

BÀI 1: LẬP TRÌNH CƠ BẢN VỚI PYTHON

Bài học này sẽ giới thiệu về ngôn ngữ lập trình Python, cung cấp các kiến thức cơ bản về các kiểu dữ liệu, biến, biểu thức, cấu trúc chương trình và lập trình hướng đối tượng trong Python. Mục tiêu của bài này là giúp sinh viên quen thuộc với các khái niệm Python cơ bản để có thể thực hành và sử dụng trong các bài tiếp theo. Ngoài ra, bài này cũng giới thiệu và hướng dẫn sinh viên làm quen, thực hành với các công cụ hỗ trợ khi lập trình Python, hệ thống kiểm soát phiên bản và sử dụng công cụ kiểm thử cho các bài tập về bảo mật thông tin.

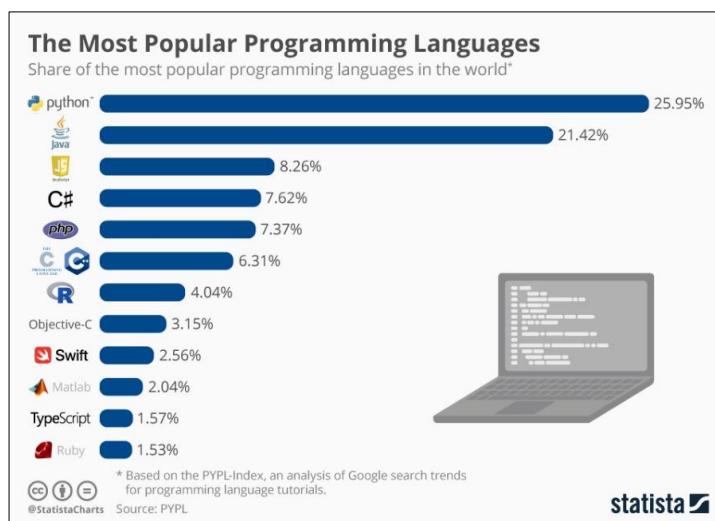
1.1 NGÔN NGỮ PYTHON

Python là một ngôn ngữ lập trình bậc cao được thông dịch, nổi bật với cú pháp rõ ràng và dễ hiểu, giúp lập trình viên tập trung vào việc giải quyết vấn đề mà không phải bận tâm quá nhiều về quy tắc ngữ pháp. Được phát triển bởi Guido van Rossum vào đầu những năm 90, Python nhanh chóng trở thành một trong những ngôn ngữ được ưa chuộng nhất toàn cầu.

Một trong những ưu điểm nổi bật của Python là tính linh hoạt, cho phép nó được sử dụng cho nhiều mục đích khác nhau như phát triển ứng dụng web, trí tuệ nhân tạo, khoa học dữ liệu, máy học, và phát triển game. Ngôn ngữ này còn có hàng loạt thư viện phong phú như NumPy, Pandas, TensorFlow, và Django, giúp mở rộng khả năng của nó.

Cộng đồng Python đóng vai trò quan trọng trong sự phát triển và lan tỏa của ngôn ngữ này. Với một cộng đồng lớn mạnh, người dùng không chỉ có thể tiếp cận nhiều tài nguyên phong phú mà còn nhận được sự hỗ trợ và chia sẻ kiến thức thông qua các diễn đàn, trang web học tập, sự kiện và các dự án mã nguồn mở.

Không chỉ linh hoạt và có cộng đồng hỗ trợ mạnh mẽ, Python còn được đánh giá cao về khả năng tích hợp dễ dàng với các ngôn ngữ và hệ thống khác. Nó cung cấp giao diện ứng dụng (API) linh hoạt, hỗ trợ nhiều nền tảng, và có khả năng mở rộng cao, điều này giúp việc phát triển các dự án phức tạp trở nên thuận lợi hơn.



So sánh độ phổ biến các ngôn ngữ lập trình

(Nguồn: <https://cdn.statcdn.com>)

Với tất cả những ưu điểm trên, Python là một ngôn ngữ thông dịch mạnh mẽ. Sự kết hợp giữa cú pháp dễ đọc, tính linh hoạt, và cộng đồng mạnh mẽ đã đặt Python ở vị thế dẫn đầu trong thế giới lập trình, khẳng định uy tín của mình trong nhiều lĩnh vực công nghệ hiện đại.

1.2 THÀNH PHẦN CỦA PYTHON

1.2.1 Biến, từ khóa

“Trong Python, biến (variables) được hiểu là một tên gọi dùng để chỉ một giá trị nhất định trong bộ nhớ. Có thể coi biến như một nhãn gán cho một giá trị dữ liệu cụ thể. Việc tạo ra biến rất đơn giản và không yêu cầu khai báo kiểu dữ liệu. Kiểu dữ liệu của biến sẽ được Python xác định tự động dựa trên giá trị mà chúng ta gán cho nó” [1].

Khi xem xét đoạn mã dưới đây, ta thấy rằng Python tự động nhận diện kiểu dữ liệu của biến theo giá trị được gán. Ngoài ra, biến trong Python có khả năng thay đổi giá

trị trong suốt quá trình thực thi chương trình, nghĩa là ta có thể cập nhật giá trị mới cho một biến đã được khởi tạo.

```
1 x = 10          # Biến x lưu trữ giá trị số nguyên 10
2 name = "Alice"    # Biến name lưu trữ chuỗi "Alice"
3 is_valid = True   # Biến is_valid lưu trữ giá trị boolean True
```

Từ khóa (keywords) trong Python là các từ đã được định nghĩa, có ý nghĩa đặc biệt trong ngôn ngữ. Các từ khóa này đã được dành riêng thực hiện một số chức năng cụ thể và không được sử dụng làm tên biến, nhãn khác trong chương trình. Ta không thể sử dụng một từ khóa như tên biến:

```
1 if = 5      # Lỗi! "if" là từ khóa, không thể sử dụng làm tên biến
```

Bảng dưới đây là danh sách các từ khóa của Python, các từ này đều có ý nghĩa đặc biệt và đã được dành riêng cho việc thực hiện các chức năng cụ thể:

STT	Từ khoá – Ý nghĩa	STT	Từ khoá – Ý nghĩa
1	False : Đại diện giá trị logic sai (Boolean).	2	None : Đại diện một giá trị không có hoặc thiếu.
3	None : Đại diện một giá trị không có hoặc thiếu.	4	and : Thực hiện phép toán logic "và".
5	as : Được sử dụng để đặt bí danh (alias) cho module hoặc các thành phần khác.	6	assert : Kiểm tra điều kiện, và nếu điều kiện sai, sẽ ném ra một lỗi (exception).
7	async : Được sử dụng khi định nghĩa các hàm bắt đồng bộ (asynchronous functions).	8	await : Sử dụng trong các hàm bắt đồng bộ để chờ kết quả trả về.
9	break : Kết thúc một vòng lặp (loop).	10	class : Được sử dụng để định nghĩa một lớp (class).

STT	Từ khoá – Ý nghĩa	STT	Từ khoá – Ý nghĩa
11	continue: Bỏ qua phần còn lại của vòng lặp hiện tại và tiếp tục với lần lặp tiếp theo.	12	def: Định nghĩa một hàm (function).
13	del: Xóa một biến, hoặc một phần tử ra khỏi danh sách (list).	14	elif: Viết tắt của "else if", dùng trong câu lệnh rẽ nhánh (conditional statement).
15	else: Dùng trong câu lệnh rẽ nhánh (conditional statement).	16	except: Xử lý các ngoại lệ (exceptions).
17	finally: Xác định khối mã sẽ được thực thi sau khi thực thi các khối try và except.	18	for: Được sử dụng để tạo vòng lặp for.
19	from: Được dùng khi import một phần (module hay package) từ một module.	20	global: Được dùng để xác định rằng biến nằm trong phạm vi toàn cục.
21	if: Sử dụng trong câu lệnh rẽ nhánh (conditional statement).	22	import: Dùng để import các module vào chương trình.
23	in: Kiểm tra một giá trị có tồn tại trong một chuỗi, danh sách, tuple.	24	is: Dùng để so sánh hai đối tượng xem chúng có cùng tham chiếu hay không.
25	lambda: Tạo các hàm nặc danh (anonymous functions).	26	nonlocal: Được dùng để xác định rằng biến nằm ở phạm vi không phải toàn cục cũng không phải là cục bộ.
27	not: Thực hiện phép toán logic "không".	28	or: Thực hiện phép toán logic "hoặc".

STT	Từ khoá – Ý nghĩa	STT	Từ khoá – Ý nghĩa
29	pass: Đánh dấu để bỏ qua đoạn mã mà không làm gì cả.	30	raise: Sử dụng để ném một ngoại lệ.
31	return: Dùng để trả về kết quả từ một hàm.	32	try: Dùng để thực hiện các lệnh có khả năng gây ra những ngoại lệ khác nhau (exceptions).
33	while: Được sử dụng để tạo vòng lặp while.	34	with: Dùng khi làm việc với các nguồn tài nguyên được quản lý tự động.
35	yield: Dùng trong hàm để trả về giá trị, lưu trạng thái của hàm đó.		

1.2.2 Kiểu dữ liệu

“Python hỗ trợ nhiều loại dữ liệu khác nhau để lưu trữ và xử lý thông tin” [1], [2]. Dưới đây là một số kiểu dữ liệu cơ bản và thông dụng trong Python:

- **Kiểu dữ liệu số (Numeric Types):**

- “**int**”: Kiểu dữ liệu số nguyên, lưu trữ số nguyên không có dấu thập phân.
- “**float**”: Kiểu dữ liệu số thực, lưu trữ số có dấu thập phân.
- “**complex**”: Kiểu dữ liệu số phức, lưu trữ số phức với phần thực, phần ảo.

- **Kiểu dữ liệu chuỗi (String Type):**

- “**str**”: Kiểu dữ liệu chuỗi, lưu trữ chuỗi ký tự, được bao quanh bởi ‘ ’ hoặc “ ”.

- **Kiểu dữ liệu Boolean (Boolean Type):**

- “**bool**”: Kiểu boolean chỉ có hai giá trị là True hoặc False. Được dùng để biểu diễn các trạng thái logic.

- **Kiểu dữ liệu danh sách (List Type):**

- “**list**”: Kiểu dữ liệu danh sách lưu trữ tập hợp các phần tử có thứ tự và có thể thay đổi.

- **Kiểu dữ liệu tuple (Tuple Type):**
 - “**tuple**”: Kiểu dữ liệu tuple tương tự như danh sách nhưng không thể thay đổi (immutable).
- **Kiểu dữ liệu từ điển (Dictionary Type):**
 - “**dict**”: Kiểu dữ liệu lưu trữ các cặp key-value không có thứ tự. Key là duy nhất, không thay đổi; value có thể thay đổi.
- **Kiểu dữ liệu tập hợp (Set Type):**
 - “**set**”: Đây là kiểu dữ liệu dùng để lưu trữ một tập hợp các phần tử không trùng lặp và không có thứ tự.
- **Kiểu dữ liệu bộ nhớ đóng (Frozen Set Type):**
 - “**frozenset**”: Tương tự như kiểu dữ liệu set nhưng không thể thay đổi (immutable).
- **Kiểu dữ liệu nhị phân (Binary Types):**
 - “**bytes**”: Lưu trữ một chuỗi byte.
 - “**bytearray**”: Tương tự như bytes nhưng có thể thay đổi (mutable).
- **Kiểu dữ liệu None (None Type):**
 - **NoneType**: Chỉ có một giá trị là “**None**”, được dùng để biểu diễn giá trị rỗng hoặc không tồn tại.

1.2.3 Các toán tử số học

“Trong Python, các toán tử số học được sử dụng để thực hiện các phép toán cơ bản trên các số. Các toán tử này cho phép bạn thực hiện các phép cộng, trừ, nhân, chia, các phép tính khác trên các biến hoặc giá trị số” [1], [2]. Sau đây là các toán tử số học cơ bản trong Python:

- **Cộng (+):** Toán tử cộng được dùng để thực hiện phép cộng giữa hai số.

```

1 a = 5
2 b = 3
3 result = a + b # Kết quả: 8
  
```

- **Trừ (-):** Toán tử trừ được dùng để thực hiện phép trừ giữa hai số.

```

1 a = 8
2 b = 4
3 result = a - b # Kết quả: 4

```

- **Nhân (*):** Toán tử nhân được dùng để thực hiện phép nhân giữa hai số.

```

1 a = 6
2 b = 7
3 result = a * b # Kết quả: 42

```

- **Chia (/):** Toán tử chia được dùng để thực hiện phép chia.

```

1 a = 20
2 b = 5
3 result = a / b # Kết quả: 4.0 (Kết quả luôn là một số thập
                  phân nếu có phần dư)

```

- **Chia lấy phần nguyên (//):** Toán tử chia lấy phần nguyên trả về kết quả là phần nguyên của phép chia.

```

1 a = 20
2 b = 3
3 result = a // b # Kết quả: 6

```

- **Chia lấy dư (%):** Toán tử này sẽ cung cấp phần còn lại sau khi thực hiện phép chia giữa hai số.

```

1 a = 20
2 b = 7
3 remainder = a % b # Kết quả: 6 (Phần dư của 20 chia cho 7)

```

- **Luỹ thừa (**):** Toán tử luỹ thừa được sử dụng để tính toán lũy thừa của một số.

```

1 a = 2
2 b = 3
3 result = a ** b # Kết quả: 8 (2^3 = 8)

```

Các toán tử số học này có thể được dùng trong các biểu thức để thực hiện các phép tính số học cơ bản. Đồng thời, Python cũng hỗ trợ việc sử dụng các dấu ngoặc để xác định ưu tiên của các phép tính, giúp thực hiện các phép toán chính xác theo ý muốn.

1.2.4 Các toán tử logic

"Trong Python, các toán tử logic được sử dụng để thực hiện phép toán logic trên các giá trị boolean (True hoặc False) hoặc các biểu thức logic. Các toán tử logic cho phép chúng ta kiểm tra và thực hiện các phép so sánh, kết hợp các điều kiện để quyết định kết quả logic" [1], [2]. Sau đây sẽ trình bày một vài toán tử logic cơ bản trong Python:

- **Phép toán AND:** Toán tử “**and**” trả về “True” nếu cả hai điều kiện đều đúng.

```
1 x = 5
2 y = 3
3 result = (x > 2) and (y < 4) # Kết quả: True
```

- **Phép toán OR:** Toán tử “**or**” trả về “True” nếu ít nhất một trong hai điều kiện đầu vào là đúng.

```
1 x = 5
2 y = 3
3 result = (x > 2) or (y > 4) # Kết quả: True
```

- **Phép toán NOT:** Toán tử “**not**” trả về “True” nếu điều kiện là “False” và ngược lại.

```
1 x = 5
2 result = not (x == 5) # Kết quả: False
```

- **Phép so sánh bằng (==):** Toán tử “**==**” sẽ so sánh hai giá trị có bằng nhau hay không.

```
1 x = 5
2 result = (x == 5) # Kết quả: True
```

- **Phép so sánh không bằng (**`!=`**):** Toán tử "`!=`" sẽ so sánh hai giá trị có khác nhau hay không.

```
1 x = 5
2 result = (x != 3) # Kết quả: True
```

- **Phép so sánh lớn hơn (**`>`**), nhỏ hơn (**`<`**):** Toán tử "`>`" sẽ so sánh xem giá trị bên trái lớn hơn giá trị bên phải hay không và toán tử "`<`" sẽ so sánh xem giá trị bên trái nhỏ hơn giá trị bên phải hay không.

```
1 x = 5
2 result1 = (x > 3) # Kết quả: True
3 result2 = (x < 3) # Kết quả: False
```

- **Phép so sánh lớn hơn hoặc bằng (**`>=`**), nhỏ hơn hoặc bằng (**`<=`**):** Toán tử "`>=`" sẽ so sánh xem giá trị bên trái lớn hơn hoặc bằng giá trị bên phải hay không và toán tử "`<=`" sẽ so sánh xem giá trị bên trái nhỏ hơn hoặc bằng giá trị bên phải hay không.

```
1 x = 5
2 result1 = (x >= 3) # Kết quả: True
3 result2 = (x <= 3) # Kết quả: False
```

1.2.5 Nhập, xuất dữ liệu

Trong Python, các hàm nhập và xuất dữ liệu chủ yếu là "**`input()`**" và "**`print()`**".

