BUỔI 1: NGÔN NGỮ LẬP TRÌNH PYTHON (1)

MỤC TIÊU:

- Nắm bắt kiến thức cơ bản và kỹ năng về lập trình Python.
- Sử dụng ngôn ngữ lập trình Python để lập trình Web.

I. GIỚI THIỆU

Python được Guido van Rossum phát triển vào cuối những năm tám mươi và đầu những năm chín mươi tại Viện nghiên cứu quốc gia về toán học và khoa học máy tính ở Hà Lan.

Python có nguồn gốc từ nhiều ngôn ngữ khác, bao gồm ABC, Modula-3, C, C++, Algol-68, SmallTalk và Unix shell và các ngôn ngữ script khác. Python có bản quyền. Giống như Perl, mã nguồn Python hiện có sẵn theo giấy phép GNU (GPL).

Python hiện được duy trì bởi một nhóm phát triển cốt lõi tại viện nghiên cứu quốc gia về toán học và khoa học máy tính ở Hà Lan. Guido van Rossum vẫn giữ một vai trò quan trọng trong việc chỉ đạo tiến trình của nó.

Python đã trở thành một trong những ngôn ngữ lập trình phổ biến nhất trên thế giới. Điều này là do Python có cú pháp đơn giản, dễ đọc, dễ hiểu và đa năng. Python có thư viện phong phú và cộng đồng hỗ trợ lớn, giúp việc phát triển ứng dụng nhanh chóng và hiệu quả.

II. CHUẨN BỊ

- Python 3 (khuyến khích dùng Python 3.10): https://www.python.org/ hoặc https://www.anaconda.com/ (lưu ý khi cài đặt. hãy tích vào ô PATH)
- Visual Studio Code: https://code.visualstudio.com/download
- Extensions cài đặt ở trong Visual Studio Code: Python

