



Python

Nội Dung

Chương 1: Giới thiệu những khái niệm cơ bản về Python	3
Chương 2: Mảng và các kiểu dữ liệu của Mảng	28
Chương 3: Modules và Package	58
Chương 4: Numpy	68
Chương 5: PANDAS	81
Chương 6: Web scraping	107
Chương 7: Ngoại lệ và I/O file	145
Chương 8: Matplotlib.....	177

Chương 1: Giới thiệu những khái niệm cơ bản về Python

Nội Dung

1.1 Chương trình đầu tiên	5
1.2 Biến và kiểu dữ liệu.....	7
1.3 Toán tử.....	<u>122</u>
1.4 Cấu trúc If- Else.....	18
1.5 Cấu trúc lặp.....	200
1.6 Bài tập chương 1	26

Giới thiệu chung

Python là một ngôn ngữ lập trình bậc cao cho các mục đích lập trình đa năng, do Guido van Rossum tạo ra và lần đầu ra mắt vào năm 1991. Python được thiết kế với ưu điểm mạnh là dễ đọc, dễ học và dễ nhớ. Python là ngôn ngữ có hình thức rất sáng sủa, cấu trúc rõ ràng, thuận tiện cho người mới học lập trình¹

Có 2 phiên bản Python đang được sử dụng phổ biến ở thời điểm hiện tại là Python 2 và Python 3. Về cơ bản, Python 2 và Python 3 có cách viết giống nhau, chỉ khác nhau ở một số thư viện hỗ trợ riêng cho Python 2 hoặc Python 3.

Có rất nhiều cách để viết chương trình Python như phần mềm:

1. Python IDE
2. Visual Studio Code
3. Jupyter notebook ← offline
4.

Các trình biên dịch online:

1. Datacamp
2. Google Colab ← online
3. w3schools
4. ...

Các ví dụ trong bài sử dụng Python 3 nền tảng online trên Google Colab và offline với phần mềm Jupyter Notebook để biên dịch.

¹ [https://vi.wikipedia.org/wiki/Python_\(ng%C3%B4n_ng%C3%BA_lập_trình\)](https://vi.wikipedia.org/wiki/Python_(ng%C3%B4n_ng%C3%BA_lập_trình))

1. 1 Chương trình đầu tiên, hello world!

Chương trình đầu tiên khi làm quen với Python là sử dụng lệnh ***print*** in một chuỗi ký tự ra màn hình.

Lệnh ***print*** có cú pháp như sau:

```
print(object(s), sep=separator, end=end, file=file, flush=flush)
```

Trong đó:

<i>object(s)</i>	Bất kỳ đối tượng nào, số lượng tùy ý. Nội dung bên trong sẽ được chuyển sang định dạng chuỗi trước khi in ra màn hình.
<i>sep='separator'</i>	Không bắt buộc. Định nghĩa cách tách đối tượng nếu có nhiều hơn một đối tượng được in. Mặc định là ‘ ’ (khoảng cách)
<i>end='end'</i>	Không bắt buộc. Chỉ ra ký tự được in khi kết thúc. Mặc định là ‘\n’ (xuống dòng)
<i>file</i>	Không bắt buộc. Một đối tượng với một phương thức ghi. Mặc định là sys.stdout
<i>flush</i>	Không bắt buộc. Một Boolean, chỉ định xem nó ra được giải phóng (True) hay được lưu vào bộ đệm (False). Mặc định là ‘False’

Sinh viên soạn thảo câu lệnh sau trong Python:

```
print("hello world")
```

Chạy chương trình ta được kết quả:

```
hello world
```

Lệnh ***print*** trong Python sử dụng định dạng chuỗi tương tự như ngôn ngữ C. Các định dạng cơ bản bao gồm:

- %s – Định dạng chuỗi ký tự (String)
- %d – Định dạng số nguyên (Integer)
- %f – Định dạng số thực (Float)
- %.<number of digits>f – Định dạng số thực với số lượng ký tự số sau dấu chấm thập phân cố định.
- %x/%X – Định dạng số Hex (%x: Dạng hex viết thường/ %X: Dạng HEX viết hoa)

Ví dụ minh họa:

```
# This prints out "Hello, John!"  
  
name = "John"  
  
print("Hello, %s!" % name)  
  
Hello, John!
```

```
# This prints out "John is 23 years old."  
  
name = "John"  
  
age = 23  
  
print("%s is %d years old." % (name, age))  
  
John is 23 years old.
```

```
a = 20.0  
  
b = 3.0  
  
c = a/b  
  
print("The result is: %d" % (c))  
  
print("The result is: %f" % (c))  
  
print("The result is: %.3f" % (c))  
  
The result is: 6  
  
The result is: 6.666667  
  
The result is: 6.667
```

1.2 Biến và kiểu dữ liệu

Python hoàn toàn là hướng đối tượng. Các biến trong Python không cần khai báo trước khi sử dụng hoặc khai báo kiểu biến như trong các ngôn ngữ khác. Mọi biến trong Python là một đối tượng.

Số nguyên:

```
myint = 7  
print(myint)
```

7

Số thực:

```
myfloat = 7.0  
print(myfloat)  
  
myfloat = float(7)  
print(myfloat)
```

7.0

7.0

Chuỗi:

Có thể khai báo trong dấu nháy đơn hoặc dấu nháy kép:

```
mystring = 'hello'  
print(mystring)  
  
mystring = "hello"  
print(mystring)
```

hello

hello

Khi muốn khai báo dấu đơn hoặc dấu nháy kép trong đoạn văn, thì toàn bộ nội dung sẽ nằm trong dấu nháy đơn/kép còn lại:

```
mystring = 'Don"t worry about apostrophes'
print(mystring)

mystring = "Don't worry about apostrophes"
print(mystring)
```

```
Don"t worry about apostrophes
Don't worry about apostrophes
```

+ Các toán tử đơn giản có thể thực hiện trên số và chuỗi:

```
one = 1

two = 2

three = one + two

print(three)

hello = "hello"

world = "world"

helloworld = hello + " " + world

print(helloworld)
```

```
3
hello world
```

+ Phép gán có thể thực hiện với nhiều biến lần lượt như sau:

```
a, b = 3, 4

print(a,b)
```

```
3 4
```

Không thể thực hiện ghép toán trên cả số và chuỗi. Ví dụ:

```
# This will not work!

one = 1

two = 2

hello = "hello"

print(one + two + hello)
```

```
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Các phép toán trên chuỗi

+ Đếm số ký tự trong chuỗi:

```
astring = "Hello world!"

print(len(astring))
```

```
12
```

+ Xác định vị trí ký tự trong chuỗi

```
astring = "Hello world!"

print(astring.index("o"))
```

```
4
```

Kết quả trả về là 4 vì thứ tự chuỗi trong Python bắt đầu từ 0 và hàm này chỉ trả về một kết quả đầu tiên.

+ Đếm số ký tự trong chuỗi:

```
astring = "Hello world!"

print(astring.count("l"))
```

```
3
```

Lệnh đếm ký tự “l” trong chuỗi “Hello world!”. Kết quả trả về là 3, tức là có 3 ký tự “l” trong chuỗi.

+ Cắt chuỗi tại vị trí cho trước

```
astring = "Hello world!"  
print(astring[3:7])  
  
lo w
```

Lệnh này sẽ cắt chuỗi tại vị trí đầu tiên (3) đến vị trí cuối -1 (6)

```
astring = "Hello world!"  
print(astring[3:7])  
print(astring[3:7:1])  
print(astring[3:7:2])  
  
lo w  
lo w  
l
```

Lệnh [3:7:2] có nghĩa là lấy từ 3 đến 7, cách 2 bước. *[start:stop:step]*.

Index âm/ index ngược (negative indexing)

Python cho phép chỉ định index số âm trong 1 danh sách, tức là đi chiều ngược từ phải qua trái với từng index trong list.

Ví dụ.

```
data = ('a', 'b', 'c')  
print(data[-1])  
print(data[0])  
print(data[1])  
  
c  
a  
b
```

+Tạo chuỗi ngược.

```
astring = "Hello world!"
print(astring[::-1])
!dlrow olleH
```

+ Chuyển đổi ký tự thành ký tự in hoa/ ký tự thường

```
astring = "Hello world!"
print(astring.upper())
print(astring.lower())
HELLO WORLD!
hello world!
```

+ Kiểm tra bắt đầu và kết thúc chuỗi có đúng “từ khóa” không

```
astring = "Hello world!"
print(astring.startswith("Hello"))
print(astring.endswith("asdfasdfsdf"))
True
False
```

+ chia chuỗi theo điều kiện.

```
astring = "Hello world!"
afewwords = astring.split(" ")
print(afewwords)
['Hello', 'world!']
```

1.3 Toán tử

Giống như bất kỳ trình biên dịch khác, Python cũng có các phép toán cộng, trừ, nhân, chia. Thứ tự thực hiện phép toán cũng giống như các phép toán thông thường thực hiện.

+ Toán tử

Toán tử (Operator)	Tên phép toán	Ví dụ
+	Cộng	$x + y$
-	Trừ	$x - y$
*	Nhân	$x * y$
/	Chia	x / y
%	Chia lấy dư	$x \% y$
**	Lấy lũy thừa	$x ** y$
//	Chia lấy phần nguyên	$x // y$

+ Toán tử gán

Toán tử gán (Assignment Operators)	Ví dụ	Ý nghĩa
=	$x = 10$	Gán giá trị 10 vào biến x
+=	$x += 3$	$x = x + 3$
-=	$x -= 3$	$x = x - 3$
*=	$x *= 3$	$x = x * 3$
/=	$x /= 3$	$x = x / 3$
%=	$x \%= 3$	$x = x \% 3$
//=	$x //= 3$	$x = x // 3$
**=	$x **= 3$	$x = x ** 3$
&=	$x \&= 3$	$x = x \& 3$

$ =$	$x = 3$	$x = x 3$
$^=$	$x ^= 3$	$x = x ^ 3$
$>>=$	$x >>= 3$	$x = x >> 3$
$<<=$	$x <<= 3$	$x = x << 3$

+ Toán tử thao tác bit (Bitwise Operators):

Toán tử	Ý nghĩa	Ý nghĩa
$\&$	AND	Lệnh AND các bit
$ $	OR	Lệnh OR các bit
$^$	XOR	Lệnh XOR các bit
\sim	NOT	Đảo bit
$<<$	Dịch sang trái	Dịch trái, thêm zero vào bên phải
$>>$	Dịch sang phải	Dịch phải, thêm giá trị giống bit ngoài cùng bên trái vào bên trái

Ví dụ

```
number = 1 + 2 * 3 / 4.0
print(number)
```

2.5

```
remainder = 11 % 3
print(remainder)
```

2

```
squared = 7 ** 2
cubed = 2 ** 3
print(squared)
```

```
print(cubed)
```

49

8

```
helloworld = "hello" + " " + "world"
```

```
print(helloworld)
```

hello world

```
lotsofhellos = "hello" * 10
```

```
print(lotsofhellos)
```

hellohellohellohellohellohellohellohellohellohello

+ Toán tử điều kiện

Toán tử điều kiện	Ý nghĩa	Ví dụ
$==$	Bằng	$x == y$
$!=$	Không bằng	$x != y$
$>$	Lớn hơn	$x > y$
$<$	Nhỏ hơn	$x < y$
\geq	Lớn hơn hoặc bằng	$x \geq y$
\leq	Nhỏ hơn hoặc bằng	$x \leq y$

Python sử dụng các biến boolean để đánh giá các điều kiện. Các giá trị boolean sẽ là True/False được trả về khi một biểu thức được so sánh hoặc đánh giá. Ví dụ:

```
x = 2

print(x == 2) # prints out True

print(x == 3) # prints out False
```

```
print(x < 3) # prints out True
True
False
True
```

Lưu ý rằng việc gán biến được thực hiện bằng toán tử dấu bằng đơn "=", trong khi việc so sánh giữa hai biến được thực hiện bằng toán tử dấu bằng kép "==" . Toán tử "không bằng" được đánh dấu là "!=".

+ Toán tử logic (Logical Operators)

Toán tử logic	Ý nghĩa	Ví dụ
and	Trả về True nếu cả 2 mệnh đề là đúng	x < 5 and x < 10
or	Trả về True nếu một trong 2 mệnh đề là đúng	x < 5 or x < 4
not	Trả về False nếu mệnh đề đúng	not(x < 5 and x < 10)

Ví dụ minh họa

```
name = "John"
age = 23

if name == "John" and age == 23:
    print("Your name is John, and you are also 23 years old.")

if name == "John" or name == "Rick":
    print("Your name is either John or Rick.")

Your name is John, and you are also 23 years old.
Your name is either John or Rick.
```

```
name = "John"

if name in ["John", "Rick"]:
    print("Your name is either John or Rick.")
```

```
Your name is either John or Rick.
```

```
print(not False) # Prints out True
print((not False) == (False)) # Prints out False
True
False
```

- + Toán tử định danh (Identity Operators) được dùng để xác định xem hai biến có đang trỏ tới cùng một đối tượng hay không. Với các kiểu dữ liệu như int, str, float,... thì toán tử này tương đương với toán tử `==`

Toán tử định danh	Ý nghĩa	Ví dụ
<code>is</code>	Trả về True nếu cả 2 cùng đối tượng	<code>x is y</code>
<code>is not</code>	Trả về True nếu 2 biến không cùng đối tượng	<code>x is not y</code>

```
a = 6
b = 6
print(a is b)
True
statement = False
another_statement = True
if statement is True:
    print("do something")
elif another_statement is True: # else if
    print("do something else")
else:
```

```
print("do another thing")
```

```
do something else
```

+Toán tử Membership (Membership Operators)

in	Trả về True nếu một chuỗi với giá trị được chỉ định có trong đối tượng	x in y
not in	Trả về True nếu một chuỗi với giá trị được chỉ định không có trong đối tượng	x not in y

Python sử dụng dấu “:” (2 chấm) và các thụt đầu dòng cho các khối thay vì sử dụng dấu ngoặc nhọn như trong C. Có thể sử dụng Tab hoặc Space đều được. Lưu ý phải thống nhất các thụt đầu dòng cho toàn bộ chương trình. Ví dụ:

```
x = 1
if x == 1:
    # indented two spaces
    print("x is 1.")
```

```
x is 1.
```

```
x = 1
if x == 1:
    # indented one tab
    print("x is 1.")
```

```
x is 1.
```

```
x = 1
```

```
if x == 1:  
    # indented four spaces  
    print("x is 1.")
```

```
x is 1.
```

1.4 Câu trúc if - else

Câu trúc chỉ có if:

```
if(điều kiện):
```

Các câu lệnh nếu điều kiện đúng

...

Lệnh này không nằm trong if

...

Câu trúc if-else

```
if(điều kiện):
```

Các câu lệnh nếu điều kiện đúng

...

```
else:
```

Các câu lệnh nếu điều kiện không đúng

...

Lệnh này không nằm trong if-else

...

Cấu trúc if-elif-else

```
if(điều kiện 1):
```

Các câu lệnh nếu điều kiện 1 đúng

...

```
elif(điều kiện 2):
```

Các câu lệnh nếu điều kiện 2 đúng

...

```
else:
```

Các câu lệnh nếu các điều kiện trên không đúng

...

Lệnh này không nằm trong if-elif-else

Python hỗ trợ các điều kiện logic thông thường từ toán học:

- Bằng: $a == b$
- Không bằng: $a != b$
- Nhỏ hơn: $a < b$
- Nhỏ hơn hoặc bằng: $a <= b$
- Lớn hơn: $a > b$
- Lớn hơn hoặc bằng: $a >= b$

Các điều kiện này có thể được sử dụng theo một số cách, phổ biến nhất là trong "câu lệnh if" và vòng lặp.

Một "câu lệnh if" được viết bằng cách sử dụng từ khóa if.

```
a = 33
b = 200
if b > a:
    print("b is greater than a")
```

b is greater than a

+elif

Tùy khóa elif nghĩa là "nếu điều kiện trước đó không đúng, thì hãy thử điều kiện này".

```
a = 33
b = 33
if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
a and b are equal
```

+else: các điều kiện còn lại

```
x = 2
if x == 2:
    print("x equals two!")
else:
    print("x does not equal to two.")
x equals two!
```

1.5 Cấu trúc lặp

Cú pháp:

+ Vòng lặp for:

Vòng lặp **for** được sử dụng để lặp qua một chuỗi liên tục (đó là một danh sách, một bộ, một từ điển, một tập hợp hoặc một chuỗi). Cú pháp:

```
for [biến lặp lại] in [Mảng]:
```

Các câu lệnh

Ví dụ có một mảng primes gồm 4 phần tử, thực hiện vòng lặp for in lần lượt các phần tử trong mảng.

```
primes = [2, 3, 5, 7]
for prime in primes:
    print(prime)
```

```
2
3
5
7
```

Vòng lặp for không yêu cầu đặt trước biến lập chỉ mục.

```
# Prints out the numbers 0,1,2,3,4
for x in range(5):
    print(x)

# Prints out 3,4,5
for x in range(3, 6):
    print(x)

# Prints out 3,5,7
for x in range(3, 8, 2):
    print(x)
```

```
0
1
2
3
4
3
4
5
3
```

5

7

+ Lặp qua một chuỗi

```
for x in "banana":  
    print(x)
```

b
a
n
a
n
a

+ Vòng lặp while:

Cú pháp:

While [điều kiện]:

Lệnh A

Lệnh B

```
# Prints out 0,1,2,3,4  
  
count = 0  
  
while count < 5:  
    print(count)  
  
    count += 1 # This is the same as count = count + 1
```

0

```
1  
2  
3  
4
```

+ Break và continue:

```
# Prints out 0,1,2,3,4  
  
count = 0  
  
while True:  
  
    print(count)  
  
    count += 1  
  
    if count >= 5:  
  
        break  
  
  
# Prints out only odd numbers - 1,3,5,7,9  
  
for x in range(10):  
  
    # Check if x is even  
  
    if x % 2 == 0:  
  
        continue  
  
    print(x)
```

```
0  
1  
2  
3  
4  
1
```

```
3
5
7
9
```

Với câu lệnh break, chúng ta có thể dừng vòng lặp trước khi nó lặp qua tất cả các mục:

Với câu lệnh continue, chúng ta có thể dừng lần lặp hiện tại của vòng lặp và tiếp tục với phần tiếp theo:

+Từ khóa else ở vòng lặp chỉ định một khối mã sẽ được thực thi khi vòng lặp kết thúc

```
# Prints out 0,1,2,3,4 and then it prints "count value reached 5"

count=0

while(count<5) :

    print(count)

    count +=1

else:

    print("count value reached %d" %(count))
```

```
0
1
2
3
4
count value reached 5
```

```
# Prints out 1,2,3,4

for i in range(1, 10):

    if(i%5==0):
```

```
        break  
  
    print(i)  
  
else:  
  
    print("this is not printed because for loop is terminated because of brea  
k but not due to fail in condition")
```

```
1  
2  
3  
4
```

```
# Prints out 1,2,3,4  
  
for i in range(1, 5):  
  
    print(i)  
  
else:  
  
    print("this line will be printed")
```

```
1  
2  
3  
4  
  
this line will be printed
```

Bài tập Chương 1

Bài tập phần 1

1. Sử dụng lệnh print, in dòng chữ “Good morning” lên màn hình.
2. Tạo các biến với giá trị là tên SV và MSSV. Sử dụng lệnh print, in tên và MSSV lên màn hình.
3. Tạo ra các biến dạng số nguyên, thực hiện các phép toán ‘+’, ‘-’, ‘*’, ‘/’ và in các kết quả đó ra màn hình.

Bài tập phần 2

4. Viết chương trình tính chu vi và diện tích hình tròn. Với bán kính hình tròn cho trước trong biến. Tính và in kết quả ra màn hình.

5. Viết chương trình Python in ra giá trị họ tên ngược lại giống như sau:

- Data = ('họ', 'tên đệm', 'tên')
- In ra màn hình kết quả: 'tên' 'tên đệm' 'họ'

Bài tập phần 3

6. Viết chương trình Python tính khoảng cách từ điểm (x1,y1) đến điểm (x2,y2)

7. Viết chương trình Python tính tổng các phần tử số trong mảng cho trước gồm cả ký tự và số.

8. Viết công thức tính $n!$ và in kết quả ra màn hình. Giá trị n cho trước trong một biến.

Bài tập phần 4

9. Viết chương trình tìm nghiệm của phương trình bậc 2. Các hệ số của phương trình bậc 2 được khai báo trước trong biến. Tính và in kết quả ra màn hình.

10. Viết chương trình Python tính toán số ngày dựa trên ngày tháng năm

Ngày tháng cho : (01, 12, 2020), (12, 12, 2020)

Kết quả mong muốn : 12 ngày

11. Viết chương trình Python in ra màn hình số cho trước là số chẵn hay số lẻ.

Bài tập phần 5

12. Viết chương trình Python chỉ in ra màn hình số chẵn trong một chuỗi cho trước.

13. Viết chương trình Python sắp xếp thứ tự các phần tử trong mảng theo thứ tự tăng dần
14. Viết chương trình Python sắp xếp thứ tự các phần tử trong mảng theo thứ tự giảm dần
15. Viết chương trình Python tìm số lớn nhất trong một dãy số cho trước.
16. Viết chương trình Python tìm số nhỏ nhất trong một dãy số cho trước
17. Viết chương trình Python tìm số lớn thứ 2 trong một dãy số cho trước
18. Viết chương trình Python tìm số nhỏ thứ 2 trong một dãy số cho trước.
19. Viết chương trình Python in ra các ký tự khác nhau trong một chuỗi ký tự cho trước. Cho biết các ký tự được lặp lại bao nhiêu lần.
20. Viết chương trình Python tính tổng các số có trong mảng với điều kiện số đó nhỏ hơn một số cho trước.

Mảng cho trước : (2,3,4,5,6,7)

Điều kiện: 5

Kết quả mong muốn: $2+3+4 = 9$

21. Cho trước số nguyên N. In ra màn hình các số nguyên tố nhỏ hơn N.
22. Cho trước số nguyên N và M. Tính tổng và in ra màn hình các số nguyên tố lớn hơn N và nhỏ hơn M
23. Cho trước số nguyên N và M. Tìm các giá trị bội chung nhỏ nhất và ước chung lớn nhất . In 2 kết quả tìm được ra màn hình.

Chương 2: Mảng và các kiểu dữ liệu của Mảng

Nội Dung

2.1 List (Liệt kê trong Python)	31
2.1.1 Giới thiệu List.....	31
2.1.2 Tạo list.....	31
2.1.2.1 Khai báo và gán giá trị các phần tử bằng phép gán	31
2.1.2.2 Tạo list bằng cách nối các list với nhau	32
2.1.2.3 Tạo list bằng cách lặp lại list đã có	32
2.1.2.4 Tạo list bằng cách copy các phần tử từ vị trí index tới cuối	33
2.1.3 Truy xuất phần tử trong list.....	33
2.1.4 Cập nhật giá trị cho phần tử trong list	34
2.1.5 Vòng lặp trong List.....	35
2.1.6 Các phương thức cơ bản của List	35
2.1.6.1 Lấy chiều dài số phần tử trong list.....	35
2.1.6.2 Thêm phần tử vào cuối list.....	35
2.1.6.3 Chèn thêm vào trước thành phần nào đó trong list	35
2.1.6.4 Xóa một phần tử trong list (del, remove, pop).....	36
2.1.6.5 Đảo ngược list	37
2.1.6.6 Sắp xếp list	37
2.1.7 Tổng hợp các hàm dùng trong list.....	39
2.2 Tuples trong Python	40
2.2.1 Giới thiệu Tuples	40
2.2.2 Tạo một Tuples.....	40

2.2.3 Truy xuất các thành phần trong Tuples	40
2.2.3.1 Truy xuất thành phần dùng ngoặc vuông có chỉ số.....	40
2.2.3.2 Truy xuất dùng chỉ số âm.....	40
2.2.3.3 Dùng khoảng chỉ số.....	40
2.2.3.4 Dùng khoảng chỉ số âm.....	41
2.2.4 Vòng lặp của Tuples	41
2.2.5 Các phương thức cơ bản trong Tuples.....	41
2.2.5.1 Thay đổi giá trị trong Tuples.....	41
2.2.5.2 Đo chiều dài của Tuples.....	41
2.2.5.3 Thêm giá trị cho Tuples	41
2.2.5.4 Tạo Tuples với 1 thành phần.....	42
2.2.5.5 Nhập hai tuples.....	42
2.2.6 Tổng hợp các hàm trong Tuples	42
2.3 Dictionary (từ điển trong Python)	43
2.3.1 Giới thiệu Dictionary.....	43
2.3.2 Tạo Dictionary.....	43
2.3.4 Cập nhật các giá trị phần tử trong dictionary	44
2.3.5 Vòng lặp trong Dictionary	44
2.3.6 Các phương thức cơ bản trong Dictionary	44
2.3.6.1 Kiểm tra xem một key name có tồn tại không?	44
2.3.6.2 Xác định thành phần trong Dict bằng hàm len()	45
2.3.6.3 Thêm thành phần vào Dictionary.....	45
2.3.6.4 Xóa những thành phần trong Dictionary (pop,del,clear)	45

2.3.6.5 Copy và xây dựng một bản sao từ Dictionary có sẵn	46
2.3.7 Tổng hợp các hàm dùng kèm với Dictionary	49
2.4 Set trong Python.....	50
2.4.1 Giới thiệu Set trong Python.....	50
2.4.2 Cách tạo một Set.....	50
2.4.3 Truy xuất thành phần trong Set	50
2.4.4 Các phương thức cơ bản của Set	50
2.4.4.1 Thêm 1 thành phần vào Set.....	50
2.4.4.2 Thêm nhiều thành phần vào Set.....	51
2.4.4.3 Đo chiều dài của set	51
2.4.4.4 Thêm bớt thành phần trong set (remove, discard, clear).....	51
2.4.4.5 Gia nhập 2 sets	51
2.4.5 Tổng hợp các hàm dùng với Set.....	52
Bài Tập Chương 2.....	54

A) Giới thiệu chung

Có 4 loại Tổng Hợp loại dữ liệu trong ngôn ngữ lập trình Python là:

- List: đây là loại được tạo nên và thay đổi được. Cho phép lặp lại các thành viên trong đó
- Tuple: đây là loại được tạo nên và không thay đổi được. Cho phép lặp lại các thành viên
- Set: là tổng hợp không được tạo nên và không thay đổi được. Không được phép lặp lại các thành viên.
- Dictionary: là tổng hợp không được tạo nên, thay đổi được và có mục chỉ dẫn (mục lục). Không lặp lại các thành viên.

Khi chọn loại tổng hợp, cần phải hiểu biết những đặc tính của loại đó. Chọn đúng bộ dữ liệu đặc thù có ý nghĩa quan trọng làm tang hay giảm hiệu suất và bảo mật.

B) Các kiểu dữ liệu mảng trong Python

2.1 List (Liệt Kê trong Python)

2.1.1 Giới thiệu List

List là loại tổng hợp được tạo nên bởi người dùng và thay đổi được. Trong Python, list được viết trong những ngoặc vuông.

Là một cấu trúc dữ liệu cơ bản trong Python, thuộc nhóm sequence.

Gồm nhiều phần tử, mỗi phần tử có một vị trí gọi là index, phần tử đầu tiên có index = 0, phần tử cuối cùng có index = số phần tử - 1

Các phần tử trong list có thể cùng hoặc khác kiểu. Tuy nhiên, cùng kiểu thì dễ xử lý tính toán.

2.1.2 Tạo list

2.1.2.1 Khai báo và gán giá trị các phần tử trong list bằng phép gán

Cách tạo list bằng phép gán “=”

Ví dụ về tạo list

```
Thislist = ["apple", "banana", "cherry"]
print(thislist)
```

2.1.2.2 Tạo list bằng cách nối các list với nhau

NHẬP 2 list lại làm 1 dùng toán “+”

```
list1 = ["a", "b", "c"]
list2 = [1, 2, 3]
```

```
list3 = list1 + list2
print(list3)
```

Dùng hàm extend để nối 2 list với nhau

```
list1 = ["a", "b", "c"]
list2 = [1, 2, 3]
```

```
list1.extend(list2)
print(list1)
```

2.1.2.3 Tạo list bằng cách lặp lại list đã có

Tạo list bằng cách copy list có sẵn

Dùng hàm copy()

```
thislist = ["apple", "banana", "cherry"]
mylist = thislist.copy()
print(mylist)
```

Dùng lệnh list()

```
thislist = ["apple", "banana", "cherry"]
mylist = list(thislist)
print(mylist)
```

Lưu ý lệnh `list()` có thể dùng tạo kiểu dữ liệu mới

```
thislist = list(("apple", "banana", "cherry"))

# note the double round-brackets

print(thislist)
```

Lưu ý nếu dùng cách này tạo kiểu dữ liệu thì có hai dấu ngoặc đơn ((

2.1.2.4 Tạo list bằng cách copy phần tử từ vị trí index tới cuối

Để copy từ vị trí index tới cuối ta dùng toán tử slicing như sau

```
# mixed list

list = ['cat', 0, 6.7]

# copying a list using slicing

new_list = list[:]
```

2.1.3 Truy xuất phần tử trong list

Muốn truy cập sử dụng dữ liệu tương ứng thì:

```
thislist = ["apple", "banana", "cherry"]

print(thislist[1])
```

Truy xuất phần tử có thể dùng chỉ số âm

Chỉ Số Âm: nghĩa là bắt đầu đếm ngược từ phần cuối của list

“-1” nghĩa là phần tử vị trí cuối cùng của List

“-2” nghĩa là phần tử kế cuối

```
thislist = ["apple", "banana", "cherry"]
print(thislist[-1])
```

Truy Xuất Một Phần Của List: bạn có thể chỉ ra một khoảng của List bằng cách dùng dấu “:”. Ví dụ:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:5])
```

Nghĩa là in ra màn hình các thành phần trong khoảng từ 2 tới 4. LUU Ý vị trí số 5 sẽ không xuất ra. VỊ TRÍ ĐẦU TIÊN CÓ CHỈ SỐ 0

Truy xuất từ vị trí đầu tiên tới khoảng nào đó:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[:4])
```

Truy xuất từ vị trí nào đó tới cuối

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:])
```

Truy xuất dùng khoảng chỉ số âm

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[-4:-1])
```

2.1.4 Cập nhật giá trị cho phần tử trong list

Thay đổi 1 giá trị thành phần nào đó trong list

```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)
```

2.1.5 Vòng lặp trong List

Bạn có thể dùng vòng lặp for trong LIST như để in hết từng thành phần từng bước 1

```
thislist = ["apple", "banana", "cherry"]
for x in thislist:
    print(x)
```

Dùng vòng lặp kiểm tra có 1 thành phần nào đó trong list hay không

```
thislist = ["apple", "banana", "cherry"]
if "apple" in thislist:
    print("Yes, 'apple' is in the fruits list")
```

để kiểm tra có thành phần apple trong list hay không?

2.1.6 Các phương thức cơ bản của list

2.1.6.1 Lấy chiều dài số phần tử trong list

Dùng hàm len()

```
thislist = ["apple", "banana", "cherry"]
print(len(thislist))
```

2. 1.6.2 Thêm phần tử vào cuối list

Thêm thành phần dùng append(), chèn vào vị trí cuối cùng

```
thislist = ["apple", "banana", "cherry"]
thislist.append("orange")
print(thislist)
```

2.1.6.3 Chèn thêm vào trước thành phần nào đó trong List

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(1, "orange")
print(thislist)
```

Trong ví dụ chèn thêm “orange” phía trước vị trí số 1 (chính là “banana”)

2.1.6.4 Xóa một phần tử trong list (del, remove, pop)

Dùng lệnh remove

```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
```

Dùng lệnh pop()

```
thislist = ["apple", "banana", "cherry"]
thislist.pop(2)
print(thislist)
```

Lưu ý pop() để remove chỉ số đặc biệt (để trống là vị trí cuối cùng)

Dùng del để bớt thành phần đặc biệt chỉ định

```
thislist = ["apple", "banana", "cherry"]
del thislist[0]
print(thislist)
```

Dùng lệnh del để xóa hết các thành phần trong list

```
thislist = ["apple", "banana", "cherry"]
del thislist
```

Xoá hết clear() toàn bộ list

```
thislist = ["apple", "banana", "cherry"]
thislist.clear()
print(thislist)
```

2.1.6.5 Đảo ngược list

```
list01 = ["apple", "banana", "cherry"]
print('list01 is', list01)

list01.reverse()

print('Reverse of list01 is', list01)
```

2.1.6.6 Sắp xếp list

Phương thức này có tác dụng sắp xếp lại các phần tử trong list theo một thứ tự xác định.

Cú pháp:

```
mylist.sort(reverse, key)
```

Trong đó:

- **mylist** là list mà bạn muốn sắp xếp.
- **reverse** là một boolean cấu hình kiểu sắp xếp. Nếu **reverse = True** thì list sẽ được sắp xếp từ lớn đến bé, nếu **reverse = False** thì list sẽ được sắp xếp theo thứ tự từ bé đến lớn. Mặc định thì **reverse = False**.
- **key** là **callback def** để xử lý list hoặc là một **lambda function** (thường được dùng để sắp xếp các list **tuple** hoặc **dictionary**).