- **Hàm "`input()`"** được dùng để nhập dữ liệu từ bàn phím. Nó cho phép nhập giá trị và trả về một chuỗi (string) biểu diễn giá trị đã nhập. Ta có thể lưu giá trị này vào biến để sử dụng sau này. Ví dụ:

```
1 name = input("Nhập tên của bạn: ")
2 print("Xin chào,", name)
```

- **Hàm "`print()`"** được sử dụng để hiển thị dữ liệu ra màn hình hoặc console. Ta có thể truyền biến, chuỗi, hoặc giá trị cần hiển thị vào hàm "**`print()`**". Ví dụ:

```
1 age = 25
2 print("Tuổi của bạn là:", age)
```

Ngoài ra, “**print()**” cũng cho phép định dạng hiển thị qua việc sử dụng các đối số khác như “**sep**” (ký tự phân cách), “**end**” (ký tự kết thúc), và các chuỗi định dạng f-string. Ví dụ:

```
1 print("Python", "là", "ngôn", "ngữ", "lập", "trình", sep="-")
2 # Kết quả: Python-là-ngôn-ngữ-lập-trình
3 print("Xin chào", end=" ")
4 print("các bạn!") # Kết quả: Xin chào các bạn!
```

1.2.6 Các cấu trúc điều khiển

Cấu trúc điều khiển là cách thức quản lý luồng của chương trình, quyết định việc thực thi các khối mã khác nhau dựa trên điều kiện hoặc vòng lặp. Các cấu trúc này cung cấp khả năng kiểm soát các điều kiện logic, lặp lại các hoạt động, thực hiện các quyết định trong đoạn mã. Có ba kiểu cấu trúc điều khiển cơ bản trong Python:

- **Câu lệnh điều kiện (Conditional Statements):** Cho phép kiểm tra điều kiện, thực hiện các hành động khác nhau dựa vào kết quả của điều kiện đó. Câu lệnh điều kiện phổ biến nhất trong Python là câu lệnh “**if**”, “**else**”, và “**elif**” (else if). Ví dụ:

```
1 x = 10
2 if x > 5:
3     print("x lớn hơn 5")
4 elif x == 5:
5     print("x bằng 5")
6 else:
7     print("x nhỏ hơn 5")
```

- **Vòng lặp (Loops):** Vòng lặp sẽ cho phép sự lặp lại việc thực hiện một khối mã cho đến khi một điều kiện nào đó được thỏa mãn. Trong Python, hai loại vòng lặp phổ biến nhất là vòng lặp “**for**” và “**while**”.

- Vòng lặp “**for**” được dùng để duyệt qua một chuỗi hoặc một tập hợp các phần tử. Ví dụ:

```

1 fruits = ["apple", "banana", "cherry"]
2 for fruit in fruits:
3     print(fruit)

```

- Vòng lặp “**while**” thực hiện việc lặp lại mã đến khi điều kiện không còn đúng nữa. Ví dụ:

```

1 count = 0
2 while count < 5:
3     print(count)
4     count += 1

```

- Câu lệnh nhảy (Jump Statements):** Câu lệnh nhảy được dùng để thay đổi luồng điều khiển của chương trình. Các câu lệnh nhảy phổ biến trong Python bao gồm “**break**”, “**continue**” và “**pass**”.

- Sử dụng câu lệnh “**break**” để kết thúc một vòng lặp:

```

1 # Tìm số chia hết cho 5 đầu tiên trong khoảng từ 1 đến 100
2 for i in range(1, 101):
3     if i % 5 == 0:
4         print("Số chia hết cho 5 đầu tiên là:", i)
5         break

```

- Câu lệnh “**continue**” được dùng để bỏ qua phần còn lại của vòng lặp hiện tại và chuyển sang vòng lặp kế tiếp:

```

1 # In các số chẵn từ 1 đến 10 và bỏ qua các số lẻ
2 for i in range(1, 11):
3     if i % 2 != 0:
4         continue
5     print(i)

```

- Sử dụng câu lệnh “**pass**” như là một tuyên bố rỗng:

```

1 # Kiểm tra điều kiện, nếu đúng thực hiện, nếu sai thì
  không làm gì
2 x = 5
3 if x > 10:
4     print("x lớn hơn 10")
5 else:
6     pass

```

1.2.7 Chuỗi

"Trong Python, chuỗi (String) được định nghĩa là một tập hợp các ký tự, được bao bọc bởi cặp dấu nháy đơn (' ') hoặc cặp dấu nháy kép (" "). Chuỗi có thể bao gồm bất kỳ loại ký tự nào, chẳng hạn như chữ cái, số và các ký tự đặc biệt" [1], [2].

- **Khai báo chuỗi trong Python:**

```

1 # Sử dụng dấu ngoặc đơn
2 string_single_quotes = 'Đây là một chuỗi sử dụng dấu ngoặc đơn.'
3 # Sử dụng dấu ngoặc kép
4 string_double_quotes = "Đây là một chuỗi sử dụng dấu ngoặc kép."
5 # Sử dụng dấu ngoặc ba
6 string_triple_quotes = '''Đây là một chuỗi
7 sử dụng dấu ngoặc ba,
8 có thể trải dài qua nhiều dòng.'''

```

- **Truy cập ký tự trong chuỗi:** Chúng ta có thể truy cập các ký tự trong chuỗi bằng cách sử dụng chỉ số của chúng trong cặp dấu ngoặc vuông []. Chú ý rằng chỉ số trong Python bắt đầu từ 0. Ví dụ:

```

1 my_string = "Hello, World!"
2 print(my_string[0]) # Kết quả: 'H'
3 print(my_string[7]) # Kết quả: 'W'

```

- **Các phép xử lý chuỗi trong Python:**

- *Cắt chuỗi (Slicing):* Cắt chuỗi là quá trình lấy một phần của chuỗi sử dụng chỉ mục hoặc phạm vi chỉ mục.

```

1 my_string = "Hello, World!"
2 print(my_string[7:]) # Lấy từ ký tự thứ 7 đến hết: Kết quả: 'World!'
3 print(my_string[:5]) # Lấy từ đầu đến ký tự thứ 4: Kết quả: 'Hello'
4 print(my_string[3:8])# Lấy từ ký tự thứ 3 đến ký tự thứ 7: Kết quả:
'lo, W'

```

- *Ghép chuỗi (Concatenation)*: Ghép chuỗi là quá trình nối chuỗi lại với nhau.

```

1 string1 = "Hello"
2 string2 = "World"
3 concatenated_string = string1 + " " + string2 # Kết quả: 'Hello
World'

```

- *Độ dài chuỗi (Length)*: Hàm “**len()**” được dùng để tính độ dài của chuỗi.

```

1 my_string = "Hello, World!"
2 length = len(my_string) # Kết quả: 13

```

- **Một số hàm dùng để xử lý các chuỗi trong Python:**

- “**upper()**”: Chuyển đổi chuỗi thành chữ hoa.
- “**lower()**”: Chuyển đổi chuỗi thành chữ thường.
- “**strip()**”: Loại bỏ khoảng trắng ở đầu và cuối chuỗi.
- “**split()**”: Phân tách chuỗi thành danh sách các từ hoặc phần tử.
- “**replace()**”: Thay thế một phần của chuỗi bằng một chuỗi khác.

```

1 my_string = "Hello, World! "
2 print(my_string.strip()) # Loại bỏ khoảng trắng: Kết quả: 'Hello,
World!'
3
4 my_string = "Hello, World!"
5 print(my_string.split(","))
6 # Phân tách chuỗi: Kết quả: ['Hello', ' World!']
7
8 my_string = "Hello, World!"
9 print(my_string.replace("Hello", "Hi"))
10 # Thay thế chuỗi: Kết quả: 'Hi, World!'

```

1.2.8 Hàm (Function)

"Hàm trong Python là một đoạn mã có thể được gọi để thực hiện một nhiệm vụ nhất định, cho phép tổ chức mã thành các phần nhỏ hơn, tái sử dụng mã và làm cho chương trình trở nên dễ đọc hơn" [1], [2].

- **Khai báo hàm:** Để định nghĩa một hàm trong Python, ta sử dụng từ khóa "def", tiếp theo là tên hàm và danh sách tham số (nếu có), sau đó là một khối mã được thụt lề bên trong dấu hai chấm.

```
1 def my_function(parameter1, parameter2):  
2     # Khối mã của hàm  
3     # Thực hiện các hoạt động dựa trên tham số được truyền vào  
4     result = parameter1 + parameter2  
5     return result
```

- **Phân loại hàm:**

- *Hàm có giá trị trả về:* Một hàm có khả năng trả lại giá trị thông qua từ khóa "**return**". Nếu không có lệnh "**return**" trong hàm, nó sẽ tự động trả về "**None**" như giá trị mặc định.
- *Hàm không có giá trị nào trả về:* Một số hàm chỉ thực hiện một công việc nhất định mà không cần trả về bất kỳ giá trị nào.

- **Cách sử dụng hàm:**

- *Gọi hàm:* Để sử dụng hàm, chỉ cần gọi tên hàm cùng với các đối số cần thiết (nếu có). Ví dụ:

```
6 result = my_function(10, 20) # Gọi hàm và lưu kết quả vào biến result  
7 print(result) # In kết quả của hàm
```

- *Tham số và đối số:* "Tham số" là các biến được định nghĩa trong định nghĩa hàm, còn "đối số" là giá trị được truyền cho tham số khi gọi hàm.

Ví dụ về khai báo và gọi hàm trong Python:

```

1 # Định nghĩa hàm tính tổng
2 def calculate_sum(a, b):
3     result = a + b
4     return result
5 # Gọi hàm và lưu kết quả vào biến
6 sum_result = calculate_sum(10, 20)
7 # In kết quả
8 print("Tổng hai số là:", sum_result)

```

Ví dụ khai báo và gọi hàm không có giá trị trả về:

```

1 # Hàm không trả về giá trị, chỉ in ra thông báo
2 def greet(name):
3     print("Xin chào,", name)
4 # Gọi hàm
5 greet("Alice")

```

1.3 KIỂU DỮ LIỆU CÓ CẤU TRÚC

1.3.1 Mảng (Array)

“Trong Python, module “**array**” cung cấp một cấu trúc dữ liệu cơ bản gọi là mảng (array) cho phép lưu trữ cùng loại dữ liệu. Mảng trong module array giới hạn các phần tử bên trong mảng phải cùng kiểu dữ liệu” [1], [2].

- Để sử dụng mảng trong Python, chúng ta cần import module “**array**”:

```

1 from array import array

```

- Khai báo mảng trong module “**array**”: Có hai tham số quan trọng khi khai báo một mảng:

- *Typecode*: Đây là một ký tự xác định kiểu dữ liệu của các phần tử trong mảng.
- *Initializer*: Đây là danh sách các giá trị ban đầu của mảng.

Ví dụ:

```

1 from array import array
2 # Khai báo một mảng số nguyên
3 int_array = array('i', [1, 2, 3, 4, 5])
4 # Khai báo một mảng số thực
5 float_array = array('f', [3.14, 2.5, 6.7])

```

16

BÀI 1: LẬP TRÌNH CƠ BẢN VỚI PYTHON

- Các kiểu dữ liệu (Typecodes) trong module array:
 - 'b': signed char (1 byte)
 - 'B': unsigned char (1 byte)
 - 'i': signed int (2 bytes)
 - 'I': unsigned int (2 bytes)
 - 'f': float (4 bytes)
 - 'd': double (8 bytes)
 - và nhiều typecodes khác.
- Phương thức và thuộc tính của mảng:
 - Truy cập phần tử trong mảng: Truy cập các phần tử trong mảng bằng cách sử dụng chỉ số của chúng.

```
7 print(int_array[0]) # Truy cập phần tử đầu tiên của mảng số nguyên  
8 print(float_array[2]) # Truy cập phần tử thứ ba của mảng số thực
```

- Cập nhật giá trị của phần tử trong mảng:

```
10 int_array[2] = 10  
11 # Cập nhật giá trị của phần tử thứ ba trong mảng số nguyên
```

- Sử dụng các phương thức của mảng: Module “**array**” cung cấp một số phương thức để thực hiện các thao tác với mảng:
 - “append()”: Thêm một phần tử vào cuối mảng.
 - “insert()”: Chèn một phần tử vào vị trí chỉ định.
 - “remove()”: Xóa phần tử ở vị trí đầu tiên có giá trị xác định.
 - “pop()”: Xóa phần tử ở vị trí chỉ định và trả về giá trị của phần tử đó.
 - “extend()”: Mở rộng mảng bằng cách thêm các phần tử từ một mảng khác.
 - “index()”: Trả về chỉ số đầu tiên của một phần tử cụ thể.
 - “count()”: Đếm số lần xuất hiện của một phần tử trong mảng.

Ví dụ:

```
13 int_array.append(6) # Thêm phần tử 6 vào cuối mảng số nguyên
14 float_array.remove(6.7) # Xóa phần tử 6.7 khỏi mảng số thực
```

1.3.2 Danh sách (List)

"Trong Python, kiểu dữ liệu danh sách (List) thường được sử dụng để lưu trữ và làm việc với các tập hợp dữ liệu có thứ tự. Mặc dù không có một kiểu dữ liệu mảng cố định như trong một số ngôn ngữ khác, nhưng danh sách trong Python có thể được sử dụng tương đương với mảng vì chúng có khả năng lưu trữ nhiều loại dữ liệu và có thể thay đổi kích thước (động)" [1], [2].

- **Khai báo một danh sách (List):**

```
1 # Danh sách số nguyên
2 my_list = [1, 2, 3, 4, 5]
3 # Danh sách chuỗi
4 names = ["Alice", "Bob", "Charlie"]
5 # Danh sách kết hợp kiểu dữ liệu
6 mixed_list = [10, "hello", 3.14, True]
```

- **Cách sử dụng danh sách:**

- Truy cập vào một phần tử ở trong danh sách:

```
8 print(my_list[0]) # Truy cập phần tử đầu tiên: Kết quả: 1
9 print(names[2]) # Truy cập phần tử thứ ba: Kết quả: 'Charlie'
```

- Cập nhật giá trị của một phần tử ở trong danh sách:

```
11 my_list[1] = 20 # Thay đổi giá trị của phần tử thứ hai
12 print(my_list) # Kết quả: [1, 20, 3, 4, 5]
```

- Thêm một phần tử vào danh sách:

```
14 names.append("David") # Thêm phần tử vào cuối danh sách
15 print(names) # Kết quả: ['Alice', 'Bob', 'Charlie', 'David']
```

- Xóa một phần tử khỏi danh sách:

```
17 del my_list[2] # Xóa phần tử thứ ba khỏi danh sách
18 print(my_list) # Kết quả: [1, 20, 4, 5]
```

- Duyệt qua từng phần tử trong danh sách:

```
20 for element in names:  
21     print(element) # In từng phần tử trong danh sách
```

- **Lưu ý khi làm việc với danh sách trong Python:**

- Chỉ số trong danh sách bắt đầu từ 0.
- Danh sách có thể chứa nhiều kiểu dữ liệu khác nhau.
- Có thể thay đổi kích thước danh sách bằng cách thêm, xóa, thay đổi phần tử.
- Có nhiều phương thức và hàm sẵn có trong Python để làm việc với danh sách như **append()**, **remove()**, **pop()**, **insert()**, **sort()**, **len()**, và nhiều hơn nữa.

1.3.3 Kiểu Tuple

“Tuple là một dạng dữ liệu quan trọng dùng để lưu trữ các tập hợp thông tin không thể thay đổi (immutable). Đặc điểm của nó là sử dụng dấu ngoặc đơn và các phần tử trong Tuple sẽ được phân cách bằng dấu phẩy” [1], [2].

- **Đặc điểm của Tuple:**

- *Không thay đổi (Immutable)*: Sau khi tạo tuple sẽ không thay đổi giá trị của các phần tử trong tuple. Điều này khác với danh sách (list), nơi có thể thay đổi giá trị của các phần tử.
- *Có thứ tự*: Các phần tử trong tuple được xác định bằng vị trí của chúng, giống như danh sách.
- *Có thể chứa nhiều kiểu dữ liệu*: Tuple có thể chứa các phần tử với các kiểu dữ liệu khác nhau.

- **Ưu điểm của Tuple:**

- Sử dụng Tuple khi muốn đảm bảo các dữ liệu không thay đổi và không bị sửa đổi sau khi đã khởi tạo.
- Tốc độ truy cập phần tử trong Tuple thường nhanh hơn so với danh sách.

