BÀI 4. CẤU TRÚC LẶP & GIT/GITHUB CƠ BẢN

4.1. Vòng lặp for

4.1.1. Khái niệm về vòng lặp và tầm quan trọng

Trong lập trình, **vòng lặp (Loop)** là một cấu trúc điều khiển cho phép một khối lệnh được thực thi lặp đi lặp lại nhiều lần. Thay vì viết cùng một đoạn code nhiều lần, bạn chỉ cần viết nó một lần và đặt nó vào một vòng lặp.

Tầm quan trọng của vòng lặp:

- **Tự động hóa:** Thực hiện các tác vụ lặp lại một cách tự động, tiết kiệm thời gian và công sức.
- **Xử lý dữ liệu lớn:** Duyệt qua các danh sách, chuỗi, hoặc tập hợp dữ liệu để thực hiện thao tác trên từng phần tử.
- **Hiệu quả code:** Làm cho code ngắn gọn, dễ đọc, và dễ bảo trì hơn.

4.1.2. Cú pháp và luồng hoạt động của vòng lặp for

Vòng lặp for trong Python được sử dụng để lặp qua các phần tử của một chuỗi (ví dụ: chuỗi ký tự, list, tuple, dictionary, set) hoặc bất kỳ đối tượng nào có thể lặp được (iterable).

Cú pháp:

```
for phan_tu in tap_hop_co_the_lap:
    # Khối lệnh này sẽ được thực thi cho mỗi phan_tu trong
tap_hop_co_the_lap
    # (Lưu ý: Phải thụt lề 4 dấu cách hoặc 1 tab)
```

Luồng hoạt động:

1. Python lấy phần tử đầu tiên từ tap_hop_co_the_lap và gán nó cho biến phan_tu.
2. Khối lệnh bên trong vòng lặp được thực thi.
3. Python lấy phần tử tiếp theo từ tap_hop_co_the_lap và lặp lại bước 2.
4. Quá trình này tiếp tục cho đến khi không còn phần tử nào trong tap_hop_co_the_lap.

Ví dụ:

```
# Lặp qua một danh sách các số
cac_so = [1, 2, 3, 4, 5]
print("Các số trong danh sách:")
for so in cac_so:
```

```
    print(so)
# Lặp qua các ký tự trong một chuỗi
ten = "Python"
print("\nCác ký tự trong tên:")
for ky_tu in ten:
    print(ky_tu)
```

4.1.3. Hàm range(): tạo dãy số, các dạng sử dụng

Hàm range() rất hữu ích khi bạn muốn lặp một số lần cụ thể hoặc tạo ra một dãy số để lặp qua. Nó trả về một đối tượng range (không phải một list), là một loại iterable.

Các dạng sử dụng:

1./ **range(stop):** Tạo dãy số từ 0 đến stop-1.

- range(5) tạo ra dãy 0, 1, 2, 3, 4.

Ví dụ:

```
print("Dãy số từ 0 đến 4:")
for i in range(5):
    print(i)
```

2./ **range(start, stop):** Tạo dãy số từ start đến stop-1.

- range(2, 7) tạo ra dãy 2, 3, 4, 5, 6.

Ví dụ:

```
print("\nDãy số từ 2 đến 6:")
for i in range(2, 7):
    print(i)
```

3./ **range(start, stop, step):** Tạo dãy số từ start đến stop-1, với bước nhảy là step.

- range(1, 10, 2) tạo ra dãy 1, 3, 5, 7, 9.
- range(10, 0, -1) tạo ra dãy 10, 9, 8, ..., 1 (đếm ngược).

Ví dụ:

```
print("\nDãy số lẻ từ 1 đến 9:")
for i in range(1, 10, 2):
    print(i)
print("\nĐếm ngược từ 5 về 1:")
for i in range(5, 0, -1):
    print(i)
```

4./ Lặp qua các phần tử trong chuỗi

Bạn có thể dễ dàng lặp qua từng ký tự trong một chuỗi bằng vòng lặp for.