Ví dụ:

```
list = ['A', 'C', 'B', 'E', 'D']

list.sort()

print(list)

# Kết quả: ['A', 'B', 'C', 'D', 'E']

list.sort(reverse=True)

print(list)

# Kết quả: ['E', 'D', 'C', 'B', 'A']

def custom_sort(elem):

    return elem[1]

list = [(1, 2), (5, 7), (7, 100), (4, 4)]

list.sort(key=custom_sort)

print(list)

# Kết quả: [(1, 2), (4, 4), (5, 7), (7, 100)]
```

2.1.7 Tổng hợp các hàm dùng trong LIST

Phương pháp	Mô tả cách sử dụng
append()	Thêm một thành phần tại cuối của list
clear()	Bỏ hết hoàn toàn thành phần trong list. Clear trống.
copy()	Trả về bản sao của list
count()	Trả về số các thành phần có trong list với giá trị chỉ định
extend()	Thêm nhiều thành phần vào cuối list hiện tại với giá trị chỉ định
index()	Trả về chỉ số đầu tiên với giá trị chỉ định
insert()	Thêm một thành phần trước vị trí chỉ định
pop()	Xoá bớt thành phần tại vị trí chỉ định
remove()	Xoá tại vị trí chỉ định
reverse()	Đảo thứ tự trong list
sort()	Xắp xếp theo trình tự nào đó trong list

2.2 Tuples trong Python

2.2.1 Giới thiệu Tuples

Tuple là kiểu tập hợp được định sẵn và không thay đổi được. Tuple sử dụng dấu ngoặc đơn

2.2.2 Tạo một Tuples

Tạo một Tuples dùng phép gán

```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

Dùng hàm tuple()

```
thistuple = tuple(("apple", "banana", "cherry")) # note the double
round-brackets
print(thistuple)

["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
```

2.2.3 Truy xuất các thành phần trong Tuples

2.2.3.1 Truy xuất thành phần dùng ngoặc vuông có chỉ số

Bạn có thể truy xuất các thành phần của tuple bằng cách dùng chỉ số trong ngoặc vuông.

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

Nghĩa là print thành phần thứ 2 trong tuple

2.2.3.2 Truy xuất dùng chỉ số âm

Có thể dùng chỉ số âm -1 chỉ vị trí cuối cùng hoặc -2 chỉ vị trí kế cuối

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[-1])
```

2.2.3.3 Dùng khoảng chỉ số

Có thể dùng khoảng 2:5 để truy xuất như sau:

```
thistuple =
("apple", "banana", "cherry", "orange", "kiwi", "melon", "mang
o")
print(thistuple[2:5])
```

Trả về vị trí thứ 3, 4 và 5.

LUU Ý: bắt đầu từ vị trí 2 tới vị trí 4 (không bao gồm vị trí 5)

2.2.3.4 Dùng khoảng chỉ số âm

```
thistuple =
("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(thistuple[-4:-1])
```

Trả về bao gồm vị trí -4 và không bao gồm vị trí -1.

2.2.4 Vòng lặp của Tuples

Tích hợp với thành phần để in ra chuỗi giá trị tương ứng

```
thistuple = ("apple", "banana", "cherry")
for x in thistuple:
    print(x)
```

Phần hàm for sẽ được học trong Chương vòng lặp của Python

2.2.5 Các phương thức cơ bản trong Tuples

2.2.5.1 Thay đổi giá trị của Tuples

```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)
print(x)
```

Vì giá trị của Tuple không thể thay đổi nên phải đổi kiểu sang List và lưu vào Y. Sau đó đổi giá trị trong list Y và từ đó gán đổi kiểu Tuple ngược lại vào X.

2.2.5.2 Đo chiều dài của Tuples

Để xác định chiều dài của Tuple sử dụng len() như sau:

```
thistuple = ("apple", "banana", "cherry")
print(len(thistuple))
```

2.2.5.3 Thêm giá trị cho Tuples

Thêm giá trị cho tuples là không thể một khi Tuple được tạo

```

thistuple = ("apple", "banana", "cherry")
thistuple[3] = "orange" # This will raise an error
print(thistuple)

```

phần code này sẽ đưa ra lỗi.

2.2.5.4 Tạo tuple với 1 thành phần, lưu ý dấu phẩy

```

thistuple = ("apple",)
print(type(thistuple))
#NOT a tuple
thistuple = ("apple")
print(type(thistuple))

```

Xoá thành phần trong tuple: là KHÔNG THỂ

Nhưng có thể xoá toàn bộ tuple

```

thistuple = ("apple", "banana", "cherry")
del thistuple
print(thistuple) #this will raise an error because the
tuple no longer exists

```

2.2.5.5 Nhập hai tuples

```

tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)
tuple3 = tuple1 + tuple2
print(tuple3)

```

2.2.6. Tổng hợp các hàm dùng trong Tuple

Phương pháp	Mô tả
count()	Trả về số lần của một giá trị đặc biệt trong Tuple
index()	Search trong tuple cho giá trị chỉ định và trả về vị trí nơi được tìm thấy

2.3 Dictionary (từ điển trong Python)

2.3.1 Giới thiệu Dictionary

Dictionaries là tập các thành phần chưa được set up sẵn, có thể thay đổi và chỉ số truy cập.

Trong Python dictionaries dùng dấu ngoặc nhọn, và có những quy định cho dữ liệu chữ và giá trị số.

2.3.2 Tạo Dictionary

Dùng phép gán để tạo dữ liệu kiểu dictionary

Ví dụ:

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

2.3.3 Truy xuất các phần tử trong Dictionary

Để truy cập các thành phần bên trong dictionaries thì có thể dùng key name bên trong dấu ngoặc vuông:

```
x = thisdict["model"]
print ('model of car is', x)
# Kết quả x là "Mustang"
```

Dùng hàm get() để có thể đạt kết quả tương tự

```
x = thisdict.get("model")
print ('model of car is', x)
# Kết quả x là "Mustang"
```

2.3.4 Cập nhật các giá trị phần tử trong dictionary

Cập nhật phần tử bằng phép gán “=”

```
thisdict = {
    "brand": "Ford",
    "model": "Mustang",
    "year": 1964
}
thisdict["year"] = 2018
```

Trong dữ liệu kiểu Dictionary thì có 2 thành phần key “year” và giá trị đi kèm với nó là “1964”. Trong phần ví dụ ta thay đổi giá trị 1964 của key name “year” thành 2018.

2.3.5 Vòng lặp trong Dictionary

Có thể sử dụng phòng lặp trong dữ liệu Dictionary bằng sử dụng vòng for như sau:

```
for x in thisdict:
    print(x)
```

Ví dụ trên in ra tất cả key name trong dữ liệu Dictionary

Có thể in ra giá trị trong Dict bằng vòng lặp:

```
for x in thisdict:
    print(thisdict[x])
```

Có thể in ra key name + giá trị trong Dict bằng vòng lặp:

```
for x, y in thisdict.items():
    print(x, y)
```

2.3.6 Các phương thức cơ bản trong Dictionary

2.3.6.1 Kiểm tra xem một key name có tồn tại không?

```
thisdict = {
    "brand": "Ford",
    "model": "Mustang",
```

```

    "year": 1964
}
if "model" in thisdict:
    print("Yes, 'model' is one of the keys in the
thisdict dictionary")

```

2.3.6.2 Xác định có bao nhiêu thành phần trong dictionary bằng hàm len().

```
print(len(thisdict))
```

2.3.6.3 Thêm thành phần vào Dictionary

```

thisdict = {
    "brand": "Ford",
    "model": "Mustang",
    "year": 1964
}
thisdict["color"] = "red"
print(thisdict)

```

Thêm key name color và giá trị red vào dữ liệu Dictionary

2.3.6.4 Xoá những thành phần trong Dictionary (pop, del, clear).

Dùng hàm pop()

```

thisdict = {
    "brand": "Ford",
    "model": "Mustang",
    "year": 1964
}
thisdict.pop("model")
print(thisdict)

```

Dùng del để xoá thành phần với key name chỉ định.

```

thisdict = {
    "brand": "Ford",
    "model": "Mustang",
    "year": 1964
}
del thisdict["model"]
print(thisdict)

```

Dùng clear() để xoá toàn bộ dữ liệu trong Dictionary

```

thisdict = {
    "brand": "Ford",
    "model": "Mustang",
    "year": 1964
}
thisdict.clear()
print(thisdict)

```

2.3.6.5 Copy và xây dựng một bản sao từ Dictionary có sẵn

Không thể copy đơn giản bằng dấu = như dict2= dict1 một cách bình thường như vậy. Bởi vì dict2 chỉ là tham khoả cho dict1 và có thể làm những thay đổi cho dict1 sẽ tự động thay đổi cho dict2.

Có nhiều cách để copy, một trong cách đó là dùng hàm copy().

```

thisdict = {
    "brand": "Ford",
    "model": "Mustang",
    "year": 1964
}
mydict = thisdict.copy()
print(mydict)

```

Dùng cách khác tạo copy là dùng chính hàm được xây dựng sẵn dict().

```

thisdict = {
    "brand": "Ford",
    "model": "Mustang",
    "year": 1964
}
mydict = dict(thisdict)
print(mydict)

```

Một Dict có thể chứa bên trong nhiều Dict khác gọi là Nest Dictionaries.

```

myfamily = {
    "child1" : {
        "name" : "Emil",
        "year" : 2004
    },
    "child2" : {
        "name" : "Tobias",
        "year" : 2007
    },
    "child3" : {
        "name" : "Linus",
        "year" : 2011
    }
}

```

Có thể tạo nest từ 3 Dicts sẵn có

```

child1 = {
    "name" : "Emil",
    "year" : 2004
}
child2 = {
    "name" : "Tobias",

```

```

    "year" : 2007
}
child3 = {
    "name" : "Linus",
    "year" : 2011
}
myfamily = {
    "child1" : child1,
    "child2" : child2,
    "child3" : child3
}

```

Xây dựng một dict() mới

```

thisdict = dict(brand="Ford", model="Mustang",
year=1964)
# note that keywords are not string literals
# note the use of equals rather than colon for
# the assignment
print(thisdict)

```

2.3.7 Tổng hợp các hàm dùng kèm với Dictionary

Phương Pháp	Mô Tả
clear()	Xoá hết tất cả thành phần của từ điển
copy()	Trả về bản copy của một Dict
fromkeys()	Trả về một dict với key chỉ định và giá trị
get()	Trả về giá trị của key chỉ định
items()	Trả về list chứa tuple và mỗi key + giá trị
keys()	Trả về list chứa các key của Dict
pop()	Xoá thành phần với key chỉ định
popitem()	Xoá cặp key-giá trị cuối cùng
setdefault()	Trả về giá trị của key chỉ định. Nếu key không tồn tại: thêm vào key và với giá trị chỉ định
update()	Cập nhật Dict với cặp key-giá trị chỉ định
values()	Trả về list tất cả giá trị trong dict

2.4 Set trong Python

2.4.1 Giới thiệu Set trong Python

Set là tập hợp những thành phần chưa thiết lập và chỉ định sẵn. Trong python, sets được viết trong ngoặc nhọn.

Set trong python là một tập các giá trị không có thứ tự. Mỗi giá trị trong set là duy nhất, không thể lặp lại và bất biến (tức bạn không thể thay đổi giá trị các phần tử trong set).

Tuy nhiên các set có thể thay đổi. Chúng ta có thể thêm, xóa các phần tử trong set.

Các set có thể được sử dụng để thực hiện các phép tập hợp như phép giao, hợp

2.4.2 Cách tạo một Set

Ví dụ tạo một set:

```
thisset = {"apple", "banana", "cherry"}  
print(thisset)
```

Tạo một set bằng chính hàm set()

```
thisset = set(("apple", "banana", "cherry")) # note the  
double round-brackets  
print(thisset)
```

2.4.3 Truy xuất thành phần trong Set

Vì set là tập giá trị không có thứ tự nên việc truy xuất 1 một phần trong set là không thể và bao lỗi. Ta chỉ có thể kiểm tra xem phần tử có tồn tại trong set bằng cách sau:

```
thisset = {"apple", "banana", "cherry"}  
print("banana" in thisset)
```

2.4.4 Các phương thức cơ bản của Set

2.4.4.1 Để thêm 1 thành phần vào Set thì dùng phương pháp add()

```
thisset = {"apple", "banana", "cherry"}  
thisset.add("orange")  
print(thisset)
```

2.4.4.2 Thêm nhiều thành phần dùng update() như sau:

```
thisset = {"apple", "banana", "cherry"}  
thisset.update(["orange", "mango", "grapes"])  
print(thisset)
```

2.4.4.3 Đo chiều dài của set

```
thisset = {"apple", "banana", "cherry"}  
print(len(thisset))
```

2.4.4.4 Thêm bớt thành phần trong set (remove, discard, clear)

Xoá bớt thành phần trong set dùng remove() hoặc discard() như sau:

```
thisset = {"apple", "banana", "cherry"}  
thisset.remove("banana")  
print(thisset)
```

Xoá thành phần banana trong set

LUU Ý: nếu thành phần banana không tồn tại sẽ phát sinh ra lỗi.

Còn với discard() thì sẽ không báo lỗi nếu thành phần không tồn tại.

Có thể dùng pop() với chỉ số để trả về giá trị mà xoá thành phần cuối cùng.

```
thisset = {"apple", "banana", "cherry"}  
x = thisset.pop()  
print(x)  
print(thisset)
```

Giá trị mà hàm pop() trả về chính là giá trị mà pop() đã xoá -> cherry

Dùng clear() để xoá toàn bộ set

```
thisset = {"apple", "banana", "cherry"}  
thisset.clear()  
print(thisset)
```

2.4.4.5 Gia nhập 2 sets

Gia nhập hai sets dùng lệnh union() hoặc update()

```
set1 = {"a", "b", "c"}  
set2 = {1, 2, 3}  
set3 = set1.union(set2)  
print(set3)
```

2.4.5 Tổng hợp các hàm dùng với Set

Phương pháp	Mô tả
add()	Thêm thành phần vào set
clear()	Xoá tất cả thành phần từ set
copy()	Trả về giá trị copy của
difference()	Trả về giá trị khác nhau giữa hai sets
difference_update()	Xoá những thành phần trong set bao gồm những thành phần khác với set chỉ định.
discard()	Xoá thành phần chỉ định
intersection()	Trả về một set, với phần giao nhau giữa hai set
intersection_update()	Trả về những thành phần đã loại bỏ phần giao nhau với set chỉ định
isdisjoint()	Trả về bất cứ giá trị không giao nhau giữa hai set
issubset()	Trả về tất cả những thành phần kề cả không chứa trong set

issuperset()	Trả về bất kể set này chứa set khác hoặc không
pop()	Xoá yếu tố từ set với số chỉ định
remove()	Xoá yếu tố chỉ định của set
symmetric_difference()	Trả về set với sự khác biệt đối xứng của 2 sets
symmetric_difference_update()	Thêm vào những khác biệt đối xứng từ set này và set khác
union()	Trả về nội dung gộp 2 set.
update()	Update set với tổ hợp set này với những cái khác

Bài tập:

Phần bài tập các kiểu dữ liệu List

1. Sử dụng kiểu dữ liệu List in ra màn hình tam giác được tạo thành từ những dấu * với chiều cao 4 đơn vị.
2. Sử dụng kiểu List viết chương trình có dữ liệu điểm trung bình môn học của học sinh Nguyễn Văn A như sau:

- Nguyễn Văn A: Toán 7.4 Lý 6.5 Hoá 7.0
- Hiển thị màn hình hướng dẫn nhập liệu search thông qua nhập tên học sinh. Ví dụ nhập Nguyễn Văn A thì nghĩa là hiển thị xuất ra tên và điểm trung bình môn học.

GÓI Ý: Áp dụng các kiểu dữ liệu List cho tên và từng môn học tương ứng cho Học Sinh như sau

Ten= [“Nguyen Van A”, “Nguyen Van B”, “Nguyen Van C”]

Toan= [“7.4”, “6.5”, “7.0”]

Tương tự cho các môn còn lại. Ví dụ search Nguyen Van A nghĩa là chọn kiểu list vị trí số 1 thì xuất ra màn hình Ten[0], Toan[0], Ly[0], Hoa[0].

3. Sử dụng kiểu List viết chương trình có kiểu dữ liệu là các loại trái cây từ apple tới mango như trên. Xuất ra màn hình tên và giá tiền/ kg của các loại trái cây từ cherry tới melon. Dữ liệu giá tiền tự chọn
4. Thêm phần check tên học sinh có tồn tại hay không ở phần đầu đoạn code và xuất ra màn hình.

THÊM VÀO CODE BÀI SỐ 2

5. Chỉnh sửa phần BÀI THỰC HÀNH SỐ 3, bỏ bớt thành phần cuối cùng. Chèn thêm trước trái kiwi thêm hai loại trái cây khác tự chọn. LUU Ý TRÁI CÂY DÙNG TIẾNG ANH.
6. Nhập từ bàn phím và lưu dữ vào từng list với các trường như sau: Mã số Sinh Viên, Họ và Tên, Ngày Tháng Năm Sinh, Tên Lớp, Điểm Thường Kỳ 1,2,3, Điểm Thi Giữa Kỳ, Điểm Thi Cuối Kỳ, Điểm trung bình tổng kết, Kết quả đậu rớt.

LUU Ý: chương trình phải có sự lựa chọn cho người dùng muốn nhập liệu hay search. VÀ search dựa trên MSSV, Tên, hoặc nhóm lớp tương ứng như DH14A, DH14B.

Phần bài tập của Dictionary

7. Thiết lập dữ liệu Dictionary của 1 sinh viên gồm STT, Họ và Tên, MSSV, Ngày tháng năm sinh, quê quán, Học Phí, Sở Thích.
 - + In ra màn hình Sở thích của Sinh viên
 - + Sau đó thay đổi giá trị sở thích và in ra lại lần nữa.
8. Chính sửa lại code bài 7
 - + **In** ra tất cả key name
 - + **Dùng** key name đã in ra để truy xuất 1 giá trị nào đó
 - + **In** ra tất cả giá trị theo key name đó.
 - + **In** ra tất cả key name và giá trị đồng thời
9. Chính sửa code phần bài 7
 - + **Kiểm tra** và in ra số lượng thành phần trong Dictionary
 - + **Thêm** vào Ngoại ngữ và Dân Tộc, và in ra kết quả tất cả thành phần trong Dictionary
 - + **Xoá** key name và giá trị “sở thích” và sau đó in ra tất cả thành phần lại lần nữa.
10. Thiết lập 3 bộ dữ liệu sinh viên gồm: Tên, MSSV, lớp.
 - + **Nest** lại 3 dict của 3 sinh viên trên thành một dict mới tên lớp DHDT12B.
 - + **In** ra key name tất cả thành phần trong dict lớp DHDT12B
 - + **Search** và xuất ra bộ dữ liệu dict của sinh viên thứ 2
 - + **Copy** dict lớp DHDT12B và tạo ra dict mới là tên môn học ANHVAN01
 - + **Thêm** 2 dict của 2 sinh viên lớp DHDT12A vào lớp ANHVAN01.
11. Chính sửa lại phần code bài 10
 - + **Thêm** vào 2 sinh viên cho lớp DHDT12B thành tổng 5 sinh viên.
 - + **Thêm** vào lớp ANHVAN01 hai sinh viên mới lớp DHDT12B

- + **Chỉnh** sửa toàn bộ dict DHDT12B, lớp ANHVAN01 và 2 sinh viên lớp DHDT12A bằng cách thêm thành phần quê quán, trình độ anh văn điểm Toiec hiện tại, tình trạng học phí.
- + **Xoá** 1 sinh viên lớp DHDT12A do chưa đóng học phí khỏi lớp ANHVAN01

Phần bài tập Tuples

12. Sử dụng kiểu Tuple viết chương trình có dữ liệu điểm trung bình môn học của học sinh Nguyễn Văn A như sau:

Nguyễn Văn A: Toán 7.4 Lý 6.5 Hóa 7.0

Hiển thị màn hình hướng dẫn nhập liệu search thông qua nhập tên học sinh. Ví dụ nhập Nguyễn Văn A thì nghĩa là hiển thị xuất ra tên và điểm trung bình môn học.

Sau đó thay đổi giá trị môn Toán của HS Nguyễn Văn A là 8.0

13. Sử dụng kiểu Tuple viết chương trình có kiểu dữ liệu là các loại trái cây từ `["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]`. Xuất ra màn hình tên và giá tiền/ kg của các loại trái cây từ cherry tới melon. Dữ liệu giá tiền tự chọn.

LƯU Ý sinh viên dùng vòng lặp để xuất giá trị ra màn hình.

14. Tạo hai Tuple với thông số sau"

Tuple 1: `["apple", "banana", "cherry"]`

Tuple 2: `["kiwi", "melon", "mango"]`

NHẬP 2 tuple lại làm 1 và in ra kết quả

LƯU Ý: dùng hàm tạo tuples.

15. Tạo ra dữ liệu sẵn có về 10 loại trái cây và giá tiền tương ứng dùng cấu trúc Tuples. Cho phép nhập liệu tên trái cây search ra giá tiền.

+ Sau đó cập nhật lại giá tiền của 5 loại trái cây trong tuple đó. Search lại lần nữa các loại trái cây và xuất ra màn hình

LƯU Ý: chương trình phải có sự lựa chọn cho người dùng muốn search tên một loại trái cây hay xuất ra toàn bộ trái cây và giá tiền tương ứng.

Bài tập về Set trong python

16. Tạo một set 10 loại trái cây và in ra màn hình

+ Kiểm tra một loại trái cây có trong set hay không? In ra kết quả Yes or No

17. Từ phần bài 16 chỉnh sửa code thêm 5 loại trái cây khác vào Set.

18. Tạo 2 set trái cây (mỗi set 5 loại trái cây và giá tiền tương ứng).

+ Search trái cây xuất ra màn hình Tên trái cây, giá tiền, và thuộc về set 1 hay 2

+ Nhập hai set làm 1 và search lại xuất lần nữa

19. Tạo 2 set trái cây (mỗi set 5 loại trái cây).

+ Tìm phần giống nhau giữa hai set này và xuất ra màn hình

+ Tìm phần set 1 khác set 2 và xuất ra màn hình

+ Tìm phần set 2 khác set 1 và xuất ra màn hình

+ Xuất ra màn hình set tổ hợp cả hai bộ set trái cây

+ Xuất ra màn hình set 1 có thêm phần khác từ set 2

+ Xuất ra màn hình set 2 có thêm phần khác từ set 1

CHƯƠNG 3: MODULES và PACKAGE

Nội Dung

3.1 Giới thiệu	59
3.2 Cách tạo một Module	59
3.3 Cách sử dụng một module	59
3.4 Các loại biến có trong module	59
3.5 Đặt tên và đổi tên module	60
3.6 Cách xây dựng một Module.....	60
3.6.1 Import vào Module	60
3.6.2 Dùng hàm dir()	60
3.6.3 Import từ module	60
3.7 Packet trong Python	
3.7.1 Giới thiệu.....	61
3.7.2 Nhập module từ package trong Python	61
3.7.3 Sử dụng package trong python	62
Bài tập chương 3	65

3.1 Giới thiệu

Modules có phần code giống như phần code của một thư viện. Một file chứa tập hợp các hàm bạn muốn bao gồm trong ứng dụng của chính bạn.

3.2 Cách tạo một Module

Để tạo một module bạn cần lưu cái code bạn muốn thành một file với đuôi “.py”. Ví dụ lưu một code với tên gọi mymodule.py có nội dung sau:

```
def greeting(name):
    print("Hello, " + name)
```

3.3 Cách sử dụng một module

Để sử dụng một module mà vừa tạo, tôi đi học app gô ta có thể dùng hàm import như sau:

```
import mymodule
mymodule.greeting("Jonathan")
```

Ví dụ trên truyền tên “Jonathan” vào hàm greeting trong file module.

3.4 Các loại biến có trong module

Một module có thể chứa nhiều hàm, như mô tả trên có thể nhiều loại biến như: arrays, dictionaries, objects.

Ví dụ lưu vào file mymodule.py nội dung sau.

```
person1 = {
    "name": "John",
    "age": 36,
    "country": "Norway"
}
```

Ví dụ trên dùng kiểu dictionaries trong module. Bạn có thể import keyword của thành phần trong dict của module bằng hàm import như sau:

```
import mymodule
a = mymodule.person1["age"]
print(a)
```

3.5 Đặt tên và đổi tên module.

Đặt tên module bất cứ tên gì bạn muốn nhưng đuôi bắt buộc phải là .py.

Đổi tên bằng cách tạo liên kết khi bạn import một module bằng sử dụng keyword as như sau:

```
import mymodule as mx
a = mx.person1["age"]
print(a)
```

3.6 Cách xây dựng một Module

Có nhiều cách xây dựng những module trong Python

3.6.1 Import vào Module bất cứ thứ gì bạn muốn.

Ví dụ: Import và sử dụng module platform

```
import platform
x = platform.system()
print(x)
```

3.6.2 Dùng hàm dir()

Sử dụng hàm dir() được xây dựng sẵn để liệt kê các hàm hoặc biến trong một module.

Hàm dir() sử dụng như sau:

Ví dụ: liệt kê tất cả các tên thuộc về module platform

```
import platform
x = dir(platform)
print(x)
```

Lưu ý rằng hàm dir() có thể sử dụng trong tất cả module, cũng có thể trong những module bạn tự tạo ra.

3.6.3 Import từ module

Bạn có thể chọn import một phần từ một module bằng sử dụng keyword:

Ví dụ tên hàm mymodule có một hàm và một dictionary

```
def greeting(name):
    print("Hello, " + name)
person1 = {
    "name": "John",
```

```

    "age": 36,
    "country": "Norway"
}

```

Ví dụ: Import chỉ duy nhất kiểu dict tên person1 từ module

```

from mymodule import person1

print (person1["age"])

```

Lưu ý: Khi đang import sử dụng keyword from, không được dùng tên module khi cần tham khảo những yếu tố trong module. Ví dụ

```
person1["age"], not mymodule.person1["age"]
```

3.7 Packet trong Python

3.7.1 Giới thiệu

Trong bài trước bạn đã tìm hiểu về module, cách tạo và gọi một module trong [Python](#). Ở bài này, chúng ta sẽ học cách phân chia code thành những module hiệu quả, rõ ràng, sử dụng các package trong Python.Thêm nữa là cách để nhập và sử dụng package của riêng bạn, hoặc package bạn tải về từ đâu đó vào chương trình Python.

Package trong Python là gì?

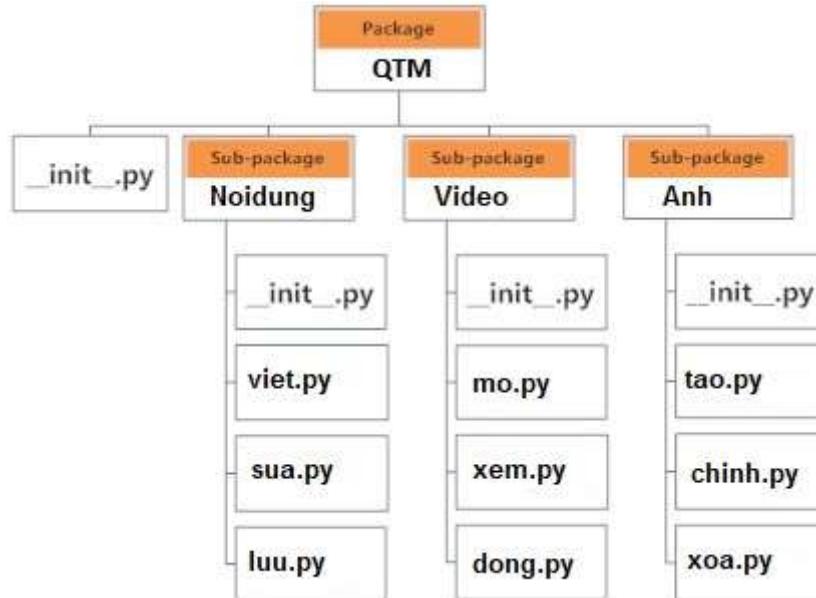
Thông thường người dùng không lưu trữ tất cả các file của mình trên máy tính ở cùng một thư mục, vị trí, mà sử dụng hệ thống phân cấp thư mục để truy cập dễ dàng hơn. Các tệp tương tự hoặc cùng liên quan đến một chủ đề nào đó sẽ được để trong cùng một thư mục, chẳng hạn các bài học về [hàm Python](#) sẽ lưu trong thư mục hàm Python. Tương tự như vậy, Python có các package cho thư mục và module cho file.

Khi chương trình đang code ngày càng lớn với rất nhiều module, chúng ta sẽ đặt những module giống nhau vào một package, và những nhóm module khác vào package khác. Điều này giúp dễ dàng quản lý chương trình hơn và nó cũng dễ hiểu hơn.

Nếu như thư mục có thể chứa các thư mục con thì package cũng vậy, trong một package có thể có package con và các module khác.

Một thư mục phải chứa file có tên `__init__.py` để Python hiểu thư mục này là một package. File này có thể để trống, nhưng thông thường các lập trình viên thường đặt code khởi tạo cho package ở đây.

Giả sử, ta đang phát triển chương trình có tên QTM, với các package con, module như sau:



3.7.2 Nhập module từ package trong Python

Ta có thể nhập các module từ package sử dụng toán tử dấu chấm (.) .

Ví dụ, nếu muốn nhập module `mo.py` trong ví dụ trên, bạn làm như sau:

```
import QTM.Video.mo
```

Nếu trong module `mo.py` chứa hàm có tên là `chon_video()`, bạn sẽ phải sử dụng tên đầy đủ để tham chiếu tới nó:

```
import QTM.Video.mo.chon_video(video1)
```

Nếu cấu trúc trên dài quá, bạn có thể nhập module mà không cần tiền tố package như sau:

```
from QTM.Video import mo
```

Ta có thể gọi hàm đơn giản như sau:

```
mo.chon_video(video1)
```

Ngoài ra, còn một cách nữa để chỉ nhập hàm được yêu cầu (lớp hoặc biến) từ module trong 1 package như sau:

```
from QTM.Video.mo import chon_video
```

Sau đó, trực tiếp gọi hàm này:

```
chon_video(video1)
```

Dù dễ dàng hơn, nhưng cách nhập sau cùng này không được khuyến khích sử dụng. Việc dùng tên đầy đủ sẽ giúp giảm tình trạng nhầm lẫn và tránh bị trùng giữa những định danh giống nhau.

Trong khi nhập các package, Python sẽ tìm kiếm danh sách thư mục được định nghĩa trong sys.path, giống như đường dẫn tìm kiếm module.

3.7.3 Sử dụng package trong Python

Trong Python, một số module trong cùng thư mục (và thư mục con) có thể kết hợp lại tạo ra một package. Package giúp đơn giản hóa hơn nữa việc sử dụng nhiều module có liên quan.

Thực hiện ví dụ sau:

Tạo thư mục my_package. Trong thư mục này tạo ra 2 file module1.py và module2.py với code như sau:

	module1.py	module2.py
1.	print('This is the module 1')	
2.		
3.	def func1():	
4.	print('Module 1 - func 1')	

Nếu dùng lại ở đây sẽ có 2 module riêng rẽ. Khi sử dụng bạn cần import từng module.

Giờ hãy tạo file __init__.py trong thư mục my_package. Lưu ý có hai ký tự _ ở trước và ở sau init. Viết code sau cho __init__.py:

1.	from my_package.module1 import *
2.	from my_package.module2 import *
3.	
4.	#hoặc cũng có thể import như thế này:
5.	#import my_package.module1
6.	#import my_package.module2

Đây là hai lệnh import thông thường mà bạn đã học. Bạn có thể sử dụng kiểu import nào cũng được. Tuy nhiên cần lưu ý cách viết tên module:

my_package.module1. Bạn cần **chỉ định tên thư mục**, nếu không Python sẽ không tìm được module tương ứng.

Tên file __init__.py có ý nghĩa đặc thù trong Python. Nếu Python nhìn thấy thư mục nào có file với tên gọi __init__.py, nó sẽ tự động coi thư mục này như một package, chứ không còn là thư mục bình thường nữa.

Bên ngoài thư mục package hãy tạo module main.py và viết code như sau:

```

1. import my_package as mp
2.
3. mp.module1.func1()
4. mp.module2.func2()
5.
6. from my_package import *
7. module1.func1();

```

Để ý rằng lệnh import giờ không còn hoạt động nữa với từng module mà là với tên thư mục my_package

Tất cả các thư mục chưa file __init__.py sẽ trở thành một package. Tên thư mục trở thành tên packet và có thể sử dụng cùng lệnh import hoặc from/ import.

Khi import một packet bạn sẽ đồng thời import tất cả các module của nó (được chỉ định trong __init__.py).

Nếu sử dụng lệnh **import <tên_package>**, bạn sẽ truy cập vào các hàm của từng module qua cú pháp **<tên_package>.<tên_module>.<tên_hàm>()**.

Nếu sử dụng **from <tên_package> import <tên_hàm>**, bạn cũng có thể sử dụng tên ngắn ngọn của hàm như bình thường

*Lưu ý: trong __init__.py và trong module sử dụng package **nên dùng cùng một cách import**. Ví dụ, cùng dùng import hoặc cùng dùng from .. import.*

Bài tập chương 3:

1. Viết chương trình tạo module infoSV.py trong đó có hai hàm tên, và hàm mssv. Nhập tên và mssv vào module thông qua hai hàm trên. Có xuất ra màn hình.
2. Tạo module dùng biến dạng dicts với thông tin nhập vào là:
 - + Số thứ tự
 - + Họ và Tên
 - + MSSV
 - + Ngày tháng năm sinh
 - + Quê quán
 - + Lớp
3. Tạo module Lớp 12A và 12B với 5 sinh viên kiểu dữ liệu dictionary gồm: Tên, mssv, ngày tháng năm sinh, quê quán.
 - + Tạo module lopLapTrinhMang
 - + Import 3 sinh viên từ lớp 12A và 2 sinh viên từ 12B
 - + Xuất ra màn hình danh sách tất cả sinh viên lớp Lập Trình Mạng
4. Viết chương trình và in giá trị theo công thức cho trước: $Q = \sqrt{[(2 * C * D)/H]}$ (bằng chữ: Q bằng căn bậc hai của [(2 nhân C nhân D) chia H]. Với giá trị cố định của C là 50, H là 30. D là dãy giá trị tùy biến, được nhập vào từ giao diện người dùng, các giá trị của D được phân cách bằng dấu phẩy.

Ví dụ: Giả sử chuỗi giá trị của D nhập vào là 100,150,180 thì đầu ra sẽ là 18,22,24.

Gợi ý:

- Nếu đầu ra nhận được là một số dạng thập phân, bạn cần làm tròn thành giá trị gần nhất, ví dụ 26.0 sẽ được in là 26.
 - Trong trường hợp dữ liệu đầu vào được cung cấp cho câu hỏi, nó được giả định là đầu vào do người dùng nhập từ giao diện điều khiển.
5. Viết một chương trình có 2 chữ số, X, Y nhận giá trị từ đầu vào và tạo ra một mảng 2 chiều. Giá trị phần tử trong hàng thứ i và cột thứ j của mảng phải là $i*j$.