- **Khai báo Tuple:**

Có thể báo một tuple bằng cách sử dụng dấu ngoặc đơn và phân tách các phần tử bằng dấu phẩy. Ví dụ:

```

1 # Tuple các số nguyên
2 my_tuple = (1, 2, 3, 4, 5)
3 # Tuple các chuỗi
4 names = ("Alice", "Bob", "Charlie")
5 # Tuple kết hợp kiểu dữ liệu
6 mixed_tuple = (10, "hello", 3.14)

```

- **Truy cập vào phần tử trong Tuple:**

Tương tự như danh sách, có thể truy cập các phần tử trong Tuple bằng cách sử dụng chỉ số của chúng. Ví dụ:

```

8 print(my_tuple[0]) # Truy cập phần tử đầu tiên: Kết quả: 1
9 print(names[2]) # Truy cập phần tử thứ ba: Kết quả: 'Charlie'

```

- **Các phương thức trong Tuple:**

- **count(value):** Đếm số lần một giá trị cụ thể trong tuple xuất hiện. Ví dụ:

```

1 my_tuple = (1, 2, 3, 1, 4, 1)
2 print(my_tuple.count(1)) # Kết quả: 3 (1 xuất hiện 3 lần trong tuple)

```

- **index(value):** Trả về chỉ số đầu tiên của một giá trị cụ thể trong tuple. Ví dụ:

```

1 my_tuple = ('a', 'b', 'c', 'd', 'b')
2 print(my_tuple.index('b'))
3 # Kết quả: 1 (chỉ số đầu tiên của 'b' trong tuple là 1)

```

Cần lưu ý rằng vì tính không thay đổi của Tuple, nó không cung cấp các phương thức như thêm mới phần tử (**append()**), xóa phần tử (**remove()**), hoặc sắp xếp (**sort()**) như danh sách (list) trong Python. Điều này làm cho Tuple không linh hoạt nhưng an toàn khi cần bảo vệ các dữ liệu mà không muốn chúng bị thay đổi sau khi đã khởi tạo.

1.3.4 Kiểu Dictionary

"Trong Python, "**dictionary**" là một kiểu dữ liệu cấu trúc, mạnh mẽ được sử dụng để lưu trữ các cặp key-value không có thứ tự. Dictionary được biểu diễn bằng dấu ngoặc nhọn "{}" và mỗi cặp key-value được phân tách bởi dấu hai chấm ":". Key trong dictionary phải là duy nhất và không thể thay đổi, trong khi value có thể là bất kỳ kiểu dữ liệu nào" [1], [2].

- **Khai báo dictionary:**

```
1 # Khai báo một dictionary rỗng
2 my_dict = {}
3 # Khai báo một dictionary với các cặp key-value
4 person = {"name": "Alice", "age": 25, "city": "New York"}
```

- **Truy cập vào giá trị trong dictionary:**

Có thể truy cập giá trị trong dictionary bằng cách sử dụng key tương ứng, ví dụ:

```
6 print(person["name"]) # In giá trị của key "name": Kết quả: "Alice"
7 print(person["age"]) # In giá trị của key "age": Kết quả: 25
```

- - **Thêm hoặc cập nhật giá trị trong dictionary:**

```
9 # Thêm một cặp key-value mới
10 person["email"] = "alice@example.com"
11 # Cập nhật giá trị của key đã tồn tại
12 person["age"] = 26
```

- **Xóa một phần tử trong dictionary:**

```
14 # Xóa một cặp key-value từ dictionary
15 del person["city"]
16 # Xóa phần tử và lấy giá trị của key
17 age = person.pop("age")
```

- **Các phương thức và phương thức dành cho dictionary:**

- “keys()”: Trả về tất cả các keys trong dictionary.
- “values()”: Trả về tất cả các values trong dictionary.
- “items()”: Trả về tất cả các cặp key-value trong dictionary dưới dạng tuple.

Ví dụ:

```
19 print(person.keys()) # In ra tất cả các keys trong dictionary
20 print(person.values()) # In ra tất cả các values trong dictionary
21 print(person.items()) # In ra tất cả các cặp key-value trong
dictionary
```

- **Lưu ý:**

- Dictionary trong Python không có thứ tự, nghĩa là không giữ trật tự của các phần tử đã được thêm vào.
- Key trong dictionary phải là duy nhất, không thể thay đổi. Còn Value có thể là bất kỳ một kiểu dữ liệu nào.

1.4 OOP TRONG PYTHON

1.4.1 Lớp (Class) và đối tượng (Object)

"Python hỗ trợ OOP hoàn toàn và có thể được sử dụng để viết mã sử dụng các khái niệm hướng đối tượng như lớp (class), đối tượng (object), kế thừa (inheritance), đa hình (polymorphism), trừu tượng hóa (abstraction) [1], [2]".

Lớp (Class): Là một khuôn mẫu để tạo ra các đối tượng. Nó chứa các thuộc tính (attributes) và phương thức (methods).

Đối tượng (Object): Là một thể hiện cụ thể của một lớp, có thể được tạo ra từ lớp bằng cách sử dụng từ khóa class.

Ví dụ sau trình bày về lớp và đối tượng trong Python:

```

1 # Định nghĩa một lớp đơn giản
2 class Car:
3     def __init__(self, brand, model):
4         self.brand = brand
5         self.model = model
6     def get_info(self):
7         return f"{self.brand} {self.model}"
8 # Tạo đối tượng từ lớp Car
9 my_car = Car("Toyota", "Corolla")
10 # Gọi phương thức của đối tượng
11 print(my_car.get_info()) # Kết quả: Toyota Corolla

```

Ở ví dụ trên, "Car" là một lớp đại diện cho các đối tượng xe ô tô. "brand" và "model" là các thuộc tính của lớp, và "get_info" là một phương thức để trả về thông tin về xe ô tô.

1.4.2 Khởi tạo (Constructor)

Trong Python, constructor được định nghĩa bằng phương thức “`__init__()`” trong lớp. Constructor này tự động được gọi khi khởi tạo một đối tượng từ lớp đó. Cú pháp của Constructor trong Python:

```
1 class ClassName:
2     def __init__(self, parameter1, parameter2, ...):
3         self.parameter1 = parameter1
4         self.parameter2 = parameter2
5         # ...
```

Trong đó:

- “`__init__()`” là tên của constructor trong Python.
- “`self`” là tham số đặc biệt đại diện cho chính đối tượng được tạo ra từ lớp.
- “`parameter1`”, “`parameter2`”,... là các tham số bạn muốn truyền vào khi tạo đối tượng.

Ví dụ sau trình bày về Constructor trong Python:

```
1 class Car:
2     def __init__(self, brand, model):
3         self.brand = brand
4         self.model = model
5
6     def get_info(self):
7         return f"{self.brand} {self.model}"
8 # Tạo đối tượng từ lớp Car và truy cập thông tin
9 my_car = Car("Toyota", "Corolla")
10 print(my_car.get_info()) # Kết quả: Toyota Corolla
```

Trong ví dụ trên, “`__init__()`” là constructor của lớp Car. Khi một đối tượng được tạo từ lớp Car, constructor này tự động được gọi. Trong constructor, “`self.brand = brand`” và “`self.model = model`” được sử dụng để gán một giá trị cho các thuộc tính của đối tượng “`my_car`”.

1.4.3 Thuộc tính (Attributes)

Attribute (thuộc tính) là các biến được liên kết với đối tượng của một lớp.

- **Phân loại thuộc tính trong Python:**

- **Thuộc tính instance (Instance Attributes):** Là các thuộc tính chỉ thuộc về một đối tượng cụ thể của một lớp.
- **Thuộc tính lớp (Class Attributes):** Là các thuộc tính thuộc về toàn bộ lớp, không chỉ thuộc về một đối tượng cụ thể nào đó. Chúng được chia sẻ giữa tất cả các đối tượng của lớp.

- **Định nghĩa thuộc tính trong Python:**

```

1 class ClassName:
2     def __init__(self, attribute1, attribute2):
3         self.attribute1 = attribute1      # Thuộc tính instance
4         self.attribute2 = attribute2      # Thuộc tính instance
5     class_attribute = "Class Attribute" # Thuộc tính lớp

```

Trong đó:

- “**attribute1**”, “**attribute2**”: Là các thuộc tính instance, được gán giá trị khi một đối tượng được tạo.
- “**class_attribute**”: Là một thuộc tính lớp, có thể truy cập thông qua tất cả các đối tượng của lớp.

- **Truy cập thuộc tính trong Python:**

```

7 # Tạo đối tượng từ lớp và truy cập các thuộc tính
8 object_name = ClassName(value1, value2)
9 print(object_name.attribute1)      # Truy cập thuộc tính instance
10 print(object_name.class_attribute) # Truy cập thuộc tính lớp

```

1.4.4 Phương thức (Methods)

Fương thức (methods) là chỉ các hàm được định nghĩa bên trong một lớp và được gắn liền với các đối tượng của lớp đó.

- **Định nghĩa một phương thức trong Python:**

```
1 class ClassName:
2     def method_name(self, parameter1, parameter2):
3         # Các thao tác hoặc xử lý
4         return something # Trả về kết quả (nếu cần)
```

Trong đó:

- “**method_name**”: Tên của phương thức.
- “**self**”: Tham chiếu đến chính đối tượng của lớp. self là bắt buộc và được dùng để truy cập các thuộc tính và phương thức khác trong cùng lớp.
- “**parameter1**”, “**parameter2**”: Các tham số mà phương thức có thể nhận (tùy chọn).
- “**return**”: Cho biết phương thức trả về một giá trị nếu cần.

- **Truy cập phương thức trong Python:**

```
6 # Tạo đối tượng từ lớp và gọi phương thức
7 object_name = ClassName()
8 # Gọi phương thức và truyền các tham số
9 object_name.method_name(value1, value2)
```

1.4.5 Kế thừa (Inheritance)

Trong Python, kế thừa (inheritance) là khả năng của một lớp con (child class) kế thừa thuộc tính và phương thức từ một lớp cha (parent class). Tuy nhiên, đa kế thừa (multiple inheritance) sẽ cho phép một lớp con kế thừa từ nhiều lớp cha.

- **Kế thừa đơn (Single Inheritance):**

```
1 class ParentClass:
2     # Định nghĩa các thuộc tính và phương thức của lớp cha
3
4 class ChildClass(ParentClass):
5     # Định nghĩa các thuộc tính và phương thức mới hoặc mở rộng từ
       lớp cha
```

- **Đa kế thừa (Multiple Inheritance):**

```

1 class ParentClass1:
2     # Định nghĩa các thuộc tính và phương thức của lớp cha 1
3 class ParentClass2:
4     # Định nghĩa các thuộc tính và phương thức của lớp cha 2
5 class ChildClass(ParentClass1, ParentClass2):
6     # Định nghĩa các thuộc tính và phương thức mới hoặc mở rộng từ
    các lớp cha

```

- **Lợi ích của kế thừa:**

- Tái sử dụng mã: Kế thừa cho phép sử dụng lại mã nguồn có sẵn từ lớp cha.
- Tính mở rộng: Lớp con có thể mở rộng hoặc thay đổi hành vi của lớp cha.
- Tổ chức mã nguồn: Kế thừa giúp tổ chức mã nguồn theo cấu trúc phân cấp, dễ dàng hiểu và bảo trì.

1.4.6 Đa hình (Polymorphism)

Đa hình (Polymorphism) là khả năng của các đối tượng có thể hiện thực hành vi khác nhau thông qua cùng một phương thức hoặc tên hàm. Phân loại:

- **Đa hình ở thời điểm biên dịch (Compile-time Polymorphism):**

Được triển khai thông qua kỹ thuật “**overloading**” và “**overriding**”:

- Overloading (Nạp chồng): Là khả năng có nhiều hàm hoặc phương thức cùng tên nhưng có các tham số khác nhau hoặc kiểu dữ liệu khác nhau. Ví dụ:

```

1 class Calculation:
2     def add(self, a, b):
3         return a + b
4     def add(self, a, b, c):
5         return a + b + c

```

- Overriding (Ghi đè): Là khả năng của một lớp con thay đổi triển khai của một phương thức mà đã được định nghĩa trong lớp cha. Ví dụ:

```

1 class Animal:
2     def make_sound(self):
3         return "Generic sound"
4 class Dog(Animal):
5     def make_sound(self):
6         return "Woof!"

```

- **Đa hình ở thời điểm chạy (Runtime Polymorphism):**

Được triển khai thông qua kỹ thuật gọi phương thức của đối tượng dựa trên lớp của đối tượng đó. Ví dụ:

```

1 class Animal:
2     def make_sound(self):
3         return "Generic sound"
4 class Dog(Animal):
5     def make_sound(self):
6         return "Woof!"
7 class Cat(Animal):
8     def make_sound(self):
9         return "Meow!"
10 # Đa hình tại thời điểm chạy
11 def animal_sound(animal):
12     return animal.make_sound()
13 dog = Dog()
14 cat = Cat()
15 print(animal_sound(dog)) # Kết quả: Woof!
16 print(animal_sound(cat)) # Kết quả: Meow!

```

Ở ví dụ trên, hàm “**animal_sound()**” có thể nhận vào các đối tượng từ các lớp khác nhau kế thừa từ **Animal**. Khi gọi “**make_sound()**”, nó sẽ thực thi phương thức của đối tượng cụ thể tùy thuộc vào lớp của đối tượng được chuyển vào.

1.4.7 Trừu tượng hóa (Abstraction)

Trùu tượng hóa (Abstraction) là quá trình ẩn các chi tiết cài đặt nội tại của một đối tượng và chỉ hiển thị các tính năng quan trọng hoặc cần thiết cho người sử dụng. Các đặc điểm của trừu tượng hóa:

- **Ứng dụng tập trung:** Trùu tượng hóa cho phép người sử dụng tập trung vào việc sử dụng đối tượng mà không cần biết chi tiết cài đặt bên trong của nó.
- **Giảm sự phức tạp:** Nó giúp giảm bớt sự phức tạp của hệ thống bằng cách che giấu các chi tiết phức tạp và chỉ hiển thị những thông tin cần thiết.
- **Tạo ra mô hình trừu tượng:** Trùu tượng hóa cho phép tạo ra mô hình trừu tượng cho các đối tượng trong thế giới thực, giúp trong việc phát triển và bảo trì mã nguồn dễ dàng hơn.

Ví dụ về trừu tượng hóa trong Python:

```
1 from abc import ABC, abstractmethod
2 # Lớp trừu tượng
3 class Animal(ABC):
4     @abstractmethod
5     def make_sound(self):
6         pass
7 # Lớp con thực hiện (định nghĩa phương thức cụ thể)
8 class Dog(Animal):
9     def make_sound(self):
10        return "Woof!"
11 # Lớp con thực hiện (định nghĩa phương thức cụ thể)
12 class Cat(Animal):
13     def make_sound(self):
14        return "Meow!"
15 # Sử dụng các đối tượng trừu tượng
16 dog = Dog()
17 cat = Cat()
18 print(dog.make_sound()) # Kết quả: Woof!
19 print(cat.make_sound()) # Kết quả: Meow!
```

Trong ví dụ trên, lớp “**Animal**” là lớp trừu tượng và có một phương thức trừu tượng “**make_sound()**”, không có triển khai cụ thể. Hai lớp con “**Dog**” và “**Cat**” kế thừa từ lớp “**Animal**” và triển khai phương thức “**make_sound()**” một cách cụ thể cho từng loài động vật.

1.5 CÔNG CỤ THỰC HÀNH

1.5.1 Trình biên dịch Python

Để bắt đầu lập trình với Python, chúng ta cần cài đặt Python vào máy tính. Sinh viên thực hiện theo các bước sau:

❖ Trên Windows:

- Truy cập vào trang web <https://www.python.org/> và chọn phiên bản Python phù hợp với hệ điều hành Windows (32-bit hoặc 64-bit). Sau đó tải về và cài đặt.
- Chạy trình cài đặt: Đảm bảo đã chọn tùy chọn "Add Python to PATH" để thêm vào biến môi trường PATH.

- Kiểm tra cài đặt: Hãy mở Command Prompt và nhập “**python --version**” hoặc “**python -V**” để kiểm tra phiên bản Python đã cài đặt.
- **Trên macOS:**
 - Cài đặt Python bằng Homebrew: Trong Terminal, nhập lệnh “**brew install python**” và theo hướng dẫn để cài đặt Python.
 - Kiểm tra cài đặt: Mở Terminal và nhập “**python3 --version**” để kiểm tra phiên bản Python đã cài đặt.
- **Trên Linux (phần lớn các bản phân phối):**
 - Mở Terminal và chạy lệnh sau: “**sudo apt-get update && sudo apt-get install python3**”.
 - Kiểm tra cài đặt: Mở Terminal và nhập “**python3 --version**” để kiểm tra.

