0.1 Cú pháp Python cơ bản

0.1.1 Cài đặt môi trường

Giống như các ngôn ngữ lập trình khác như Java, C++,... khi muốn chạy một chương trình Python, trước hết cần cài đặt một môi trường có thể đọc và chạy được Python.

Hiện phiên bản mới nhất của Python được cập nhật và dowload miễn phí trên trang chủ python.org là Python 3.9.7.

Cùng với sự phát triển và phổ biến của Python, hiện có rất nhiều các editor có thể được sử dụng để lập trình Python. Dựa trên kết quả đánh giá chức năng, số lượng người dùng và phản hồi tích cực từ phía người dùng, có thể kể đến một số phần mềm lập trình như sau:

- **PyDev** - phát triển bởi Eclipse Foundation.
- **PyCharm** - phát triển bởi JetBrains.
- **Visual Studio Code** - phát triển bởi Mycrosoft.
- **Sublime Text** - phát triển bởi Sublime HQ.
- **Spyder** - phát triển bởi Pierre Raybaut, chuyên dùng để lập trình khoa học.

0.1.2 Biến

Biến chính xác như tên gọi của nó, tức là giá trị của nó có thể thay đổi. Các biến có thể giúp lưu trữ bất cứ cái gì nếu có thể định nghĩa được nó. Các biến chỉ là một phần của bộ nhớ máy tính nơi lưu trữ một số thông tin.

Xác định kiểu dữ liệu của biến bằng lệnh type(biến).

Có kiểu biến toàn cục và biến địa phương, biến được khai báo ngoài hàm là biến toàn cục còn biến được khai báo trong hàm là biến địa phương.

Ví dụ 0.1. Ví dụ về khai báo biến:

```
# Biến toàn cục
x = 5
def function():
    # biến cục bộ
    x = 10.5
    print(x)
```

0.1.3 Toán tử

Về cơ bản trong Python cũng có các toán tử số học, quan hệ, logic...tương tự như các ngôn ngữ khác.

- **Toán tử số học**

Toán tử	Mô tả	Ví dụ: a=20, b=10
+	Phép cộng	$a + b = 30$
	Phép trừ	$a - b = 10$
*	Phép nhân	$a * b = 200$
/	Phép chia lấy phần nguyên.	$a/b = 2$
%	Phép chia lấy phần dư.	$a \% b = 0$
**	Phép lấy số mũ	$a **b = 10^{20}$
//	Floor Division	$9//2 = 4$

Bảng 1: Toán tử số học

- **Toán tử quan hệ**

Toán tử	Mô tả	Ví dụ: a=20, b=10
==	So sánh bằng	$(a == b)$ cho kết quả False
! =	So sánh không bằng.	$(a! = b)$ cho kết quả True.
<>	Không bằng	$(a <> b)$ cho kết quả True.
>	Lớn hơn.	$(a > b)$ cho kết quả True.
<	Nhỏ hơn.	$(a < b)$ cho kết quả False
>=	Lớn hơn hoặc bằng.	$(a >= b)$ cho kết quả True
<=	Nhỏ hơn hoặc bằng.	$(a <= b)$ cho kết quả False

Bảng 2: Toán tử quan hệ

- **Toán tử logic**

Toán tử	Mô tả	Ví dụ: a=10, b=20
and	Nếu cả 2 biểu thức là True thì cho kết quả True.	$(a > 9 and b > 19) => True$
or	Nếu 1 trong 2 là True thì cho kết quả True.	$(a > 9 \, or \, b > 100) => True$
not	Được sử dụng phủ định logic của biểu thức.	$not\ (a > 9) => False$ cho kết quả True.

Bảng 3: Toán tử logic

- **Toán tử gán**
- **Toán tử khai thác**
- **Toán tử nhận dạng** : is và not is
- **Toán tử thao tác bit**

Thứ tự ưu tiên của các toán tử: Việc áp dụng nhiều toán tử trong một câu lệnh cần được chú ý đến thứ tự ưu tiên của các toán tử đó. Theo thứ tự ưu tiên giảm dần như sau: toán tử số