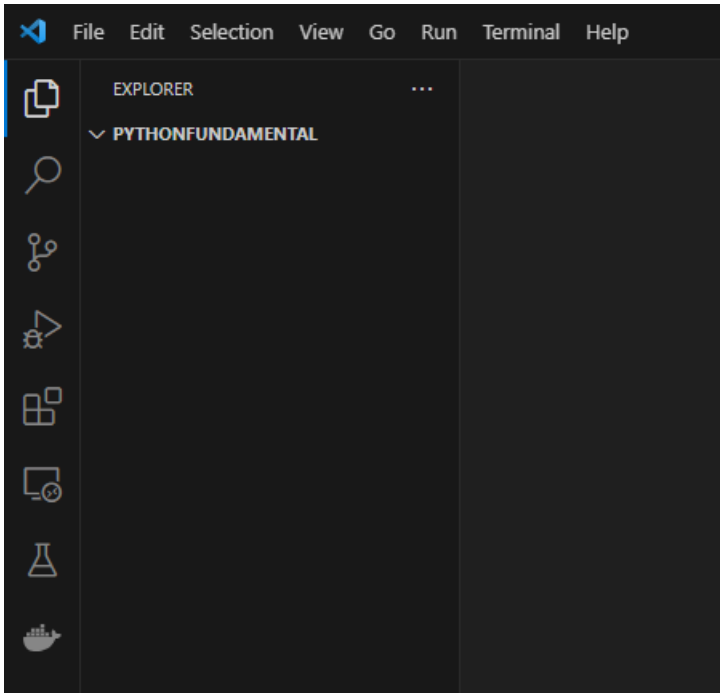
III. THỰC HÀNH

1. Chương trình Python đầu tiên (HelloWorld.py)

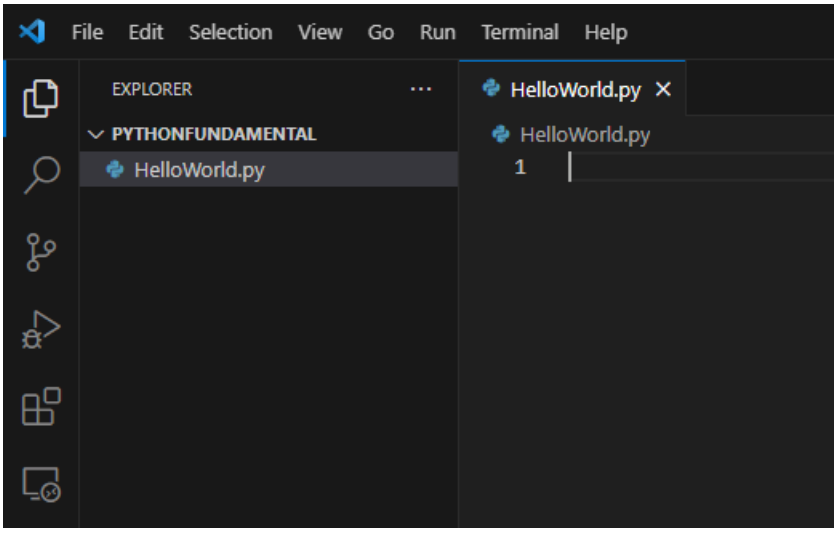
Đầu tiên, để lập trình với ngôn ngữ Python một cách dễ dàng, chúng ta sẽ sử dụng IDE Visual Studio Code để dễ dàng lập trình một cách hơn.

Bắt đầu bằng việc mở Visual Studio Code, sau đó vào phần Open Folder, hãy chọn thư mục để làm việc và lưu trữ tệp tin ở trên đó.

Ví dụ ở đây chúng ta sẽ tạo 1 thư mục tên là "PythonFundamental"

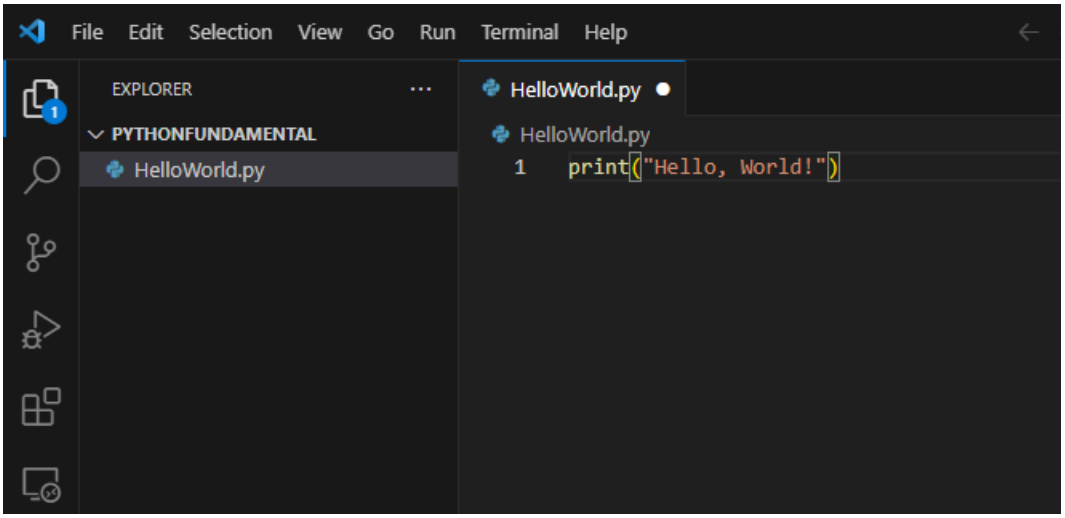


Sau đó chúng ta sẽ tạo tệp Python (.py) đặt tên là "HelloWorld.py"



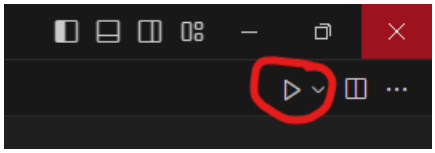
Bây giờ chúng ta sẽ bắt đầu lập trình tệp Python đầu tiên bằng cách in ra màn hình "Hello, World!" bằng câu lệnh sau:

```
print("Hello, World!")
```



Lưu tệp bằng cách vào File > Save hoặc Ctrl+S rồi bắt đầu chạy chương trình theo 2 cách:

- Terminal: python HelloWorld.py
- Visual Studio Code: bấm nút chạy ở trên góc phải



Kết quả:

```
Hello, World!
```

2. Các kiểu dữ liệu cơ bản trong Python

- **Kiểu số (Number):**
  int: Kiểu số nguyên, ví dụ: 5, -10, 1000
  float: Kiểu số thực, ví dụ: 3.14, -0.5, 2.0

```
# Khai báo số nguyên
SoNguyen = 10

# Khai báo số thực
SoThuc = 10.5
```