1.5.2 IDE Visual Studio Code

“Visual Studio Code (VS Code) là trình soạn thảo mã nguồn (IDE) phổ biến được sử dụng rộng rãi cho lập trình” [3]. Một số điểm mạnh của VS Code khi sử dụng cho lập trình Python:

- Hỗ trợ ngôn ngữ Python: VS Code cung cấp hỗ trợ tốt cho Python.
- Extension và tích hợp: Có thể cài đặt các extension để cung cấp các tính năng mở rộng như debugging, linting, phân tích code,....
- Debugging Python: VS Code cung cấp công cụ debugging tích hợp cho Python.
- Terminal tích hợp: Có terminal tích hợp giúp thực thi các lệnh Python và quản lý môi trường làm việc một cách thuận tiện.
- Tích hợp Git: Cung cấp tích hợp với Git để quản lý phiên bản và hợp tác với các dự án Python trong quá trình phát triển.
- Mở rộng và tùy chỉnh: Có thể mở rộng VS Code thông qua các extension.

1.5.3 Hệ thống kiểm soát phiên bản (Git)

“Các dự án trong thực tế thường có nhiều lập trình viên làm việc song song. Vì vậy, cần có một hệ thống kiểm soát phiên bản như Git là cần thiết để đảm bảo không

có xung đột mã giữa các lập trình viên. Ngoài ra, những yêu cầu trong dự án thay đổi thường xuyên. Vì vậy, cần có một hệ thống có thể cho phép nhà phát triển quay lại phiên bản cũ hơn của mã” [4].

Một số khái niệm trong Git:

- Repository (Kho hoặc repo): Là nơi lưu trữ tất cả các file, thư mục và lịch sử thay đổi của dự án.
- Commit: Là hành động lưu trữ các thay đổi trong mã nguồn vào repo. Dùng lệnh “**git commit**” để tạo ra một commit trong Git.
- Branch (Nhánh): Là một phiên bản song song của mã nguồn trong repo, giúp phát triển các tính năng, sửa lỗi mà không làm ảnh hưởng đến nhánh chính.
- Merge và Pull Request (PR): Merge là quá trình hợp nhất các thay đổi từ một nhánh vào nhánh khác. Pull Request (PR) là một yêu cầu được tạo ra để hợp nhất các thay đổi từ một nhánh vào nhánh chính.
- Remote: Là các repo trên máy chủ từ xa như GitHub, GitLab, nơi có thể chia sẻ và đồng bộ hóa code.
- Staging Area (Index): Là nơi lưu trữ các thay đổi đã được chuẩn bị để commit. Trước khi commit, các thay đổi cần được thêm vào staging area.
- Conflict (Xung đột): Xảy ra khi có sự mâu thuẫn giữa các thay đổi trên cùng một dòng trong mã nguồn từ các nhánh khác nhau.

1.6 BÀI TẬP THỰC HÀNH

1.6.1 Bài thực hành 01: Cài đặt môi trường

Sinh viên tiến hành cài đặt môi trường lập trình cho ngôn ngữ Python và tạo các tài khoản cần thiết, bao gồm các phần sau:

1. Cài đặt Python:

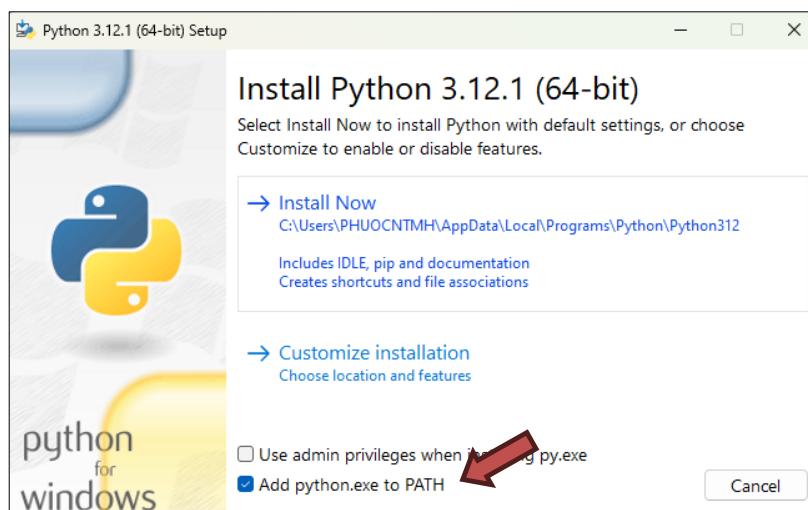
- Truy cập trang web <https://www.python.org/> ➔ Download và chọn phiên bản phù hợp với hệ điều hành.

30

BÀI 1: LẬP TRÌNH CƠ BẢN VỚI PYTHON



- Chạy trình cài đặt: Đảm bảo rằng đã tích chọn tùy chọn "Add Python to PATH" để thêm Python vào trong biến môi trường PATH.



- Kiểm tra cài đặt: Hãy mở Command Prompt và nhập vào "python -version" hoặc "python -V".

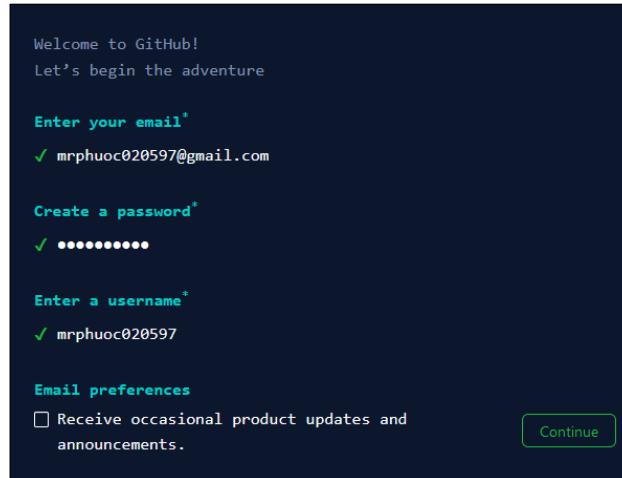
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

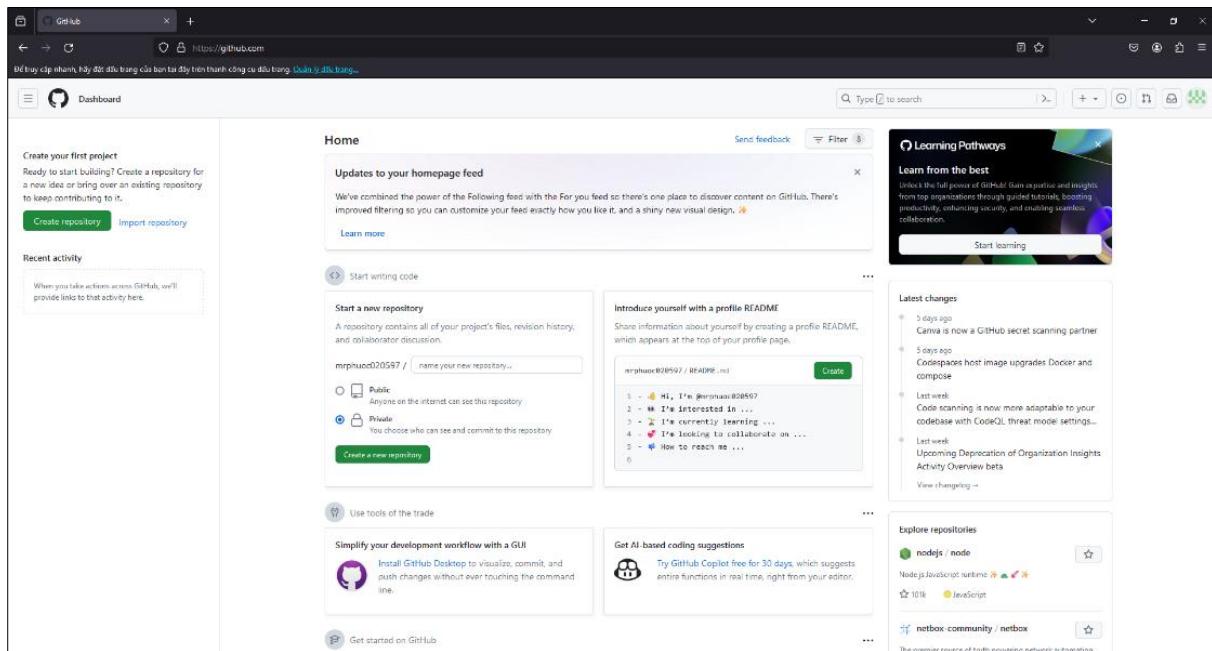
PS C:\Users\PHUOCNTMH> python --version
Python 3.12.3
PS C:\Users\PHUOCNTMH> |
```

2. Tạo tài khoản Github, cài đặt Git, tạo và kéo repo về local:

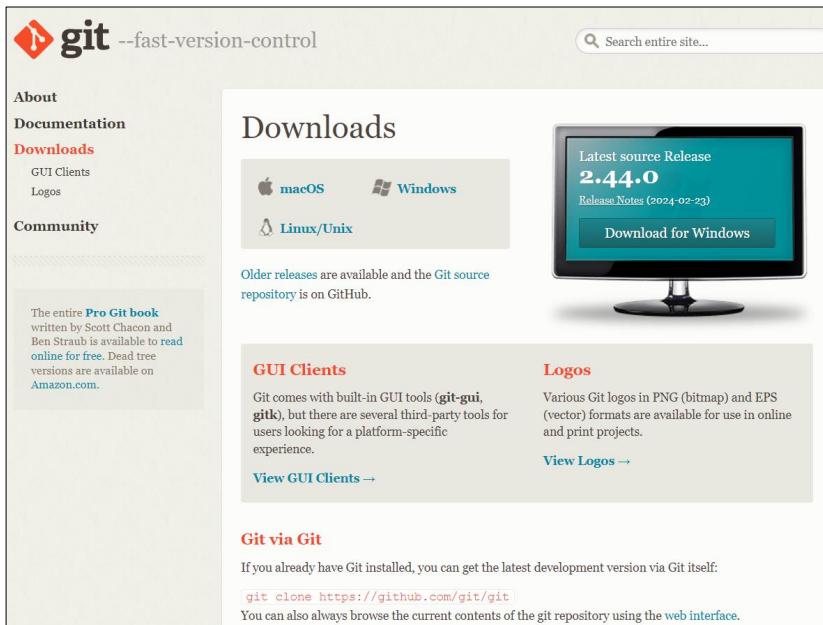
- Truy cập <https://github.com/> → Chọn “Sign up” để đăng ký tài khoản.



- Sau khi xác thực thông tin thành công, tiến hành đăng nhập, chúng ta được giao diện như sau:



- Truy cập <https://git-scm.com/downloads> và chọn phiên bản phù hợp với hệ điều hành.



- Chạy file vừa tải về, tiến hành cài đặt phần mềm vào máy tính.
- Kiểm tra cài đặt: Hãy mở Command Prompt và nhập “git --version”.

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\PHUOCNTMH> git --version
git version 2.44.0.windows.1
PS C:\Users\PHUOCNTMH> |
```

- Vào địa chỉ <https://github.com/> ➔ Chọn “+” ➔ Chọn “New Repository”.
 - Repository name: Nhập “bmttnc-hutech-<MSV>”.
 - Description: Nhập “< họ_tên_SV>_<MSV>”.
 - Chọn “Private”.
 - Chọn “Add a README file”.
 - Chọn “Create repository” để tạo mới repo.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (*).

Owner * Repository name *

 mrphuoc020597 / bmtt-nc-hutech-151106024
bmtt-nc-hutech-1511060249 is available.

Great repository names are short and memorable. Need inspiration? How about `expert-meme` ?

Description (optional)
NguyenTrongMinhHongPhuoc_1511060249

Public Anyone on the internet can see this repository. You choose who can commit.
 Private You choose who can see and commit to this repository.

Initialize this repository with
 Add a README file This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore
.gitignore template: None

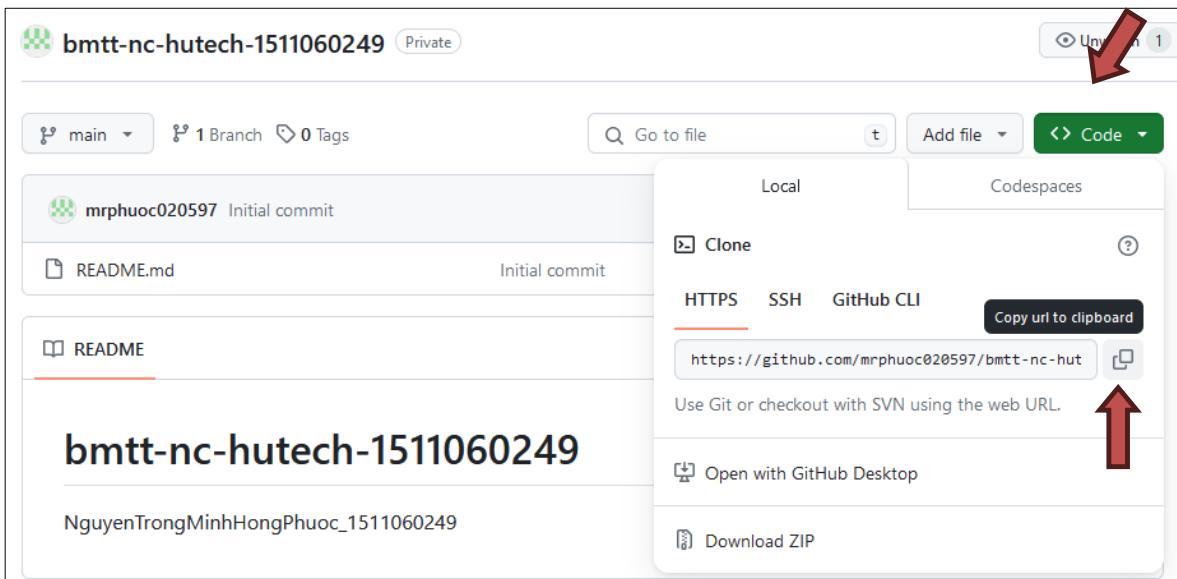
Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license
License: None

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

This will set `main` as the default branch. Change the default name in your [settings](#).

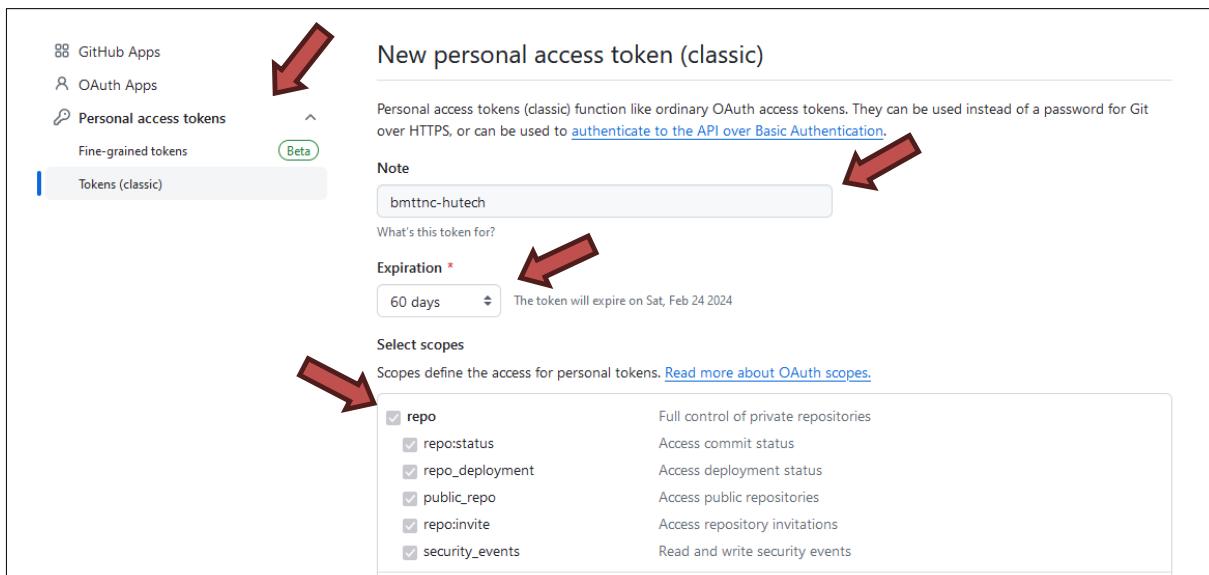
- Chọn "<> Code" → HTTPS → Nhấn copy đường dẫn tới repo.