Toán tử	Mô tả	Ví dụ
$=$	Phép gán bằng.	$c = a + b$ gán $a + b$ cho c
$+=$	Phép gán cộng.	$c+ = a <=> c = c + a$
$-=$	Phép gán trừ.	$c - = a <=> c = c - a$
$*=$	Phép gán nhân.	$c * = a <=> c = c * a$
$/=$	Phép gán chia lấy phần nguyên.	$c / = a <=> c = c/a$
$\%=$	Phép gán chia lấy phần dư.	$c \% = a <=> c = c\%a$
$**=$	Phép gán lũy thừa.	$c ** = a <=> c = c * *a$
$//=$	Phép gán chia floor.	$c // = a <=> c = c//a$

Bảng 4: Toán tử gán

Toán tử	Mô tả
in	Trả về true nếu tìm thấy biến a trong tập các giá trị sau in, ngược lại trả về false.
not in	Trả về false nếu tìm thấy biến a trong tập các giá trị sau in, ngược lại trả về false.

Bảng 5: Toán tử khai thác

Toán tử	Mô tả
is	Kiểm tra xem hai giá trị có bằng nhau hay không. Toán tử này sẽ trả về True nếu a == b và ngược lại
not is	Ngược lại của is.

Bảng 6: Toán tử nhận dạng

Toán tử	Mô tả
&	AND
|	OR
^	XOR
~	NOT
<<	Dịch trái nhị phân
>>	Dịch phải nhị phân

Bảng 7: Toán tử thao tác bit

học, toán tử thao tác bit, toán tử quan hệ, toán tử gán, toán tử identify, toán tử membership, toán tử logic.

0.1.4 Các kiểu dữ liệu

Python cũng hỗ trợ các kiểu dữ liệu căn bản như:char, string, int, float, double... Ngoài ra Python cung cấp một loạt các dữ liệu phức hợp, thường được gọi là các chuỗi (sequence), sử dụng để nhóm các giá trị khác nhau như list, set , tuple và dictionary. Áp dụng hợp lý các kiểu dữ liệu này có thể khiến việc lập trình nhanh hơn rất nhiều.

- **List**: Đây là kiểu dữ liệu được đánh giá là đa năng nhất. List được biểu diễn bằng dãy các giá trị, được phân tách nhau bằng dấu phẩy, nằm trong dấu []. List không giới hạn số lượng mục, có thể có nhiều kiểu dữ liệu khác nhau trong cùng một list, như chuỗi, số nguyên, số thập phân,...

  Các giá trị trong list có chỉ số, có thứ tự, có thể thay đổi và cho phép lặp lại.

  Ví dụ 0.2. Khai báo và sử dụng kiểu dữ liệu danh sách:

```
list1 = [1, 2, 3.5, 10]
list2 = [1, 1, 'hello', 'my list', 10.1, True]
```

- **Tuple**: Tuple trong Python là một chuỗi các phần tử có thứ tự giống như list. Sự khác biệt giữa list và tuple là không thể thay đổi các phần tử trong tuple khi đã gán, nhưng trong list có thể thay đổi.

  Tuple thường được sử dụng cho các dữ liệu không cho phép sửa đổi và nhanh hơn list vì nó không thể thay đổi tự động.

  Ví dụ 0.3. Ví dụ về kiểu dữ liệu tuple:

```
mytuple = ("apple", "banana", "cherry")
mytuple = (1, 1, 2, 3.5, 5, "hello")
```

- **Set** : Set trong Python là tập hợp các phần tử duy nhất, không có thứ tự. Các phần tử trong set phân cách nhau bằng dấu phẩy và nằm trong dấu ngoặc nhọn . Các phần tử trong set có thể thay đổi.

  Ví dụ 0.4. Ví dụ về kiểu dữ liệu set:

```
set1 = {"apple", "banana", "cherry"}
set2 = {1, 5, 7, 9, 3}
set3 = {True, False, False}
```

- **Dictionary**: Dictionary là tập hợp các cặp khóa giá trị không có thứ tự. Các dictionary được tối ưu hóa để trích xuất dữ liệu với điều kiện phải biết được khóa để lấy giá trị. Trong Python, dictionary được định nghĩa trong dấu ngoặc nhọn với mỗi phần tử là một cặp theo dạng key:value. Key và value này có thể là bất kỳ kiểu dữ liệu nào.