Ví dụ:

```
thong_diep = "Hello Python!"
print("Các ký tự trong thông điệp:")
for char in thong_diep:
    print(char)
# Đếm số lần xuất hiện của một ký tự
count_o = 0
for char in thong_diep:
    if char == 'o' or char == 'O':
        count_o += 1
print(f"Số lần ký tự 'o' hoặc 'O' xuất hiện: {count_o}")
```

4.2. Vòng lặp while & Kiểm soát vòng lặp

4.2.1. Cú pháp và luồng hoạt động của vòng lặp while

Vòng lặp while được sử dụng để thực thi một khối lệnh **miễn là một điều kiện nào đó còn đúng (True)**. Nó tiếp tục lặp cho đến khi điều kiện trở thành False.

Cú pháp:

```
while dieu_kien:
    # Khối lệnh này sẽ được thực thi MIỄN LÀ dieu_kien là True
    # (Lưu ý: Phải thụt lề 4 dấu cách hoặc 1 tab)
```

Luồng hoạt động:

1. Python kiểm tra dieu_kien.
2. Nếu dieu_kien là True: Thực thi khối lệnh bên dưới while. Sau đó quay lại bước 1.
3. Nếu dieu_kien là False: Thoát khỏi vòng lặp và tiếp tục thực thi các lệnh sau vòng lặp.

Ví dụ:

```
dem = 1
print("Đếm từ 1 đến 5:")
while dem <= 5:
    print(dem)
    dem += 1 # Phải có lệnh làm thay đổi điều kiện để vòng lặp dừng
print("Vòng lặp đã kết thúc.")
```

4.2.2. Lưu ý về vòng lặp vô hạn và cách tránh

Vòng lặp vô hạn (Infinite Loop) là vòng lặp mà điều kiện của nó không bao giờ trở thành False, dẫn đến việc chương trình lặp đi lặp lại mãi mãi và không bao giờ dừng. Điều này thường là một lỗi.

Cách tránh vòng lặp vô hạn:

- **Đảm bảo điều kiện thay đổi:** Luôn có một hoặc nhiều lệnh bên trong vòng lặp làm thay đổi các biến được sử dụng trong điều kiện while sao cho cuối cùng điều kiện đó sẽ trở thành False.
- **Sử dụng break:** Trong một số trường hợp, bạn có thể chủ động thoát khỏi vòng lặp bằng lệnh break (sẽ học bên dưới).
- **Kiểm tra cẩn thận:** Luôn suy nghĩ về tất cả các trường hợp có thể xảy ra và đảm bảo vòng lặp sẽ dừng trong mọi tình huống.

Ví dụ lỗi vòng lặp vô hạn:

```
# dem = 1
# while dem <= 5:
#     print(dem)
#     # Quên tăng dem lên, nên dem sẽ mãi mãi là 1 và vòng lặp không bao
giờ dừng!
```

4.2.3. Các lệnh điều khiển vòng lặp: break, continue, pass

Các lệnh này cung cấp khả năng kiểm soát linh hoạt hơn đối với luồng thực thi bên trong vòng lặp.

1./ **break** (Thoát vòng lặp):

- Khi Python gặp lệnh break, sẽ ngay lập tức thoát ra khỏi vòng lặp gần nhất chứa nó.
- Các lệnh còn lại trong khối vòng lặp sau break sẽ không được thực thi cho lần lặp hiện tại, và vòng lặp cũng sẽ không tiếp tục sang các lần lặp tiếp theo.

Ví dụ:

```
print("Sử dụng break:")
for i in range(1, 10):
    if i == 5:
        print("Đã tìm thấy 5, thoát vòng lặp.")
        break # Thoát khỏi vòng lặp khi i bằng 5
    print(i)
```

2./ **continue** (Bỏ qua lần lặp hiện tại):

- Khi Python gặp lệnh continue, nó sẽ bỏ qua phần còn lại của khối lệnh trong lần lặp hiện tại.
- Chương trình sẽ chuyển ngay sang lần lặp tiếp theo của vòng lặp. **Ví dụ:**

Ví dụ:

```
print("\nSử dụng continue:")

for i in range(1, 6):
    if i == 3:
        print(f"Bỏ qua số {i}.")
        continue # Bỏ qua print(i) nếu i bằng 3

    print(i)
```

3./ **pass** (Làm rỗng):

- pass là một câu lệnh rỗng. Nó không làm gì cả.
- Đôi khi, bạn cần có một câu lệnh về mặt cú pháp nhưng không muốn thực hiện bất kỳ hành động nào. pass được sử dụng như một chỗ giữ chỗ (placeholder).
- Thường dùng trong các hàm, lớp, hoặc vòng lặp khi bạn chưa biết code sẽ viết gì nhưng muốn chương trình chạy mà không báo lỗi cú pháp.