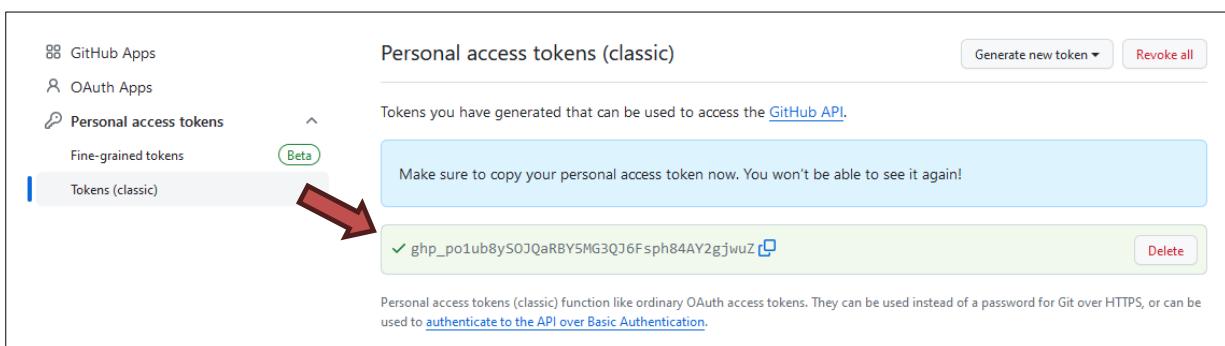
34

BÀI 1: LẬP TRÌNH CƠ BẢN VỚI PYTHON

- Vào <https://github.com/settings/profile> → Chọn “Developer Settings” → Chọn “Personal access tokens” → “Tokens (classic)” → Chọn “Generate a personal access token”.
- Nhập Note là “bmtt-hutech”, ở Expiration chọn “60 days”. Chọn các mục trong “Select scopes” → Chọn “Generate token” để tạo mới token.



- Lưu trữ “Personal access tokens (classic)” tránh thất lạc cho các lần dùng tiếp theo.



- Tại máy tính thực hành, tiến hành clone repo về local. Mở Command Prompt (lưu ý chuyển ở đĩa hiện thời sang ổ đĩa thực hành):

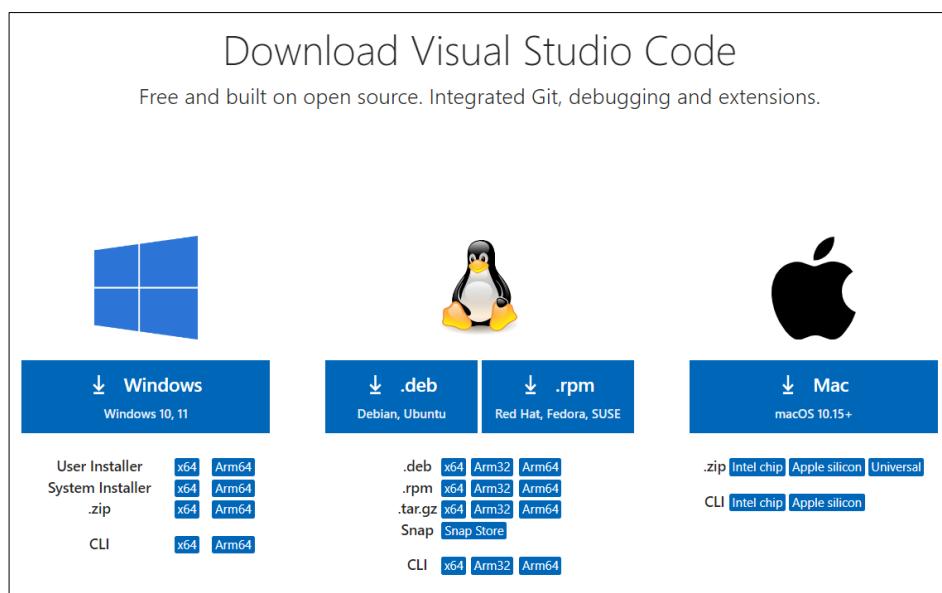
```
C:\Users\PHUOCNTMH>D:  
D:\>git clone <đường dẫn tới repo github của sinh viên>  
Cloning into 'bmtt-nc-hutech-1511060249'...
```

- Cửa sổ “Connect to Github” hiện ra → Chọn “Token” → Nhập “Personal access token” vừa được tạo ở bước trên vào. Repo được clone về thành công.

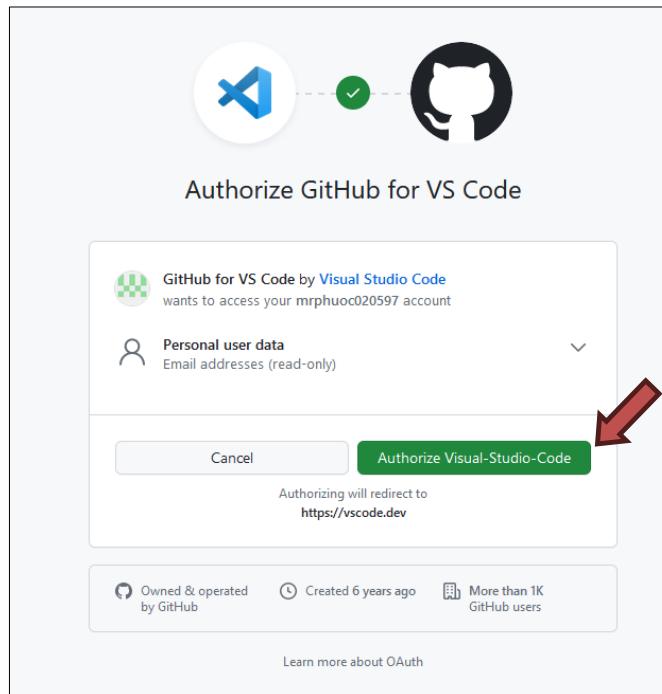
```
D:\>git clone <đường dẫn tới repo github của sinh viên>
Cloning into 'bmtt-nc-hutech-1511060249'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
```

3. Cài đặt Visual Studio Code và cấu hình:

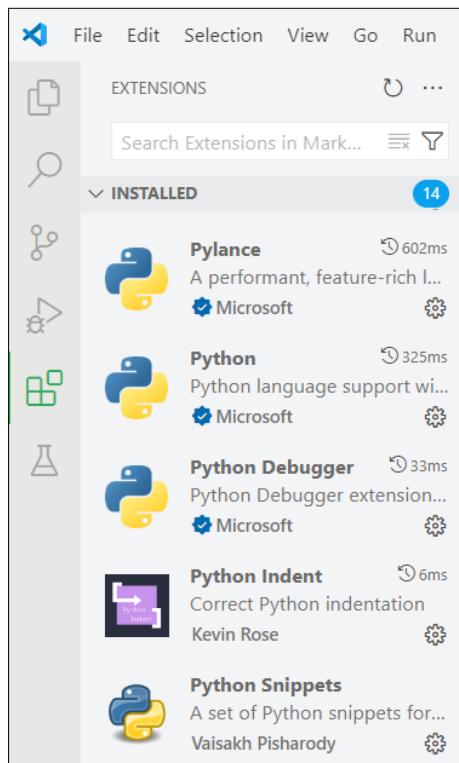
- Vào trang <https://code.visualstudio.com/download> và tải phiên bản phù hợp với hệ điều hành.



- Sau khi tải file cài đặt về, tiến hành cài đặt phần này mềm vào máy tính.
- Mở VS Code, chọn “Accounts” → Chọn “Sign in to Sync Settings”. Chọn “Authorize Visual-Studio-Code”. Hệ thống sẽ đưa trở về giao diện VS Code. Kiểm tra tại mục “Accounts” xem đã có tài khoản được kết nối hay chưa.

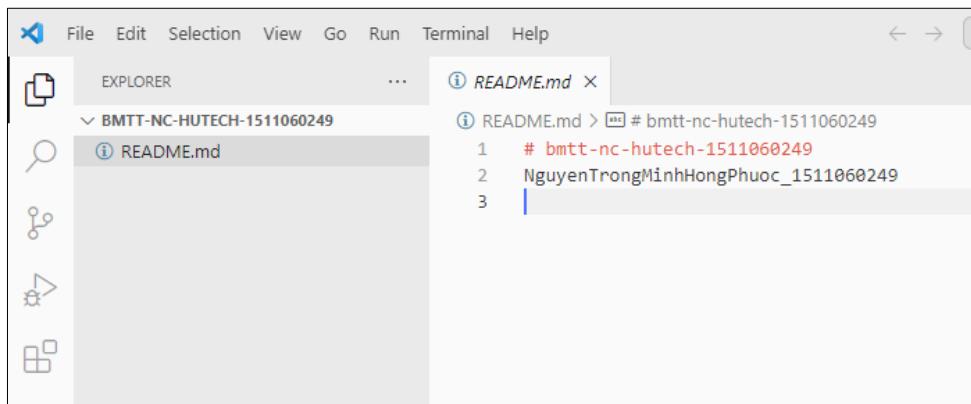


- Vào thẻ “Extensions”, tiến hành cài đặt các extension sau: Python, Pylance, Python Indent, Python Snippets.

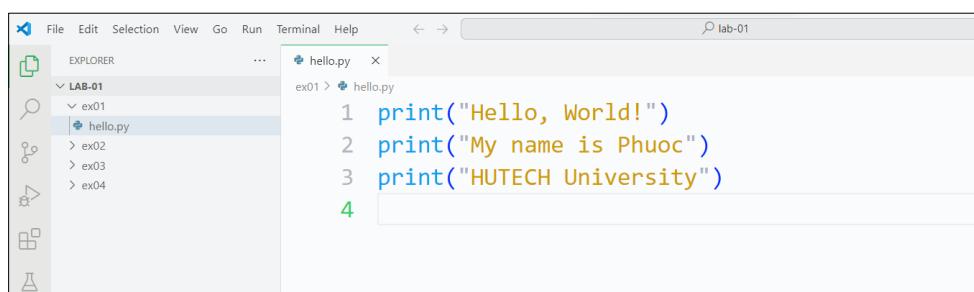


4. Ví dụ cơ bản với Python:

- Mở VS Code → Chọn “File” → “Open Folder...”. Chọn đường dẫn đến project được clone về từ Github ở bước trên → Chọn “Yes, I trust the authors”.



- Tạo mới thư mục “lab-01”. Trong thư mục “lab-01” tạo file “hello.py”.



- Lưu file. Vào menu “Terminal” → “New Terminal” hoặc nhấn tổ hợp phím Ctrl + Shift + ` . Kiểm tra chương trình vừa tạo.

```

PS D:\bmtt-nc-hutech-1511060249> cd .\lab-01\
PS D:\bmtt-nc-hutech-1511060249\lab-01> python .\hello.py
Hello, World!
My name is Phuoc
HUTECH University
PS D:\bmtt-nc-hutech-1511060249\lab-01>

```

38

BÀI 1: LẬP TRÌNH CƠ BẢN VỚI PYTHON

- Cài đặt thông tin “Git account's default identity”:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS D:\bmmt-nc-hutech-1511060249\lab-01> git config --global user.email "mrphuoc020597@gmail.com"
PS D:\bmmt-nc-hutech-1511060249\lab-01> git config --global user.name "mrphuoc020597"
PS D:\bmmt-nc-hutech-1511060249\lab-01> git config --global user.email
mrphuoc020597@gmail.com
PS D:\bmmt-nc-hutech-1511060249\lab-01> git config --global user.name
mrphuoc020597
```

- Đưa các thay đổi lên remote repo:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS D:\bmmt-nc-hutech-1511060249\lab-01> git add .
PS D:\bmmt-nc-hutech-1511060249\lab-01> git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   hello.py

PS D:\bmmt-nc-hutech-1511060249\lab-01> git commit -m "lab01 hello world"
[main b9d827e] lab01 hello world
 1 file changed, 3 insertions(+)
  create mode 100644 lab-01/hello.py
PS D:\bmmt-nc-hutech-1511060249\lab-01> git push origin
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 385 bytes | 385.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/mrphuoc020597/bmmt-nc-hutech-1511060249.git
  8a5d785..b9d827e main -> main
PS D:\bmmt-nc-hutech-1511060249\lab-01>
```

- Kiểm tra trên website <https://github.com>:

The screenshot shows a GitHub repository page for 'bmmt-nc-hutech-1511060249'. The repository has 1 branch (main) and 0 tags. The 'lab-01' branch contains a single commit by user 'mrphuoc020597' titled 'lab01 hello world', pushed 1 minute ago. A README.md file is also present, created 51 minutes ago.

1.6.2 Bài thực hành 02: Lập trình Python cơ bản

- **Câu 01:** Viết chương trình yêu cầu người dùng nhập vào họ tên và tuổi của họ, sau đó in ra màn hình thông điệp chào mừng với thông tin vừa nhập.

Hướng dẫn: Tạo file “ex02_01.py” trong “lab_01”.

```
1 # Nhập tên và tuổi từ người dùng
2 ten = input("Nhập tên của bạn: ")
3 tuoi = input("Nhập tuổi của bạn: ")
4 # In thông điệp chào mừng với thông tin vừa nhập
5 print("Chào mừng,", ten, "! Bạn", tuoi, "tuổi.")
```

Kết quả:

```
ex02_01.py x
ex02 > ex02_01.py > ...
1 # Nhập tên và tuổi từ người dùng
2 ten = input("Nhập tên của bạn: ")
3 tuoi = input("Nhập tuổi của bạn: ")
4 # In thông điệp chào mừng với thông tin vừa nhập
5 print("Chào mừng,", ten, "! Bạn", tuoi, "tuổi.")
6

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\lab-01> cd ex02
PS D:\lab-01\ex02> python ex02_01.py
Nhập tên của bạn: Phuoc Nguyen
Nhập tuổi của bạn: 26
Chào mừng, Phuoc Nguyen ! Bạn 26 tuổi.
PS D:\lab-01\ex02>
```

- **Câu 02:** Viết chương trình thực hiện tính diện tích của hình tròn với bán kính được nhập từ người dùng. Sử dụng giá trị Pi là 3.14.

Hướng dẫn: Tạo file “ex02_02.py” trong “lab_01”.

```
1 # Nhập bán kính từ người dùng
2 ban_kinh = float(input("Nhập bán kính của hình tròn: "))
3 # Tính diện tích của hình tròn
4 dien_tich = 3.14 * (ban_kinh ** 2)
5 # In diện tích của hình tròn ra màn hình
6 print("Diện tích của hình tròn là:", dien_tich)
```

- Kết quả:

```

1 # Nhập bán kính từ người dùng
2 ban_kinh = float(input("Nhập bán kính của hình tròn: "))
3 # Tính diện tích của hình tròn
4 dien_tich = 3.14 * (ban_kinh ** 2)

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS D:\lab-01> cd ex02
PS D:\lab-01\ex02> python ex02_02.py
Nhập bán kính của hình tròn: 5.7
Diện tích của hình tròn là: 102.0186
PS D:\lab-01\ex02>

- Câu 03:** Viết chương trình kiểm tra một số nhập vào từ người dùng có phải là số chẵn hay không.

Hướng dẫn: Tạo file “ex02_03.py” trong “lab_01”.

```

1 # Nhập số từ người dùng
2 so = int(input("Nhập một số nguyên: "))
3 # Kiểm tra xem số đó có phải số chẵn hay không
4 if so % 2 == 0:
5     print(so, "là số chẵn.")
6 else:
7     print(so, "không phải là số chẵn.")

```

Kết quả:

```

1 # Nhập số từ người dùng
2 so = int(input("Nhập một số nguyên: "))
3 # Kiểm tra xem số đó có phải số chẵn hay không
4 if so % 2 == 0:

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS D:\lab-01> cd ex02
PS D:\lab-01\ex02> python ex02_03.py
Nhập một số nguyên: 10
10 là số chẵn.
PS D:\lab-01\ex02> python ex02_03.py
Nhập một số nguyên: 7
7 không phải là số chẵn.
PS D:\lab-01\ex02>

- Câu 04:** Xây dựng một chương trình nhằm tìm tất cả các số nằm trong khoảng từ 2000 đến 3200 (bao gồm cả 2000 và 3200) mà chia hết cho 7 và không phải là bội số của 5. Các số tìm được sẽ được in ra trên một dòng, ngăn cách bởi dấu phẩy.