  Ví dụ 0.5. Ví dụ về kiểu dữ liệu dictionary:

```
thisdict = {
    "brand": "Ford",
    "model": "Mustang",
    "year": 1964
  }
```

0.1.5 Hàm

Trong Python, hàm là một nhóm các lệnh có liên quan đến nhau được dùng để thực hiện một tác vụ, nhiệm vụ cụ thể nào đó. Hàm giúp chia chương trình thành những khối/phần/-module nhỏ hơn.

- Khi khởi tạo hàm không có kiểu dữ liệu trả về tuy nhiên ta vẫn có thể dùng lệnh return để trả về giá trị xác định.
- Các tham số mặc định của hàm là các tham số không bắt buộc phải cung cấp khi gọi hàm và khi đó giá trị của nó là giá trị mặc định. Sử dụng tham số mặc định bằng cách dùng toán tử =.
- Tham số tùy biến sử dụng khi ta không biết trước số lượng tham số sẽ truyền vào hàm. Khi đó biến có thể coi như một mảng các giá trị. Sử dụng tham số tùy biến bằng cách đặt dấu * trước tên tham số.

Ví dụ 0.6. Ví dụ về khai báo và định nghĩa hàm:

```
a = "Hello"

def Hello(a):
    print(a + "World")    # Hello World
```

0.1.6 Lập trình hướng đối tượng (OOP): Class

Python là một ngôn ngữ lập trình hỗ trợ hướng đối tượng tương tự như Java, C++, PHP... Nó cũng có các nguyên lý cơ bản của một ngôn ngữ hướng đối tượng đó là tính đóng gói, tính kế thừa, tính đa hình.

Hầu hết mọi thứ trong Python đều là một object, với tính chất và phương thức của riêng nó.

Class chính là một kiểu dữ liệu đặc biệt, dùng để tạo nên một đối tượng.

Ví dụ 0.7. Ví dụ về khai báo một class:

```
class MyClass:
    x = 5
```

Chúng ta có thể tạo một đối tượng từ class đã tạo:

Ví dụ 0.8. Tạo một đối tượng p1, và in ra giá trị của x:

```
p1 = MyClass()
print(p1.x)
```

Hàm __init__()

Để hiểu rõ hơn ý nghĩa của class ta cần hiểu về hàm __init__(). Mỗi classes đều có một hàm __init__(), hàm này sẽ được thực thi ngay khi class được bắt đầu.

Ta sử dụng hàm __init__() để gán gía trị, thuộc tính và các thao tác cần thiết khi đối tượng được tạo.

Ví dụ 0.9. Tạo một class Person, sử dụng hàm __inti__() để gán tên và tuổi:

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

p1 = Person("John", 36)

print(p1.name)  # John
print(p1.age)   # 36
```

Phương thức của đối tượng

Mỗi đối tượng có thể có các phương thức. Phương thức là những hàm thuộc về đối tượng ấy.

Ví dụ 0.10. Tạo phương thức in ra lời chào, và sử dụng nó trên đối tượng p1:

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def myfunc(self):
        print("Hello my name is " + self.name)

p1 = Person("John", 36)
p1.myfunc()
```

0.2 Một số thư viện của Python

Do hạn chế trong việc trình bày báo cáo nên em chỉ nêu một số ý chính của các thư viện và một số ví dụ. Các thao tác với thư viện đầy đủ hơn em đặt ở trong các file jupyter notebook của riêng từng thư viện.

0.2.1 Numpy

Numpy (viết tắt của "Numeric Python" hoặc "Numerical Python") là một thư viện lõi phục vụ cho khoa học máy tính của Python, hỗ trợ cho việc tính toán các mảng nhiều chiều, có kích thước lớn với các hàm đã được tối ưu áp dụng lên các mảng nhiều chiều đó. Numpy đặc biệt hữu ích khi thực hiện các hàm liên quan tới Đại Số Tuyến Tính.

Để cài đặt numpy, ta có 2 cách:

- Sử dụng Command Prompt:

```
pip install numpy
```

- Hoặc sử dụng Anaconda Prompt:

```
conda install numpy
```

- **Khai báo thư viện**:

```
import numpy as np
```

- **Khởi tạo mảng** : Numpy cho phép ta tạo các mảng 1 chiều hay nhiều chiều với nhiều cách thức khác nhau.