Lưu ý: i=0,1,...,X-1; j=0,1,...,Y-1.

Ví dụ: Giá trị X, Y nhập vào là 3,5 thì đầu ra là: [[0, 0, 0, 0, 0], [0, 1, 2, 3, 4], [0, 2, 4, 6, 8]]

Gợi ý:

Viết lệnh để nhận giá trị X, Y từ giao diện điều khiển do người dùng nhập vào.

6. Viết một chương trình chấp nhận chuỗi từ do người dùng nhập vào, phân tách nhau bởi dấu phẩy và in những từ đó thành chuỗi theo thứ tự bảng chữ cái, phân tách nhau bằng dấu phẩy.

Giả sử đầu vào được nhập là: without,hello,bag,world, thì đầu ra sẽ là: bag,hello,without,world.

Gợi ý:

Trong trường hợp dữ liệu đầu vào được nhập vào chương trình nó nên được giả định là dữ liệu được người dùng nhập vào từ giao diện điều khiển.

7. Viết một chương trình chấp nhận chuỗi là các dòng được nhập vào, chuyển các dòng này thành chữ in hoa và in ra màn hình. Giả sử đầu vào là:

Hello world Practice makes perfect

Thì đầu ra sẽ là:

HELLO WORLD PRACTICE MAKES PERFECT

Gợi ý:

Trong trường hợp dữ liệu đầu vào được nhập vào chương trình nó nên được giả định là dữ liệu được người dùng nhập vào từ giao diện điều khiển.

8. Viết một chương trình chấp nhận đầu vào là một chuỗi các từ tách biệt bởi khoảng trắng, loại bỏ các từ trùng lặp, sắp xếp theo thứ tự bảng chữ cái, rồi in chúng.

Giả sử đầu vào là: hello world and practice makes perfect and hello world again

Thì đầu ra là: again and hello makes perfect practice world

Gợi ý:

- Trong trường hợp dữ liệu đầu vào được nhập vào chương trình nó nên được giả định là dữ liệu được người dùng nhập vào từ giao diện điều khiển.
- Sử dụng set để loại bỏ dữ liệu trùng lặp tự động và dùng sorted() để sắp xếp dữ liệu.

9. Viết một chương trình chấp nhận đầu vào là chuỗi các số nhị phân 4 chữ số, phân tách bởi dấu phẩy, kiểm tra xem chúng có chia hết cho 5 không. Sau đó in các số chia hết cho 5 thành dãy phân tách bởi dấu phẩy.

Ví dụ đầu vào là: 0100,0011,1010,1001

Đầu ra sẽ là: 1010

Gợi ý:

Trong trường hợp dữ liệu đầu vào được nhập vào chương trình nó nên được giả định là dữ liệu được người dùng nhập vào từ giao diện điều khiển.

10. Viết một chương trình tìm tất cả các số trong đoạn 1000 và 3000 (tính cả 2 số này) sao cho tất cả các chữ số trong số đó là số chẵn. In các số tìm được thành chuỗi cách nhau bởi dấu phẩy, trên một dòng.

Gợi ý

Trong trường hợp dữ liệu đầu vào được nhập vào chương trình nó nên được giả định là dữ liệu được người dùng nhập vào từ giao diện điều khiển.

Chương 4: Numpy

Nội Dung

4.1 Giới thiệu	69
4.2 Ưu điểm	69
4.3 Mảng 1 chiều.....	69
4.3.1 Tạo mảng 1 chiều	69
4.3.2 Thao tác trên mảng	70
4.4 Mảng nhiều chiều.....	74
4.4.1 Tạo mảng nhiều chiều	74
4.4.2 Thao tác trên mảng nhiều chiều.....	75
BÀI TẬP	80

4.1 Giới thiệu:

- Là một thư viện bao gồm các đối tượng mảng đa chiều (multidimensional array) và một tập hợp các phương thức để xử lý mảng đa chiều đó.
- Sử dụng Numpy, các toán tử toán học và logic có thể được thực hiện
- Thay thế cho List
- Cài đặt: pip install numpy

4.2 Ưu điểm :

- Chứa bộ các giá trị
- Có thể chứa các loại dữ liệu khác nhau
- Có thể thay đổi, thêm, xóa
- Đầy đủ chức năng tính toán
- Rất nhiều thư viện được xây dựng sẵn trong Numpy

4.3 Mảng một chiều:

4.3.1 Tạo mảng 1 chiều:

- o Từ object có định dạng array: sử dụng array()

```
import numpy as np
# tao mang mot chieu
array_1 = np.array([1, 2, 3])
print(array_1)
print(type(array_1))

[1 2 3]
<class 'numpy.ndarray'>
```

- o Từ list, tuple: sử dụng asarray()

```
import numpy as np
x = [1,2,3]
a = np.asarray(x)
print(a)

[1 2 3]
```

```
# Tao mang tu Tuple
x = (1, 2, 3)
a = np.asarray(x)
print(a)

[1 2 3]
```

- o Từ range: sử dụng **numpy.arange(start, stop, step, dtype)**

```
mang_2 = np.arange(10,20,2)
print(mang_2)
```

```
[10 12 14 16 18]
```

- Từ range có khoảng cách đều nhau: sử dụng **numpy.linspace(start, stop, num, endpoint, dtype)**

```
mang_3 = np.linspace(10,20, 5)
print(mang_3)
mang_4 = np.linspace(10,20, 5, endpoint = False)
print(mang_4)
```

```
[10. 12.5 15. 17.5 20.]
[10. 12. 14. 16. 18.]
```

4.3.2 Thao tác trên mảng:

- Số phần tử của mảng

```
print(array_1)
print(array_1.shape[0])
```

```
[1 2 3]
3
```

- Cập nhật phần tử trong mảng

```
array_1[1] = 250
```

```
print(array_1)
```

```
[ 1 250  3]
```

- Truy xuất phần tử trên mảng theo index (từ 0 đến chiều dài mảng -1)

Index:	0	1	2	3	4
Value:	88	19	46	74	94

- Truy x

Index:	0	1	2	3	4
Value:	88	19	46	74	94

-1

```
# Truy xuất phần tử trong mảng
mang_4 = np.array([1, 3, 2, 4, 5, 7, 9])
print(mang_4)
print(mang_4[3])
print(mang_4[3:])
print(mang_4[3:5])
print(mang_4[mang_4 < 5]) # có kèm điều kiện

[1 3 2 4 5 7 9]
4
[4 5 7 9]
[4 5]
[1 3 2 4]
```

- Thêm phần tử vào cuối mảng: sử dụng append()

```
# Thêm các phần tử vào cuối mảng
mang_5 = np.arange(10)
mang_5 = np.append(mang_5, [25, 30])
print(mang_5)

[ 0  1  2  3  4  5  6  7  8  9 25 30]
```

- Thêm phần tử vào index trong mảng: sử dụng insert()

```
# Thêm phần tử vào index = 3, giá trị 100
print(mang_5)
mang_5 = np.insert(mang_5, 3, 100)
print(mang_5)

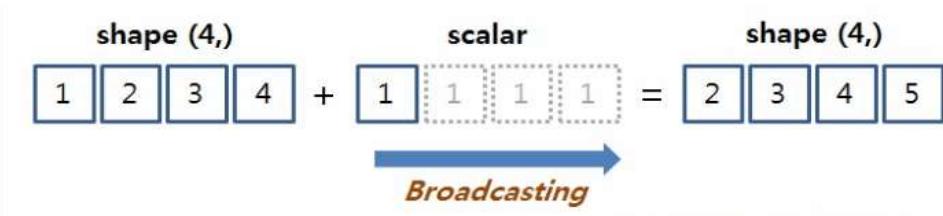
[ 0  1  2  3  4  5  6  7  8  9 25 30]
[ 0  1  2 100  3  4  5  6  7  8  9 25 30]
```

- Xóa phần tử tại index trong mảng: sử dụng delete()

```
# Xóa phần tử tại vị trí index = 11
print(mang_5)
mang_5 = np.delete(mang_5, 11)
print(mang_5)

[ 0  1  2 100  3  4  5  6  7  8  9 25 30]
[ 0  1  2 100  3  4  5  6  7  8  9 30]
```

- Broadcasting: thực hiện các phép toán số học trên mảng



```
import numpy as np

a = np.array([1, 2, 3, 4])
b = np.array([5, 6, 7, 8])
mang_cong = a + b
print(mang_cong)
mang_tru = a - b
print(mang_tru)
mang_nhan = a * b
print(mang_nhan)
mang_chia = a / b
print(mang_chia)

[ 6  8 10 12]
[-4 -4 -4 -4]
[ 5 12 21 32]
[0.2          0.33333333 0.42857143 0.5      ]
```

- Tìm max, min: sử dụng max() hoặc amax(), min hoặc amin()

```
# max, min
d = np.array([3, 10, 9, 12, 8, 5])
print(np.amax(d))
print(np.amin(d))

12
3
```

- Tính tổng giá trị: sử dụng sum()

```
arr = np.arange(10) # array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
arr.sum()
```

45

- Giá trị trung bình số học: sử dụng mean()

```
print(d)
print(np.mean(d))
```

```
[ 3 10  9 12  8  5]
7.833333333333333
```

- Giá trị trung bình giữa: sử dụng median() dựa trên mảng đã sắp xếp, lấy trung bình phần tử ở giữa mảng

```
print(d)
print(np.median(d)) # 3, 5, 8, 9, 10, 12 => trung bình giữa = (8+9)/2
```

```
[ 3 10  9 12  8  5]
8.5
```

- Sắp xếp tăng dần trong mảng: sử dụng sort()

```
mang_5 = np.array([3, 1, 2, 4, 7, 3, 8, 9, 5, 6])
print(mang_5)
print(np.sort(mang_5))
```

```
[3 1 2 4 7 3 8 9 5 6]
[1 2 3 3 4 5 6 7 8 9]
```

- Tìm index của các phần tử trong mảng thỏa điều kiện: sử dụng where()

```
print(mang_5)
indexes = np.where(mang_5 >= 5)
print(indexes)
print(mang_5[indexes])
```

```
[3 1 2 4 7 3 8 9 5 6]
(array([4, 6, 7, 8, 9], dtype=int32),
 [7 8 9 5 6])
```

Tìm các phần tử trong mảng thỏa điều kiện: sử dụng extract(điều kiện, mảng)

```
print(mang_5)
print(np.extract(np.mod(mang_5,2)==0, mang_5))
print(np.extract(mang_5 >= 5, mang_5))
```

```
[3 1 2 4 7 3 8 9 5 6]
[2 4 8 6]
[7 8 9 5 6]
```

4.4 Mảng nhiều chiều:

4.4.1 Tạo mảng nhiều chiều:

- o Từ object có định dạng array: sử dụng array()

```
import numpy as np
# Tạo mảng
array_2 = np.array([[1, 2, 3], [4, 5, 6]])
print(array_2)
print(type(array_2))
```

```
[[1 2 3]
 [4 5 6]]
<class 'numpy.ndarray'>
```

- o Có giá trị mặc định là 0, 1: dùng zeros(), ones()

```
# tạo mảng với 0 hoặc 1
mang_zeros = np.zeros([2,2,2], dtype = int)
print("Mảng zeros")
print(mang_zeros)
print("Mảng ones")
mang_ones = np.ones([2,2], dtype = int)
print(mang_ones)
```

```
Mảng zeros
[[[0 0]
 [0 0]]]
```

```
Mảng ones
[[1 1]
 [1 1]]]
```

4.4.2 Thao tác trên mảng nhiều chiều:

- o Số dòng, cột trên mảng

```
print(array_2)
print(array_2.shape)
```

```
[[1 2 3]
 [4 5 6]]
(2, 3)
```

- o Cập nhật phần tử trong mảng

```
# cập nhật
array_2[0][0] = 100
print(array_2)
```

```
[[100 2 3]
 [4 5 6]]
```

- o Truy xuất phần tử trên mảng: theo ma trận, dòng, cột (index = 0 -> chiều dài - 1)

```
# truy xuất phần tử trong mảng
print(array_2)
print(array_2[1,2])
print(array_2[1,:])
```

```
[[1 2 3]
 [4 5 6]]
6
[4 5 6]
```

```
print(b)
print("b[0][1][2] =", b[0][1][2]) # ma trận 0, dòng 1, cột 2
```

```
[[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

```
[[[12 13 14 15]
 [16 17 18 19]
 [20 21 22 23]]]
b[0][1][2] = 6
```

- Thay đổi kích cỡ mảng: dùng shape, reshape(số_lượng_ma_trận, số_dòng, số_cột)

```
# thay đổi kích cỡ mảng
print(array_2)
print(array_2.shape)
array_2.shape = (3,2)
print(array_2)
```

```
[[100    2    3]
 [ 4     5    6]]
(2, 3)
[[100    2]
 [ 3     4]
 [ 5     6]]
```

```
# thay đổi kích cỡ mảng dùng reshape
a = np.arange(24)
print(a)
b = a.reshape(2, 3, 4) # ma trận, dòng, cột
print(b)
print(b[0][1][2])
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23]
[[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
 [[12 13 14 15]
 [16 17 18 19]
 [20 21 22 23]]]
```

6

- Tìm max, min: sử dụng amax(), amin()

```
print("Max:", np.amax(a))
print("Min:", np.amin(a))
print("Max - theo từng cột:", np.amax(a, 0))
print("Min - theo từng cột:", np.amin(a, 0))
print("Max - theo từng dòng:", np.amax(a, 1))
print("Min - theo từng dòng:", np.amin(a, 1))
```

```
Max: 9
Min: 2
Max - theo từng cột: [8 7 9]
Min - theo từng cột: [2 4 3]
Max - theo từng dòng: [7 8 9]
Min - theo từng dòng: [3 3 2]
```

- Giá trị trung bình số học: sử dụng mean()

```
b = np.array([[ [3,7],[8,4]],[[2,4], [1, 5]]])
print(b)
print('Mean b:')
print(np.mean(b))
print('Mean b[0]:')
print(np.mean(b[0]))
```

```
[[[3 7]
 [8 4]]
```

```
[[2 4]
 [1 5]]]
Mean b:
4.25
Mean b[0]:
5.5
```

- Giá trị trung bình giữa: sử dụng median() dựa trên mảng đã sắp xếp, lấy trung bình phần tử ở giữa mảng

```
print(b)
print('Median b:')
print(np.median(b))
print('Median b[1]:')
print(np.median(b[1]))
```

```
[[[3 7]
 [8 4]]
```

```
[[2 4]
 [1 5]]]
Median b:
4.0
Median b[1]:
3.0
```

- Sắp xếp tăng dần trong mảng: sử dụng sort()

```
c = np.array([[ [9,7],[8,4]],[[2,4], [6, 5]]])
print(c)
print("Sort c")
print(np.sort(c))
```

```
[[[9 7]
 [8 4]]]
```

```
[[2 4]
 [6 5]]]
Sort c
[[[7 9]
 [4 8]]]
```

```
[[2 4]
 [5 6]]]
```

```
print('Sort c[0] axis 1 (theo dòng):')
print(np.sort(c[0], axis = 1))
```

```
Sort c[0] axis 1 (theo dòng):
[[7 9]
 [4 8]]
```

```
print('Sort c[0] axis 0 (theo cột):')
print(np.sort(c[0], axis = 0))
```

```
Sort c[0] axis 0:
[[8 4]
 [9 7]]
```

- Tìm index của các phần tử trong mảng thỏa điều kiện: sử dụng where()

```
print(c)
cond = np.where(c>5)
print(cond)
print(c[cond])
```

```
[[[9 7]
 [8 4]]]
```

```
[[2 4]
 [6 5]]]
```

```
(array([0, 0, 0, 1], dtype=int32), array([0, 0, 1, 1], dtype=int32), array([0, 1, 0, 0], dtype=int32))
[9 7 8 6]
```

- Tìm các phần tử trong mảng thỏa điều kiện: sử dụng extract(điều kiện, mảng)

```
print(c)
print("Trên c:",np.extract(c >= 5, c))
print("Trên c[0]:",np.extract(c[0] >= 5, c[0]))
```

```
[[[9 7]
 [8 4]]]
```

```
[[2 4]
 [6 5]]]
```

```
Trên c: [9 7 8 6 5]
Trên c[0]: [9 7 8]
```

BÀI TẬP:

- BT1:** 1. Cho 2 array arr_a = [1,2,3,2,3,4,3,4,5,6] và arr_b = [7,2,10,2,7,4,9,4,9,8], tạo array arr_c chỉ lấy duy nhất các phần tử xuất hiện ở cả arr_a và arr_b.
2. Từ arr_a và arr_b ở câu 1 => Tạo arr_d chứa các phần tử chỉ xuất hiện ở arr_a.
3. Cho arr_e = np.array([2, 6, 1, 9, 10, 3, 27, 8, 6, 25, 16]), hãy tạo arr_f chỉ chứa các phần tử có giá trị từ 5 đến 10 của arr_e.
- BT2:** Hãy hoàn tất file đính kèm với các yêu cầu sau:
1. Tạo arr_zeros có 10 phần tử 0, cập nhật phần tử ở vị trí thứ 5 là 1.
 2. Tạo arr_h có giá trị từ 10 đến 24. In danh sách các phần tử theo thứ tự đảo ngược của arr_h.
 3. Cho arr_k = np.array([1, 2, 0, 8, 2, 0, 1, 3, 0, 5, 0]), tạo arr_l từ arr_k với các phần tử khác 0.
 4. Từ arr_l của câu 3, thêm 2 phần tử có giá trị là 10 và 20 vào cuối array.
 5. Từ array của câu 4, thêm phần tử có giá trị 100 vào vị trí có index = 5.
 6. Từ array của câu 5, xóa các phần tử tại vị trí có index = 0, 1, 2.

BT3: Hãy hoàn tất file đính kèm với các yêu cầu sau:

1. Tạo array arr có kích thước 3x3 với các giá trị True
2. Cho arr_1D = np.array([0 1 2 3 4 5 6 7 8]). Tạo array 2 chiều có kích thước 3x3 từ arr_1D, và lưu vào arr_2D Trong arr_2D, chuyển cột 1 sang cột 3 và ngược lại.
3. Từ arr_2D của câu 2 (sau khi đổi thứ tự cột), chuyển dòng 1 sang dòng 2 và ngược lại.
4. Từ arr_2D của câu 3, đảo ngược các dòng của arr_2D.
5. Từ arr_2D của câu 4, đảo ngược các cột của arr_2D.
6. Cho arr_2D_null = np.array([[1, 2, 3], [np.NaN, 5, 6], [7, np.NaN, 9], [4, 5, 6]]), Kiểm tra trong array có giá trị rỗng không?
7. Từ arr_2D_null của câu 6, thay thế giá trị null bằng 0.

Chương 5: PANDAS

Nội Dung	
5.1 Giới thiệu	82
5.2 Ưu điểm	83
5.3 Series	83
5.4 Thao tác trên Series.....	84
5.5 Tổng hợp các hàm sử dụng	88
5.6 Data Frame.....	89
5.6.1 Tạo Data Frame	89
5.6.2 Thao tác trên Data Frame	92
5.6.3 Thống kê các hàm.....	96
5.7 Trực quan hóa trong Pandas	98
5.7.1 Giới thiệu.....	98
5.7.2 Bar plot	98
5.7.3 Boxplot	99
5.7.5 Biểu đồ histogram.....	100
5.7.6 Biểu đồ plot	100
5.7.7 Biểu đồ Pie	101
Bài tập	102

5.1 Giới thiệu:

- Là thư viện Python mã nguồn mở theo BSD-licensed, hiệu suất cao
- Cung cấp các công cụ phân tích dữ liệu, cấu trúc dữ liệu dễ dàng sử dụng cho NNLT Python
- Python với Pandas được sử dụng trong nhiều lĩnh vực bao gồm khoa học, kinh tế, phân tích thống kê...
- Cài đặt: pip install pandas
- Pandas có 3 kiểu cấu trúc dữ liệu:
 - Series
 - DataFrame
 - Panel
- Các cấu trúc dữ liệu này được xây dựng trên Numpy array, có tốc độ nhanh.
- Có thể xem Data Frame là một container của series, Panel là một container của Data Frame

- Data Structure	- Dimension	- Mô tả
- Series	- 1	- 1D array được gán nhãn đồng nhất, có kích thước không thay đổi
- Data Frame	- 2	- 2D array được gán nhãn, cấu trúc bảng có kích thước có thể thay đổi, các cột có khả năng không đồng nhất

5.2/ Ưu điểm:

- Hỗ trợ đa dạng dữ liệu
- Tích hợp dữ liệu
- Chuyển đổi dữ liệu
- Hỗ trợ dữ liệu time-series
- Thống kê mô tả

5.3/ Series:**a/ Khái niệm:**

- ❑ Series là mảng một chiều có gán nhãn, có khả năng chứa các phần tử với kiểu dữ liệu khác nhau (integer, string, float, python objects...).
- ❑ Các axis label của Series được gọi là index.

b/ Tạo Series:

- Phương thức tạo series

pandas.Series(data, index, dtype, copy)

- Trong đó:
 - Data: dữ liệu từ nhiều dạng khác nhau như ndarray, list, constants
 - Index: giá trị của index là duy nhất và số lượng index = số lượng phần tử. Mặc định sẽ có giá trị trong np.arange(n) nếu người dùng không tự tạo index.
 - Dtype: kiểu dữ liệu (tùy chọn)
 - Copy: tạo bản copy (tùy chọn)
- Tạo series từ ndarray, với index tự động

```

import numpy as np
nd_array_1 = np.array(['a','b','c','d'])
s1 = pd.Series(nd_array_1)
print(s1)

0    a
1    b
2    c
3    d
dtype: object

```

- Tạo series từ ndarray, với index do coder tạo

```

# Create a Series from ndarray
print(nd_array_1)
s2 = pd.Series(nd_array_1, index=[100,101,102,103])
print(s2)

['a' 'b' 'c' 'd']
100    a
101    b
102    c
103    d
dtype: object

```

- Tạo series từ dictionary

```

# Create a Series from dict
dic_1 = {'a' : 0., 'b' : 1., 'c' : 2.}
s3 = pd.Series(dic_1)
print(s3)

a    0.0
b    1.0
c    2.0
dtype: float64

```

5.4 Thao tác trên Series:

- Truy xuất phần tử trên series: theo index (label)

```

a    1
b    2
c    3
d    4
e    5
dtype: int64
Cách 1: 1
Cách 2: 1
Từ đầu đến cận 2:
a    1
b    2
dtype: int64
Từ 1 đến cận 3:
b    2
c    3
dtype: int64
Từ -3:
c    3
d    4
e    5
dtype: int64
Truy xuất nhiều phần tử theo index:
a    1
c    3
d    4
dtype: int64

```

```

# get element value from Series
s5 = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])
print(s5)
print('Cách 1:',s5['a'])
print('Cách 2:',s5[0])
print('Từ đầu đến cận 2:')
print(s5[:2])
print('Từ 1 đến cận 3:')
print(s5[1:3])
print('Từ -3:')
print(s5[-3:])
print('Truy xuất nhiều phần tử theo index:')
print(s5[['a','c','d']])

```

- Cập nhật phần tử

```

# cập nhật phần tử
s3 = pd.Series([12,21,11],index = ['idx1','idx2','idx3'])
s3['idx3'] = 100
print(s3)

```

```

idx1    12
idx2    21
idx3    100
dtype: int64

```

- Thêm phần tử

```
#Thêm phần tử vào Series - tương tự dictionary
print('Trước khi thêm:\n', s)
s['f'] = 6
print('Sau khi thêm:\n', s)
```

Trước khi thêm:

```
a    5
b    2
c    3
d    4
e    5
dtype: int32
Sau khi thêm:
```

a 5

b 2

c 3

d 4

e 5

f 6

dtype: int64

- Xóa phần tử

```
#Xóa phần tử
print(s)
t= s.drop(['a','e'])
print(t)
tt= s.drop('a')
print(tt)
```

```
a    5
b    2
c    10
d    4
e    5
f    6
dtype: int64
b    2
c    10
d    4
f    6
dtype: int64
b    2
c    10
d    4
e    5
f    6
dtype: int64
```

- Kiểm tra series rỗng: dùng empty

```
print(s1.empty)
```

False

```
print(s2.empty)
```

False

- Lấy giá trị của series dưới dạng array: dùng values

```
# get values as array
print(s1.values)

[-0.29088241  2.26262866 -0.04261633  0.64097325 -0.72238851]

print(s2.values)

[1 2 3 4 5]
```

- Lấy n dòng đầu(head)/dòng cuối(tail); dùng head(n)/tail(n)

<code>print(s1.head(2))</code>	<code>print(s2.tail(3))</code>
<code>0 -0.290882 1 2.262629 dtype: float64</code>	<code>c 3 d 4 e 5 dtype: int64</code>

- Xóa phần tử theo index/label

<code>a 1 b 2 c 3 d 4 e 5 dtype: int64</code>	<code>print(s2) s3 = s2.drop(s2.index[[0, 2, 4]]) print(s3)</code>	<code>b 2 d 4 dtype: int64</code>
	<code>s3 = s2.drop(['c', 'd']) print(s3)</code>	<code>a 1 b 2 e 5 dtype: int64</code>

- Tạo series mới bằng cách ánh xạ từ series đang có: dùng map(lambda)

<code>s2</code>	<code>s3 = s2.map(lambda x: x * x)</code>
<code>a 1 b 2 c 3 d 4 e 5 dtype: int64</code>	<code>a 1 b 4 c 9 d 16 e 25 dtype: int64</code>

5.5 Tổng hợp các hàm sử dụng

S.No.	Function	Description
1	count()	Trả về số lượng các phần tử khác null
2	sum()	Trả về tổng các giá trị
3	mean()	Trả về giá trị số trung bình của các giá trị
4	median()	Trả về giá trị trung bình giữa của các giá trị
5	mode()	Trả về giá trị có tần suất xuất hiện nhiều nhất
6	std()	Trả về độ lệch chuẩn của các giá trị
7	min()	Trả về giá trị nhỏ nhất
8	max()	Trả về giá trị lớn nhất
9	abs()	Trả về giá trị tuyệt đối
10	prod()	Product of Values
11	cumsum()	Trả về tổng tích lũy (Cumulative Sum)
12	cumprod()	Cumulative Product

- Thông tin thống kê chung: dùng describe()

```
print(s2.values)
print(s2.describe())

[-3.24  2.98  3.98  2.56  3.2   4.6   3.8   3.78  2.98 -4.8   4.1   3.65
 2.98]
count    13.000000
mean     2.351538
std      2.900390
min     -4.800000
25%     2.980000
50%     3.200000
75%     3.800000
max     4.600000
dtype: float64
```

5.6 Data Frame:

- ❑ DataFrame là một cấu trúc dữ liệu 2D (two-dimensional), dữ liệu được tổ chức theo dòng và cột.
- ❑ Đặc điểm
 - Các cột có nhiều kiểu dữ liệu khác nhau
 - Kích thước có thể thay đổi
 - Trục được gán nhãn (dòng và cột)
 - Có thể thực hiện các phép tính số học theo dòng và cột

5.6.1 Tạo DataFrame:

- ❑ Phương thức tạo Data frame:

pandas.DataFrame(data, index, columns, dtype, copy)

- Trong đó:
 - data: dữ liệu (list, dict, series, ndarray, dataframe)
 - index: danh sách các dòng; index = ['tên dòng 1', 'tên dòng 2']
 - columns: danh sách các cột; columns = ['tên cột 1', 'tên cột 2']
 - dtype: kiểu dữ liệu của các cột
 - copy: copy dữ liệu hay không, mặc định là False

- Tạo data frame từ list

```
lst = [1,2,3,4,5]
df1 = pd.DataFrame(lst)
print(df1)
```

		Name	Age
0	1	Bibi	7
1	2	Xuxu	5
2	3	Kent	10
3	4		
4	5		

```
lst2 = [['Bibi',7],['Xuxu',5],['Kent',10]]
df2 = pd.DataFrame(lst2,columns=['Name','Age'])
print(df2)
```

- Tạo data frame từ dictionary ndarray/list

```
dic1= {'Name':['Bibi', 'Xuxu', 'Kent', 'Mickey'],'Age':[7,5,10,3]}
df3 = pd.DataFrame(dic1, index=['sv1','sv2','sv3','sv4'], columns=['Name', 'Age'])
print(df3)
```

	Name	Age
sv1	Bibi	7
sv2	Xuxu	5
sv3	Kent	10
sv4	Mickey	3

- Tạo data frame từ list dictionary

```
# Create a DataFrame from List of Dicts
data = [{"Ten": "An", "Tuoi": 12, "Diem": 8.5}, {"Ten": "Hoa", "Tuoi": 12, "Diem": 7.5},
         {"Ten": "An", "Tuoi": 12}]
df = pd.DataFrame(data, index = ['st1', 'st2', 'st3'])
print(df)
```

	Diem	Ten	Tuoi
st1	8.5	An	12
st2	7.5	Hoa	12
st3	NaN	An	12

```
data = [{"a": 1, "b": 2}, {"a": 5, "b": 10, "c": 20}]
# 2 cột, tên cột là tên key của dictionary
df1 = pd.DataFrame(data, index=['first', 'second'], columns=['a', 'b'])

# 2 cột, tên một cột là tên key của dictionary, tên một cột khác
df2 = pd.DataFrame(data, index=['first', 'second'], columns=['a', 'b1'])
print(df1)
print(df2)
```

	a	b
first	1	2
second	5	10
	a	b1
first	1	NaN
second	5	NaN

- Tạo data frame từ dictionary series

```
# Create a DataFrame from Dict of Series
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
      'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd']),
      'three' : pd.Series([1, 2, 3], index=['a', 'b', 'c'])}
}

df5 = pd.DataFrame(d, columns = ['one', 'two', 'three'])
print(df5)
```

	one	two	three
a	1.0	1	1.0
b	2.0	2	2.0
c	3.0	3	3.0
d	NaN	4	NaN

5.6.2 Thao tác trên DataFrame:

- Lấy dữ liệu theo cột

```
# Lấy dữ liệu theo cột
print(df5)
print("Lấy cột 'one':")
print(df5['one'])
```

	one	two	three
a	1.0	1	1.0
b	2.0	2	2.0
c	3.0	3	3.0
d	NaN	4	NaN

Lấy cột 'one':

a	1.0
b	2.0
c	3.0
d	NaN

- Thêm cột

Name: one, dtype: float64

```
# Thêm cột mới
df5['four']=pd.Series([10,20,30, 40],
                      index=['a','b','c','d'])
print(df5)
```

	one	two	three	four
a	1.0	1	1.0	10
b	2.0	2	2.0	20
c	3.0	3	3.0	30
d	NaN	4	NaN	40

- Xóa cột dùng: dùng del/pop()

```
# using del function
print ("Dùng del:")
del df5['three']
print(df5)
```

Dùng del:

```
# using pop function
print ("Dùng pop")
df5.pop('one')
print(df5)
```

Dùng pop

Bộ Môn Máy Tính

- Xem danh sách các cột: dùng columns

	Name	Age
2	Alex	23.0
0	Jack	27.0
1	Clarke	32.0

```
print(df.columns)
Index(['Name', 'Age'], dtype='object')
```

- Lấy dữ liệu theo dòng

	one	two
a	1.0	1
b	2.0	2
c	3.0	3
d	NaN	4

```
# Lấy dữ liệu trên dòng 'b'
print(df.loc['b'])

# dùng index
df = pd.DataFrame(d)
print(df.iloc[1])

# chọn nhiều dòng
print(df[2:4])
```

	one	two
one	2.0	
two	2.0	
Name:	b, dtype: float64	

	one	two
c	3.0	3
d	NaN	4

- Cập nhật dữ liệu của dòng

	df	
	Name	Age
2	Alex	23.0
0	Jack	27.0
1	Clarke	32.0

```
# Cập nhật tuổi của dòng 1 thành 29
df.loc[1, 'Age'] = 29
df
```

	Name	Age
2	Alex	23.0
0	Jack	27.0
1	Clarke	29.0

- Lấy dữ liệu theo điều kiện

```

d = {'name' : pd.Series(['An', 'Hoa', 'Trung', 'Huy', 'Thanh', 'Dung'], index=[49,48,47, 1, 2, 3]),
      'age' : pd.Series([15, 22, 36,27,55],index=[49,48,47, 1, 3])}

df=pd.DataFrame(d,columns=['name','age'])
print('Dataframe\n',df)

print('Giá trị dòng 3- cột name là:',df.loc[3,'name'])

print('Lọc những nhân viên có tuổi >30: \n', df.loc[df['age']>30])

print('Lọc tên những nhân viên có tuổi >30: \n', df.loc[df['age']>30,'name'])

Dataframe
   name    age
1   Huy   27.0
2   Thanh  NaN
3   Dung   55.0
47  Trung  36.0
48  Hoa    22.0
49  An     15.0
Giá trị dòng 3- cột name là: Dung
Lọc những nhân viên có tuổi >30:
   name    age
3   Dung   55.0
47  Trung  36.0
Lọc tên những nhân viên có tuổi >30:
   3   Dung
  47  Trung
Name: name, dtype: object

```

- Thêm dòng: dùng append()

	one	two	# thêm dòng dùng append()
a	1.0	1	df2 = pd.DataFrame([{ 'one': 1, 'two': 2},{'one': 5, 'two': 10}], index = ['e','f'])
b	2.0	2	df = df.append(df2)
c	3.0	3	print(df)
d	NaN	4	
e			
f			

one two

a 1.0 1
b 2.0 2
c 3.0 3
d NaN 4
e 1.0 2
f 5.0 10

- Xóa dòng: dùng drop()

```

# xóa dòng theo Label
df = df.drop(['a','b'])
print(df)

```

	one	two
c	3.0	3
d	NaN	4
e	1.0	2
f	5.0	10

- `print(df.sort_index())` dùng sort_index()

	Name	Age
0	Jack	27.0
1	Clarke	32.0
2	Alex	23.0

	Name	Age
2	Alex	23.0
0	Jack	27.0
1	Clarke	32.0

- Sắp xếp theo value: dùn

```
print(df.sort_values(by='Name'))
```

	Name	Age
2	Alex	23.0
1	Clarke	32.0
0	Jack	27.0

- Xem thông tin Data Frame: dùng info()