Hướng dẫn: Tạo file “ex02_04.py” trong “lab_01”.

```

1 # Tạo một danh sách rỗng để lưu kết quả
2 j=[]
3 # Duyệt qua tất cả các số trong đoạn từ 2000 đến 3200, kiểm tra xem
   số i có chia hết cho 7 và không phải là bội số của 5 không
4 for i in range(2000, 3201):
5     if (i % 7 == 0) and (i % 5 != 0):
6         j.append(str(i))
7 print (','.join(j))

```

Kết quả:

```

ex02_04.py x
ex02 > ex02_04.py > ...
1 # Tạo một danh sách rỗng để lưu kết quả
2 j=[]
3 # Duyệt qua tất cả các số trong đoạn từ 2000 đến 3200, kiểm tra xem
   số i có chia hết cho 7 và không phải là bội số của 5 không
4 for i in range(2000, 3201):
5     if (i % 7 == 0) and (i % 5 != 0):
6         j.append(str(i))

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS + ... ^ x
PS D:\lab-01> cd ex02
PS D:\lab-01\ex02> python ex02_04.py
2002,2009,2016,2023,2037,2044,2051,2058,2072,2079,2086,2093,2107,2114,2121,2128,2142,2149,2156,2163,2177,2184,2191,2198,2212,2219,2226,2233,2247,225
4,2261,2268,2282,2289,2296,2303,2317,2324,2331,2338,2352,2359,2366,2373,2387,2394,2401,2408,2422,2429,2436,2443,2457,2464,2471,2478,2492,2499,2506,2
513,2527,2534,2541,2548,2562,2569,2576,2583,2597,2604,2611,2618,2632,2639,2646,2653,2667,2674,2681,2688,2702,2709,2716,2723,2737,2744,2751,2758,2772
,2779,2786,2793,2807,2814,2821,2828,2842,2849,2856,2863,2877,2884,2891,2898,2912,2919,2926,2933,2947,2954,2961,2968,2982,2989,2996,3003,3017,3024,30
31,3038,3052,3059,3066,3073,3087,3094,3101,3108,3122,3129,3136,3143,3157,3164,3171,3178,3192,3199
PS D:\lab-01\ex02>

```

- Câu 05:** Xây dựng một chương trình nhằm nhập số giờ làm việc hàng tuần của nhân viên và mức lương theo giờ tiêu chuẩn. Từ đó, thực hiện tính toán số tiền thực nhận của nhân viên. Cần lưu ý rằng số giờ tiêu chuẩn mỗi tuần là 44 giờ, và mỗi giờ làm thêm sẽ được trả 150% so với mức lương theo giờ tiêu chuẩn.

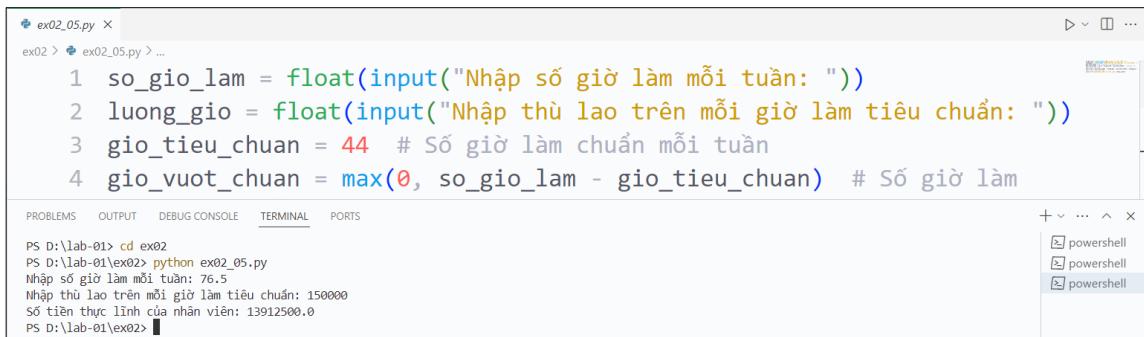
Hướng dẫn: Tạo file “ex02_05.py” trong “lab_01”.

```

1 so_gio_lam = float(input("Nhập số giờ làm mỗi tuần: "))
2 luong_gio = float(input("Nhập thù lao trên mỗi giờ làm tiêu chuẩn: "))
3 gio_tieu_chuan = 44 # Số giờ làm chuẩn mỗi tuần
4 gio_vuot_chuan = max(0, so_gio_lam - gio_tieu_chuan) # Số giờ làm
   vượt chuẩn mỗi tuần
5 thuc_linh = gio_tieu_chuan * luong_gio + gio_vuot_chuan * luong_gio *
   1.5 # Tính tổng thu nhập
6 print(f"Số tiền thực lĩnh của nhân viên: {thuc_linh}")

```

Kết quả:



```

ex02_05.py ×
ex02 > ex02_05.py > ...
1 so_gio_lam = float(input("Nhập số giờ làm mỗi tuần: "))
2 luong_gio = float(input("Nhập thù lao trên mỗi giờ làm tiêu chuẩn: "))
3 gio_tieu_chuan = 44 # Số giờ làm chuẩn mỗi tuần
4 gio_vuot_chuan = max(0, so_gio_lam - gio_tieu_chuan) # Số giờ làm
5 luong_vuot_chuan = luong_gio * gio_vuot_chuan
6 luong_thuc_linh = so_gio_lam * luong_gio + luong_vuot_chuan
7 print(luong_thuc_linh)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\lab-01> cd ex02
PS D:\lab-01\ex02> python ex02_05.py
Nhập số giờ làm mỗi tuần: 76.5
Nhập thù lao trên mỗi giờ làm tiêu chuẩn: 150000
Số tiền thực lĩnh của nhân viên: 13912500.0
PS D:\lab-01\ex02>

```

- **Câu 06:** Tạo một chương trình để nhập vào hai số X và Y; từ đó, xây dựng một mảng hai chiều. Giá trị của mỗi phần tử tại hàng i và cột j của mảng sẽ là $i*j$, với i chạy từ 0 đến X-1 và j từ 0 đến Y-1. Chẳng hạn, nếu X và Y được nhập là 3 và 5, thì kết quả sẽ là: $[[0, 0, 0, 0, 0], [0, 1, 2, 3, 4], [0, 2, 4, 6, 8]]$.

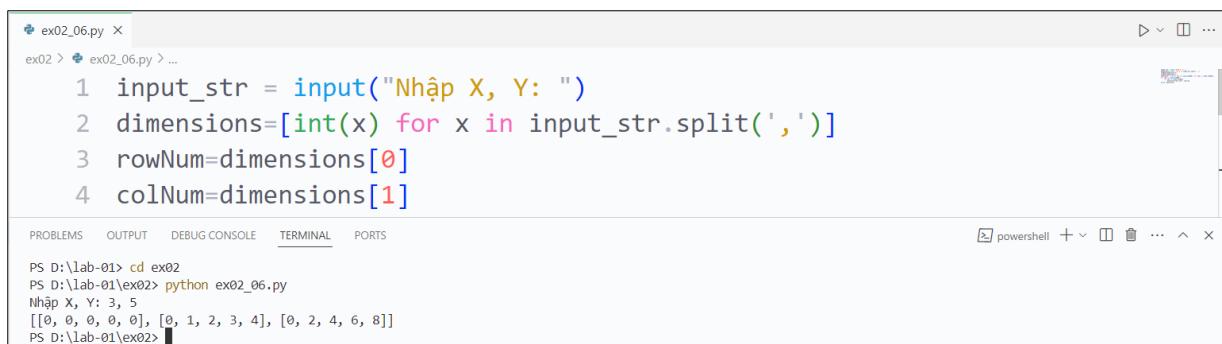
Hướng dẫn: Tạo file “ex02_06.py” trong “lab_01”.

```

1 input_str = input("Nhập X, Y: ")
2 dimensions=[int(x) for x in input_str.split(',')]
3 rowNum=dimensions[0]
4 colNum=dimensions[1]
5 multilist = [[0 for col in range(colNum)] for row in range(rowNum)]
6 for row in range(rowNum):
7     for col in range(colNum):
8         multilist[row][col]= row*col
9 print (multilist)

```

Kết quả:



```

ex02_06.py ×
ex02 > ex02_06.py > ...
1 input_str = input("Nhập X, Y: ")
2 dimensions=[int(x) for x in input_str.split(',')]
3 rowNum=dimensions[0]
4 colNum=dimensions[1]
5 multilist = [[0 for col in range(colNum)] for row in range(rowNum)]
6 for row in range(rowNum):
7     for col in range(colNum):
8         multilist[row][col]= row*col
9 print (multilist)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\lab-01> cd ex02
PS D:\lab-01\ex02> python ex02_06.py
Nhập X, Y: 3, 5
[[0, 0, 0, 0, 0], [0, 1, 2, 3, 4], [0, 2, 4, 6, 8]]
PS D:\lab-01\ex02>

```

- **Câu 07:** Xây dựng một chương trình để nhận các chuỗi đầu vào là những dòng được nhập, sau đó chuyển đổi các dòng này thành chữ hoa và hiển thị kết quả lên màn hình.

Hướng dẫn: Tạo file “ex02_07.py” trong “lab_01”.

```

1 # Nhập các dòng từ người dùng
2 print("Nhập các dòng văn bản (Nhập 'done' để kết thúc):")
3 lines = []
4 while True:
5     line = input()
6     if line.lower() == 'done':
7         break
8     lines.append(line)
9 # Chuyển các dòng thành chữ in hoa và in ra màn hình
10 print("\nCác dòng đã nhập sau khi chuyển thành chữ in hoa:")
11 for line in lines:
12     print(line.upper())

```

Kết quả:

```

ex02_07.py x
ex02 > ex02_07.py > ...
1 # Nhập các dòng từ người dùng
2 print("Nhập các dòng văn bản (Nhập 'done' để kết thúc):")
3 lines = []
4 while True:
5     line = input()
6     if line.lower() == 'done':
7         break
8     lines.append(line)
9 # Chuyển các dòng thành chữ in hoa và in ra màn hình
10 print("\nCác dòng đã nhập sau khi chuyển thành chữ in hoa:")
11 for line in lines:
12     print(line.upper())
13

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Nhập các dòng văn bản (Nhập 'done' để kết thúc):
hutech university
Thuc hanh Bao mat thong tin nang cao
Lap trinh bang ngon ngu Python
done

Các dòng đã nhập sau khi chuyển thành chữ in hoa:
HUTECH UNIVERSITY
THUC HANH BAO MAT THONG TIN NANG CAO
LAP TRINH BANG NGON NGU PYTHON
PS D:\lab-01\ex02>

```

- Câu 08:** Hãy phát triển một chương trình để nhập vào một chuỗi các số nhị phân với 4 chữ số, phân cách bằng dấu phẩy. Chương trình sẽ kiểm tra từng số để xác định xem chúng có chia hết cho 5 hay không, sau đó in ra những số thỏa mãn điều kiện này, cũng được phân tách bằng dấu phẩy. Ví dụ, nếu đầu vào là: '0100', '0011', '1010', '1001', thì kết quả đầu ra sẽ là: '1010'.

Hướng dẫn: Tạo file "ex02_08.py" trong "lab_01".

```

1 # Hàm kiểm tra số nhị phân có chia hết cho 5 không
2 def chia_het.Cho_5(so_nhi_phan):
3     # Chuyển số nhị phân sang số thập phân
4     so_thap_phan = int(so_nhi_phan, 2)
5     # Kiểm tra xem số thập phân có chia hết cho 5 không
6     if so_thap_phan % 5 == 0:
7         return True
8     else:
9         return False
10 # Nhập chuỗi số nhị phân từ người dùng
11 chuoisо_nhi_phan = input("Nhập chuỗi số nhị phân (phân tách bởi dấu
12 phẩy): ")
13
14 # Tách chuỗi thành các số nhị phân và kiểm tra số chia hết cho 5
15 so_nhi_phan_list = chuoisо_nhi_phan.split(',')
16 so_chia_het.Cho_5 = [so for so in so_nhi_phan_list if chia_het.Cho_5
17 (so)]
18
19 # In ra các số nhị phân chia hết cho 5
20 if len(so_chia_het.Cho_5) > 0:
21     ket_qua = ', '.join(so_chia_het.Cho_5)
22     print("Các số nhị phân chia hết cho 5 là:", ket_qua)
23 else:
24     print("Không có số nhị phân nào chia hết cho 5 trong chuỗi đã
25 nhập.")

```

Kết quả:

```

ex02 > ex02_08.py > ...
1 # Hàm kiểm tra số nhị phân có chia hết cho 5 không
2 def chia_het.Cho_5(so_nhi_phan):
3     # Chuyển số nhị phân sang số thập phân
4     so_thap_phan = int(so_nhi_phan, 2)
5
6     if so_thap_phan % 5 == 0:
7         return True
8     else:
9         return False
10
11 so_nhi_phan_list = chuoisо_nhi_phan.split(',')
12 so_chia_het.Cho_5 = [so for so in so_nhi_phan_list if chia_het.Cho_5
13 (so)]
14
15 # In ra các số nhị phân chia hết cho 5
16 if len(so_chia_het.Cho_5) > 0:
17     ket_qua = ', '.join(so_chia_het.Cho_5)
18     print("Các số nhị phân chia hết cho 5 là:", ket_qua)
19 else:
20     print("Không có số nhị phân nào chia hết cho 5 trong chuỗi đã
21 nhập.")

PS D:\lab-01\ex02> cd ex02
PS D:\lab-01\ex02> python ex02_08.py
Nhập chuỗi số nhị phân (phân tách bởi dấu phẩy): 0100, 0011, 1010, 1001
Các số nhị phân chia hết cho 5 là: 1010
PS D:\lab-01\ex02>

```

- **Câu 09:** Viết hàm kiểm tra xem một số được nhập vào có phải là số nguyên tố hay không.

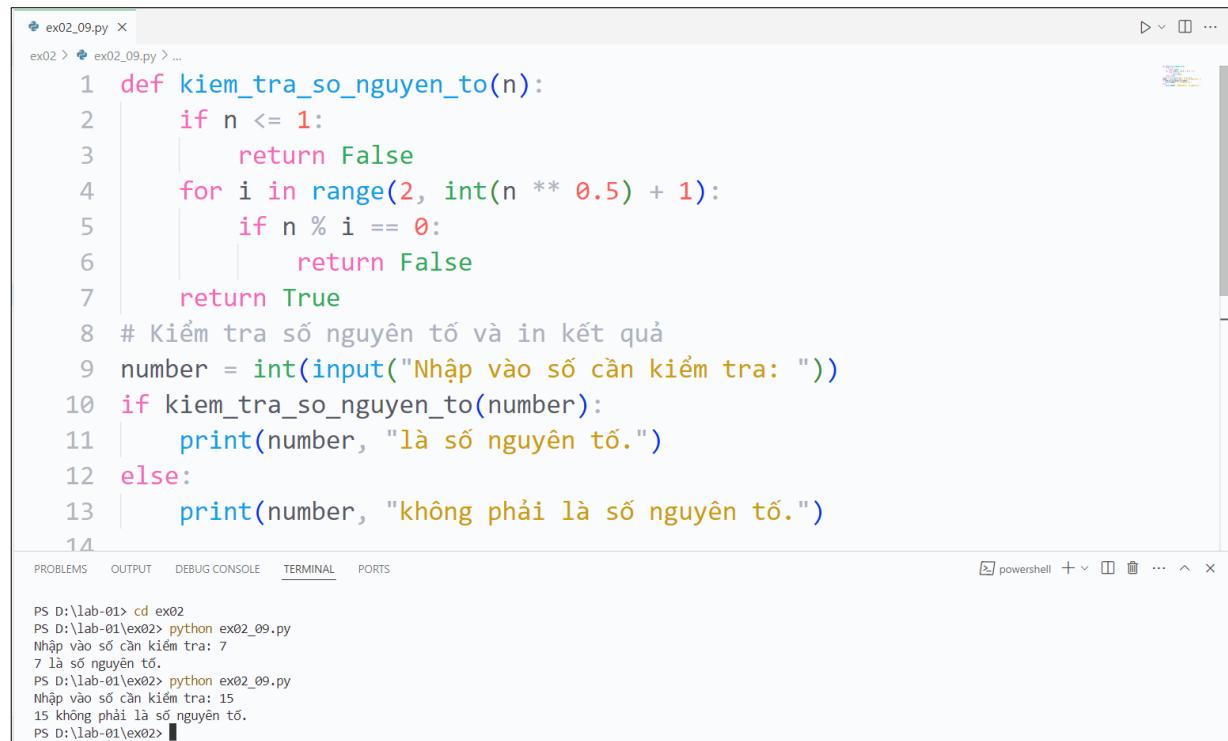
Hướng dẫn: Tạo file “ex02_09.py” trong “lab_01”.

```

1 def kiem_tra_so_nguyen_to(n):
2     if n <= 1:
3         return False
4     for i in range(2, int(n ** 0.5) + 1):
5         if n % i == 0:
6             return False
7     return True
8 # Kiểm tra số nguyên tố và in kết quả
9 number = int(input("Nhập vào số cần kiểm tra: "))
10 if kiem_tra_so_nguyen_to(number):
11     print(number, "là số nguyên tố.")
12 else:
13     print(number, "không phải là số nguyên tố.")