- **Kiểu chuỗi (String):**
  str: Kiểu dữ liệu chuỗi ký tự, ví dụ: "Hello", 'Python', "123".

```
# Khai báo chuỗi
Chuoi = "Hello World"
```

- **Kiểu boolean:**
  bool: Kiểu dữ liệu logic, chỉ có hai giá trị True (đúng) hoặc False (sai).

```
# khai báo giá trị logic
GiaTriLogic = True
```

3. Biến trong Python

Biến là một cái tên được sử dụng để định danh và lưu trữ giá trị trong bộ nhớ. Mỗi biến có một kiểu dữ liệu và có thể thay đổi giá trị trong quá trình thực thi của chương trình.

Để khai báo biến trong Python, chỉ cần sử dụng cú pháp sau:

```
ten_bien = <gia_tri>
```

Ví dụ:

```
x = 5
y = "Hello"
```

Các quy tắc khi đặt tên biến:

- Tên biến chỉ chứa chữ cái, chữ số và dấu gạch dưới (_).
- Tên biến không được bắt đầu bằng số.
- Tên biến không được chứa các ký tự đặc biệt như !, @, #, $, %, etc.
- Tên biến không được trùng với các từ khoá (keywords) của Python như if, for, while,...

Để nhập giá trị từ bàn phím vào biến, có thể sử dụng hàm input(). Hàm này trả về một chuỗi và cần chuyển đổi nó thành kiểu dữ liệu mong muốn (int, float,...) nếu cần.

```
name = input("Nhập tên của bạn: ")
age = int(input("Nhập tuổi của bạn: "))
```

Lưu ý:

- Hàm input() luôn trả về một chuỗi, vì vậy nếu cần một kiểu dữ liệu khác, hãy chuyển đổi nó bằng cách sử dụng các hàm như int(), float(),...
- Hàm input() luôn trả về một chuỗi, vì vậy nếu cần một kiểu dữ liệu khác, hãy chuyển đổi nó bằng cách sử dụng các hàm như int(), float(),...

4. Các toán tử trong Python

Python hỗ trợ một loạt các toán tử cho các phép tính, so sánh và các phép toán khác. Dưới đây là một danh sách các toán tử cơ bản trong Python:

a. Toán tử số học:

- **Phép cộng (+):** Cộng hai số hoặc nối chuỗi.
- **Phép trừ (-):** Trừ một số từ số khác.
- **Phép nhân (*):** Nhân hai số hoặc nhân một chuỗi với một số nguyên dương để lặp lại chuỗi.
- **Phép chia (/):** Chia một số cho số khác.
- **Phép chia lấy phần nguyên (//):** Chia và lấy phần nguyên của kết quả.
- **Phép chia lấy dư (%):** Trả về phần dư của phép chia.
- **Luỹ thừa (**):** Tính lũy thừa của một số.

b. Toán tử so sánh:

- **Bằng (==):** So sánh xem hai giá trị có bằng nhau không.
- **Khác (!=):** So sánh xem hai giá trị có khác nhau không.
- **Lớn hơn (>):** Kiểm tra xem giá trị bên trái có lớn hơn giá trị bên phải không.
- **Nhỏ hơn (<):** Kiểm tra xem giá trị bên trái có nhỏ hơn giá trị bên phải không.
- **Lớn hơn hoặc bằng (>=):** Kiểm tra xem giá trị bên trái có lớn hơn hoặc bằng giá trị bên phải không.
- **Nhỏ hơn hoặc bằng (<=):** Kiểm tra xem giá trị bên trái có nhỏ hơn hoặc bằng giá trị bên phải không.

c. Toán tử logic:

- **Và (and):** Trả về True nếu cả hai biểu thức đều đúng.
- **Hoặc (or):** Trả về True nếu ít nhất một biểu thức đúng.
- **Phủ định (not):** Đảo ngược giá trị của biểu thức.

d. Toán tử gán:

- **Gán (=):** Gán giá trị của biểu thức bên phải cho biến bên trái.
- **Cộng và gán (+=):** Cộng giá trị bên phải vào biến bên trái và gán lại vào biến bên trái.
- **Trừ và gán (-=):** Trừ giá trị bên phải từ biến bên trái và gán lại vào biến bên trái.
- **Nhân và gán (*=):** Nhân giá trị bên phải với biến bên trái và gán lại vào biến bên trái.