  Ví dụ 0.11. Một số cách tạo mảng:

```
# Tạo mảng 1 chiều
a = np.array([1, 2, 3, 4])   # a =  [1 2 3 4]

# Mảng 2 chiều(nhiều chiều)
b = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])   # b =  [[1 2 3 4]
                                             #       [5 6 7 8]]

# Tạo mảng với các giá trị chưa biết nhưng đã biết kích thước
d = np.zeros((3, 4))
```

- **Các thao tác với mảng** : Numpy cho phép ta thực hiện các thao tác với mảng như: thực hiện các phép toán giữa các mảng, giữa các giá trị trên mảng, index, slicing, điều khiển kích thước của mảng, ...

  Ví dụ 0.12. Một vài ví dụ:

```
A = np.array([[1, 1],
              [0, 1]])

B = np.array([[2, 0],
              [3, 4]])

arr1 = A * B          # Phép nhân theo từng phần tử
arr2 = A @ B          # Phép nhân ma trận. Cách khác: A.dot(B)
A *= 3                # Nhân mỗi phần tử của A với 3
A.sum(axis=1)         # Tổng các phần tử của A theo hàng
```

0.2.2 Pandas

Thư viện pandas trong python là một thư viện mã nguồn mở, hỗ trợ đắc lực trong thao tác dữ liệu. Đây cũng là bộ công cụ phân tích và xử lý dữ liệu mạnh mẽ của ngôn ngữ lập trình python. Thư viện này được sử dụng rộng rãi trong cả nghiên cứu lẫn phát triển các ứng dụng về khoa học dữ liệu. Thư viện này sử dụng một cấu trúc dữ liệu riêng là Dataframe. Pandas cung cấp rất nhiều chức năng xử lý và làm việc trên cấu trúc dữ liệu này.

Để cài đặt pandas, ta có 2 cách:

- Sử dụng Command Prompt:

```
pip install pandas
```

- Hoặc sử dụng Anaconda Prompt:

```
conda install pandas
```

- **Khai báo thư viện**:

```
import pandas as pd
```

- **Khởi tạo** Pandas có 2 cấu trúc dữ liệu cơ bản là: Series (1 chiều) và DataFrame (2 chiều)

  Có nhiều cách để tạo một cấu trúc dữ liệu trên pandas như: truyền trực tiếp giá trị, lấy từ mảng trên Numpy, từ file, ...

  Ví dụ 0.13. Một số cách tạo một cấu trúc dữ liệu trên pandas:

```
# Tạo một series
series = pd.Series([1, 2, 4, 5, 7, 9, np.nan])

# Tạo DataFrame bằng cách truyền giá trị từ cấu trúc dữ liệu dictionary trong python
df2 = pd.DataFrame({
    "A" : [1, 2, 1, 2],
    "B" : pd.Timestamp("20130102"),
    "C" : pd.Series(1, index=list(range(4)), dtype="float32"),
    "D" : np.array([3]*4, dtype='int32'),
    "E" : pd.Categorical(["test", "train", "validation", "test"]),
    "F" : "Foo"
})
```

- **Xem xét, lấy dữ liệu từ cấu trúc dữ liệu** Pandas cho phép ta thao tác một cách trực quan, đơn giản với các cấu trúc dữ liệu. Ta có in ra dữ liệu, thêm, xóa, thay đổi, thống kê, chọn từng phần dữ liệu,... Ta cũng có thể lưu các dạng cấu trúc dữ liệu trên pandas thành các định dạng file khác nhau như csv, hdf5, ...

  Ví dụ 0.14. Một số thao tác với dữ liệu:

```
df.head()              # Xem 10 hàng đầu tiên của dữ liệu
df.index               # Xem các tên chỉ số
df.columns             # Xem các cột
df.to_numpy()          # chuyển từ DataFrame sang mảng trong Numpy
df.describe()          # Hiển thị thống kê của dữ liệu
```

0.2.3 Matplotlib

Matplotlib là một thư viện của Python được sử dụng để vẽ đồ thị với sự trợ giúp của các thư viện khác như Numpy và Pandas. Nó được sử dụng để tạo ra các nhiễu thống kê(statistical interferences, xuất hiện khi hai phân phối xác xuất trùng lặp lên nhau) và vẽ đồ thị 2D của các mảng. Matplotlib sử dụng pyplot để cung cấp giao diện tương tự như MATLAB và là một mã nguồn mở. Để cài đặt matplotlib, ta có 2 cách:

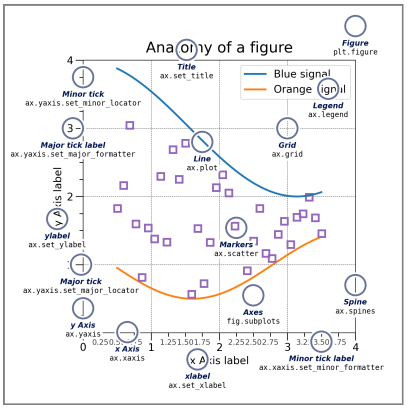
- Sử dụng Command Prompt:

```
pip install matplotlib
```

- Hoặc sử dụng Anaconda Prompt:

```
conda install matplotlib
```

Các thành phần trong đồ thị của Matplotlib:



Hình 0.1: Cấu trúc của đồ thị trong matplotlib

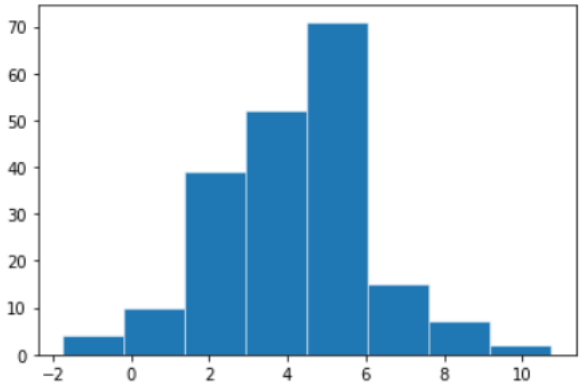
- **Khai báo thư viện**:

```
import numpy as np
```

Ví dụ 0.15. Một số dạng đồ thị:

```
# hist(x)
x = 4 + np.random.normal(0, 2, 200)
fig, ax = plt.subplots()
ax.hist(x, bins=8, linewidth=0.5, edgecolor='white')
```

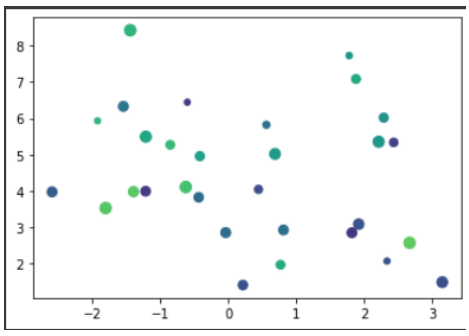
```
plt.show()
```



Hình 0.2: Đồ thị hist

Ví dụ 0.16. Một số dạng đồ thị:

```
# scatter(x, y)
x = np.random.normal(0, 2, 30)
y = 4 + np.random.normal(0, 2, 30)
# Chỉ định kích thước và màu sắc cho đồ thị được vẽ.
sizes = np.random.uniform(15, 80, 30)
colors =  np.random.uniform(15, 80, 30)
fig, ax = plt.subplots()
ax.scatter(x, y, s=sizes, c=colors, vmin=0, vmax=100)
plt.show()
```



Hình 0.3: Đồ thị scatter

0.2.4 Seaborn

Seaborn cũng là một thư viện của Python giúp vẽ các đồ thị với sự trợ giúp của Matplotlib, Numpy và Pandas. Nó có thể được coi là một phần mở rộng của một thư viện Matplotlib vì nó được xây dựng dựa trên đó. Seaborn giúp quan sát các biến dữ liệu dạng đơn trị và đa trị. Nó còn có các đồ thị rất trực quan và đẹp mắt.

Cài đặt và khai báo seaborn:

- Sử dụng Command Prompt:

```
pip install seaborn
```

- Hoặc sử dụng Anaconda Prompt:

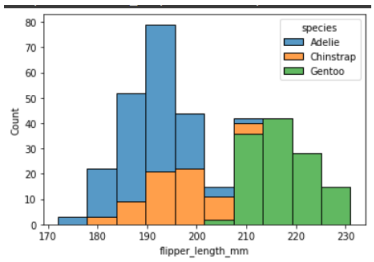
```
conda install seaborn
```

- Khai báo thư viện:

```
import seaborn as sns
```

Ví dụ 0.17. Ví dụ đồ thị:

```
penguins = sns.load_dataset('penguins')
sns.histplot(data=penguins, x='flipper_length_mm', hue='species', multiple='stack')
```



Hình 0.4: Đồ thị trong seaborn