```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3 entries, 2 to 1
Data columns (total 2 columns):
Name    3 non-null object
Age     3 non-null float64
dtypes: float64(1), object(1)
memory usage: 60.0+ bytes
None
```

- Kiểm tra bất kỳ phần tử nào đều là True: dùng any()

```
>>> df = pd.DataFrame({"A": [1, 2], "B": [0, 2], "C": [0, 0]})  
>>> df  
   A  B  C  
0  1  0  0  
1  2  2  0
```

```
>>> df.any()  
A    True  
B    True  
C   False  
dtype: bool
```

- Kiểm tra mọi phần tử đều là True: dùng all()

```
>>> df = pd.DataFrame({'col1': [True, True], 'col2': [True, False]})  
>>> df  
   col1  col2  
0  True  True  
1  True  False
```

5.6.3 Thống kê các hàm

S.No.	Function	Description
1	count(axis={0 hoặc 1})	Trả về số lượng các phần tử khác null
2	sum(axis={0 hoặc 1})	Trả về tổng các giá trị
3	.mean(axis={0 hoặc 1})	Trả về giá trị số trung bình của các giá trị
4	median(axis={0 hoặc 1})	Trả về giá trị trung bình giữa của các giá trị
5	mode(axis={0 hoặc 1})	Trả về giá trị có tần xuất xuất hiện nhiều nhất

6	<code>std(axis={0 hoặc 1})</code>	Trả về độ lệch chuẩn của các giá trị
7	<code>min(axis={0 hoặc 1})</code>	Trả về giá trị nhỏ nhất
8	<code>max(axis={0 hoặc 1})</code>	Trả về giá trị lớn nhất
9	<code>abs()</code>	Trả về giá trị tuyệt đối
10	<code>prod()</code>	Product of Values
11	<code>cumsum()</code>	Trả về tổng tích lũy (Cumulative Sum)
12	<code>cumprod()</code>	Cumulative Product

- Thông tin thống kê chung: dùng `describe()`

```
print(df.describe())
```

	Age	Rating
count	12.000000	12.000000
mean	31.833333	3.743333
std	9.232682	0.661628
min	23.000000	2.560000
25%	25.000000	3.230000
50%	29.500000	3.790000
75%	35.500000	4.132500
max	51.000000	4.800000

```
print(df.describe(include='all'))
```

	Age	Name	Rating
count	12.000000	12	12.000000
unique	NaN	11	NaN
top	NaN	Thanh	NaN
freq	NaN	2	NaN
mean	31.833333	NaN	3.743333
std	9.232682	NaN	0.661628
min	23.000000	NaN	2.560000
25%	25.000000	NaN	3.230000
50%	29.500000	NaN	3.790000
75%	35.500000	NaN	4.132500
max	51.000000	NaN	4.800000

5.7 Trực quan hóa trong Pandas:

5.7.1 Giới thiệu :

Có thể trực quan hóa dữ liệu bằng biểu đồ:

Plotting

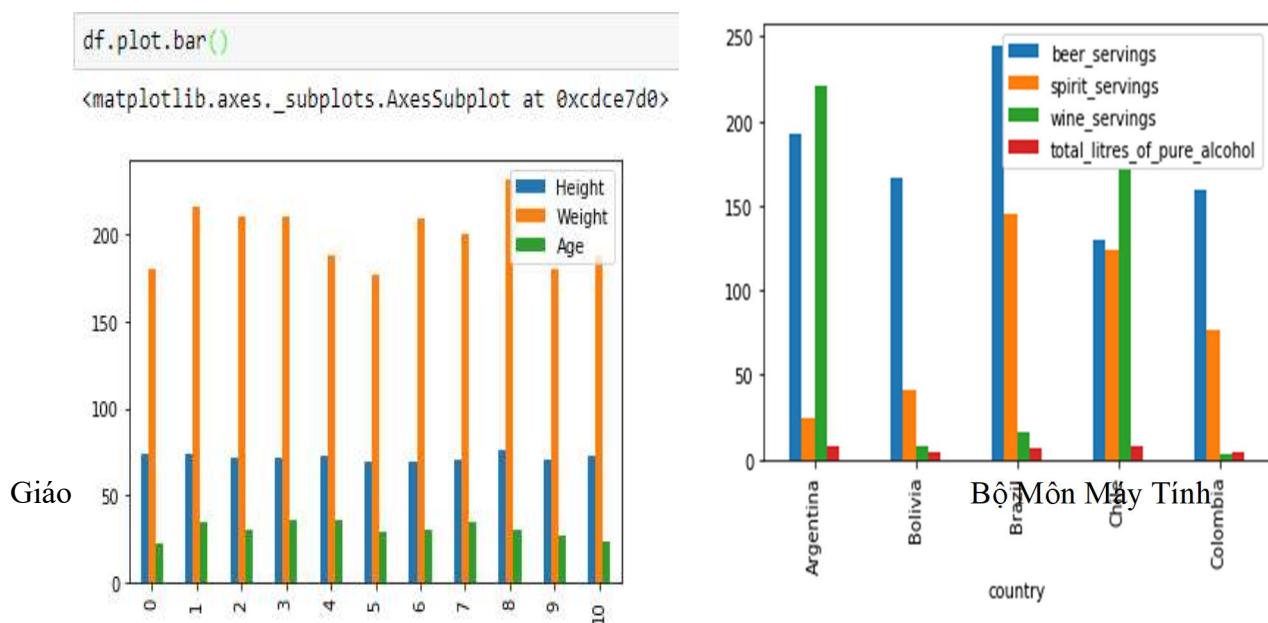
`DataFrame.plot` is both a callable method and a namespace attribute for specific plotting methods of the form `DataFrame.plot.<kind>`.

<code>DataFrame.plot([x, y, kind, ax, ...])</code>	DataFrame plotting accessor and method
<code>DataFrame.plot.area([x, y])</code>	Area plot
<code>DataFrame.plot.bar([x, y])</code>	Vertical bar plot.
<code>DataFrame.plot.bash([x, y])</code>	Make a horizontal bar plot.
<code>DataFrame.plot.box([by])</code>	Make a box plot of the DataFrame columns.
<code>DataFrame.plot.density([bw_method, ind])</code>	Generate Kernel Density Estimate plot using Gaussian kernels.
<code>DataFrame.plot.hexbin(x, y[, C, ...])</code>	Generate a hexagonal binning plot.
<code>DataFrame.plot.hist([by, bins])</code>	Draw one histogram of the DataFrame's columns.
<code>DataFrame.plot.kde([bw_method, ind])</code>	Generate Kernel Density Estimate plot using Gaussian kernels.
<code>DataFrame.plot.line([x, y])</code>	Plot DataFrame columns as lines.
<code>DataFrame.plot.pie([y])</code>	Generate a pie plot.
<code>DataFrame.plot.scatter(x, y[, s, c])</code>	Create a scatter plot with varying marker point size and color.
<code>DataFrame.boxplot([column, by, ax, ...])</code>	Make a box plot from DataFrame columns.
<code>DataFrame.hist([column, by, grid, ...])</code>	Make a histogram of the DataFrame's.

5.7.2 Bar plot:

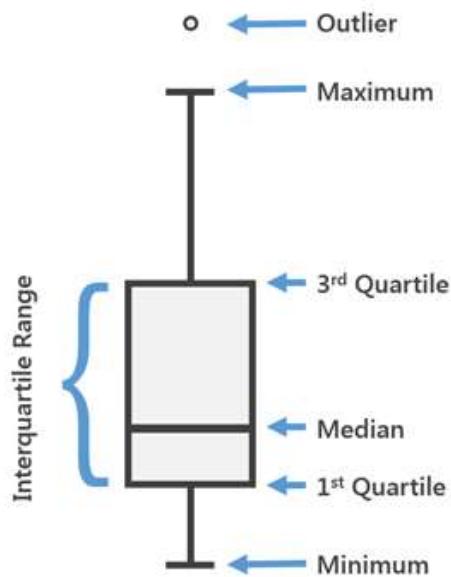
- Là dạng biểu đồ biểu diễn dữ liệu dưới dạng các khối chữ nhật, mô tả sự so sánh giữa các category khác nhau, một trục là category, một trục là giá trị của category đó.
- Tạo bar plot theo trục tung của Data frame columns: dùng `df.plot.bar()`

Ví dụ:



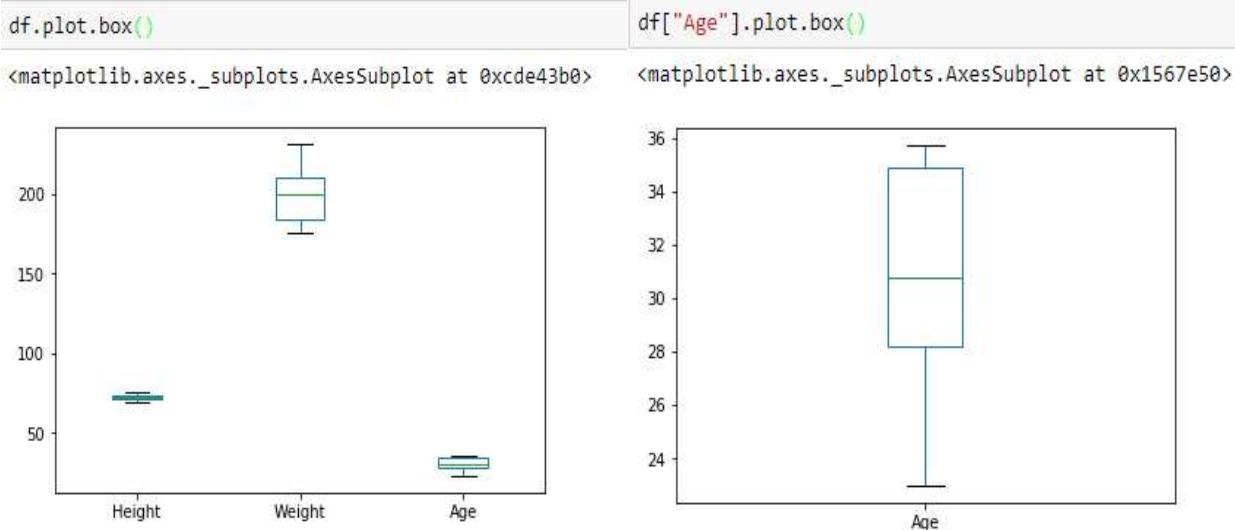
5.7.3 Boxplot:

- Là dạng biểu đồ box trên đó mô tả các giá trị của dữ liệu như sau:



- Tạo boxplot của Data frame columns: dùng **df.plot.box()**

Ví dụ:

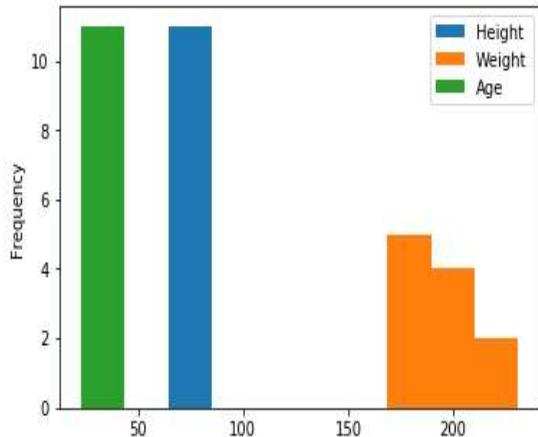


5.7.4 Biểu đồ histogram:

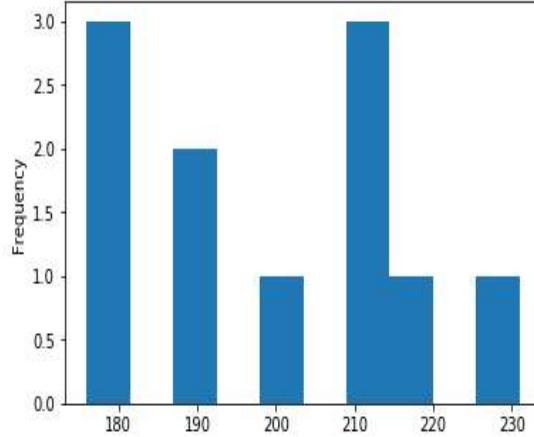
- Là dạng biểu đồ biểu diễn dạng phân phối tần suất của một tập dữ liệu liên tục.
- Tạo histogram của Data frame columns: dùng **df.plot.hist()**

Ví dụ:

```
df.plot.hist()
<matplotlib.axes._subplots.AxesSubplot at 0x5aeaef0>
```



```
df['Weight'].plot.hist()
<matplotlib.axes._subplots.AxesSubplot at 0xcb6bc70>
```

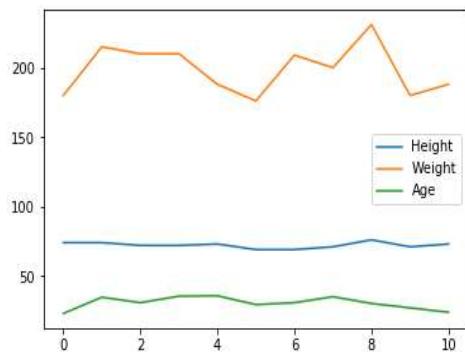


5.7.6 Biểu đồ plot:

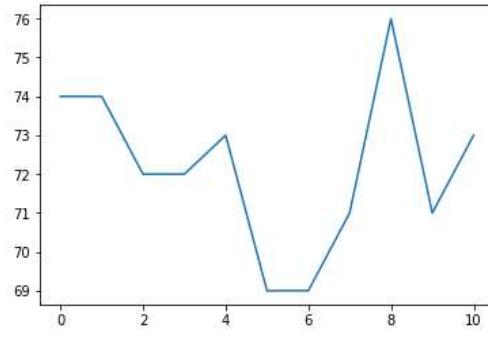
- Là dạng biểu đồ thể hiện các giá trị dưới các điểm nối với nhau bằng các line
- Tạo plot của Data frame columns: dùng **df.plot()**

Ví dụ:

```
df.plot()
<matplotlib.axes._subplots.AxesSubplot at 0xe179970>
```



```
df['Height'].plot()
<matplotlib.axes._subplots.AxesSubplot at 0x16e6d70>
```

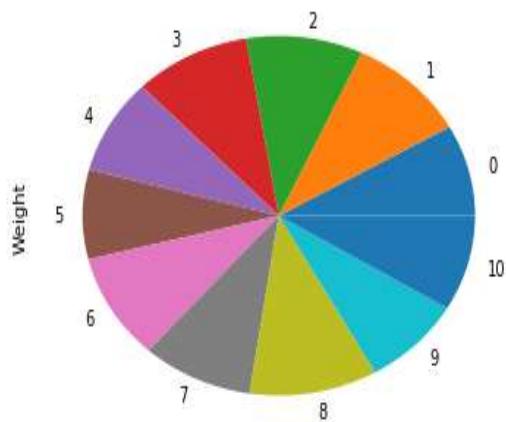


5.7.7 Biểu đồ Pie:

- Là dạng biểu đồ tròn mà mỗi giá trị là một miếng của biểu đồ tròn đó
- Tạo pie của Data frame column: dùng **df.plot.pie()**

Ví dụ:

```
df['Weight'].plot.pie()  
<matplotlib.axes._subplots.AxesSubplot at 0xcaa4430>
```



BÀI TẬP:

BT1: Thực hiện những yêu cầu liên quan đến series

Part 1: Thực hiện các phép toán trên series

Câu 1:

- a/ Cho arr_1 là mảng số nguyên chẵn [2, 4, 6, 8, 10], arr_2 là mảng số nguyên lẻ [1, 3, 5, 7, 11]. Tạo biến kiểu Serries ser1 từ arr_1, ser2 từ arr_2. In danh sách các phần tử của ser1 và ser2
- b/ Thực hiện phép toán và thể hiện kết quả của: ser1 + ser2
- c/ Thực hiện phép toán và thể hiện kết quả của: ser1 - ser2
- d/ Thực hiện phép toán và thể hiện kết quả của: ser1 * ser2
- e/ Thực hiện phép toán và thể hiện kết quả của: ser1 / ser2

Câu 2

- a/ Kiểm tra xem các phần tử của ser1 có > các phần tử của ser2 không?
- b/ Kiểm tra xem các phần tử của ser1 có < các phần tử của ser2 không?
- c/ Kiểm tra xem các phần tử của ser1 có = các phần tử của ser2 không?

Câu 3

- a/ Thêm 2 phần tử [6, 12] vào ser2. In lại danh sách các phần tử của ser2.
- b/ Tạo series ser3 chỉ chứa các phần tử có trong ser1 mà không có trong ser2. In danh sách các phần tử của ser3
- c/ Tạo series ser4 chỉ chứa các phần tử có trong ser2 mà không có trong ser1. In danh sách các phần tử của ser4

Câu 4: Tạo series ser5 chứa các phần tử chỉ có trong ser1 và chỉ có trong ser2. In danh sách các phần tử của ser5

Part 2: Truy xuất các phần tử, và thống kê thông tin trên series

Câu 1

a/ Tạo series ser6 có 35 phần tử số nguyên ngẫu nhiên có giá trị trong khoảng từ 1 đến 9. Cho biết kích thước (shape) của ser6. Xem 5 dòng dữ liệu đầu tiên (head) và 5 dòng dữ liệu cuối cùng (tail) có trong ser6

b/ In danh sách các phần tử của ser6 theo dạng array

c/ Cho biết thông tin thống kê chung (describe()) của ser6

d/ Cho biết tổng của các phần tử có trong ser6

e/ Cho biết phần tử có tần suất xuất hiện nhiều nhất trong ser6

Câu 2: Liệt kê các dòng trong ser6 mà giá trị chia hết cho 2 và cho 3

Câu 3: In các phần tử ở vị trí 0, 5, 10, 15 trong ser6

Câu 4: In ra các giá trị unique (array) trong ser6

Câu 5: Tạo series ser7 với mỗi phần tử có giá trị = lập phương của phần tử trong ser6. Xem 5 dòng dữ liệu đầu tiên (head) của ser7

Part 3: Tạo series từ list, chuỗi và biểu thức điều kiện

Câu 1: Cho list sau:

```
lst = ["abc", "defg", "htlmj", "dfg", "ljsac"]
```

a/ Tạo series ser_chuoi từ lst

b/ Tạo series ser_dodai với mỗi phần tử có giá trị là chiều dài của mỗi phần tử trong ser_chuoi

Câu 2: Cho ser = pd.Series(np.array([1, 2, 4, 5, 8, 7, 6, 9])). Sử dụng biểu thức điều kiện thích hợp để in ra các dòng trong ser có giá trị là số nguyên tố

Câu 3: Cho mẫu email như sau:

```
pattern = '[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,4}'
```

Tạo một series ser_ch, với mỗi phần tử trong ser_ch là một chuỗi

Gợi ý: 'reading newspaper from tuoitre.vn', 'tubirona@gmail.com', 'nguyen.nn@yahoo.com',

'tran_2014@hotmail.com.vn'

In ra những dòng trong ser_ch thỏa điều kiện chuỗi là email

Câu 4: Cho series:

```
ser_names = pd.Series(['Manufacturer', 'Model', 'CarType', 'Min_Price', 'Price', 'Max_Price',
'MPG_city', 'MPG_highway', 'AirBags', 'DriveTrain', 'Cylinders',
'EngineSize', 'Horsepower', 'RPM', 'Rev_per_mile', 'Man_trans_avail',
'Fuel_tank_capacity', 'Passengers', 'Length', 'Wheelbase', 'Width',
'Turn_circle', 'Rear_seat_room', 'Luggage_room', 'Weight', 'Origin',
'Make'])
```

Sử dụng biểu thức điều kiện thích hợp để in ra các dòng của ser_names thỏa điều kiện trong chuỗi có chữ 'Price'

BT2:

Cho tập tin dữ liệu auto.csv, và list headers sau:

```
headers = ["symboling","normalized-losses","make","fuel-type","aspiration", "num-of-doors","body-style",
"drive-wheels","engine-location","wheel-base", "length","width","height","curb-weight","engine-type",
"num-of-cylinders", "engine-size","fuel-system","bore","stroke","compression-ratio",
"horsepower","peak-rpm","city-mpg","highway-mpg","price"]
```

Câu 1: Đọc dữ liệu từ tập tin auto.csv, kết hợp với list headers đã cho để tạo thành dataframe df có tiêu đề hoàn chỉnh. Liệt kê kiểu dữ liệu các cột của df. Liệt kê 5 dòng dữ liệu đầu tiên (head) của df

Câu 2: Trong tập dữ liệu, có một số dữ liệu bị thiếu đi kèm với dấu ?, bạn hãy thay thế các dữ liệu này thành NaN. In lại 5 dòng dữ liệu đầu tiên của df

Câu 3: Đổi kiểu dữ liệu các cột thành kiểu dữ liệu đúng theo mô tả sau

```
# 1. make: alfa-romero, audi, bmw, chevrolet, dodge, honda, isuzu, jaguar, mazda, mercedes-benz,
porsche,...
```

```
# 2. fuel-type: diesel, gas.
```

```
# 3. num-of-doors: four, two.
```

```
# 4. body-style: hardtop, wagon, sedan, hatchback, convertible.
```

```
# 5. drive-wheels: 4wd, fwd, rwd.
```

6. engine-location: front, rear.
7. wheel-base: continuous from 86.6 120.9.
8. length: continuous from 141.1 to 208.1.
9. width: continuous from 60.3 to 72.3.
10. height: continuous from 47.8 to 59.8.
11. curb-weight: continuous from 1488 to 4066.
12. engine-size: continuous from 61 to 326.
13. bore: continuous from 2.54 to 3.94.
14. stroke: continuous from 2.07 to 4.17.
15. horsepower: continuous from 48 to 288.
16. peak-rpm: continuous from 4150 to 6600.
17. city-mpg: continuous from 13 to 49.
18. highway-mpg: continuous from 16 to 54.
19. price: continuous from 5118 to 45400.

Câu 4: Xóa bỏ các cột không cần thiết, chỉ giữ lại các cột:

```
# "make","fuel-type","num-of-doors","body-style","drive-wheels","engine-location","wheel-base",
"length","width","height","curb-weight","engine size","bore","stroke","horsepower","peak-rpm","city-
mpg","highway-mpg","price"
```

Câu 5: Liệt kê số dòng có giá trị null trong các thuộc tính

Câu 6: Nếu cột 'num-of-doors' nếu có chứa giá trị NaN thì thay bằng giá trị phổ biến nhất của cột 'num-of-doors'

Câu 7: Xóa tất cả các dòng có giá trị NaN trong cột 'price' (nếu có)

Câu 8: Các cột kiểu số khác nếu chứa giá trị NaN thì thay thế bằng giá trị trung bình

Câu 9: Đặt lại chỉ mục (index) sau khi đã bỏ các dòng có giá trị NaN trong cột 'price' và các cập nhật trên dữ liệu

Câu 10: In thông tin thống kê chung của df

Câu 11: Cho biết thông tin thống kê chung của các cột có kiểu dữ liệu là object.

Câu 12: Cho biết số lượng của mỗi loại hệ thống dẫn động (drive-wheels)?

Câu 13: Cho biết số lượng của mỗi loại động cơ (engine-location) ?

Câu 14: Cho biết các giá trị của cột drive-wheels ?

Câu 15: Tính giá (price) trung bình của mỗi loại hệ dẫn động (drive-wheels).

Câu 16: Tính giá (price) trung bình của mỗi chiếc xe dựa vào kiểu dáng thân xe (body-style).

Câu 17: Tính giá (price) trung bình mỗi kiểu dáng thân xe của các loại hệ dẫn động (drive-wheels).

Câu 18: Từ câu trên, sử dụng hàm pivot để tạo bảng từ các nhóm cho dễ xem hơn.

(drive-wheel thể hiện giá trị các dòng, body-style thể hiện giá trị các cột)

Câu 19: Vẽ biểu đồ thể hiện phân bố dữ liệu của cột price (histogram)

Câu 20: Thể hiện hệ số tương quan của df

Câu 21: Vẽ biểu đồ thể hiện giá trị giữa 'engine-size' và 'price'

Câu 22: Vẽ biểu đồ boxplot thể hiện giá xe thay đổi như thế nào với từng loại body-style

Chương 6: Web scraping

Nội Dung

6.1 Giới thiệu thư viện BeautifulSoup:	109
6.2 Tạo đối tượng beautifulsoup	109
6.3 Tag	109
6.3.1 Name.....	110
6.3.2 Attributes	110
6.4 Tìm kiếm trong cây	113
6.5 Các loại bộ lọc	115
6.5.1 Chuỗi	115
6.5.2 Regular Expression.....	115
6.5.3 List.....	116
6.5.4 True.....	116
6.5.5 find_all()	117
6.5.6 Name argument	118
6.5.7 Keyword arguments.....	118
6.5.8 Tìm kiếm bằng CSS.....	120
6.5.9 String argument	123
6.5.10 Limit argument	124
6.5.11 Recursive argument	124
6.5.12 Gọi một thẻ như gọi find_all().....	126
6.6 CSS selectors	127

6.7 Out put 132.....	133
6.7.1 Xuất định dạng đúng chuẩn	133
6.7.2 Không cần đúng chuẩn	135
6.8 Định dạng đầu ra	136
6.9 get_text()	141
Bài tập	143

6.1 Giới thiệu thư viện BeautifulSoup:

BeautifulSoup là một thư viện Python dùng để lấy dữ liệu ra khỏi các file HTML và XML. Nó hoạt động cùng với các parser (trình phân tích cú pháp) cung cấp cho bạn các cách để điều hướng, tìm kiếm và chỉnh sửa trong parse tree (cây phân tích được tạo từ parser). Nhờ các parser này nó đã giúp các lập trình viên tiết kiệm được nhiều giờ làm việc.

Cài đặt thư viện beautiful soup:

Pip install beautifulsoup4

6.2 Tạo đối tượng beautifulsoup:

Để bóc tách hay parse một tài liệu HTML, ta cần cho nó vào hàm BeautifulSoup() – hàm này sẽ trả về một BeautifulSoup object. Bạn có thể truyền cho nó một File object đang mở hoặc một chuỗi html.

```
from bs4 import BeautifulSoup

with open("index.html") as fp:
    soup = BeautifulSoup(fp)

soup = BeautifulSoup("<html>data</html>")
```

Đầu tiên, tài liệu được chuyển đổi thành Unicode, và các phần tử HTML được chuyển đổi sang ký tự Unicode:

```
BeautifulSoup("Sacr&eacute; bleu!")
<html><head></head><body>Sacré bleu!</body></html>
```

Soup được băm bằng parser tốt nhất hiện đang có. Ta có các parser và các cách sử dụng của nó.

lxml, html.parser, xml, html5lib

6.3 Tag:

Một Tag object tương ứng với một tag (còn được gọi là một thẻ hoặc một phần tử - element) trong tài liệu html hay xml:

```
soup = BeautifulSoup('<b class="boldest">Extremely bold</b>')
tag = soup.b
type(tag)
# <class 'bs4.element.Tag'>
```

Một Tag có chứa nhiều thuộc tính và phương thức, nhưng hai thuộc tính quan trọng nhất của một Tag đó là Name và Attributes.

6.3.1/ Name:

Tag nào cũng có tên, ta có thể lấy nó ra thông qua thuộc tính .name:

```
tag.name
# u'b'
```

Nếu bạn thay đổi tên của Tag, thay đổi này sẽ được lưu lại trên tài liệu HTML được băm bởi BeautifulSoup:

```
tag.name = "blockquote"
tag
# <blockquote class="boldest">Extremely bold</blockquote>
```

6.3.2 Attributes:

Một tag có thể chứa nhiều thuộc tính. Ví dụ như tag `<b id="boldest">` có thuộc tính id và giá trị là boldest bằng cách xử lý tag như một dict, bạn có thể dễ dàng lấy ra giá trị của một thuộc tính khi có được tên của thuộc tính đó.

```
tag['id']
# u'boldest'
```

Bạn có thể truy cập dict trực tiếp bằng .attrs :

```
tag.attrs
# {u'id': 'boldest'}
```

Bạn có thể thêm, sửa, xóa các thuộc tính của tag. Một lần nữa, ta xử lý tag như một dict thông thường:

```
tag['id'] = 'verybold'
tag['another-attribute'] = 1
tag
# <b another-attribute="1" id="verybold"></b>
del tag['id']
del tag['another-attribute']
tag
# <b></b>
tag['id']
# KeyError: 'id'
print(tag.get('id'))
# None
```

Thuộc tính có nhiều giá trị:

Trong HTML4 định nghĩa một vài thuộc tính mà trong nó có thể chứa một lúc nhiều giá trị. Trong HTML5 thì đã loại bỏ một vài cái trong số chúng. Thuộc tính nhiều giá trị mà chúng ta thường gặp nhất là class, những thuộc tính khác bao gồm rel, rev, accept-charset, headers và accesskey. BeautifulSoup sẽ chuyển những giá trị của các thuộc tính này vào một list, và ta có thể xử lý chúng như một list thông thường:

```
css_soup = BeautifulSoup('<p class="body"></p>')
css_soup.p['class']
# ["body"]

css_soup = BeautifulSoup('<p class="body strikeout"></p>')
```

```
css_soup.p['class']
```

```
# ["body", "strikeout"]
```

Nếu một thuộc tính trông giống như là một thuộc tính nhiều giá trị. Nhưng nếu thuộc tính đó không được định nghĩa trong bất cứ phiên bản HTML tiêu chuẩn nào thì BeautifulSoup sẽ để giá trị thuộc tính đó một mình mà không tách ra thành một list như ở bên trên:

```
id_soup = BeautifulSoup('<p id="my id"></p>')
```

```
id_soup.p['id']
```

```
# 'my id'
```

Khi bạn biến một tag trở về thành một chuỗi, thì các giá trị của thuộc tính sẽ được hợp lại:

```
rel_soup = BeautifulSoup('<p>Back to the <a rel="index">homepage</a></p>')
```

```
rel_soup.a['rel']
```

```
# ['index']
```

```
rel_soup.a['rel'] = ['index', 'contents']
```

```
print(rel_soup.p)
```

```
# <p>Back to the <a rel="index contents">homepage</a></p>
```

Bạn có thể vô hiệu hóa điều này bằng cách truyền vào multi_valued_attributes=None như một keyword argument trong BeautifulSoup constructor:

```
no_list_soup = BeautifulSoup('<p class="body strikeout"></p>', 'html', multi_valued_attributes=None)
```

```
no_list_soup.p['class']
```

```
# u'body strikeout'
```

Bạn có thể dùng get_attribute_list để nhận giá trị luôn là một list, bất kể nó có là thuộc tính đa giá trị hay không.