Ví dụ:

```
print("\nSử dụng pass:")

for i in range(1, 4):
    if i == 2:
        pass # Không làm gì cả khi i bằng 2
        print("Chỉ là một bước 'pass' khi i là 2.") # Dòng này vẫn chạy
vì pass không thoát lặp
    else:
        print(f"Giá trị hiện tại: {i}")
```

Ví dụ khác: một hàm chưa triển khai:

```
def ham_chua_hoan_thien():
    pass # Tránh lỗi cú pháp khi hàm rỗng
```

4.3. Giới thiệu Git & GitHub

4.3.1. Khái niệm Hệ thống kiểm soát phiên bản (Version Control System) và Git

Khi bạn làm việc với code, bạn thường xuyên thay đổi, thêm bớt, sửa lỗi. Việc này có thể gây ra rủi ro: mất code, khó quay lại phiên bản cũ, khó làm việc nhóm.

Hệ thống kiểm soát phiên bản (Version Control System - VCS) là công cụ giúp quản lý các thay đổi đối với code hoặc bất kỳ loại file nào khác theo thời gian. Nó cho phép bạn:

- Theo dõi lịch sử thay đổi.
- Quay lại bất kỳ phiên bản nào trước đó.
- Làm việc cộng tác với nhiều người trên cùng một dự án mà không ghi đè lên nhau.

Git là một hệ thống kiểm soát phiên bản phân tán (Distributed Version Control System - DVCS) phổ biến nhất hiện nay.

- "Phân tán" có nghĩa là mỗi lập trình viên đều có một bản sao hoàn chỉnh của lịch sử dự án trên máy tính cục bộ của họ. Điều này giúp làm việc offline và tăng tốc độ.
- Git do Linus Torvalds (người tạo ra Linux) phát triển.

4.3.2. Giới thiệu GitHub: nền tảng lưu trữ mã nguồn và cộng tác

GitHub là một nền tảng dựa trên web sử dụng Git để lưu trữ các kho lưu trữ mã nguồn (repositories) và tạo điều kiện cho việc cộng tác giữa các lập trình viên.

- Nó không phải là Git, mà là một dịch vụ cung cấp giao diện và các tính năng bổ sung để quản lý các dự án Git từ xa.
- Là nơi phổ biến nhất để các lập trình viên chia sẻ code, đóng góp vào các dự án mã nguồn mở và làm việc nhóm.

4.3.3. Các khái niệm cơ bản: Repository, Commit, Branch

1./ **Repository** (Kho lưu trữ / Repo):

- Là một thư mục chứa tất cả các file của dự án của bạn, cùng với toàn bộ lịch sử thay đổi (tất cả các phiên bản của các file đó) được Git quản lý.
- Có hai loại:
  - **Local Repository:** Kho lưu trữ trên máy tính của bạn.
  - **Remote Repository:** Kho lưu trữ trên một máy chủ từ xa (như GitHub).

2./ Commit:

- Là một "ảnh chụp nhanh" (snapshot) của dự án của bạn tại một thời điểm cụ thể.
- Mỗi commit đại diện cho một bộ thay đổi (thêm tính năng, sửa lỗi, cập nhật tài liệu).
- Mỗi commit có một thông báo (message) mô tả những thay đổi đã được thực hiện và một mã định danh duy nhất (hash).
- Commit là hành động lưu lại các thay đổi vào lịch sử Git cục bộ.

