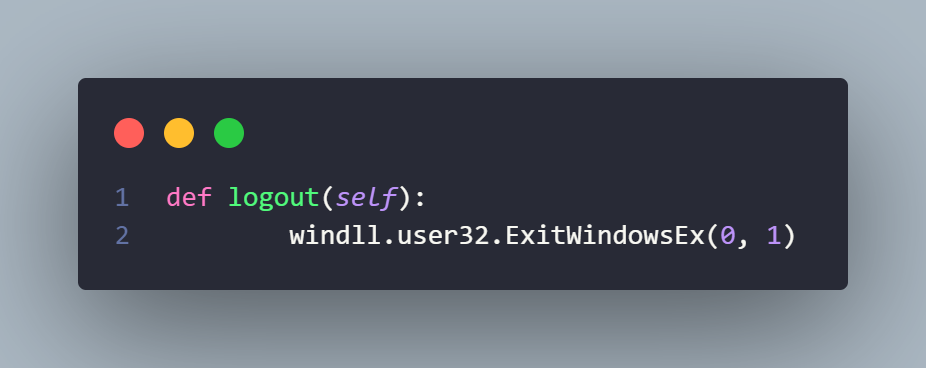
* Sử dụng **subprocess.run** để chạy lệnh **taskkill** để kết thúc một process dựa trên tên được truyền vào
  + **f"taskkill /f /im {process\_name}"**: Chuỗi lệnh được tạo ra bằng cách sử dụng f-string để chèn tên tiến trình vào lệnh **taskkill**. Đối số **/f** yêu cầu kết thúc tiến trình mạnh mẽ (force), và **/im** chỉ định tiến trình bằng tên của nó.
  + **shell=True**: Cho phép sử dụng PowerShell để thực hiện câu lệnh.
  + **check=True**: Nếu đặt thành True, nếu lệnh thực thi trả về mã lỗi khác 0, một **subprocess.CalledProcessError** sẽ được ném. Điều này có nghĩa là nếu lệnh **taskkill** thất bại, nó sẽ ném một ngoại lệ.
  + **text=True**: Đảm bảo rằng đầu ra và lỗi của quá trình con được trả về dưới dạng chuỗi thay vì byte.
  + **capture\_output=True**: Bật chế độ bắt đầu đầu ra từ quá trình con. Điều này làm cho đầu ra của lệnh **taskkill** có sẵn trong thuộc tính stdout của đối tượng **CompletedProcess**.
  + **return result.stdout.strip()**: Trả về đầu ra của lệnh taskkill dưới dạng chuỗi đã được loại bỏ khoảng trắng ở đầu và cuối chuỗi. Điều này có thể là một chuỗi thông báo xác nhận từ **taskkill** hoặc thông báo lỗi nếu có.
* Nếu có lỗi, hàm sẽ trả về một thông báo lỗi.

## powerController.py

* ***shutdown:***



* Sử dụng hàm **os.system** để thực thi lệnh **shutdown /s /t 60**, tắt máy tính sau 60 giây.
* ***logout:***



* Sử dụng hàm **ExitWindowsEx** từ **windll.user32** để đăng xuất người dùng khỏi hệ thống. Tham số **0** đại diện cho đăng xuất, và **1** đại diện cho lựa chọn đóng tất cả các ứng dụng trước khi đăng xuất.

# Hướng dẫn sử dụng

## Thiết lập

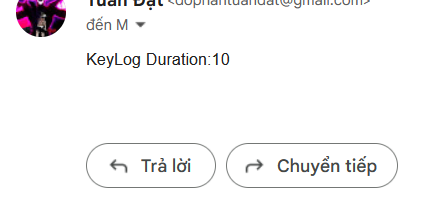
### Thiết bị được điều khiển

### Thiết bị điều khiển

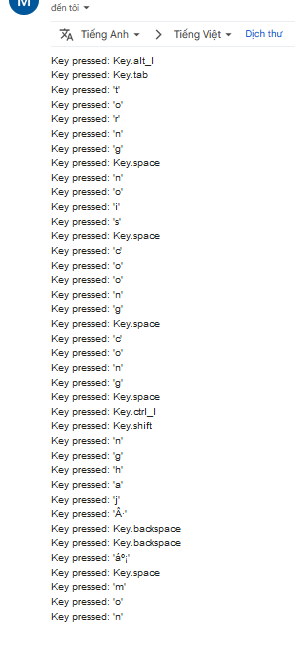
* Không cần thiết lập. Chỉ cần gửi email cho [2d2h.computernetwork.clc.fitus@gmail.com](mailto:2d2h.computernetwork.clc.fitus@gmail.com)
* Có thể gửi từ bất kỳ thiết bị nào
* Mail không nhất thiết phải có tiêu đề. Trong nội dung mail sẽ có những từ khóa để kích hoạt tính năng nhất định
* Từ khóa và các yêu cầu phải nằm trong một dòng riêng biệt. Mọi nội dung khác của email sẽ được bỏ qua

## Ghi bàn phím

* Gửi mail có chứa dòng lênh: **KeyLog Duration:<thời gian (giây)>**
* VD: **Keylog Duration:10** để ghi bàn phím trong 10 giây kể từ khi chương trình đọc được mail

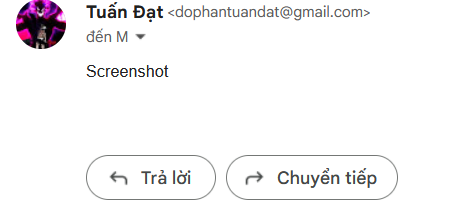
****

* Sau khi hết thời hạn ghi bàn phím, người gửi sẽ nhận được reply là nội dung đã ghi được

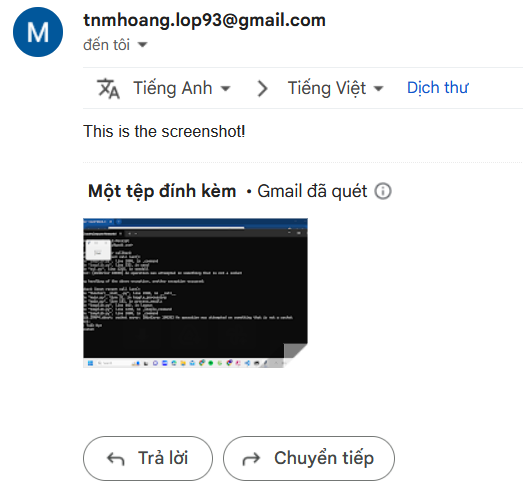
****

## Chụp màn hình

* Gửi emal có chứa dòng lệnh: **Screenshot**



* Sau một lúc, người gửi sẽ nhận được một thư chứa ảnh chụp màn hình của thiết bị được điều khiển



# Đóng góp

# Tài liệu tham khảo

* Đọc email bằng python: <https://thepythoncode.com/article/reading-emails-in-python>
* Sử dụng module subprocess: <https://www.dataquest.io/blog/python-subprocess/>
* Gửi emal bằng python: <https://www.tutorialspoint.com/python/python_sending_email.htm>
* Cách ghi lại bàn phím bằng python: <https://www.youtube.com/watch?v=mDY3v2Xx-Q4&t=163s>
* Chụp ảnh màn hình bằng python: <https://nitratine.net/blog/post/how-to-take-a-screenshot-in-python-using-pil/>
* Tắt máy tính bằng python: <https://www.geeksforgeeks.org/python-script-to-shutdown-computer/>