- **Chia và gán (/=):** Chia giá trị bên phải cho biến bên trái và gán lại vào biến bên trái.
- **Chia lấy phần nguyên và gán (//=):** Chia và lấy phần nguyên của kết quả, sau đó gán vào biến bên trái.
- **Chia lấy dư và gán (%=):** Chia và trả về phần dư, sau đó gán vào biến bên trái.

Ví dụ:

```
x = 10
y = 5

# Toán tử số học
sum_result = x + y
sub_result = x - y
mul_result = x * y
div_result = x / y
floor_div_result = x // y
mod_result = x % y
exponent_result = x ** y

# Toán tử so sánh
is_equal = x == y
is_not_equal = x != y
is_greater = x > y
is_less = x < y
is_greater_or_equal = x >= y
is_less_or_equal = x <= y

# Toán tử logic
and_result = (x > 0) and (y > 0)
or_result = (x > 0) or (y > 0)
not_x = not (x > 0)

# Toán tử gán
x += 1  # Tương đương với x = x + 1
```

5. List, Tuple, Dictionary

Trong Python, List, Tuple và Dictionary là ba kiểu dữ liệu cơ bản dùng để lưu trữ và quản lý dữ liệu. Mỗi kiểu dữ liệu này có đặc điểm và cách sử dụng riêng biệt.

a. List

List là một kiểu dữ liệu linh hoạt cho phép lưu trữ nhiều giá trị khác nhau trong một biến. Các giá trị trong list được phân cách bởi dấu phẩy và được bao quanh bởi dấu ngoặc vuông [].

Khởi tạo và truy cập phần tử:

```
# Các cách khai báo list
List1 = [1, 2, 3, 4, 5]
List2 = [1, "Hello", 3.5, True]
List3 = list((1, 2, 3, 4, 5))
```

Để truy cập các phần tử trong list, sử dụng chỉ số (index) của phần tử đó, bắt đầu từ 0:

```
first_element = my_list[0]  # Lấy phần tử đầu tiên (có chỉ số 0)
second_element = my_list[1]  # Lấy phần tử thứ hai (có chỉ số 1)
```

List có thể thay đổi (mutable), có nghĩa là có thể thay đổi giá trị của các phần tử sau khi list đã được khởi tạo.

Một số Phương thức mở rộng cho List:

- **append():** Thêm một phần tử vào cuối danh sách

```
my_list = [1, 2, 3]
my_list.append(4)
print(my_list)  # Output: [1, 2, 3, 4]
```

- **extend():** Nối một danh sách (hoặc iterable) vào cuối danh sách hiện tại**.**

```
my_list = [1, 2, 3]
another_list = [4, 5, 6]
my_list.extend(another_list)
print(my_list)  # Output: [1, 2, 3, 4, 5, 6]
```

- **insert():** Chèn một phần tử vào vị trí chỉ định**.**

```
my_list = [1, 2, 3]
my_list.insert(1, 5)  # Chèn số 5 vào vị trí có chỉ số 1
print(my_list)  # Output: [1, 5, 2, 3]
```

- **remove():** Xóa một phần tử cụ thể từ danh sách.

```
my_list = [1, 2, 3, 2]
my_list.remove(2)  # Xóa phần tử đầu tiên có giá trị là 2
print(my_list)  # Output: [1, 3, 2]
```

- **pop():** Xóa và trả về phần tử tại chỉ mục đã cho.

```
my_list = [1, 2, 3]
popped_element = my_list.pop(1)  # Xóa và trả về phần tử tại chỉ số 1
print(my_list)  # Output: [1, 3]
print(popped_element)  # Output: 2
```

- **clear():** Xóa tất cả phần tử.

```
my_list = [1, 2, 3]
my_list.clear()
print(my_list)  # Output: []
```

- **index():** Trả về chỉ mục (index) đầu tiên của một phần tử.

```
my_list = [1, 2, 3, 2]
index = my_list.index(2)  # Trả về chỉ mục đầu tiên của phần tử có giá trị 2
print(index)  # Output: 1
```