```

Kết quả:



```

ex02 > ex02_09.py > ...
1 def kiem_tra_so_nguyen_to(n):
2     if n <= 1:
3         return False
4     for i in range(2, int(n ** 0.5) + 1):
5         if n % i == 0:
6             return False
7     return True
8 # Kiểm tra số nguyên tố và in kết quả
9 number = int(input("Nhập vào số cần kiểm tra: "))
10 if kiem_tra_so_nguyen_to(number):
11     print(number, "là số nguyên tố.")
12 else:
13     print(number, "không phải là số nguyên tố.")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\lab-01> cd ex02
PS D:\lab-01\ex02> python ex02_09.py
Nhập vào số cần kiểm tra: 7
7 là số nguyên tố.
PS D:\lab-01\ex02> python ex02_09.py
Nhập vào số cần kiểm tra: 15
15 không phải là số nguyên tố.
PS D:\lab-01\ex02>

```

46

BÀI 1: LẬP TRÌNH CƠ BẢN VỚI PYTHON

- **Câu 10:** Viết một hàm nhận vào một chuỗi và trả về chuỗi đảo ngược của nó.

Hướng dẫn: Tạo file “ex02_10.py” trong “lab_01”.

```
1 def dao_nguoc_chuoi(chuoi):
2     return chuoi[::-1]
3 # Sử dụng hàm và in kết quả
4 input_string = input("Mời nhập chuỗi cần đảo ngược: ")
5 print("Chuỗi đảo ngược là:", dao_nguoc_chuoi(input_string))
```

Kết quả:

The screenshot shows the VS Code interface. The code editor contains the Python script `ex02_10.py`. The terminal below shows the command `python ex02_10.py` being run, followed by the user input "Mời nhập chuỗi cần đảo ngược: hutech university" and the program's output "Chuỗi đảo ngược là: ytisrevinu hcetuh".

```
ex02 > ex02_10.py > ...
1 def dao_nguoc_chuoi(chuoi):
2     return chuoi[::-1]
3 # Sử dụng hàm và in kết quả
4 input_string = input("Mời nhập chuỗi cần đảo ngược: ")
5 print("Chuỗi đảo ngược là:", dao_nguoc_chuoi(input_string))

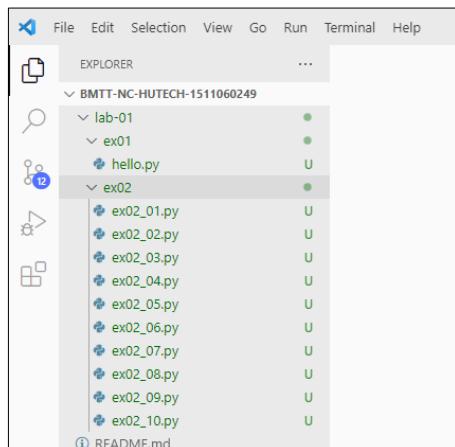
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS D:\lab-01> cd ex02
PS D:\lab-01\ex02> python ex02_10.py
Mời nhập chuỗi cần đảo ngược: hutech university
Chuỗi đảo ngược là: ytisrevinu hcetuh
PS D:\lab-01\ex02>
```

Cập nhật thư mục theo các bài tập (nhập lệnh vào cửa sổ Terminal của VS Code):

```
mkdir ex01
move .\hello.py .\ex01\
mkdir ex02
Move-Item .\ex02_*.py .\ex02\
```

Kết quả:



Đưa các thay đổi lên Git (tách nhánh lab01 từ nhánh master, đưa các thay đổi hiện tại vào stash):

```
git add .
git stash
git checkout -b lab01
git stash apply
git add .
git commit -m "[add] lab-01 ex01 ex02"
git push origin lab01
```

```
File Edit Selection View Go Run Terminal Help
EXPLORER PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\bmmt-nc-hutech-1511060249> git add .
PS D:\bmmt-nc-hutech-1511060249> git stash
Saved working directory and index state WIP on main: b9d827e lab01 hello world
PS D:\bmmt-nc-hutech-1511060249> git checkout -b lab-01
Switched to a new branch 'lab-01'
PS D:\bmmt-nc-hutech-1511060249> git stash apply
On branch lab-01
Changes to be committed:
(use "git restore --staged <file>..." to unstage)
  new file: lab-01/ex01/hello.py
  new file: lab-01/ex02/ex02_01.py
  new file: lab-01/ex02/ex02_02.py
  new file: lab-01/ex02/ex02_03.py
  new file: lab-01/ex02/ex02_04.py
  new file: lab-01/ex02/ex02_05.py
  new file: lab-01/ex02/ex02_06.py
  new file: lab-01/ex02/ex02_07.py
  new file: lab-01/ex02/ex02_08.py
  new file: lab-01/ex02/ex02_09.py
  new file: lab-01/ex02/ex02_10.py

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    deleted: lab-01/hello.py

PS D:\bmmt-nc-hutech-1511060249> git add .
PS D:\bmmt-nc-hutech-1511060249> git commit -m "[add] lab-01 ex01 ex02"
[lab-01 8a81644] [add] lab-01 ex01 ex02
11 files changed, 90 insertions(+)
rename lab-01/ => ex01/hello.py (100%)
create mode 100644 lab-01/ex02/ex02_01.py
create mode 100644 lab-01/ex02/ex02_02.py
create mode 100644 lab-01/ex02/ex02_03.py
create mode 100644 lab-01/ex02/ex02_04.py
create mode 100644 lab-01/ex02/ex02_05.py
create mode 100644 lab-01/ex02/ex02_06.py
create mode 100644 lab-01/ex02/ex02_07.py
create mode 100644 lab-01/ex02/ex02_08.py
create mode 100644 lab-01/ex02/ex02_09.py
create mode 100644 lab-01/ex02/ex02_10.py
PS D:\bmmt-nc-hutech-1511060249> git push origin lab-01
Enumerating objects: 16, done.
Counting objects: 100% (16/16), done.
Delta compression using up to 8 threads
Compressing objects: 100% (14/14), done.
Writing objects: 100% (14/14), 2.96 KiB | 2.96 MiB/s, done.
Total 14 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'lab-01' on GitHub by visiting:
remote:   https://github.com/mrphuoc020597/bmmt-nc-hutech-1511060249/pull/new/lab-01
remote:
To https://github.com/mrphuoc020597/bmmt-nc-hutech-1511060249.git
 * [new branch]      lab-01 -> lab-01
PS D:\bmmt-nc-hutech-1511060249>
```

1.6.3 Bài thực hành 03: List, Tuple, Dictionary

- Câu 01:** Viết một chương trình để tính tổng của tất cả các số chẵn trong một List.

Hướng dẫn: Tại folder “lab-01”, tạo folder “ex03”. Tạo file “ex03_01.py” trong “ex03”.

```

1 def tinh_tong_so_chan(lst):
2     tong = 0
3     for num in lst:
4         if num % 2 == 0:
5             tong += num
6     return tong
7
8 # Nhập danh sách từ người dùng và xử lý chuỗi
9 input_list = input("Nhập danh sách các số, cách nhau bằng dấu phẩy: ")
10 numbers = list(map(int, input_list.split(',')))
11
12 # Sử dụng hàm và in kết quả
13 tong_chan = tinh_tong_so_chan(numbers)
14 print("Tổng các số chẵn trong List là:", tong_chan)

```

Kết quả:

```

ex03 > ex03_01.py > ...
1 def tinh_tong_so_chan(lst):
2     tong = 0
3     for num in lst:
4         if num % 2 == 0:
5             tong += num
6     return tong
7
8 # Nhập danh sách từ người dùng và xử lý chuỗi
9 input_list = input("Nhập danh sách các số, cách nhau bằng dấu phẩy: ")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS D:\lab-01> cd ex03
PS D:\lab-01\ex03> python ex03_01.py
Nhập danh sách các số, cách nhau bằng dấu phẩy: 1,-2,3,4,5,-6,7,8,-9
Tổng các số chẵn trong List là: 4
PS D:\lab-01\ex03>

- Câu 02:** Viết chương trình để đảo ngược vị trí của các phần tử trong một danh sách.

Hướng dẫn: Tạo file “ex03_02.py” trong “ex03”.

```
1 def dao_nguoc_list(lst):
2     return lst[::-1]
3
4 # Nhập danh sách từ người dùng và xử lý chuỗi
5 input_list = input("Nhập danh sách các số, cách nhau bằng dấu phẩy: ")
6 numbers = list(map(int, input_list.split(',')))
7
8 # Sử dụng hàm và in kết quả
9 list_dao_nguoc = dao_nguoc_list(numbers)
10 print("List sau khi đảo ngược:", list_dao_nguoc)
```

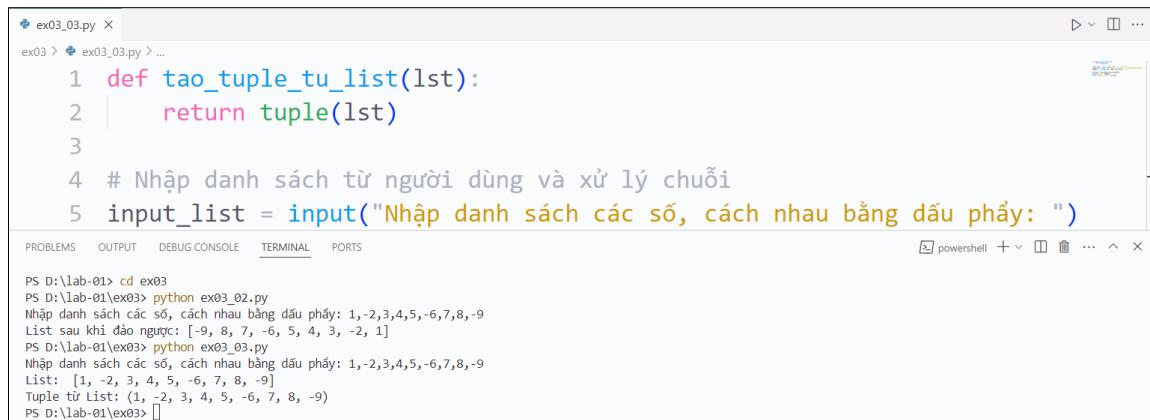
Kết quả:

- **Câu 03:** Viết chương trình để tạo một Tuple từ một List nhập vào từ bàn phím.

Hướng dẫn: Tao file “ex03_03.py” trong “ex03”.

```
1 def tao_tuple_tu_list(lst):
2     return tuple(lst)
3
4 # Nhập danh sách từ người dùng và xử lý chuỗi
5 input_list = input("Nhập danh sách các số, cách nhau bằng dấu phẩy: ")
6 numbers = list(map(int, input_list.split(',')))
7
8 my_tuple = tao_tuple_tu_list(numbers)
9 print("List: ", numbers)
10 print("Tuple từ List:", my_tuple)
```

Kết quả:



```

ex03_03.py > ...
1 def tao_tuple_tu_list(lst):
2     return tuple(lst)
3
4 # Nhập danh sách từ người dùng và xử lý chuỗi
5 input_list = input("Nhập danh sách các số, cách nhau bằng dấu phẩy: ")
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell + v ⌂ ... ^ x
PS D:\lab-01> cd ex03
PS D:\lab-01\ex03> python ex03_02.py
Nhập danh sách các số, cách nhau bằng dấu phẩy: 1,-2,3,4,5,-6,7,8,-9
List sau khi đảo ngược: [-9, 8, 7, -6, 5, 4, 3, -2, 1]
PS D:\lab-01\ex03> python ex03_03.py
Nhập danh sách các số, cách nhau bằng dấu phẩy: 1,-2,3,4,5,-6,7,8,-9
List: [1, -2, 3, 4, 5, -6, 7, 8, -9]
Tuple từ List: (1, -2, 3, 4, 5, -6, 7, 8, -9)
PS D:\lab-01\ex03> 
```

- **Câu 04:** Viết chương trình để truy cập các phần tử đầu tiên và cuối cùng trong một Tuple.

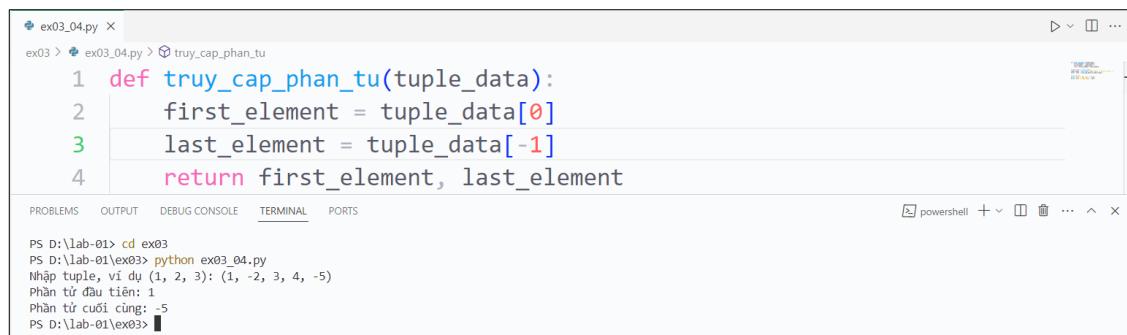
Hướng dẫn: Tạo file “ex03_04.py” trong “ex03”.

```

1 def truy_cap_phan_tu(tuple_data):
2     first_element = tuple_data[0]
3     last_element = tuple_data[-1]
4     return first_element, last_element
5
6 # Nhập tuple từ người dùng
7 input_tuple = eval(input("Nhập tuple, ví dụ (1, 2, 3): "))
8 first, last = truy_cap_phan_tu(input_tuple)
9
10 # In kết quả
11 print("Phần tử đầu tiên:", first)
12 print("Phần tử cuối cùng:", last)

```

Kết quả:



```

ex03_04.py > ...
ex03_04.py > truy_cap_phan_tu(tuple_data):
1 def truy_cap_phan_tu(tuple_data):
2     first_element = tuple_data[0]
3     last_element = tuple_data[-1]
4     return first_element, last_element
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell + v ⌂ ... ^ x
PS D:\lab-01> cd ex03
PS D:\lab-01\ex03> python ex03_04.py
Nhập tuple, ví dụ (1, 2, 3): (1, -2, 3, 4, -5)
Phân tử đầu tiên: 1
Phân tử cuối cùng: -5
PS D:\lab-01\ex03> 
```

- **Câu 05:** Viết chương trình để đếm số lần xuất hiện của mỗi phần tử trong một List và lưu kết quả vào một Dictionary.

Hướng dẫn: Tạo file “ex03_05.py” trong “ex03”.

```

1 def dem_so_lan_xuat_hien(lst):
2     count_dict = {}
3     for item in lst:
4         if item in count_dict:
5             count_dict[item] += 1
6         else:
7             count_dict[item] = 1
8     return count_dict
9
10 # Nhập danh sách từ người dùng
11 input_string = input("Nhập danh sách các từ, cách nhau bằng dấu cách:
12 ")
12 word_list = input_string.split()
13
14 # Sử dụng hàm và in kết quả
15 so_lan_xuat_hien = dem_so_lan_xuat_hien(word_list)
16 print("Số lần xuất hiện của các phần tử:", so_lan_xuat_hien)

```

Kết quả:

The screenshot shows a code editor with a Python file named "ex03_05.py". The code defines a function "dem_so_lan_xuat_hien" that takes a list as input and returns a dictionary of word counts. It uses a for loop and an if statement to update the dictionary. Below the code editor is a terminal window showing the command "python ex03_05.py" being run, followed by the output: "Nhập danh sách các từ, cách nhau bằng dấu cách: hutech, khoa, cong, nghe, thong, tin, bao, mat, thong, tin, Số lần xuất hiện của các phần tử: {'hutech': 1, 'khoa': 1, 'cong': 1, 'nghe': 1, 'thong': 2, 'tin': 2, 'bao': 1, 'mat': 1}".

- **Câu 06:** Viết chương trình để xóa một phần tử từ Dictionary theo key đã cho.

Hướng dẫn: Tạo file “ex03_06.py” trong “ex03”.