3./ **Branch** (Nhánh):

- Là một dòng phát triển độc lập trong dự án của bạn.
- Khi bạn tạo một branch mới, bạn đang tạo một bản sao của dự án tại thời điểm đó để làm việc mà không ảnh hưởng đến code chính (thường là main hoặc master branch).
- Điều này giúp nhiều người làm việc trên các tính năng khác nhau cùng lúc mà không xung đột, sau đó hợp nhất (merge) các thay đổi lại khi hoàn thành.

4.3.4. Quy trình làm việc cơ bản: clone, add, commit, push

Đây là quy trình làm việc phổ biến khi bạn bắt đầu một dự án từ GitHub hoặc muốn đóng góp vào một dự án hiện có.

1./ **git clone** [URL_repository]:

- Tải về một bản sao của kho lưu trữ từ xa (trên GitHub) về máy tính cục bộ của bạn.
- Bạn chỉ cần làm điều này một lần cho mỗi dự án.

```
git clone https://github.com/your_username/your_repo.git
```

2./ **git add** [tên_file] hoặc git add .:

- **Stage** các thay đổi của bạn. Điều này có nghĩa là bạn chọn những file hoặc những thay đổi cụ thể mà bạn muốn đưa vào commit tiếp theo.
- git add . sẽ thêm tất cả các thay đổi trong thư mục hiện tại vào khu vực staging.

```
git add ten_file_moi.py
git add . # Thêm tất cả các thay đổi
```

3./ **git commit -m** "Thông báo commit của bạn":

- Tạo một **commit** mới với các thay đổi đã được add vào khu vực staging.
- Thông báo commit (-m "...") rất quan trọng, nó phải mô tả rõ ràng những gì bạn đã thay đổi.

```
git commit -m "Thêm tính năng đăng nhập người dùng"
```

4./ git push origin [tên_branch]:

- Tải các commit từ kho lưu trữ cục bộ của bạn lên kho lưu trữ từ xa (trên GitHub).
- origin là tên mặc định cho kho lưu trữ từ xa mà bạn đã clone từ đó.
- [tên_branch] thường là main hoặc master.

```
git push origin main
```

4.3.5. Quy trình lặp lại:

1. Sửa đổi code.
2. git add . (để chuẩn bị các thay đổi).
3. git commit -m "Thông báo commit" (để lưu thay đổi cục bộ).
4. git push origin main (để đẩy thay đổi lên GitHub).

BÀI TẬP:

1. Sử dụng **for loop** để tính tổng 100 số nguyên đầu tiên.
2. Viết chương trình sử dụng **while loop** để yêu cầu người dùng nhập mật khẩu, lặp lại cho đến khi mật khẩu đúng ("admin123").
3. Sử dụng for loop và range() để in ra bảng cửu chương (ví dụ: bảng nhân 7).
4. Viết chương trình tìm **số nguyên tố** trong khoảng từ 1 đến 50. Sử dụng **nested loop** (vòng lặp lồng nhau).
5. Xây dựng game đoán số. Sử dụng while loop, input() và break. Nếu người dùng đoán sai 5 lần thì dùng break để kết thúc vòng lặp.

THỰC HÀNH 04: CẤU TRÚC LẶP & GIT/GITHUB

4.1. Ứng dụng vòng lặp for

4.1.1. Viết chương trình in bảng cửu chương

Mục tiêu: Sử dụng vòng lặp for và hàm range() để tạo dãy số.

Bài tập: Tạo file multiplication_table.py. Yêu cầu người dùng nhập một số nguyên N (từ 2 đến 9). Sử dụng vòng lặp for để in ra bảng cửu chương của số đó từ 1 đến 10.

Ví dụ: Nếu người dùng nhập 7, kết quả sẽ như sau:

```
Bảng cửu chương của 7:
7 x 1 = 7
7 x 2 = 14
...
7 x 10 = 70
```

Gợi ý:

```
# multiplication_table.py

print("--- BẢNG CỬU CHƯƠNG ---")
try:
    number_str = input("Nhập một số nguyên (từ 2 đến 9) để in bảng cửu
chương: ")
    number = int(number_str)

    if 2 <= number <= 9:
        print(f"\nBảng cửu chương của {number}:")
        for i in range(1, 11): # Lặp từ 1 đến 10
            result = number * i
            print(f"{number} x {i} = {result}")
    else:
        print("Lỗi: Vui lòng nhập số trong khoảng từ 2 đến 9.")

except ValueError:
    print("Lỗi: Vui lòng nhập một số nguyên hợp lệ.")
```

4.1.2. Tính tổng các số từ 1 đến N

Mục tiêu: Sử dụng vòng lặp for để thực hiện phép tích lũy (tổng).