```
id_soup.p.get_attribute_list('id') # ["my id"]
```

Nếu bạn bấm tài liệu như một XML, nó sẽ không còn là thuộc tính đa giá trị nữa:

```
xml_soup = BeautifulSoup('<p class="body strikeout"></p>', 'xml')
```

```
xml_soup.p['class']
```

```
# u'body strikeout'
```

Một lần nữa, bạn có thể cấu hình điều này bằng cách sử dụng đối số multi_valued_attributes:

```
class_is_multi= { '*' : 'class'}
```

```
xml_soup = BeautifulSoup('<p class="body strikeout"></p>', 'xml',
multi_valued_attributes=class_is_multi)
```

```
xml_soup.p['class']
```

```
# [u'body', u'strikeout']
```

Có thể bạn sẽ không cần phải làm điều này, nhưng nếu bạn làm thế, sử dụng giá trị mặc định như hướng dẫn. Chúng thực hiện các quy tắc được mô tả trong đặc tả HTML:

```
from bs4.builder import builder_registry
```

```
builder_registry.lookup('html').DEFAULT_CDATA_LIST_ATTRIBUTES
```

6.4 Tìm kiếm trong cây

Beautiful Soup định nghĩa rất nhiều phương thức dùng để tìm kiếm trong parse tree, nhưng chúng rất giống nhau, phổ biến nhất: find() và find_all().

Một lần nữa, ta sẽ sử dụng tài liệu “three sisters” làm ví dụ:

```
html_doc = """
```

```
<html><head><title>The Dormouse's story</title></head>
```

```
<body>
```

```
<p class="title"><b>The Dormouse's story</b></p>
```

```
<p class="story">Once upon a time there were three little sisters; and their names were
```

```

<a href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0L1x0DM4mpSt97%2ftYgbxlC2B7n4pvJNhvvRwo8bxjO4B" class="sister" id="link1">Elsie</a>,
<a href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0OPun6GIXb9bh0UOl0N9WCYbJtHZQd%2fvB08D2UeudkPP" class="sister" id="link2">Lacie</a> and
<a href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0LirHL60gbBHH3VIishi9CqgtHAKmbGoKNvFheNkumnh" class="sister" id="link3">Tillie</a>;
and they lived at the bottom of a well.</p>

<p class="story">...</p>
"""""
from bs4 import BeautifulSoup
soup = BeautifulSoup(html_doc, 'html.parser')

```

Bằng cách truyền vào một hàm như `find_all()` một bộ lọc với đối số, ta có thể đi tới một phần trong tài liệu mà bạn quan tâm.

6.5 Các loại bộ lọc:

6.5.1 Chuỗi:

Đây là bộ lọc đơn giản nhất. Truyền một chuỗi vào một phương thức tìm kiếm và BeautifulSoup sẽ thực hiện so khớp chính xác với chuỗi đó. Đoạn code này tìm tất cả thẻ có trong tài liệu:

```
soup.find_all('b')

# [<b>The Dormouse's story</b>]
```

Nếu bạn truyền vào một byte string, BeautifulSoup sẽ cho rằng chuỗi được mã hóa UTF-8. Thay vào đó bạn có thể tránh điều này bằng cách truyền vào một Unicode string thay thế.

6.5.2 Regular Expression:

Nếu bạn truyền vào một regular expression object, BeautifulSoup sẽ lọc theo regular expression đó bằng cách sử dụng phương thức search() của nó. Đoạn code này sẽ tìm tất cả các thẻ có tên bắt đầu bằng chữ “b”; trong trường hợp này là thẻ và thẻ <body>:

```
import re

for tag in soup.find_all(re.compile("^b")):
    print(tag.name)

# body

# b
```

Đoạn code này sẽ tìm tất cả các thẻ mà trong tên có chữ “t”:

```
for tag in soup.find_all(re.compile("t")):
    print(tag.name)

# html

# title
```

6.5.3 List:

Nếu bạn truyền vào một list, Beautiful Soup sẽ cho phép một chuỗi match bất kì phần tử nào có trong list đó. Code này sẽ tìm tất cả thẻ `<a>` và thẻ ``:

```
soup.find_all(["a", "b"])

# [<b>The Dormouse's story</b>,

# <a class="sister"
href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0L1x0DM4mpSt97%2ftYgbxlC2B7n4pvJNhvRwo8b
xiO4B" id="link1">Elsie</a>,
# <a class="sister"
href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0OPun6GIXb9bh0UOl0N9WCYbJtHZQd%2fvB08D2
UeudkPP" id="link2">Lacie</a>,
# <a class="sister"
href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0LirHL60gbBHH3VIishi9CqgtHAKmbGoKNvFheNku
mnh" id="link3">Tillie</a>]
```

6.5.4 True:

Giá trị True sẽ so khớp tất cả. Đoạn code này sẽ tìm tất cả thẻ có trong tài liệu, nhưng không có text string nào:

```
for tag in soup.find_all(True):
    print(tag.name)

# html
# head
# title
# body
# p
# b
```

```
# p
# a
# a
# a
# p
```

6.5.5 find_all()

`find_all(name, attrs, recursive, string, limit, **kwargs)`

Phương thức `find_all()` sẽ duyệt qua tất cả các thẻ con và lấy tất cả các thẻ mà phù hợp với bộ lọc của bạn. Mình sẽ đưa ra một số ví dụ về các loại bộ lọc, như ví dụ dưới đây:

`soup.find_all("title")`

```
# [<title>The Dormouse's story</title>]
```

`soup.find_all("p", "title")`

```
# [<p class="title"><b>The Dormouse's story</b></p>]
```

`soup.find_all("a")`

```
# [<a class="sister"
```

```
href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0L1x0DM4mpSt97%2ftYgbxlC2B7n4pvJNhhvRwo8b
xiO4B" id="link1">Elsie</a>,
```

```
# <a class="sister"
```

```
href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0OPun6GIXb9bh0UOl0N9WCYbJtHZQd%2fvB08D2
UeudkPP" id="link2">Lacie</a>,
```

```
# <a class="sister"
```

```
href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0LirHL60gbBHH3Viishi9CqgtHAKmbGoKNvFheNku
mnh" id="link3">Tillie</a>]
```

```
soup.find_all(id="link2")

# [<a class="sister"
href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0OPun6GIXb9bh0UOloN9WCYbJtHZQd%2fvB08D2
UeudkPP" id="link2">Lacie</a>]

import re

soup.find(string=re.compile("sisters"))

# u'Once upon a time there were three little sisters; and their names were\n'
```

6.5.6 Name argument:

Truyền một giá trị vào name và bạn sẽ nói cho Beautiful Soup biết là chỉ xem xét những thẻ có tên là giá trị được truyền vào name. Các text string sẽ bị bỏ qua cũng như các thẻ có tên không khớp.

Đây là cách sử dụng đơn giản nhất:

```
soup.find_all("title")

# [<title>The Dormouse's story</title>]
```

Xem lại phần Các loại bộ lọc, giá trị truyền vào name có thể là một chuỗi, biểu thức chính quy, list, function, hoặc True.

6.5.7 Keyword arguments

Bất kỳ đối số nào không có trong hàm find_all() đều sẽ được chuyển thành một dạng bộ lọc trong thuộc tính của thẻ. Nếu bạn truyền vào một giá trị cho một đối số được gọi là id, Beautiful Soup sẽ lọc các thẻ có thuộc tính id với giá trị được chỉ định:

```
soup.find_all(id='link2')

# [<a class="sister"
href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0OPun6GIXb9bh0UOloN9WCYbJtHZQd%2fvB08D2
UeudkPP" id="link2">Lacie</a>]
```

Nếu bạn truyền một giá trị vào href, Beautiful Soup sẽ lọc dựa theo thuộc tính href:

```
soup.find_all(href="/redirect?Id=VV4DCjRYHnaSds2LlPoP7D5C1bH%2bvGxljiVtPSdQHhws6gq91nZVQ8U74LNWms1V"
```

```
# [<a class="sister"
 href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0L1x0DM4mpSt97%2ftYgbxlC2B7n4pvJNhvvRwo8bxO4B" id="link1">Elsie</a>]
```

Bạn cũng có thể lọc các thuộc tính dựa theo một chuỗi, biểu thức chính quy, list, function, hay bằng giá trị True:

Đoạn code này sẽ tìm tất cả các thẻ mà trong nó có thuộc tính id, bắt đầu giá trị trong nó là gì:

```
soup.find_all(id=True)

# [<a class="sister"
 href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0L1x0DM4mpSt97%2ftYgbxlC2B7n4pvJNhvvRwo8bxO4B" id="link1">Elsie</a>,

# <a class="sister"
 href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0OPun6GIXb9bh0UOloN9WCYbJtHZQd%2fvB08D2UeudkPP" id="link2">Lacie</a>,

# <a class="sister"
 href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0LirHL60gbBHH3VIishi9CqgtHAKmbGoKNvFheNkumnh" id="link3">Tillie</a>]
```

Bạn có thể lọc nhiều thuộc tính cùng một lúc bằng cách truyền vào nhiều hơn một keyword argument:

```
soup.find_all(href="/redirect?Id=VV4DCjRYHnaSds2LlPoP7IYWyJrlG8f8MIWKFhNe73RQef%2b%2fpwlqwlc%2bcVx%2bmwJJ" id='link1')

# [<a class="sister"
 href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0L1x0DM4mpSt97%2ftYgbxlC2B7n4pvJNhvvRwo8bxO4B" id="link1">three</a>]
```

Một vài thuộc tính, như data-* trong HTML 5, có tên không thể được sử dụng như tên của keyword arguments:

```
data_soup = BeautifulSoup('<div data-foo="value">foo!</div>')

data_soup.find_all(data-foo="value")

# SyntaxError: keyword can't be an expression
```

Bạn có thể sử dụng những thuộc tính này để lọc bằng cách đưa chúng vào một dict, dict này sẽ được truyền vào đối số attrs trong hàm find_all():

```
data_soup.find_all(attrs={"data-foo": "value"})

# [<div data-foo="value">foo!</div>]
```

Bạn không thể sử dụng một keyword argument để tìm phần tử name trong HTML, vì BeautifulSoup sử dụng đối số name để chứa tên của các thẻ. Thay vào đó, bạn hãy đưa nó vào một dict thông qua đối số attrs:

```
name_soup = BeautifulSoup('<input name="email"/>')

name_soup.find_all(name="email")

# []

name_soup.find_all(attrs={"name": "email"})

# [<input name="email"/>]
```

6.5.8 Tìm kiếm bằng class CSS

Thật hữu ích khi có thể tìm kiếm một tag bằng một class CSS cụ thể. Nhưng tên của thuộc tính CSS, “class”, nó trùng với từ khóa class trong Python. Sử dụng class như một keyword argument sẽ bắn ra syntax error. Từ Beautiful Soup 4.1.2, bạn có thể tìm kiếm bằng CSS class bằng cách sử dụng keyword argument class_ :

```
soup.find_all("a", class_="sister")

# [<a class="sister"
    href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0L1x0DM4mpSt97%2ftYgbxlC2B7n4pvJNhhvRwo8b
    xiO4B" id="link1">Elsie</a>,

    # <a class="sister"
    href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0OPun6GIXb9bh0UOl0N9WCYbJtHZQd%2fvB08D2
    UeudkPP" id="link2">Lacie</a>,
```

```
# <a class="sister"
href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0LirHL60gbBHH3Viishi9CqgtHAKmbGoKNvFheNku
mnh" id="link3">Tillie</a>]
```

Giống như bất kỳ keyword argument nào, bạn có thể truyền vào class_ một chuỗi, một biểu thức chính quy, hàm hoặc True:

```
soup.find_all(class_=re.compile("itl"))
# [<p class="title"><b>The Dormouse's story</b></p>]
```

```
def has_sicharacters(css_class):
    return css_class is not None and len(css_class) == 6
```

```
soup.find_all(class_=has_sicharacters)
# [<a class="sister"
href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0L1x0DM4mpSt97%2ftYgbxlC2B7n4pvJNhvRwo8
bxIO4B" id="link1">Elsie</a>,
<a class="sister"
href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0OPun6GIXb9bh0UOloN9WCYbJtHZQd%2fvB08D
2UeudkPP" id="link2">Lacie</a>,
<a class="sister"
href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0LirHL60gbBHH3Viishi9CqgtHAKmbGoKNvFheN
kumnh" id="link3">Tillie</a>]
```

Hãy nhớ rằng, một thẻ đơn có thể có nhiều giá trị trong thuộc tính "class" của nó. Khi bạn tìm kiếm một thẻ match với CSS class chỉ định, bạn có thể match bất kỳ CSS class nào của nó.

```
css_soup = BeautifulSoup('<p class="body strikeout"></p>')
css_soup.find_all("p", class_="strikeout")
# [<p class="body strikeout"></p>]
```

```
css_soup.find_all("p", class_="body")
# [<p class="body strikeout"></p>]
```

Bạn cũng có thể tìm kiếm cách đưa ý nguyên giá trị của thuộc tính class vào:

```
css_soup.find_all("p", class_="body strikeout")
# [<p class="body strikeout"></p>]
```

Nhưng tìm ngược lại thì lại không hoạt động, ở đây chuỗi strikeout body bị đảo ngược so với body strikeout, cho nên nó không thể tìm thấy:

```
css_soup.find_all("p", class_="strikeout body")
# []
```

Nếu bạn muốn tìm kiếm thẻ khớp hai hay nhiều hơn class CSS, bạn có thể dùng CSS selector:

```
css_soup.select("p.strikeout.body")
# [<p class="body strikeout"></p>]
```

Trong các phiên bản cũ hơn của Beautiful Soup, thì không có shortcut class_, bạn có thể sử dụng mèo attrs được đề cập bên trên. Tạo một dict chứa key là "class" và value là một chuỗi (hoặc một regular expression, hay bất cứ thứ gì) bạn muốn tìm:

```
soup.find_all("a", attrs={"class": "sister"})
# [<a class="sister"
    href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0L1x0DM4mpSt97%2ftYgbxlC2B7n4pvJNhhvRwo8b
    xiO4B" id="link1">Elsie</a>,
    <a class="sister"
    href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0OPun6GIXb9bh0UOl0N9WCYbJtHZQd%2fvB08D2
    UeudkPP" id="link2">Lacie</a>,
    <a class="sister"
    href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0LirHL60gbBHH3VIishi9CqgtHAKmbGoKNvFheNku
    mnh" id="link3">Tillie</a>]
```

6.5.9 String argument

Với string bạn có thể tìm kiếm chuỗi thay cho thẻ. Như name và keyword arguments, bạn có thể truyền vào một chuỗi, một biểu thức chính quy, một list, một hàm, hoặc giá trị True. Dưới đây là một ví dụ:

```
soup.find_all(string="Elsie")
# [u'Elsie']

soup.find_all(string=["Tillie", "Elsie", "Lacie"])
# [u'Elsie', u'Lacie', u'Tillie']

soup.find_all(string=re.compile("Dormouse"))
[u"The Dormouse's story", u"The Dormouse's story"]

def is_the_only_string_within_a_tag(s):
    """Return True if this string is the only child of its parent tag."""
    return (s == s.parent.string)

soup.find_all(string=is_the_only_string_within_a_tag)
# [u"The Dormouse's story", u"The Dormouse's story", u'Elsie', u'Lacie', u'Tillie', u'...']
```

Mặc dù string là để tìm chuỗi, bạn có thể kết hợp nó với các đối số tìm kiếm khác: BeautifulSoup sẽ tìm tất cả các thẻ có .string khớp với giá trị string. Đoạn code này sẽ tìm những thẻ <a> có .string là "Elsie":

```
soup.find_all("a", string="Elsie")
```

```
# [<a href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0L1x0DM4mpSt97%2ftYgbxlC2B7n4pvJNhvvRwo8bxio4B" class="sister" id="link1">Elsie</a>]
```

string là một đối số mới trong BeautifulSoup 4.4.0. Trong các phiên bản trước đó nó được gọi là text:

```
soup.find_all("a", text="Elsie")
```

```
# [<a href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0L1x0DM4mpSt97%2ftYgbxlC2B7n4pvJNhvvRwo8bxio4B" class="sister" id="link1">Elsie</a>]
```

6.5.10 Limit argument

`find_all()` trả về tất cả những thẻ và chuỗi khớp với bộ lọc của bạn. Điều này có thể khá tốn thời gian nếu tài liệu của bạn lớn. Nếu bạn không cần lấy hết kết quả, bạn có thể truyền thêm một số cho đối số limit. Nó hoạt động giống như keyword LIMIT trong SQL. Nó nói cho BeautifulSoup biết rằng ngừng thu thập kết quả sau khi lấy được một số lượng nhất định.

Có ba link trong tài liệu "three sisters", nhưng đoạn code này chỉ tìm thấy hai:

```
soup.find_all("a", limit=2)
```

```
# [<a class="sister" href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0L1x0DM4mpSt97%2ftYgbxlC2B7n4pvJNhvvRwo8bxio4B" id="link1">Elsie</a>, <a class="sister" href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0OPun6GIXb9bh0UOloN9WCYbJtHZQd%2fvB08D2UeudkPP" id="link2">Lacie</a>]
```

6.5.11 Recursive argument

Nếu bạn gọi `mytag.find_all()`, BeautifulSoup sẽ kiểm tra tất cả các descendant (con cháu) của mytag, con của con của nó, vân vân. Nếu bạn muốn BeautifulSoup duyệt trực tiếp phần tử con, bạn có thể truyền recursive=False, không đệ quy tiếp vào các phần tử con của phần tử con. Hãy xem sự khác biệt dưới đây:

```
soup.html.find_all("title")
```

```
# [<title>The Dormouse's story</title>]

soup.html.find_all("title", recursive=False)

[]
```

Đây là phần đó trong tài liệu:

```
<html>

<head>

<title>The Dormouse's story</title>

</head>

...
```

Thẻ `<title>` nằm bên dưới thẻ `<html>`, nhưng nó không trực tiếp nằm dưới `<html>` mà nó phải đi qua thẻ `<head>`. Beautiful Soup tìm thấy thẻ `<title>` khi nó được phép xem tất cả các descendant của thẻ `<html>` nhưng khi `recursive=False` nó sẽ giới hạn Beautiful Soup ở phần tử con trực tiếp của thẻ `<html>` vì thế nó sẽ không tìm thấy gì.

Beautiful Soup cung cấp nhiều phương thức tree-searching (sẽ được nói dưới đây), và hầu hết chúng đều có các đối số giống như `find_all()`: `name`, `attrs`, `string`, `limit`, và `keyword argument`. Nhưng `recursive` thì khác: Nó chỉ có ở hai phương thức `find_all()` và `find()`. Truyền `recursive=False` vào một phương thức như `find_parents()` sẽ không hữu ích.

6.5.12 Gọi một thẻ như gọi find_all()

Vì find_all() là phương thức phổ biến nhất trong Beautiful Soup search API, bạn có thể sử dụng nó một cách gọn hơn bằng cách lược bỏ nó. Nếu bạn coi BeautifulSoup object hoặc Tag object như thể nó là một hàm, thì nó cũng giống như gọi find_all() trên object đó. Hai dòng code này là tương tự nhau:

```
soup.find_all("a")
```

```
soup("a")
```

Hai dòng này cũng vậy:

```
soup.title.find_all(string=True)
```

```
soup.title(string=True)
```

4.3/ find()

```
find(name, attrs, recursive, string, **kwargs)
```

```
1
```

Phương thức find_all() quét toàn bộ tài liệu để tìm kiếm kết quả. Nhưng đôi khi bạn chỉ muốn tìm một kết quả. Nếu bạn biết một tài liệu chỉ có một thẻ <body>, thì thật lãng phí thời gian để quét toàn bộ tài liệu để tìm kiếm thêm. Thay vì thêm limit=1 mỗi khi gọi find_all, bạn có thể sử dụng phương thức find(). Hai dòng code này gần như giống nhau:

```
soup.find_all('title', limit=1)
```

```
# [<title>The Dormouse's story</title>]
```

```
soup.find('title')
```

```
# <title>The Dormouse's story</title>
```

Chỉ khác đó là find_all() trả về một list chứa kết quả, và find() thì trả về kết quả.

Nếu find_all() không thể tìm thấy gì, nó sẽ trả về một list rỗng. Còn với find(), không tìm thấy gì thì nó trả về None:

```
print(soup.find("nosuchtag"))
```

```
# None
```

```
# <title>The Dormouse's story</title>
```

6.6 CSS selectors

Trong phiên bản 4.7.0, BeautifulSoup hỗ trợ hầu hết các CSS4 selectors thông qua dự án SoupSieve. Nếu bạn đã cài đặt BeautifulSoup thông qua pip, thì SoupSieve đã được cài đặt cùng lúc đó, vì thế bạn không cần phải làm gì thêm.

BeautifulSoup có phương thức `.select()` sử dụng SoupSieve để chạy một CSS selector lên tài liệu đã được parse và trả về tất cả các phần tử match. Tag có một phương thức tương tự chạy một CSS selector lên `contents` của một thẻ.

Tài liệu SoupSieve liệt kê tất cả các CSS selectors hiện tại được hỗ trợ, nhưng ở đây ta chỉ đề cập tới những CSS selectors cơ bản:

Bạn có thể tìm kiếm các tag:

```
soup.select("title")
```

```
# [<title>The Dormouse's story</title>]
```

```
soup.select("p:nth-of-type(3)")
```

```
# [<p class="story">...</p>]
```

Tìm các thẻ bên dưới các thẻ khác:

```
soup.select("body a")
```

```
# [<a class="sister"
```

```
href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0L1x0DM4mpSt97%2ftYgbxlC2B7n4pvJNhhvRwo8b
xiO4B" id="link1">Elsie</a>,
```

```
# <a class="sister"
```

```
href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0OPun6GIXb9bh0UOloN9WCYbJtHZQd%2fvB08D2
UeudkPP" id="link2">Lacie</a>,
```

```
# <a class="sister"
```

```
href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0LirHL60gbBHH3VIishi9CqgtHAKmbGoKNvFheNku
mnh" id="link3">Tillie</a>]
```

```
soup.select("html head title")
# [<title>The Dormouse's story</title>]
```

Tìm những tag con trực tiếp của tag khác:

```
soup.select("head > title")
# [<title>The Dormouse's story</title>]

soup.select("p > a")
# [<a class="sister"
    href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0L1x0DM4mpSt97%2ftYgbx1C2B7n4pvJNhhvRwo8
    bxiO4B" id="link1">Elsie</a>,
    # <a class="sister"
    href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0OPun6GIXb9bh0UOloN9WCYbJtHZQd%2fvB08D
    2UeudkPP" id="link2">Lacie</a>,
    # <a class="sister"
    href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0LirHL60gbBHH3Vlishi9CqgtHAKmbGoKNvFheN
    kumnh" id="link3">Tillie</a>]

soup.select("p > a:nth-of-type(2)")
# [<a class="sister"
    href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0OPun6GIXb9bh0UOloN9WCYbJtHZQd%2fvB08D
    2UeudkPP" id="link2">Lacie</a>]

soup.select("p > #link1")
```

```
# [<a class="sister"
  href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0L1x0DM4mpSt97%2ftYgbxlC2B7n4pvJNhhvRwo8
  bxiO4B" id="link1">Elsie</a>]

soup.select("body > a")

# []
```

Tìm các siblings của tag:

```
soup.select("#link1 ~ .sister")

# [<a class="sister"
  href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0OPun6GIXb9bh0UOl0N9WCYbJtHZQd%2fvB08D2
  UeudkPP" id="link2">Lacie</a>,
  <a class="sister"
  href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0LirHL60gbBHH3Viishi9CqgtHAKmbGoKNvFheNku
  mnh" id="link3">Tillie</a>]

soup.select("#link1 + .sister")

# [<a class="sister"
  href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0OPun6GIXb9bh0UOl0N9WCYbJtHZQd%2fvB08D2
  UeudkPP" id="link2">Lacie</a>]
```

Tìm thẻ bằng CSS class:

```
soup.select(".sister")

# [<a class="sister"
  href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0L1x0DM4mpSt97%2ftYgbxlC2B7n4pvJNhhvRwo8
  bxiO4B" id="link1">Elsie</a>,
```

```
# <a class="sister"
href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0OPun6GIXb9bh0UOloN9WCYbJtHZQd%2fvB08D
2UeudkPP" id="link2">Lacie</a>,
# <a class="sister"
href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0LirHL60gbBHH3VIishi9CqgtHAKmbGoKNvFheN
kumnh" id="link3">Tillie</a>]

soup.select("[class~=sister]")

# [<a class="sister"
href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0L1x0DM4mpSt97%2ftYgbxlC2B7n4pvJNhvvRwo8
bxio4B" id="link1">Elsie</a>,
# <a class="sister"
href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0LirHL60gbBHH3VIishi9CqgtHAKmbGoKNvFheN
kumnh" id="link2">Lacie</a>,
# <a class="sister"
href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0LirHL60gbBHH3VIishi9CqgtHAKmbGoKNvFheN
kumnh" id="link3">Tillie</a>]
```

Tìm thẻ bằng CSS ID:

```
soup.select("#link1")

# [<a class="sister"
href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0L1x0DM4mpSt97%2ftYgbxlC2B7n4pvJNhvvRwo8b
xiO4B" id="link1">Elsie</a>]

soup.select("a#link2")

# [<a class="sister"
href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0OPun6GIXb9bh0UOloN9WCYbJtHZQd%2fvB08D2
UeudkPP" id="link2">Lacie</a>]
```

Tìm thẻ match với bất kỳ selector nào nằm trong danh sách các selectors:

```
soup.select("#link1,#link2")

# [<a class="sister"
href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0L1x0DM4mpSt97%2ftYgbxlC2B7n4pvJNhhvRwo8b
xiO4B" id="link1">Elsie</a>,

# <a class="sister"
href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0OPun6GIXb9bh0UOl0N9WCYbJtHZQd%2fvB08D2
UeudkPP" id="link2">Lacie</a>]
```

Kiểm tra sự tồn tại của một thuộc tính:

```
soup.select('a[href]')

# [<a class="sister"
href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0L1x0DM4mpSt97%2ftYgbxlC2B7n4pvJNhhvRwo8b
xiO4B" id="link1">Elsie</a>,

# <a class="sister"
href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0OPun6GIXb9bh0UOl0N9WCYbJtHZQd%2fvB08D2
UeudkPP" id="link2">Lacie</a>,

# <a class="sister"
href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0LirHL60gbBHH3Viishi9CqgtHAKmbGoKNvFheNku
mnh" id="link3">Tillie</a>]
```

Tìm thẻ bằng giá trị của thuộc tính:

```
soup.select('a[href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0L1x0DM4mpSt97%2ftYgbxlC3lW0c
WzhMK4ll9WwwLq4Ce"]

# [<a class="sister"
href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0L1x0DM4mpSt97%2ftYgbxlC2B7n4pvJNhhvRwo8
bxiO4B" id="link1">Elsie</a>]

soup.select('a[href^="http://example.com/]')
```

```
# [<a class="sister"
  href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0L1x0DM4mpSt97%2ftYgbxlC2B7n4pvJNhhvRwo8
  bxiO4B" id="link1">Elsie</a>,
  # <a class="sister"
  href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0OPun6GIXb9bh0UOloN9WCYbJtHZQd%2fvB08D
  2UeudkPP" id="link2">Lacie</a>,
  # <a class="sister"
  href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0LirHL60gbBHH3VIishi9CqgtHAKmbGoKNvFheN
  kumnh" id="link3">Tillie</a>]

soup.select('a[href$="tillie"]')

# [<a class="sister"
  href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0LirHL60gbBHH3VIishi9CqgtHAKmbGoKNvFheN
  kumnh" id="link3">Tillie</a>]

soup.select('a[href*=".com/el"]')

# [<a class="sister"
  href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0L1x0DM4mpSt97%2ftYgbxlC2B7n4pvJNhhvRwo8
  bxiO4B" id="link1">Elsie</a>]
```

Ngoài ra cũng có phương thức select_one() lấy ra thẻ đầu tiên match với selector:

```
soup.select_one(".sister")

# <a class="sister"
  href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0L1x0DM4mpSt97%2ftYgbxlC2B7n4pvJNhhvRwo8
  bxiO4B" id="link1">Elsie</a>
```

Nếu bạn parse XML định nghĩa namespaces, bạn có thể sử dụng chúng trong CSS selectors:

```

from bs4 import BeautifulSoup

xml = """<tag xmlns:ns1="http://namespace1/" xmlns:ns2="http://namespace2/">
<ns1:child>I'm in namespace 1</ns1:child>
<ns2:child>I'm in namespace 2</ns2:child>
</tag> """
soup = BeautifulSoup(xml, "xml")

soup.select("child")
# [<ns1:child>I'm in namespace 1</ns1:child>, <ns2:child>I'm in namespace 2</ns2:child>]

soup.select("ns1|child", namespaces=namespaces)
# [<ns1:child>I'm in namespace 1</ns1:child>]

```

Khi xử lý CSS selector sử dụng namespaces, Beautiful Soup sử dụng các từ viết tắt namespace nó đã tìm thấy khi parse tài liệu. Bạn có thể ghi đè chúng bằng cách truyền vào một dict chứa các từ viết tắt của riêng bạn:

```

namespaces = dict(first="http://namespace1/", second="http://namespace2/")
soup.select("second|child", namespaces=namespaces)
# [<ns1:child>I'm in namespace 2</ns1:child>]

```

Công cụ CSS selector là một công cụ hữu ích cho những ai đã biết về cú pháp của CSS selector. Bạn có thể làm tất cả điều này với Beautiful Soup API. Và nếu CSS selector là tất cả những gì bạn cần, bạn nên parse tài liệu bằng lxml: Nó nhanh hơn rất nhiều. Nó cho phép bạn kết hợp CSS selectors với Beautiful Soup API.

6.5Output

6.7.1 Xuất định dạng đúng chuẩn

Phương thức pretty() sẽ trả về một Beautiful Soup parse tree trong một chuỗi Unicode được format đúng chuẩn tài liệu HTML/XML, mỗi thẻ hay chuỗi HTML/XML nằm trên một dòng và được tab vào trong:

```
markup = '<a
href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0Pe0IPp5hh%2bXZ0lT1ERy9m6hwRC04QM%2f8a
5aL6Y5X8h" linked to <i>example.com</i></a>'

soup = BeautifulSoup(markup)

soup.prettify()

# '<html>\n <head>\n </head>\n <body>\n   <a
href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0Pe0IPp5hh%2bXZ0lT1ERy9m7Z1%2bs1zMhKDdU
p6WzjX6%2fv"

print(soup.prettify())

# <html>
# <head>
# </head>
# <body>
#   <a
href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0Pe0IPp5hh%2bXZ0lT1ERy9m4g%2f7SQKQWmq3
i%2bSO84ccyY"

#   I linked to
#   <i>
#     example.com
#   </i>
# </body>
# </html>'
```

```
# </i>
# </a>
# </body>
# </html>
```

Bạn có thể gọi pretty() trên top-level BeautifulSoup object, hoặc trên bất kì Tag object của nó:

```
print(soup.a.prettify())

# <a
href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0Pe0IPp5hh%2bXZ0lT1ERy9m4g%2f7SQKQWmq3i
%2bSO84ccyY"

# I linked to

# <i>
# example.com

# </i>

# </a>
```

6.7.2 Không cần đúng chuẩn

Nếu bạn chỉ muốn một chuỗi không cần định dạng đúng chuẩn, bạn có thể gọi unicode() hoặc str() trên một BeautifulSoup object, hoặc một Tag bên trong nó:

```
str(soup)

# '<html><head></head><body><a
href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0Pe0IPp5hh%2bXZ0lT1ERy9m6hwcRC04QM%2f8a
5aL6Y5X8h" linked to <i>example.com</i></a></body></html>'

unicode(soup.a)
```

```
# u'<a href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0Pe0IPp5hh%2bXZ0lT1ERy9m6hwRC04QM%2f8a5aL6Y5X8h" linked to <i>example.com</i></a>'
```

Hàm str() trả về một chuỗi được mã hóa UTF-8.

Bạn cũng có thể gọi encode() để nhận được một bytestring và decode() để nhận lại Unicode.

6.8 Định dạng đầu ra

Nếu bạn gọi BeautifulSoup() trên một tài liệu có chứa các HTML entity giống như này: ““&lquot;”, chúng sẽ được chuyển đổi sang ký tự Unicode:

```
soup = BeautifulSoup("&ldquo;Dammit!&rdquo; he said.")
```

```
unicode(soup)
```

```
# u'<html><head></head><body>\u201cDammit!\u201d he said.</body></html>'
```

Nếu sau đó bạn muốn chuyển đổi tài liệu đó thành một chuỗi, các ký tự Unicode sẽ được mã hóa thành UTF-8. Bạn sẽ không thể nhận lại những ký tự HTML đó nữa:

```
str(soup)
```

```
# '<html><head></head><body>\xe2\x80\x9cDammit!\xe2\x80\x9d he said.</body></html>'
```

Mặc định, những ký tự duy nhất được escape (nhiều ký tự escaped sequence sẽ hiểu cái này) đó là ký tự & và cặp ký tự < và >. Chúng sẽ được chuyển thành “&”, “<”, và “>”, thì thẻ BeautifulSoup không vô tình tạo ra HTML hay XML không hợp lệ:

```
soup = BeautifulSoup("<p>The law firm of Dewey, Cheatem, & Howe</p>")
```

```
soup.p
```

```
# <p>The law firm of Dewey, Cheatem, &amp; Howe</p>
```

```
soup = BeautifulSoup('<a href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0KlyAIB1M5LQGR6qO74mpXXNza7iN3Ch26t1XD1akMIUxSmQmvID08tz7wD6V6JIJQ%3d%3d" link</a>')
```

```
soup.a
```

```
# <a href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0KlyAIB1M5LQGR6qO74mpXWUBLnf4q12C6Qqoas8%2b7ONnjWBGCygZP5Uk4CNC0vYr0BJpRHnTHgaMtD9KPAgW%2fM%3d" link</a>
```

Bạn có thể thay đổi hành vi này bằng cách cung cấp giá trị cho đối số formatter trong các hàm prettify(), encode(), hoặc decode(). Beautiful Soup cung cấp cho đối số formatter 6 giá trị. Mặc định là formatter="minimal". Các chuỗi sẽ được xử lý đủ để đảm bảo rằng Beautiful Soup tạo ra HTML/XML hợp lệ:

```
french = "<p>Il a dit &lt;&lt;Sacr&acute; bleu!&gt;&gt;</p>"  
soup = BeautifulSoup(french)  
print(soup.prettify(formatter="minimal"))  
  
# <html>  
# <body>  
# <p>  
#   Il a dit &lt;&lt;Sacré bleu!&gt;&gt;  
# </p>  
# </body>  
# </html>
```

Nếu bạn truyền vào formatter="html" , Beautiful Soup sẽ chuyển chuỗi Unicode thành các ký tự HTML bất cứ khi nào có thể:

```
print(soup.prettify(formatter="html"))  
  
# <html>  
# <body>  
# <p>  
#   Il a dit &lt;&lt;Sacr&acute; bleu!&gt;&gt;
```

```
#   </p>
#
# </body>
#
# </html>
```

Nếu bạn truyền vào formatter="html5", nó giống như formatter="html", như BeautifulSoup sẽ bỏ qua dấu gạch chéo trong các void tag như
, :

```
soup = BeautifulSoup("<br>")

print(soup.encode(formatter="html"))

# <html><body><br/></body></html>

print(soup.encode(formatter="html5"))

# <html><body><br></body></html>
```

Nếu bạn truyền vào formatter=None, BeautifulSoup sẽ không sửa đổi chuỗi nào khi output. Đây là tùy chọn nhanh nhất, nhưng nó có thể khiến BeautifulSoup tạo HTML/XML không hợp lệ, như ví dụ sau:

```
print(soup.prettify(formatter=None))

# <html>
#
# <body>
#
#   <p>
#
#     Il a dit <<Sacré bleu!>>
#
#   </p>
#
# </body>
#
# </html>
```

```

link_soup = BeautifulSoup('<a
href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0KlyAIB1M5LQGR6qO74mpXXNza7iN3Ch26t1XD1
akMIUxSmQmvlD08tz7wD6V6JlJQ%3d%3d" link</a>')

print(link_soup.a.encode(formatter=None))

# <a
href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0KlyAIB1M5LQGR6qO74mpXXNza7iN3Ch26t1XD1
akMIUxSmQmvlD08tz7wD6V6JlJQ%3d%3d" link</a>

```

Nếu bạn muốn kiểm soát sâu hơn output của bạn, bạn có thể sử dụng class Formatter của Beautiful Soup. Ở đây Formatter chuyển thành in hoa cho dù chúng xuất hiện trong một nút văn bản hoặc trong một giá trị thuộc tính:

Cuối cùng, nếu bạn truyền một hàm vào formatter, Beautiful Soup sẽ gọi hàm đó một lần cho mỗi chuỗi và giá trị của thuộc tính trong tài liệu. Bạn có thể làm bất cứ thứ gì bạn muốn trong hàm này. Ở ví dụ bên dưới, hàm không làm gì khác ngoài chuyển chuỗi thành chữ in hoa:

```

from bs4.formatter import HTMLFormatter

def uppercase(str):
    return str.upper()

formatter = HTMLFormatter(uppercase)

print(soup.prettify(formatter=formatter))

# <html>
# <body>
# <p>
#   IL A DIT <<SACRÉ BLEU!>>
# </p>

```

```
# </body>
# </html>

print(link_soup.a.prettify(formatter=formatter))

# <a
href="/redirect?Id=FNIJAniE9NmhFYxa7PLViRHUQgQzIisaMngwQ1wz0kb1kOWIiFwcBOusnJvpf
GsB1qDYBTRTjSQYa4s43G5fbA%3d%3d"

# A LINK

# </a>
```

Subclassing HTMLFormatter hoặc XMLFormatter sẽ cung cấp cho bạn quyền kiểm soát sâu hơn đầu ra. Ví dụ, mặc định Beautiful Soup sắp xếp các thuộc tính trong mỗi thẻ:

```
attr_soup = BeautifulSoup(b'<p z="1" m="2" a="3"></p>')
print(attr_soup.p.encode())

# <p a="3" m="2" z="1"></p>
```

Để tắt tính năng này, bạn có thể định nghĩa lại phương thức Formatter.attribute(), kiểm soát thuộc tính nào là đầu ra và theo thứ tự nào. Việc này cũng lọc ra thuộc tính m bất cứ khi nào nó xuất hiện:

```
class UnsortedAttributes(HTMLFormatter):

    def attributes(self, tag):
        for k, v in tag.attrs.items():
            if k == 'm':
                continue
            yield k, v
```

```
print(attr_soup.p.encode(formatter=UnsortedAttributes()))
# <p z="1" a="3"></p>
```

Cuối cùng: Nếu bạn tạo một Cdata object, text bên trong Object đó luôn trình bày chính xác như nó xuất hiện, không có định dạng. Beautiful Soup sẽ gọi formatter method, chỉ trong trường hợp bạn viết một phương thức đếm tất cả các string có trong tài liệu hay một cái gì đó. Nhưng nó sẽ bỏ qua giá trị trả về:

```
from bs4.element import CData
soup = BeautifulSoup("<a></a>")
soup.a.string = CData("one < three")
print(soup.a.prettify(formatter="xml"))
# <a>
# <![CDATA[one < three]]>
# </a>
```

6.9 get_text()

Nếu bạn chỉ muốn lấy phần text có trong tài liệu hoặc trong một thẻ, bạn có thể sử dụng `get_text()` method. Nó trả về một chuỗi Unicode là tất cả text trong một tài liệu hoặc bên dưới một thẻ:

```
markup = '<a href="/redirect?Id=f%2fKgPq4IDV0SyEq0zfYr0Pe0IPp5hh%2bXZ0lT1ERy9m7IzJaFrOe03lux7H4%2fiNPy" linked to <i>example.com</i>\n</a>'

soup = BeautifulSoup(markup)

soup.get_text()
u'\nI linked to example.com\n'

soup.i.get_text()
u'example.com'
```

Bạn có thể chỉ định chuỗi để nối các bit văn bản lại với nhau:

```
soup.get_text("|")  
# u'\nI linked to |example.com|\n'
```

Bạn có thể nói cho Beautiful Soup cắt khoảng trắng ở nơi bắt đầu và kết thúc của mỗi bit văn bản:

```
soup.get_text("|", strip=True)  
# u'I linked to|example.com'
```

Nhưng lúc đó bạn có thể sử dụng .stripped_strings generator thay thế, và sau đó bạn có thể tự xử lý:

```
[text for text in soup.stripped_strings]  
# [u'I linked to', u'example.com']
```

BÀI TẬP:

BT1: Cho nội dung trang web đơn giản sau:

```
<html>
<head>
    <title>
        demo web cho bai swraping web
    </title>
</head>
<body>
    <table border=1 width="800px" height="100px">
        <tr>
            <th> Ma so sinh vien</th>
            <th> Ten sinh vien </th>
        </tr>
        <tr>
            <td> 12312312</td>
            <td> Nguyen van a</td>
        </tr>
        <tr>
            <td> 467643784</td>
            <td> Nguyen van b</td>
        </tr>
    </table>
</body>
```

</html>

Yêu cầu: lấy dữ liệu là mã sinh viên và tên sinh viên từ trang web trên.

BT2:

Cho link trang web sau :

https://en.wikipedia.org/wiki/List_of_districts_of_Vietnam#H%C1%BB%C93_Ch%C3%AD_Minh_City

Yêu cầu: trích xuất tên các quận huyện thuộc về TPHCM.

Chương 7: Ngoại lệ và I/O File

7.1. Xử lý ngoại lệ.....	146
7.2. Sự cố mà không cần xử lý ngoại lệ.....	146
7.3. Xử lý ngoại lệ trong python.....	147
7.4. Khai báo nhiều ngoại lệ.....	152
7.5. Xử lý ngoại lệ với try...finally.....	153
7.6. Xử lý ngoại lệ với raise().....	155
7.7. Ngoại lệ tùy chỉnh	157
7.8. Xử lý dữ liệu trên File	157
7.9. Phương thức open ().....	158
7.10. Phương thức close().....	160
7.11. Sử dụng câu lệnh with.....	1601
7.12. Ghi file	1611
7.13. Đọc file qua vòng lặp for	164
7.14. Đọc các dòng của file	164
7.15. Tạo một file mới.....	166
7.16. Vị trí con trỏ file	167
7.17. Sửa đổi vị trí con trỏ file.....	168
7.18. Module OS trong Python	169
7.19. Các phương thức làm việc với file.....	172
7.20. Bài tập xử lý ngoại lệ	174
7.21. Bài tập xử lý File	174

7.1. Xử lý ngoại lệ

Một ngoại lệ có thể được định nghĩa là một điều kiện bất thường trong một chương trình dẫn đến sự gián đoạn trong dòng chương trình.

Bất cứ khi nào một ngoại lệ xảy ra, chương trình sẽ dừng việc thực thi và do đó đoạn mã khác không được thực thi. Do đó, một ngoại lệ là các lỗi thời gian chạy không thể xử lý đối với tập lệnh Python. Một ngoại lệ là một đối tượng Python đại diện cho một lỗi

Python cung cấp một cách để xử lý ngoại lệ để mã có thể được thực thi mà không bị gián đoạn. Nếu chúng tôi không xử lý ngoại lệ, trình thông dịch sẽ không thực thi tất cả mã tồn tại sau ngoại lệ.

Python cung cấp số lượng ngoại lệ tích hợp sẵn, trong bài học này sẽ mô tả các ngoại lệ tiêu chuẩn phổ biến. Dưới đây là danh sách các ngoại lệ phổ biến có thể được đưa ra từ một chương trình Python chuẩn.

ZeroDivisionError: Xảy ra khi một số bị chia hết cho số không

NameError: Nó xảy ra khi không tìm thấy tên. Nó có thể là cục bộ hoặc toàn cục.

IndentationError: Nếu thụt lề sai được đưa ra.

IOError: Hiển thị khi lỗi Input và Output.

EOFError: Nó xảy ra khi đến cuối tệp, nhưng các hoạt động đang được thực hiện.

7.2. Sự cố mà không cần xử lý ngoại lệ

Như chúng ta đã thảo luận, ngoại lệ là một điều kiện bất thường làm tạm dừng quá trình thực thi chương trình.

Giả sử chúng ta có hai biến a và b, lấy dữ liệu đầu vào từ người dùng và thực hiện phép chia các giá trị này. Điều gì sẽ xảy ra nếu người dùng nhập số 0 làm mẫu số? Nó sẽ làm gián đoạn việc thực thi chương trình và thông qua một ngoại lệ ZeroDivision. Hãy xem ví dụ sau.

Code:

```

1. a = int(input("Enter a:"))
2. b = int(input("Enter b:"))
3. c = a/b
4. print("a/b = %d" %c)
5. #other code:
6. print("Hi I am other part of the program")

```

Kết quả:

```

Enter a:10
Enter b:0
Traceback (most recent call last):
File "exception-test.py", line 3, in <module>
    c = a/b;
ZeroDivisionError: division by zero

```

Chương trình trên là đúng về mặt cú pháp, nhưng nó thông qua lỗi do đầu vào bắt thường. Loại chương trình đó có thể không phù hợp hoặc không được khuyến nghị cho các dự án vì những dự án này được yêu cầu thực hiện liên tục. Đó là lý do tại sao xử lý ngoại lệ đóng một vai trò thiết yếu trong việc xử lý những ngoại lệ không mong muốn này. Chúng ta có thể xử lý các trường hợp ngoại lệ này theo cách sau.

7.3. Xử lý ngoại lệ trong python

7.3.1. Câu lệnh try-except

Nếu chương trình Python chứa mã đáng ngờ có thể đưa ra ngoại lệ, chúng ta phải đặt mã đó vào khối try. Khối try phải được sau với câu lệnh except, chứa một khối mã sẽ được thực thi nếu có một số ngoại lệ trong khối try.



Cú pháp:

1. **try**:
2. **#block of code**
3. **except** Exception1:
4. **#block of code**
5. **except** Exception2:
6. **#block of code**
7. **#other code**

Hãy xem xét ví dụ sau:

1. **try**:
2. a = int(input("Enter a:"))
3. b = int(input("Enter b:"))
4. c = a/b
5. **except**:
6. **print**("Can't divide with zero")

Kết quả:

```

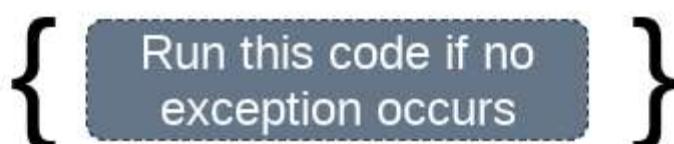
Enter a:10
Enter b:0
Can't divide with zero

```

Chúng ta cũng có thể sử dụng câu lệnh else với câu lệnh try-except trong đó, chúng ta có thể đặt mã sẽ được thực thi trong kịch bản nếu không có ngoại lệ nào xảy ra trong khôi try.

Cú pháp để sử dụng câu lệnh else với câu lệnh try-except được đưa ra bên dưới.

1. **try:**
2. #block of code
3. **except** Exception1:
4. #block of code
5. **else:**
6. #this code executes if no except block is executed

try**except****else**

Hãy xem xét chương trình sau đây:

1. **try:**
2. a = int(input("Enter a:"))
3. b = int(input("Enter b:"))
4. c = a/b
5. **print("a/b = %d"%c)**
6. **# Using Exception with except statement. If we print(Exception) it will return exception class**

```

7. except Exception:
8.   print("can't divide by zero")
9.   print(Exception)
10. else:
11.   print("Hi I am else block")

```

Kết quả:

Enter a:10

Enter b:0

can't divide by zero

<class 'Exception'>

7.3.2. Câu lệnh except không có ngoại lệ

Python cung cấp sự linh hoạt không chỉ định tên của ngoại lệ với câu lệnh ngoại lệ.

Ví dụ:

```

1. try:
2.   a = int(input("Enter a:"))
3.   b = int(input("Enter b:"))
4.   c = a/b;
5.   print("a/b = %d"%c)
6. except:
7.   print("can't divide by zero")
8. else:
9.   print("Hi I am else block")

```

7.3.3. Câu lệnh except với biến ngoại lệ

Chúng ta có thể sử dụng biến ngoại lệ với câu lệnh except. Nó được sử dụng bằng cách sử dụng từ khóa as. đối tượng này sẽ trả về nguyên nhân của ngoại lệ. Hãy xem xét ví dụ sau:

```

1. try:
2.     a = int(input("Enter a:"))
3.     b = int(input("Enter b:"))
4.     c = a/b
5.     print("a/b = %d"%c)
6. # Using exception object with the except statement
7. except Exception as e:
8.     print("can't divide by zero")
9.     print(e)
10. else:
11.     print("Hi I am else block")

```

Kết quả:

```
Enter a:10
```

```
Enter b:0
```

```
can't divide by zero
```

```
division by zero
```

7.3.4. Những điểm cần nhớ

Python tạo điều kiện cho chúng ta không chỉ định ngoại lệ bằng câu lệnh except.

Chúng ta có thể khai báo nhiều ngoại lệ trong câu lệnh except vì khối try có thể chứa các câu lệnh đưa ra các loại ngoại lệ khác nhau.

Chúng ta cũng có thể chỉ định một khối khác cùng với câu lệnh try-except, lệnh này sẽ được thực thi nếu không có ngoại lệ nào được nêu ra trong khối try.

Các câu lệnh không ngoại lệ nên được đặt bên trong khối else.

Ví dụ:

```

1. try:
2.     #this will throw an exception if the file doesn't exist.
3.     fileptr = open("file.txt","r")
4. except IOError:
5.     print("File not found")
6. else:
7.     print("The file opened successfully")
8.     fileptr.close()

```

Kết quả:

File not found

7.4. Khai báo nhiều ngoại lệ

Python cho phép chúng ta khai báo nhiều ngoại lệ với mệnh đề `except`. Khai báo nhiều ngoại lệ rất hữu ích trong trường hợp khối `try` chứa nhiều ngoại lệ. Cú pháp được đưa ra dưới đây.

```

1. try:
2.     #block of code
3. except (<Exception 1>,<Exception 2>,<Exception 3>,...<Exception n>)
4.     #block of code
5. else:
6.     #block of code

```

Hãy xem xét ví dụ sau.

```

1. try:
2.     a=10/0;
3. except(ArithmeticError, IOError):

```

```
4.     print("Arithmetic Exception")
5. else:
6.     print("Successfully Done")
```

Kết quả:

```
Arithmetic Exception
```

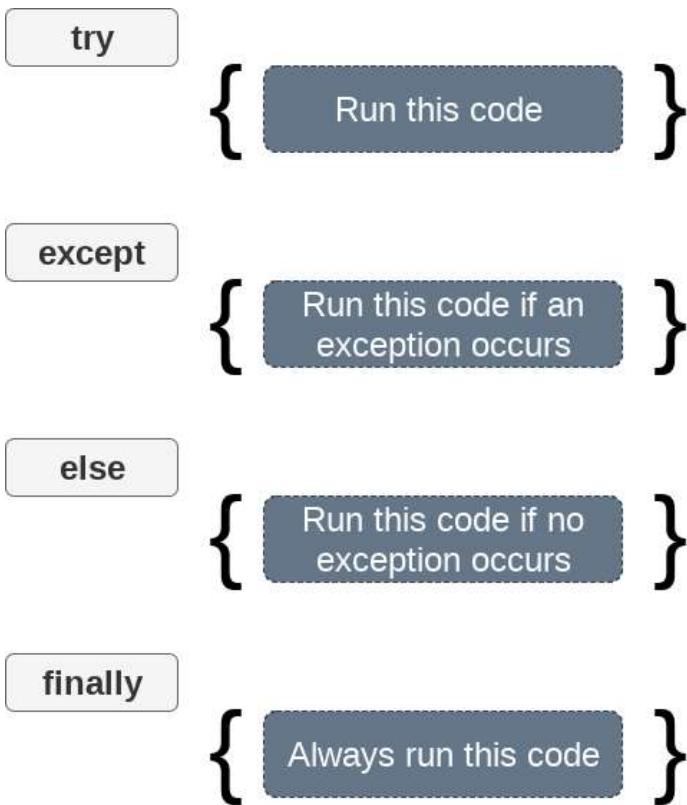
7.5. Xử lý ngoại lệ với try...finally

Python cung cấp câu lệnh finally, được sử dụng với câu lệnh try. Nó được thực thi bất kể trường hợp ngoại lệ nào xảy ra và được sử dụng để giải phóng tài nguyên bên ngoài. Finally cung cấp một sự đảm bảo về việc thực thi.

Chúng ta có thể sử dụng khối Finally với khối try, trong đó chúng ta có thể xử lý mã cần thiết, mã này phải được thực thi trước khi câu lệnh try có một ngoại lệ.

Cú pháp để sử dụng khối cuối cùng được đưa ra dưới đây.

```
1. try:
2.     # block of code
3.     # this may throw an exception
4. finally:
5.     # block of code
6.     # this will always be executed
```



Ví dụ:

```

1. try:
2.     fileptr = open("file2.txt","r")
3. try:
4.     fileptr.write("Hi I am good")
5. finally:
6.     fileptr.close()
7.     print("file closed")
8. except:
9.     print("Error")
  
```

Kết quả:

file closed

Error**7.6. Xử lý ngoại lệ với raise()**

Một ngoại lệ có thể được nâng lên một cách bắt buộc bằng cách sử dụng mệnh đề raise trong Python. Nó rất hữu ích trong trường hợp đó, chúng ta cần đưa ra một ngoại lệ để dừng việc thực thi chương trình.

Ví dụ, có một chương trình yêu cầu bộ nhớ 2GB để thực thi và nếu chương trình cố gắng chiếm 2GB bộ nhớ, thì chúng ta có thể đưa ra một ngoại lệ để dừng việc thực thi chương trình.

Cú pháp để sử dụng câu lệnh tăng được đưa ra dưới đây.

1. **raise** Exception_class,<value>

Lưu ý:

Để đưa ra một ngoại lệ, câu lệnh raise được sử dụng. Tên lớp ngoại lệ theo sau nó.

Một ngoại lệ có thể được cung cấp với một giá trị có thể được đưa ra trong ngoặc đơn.

Để truy cập giá trị, từ khóa "as" được sử dụng. "e" được sử dụng như một biến tham chiếu lưu trữ giá trị của ngoại lệ.

Chúng ta có thể chuyển giá trị cho một ngoại lệ để chỉ định loại ngoại lệ.

Ví dụ 1:

```

1. try:
2.     age = int(input("Enter the age:"))
3.     if(age<18):
4.         raise ValueError
5.     else:
6.         print("the age is valid")
7.     except ValueError:
8.         print("The age is not valid")

```

Kết quả:

Enter the age:17

The age is not valid

Ví dụ 2: Tăng ngoại lệ với tin nhắn

```

1. try:
2.     num = int(input("Enter a positive integer: "))
3.     if(num <= 0):
4.         # we can pass the message in the raise statement
5.         raise ValueError("That is a negative number!")
6.     except ValueError as e:
7.         print(e)

```

Kết quả:

Enter a positive integer: -5

That is a negative number!

Ví dụ 3:

```

1. try:
2.     a = int(input("Enter a:"))
3.     b = int(input("Enter b:"))
4.     if b is 0:
5.         raise ArithmeticError
6.     else:
7.         print("a/b = ",a/b)
8.     except ArithmeticError:
9.         print("The value of b can't be 0")

```

Kết quả:

Enter a:10

Enter b:0

The value of b can't be 0

7.7. Ngoại lệ tùy chỉnh

Python có thể tạo các ngoại lệ được nhận từ chương trình và được phát hiện bằng cách sử dụng mệnh đề ngoại trừ.

Ví dụ:

```

1. class ErrorInCode(Exception):
2.     def __init__(self, data):
3.         self.data = data
4.     def __str__(self):
5.         return repr(self.data)
6.
7. try:
8.     raise ErrorInCode(2000)
9. except ErrorInCode as ae:
10.    print("Received error:", ae.data)

```

Kết quả:

Received error: 2000

7.8. Xử lý dữ liệu trên File

Xử lý dữ liệu trên File đóng vai trò quan trọng cho các bài toán cần lưu trữ dữ liệu và tái sử dụng khi cần thiết. Python cung cấp sẵn các câu lệnh đơn giản để xử lý dữ liệu trên file dễ dàng.

Trong Python, các file được xử lý ở hai chế độ là văn bản hoặc nhị phân. File có thể ở định dạng văn bản hoặc nhị phân và mỗi dòng của tệp được kết thúc bằng ký tự đặc biệt.

Nội dung càn thực hiện sau bài học này:

- Mở file
- Đọc và ghi trên file
- Đóng file

7.9. Phương thức open ()

Python cung cấp hàm **open()** có hai đối số: tên file và mode truy cập. Hàm trả về một đối tượng file để có thể sử dụng để đọc, ghi,v.v.

Cú pháp:

```
file object = open(<file-name>, <access-mode>, <buffering>)
```

Các file có thể được truy cập bằng các chế độ khác nhau như đọc, ghi hoặc nối thêm. Sau đây là chi tiết về chế độ truy cập để mở file.

MODE	MÔ TẢ
‘r’	Chế độ chỉ được phép đọc.
‘r+’	Chế độ được phép đọc và ghi
‘rb’	Mở file chế độ đọc cho định dạng nhị phân. Con trỏ tại phần bắt đầu của file
‘rb+’ ‘r+b’	Mở file để đọc và ghi trong định dạng nhị phân. Con trỏ tại phần bắt đầu của file
‘w’	Mở file để ghi. Nếu file không tồn tại thì sẽ tạo mới file và ghi nội dung, nếu file đã tồn tại thì sẽ bị cắt bớt (truncate) và ghi đè lên nội dung cũ
‘w+’	Mở file để đọc và ghi. Nếu file không tồn tại thì sẽ tạo mới file và ghi nội dung, nếu file đã tồn tại thì sẽ bị cắt bớt (truncate) và ghi đè lên nội dung cũ

‘wb’	Mở file để ghi cho dạng nhị phân. Nếu file không tồn tại thì sẽ tạo mới file và ghi nội dung, nếu file đã tồn tại thì sẽ bị cắt bớt (truncate) và ghi đè lên nội dung cũ
‘wb+’, ‘w+b’	Mở file để đọc và ghi cho dạng nhị phân. Nếu file không tồn tại thì sẽ tạo mới file và ghi nội dung, nếu file đã tồn tại thì sẽ bị cắt bớt (truncate) và ghi đè lên nội dung cũ
‘a’	Mở file chế độ ghi tiếp. Nếu file đã tồn tại rồi thì nó sẽ ghi tiếp nội dung vào cuối file, nếu file không tồn tại thì tạo một file mới và ghi nội dung vào đó.
‘a+’	Mở file chế độ đọc và ghi tiếp. Nếu file đã tồn tại rồi thì nó sẽ ghi tiếp nội dung vào cuối file, nếu file không tồn tại thì tạo một file mới và ghi nội dung vào đó.
‘ab’,	Mở file chế độ ghi tiếp ở dạng nhị phân. Nếu file đã tồn tại rồi thì nó sẽ ghi tiếp nội dung vào cuối file, nếu file không tồn tại thì tạo một file mới và ghi nội dung vào đó.
‘ab+’, ‘a+b’	Mở file chế độ đọc và ghi tiếp ở dạng nhị phân. Nếu file đã tồn tại rồi thì nó sẽ ghi tiếp nội dung vào cuối file, nếu file không tồn tại thì tạo một file mới và ghi nội dung vào đó.
‘x’	Mở file chế độ ghi. Tạo file độc quyền mới (exclusive creation) và ghi nội dung, nếu file đã tồn tại thì chương trình sẽ báo lỗi
‘x+’	Mở file chế độ đọc và ghi. Tạo file độc quyền mới (exclusive creation) và ghi nội dung, nếu file đã tồn tại thì chương trình sẽ báo lỗi

‘xb’	Mở file chế độ ghi dạng nhị phân. Tạo file đọc quyền mới và ghi nội dung, nếu file đã tồn tại thì chương trình sẽ báo lỗi
‘xb+’ ‘x+b’	Mở file chế độ đọc và ghi dạng nhị phân. Tạo file đọc quyền mới và ghi nội dung, nếu file đã tồn tại thì chương trình sẽ báo lỗi
‘b’	Mở file ở chế độ nhị phân
‘t’	Mở file ở chế độ văn bản (mặc định)

Hãy xem ví dụ đơn giản để mở một tệp có tên "file.txt" (được lưu trữ trong cùng một thư mục) ở chế độ chỉ đọc (r) và in nội dung của tệp đó trên màn hình.

Ví dụ:

```
#opens the file file.txt in read mode
fileptr = open("file.txt", "r")
if fileptr:
    print("file is opened successfully")
```

Output:

```
<class '_io.TextIOWrapper'>
file is opened successfully
```

Trong ví dụ trên, "file.txt" là đối số đầu tiên và "r" là đối số thứ 2. Trong đó, **fileptr** là đối tượng của file và nếu mở thành công sẽ in ra màn hình với nội dung của lệnh print.

7.10. Phương thức close()

Phương thức **close()** được Python cung cấp nhằm đóng file lại khi đã thực hiện xong các thao tác cần thiết.

Cú pháp:

```
fileobject.close()
```

Ví dụ:

1. `# opens the file file.txt in read mode`
2. `fileptr = open("file.txt","r")`
3. `if fileptr:`
4. `print("file is opened successfully")`
5. `#closes the opened file`
6. `fileptr.close()`

Sau khi đóng file, chương trình không thể thực hiện bất kỳ thao tác nào trong file. File cần được đóng đúng cách. Nếu bất kỳ ngoại lệ nào xảy ra trong khi thực hiện một số thao tác trong file thì chương trình sẽ kết thúc mà không đóng file.

Chúng ta nên sử dụng phương pháp sau để khắc phục loại vấn đề như vậy.

1. `try:`
2. `fileptr = open("file.txt")`
3. `# perform file operations`
4. `finally:`
5. `fileptr.close()`

7.11. Sử dụng câu lệnh with

Câu lệnh with đã được giới thiệu trong python 2.5. Câu lệnh with hữu ích trong trường hợp thao tác các file. Nó được sử dụng trong trường hợp một cặp câu lệnh được thực thi với một khối mã ở giữa.

Cú pháp để mở file bằng câu lệnh with được đưa ra bên dưới.

1. `with open(<file name>, <access mode>) as <file-pointer>:`
2. `#statement suite`

Lợi thế của việc sử dụng với câu lệnh with là nó cung cấp sự đảm bảo để đóng file bất kể cách khống lồng nhau thoát ra.

Luôn luôn khuyến nghị sử dụng câu lệnh with trong trường hợp tệp vì: nếu ngắn, trả về hoặc ngoại lệ xảy ra trong khối mã lồng nhau thì nó sẽ tự động đóng file, chúng ta không cần phải viết hàm close () ... Nó không để cho file bị hỏng.

Ví dụ:

1. with open("file.txt",'r') as f:
2. content = f.read();
3. **print**(content)

7.12. Ghi file

Để ghi một số văn bản vào file, chúng ta cần mở file bằng phương pháp mở với một trong các chế độ truy cập sau.

w: Nó sẽ ghi đè lên file nếu có bất kỳ file nào tồn tại. Con trỏ nằm ở đầu file.

a: Nó sẽ nối file hiện có. Con trỏ nằm ở cuối file. Nó tạo một file mới nếu không có file nào tồn tại.

Ví dụ 1:

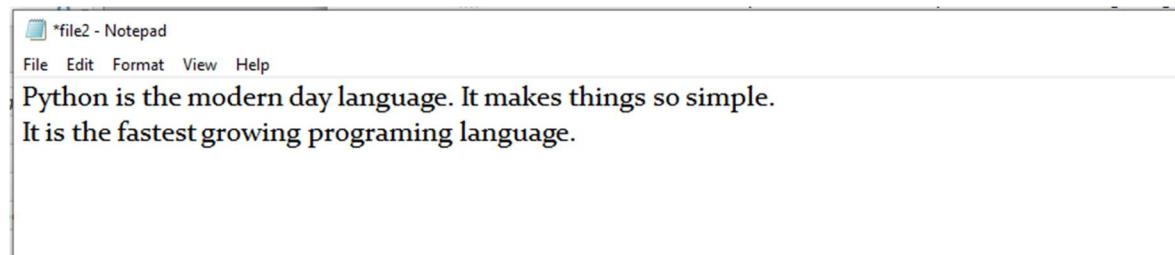
1. # open the file.txt in append mode. Create a new file if no such file exists.
2. fileptr = open("file2.txt", "w")
3. # appending the content to the file
4. fileptr.write("""Python is the modern day language. It makes things so simple.
It is the fastest-growing programming language""")
5. # closing the opened the file
6. fileptr.close()

Output:

File2.txt

Python is the modern-day language. It makes things so simple. It is the fastest growing programming language.

Snapshot of the file2.txt



Chúng ta đã mở file ở chế độ w. File file1.txt không tồn tại, nó đã tạo một file mới và chúng ta đã ghi nội dung vào file bằng hàm write () .

Ví dụ 2

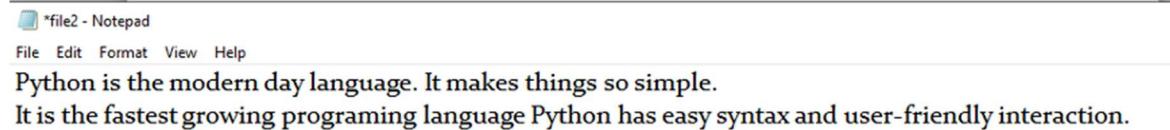
1. #open the file.txt in write mode.
2. fileptr = open("file2.txt","a")
- 3.
4. #overwriting the content of the file
5. fileptr.write(" Python has an easy syntax and user-friendly interaction.")
- 6.
7. #closing the opened file
8. fileptr.close()

Output:

Python is the modern day language. It makes things so simple.

It is the fastest growing programming language Python has an easy syntax and user-friendly interaction.

Snapshot of the file2.txt



```
*file2 - Notepad
File Edit Format View Help
Python is the modern day language. It makes things so simple.
It is the fastest growing programming language Python has an easy syntax and user-friendly interaction.
```

Chúng ta có thể thấy rằng nội dung của file đã được sửa đổi. Chúng ta đã mở tệp ở một chế độ và nó đã thêm nội dung vào tệp file2.txt hiện có.

Để đọc tệp bằng lệnh Python, Python cung cấp phương thức read (). Phương thức read () đọc một chuỗi từ tệp. Nó có thể đọc dữ liệu trong văn bản cũng như định dạng nhị phân.

Cú pháp:

```
fileobj.read(<count>)
```

Ở đây, số đếm là số byte được đọc từ tệp bắt đầu từ đầu tệp. Nếu số lượng không được chỉ định, thì nó có thể đọc nội dung của tệp cho đến cuối.

Ví dụ:

1. #open the file.txt in read mode. causes error if no such file exists.
2. fileptr = open("file2.txt","r")
3. #stores all the data of the file into the variable content
4. content = fileptr.read(10)
5. # prints the type of the data stored in the file
6. print(type(content))
7. #prints the content of the file
8. print(content)
9. #closes the opened file
10. fileptr.close()

Output:

```
<class 'str'>
```

Python is

Trong đoạn mã trên, chúng ta đã đọc nội dung của file2.txt bằng cách sử dụng hàm read().

Chúng ta đã chuyển giá trị đếm là mười có nghĩa là nó sẽ đọc mười ký tự đầu tiên từ tệp.

Nếu chúng ta sử dụng dòng sau, thì nó sẽ in tất cả nội dung của tệp.

1. content = fileptr.read()
2. print(content)

Output:

Python is the modern-day language. It makes things so simple.

It is the fastest-growing programming language Python has easy syntax and user-friendly interaction.

7.13. Đọc file qua vòng lặp for

Ví dụ:

1. #open the file.txt in read mode. causes an error if no such file exists.

```

2. fileptr = open("file2.txt","r");
3. #running a for loop
4. for i in fileptr:
5.     print(i) # i contains each line of the file

```

Output:

Python is the modern day language.

It makes things so simple.

Python has easy syntax and user-friendly interaction.

5.14. Đọc các dòng của file

Python tạo điều kiện để đọc từng dòng tệp bằng cách sử dụng phương thức hàm readline () .

Phương thức readline () đọc các dòng của tệp ngay từ đầu, tức là nếu chúng ta sử dụng phương thức readline () hai lần, thì chúng ta có thể nhận được hai dòng đầu tiên của tệp.

Hãy xem xét ví dụ sau chứa một hàm readline () đọc dòng đầu tiên của tệp "file2.txt" chứa ba dòng.

Ví dụ 1:

```

1. #open the file.txt in read mode. causes error if no such file exists.
2. fileptr = open("file2.txt","r");
3. #stores all the data of the file into the variable content
4. content = fileptr.readline()
5. content1 = fileptr.readline()
6. #prints the content of the file
7. print(content)
8. print(content1)
9. #closes the opened file
10. fileptr.close()

```

Output:

Python is the modern day language.

It makes things so simple.

We called the **readline()** function two times that's why it read two lines from the file.

Python provides also the **readlines()** method which is used for the reading lines. It returns the list of the lines till the end of **file(EOF)** is reached.

Chúng ta đã gọi hàm `readline()` hai lần, đó là lý do tại sao nó đọc hai dòng từ tệp.

Python cũng cung cấp phương thức `readlines()` được sử dụng cho các dòng đọc. Nó trả về danh sách các dòng cho đến khi đạt đến cuối của file(**EOF**).

Ví dụ 2:

1. `#open the file.txt in read mode. causes error if no such file exists.`
2. `fileptr = open("file2.txt","r");`
- 3.
4. `#stores all the data of the file into the variable content`
5. `content = fileptr.readlines()`
- 6.
7. `#prints the content of the file`
8. `print(content)`
- 9.
10. `#closes the opened file`
11. `fileptr.close()`

Output:

`[Python is the modern day language.\n', 'It makes things so simple.\n', 'Python has easy syntax and user-friendly interaction.]`

[7.15. Tạo một file mới](#)

Tệp mới có thể được tạo bằng cách sử dụng một trong các chế độ truy cập sau với phương thức `open()`.

x: nó tạo một tệp mới với tên được chỉ định. Nó gây ra lỗi khi tồn tại một tệp có cùng tên.

a: Nó tạo một tệp mới với tên được chỉ định nếu không có tệp nào như vậy tồn tại. Nó gắn nội dung vào tệp nếu tệp đã tồn tại với tên được chỉ định.

w: Nó tạo một tệp mới với tên được chỉ định nếu không có tệp nào như vậy tồn tại. Nó ghi đè lên tệp hiện có.

Ví dụ 1

1. `#open the file.txt in read mode. causes error if no such file exists.`
2. `fileptr = open("file2.txt","x")`
3. `print(fileptr)`
4. `if fileptr:`
5. `print("File created successfully")`

Output:

```
<_ io.TextIOWrapper name='file2.txt' mode='x' encoding='cp1252'>
File created successfully
```

7.16. Vị trí con trỏ file

Python cung cấp phương thức tell () được sử dụng để in số byte mà con trỏ tệp hiện đang tồn tại. Hãy xem xét ví dụ sau.

1. `# open the file file2.txt in read mode`
2. `fileptr = open("file2.txt","r")`
3. `#initially the filepointer is at 0`
4. `print("The filepointer is at byte :",fileptr.tell())`
5. `#reading the content of the file`
6. `content = fileptr.read();`
7. `#after the read operation file pointer modifies. tell() returns the location of the fileptr.`
8. `print("After reading, the filepointer is at:",fileptr.tell())`

Output:

```
The filepointer is at byte : 0
After reading, the filepointer is at: 117
```

7.17. Sửa đổi vị trí con trỏ file

Trong các ứng dụng thực tế, đôi khi chúng ta cần thay đổi vị trí con trỏ tệp ở bên ngoài vì chúng ta có thể cần đọc hoặc ghi nội dung ở nhiều vị trí khác nhau.

Vì mục đích này, Python cung cấp cho chúng ta phương thức seek () cho phép chúng ta sửa đổi vị trí con trỏ tệp từ bên ngoài.

Cú pháp:

```
<file-ptr>.seek(offset[, from])
```

Phương thức seek () chấp nhận hai tham số:

offset: Nó đề cập đến vị trí mới của con trỏ tệp trong tệp.

from: Nó chỉ ra vị trí tham chiếu từ nơi các byte sẽ được di chuyển. Nếu nó được đặt thành 0, phần đầu của tệp được sử dụng làm vị trí tham chiếu. Nếu nó được đặt thành 1, vị trí hiện tại của con trỏ tệp được sử dụng làm vị trí tham chiếu. Nếu nó được đặt thành 2, phần cuối của con trỏ tệp được sử dụng làm vị trí tham chiếu.

Ví dụ:

1. # open the file file2.txt in read mode
2. fileptr = open("file2.txt","r")
3. #initially the filepointer is at 0
4. print("The filepointer is at byte :",fileptr.tell())
5. #changing the file pointer location to 10.
6. fileptr.seek(10);
7. #tell() returns the location of the fileptr.
8. print("After reading, the filepointer is at:",fileptr.tell())

Output:

The filepointer is at byte : 0

After reading, the filepointer is at: 10

7.18. Module OS trong Python

7.18.1. Đổi tên file

Module OS cho phép tương tác với hệ điều hành. Nó cung cấp các chức năng liên quan đến hoạt động xử lý tệp như đổi tên, xóa, v.v. Phương thức rename () để đổi tên tệp được chỉ định thành một tên mới. Cú pháp để sử dụng phương thức rename () được đưa ra dưới đây.

Cú pháp:

```
rename(current-name, new-name)
```

Đối số đầu tiên là tên tệp hiện tại và đối số thứ hai là tên đã sửa đổi. Chúng ta có thể thay đổi tên tệp bằng cách bỏ qua hai đối số này.

Ví dụ 1:

1. **import** os
2. **#rename file2.txt to file3.txt**
3. os.rename("file2.txt","file3.txt")

Output:

Đoạn mã trên đổi tên **file2.txt** thành **file3.txt**

7.18.2. Xóa tệp trong os

Module os cung cấp phương thức remove () được sử dụng để xóa tệp được chỉ định. Cú pháp để sử dụng phương thức remove () được đưa ra dưới đây.