```

1 def xoa_phan_tu(dictionary, key):
2     if key in dictionary:
3         del dictionary[key]
4         return True
5     else:
6         return False
7
8 # Sử dụng hàm và in kết quả
9 my_dict = {'a': 1, 'b': 2, 'c': 3, 'd': 4}
10 key_to_delete = 'b'
11 result = xoa_phan_tu(my_dict, key_to_delete)
12 if result:
13     print("Phần tử đã được xóa từ Dictionary:", my_dict)
14 else:
15     print("Không tìm thấy phần tử cần xóa trong Dictionary.")

```

Kết quả:

```

ex03_06.py X
ex03 > ex03_06.py > ...
1 def xoa_phan_tu(dictionary, key):
2     if key in dictionary:
3         del dictionary[key]
4         return True
5     else:
6         return False
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\lab-01> cd ex03
PS D:\lab-01\ex03> python ex03_06.py
Phần tử đã được xóa từ Dictionary: {'a': 1, 'c': 3, 'd': 4}
PS D:\lab-01\ex03>

```

Đưa các thay đổi lên Git repo:

```

git add .
git commit -m "[add] lab-01 ex03"
git push origin lab01

```

```

PS D:\bmtt-nc-hutech-1511060249\lab-01\ex03> git push origin lab-01
Enumerating objects: 12, done.
Counting objects: 100% (12/12), done.
Delta compression using up to 8 threads
Compressing objects: 100% (10/10), done.
Writing objects: 100% (10/10), 2.02 KiB | 2.02 MiB/s, done.
Total 10 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/mrphuoc020597/bmtt-nc-hutech-1511060249.git
 8a81644..1b91322 lab-01 -> lab-01
PS D:\bmtt-nc-hutech-1511060249\lab-01\ex03>

```

1.6.4 Bài thực hành 04: OOP trong Python

Hãy xây dựng một chương trình Python quản lý thông tin sinh viên. Mỗi sinh viên sẽ có các thuộc tính: mã sinh viên (id), tên, giới tính, chuyên ngành, và điểm trung bình (hệ 10). Trong đó, mã sinh viên sẽ tự động tăng cho mỗi đối tượng mới.

Học lực được tính như sau:

- Loại Giỏi: nếu điểm trung bình từ 8 trở lên.
- Loại Khá: nếu điểm trung bình từ 6,5 đến dưới 8.
- Loại Trung bình: nếu điểm trung bình từ 5 đến dưới 6,5.
- Loại Yếu: nếu điểm trung bình dưới 5.

Yêu cầu: Tạo ra một menu với các chức năng sau:

- Thêm sinh viên.
- Cập nhật thông tin sinh viên theo ID.
- Xóa sinh viên theo ID.
- Tìm kiếm sinh viên qua tên.
- Sắp xếp danh sách sinh viên theo điểm trung bình.
- Sắp xếp danh sách sinh viên theo tên chuyên ngành.
- Hiển thị danh sách sinh viên.

Hướng dẫn:

- Tại folder "lab-01", tạo folder "ex04". Tạo file "SinhVien.py" trong "ex04".

```
1 class SinhVien:  
2     def __init__(self, id, name, sex, major, diemTB):  
3         self._id = id  
4         self._name = name  
5         self._sex = sex  
6         self._major = major  
7         self._diemTB = diemTB  
8         self._hocLuc = ""
```

- Tạo file “QuanLySinhVien.py” trong “ex04”.

```
1 from SinhVien import SinhVien
2
3 class QuanLySinhVien:
4     listSinhVien = []
5
6     def generateID(self):
7         maxId = 1
8         if (self.soLuongSinhVien() > 0):
9             maxId = self.listSinhVien[0]._id
10            for sv in self.listSinhVien:
11                if (maxId < sv._id):
12                    maxId = sv._id
13            maxId = maxId + 1
14        return maxId
15
16    def soLuongSinhVien(self):
17        return self.listSinhVien.__len__()
18
19    def nhapSinhVien(self):
20        svId = self.generateID()
21        name = input("Nhập tên sinh viên: ")
22        sex = input("Nhập giới tính sinh viên: ")
23        major = input("Nhập chuyên ngành của sinh viên: ")
24        diemTB = float(input("Nhập điểm của sinh viên: "))
25        sv = SinhVien(svId, name, sex, major, diemTB)
26        self.xepLoaiHocLuc(sv)
27        self.listSinhVien.append(sv)
28
29    def updateSinhVien(self, ID):
30        sv:SinhVien = self.findById(ID)
31        if (sv != None):
32            name = input("Nhập tên sinh viên: ")
33            sex = input("Nhập giới tính sinh viên: ")
34            major = int(input("Nhập chuyên ngành của sinh viên: "))
35            diemTB = float(input("Nhập điểm của sinh viên: "))
36            sv._name = name
37            sv._sex = sex
38            sv._major = major
39            sv._diemTB = diemTB
40            self.xepLoaiHocLuc(sv)
```

```
41     else:
42         print("Sinh viên có ID = {} không tồn tại.".format(ID))
43
44     def sortByID(self):
45         self.listSinhVien.sort(key=lambda x: x._id, reverse=False)
46
47     def sortByName(self):
48         self.listSinhVien.sort(key=lambda x: x._name, reverse=False)
49
50     def sortByDiemTB(self):
51         self.listSinhVien.sort(key=lambda x: x._diemTB, reverse=False)
52
53     def findByID(self, ID):
54         searchResult = None
55         if (self.soluongSinhVien() > 0):
56             for sv in self.listSinhVien:
57                 if (sv._id == ID):
58                     searchResult = sv
59             return searchResult
60
61     def findByName(self, keyword):
62         listSV = []
63         if (self.soluongSinhVien() > 0):
64             for sv in self.listSinhVien:
65                 if (keyword.upper() in sv._name.upper()):
66                     listSV.append(sv)
67             return listSV
68
69     def deleteById(self, ID):
70         isDeleted = False
71         sv = self.findByID(ID)
72         if (sv != None):
73             self.listSinhVien.remove(sv)
74             isDeleted = True
75         return isDeleted
76
77     def xepLoaiHocLuc(self, sv:SinhVien):
78         if (sv._diemTB >= 8):
79             sv._hocLuc = "Gioi"
80         elif (sv._diemTB >= 6.5):
81             sv._hocLuc = "Kha"
```

```

81         sv._hocLuc = "Kha"
82     elif (sv._diemTB >= 5):
83         sv._hocLuc = "Trung binh"
84     else:
85         sv._hocLuc = "Yeu"
86
87     def showSinhVien(self, listSV):
88         print("{:<8} {:<18} {:<8} {:<8}{:<8} {:<8}"
89             .format("ID", "Name", "Sex", "Major", "Diem TB", "Hoc
90             Luc"))
91         if (listSV.__len__() > 0):
92             for sv in listSV:
93                 print("{:<8} {:<18} {:<8} {:<8}{:<8} {:<8} "
94                     .format(sv._id, sv._name, sv._sex, sv._major,
95                     sv._diemTB, sv._hocLuc))
96
97     def getListSinhVien(self):
98         return self.listSinhVien

```

- Tạo file “Main.py” trong “ex04”.

```

1  from QuanLySinhVien import QuanLySinhVien
2
3  qlsv = QuanLySinhVien()
4  while (1 == 1):
5      print("\nCHUONG TRINH QUAN LY SINH VIEN")
6      print("*****MENU*****")
7      print("** 1. Them sinh vien.          **")
8      print("** 2. Cap nhat thong tin sinh vien boi ID.      **")
9      print("** 3. Xoa sinh vien boi ID.          **")
10     print("** 4. Tim kiem sinh vien theo ten.      **")
11     print("** 5. Sap xep sinh vien theo diem trung binh.      **")
12     print("** 6. Sap xep sinh vien theo ten chuyen nganh.      **")
13     print("** 7. Hien thi danh sach sinh vien.      **")
14     print("** 0. Thoat          **")
15     print("*****")
16
17     key = int(input("Nhap tuy chon: "))
18     if (key == 1):
19         print("\n1. Them sinh vien.")
20         qlsv.nhapSinhVien()
21         print("\nThem sinh vien thanh cong!")

```

```
22     elif (key == 2):
23         if (qlsv.soLuongSinhVien() > 0):
24             print("\n2. Cap nhat thong tin sinh vien. ")
25             print("\nNhap ID: ")
26             ID = int(input())
27             qlsv.updateSinhVien(ID)
28         else:
29             print("\nSanh sach sinh vien trong!")
30     elif (key == 3):
31         if (qlsv.soLuongSinhVien() > 0):
32             print("\n3. Xoa sinh vien.")
33             print("\nNhap ID: ")
34             ID = int(input())
35             if (qlsv.deleteById(ID)):
36                 print("\nSinh vien co id = ", ID, " da bi xoa.")
37             else:
38                 print("\nSinh vien co id = ", ID, " khong ton tai.")
39         else:
40             print("\nSanh sach sinh vien trong!")
41     elif (key == 4):
42         if (qlsv.soLuongSinhVien() > 0):
43             print("\n4. Tim kiem sinh vien theo ten.")
44             print("\nNhap ten de tim kiem: ")
45             name = input()
46             searchResult = qlsv.findByName(name)
47             qlsv.showSinhVien(searchResult)
48         else:
49             print("\nSanh sach sinh vien trong!")
50     elif (key == 5):
51         if (qlsv.soLuongSinhVien() > 0):
52             print("\n5. Sap xep sinh vien theo diem trung binh (GPA). ")
53             qlsv.sortByDiemTB()
54             qlsv.showSinhVien(qlsv.getListSinhVien())
55         else:
56             print("\nSanh sach sinh vien trong!")
57     elif (key == 6):
```

```

58     if (qlsv.soluongSinhVien() > 0):
59         print("\n6. Sap xep sinh vien theo ten.")
60         qlsv.sortByName()
61         qlsv.showSinhVien(qlsv.getListSinhVien())
62     else:
63         print("\nSanh sach sinh vien trong!")
64     elif (key == 7):
65         if (qlsv.soluongSinhVien() > 0):
66             print("\n7. Hien thi danh sach sinh vien.")
67             qlsv.showSinhVien(qlsv.getListSinhVien())
68         else:
69             print("\nSanh sach sinh vien trong!")
70     elif (key == 0):
71         print("\nBan da chon thoat chuong trinh!")
72         break
73     else:
74         print("\nKhong co chuc nang nay!")
75         print("\nHay chon chuc nang trong hop menu.")

```

Kết quả: Chạy file “Main.py” và kiểm tra chương trình.

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

*****
*****MENU*****
PS D:\bmtt-nc-hutech-1511060249\lab-01\ex04> python .\Main.py
CHUONG TRINH QUAN LY SINH VIEN
*****
** 1. Them sinh vien.          **
** 2. Cap nhat thong tin sinh vien boi ID.      **
** 3. Xoa sinh vien boi ID.      **
** 4. Tim kiem sinh vien theo ten.    **
** 5. Sap xep sinh vien theo diem trung binh.  **
** 6. Sap xep sinh vien theo ten chuyen nganh.   **
** 7. Hien thi danh sach sinh vien.    **
** 8. Thoat.                      **
*****
Nhap tuy chon: 1
1. Them sinh vien.
Nhap ten sinh vien: phuoc nguyen
Nhap gioi tinh sinh vien: nam
Nhap chuyen nganh cua sinh vien: cmtt
Nhap diem cua sinh vien: 8.5

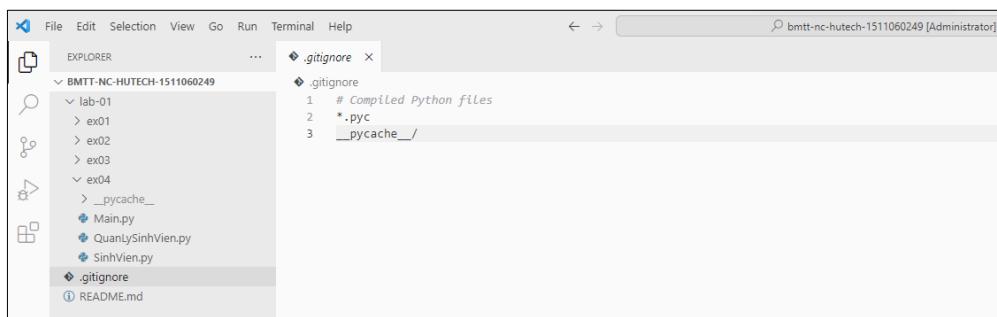
Them sinh vien thanh cong!

CHUONG TRINH QUAN LY SINH VIEN
*****
** 1. Them sinh vien.          **
** 2. Cap nhat thong tin sinh vien boi ID.      **
** 3. Xoa sinh vien boi ID.      **
** 4. Tim kiem sinh vien theo ten.    **
** 5. Sap xep sinh vien theo diem trung binh.  **
** 6. Sap xep sinh vien theo ten chuyen nganh.   **
** 7. Hien thi danh sach sinh vien.    **
** 8. Thoat.                      **
*****
Nhap tuy chon: 7
7. Hien thi danh sach sinh vien.
ID      Name      Sex      Major      Diem TB      Hoc Luc
1      phuoc nguyen      nam      cmtt      8.5      Gioi

```

Thêm file “.gitignore” tại thư mục gốc để không đưa các file cache trong thư mục “__pycache__” lên Git repo.

```
# Compiled Python files
*.pyc
__pycache__/
```



Đưa các thay đổi lên Git repo:

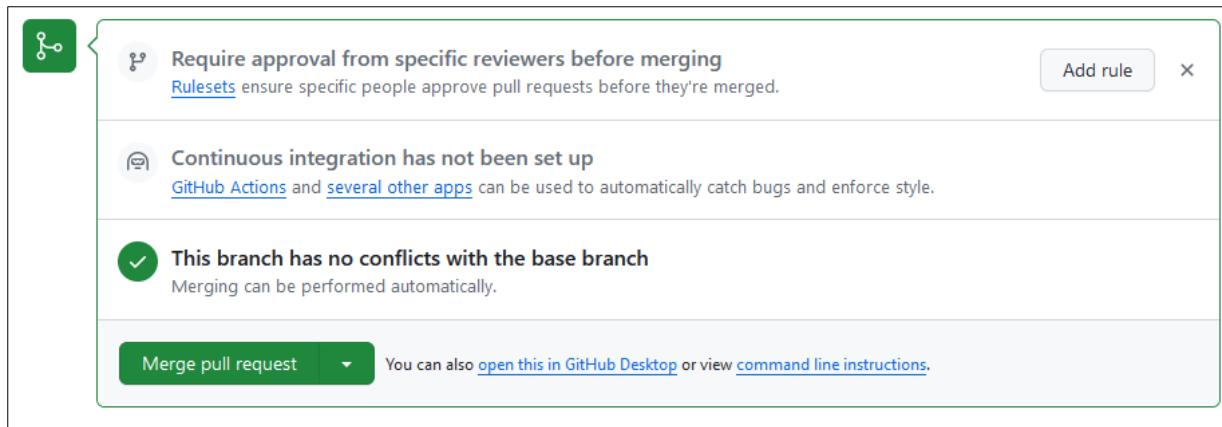
```
git add .
git commit -m "[add] lab-01 ex04 and .gitignore"
git push origin lab01
```

Tạo PR từ branch lab01 về branch main trên Git repo:

- Vào địa chỉ của Git repo. Chọn menu “Pull requests”. Chọn “New pull request”.
- Chọn compare branch là “lab01”. Chọn base branch là “main”. Kiểm tra các thay đổi và nhấn “Create pull request”.

Commit Details	Date	SHA
[add] lab-01 ex02	1 hour ago	8a81644
[add] lab-01 ex03	1 hour ago	1b91322
[add] lab-01 ex04	32 minutes ago	8dd705a
[add] .gitignore	30 minutes ago	dc3c34c
[add] lab-01 ex04	30 minutes ago	1f4fece
[add] lab-01 ex04	28 minutes ago	e18a9c7
[add] lab-01 ex04	27 minutes ago	14545cb

- Một lần nữa, kiểm tra lại các thay đổi đã phù hợp hay chưa. Sau đó chọn “Merge pull request” để hoàn tất.



Kết quả:



1.7 BÀI TẬP MỞ RỘNG

- **Câu 01:** Hãy phát triển một chương trình để liệt kê tất cả các hoán vị của danh sách [1, 2, 3]. Bạn có thể tham khảo việc sử dụng "**itertools.permutations()**" để thu thập tất cả các hoán vị có thể của danh sách này.
- **Câu 02:** Hãy xây dựng một chương trình để tính tổng của tất cả các số nguyên dương và âm có trong một chuỗi được nhập vào. Ví dụ:

Chuỗi ban đầu là "-100#^sdfkj8902w3ir021@swf-20".

Kết quả: Giá trị dương: 9046. Giá trị âm: -120.