Bài tập: Tạo file sum_to_n.py. Yêu cầu người dùng nhập một số nguyên dương N. Sử dụng vòng lặp for để tính tổng tất cả các số nguyên từ 1 đến N. In kết quả tổng ra màn hình.

Ví dụ: Nếu N = 5, tổng là 1 + 2 + 3 + 4 + 5 = 15.

Gợi ý:

```
# sum_to_n.py

print("--- TÍNH TỔNG TỪ 1 ĐẾN N ---")
try:
    n_str = input("Nhập một số nguyên dương N: ")
    n = int(n_str)

    if n > 0:
        total_sum = 0 # Biến để lưu trữ tổng
        for i in range(1, n + 1): # Lặp từ 1 đến N
            total_sum += i # Tích lũy tổng
        print(f"Tổng các số từ 1 đến {n} là: {total_sum}")
    else:
        print("Lỗi: Vui lòng nhập một số nguyên dương.")

except ValueError:
    print("Lỗi: Vui lòng nhập một số nguyên hợp lệ.")
```

4.1.3. Đếm số ký tự nguyên âm/phụ âm trong một chuỗi

Mục tiêu: Lặp qua chuỗi, sử dụng điều kiện và toán tử logic.

Bài tập: Tạo file vowel_consonant_counter.py. Yêu cầu người dùng nhập một chuỗi văn bản. Sử dụng vòng lặp for để đếm số lượng ký tự nguyên âm (a, e, i, o, u, A, E, I, O, U) và số lượng ký tự phụ âm trong chuỗi đó. Bỏ qua các ký tự không phải chữ cái (số, dấu cách, ký hiệu). In ra tổng số nguyên âm và tổng số phụ âm.

Gợi ý:

```
# vowel_consonant_counter.py

print("--- ĐẾM NGUYÊN ÂM VÀ PHỤ ÂM ---")
text = input("Nhập một chuỗi văn bản: ")

vowels = "aeiouAEIOU"
vowel_count = 0
consonant_count = 0
```

```
for char in text:
    if char.isalpha(): # Kiểm tra xem ký tự có phải là chữ cái không
        if char in vowels: # Kiểm tra xem ký tự có trong chuỗi nguyên âm
không
            vowel_count += 1
        else:
            consonant_count += 1
    # else: # Có thể thêm phần này để đếm các ký tự khác nếu muốn
    #     print(f"Bỏ qua ký tự không phải chữ cái: {char}")

print(f"Số nguyên âm: {vowel_count}")
print(f"Số phụ âm: {consonant_count}")
```

4.2. Ứng dụng vòng lặp while & break/continue

4.2.1. Xây dựng chương trình đoán số (cho đến khi đoán đúng)

Mục tiêu: Sử dụng vòng lặp while với điều kiện dừng linh hoạt.

Bài tập: Tạo file number_guessing_game.py.

1. Chương trình sẽ "nghĩ" một số bí mật (ví dụ: secret_number = 7).
2. Sử dụng vòng lặp while để liên tục yêu cầu người dùng đoán số.
3. Nếu số đoán nhỏ hơn số bí mật, in ra "Quá nhỏ! Thử lại.".
4. Nếu số đoán lớn hơn số bí mật, in ra "Quá lớn! Thử lại.".
5. Nếu đoán đúng, in ra "Chúc mừng! Bạn đã đoán đúng số bí mật!" và kết thúc trò chơi (sử dụng break).
6. Đếm số lần đoán của người chơi và in ra khi họ thắng.