```
remove(file-name)
```

Ví dụ 1

1. **import** os;
2. **#deleting the file named file3.txt**
3. os.remove("file3.txt")

5.18.3. Tạo thư mục mới

Phương thức mkdir () được sử dụng để tạo các thư mục trong thư mục làm việc hiện tại. Cú pháp để tạo thư mục mới được đưa ra dưới đây.

Cú pháp:

```
mkdir(directory name)
```

Ví dụ 1:

1. **import** os
2. #creating a new directory with the name new
3. os.mkdir("new")

7.18.4. Phương thức getcwd()

Phương thức này trả về thư mục làm việc hiện tại.

Cú pháp để sử dụng phương thức getcwd () được đưa ra bên dưới.

Cú pháp:

```
os.getcwd()
```

Ví dụ:

1. **import** os
2. os.getcwd()

Output:

```
'C:\\\\Users\\\\DEVANSH SHARMA'
```

5.18.5. Thay đổi thư mục làm việc hiện tại

Phương thức chdir () được sử dụng để thay đổi thư mục làm việc hiện tại thành một thư mục được chỉ định.

Cú pháp để sử dụng phương thức chdir () được đưa ra dưới đây.

Cú pháp:

```
chdir("new-directory")
```

Ví dụ:

1. **import** os
2. # Changing current directory with the new directory
3. os.chdir("C:\\\\Users\\\\DEVANSH SHARMA\\\\Documents")
4. #It will display the current working directory

5. os.getcwd()

Output:

```
'C:\\\\Users\\\\DEVANSH SHARMA\\\\Documents'
```

7.18.6. Xóa thư mục

Phương thức rmdir () được sử dụng để xóa thư mục được chỉ định.

Cú pháp để sử dụng phương thức rmdir () được đưa ra dưới đây.

Cú pháp:

```
os.rmdir(directory name)
```

Ví dụ 1:

1. **import** os
2. **#removing the new directory**
3. os.rmdir("directory_name")

Nó sẽ xóa thư mục được chỉ định.

7.18.7. Ghi đầu ra vào file

Phương thức check_call () của module subprocess được sử dụng để thực thi một tập lệnh Python và ghi đầu ra của tập lệnh đó vào một tệp.

Ví dụ sau chứa hai tập lệnh python. Tập lệnh file1.py thực thi script file.py và ghi đầu ra của nó vào tập tin văn bản output.txt.

Ví dụ:

file.py

1. temperatures=[10,-20,-289,100]
2. **def** c_to_f(c):
3. **if** c< -273.15:
4. **return** "That temperature doesn't make sense!"
5. **else**:
6. f=c*9/5+32
7. **return** f
8. **for** t **in** temperatures:
9. **print**(c_to_f(t))

file1.py

1. **import** subprocess
- 2.
3. with open("output.txt", "wb") as f:
4. subprocess.check_call(["python", "file.py"], stdout=f)

7.19. Các phương thức làm việc với file

Đối tượng file cung cấp các phương thức sau để thao tác trên các hệ điều hành khác nhau.

PHƯƠNG THÚC	MÔ TẢ
close()	Đóng một file đang mở. Nó không thực thi được nếu tập tin đã bị đóng.
fileno()	Trả về một số nguyên mô tả file (file descriptor).
flush()	Xóa sạch bộ nhớ đệm của luồng file.
isatty()	Trả về TRUE nếu file được kết nối với một thiết bị đầu cuối.
read(n)	Đọc n ký tự trong file.
readable()	Trả về TRUE nếu file có thể đọc được.
readline(n=-1)	Đọc và trả về một dòng từ file. Đọc nhiều nhất n byte/ký tự nếu được chỉ định.
readlines(n=-1)	Đọc và trả về một danh sách các dòng từ file. Đọc nhiều nhất n byte/ký tự nếu được chỉ định.
seek(offset,from=SEEK_SET)	Thay đổi vị trí hiện tại bên trong file.

seekable()	Trả về TRUE nếu luồng hỗ trợ truy cập ngẫu nhiên.
tell()	Trả về vị trí hiện tại bên trong file.
truncate(size=None)	Cắt gọn kích cỡ file thành kích cỡ tham số size.
writable()	Trả về TRUE nếu file có thể ghi được.
write(s)	Ghi s ký tự vào trong file và trả về.
writelines(lines)	Ghi một danh sách các dòng và file.

Bài tập**7.20. Bài tập xử lý ngoại lệ**

- Đưa ra một RuntimeError exception

Gợi ý: sử dụng raise()

- Viết hàm để tính $5/0$ và sử dụng try/exception để bắt lỗi.

Gợi ý: Sử dụng try/exception để bắt lỗi

- Định nghĩa một class exception tùy chỉnh, nhận một thông báo là thuộc tính.

Gợi ý: Để định nghĩa một class exception tùy chỉnh, chúng ta phải định nghĩa một class kế thừa từ Exception.

5.21. Bài tập xử lý File**CƠ BẢN:**

- Viết chương trình Python để đọc toàn bộ tệp văn bản.
- Viết chương trình Python để đọc n dòng đầu tiên của tệp.
- Viết chương trình Python để nối văn bản vào tệp và hiển thị văn bản.
- Viết chương trình Python để đọc n dòng cuối cùng của tệp.
- Viết chương trình Python để đọc từng dòng một tệp và lưu trữ thành danh sách.
- Viết chương trình Python để đọc từng dòng một, lưu trữ nó vào một biến.
- Viết chương trình Python để đọc từng dòng một tệp, lưu trữ nó vào một mảng.
- Viết chương trình python để tìm những từ dài nhất.
- Viết chương trình Python để đếm số dòng trong tệp văn bản.
- Viết chương trình Python để đếm tần suất xuất hiện của các từ trong tệp.
- Viết chương trình Python để ghi danh sách vào tệp.

12. Viết chương trình Python để sao chép nội dung của tệp sang tệp khác
13. Viết chương trình Python để kết hợp từng dòng từ tệp đầu tiên với dòng tương ứng trong tệp thứ hai.
14. Viết chương trình Python để đọc một dòng ngẫu nhiên từ một tệp
15. Viết một chương trình Python để đánh giá xem một tệp có bị đóng hay không
16. Viết chương trình Python để tạo 26 tệp văn bản có tên A.txt, B.txt, v.v. cho đến Z.txt

NÂNG CAO:

1. Bạn được cung cấp một tệp có tên class_scores.txt, trong đó mỗi dòng của tệp chứa tên người dùng và điểm kiểm tra được phân tách bằng dấu cách, như dưới đây:

Hoang	3
Phuong	2

Viết mã quét qua tệp, thêm 5 điểm vào mỗi điểm kiểm tra, và xuất tên người dùng và điểm kiểm tra mới vào tệp mới, score2.txt.

2. Bạn được cung cấp một tệp có tên là Grade.txt, trong đó mỗi dòng của tệp chứa tên người dùng và ba điểm kiểm tra được phân tách bằng dấu cách, như dưới đây:

Hoang	5	2	5
Phuong	8	8	7

Viết mã quét qua tệp và xác định có bao nhiêu học sinh đậu cả ba bài kiểm tra.

3. Bạn được cung cấp một tệp có tên là student.txt. Một dòng điển hình trong tệp trông giống như:

Hoang hoang@iuh.edu.vn 0968836883

Có một tên, một địa chỉ email và một số điện thoại, mỗi cái được phân tách bằng các tab. Viết một chương trình đọc từng dòng của tệp và đổi với mỗi dòng, viết hoa chữ cái đầu tiên họ và tên và thêm mã vùng 84 vào số điện thoại. Chương trình của bạn nên ghi điều này vào một tệp mới có tên là student2.txt. Đây là những gì dòng đầu tiên của tệp sẽ giống như sau:

Hoang hoang@iuh.edu.vn 840968836883

Chương 8: Matplotlib

Nội Dung

8.1. Mục tiêu bài học	178
8.2. Công cụ lập trình:	178
8.3. Giới thiệu Matplotlib.....	179
8.4. Các loại biểu đồ trong Matplotlib.....	183
8.5. Biểu đồ đường và biểu đồ rời rạc	187
8.6. Multiple Subplots.....	194
8.7. Chỉnh sửa các thành phần của đồ thị.....	203
8.8. Tùy chỉnh spines và ticks của đồ thị.....	213
8.9. Chú thích (Annotation).....	222
8.10. Legends	225
8.11. Màu sắc	231
8.12. Colormaps	237
8.13. Stylesheets	244
8.14. Contour	250
8.15. Biểu diễn lỗi qua errorbars	257
8.16. Histogram.....	258
8.17. Bài tập	264

8.1. Mục tiêu bài học

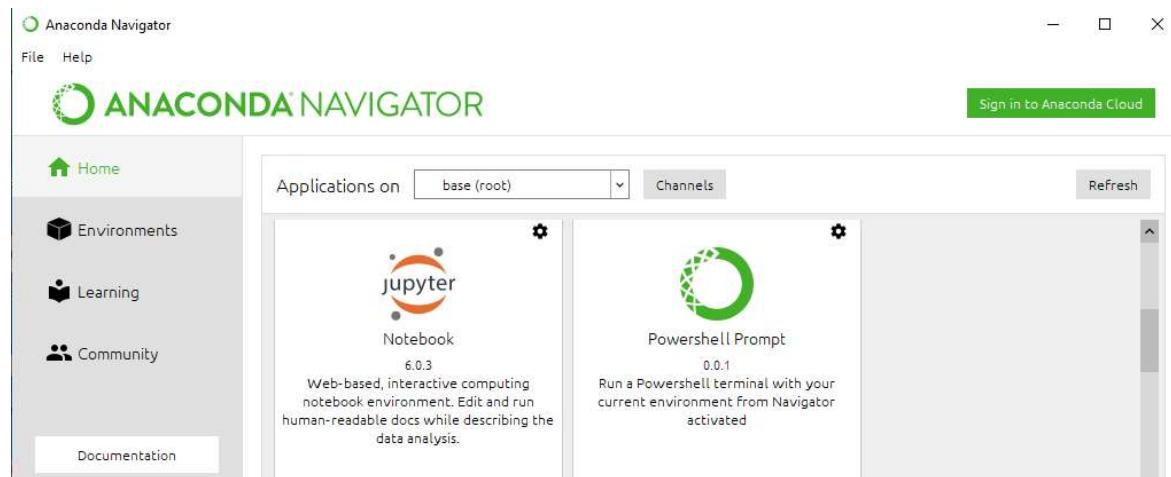
- Trang bị cho sinh viên kiến thức cơ bản về Matplotlib.
- Ứng dụng vào các bài toán biểu diễn dữ liệu trong thực tế.
- Tự nâng cấp nhật kiến thức Matplotlib thông qua tài liệu:
<https://matplotlib.org/tutorials/index.html>

8.2. Công cụ lập trình:

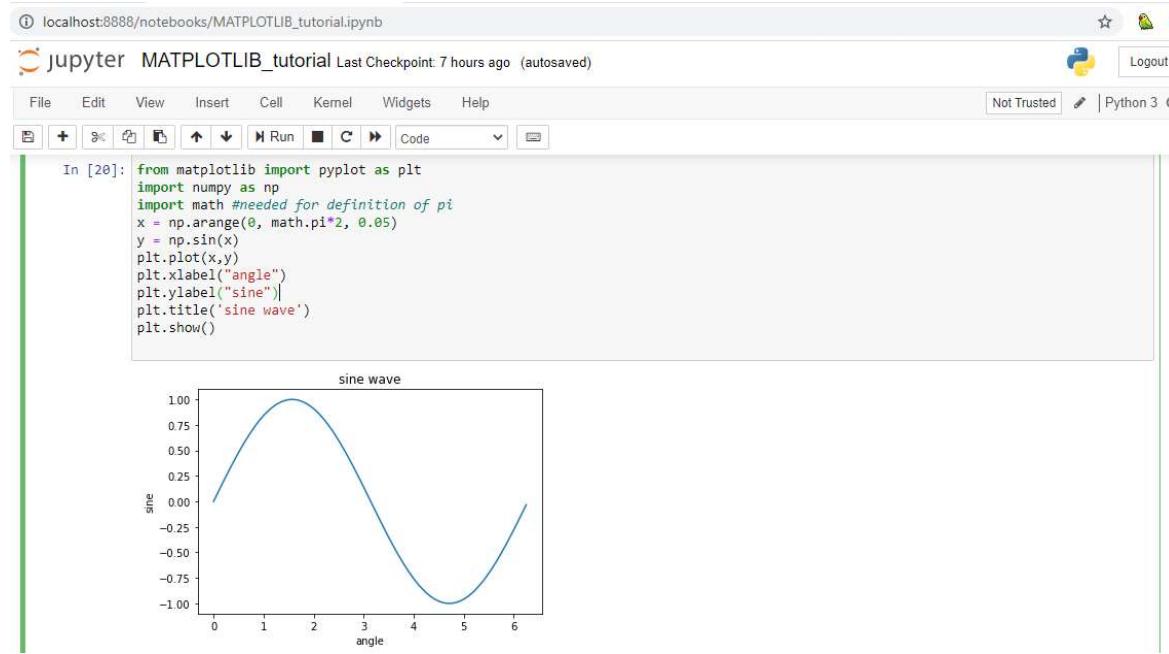
Jupyter Notebook. có thể tự cài đặt thông qua câu lệnh:

```
pip install jupyter notebook
```

Hoặc dùng anaconda từ link: <https://www.anaconda.com/products/individual>



Giao diện Jupyter Notebook:



8.3. Giới thiệu Matplotlib

Matplotlib ban đầu được viết bởi John D. Hunter vào năm 2003. Phiên bản ổn định hiện tại là 2.2.0 được phát hành vào tháng 1 năm 2018.

Matplotlib là một module của Python và được xem là một sự lựa chọn hoàn hảo thay thế cho MATLAB, nếu sử dụng kết hợp với Numpy và Scipy. Trong khi MATLAB đắt đỏ và mã nguồn đóng, Matplotlib lại miễn phí và mã nguồn mở. Nó cũng là ngôn ngữ hướng đối tượng. Hơn nữa, nó có thể được sử dụng với bộ công cụ GUI mục đích chung như wxPython, Qt, và GTK+. Cũng có một thủ tục "pylab", được thiết kế để giống với MATLAB. Điều này có thể làm cho những người quen sử dụng MATLAB dễ dàng chuyển sang dùng Matplotlib.

Sử dụng Matplotlib giúp công việc trở lên dễ dàng:"matplotlib cố gắng làm những điều khó khăn, phức tạp trở lên dễ dàng nhất có thể. Bạn có thể tạo ra các hình vẽ, histograms, phô, biểu đồ thanh, errorcharts, scatterplots, vv, với chỉ một vài dòng mã."

Matplotlib cung cấp một số interfaces để tương tác với thư viện matplotlib: Object-Oriented API, The Scripting Interface (pyplot), The MATLAB Interface (pylab). Pyplot và pylab đều là lightweight interfaces, tuy nhiên Pyplot cung cấp một giao diện thủ tục các thư viện vẽ hướng đối tượng trong matplotlib. Các lệnh vẽ của nó được thiết kế tương tự với Matlab cả về

cách đặt tên và ý nghĩa các đối số. Cách thiết kế này đã giúp cho việc sử dụng pyplot dễ dàng và dễ hiểu hơn.

Bài học này sẽ sử dụng giao diện pyplot thay vì hai giao diện còn lại. Nếu chúng ta muốn can thiệp sâu hơn, với nhiều tùy chỉnh hơn thì Object-Oriented API sẽ là lựa chọn thích hợp.

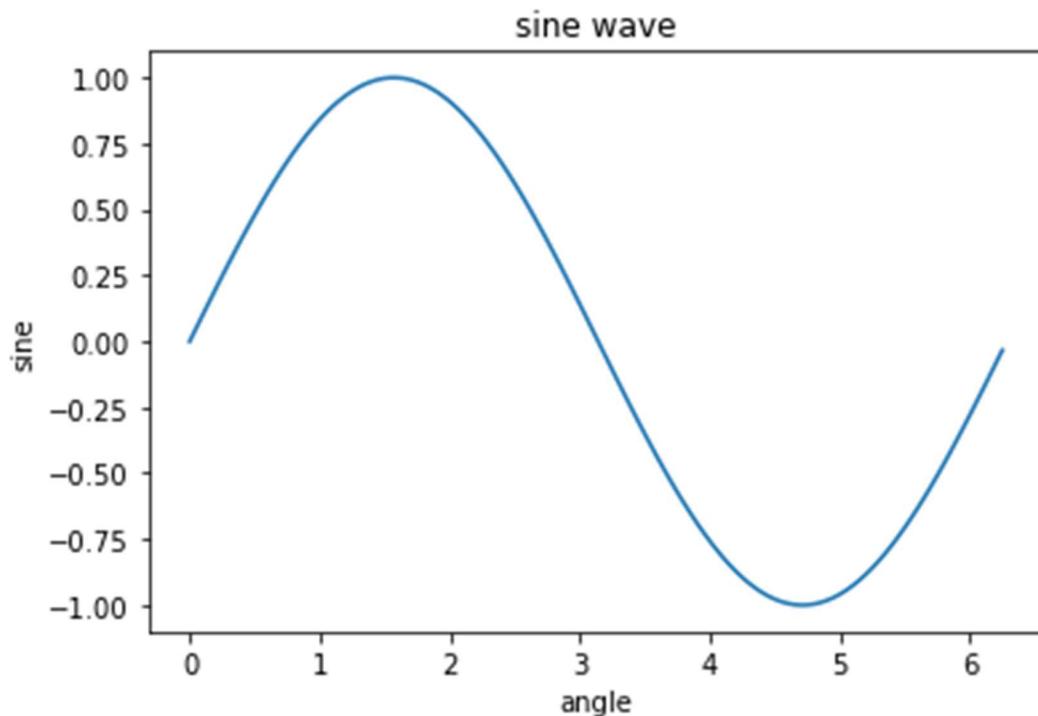
8.3.1. Import giao diện pyplot

Cú pháp;

```
import matplotlib.pyplot as plt
```

Ví dụ:

```
from matplotlib import pyplot as plt
import numpy as np
import math #needed for definition of pi
x = np.arange(0, math.pi*2, 0.05)
y = np.sin(x)
plt.plot(x,y)
plt.xlabel("angle")
plt.ylabel("sine")
plt.title('sine wave')
plt.show()
```



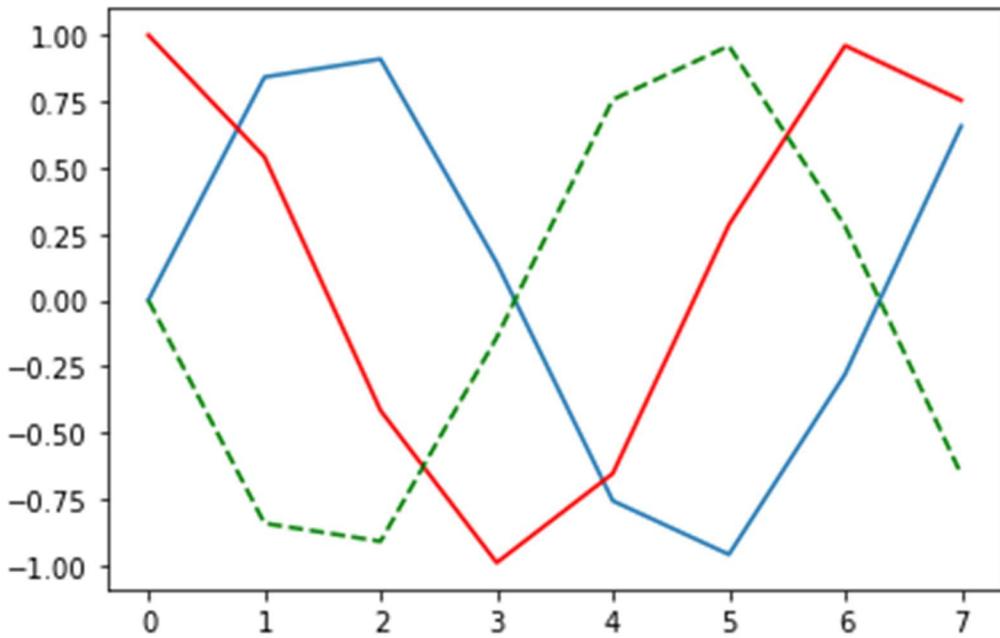
8.3.2. Import giao diện pylab

Cú pháp:

```
from pylab import *
```

Ví dụ:

```
from pylab import *
plot(x, sin(x))
plot(x, cos(x), 'r-')
plot(x, -sin(x), 'g--')
show()
```



8.3.3. Import giao diện Object-Oriented API

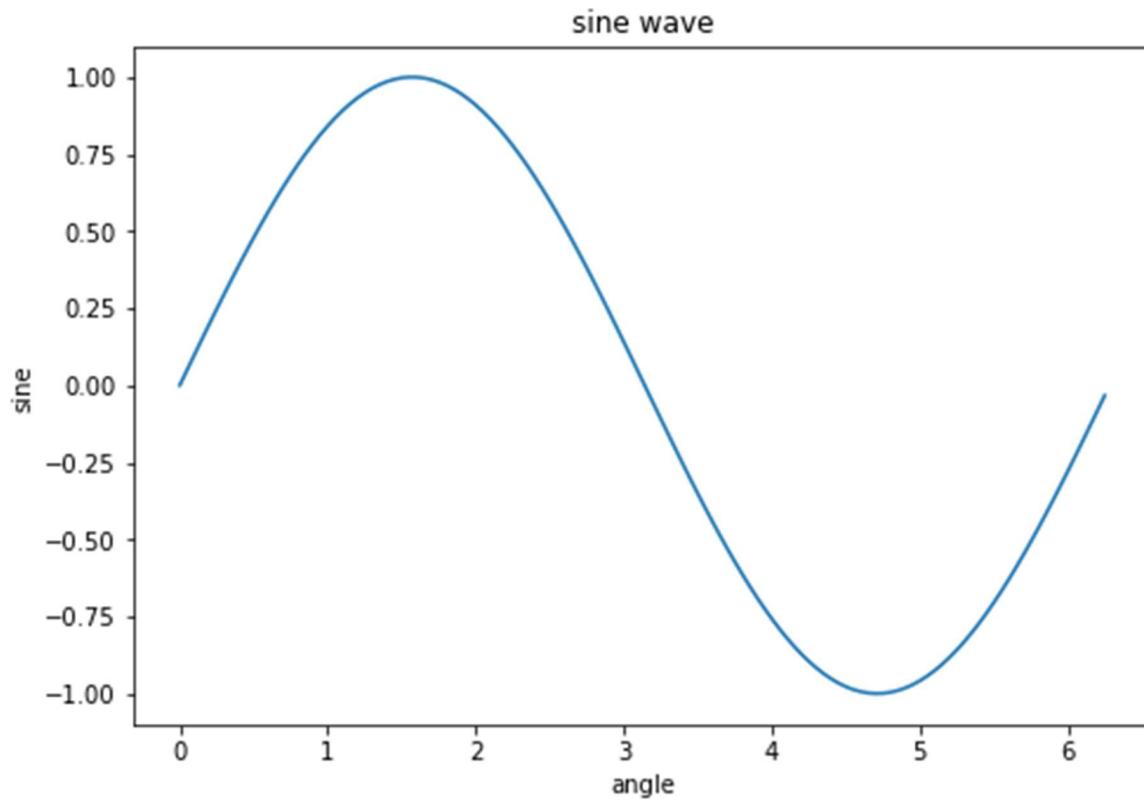
Cú pháp:

```
from matplotlib.backends.backend_agg import FigureCanvasAgg
from matplotlib.figure import Figure
```

Ví dụ:

```
from matplotlib.backends.backend_agg import FigureCanvasAgg
from matplotlib.figure import Figure
import numpy as np
import math
x = np.arange(0, math.pi*2, 0.05)
y = np.sin(x)
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
ax.plot(x,y)
ax.set_title("sine wave")
ax.set_xlabel('angle')
```

```
ax.set_ylabel('sine')
plt.show()
```



8.4. Các loại biểu đồ trong Matplotlib

Chúng ta sẽ bắt đầu với một vài biểu đồ đơn giản. Một đồ thị là đồ thị hai hoặc ba chiều thể hiện mối quan hệ qua các điểm, đường cong, hoặc các bars.

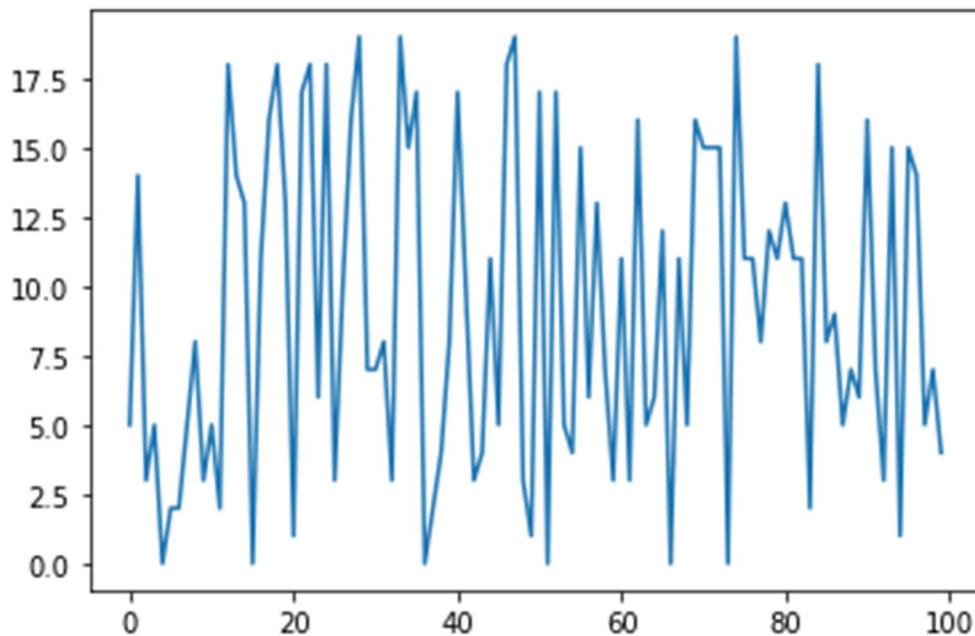
Chú ý: Nếu ta sử dụng ipython, ta cần thêm “%matplotlib inline” trước khi vẽ. Để hiển thị hình vẽ (plot) ta sử dụng plt.show() (Sử dụng Jupyter Notebook không cần phương thức này).

8.4.1. Line plot

Ví dụ: Vẽ một line graph từ một danh sách sinh ngẫu nhiên.

```
import matplotlib.pyplot as plt
import numpy as np
plt.plot(np.random.randint(20,size=100))
```

```
plt.show()
```

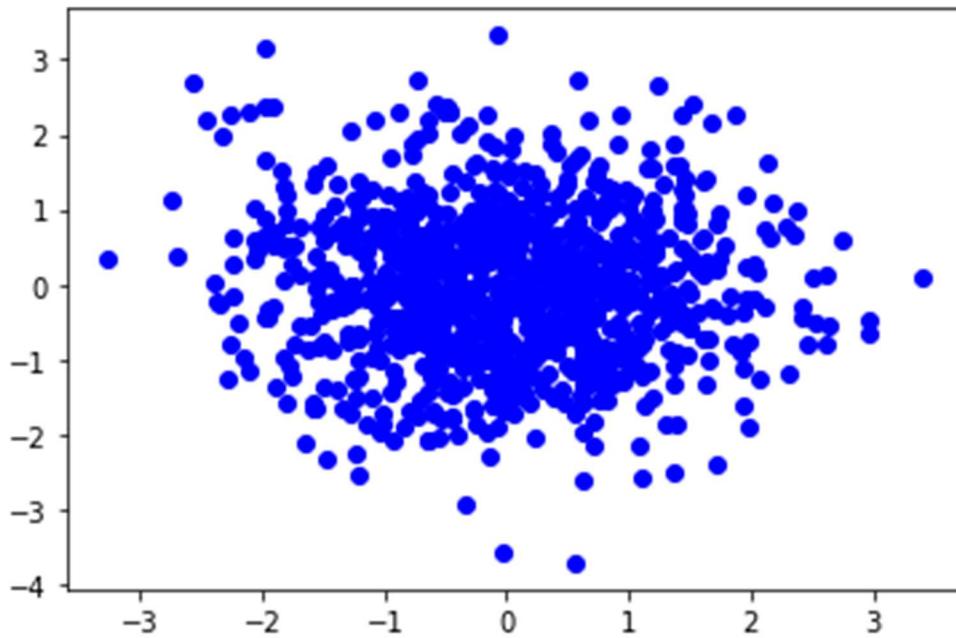


8.4.2. Scatter plot

Ví dụ: Ví dụ: Sinh ngẫu nhiên 1000 điểm và biểu diễn chúng trên đồ thị bằng đồ thị rác (Scatter).

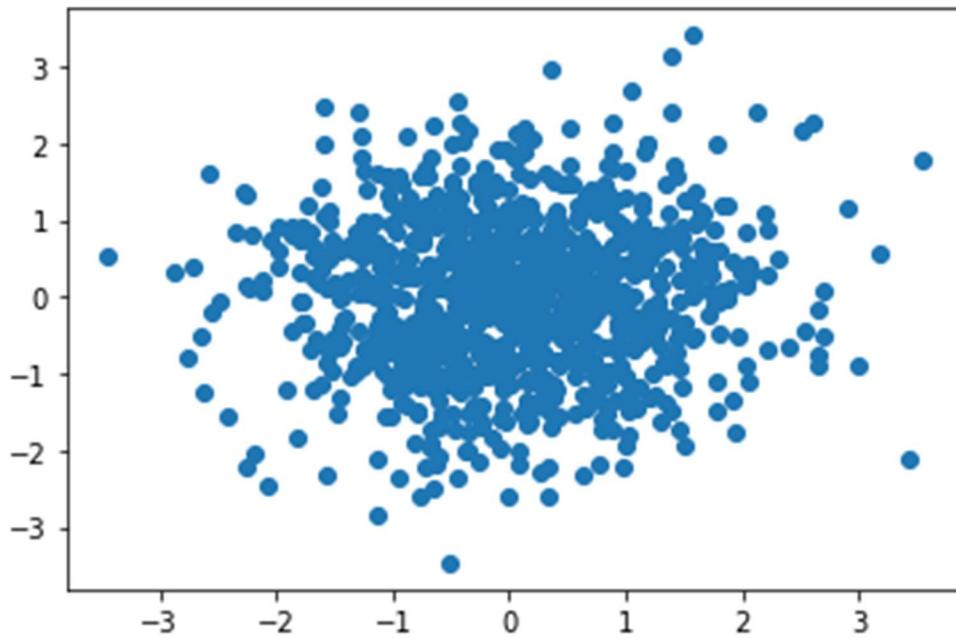
Import thư viện và sinh dữ liệu:

```
#Sử dụng plot() để vẽ đồ thị rác.
import matplotlib.pyplot as plt
import numpy as np
x = np.random.randn(1000)
y = np.random.randn(1000)
plt.plot(x,y,"ob")
plt.show()
```



#Hoặc ta cũng có thể sử dụng scatter()

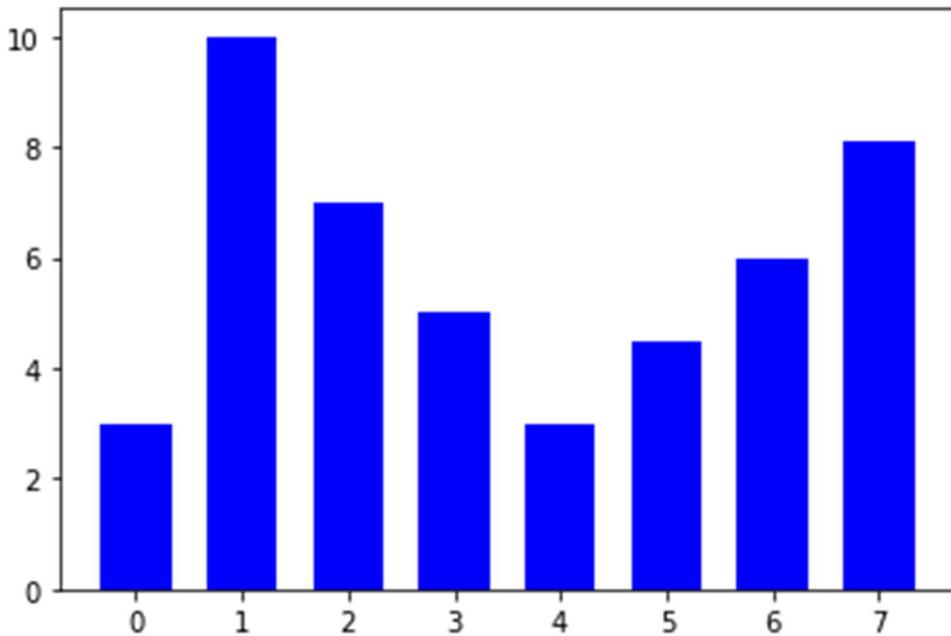
```
import matplotlib.pyplot as plt  
import numpy as np  
  
x = np.random.randn(1000)  
y = np.random.randn(1000)  
plt.scatter(x,y)  
plt.show()
```



8.4.3. Bar Chart

Ví dụ: Vẽ biểu đồ thanh (bar chart) từ danh ngẫu nhiên.