- **count():** Đếm số lần một phần tử xuất hiện trong danh sách.

```
my_list = [1, 2, 3, 2, 2]
count = my_list.count(2)  # Đếm số lần phần tử có giá trị 2 xuất hiện
print(count)  # Output: 3
```

- **sort():** Sắp xếp danh sách theo thứ tự tăng dần hoặc giảm dần.

```
my_list = [3, 1, 2, 4]
my_list.sort()  # Sắp xếp tăng dần
print(my_list)  # Output: [1, 2, 3, 4]

my_list.sort(reverse=True)  # Sắp xếp giảm dần
print(my_list)  # Output: [4, 3, 2, 1]
```

- **reverse():** Đảo ngược thứ tự các phần tử trong danh sách.

```
my_list = [1, 2, 3, 4]
my_list.reverse()
print(my_list)  # Output: [4, 3, 2, 1]
```

b. Tuple

Tuple cũng tương tự như list, nhưng không thể thay đổi giá trị của các phần tử sau khi tuple đã được khởi tạo. Các phần tử trong tuple được phân cách bởi dấu phẩy và được bao quanh bởi dấu ngoặc đơn ().

Khởi tạo và truy cập phần tử:

```
my_tuple = (1, 2, 3, 4, 5)
```

Để truy cập các phần tử trong tuple, cũng sử dụng chỉ số như trong list:

```
first_element = my_tuple[0]
second_element = my_tuple[1]
```

Tuple không thể thay đổi (immutable), có nghĩa là không thể thay đổi giá trị của các phần tử sau khi tuple đã được khởi tạo.

c. Dictionary

Dictionary (hay dict) là một kiểu dữ liệu cho phép lưu trữ dữ liệu dưới dạng cặp key-value. Mỗi key phải là duy nhất trong một dictionary.

Khởi tạo và truy cập phần tử:

```
my_dict = {'name': 'John', 'age': 30, 'city': 'New York'}
```

Để truy cập giá trị của một key trong dictionary:

```
name = my_dict['name']  # Lấy giá trị tương ứng với key 'name'
age = my_dict['age']  # Lấy giá trị tương ứng với key 'age'
```

Dictionary là một cấu trúc dữ liệu linh hoạt cho việc lưu trữ và truy cập dữ liệu theo key-value pairs.

```
my_dict = {'name': 'John', 'age': 30, 'city': 'New York'}
print(my_dict['name'])  # Output: 'John'
print(my_dict['age'])  # Output: 30
print(my_dict['city'])  # Output: 'New York'
```

Một số phương thức mở rộng:

- **clear():** Xóa tất cả các cặp key-value từ dictionary.

```
my_dict = {'name': 'John', 'age': 30, 'city': 'New York'}
my_dict.clear()
print(my_dict)  # Output: {}
```

- **get():** Trả về giá trị của key được chỉ định. Nếu key không tồn tại, nó sẽ trả về giá trị mặc định (hoặc None nếu không được cung cấp).

```
my_dict = {'name': 'John', 'age': 30, 'city': 'New York'}
name = my_dict.get('name')
gender = my_dict.get('gender', 'Unknown') # Nếu 'gender' không tồn tại, trả về
'Unknown'
print(name)  # Output: 'John'
print(gender)  # Output: 'Unknown'
```

- **items():** Trả về một danh sách các cặp key-value dưới dạng tuple.

```
my_dict = {'name': 'John', 'age': 30, 'city': 'New York'}
items = my_dict.items()
print(items)  # Output: dict_items([('name', 'John'), ('age', 30), ('city', 'New
York')])
```

- **keys():** Trả về một danh sách các key trong dictionary.

```
my_dict = {'name': 'John', 'age': 30, 'city': 'New York'}
keys = my_dict.keys()
print(keys)  # Output: dict_keys(['name', 'age', 'city'])
```

- **values():** Trả về một danh sách các giá trị trong dictionary.

```
my_dict = {'name': 'John', 'age': 30, 'city': 'New York'}
values = my_dict.values()
print(values)  # Output: dict_values(['John', 30, 'New York'])
```

- **pop():** Xóa và trả về giá trị của một key.

```
my_dict = {'name': 'John', 'age': 30, 'city': 'New York'}
name = my_dict.pop('name')
print(my_dict)  # Output: {'age': 30, 'city': 'New York'}
print(name)  # Output: 'John'
```

- **popitem():** Xóa và trả về cặp key-value cuối cùng.

```
my_dict = {'name': 'John', 'age': 30, 'city': 'New York'}
item = my_dict.popitem()
print(my_dict)  # Output: {'name': 'John', 'age': 30}
print(item)  # Output: ('city', 'New York')
```

6. Câu lệnh điều kiện và vòng lặp

a. Câu lệnh điều kiện

Câu lệnh điều kiện if được sử dụng để thực thi một khối mã chỉ khi một điều kiện cụ thể đúng.