Gợi ý:

```
# number_guessing_game.py

import random # Module để tạo số ngẫu nhiên

print("--- TRÒ CHƠI ĐOÁN SỐ ---")
secret_number = random.randint(1, 20) # Tạo số bí mật từ 1 đến 20
guess_count = 0

print("Tôi đang nghĩ một số từ 1 đến 20. Hãy thử đoán xem!")
```

```
while True: # Vòng lặp vô hạn, sẽ thoát bằng break
    try:
        guess_str = input("Nhập số bạn đoán: ")
        guess = int(guess_str)
        guess_count += 1

        if guess < secret_number:
            print("Quá nhỏ! Thử lại.")
        elif guess > secret_number:
            print("Quá lớn! Thử lại.")
        else:
            print(f"Chúc mừng! Bạn đã đoán đúng số bí mật {secret_number}
sau {guess_count} lần đoán.")
            break # Thoát vòng lặp khi đoán đúng

    except ValueError:
        print("Lỗi: Vui lòng nhập một số nguyên hợp lệ.")
```

4.2.2. Viết chương trình yêu cầu nhập liệu cho đến khi người dùng nhập đúng định dạng

Mục tiêu: Sử dụng while để vòng lặp tiếp tục cho đến khi điều kiện nhập liệu được thỏa.

Bài tập: Tạo file input_validation.py. Viết chương trình yêu cầu người dùng nhập một địa chỉ email. Chương trình sẽ tiếp tục hỏi cho đến khi địa chỉ email nhập vào chứa cả ký tự @ và ký tự '.'.

Gợi ý:

```
# input_validation.py

print("--- NHẬP LIỆU ĐÚNG ĐỊNH DẠNG ---")
email = ""
while "@" not in email or "." not in email:
    email = input("Vui lòng nhập địa chỉ email của bạn (phải chứa '@' và
'.'): ")
    if "@" not in email or "." not in email:
        print("Địa chỉ email không hợp lệ. Vui lòng thử lại.")
    else:
        print(f"Địa chỉ email hợp lệ: {email}")
```

4.2.3. Thực hành sử dụng break và continue trong các kịch bản cụ thể

Mục tiêu: Hiểu rõ và áp dụng linh hoạt break và continue.

Bài tập: Tạo file loop_control_examples.py.

1./ Ví dụ break:

- In các số từ 1 đến 10. Dừng vòng lặp ngay lập tức nếu gặp số chia hết cho 7 (ví dụ: 7).

2./ Ví dụ continue:

- In các số từ 1 đến 10. Bỏ qua việc in ra những số chia hết cho 3.

Gợi ý:

```
# loop_control_examples.py
print("--- SỬ DỤNG BREAK ---")
for i in range(1, 11):
    if i % 7 == 0:
        print(f"Số {i} chia hết cho 7. Thoát vòng lặp.")
        break # Dừng ngay vòng lặp
    print(i)

print("\n--- SỬ DỤNG CONTINUE ---")
for i in range(1, 11):
    if i % 3 == 0:
        print(f"Bỏ qua số {i} (chia hết cho 3).")
        continue # Bỏ qua phần còn lại của lần lặp này, chuyển sang lần
lặp kế tiếp
    print(i)
```

4.3. Thực hành Git/GitHub cơ bản

Lưu ý quan trọng: Để thực hiện phần này, bạn cần cài đặt Git trên máy tính. Bạn có thể tải Git từ git-scm.com/downloads. Sau khi cài đặt, mở Terminal/Command Prompt và gõ git --version để kiểm tra.

4.3.1. Tạo tài khoản GitHub và kho lưu trữ mới

Mục tiêu: Thiết lập nền tảng để làm việc với Git/GitHub.

Hướng dẫn:

1./ **Đăng ký tài khoản GitHub:** Truy cập github.com và đăng ký một tài khoản miễn phí nếu bạn chưa có.

2./ Tạo một kho lưu trữ (Repository) mới:

- Sau khi đăng nhập, nhấp vào biểu tượng dấu + ở góc trên bên phải màn hình và chọn "New repository".

- Đặt tên cho Repository của bạn (ví dụ: my-first-python-repo).
- Chọn "Public" hoặc "Private" (Public để người khác xem được, Private chỉ mình bạn hoặc người được cấp quyền xem).
- **Tích chọn "Add a README file"** (Rất khuyến khích!). File README.md sẽ mô tả về dự án của bạn.
- Nhấp vào "Create repository".

4.3.2. Thực hành git clone, git add, git commit, git push với một dự án Python đơn giản

Mục tiêu: Áp dụng quy trình làm việc Git cơ bản.

Hướng dẫn:

1./ Clone Repository về máy tính:

- Trên trang GitHub của repository bạn vừa tạo, tìm nút "Code" màu xanh lá cây.
- Nhấp vào đó và sao chép URL HTTPS (ví dụ: https://github.com/your_username/my-first-python-repo.git).
- Mở Terminal hoặc Command Prompt trên máy tính của bạn.
- Di chuyển đến thư mục mà bạn muốn lưu dự án (ví dụ: cd Documents/Projects).
- Thực hiện lệnh git clone:

```
git clone https://github.com/your_username/my-first-python-repo.git
```

- Bây giờ, sẽ thấy một thư mục mới có tên my-first-python-repo trên máy tính của bạn.

2./ Thêm file Python mới:

- Di chuyển vào thư mục dự án vừa clone:

```
cd my-first-python-repo
```

- Mở thư mục này trong VS Code (File -> Open Folder...).
- Tạo một file Python mới bên trong thư mục này, ví dụ: hello_git.py.
- Viết vài dòng code Python vào đó:

```
# hello_git.py
print("Hello from Git and GitHub!")
print("This is my first Git commit.")
```

- Lưu file.

3./ Kiểm tra trạng thái thay đổi (git status):

- Trong Terminal của VS Code (đảm bảo bạn đang ở trong thư mục my-first-python-repo), gõ:

```
git status
```

- Bạn sẽ thấy hello_git.py xuất hiện dưới "Untracked files" (các file chưa được theo dõi).

4./ Thêm file vào khu vực staging (git add):

- Để chuẩn bị file hello_git.py cho commit:

```
git add hello_git.py
```

- Chạy git status lại. Bạn sẽ thấy hello_git.py nằm trong mục "Changes to be committed" (các thay đổi sẵn sàng để commit).

5./ Tạo commit (git commit):

- Ghi lại các thay đổi vào lịch sử cục bộ:

```
git commit -m "Add initial hello_git.py file"
```

- Bạn sẽ nhận được thông báo về commit của mình.

6./ Đẩy thay đổi lên GitHub (git push):

- Để đưa các commit cục bộ lên kho lưu trữ trên GitHub:

```
git push origin main # Hoặc master, tùy thuộc vào tên branch chính của bạn
```

- Bạn có thể sẽ được yêu cầu nhập tên người dùng và mật khẩu GitHub (hoặc Personal Access Token).

7./ Kiểm tra trên GitHub:

- Mở trình duyệt và truy cập lại trang GitHub của repository của bạn.
- Bạn sẽ thấy file hello_git.py đã xuất hiện trong danh sách file, và lịch sử commit của bạn đã được cập nhật.

4.3.3. Khám phá giao diện GitHub

Mục tiêu: Làm quen với các tính năng cơ bản của GitHub.

Hướng dẫn: Khi ở trên trang GitHub của repository của bạn:

- **"Code" tab:** Đây là nơi bạn thấy các file và thư mục của dự án.

- **"Commits" tab:** Nhấp vào đây để xem toàn bộ lịch sử các commit, mỗi commit có một thông báo và tác giả. Bạn có thể nhấp vào từng commit để xem chi tiết những thay đổi cụ thể.
- **"Branches" tab:** Xem các nhánh khác nhau trong dự án.
- **"Pull requests" / "Issues" tab:** Các tính năng quan trọng cho việc cộng tác và quản lý dự án (sẽ được học sâu hơn sau này).
- **"Settings" tab:** Cấu hình repository (ví dụ: đổi tên, xóa).

Hãy dành thời gian khám phá và làm quen với giao diện này, nó sẽ là công cụ bạn sử dụng rất thường xuyên trong hành trình lập trình của mình!