Cú pháp thông thường:

```
if condition:
    # Nếu điều kiện đúng, thực thi khối mã này
else:
    # Nếu điều kiện sai, thực thi khối mã này
```

Ví dụ:

```
x = 10
if x > 5:
    print("x lớn hơn 5")
else:
    print("x không lớn hơn 5")
```

Ngoài cú pháp trên, câu lệnh điều kiện đầy đủ nhất có thêm elif (viết tắt của "else if") là một phần của câu lệnh điều kiện if-elif-else trong Python. Nó cho phép kiểm tra nhiều điều kiện khác nhau và thực thi mã tương ứng với điều kiện đầu tiên đúng.

Cú pháp:

```
if condition_1:
    # Thực thi khi condition_1 đúng
elif condition_2:
    # Thực thi khi condition_2 đúng
elif condition_3:
    # Thực thi khi condition_3 đúng
#...
else:
    # Thực thi nếu không có điều kiện nào đúng
```

Ví dụ:

```
x = 10

if x > 5:
    print("x lớn hơn 5")
elif x == 5:
    print("x bằng 5")
else:
    print("x nhỏ hơn 5")
```

Lưu ý:

Câu điều kiện không nhất thiết đầy đủ cấu trúc if-elif-else, trong một số trường hợp có thể chỉ cần sử dụng if là đủ.

b. Vòng lặp for

Vòng lặp for là một cấu trúc điều khiển cho phép lặp lại một khối mã một số lượng cố định lần.

Lặp qua các phần tử trong một chuỗi hoặc danh sách:

```
for item in iterable:
    # Thực thi khối mã với mỗi phần tử trong iterable
```

Ví dụ:

```
my_list = [1, 2, 3, 4, 5]
for num in my_list:
    print(num) # Output: 1 2 3 4 5
```

Lặp qua một phạm vi số nguyên:

```
for i in range(start, end, step):
    # Thực thi khối mã với mỗi số nguyên trong phạm vi
```

Ví dụ:

```
for i in range(5):
    print(i) # Output: 0 1 2 3 4
```

Lưu ý:

Với trường hợp sử dụng **range()** thì start và step là tùy chọn (Không cung cấp start hay step thì mặc định start là 0, step là 1)

c. Vòng lặp While

Vòng lặp while trong Python được sử dụng để lặp lại một khối mã miễn là một điều kiện cụ thể vẫn còn đúng. Nó kiểm tra điều kiện trước khi thực thi khối mã và sẽ tiếp tục lặp lại cho đến khi điều kiện trở thành sai.

Cú pháp:

```
while condition:
    # Thực thi khối mã miễn là điều kiện còn đúng
```

Ví dụ:

```
while x < 5:
    print(x)
    x += 1
```

Kết quả:

```
0
1
2
3
4
```

Lưu ý:

- **Đảm bảo điều kiện dừng**: Trong vòng lặp while, điều kiện kiểm tra trước khi thực thi khối mã. Đảm bảo rằng điều kiện sẽ trở thành sai tại một điểm nào đó để tránh vòng lặp vô hạn.
- **Cập nhật biến điều kiện**: Đảm bảo cập nhật biến sử dụng trong điều kiện kiểm tra. Nếu không, điều kiện có thể không bao giờ trở thành sai.
- **Điều kiện ban đầu**: Chắc chắn rằng điều kiện ban đầu là đúng để khởi đầu vòng lặp. Nếu điều kiện ban đầu là sai, vòng lặp sẽ không được thực hiện.

- **Sử dụng vòng lặp while khi không biết trước số lần lặp cần thiết**: Khi không biết trước số lần lặp cần thiết và muốn dừng lại dựa trên một điều kiện cụ thể, sử dụng vòng lặp while là lựa chọn hợp lý.