```
import matplotlib.pyplot as plt
import numpy as np
y = [3, 10, 7, 5, 3, 4.5, 6, 8.1]
N = len(y)
x = range(N)
width = 1/1.5
plt.bar(x, y, width, color="blue")
plt.show()
```



Tham khảo các loại biểu đồ khác tại: <https://matplotlib.org/tutorials/index.html>

8.5. Biểu đồ đường và biểu đồ rời rạc

Matplotlib's plot có thể được sử dụng để vẽ biểu đồ line, scatter (xem ví dụ về scatter và line trong phần 2). Chúng ta sẽ nhận thấy plot vô cùng mạnh mẽ, linh hoạt và có thể sử dụng dữ liệu ở nhiều định dạng khác nhau cho phép tùy biến mỗi line trong graphs chỉ với một chuỗi định dạng đơn giản hoặc qua tập các đối số.

plot(*args, **kwargs) khi đó args là một đối số có độ dài thay đổi, cho phép nhiều cặp x, y với một chuỗi định dạng tùy chọn. kwargs có thể được dùng để thiết lập thuộc tính đường (bất kỳ thuộc tính nào có phương thức set_ *). Bạn có thể sử dụng tính năng này để đặt tùy chỉnh độ rộng đường (linewidth), màu của markers, vv. Mặc định chúng sẽ được gán tự động bởi hệ thống.

Tham số định dạng của pyplot.plot là một string gồm các kí tự. Kí tự đầu tiên định nghĩa kiểu biểu diễn đường (line) hoặc kiểu biểu diễn rời rạc/tán xạ (scatter), trong khi đó kí tự thứ hai chọn màu cho đồ thị. Vị trí của các kí tự có tính chất giao hoán, tức là ta có thể hoán đổi vị trí cho nhau.

Khi ta không chỉ định chuỗi định dạng thì nó sẽ nhận giá trị “-b”.

Các ký tự chuỗi định dạng sau đây được chấp nhận để tùy chỉnh kiểu đường thẳng (được bôi đỏ) hoặc điểm đánh dấu (được bôi đen).

character	description	character	Description
'-'	solid line style	'3'	tri_left marker
'--'	dashed line style	'4'	tri_right marker
'-.'	dash-dot line style	's'	square marker
::	dotted line style	'p'	pentagon marker
'.'	point marker	'*'	star marker
,	pixel marker	'h'	hexagon1 marker
'o'	circle marker	'H'	hexagon2 marker
'v'	triangle_down marker	'+'	plus marker
'^'	triangle_up marker	'x'	x marker
'<'	triangle_left marker	'D'	diamond marker
'>'	triangle_right marker	'd'	thin_diamond marker
'l'	tri_down marker	' '	vline marker
'2'	tri_up marker	'_'	hline marker

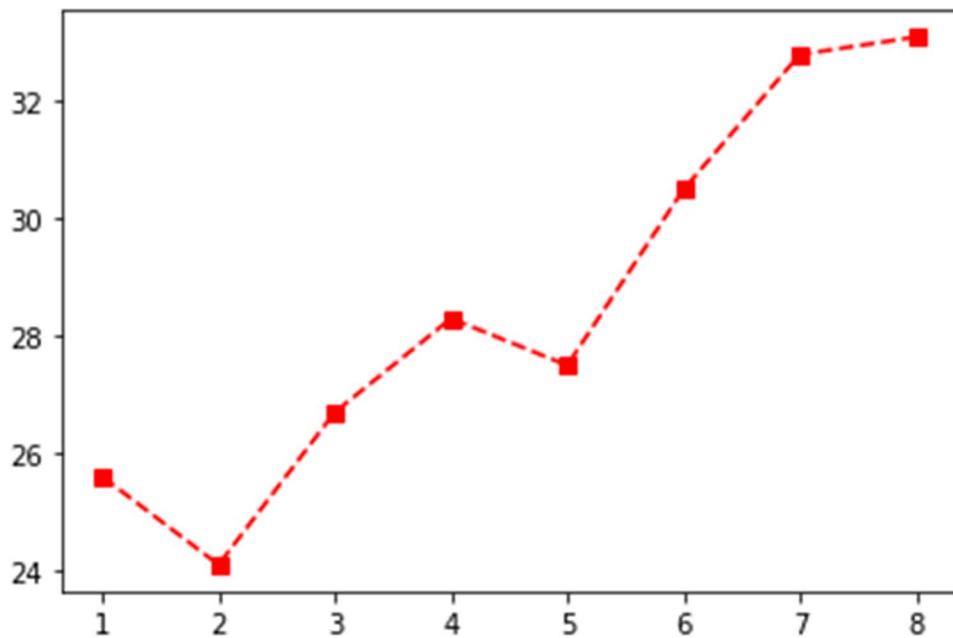
Màu sắc sẽ theo quy tắc sau:

	color	character	color
'b'	blue	'm'	magenta
'g'	green	'y'	yellow
'r'	red	'k'	black
'c'	cyan	'w'	white

Ví dụ: Dùng plot() vẽ line graph kết hợp với scatter để biểu diễn dữ liệu thời tiết trong 8 ngày.

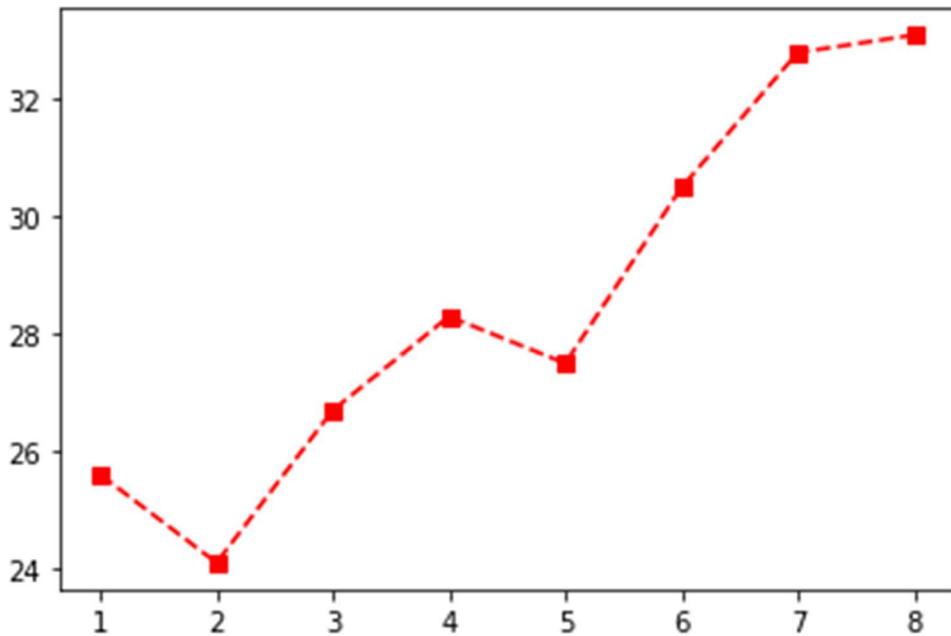
Import và khởi tạo dữ liệu:

```
import matplotlib.pyplot as plt
days = list(range(1,9))
celsius_values = [25.6, 24.1, 26.7, 28.3, 27.5, 30.5, 32.8, 33.1]
#Dùng chuỗi '--r' để vẽ line graph.
plt.plot(days, celsius_values,'--r')
#Dùng 'sr' để vẽ các điểm rời rạc.
plt.plot(days,celsius_values, 'sr')
plt.show()
```



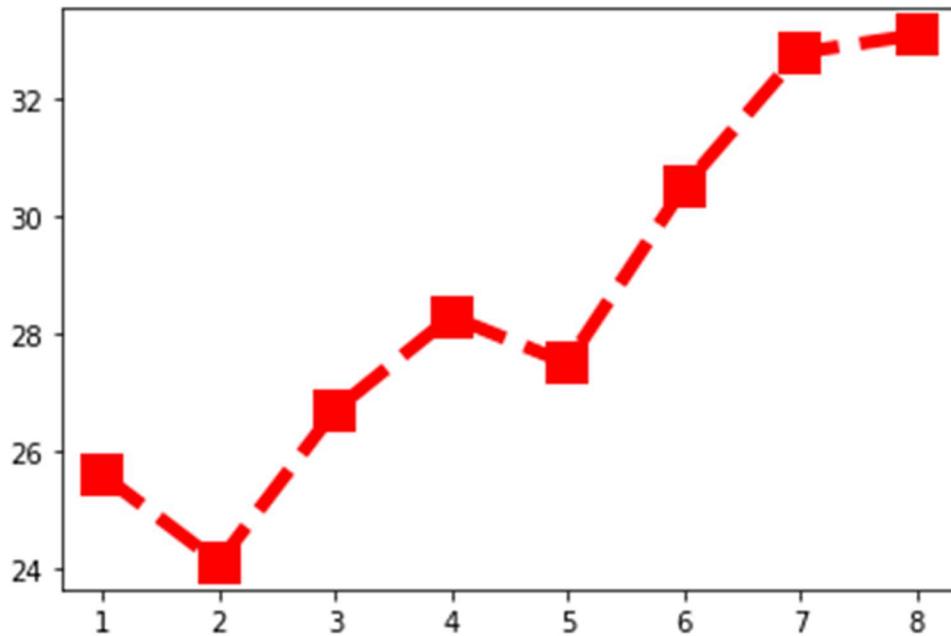
Trong ví dụ trên dùng tới hai lần gọi `plot()`, tuy nhiên với cùng một yêu cầu như trên điều đó là không cần thiết. Thay vào đó ta dùng định dạng “`--sr`” cũng cho ta kết quả tương tự.

```
import matplotlib.pyplot as plt
days = list(range(1,9))
celsius_values = [25.6, 24.1, 26.7, 28.3, 27.5, 30.5, 32.8, 33.1]
#Dùng “--sr”
plt.plot(days,celsius_values, '--sr')
plt.show()
```



‘r’ để thể hiện màu, ‘--’ để vẽ line graph, còn ‘s’ để vẽ các đánh dấu rời rạc hình vuông trên hình vẽ. Sử dụng tập đối số để tùy biến graph. Một số thuộc tính line2D hữu ích sau: ‘color’, ‘linewidth’ hoặc ‘lw’, ‘marker’, ‘linestyle’ or ‘ls’, ‘markeredgewidth’. Ta có thể dùng kết hợp với chuỗi định dạng hoặc chỉ sử dụng đối số phía trên để tùy biến cho đồ thị. Làm lại ví dụ phía trên và tăng kích thước line và makers với cách này.

```
import matplotlib.pyplot as plt
days = list(range(1,9))
celsius_values = [25.6, 24.1, 26.7, 28.3, 27.5, 30.5, 32.8, 33.1]
plt.plot(days,celsius_values,color = 'r', ls = '--', marker = 's', lw = 5,
markeredgewidth = 10)
plt.show()
```



Sự mạnh mẽ của `plot()` còn thể hiện bởi khả năng hỗ trợ vẽ cùng lúc nhiều dữ liệu với tùy chỉnh riêng cho từng nhóm chỉ trong một lần plot. Trong ví dụ sau, chúng ta sẽ vẽ 3 đường để biểu diễn sự biến thiên nhiệt độ trong ngày và nhiệt độ trung bình. Các bạn sẽ nhận thấy đường nét đứt màu đỏ để vẽ đường max, đường nét đứt màu xanh để biểu diễn min, và đường đậm đen ở giữa để biểu diễn nhiệt độ trung bình.

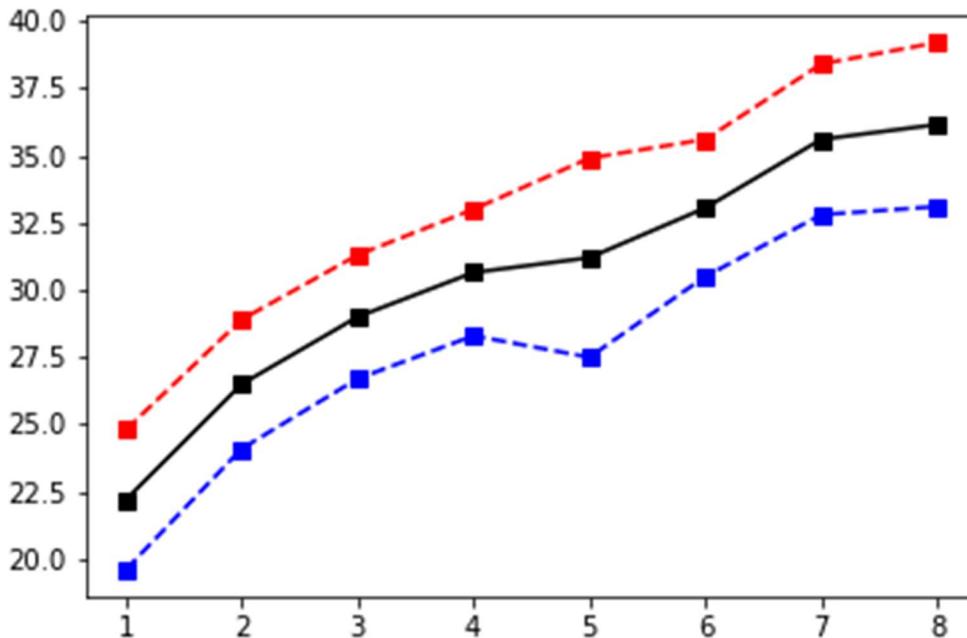
```
import matplotlib.pyplot as plt
import numpy as np
import math

#ham tinh trung binh cac phan tu cua list
def mean(numbers):
    return float(sum(numbers)) / max(len(numbers), 1)

days = list(range(1,9))
celsius_min = [19.6, 24.1, 26.7, 28.3, 27.5, 30.5, 32.8, 33.1]
celsius_max = [24.8, 28.9, 31.3, 33.0, 34.9, 35.6, 38.4, 39.2]

#tim trung binh cac phan tu cua 2 list
celsius_avg = [mean(i) for i in zip(celsius_min,celsius_max)]
```

```
#celsius_avg = map(lambda x: np.average(x),zip(celsius_min,celsius_max))
plt.plot(days,celsius_min,color = 'b', ls = '--', marker = 's')
plt.plot(days,celsius_max,color = 'r', ls = '--', marker = 's')
plt.plot(days,celsius_avg,color = 'k', ls = '-.', marker = 's')
plt.show()
```



Ngoài cách sử dụng plt.plot() để vẽ scatter graph như đã đề cập phía trên, chúng ta còn một lựa chọn khác sử dụng plt.scatter(). Cú pháp hàm.

```
matplotlib.pyplot.scatter(x, y, s=None, c=None, marker=None, cmap=None,
norm=None, vmin=None, vmax=None, alpha=None, linewidths=None,
verts=None, edgecolors=None, hold=None, data=None, **kwargs)
```

Tham khảo thêm website:

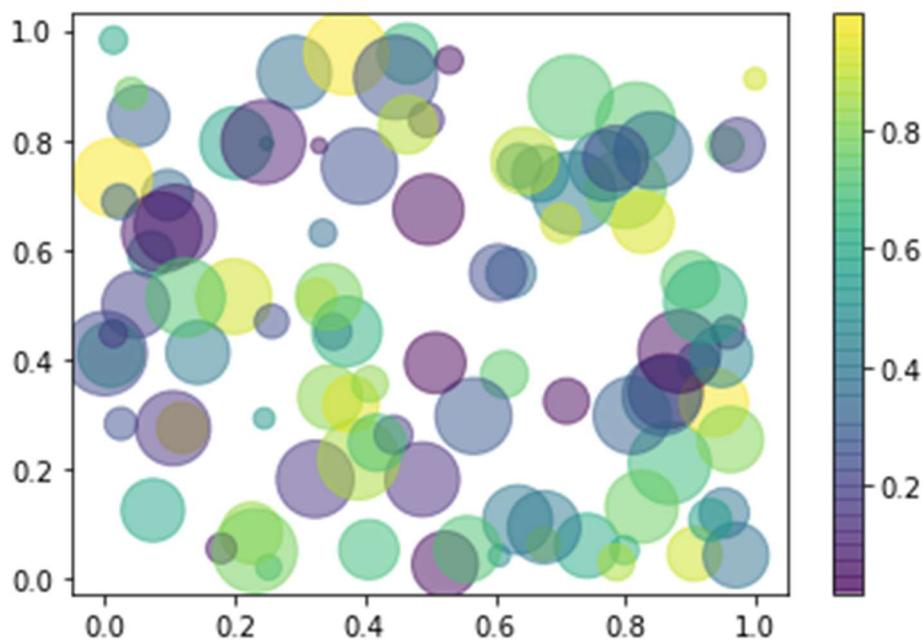
https://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html để biết thêm thông tin về ý nghĩa của từng đối số. Điểm lợi thế khi dùng phương thức này là giúp ta có thể biểu diễn nhanh các điểm rời rạc với kích thước và màu thay đổi. Ví dụ sau sẽ miêu tả tiện lợi này.

```
import matplotlib.pyplot as plt
import numpy as np
```

```

x = np.random.rand(100)
y = np.random.rand(100)
colors = np.random.rand(100)
sizes = 1000 * np.random.rand(100)
plt.scatter(x, y, c=colors, s=sizes, alpha=0.5,cmap='viridis')
plt.colorbar(); # show color scale
plt.show()

```



8.6. Multiple Subplots

Chúng ta đã đưa ra rất nhiều ví dụ để vẽ đồ thị trong các phần đã học. Một câu hỏi thường gặp là làm thế nào để có nhiều plots trong một đồ thị?

Trong trường hợp đơn giản nhất điều này có thể có nghĩa là, bạn có một đường cong và bạn muốn một đường cong in qua nó. Đây không phải là một vấn đề, bởi vì nó sẽ đủ để đưa hai plots trong kích bản của bạn, như chúng ta đã thấy trước đây. Trường hợp thú vị hơn là, nếu bạn muốn hai plots trong một cửa sổ. Trong một cửa sổ có nghĩa là chúng phải ở hai vị trí con riêng lẻ, nghĩa là không được in trên nhau. Ý tưởng là có nhiều hơn một đồ thị trong một cửa sổ và mỗi đồ thị xuất hiện trong subplot riêng của mình.

Trong tài liệu này sẽ hướng dẫn về hai nội dung để hoàn thành được ý tưởng trên: subplot; gridspec

8.6.1. Subplot

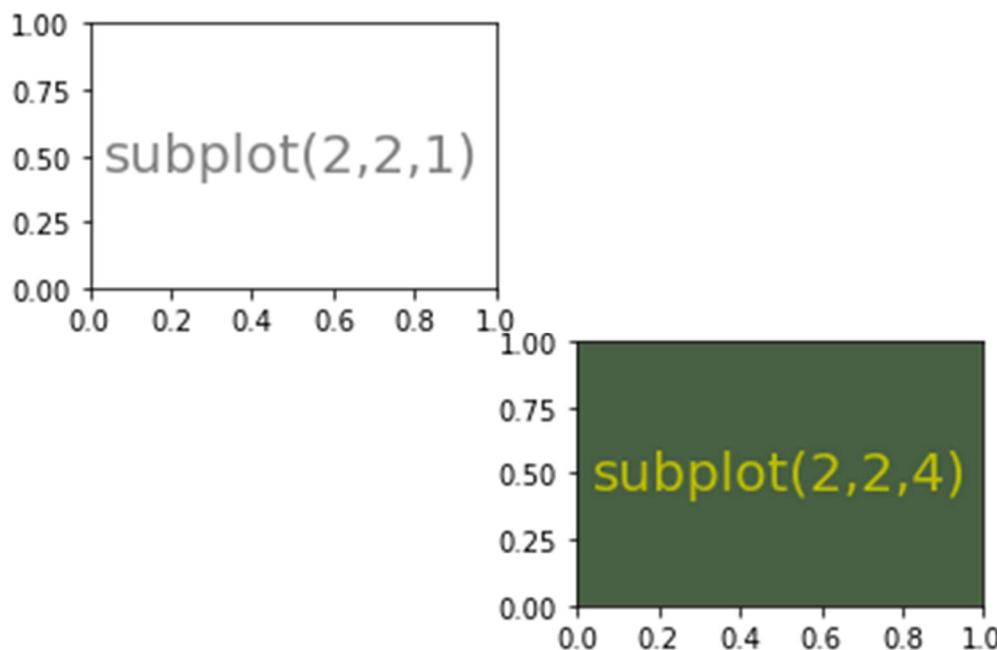
Cú pháp:

```
subplot(nrows, ncols, plot_number)
```

Nếu một subplot được áp dụng cho một con số, con số này sẽ được chia thành các trục con 'nrows' * 'ncols'. Tham số 'plot_number' xác định subplot mà hàm phải tạo ra. 'plot_number' có thể dao động từ 1 đến tối đa là 'nrows' * 'ncols'. Nếu các giá trị của ba tham số nhỏ hơn 10, thì hàm subplot có thể được gọi với một tham số int, trong đó hàng trăm đại diện cho 'nrows', hàng chục đại diện cho 'ncols' và hàng đơn vị biểu diễn 'plot_number'. Điều này có nghĩa là: Thay vì subplot (2, 3, 4) chúng ta có thể viết subplot (234). Ví dụ sau đây tạo ra hai subplot của lưới 2x2

```
import matplotlib.pyplot as plt
python_course_green = "#476042"
plt.figure(figsize=(6, 4))
plt.subplot(221) # equivalent to: plt.subplot(2, 2, 1)
plt.text(0.5, # x coordinate, 0 leftmost positioned, 1 rightmost
        0.5, # y coordinate, 0 topmost positioned, 1 bottommost
        'subplot(2,2,1)', # the text which will be printed
        horizontalalignment='center', # shortcut 'ha'
        verticalalignment='center', # shortcut 'va'
        fontsize=20, # can be named 'font' as well
        alpha=.5 # float (0.0 transparent through 1.0 opaque)
    )
plt.subplot(224, facecolor=python_course_green)
plt.text(0.5, 0.5, 'subplot(2,2,4)', ha='center', va='center', fontsize=20, color="y")
```

```
plt.show()
```



Mặc dù cách tiếp cận trước đó có thể chấp nhận được, nhưng theo tiếp cận hướng đối tượng bằng cách sử dụng các thẻ hiện của lớp figure. Chúng ta chứng minh điều này bằng cách viết lại ví dụ trước. Trong trường hợp này, chúng ta phải sử dụng phương thức "add_subplot" cho đối tượng figure.

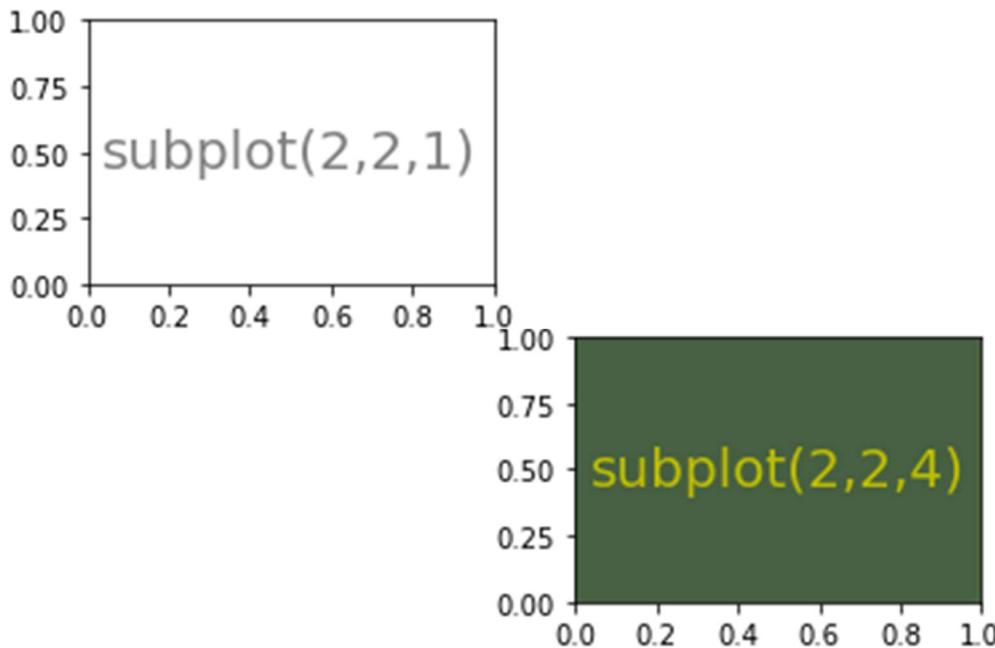
```
import matplotlib.pyplot as plt
python_course_green = "#476042"
fig = plt.figure(figsize=(6, 4))
sub1 = fig.add_subplot(221) # equivalent to: plt.subplot(2, 2, 1)
sub1.text(0.5, #x coordinate, 0 leftmost positioned, 1 rightmost
          0.5, #y coordinate, 0 topmost positioned, 1 bottommost
          'subplot(2,2,1)', # the text which will be printed
          horizontalalignment='center', # shortcut 'ha'
          verticalalignment='center', # shortcut 'va'
```

```

fontsize=20, # can be named 'font' as well
alpha=.5 #float (0.0 transparent through 1.0 opaque)
)

#sử dụng phương thức add_subplot
sub2 = fig.add_subplot(224, facecolor=python_course_green)
sub2.text(0.5, 0.5, subplot(2,2,4)'ha='center', va='center', fontsize=20, color="y")
plt.show()

```



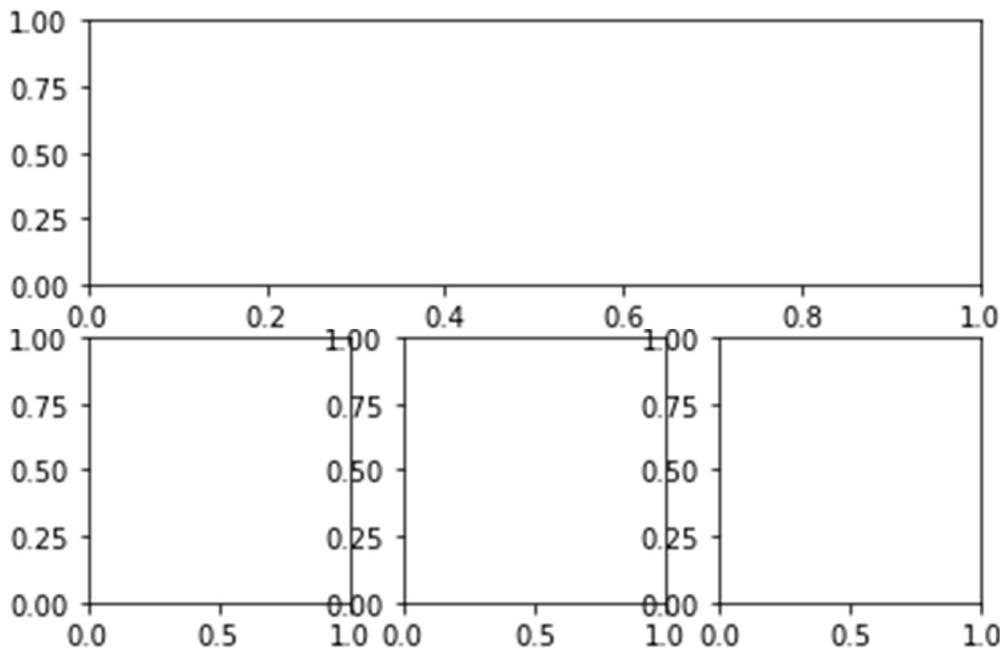
Để loại bỏ, hay tùy biến ticks trong các subplot, với cách tiếp cận đầu tiên ta có thể sử dụng phương thức `xticks`,`yticks()` của lớp `pyplot`, tuy nhiên cách sử dụng OOP ở ví dụ thứ hai ta cần sử dụng phương thức `set_xticks()`, và `set_yticks()` của lớp `Axes` trả về từ phương thức `add_subplot()`. Các bạn tự thực hành phần này nhé. Các bạn đoán thử xem làm thế nào có thể tạo được một figure với cách bố trí các subplots như sau: (các bạn chú ý phần màu đã trình bày trong lời giải sẽ giúp các bạn hiểu về cách chia blocks) Ví dụ 1:

```

import matplotlib.pyplot as plt
X = [ (2,1,1), (2,3,4), (2,3,5), (2,3,6) ]

```

```
for nrows, ncols, plot_number in X: plt.subplot(nrows, ncols, plot_number)
plt.show()
```

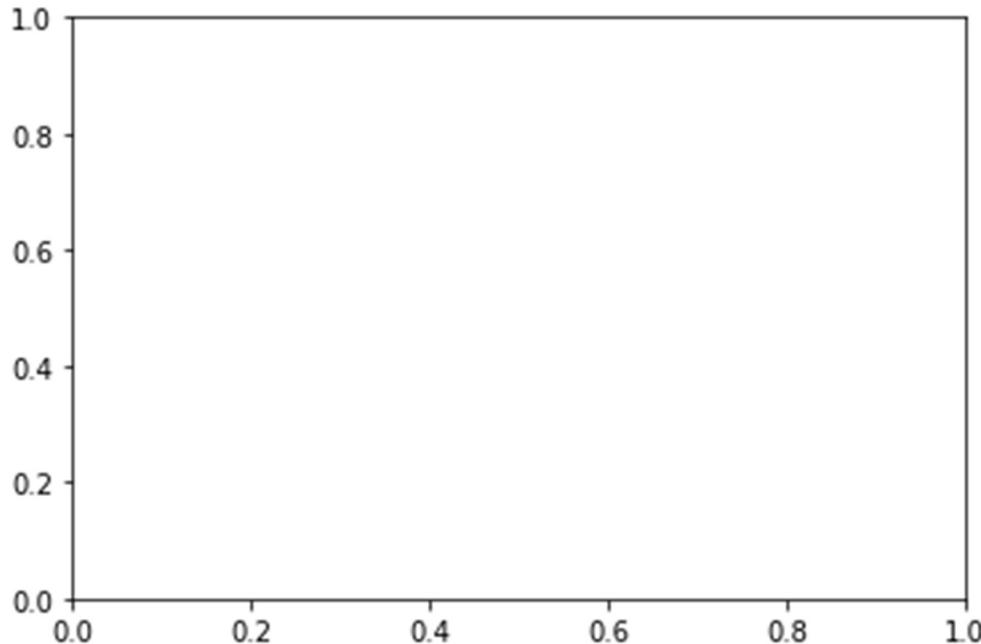


8.6.2. gridspec

'matplotlib.gridspec' chứa một lớp GridSpec. Nó có thể được sử dụng như là một thay thế cho subplot để xác định hình học của subplots được tạo ra. Ý tưởng cơ bản đằng sau GridSpec là một 'lưới'. Một lưới được thiết lập với số hàng và cột. Chúng ta phải xác định sau này, bao nhiêu lưới mà một subplot nên phủ.

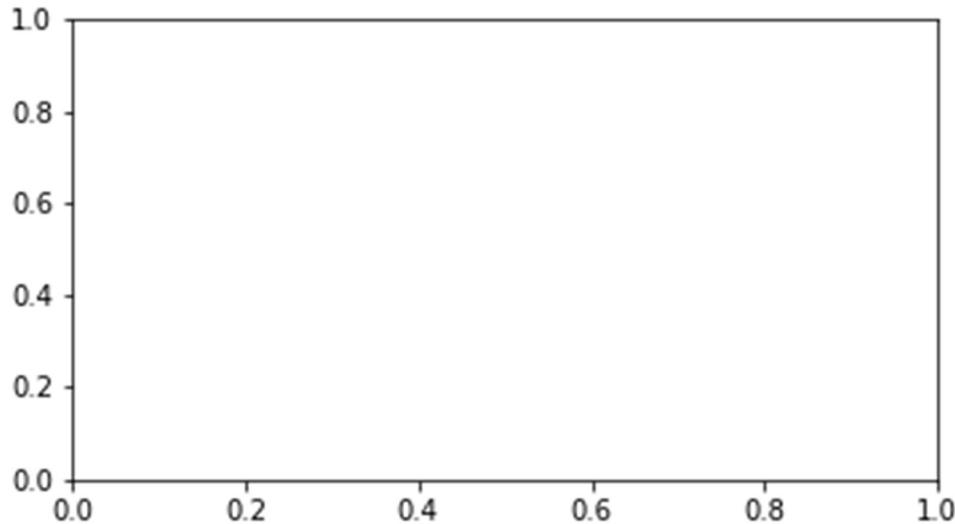
Ví dụ sau cho thấy trường hợp nhỏ hoặc đơn giản nhất, tức là lưới 1x1

```
import matplotlib.pyplot as plt
from matplotlib.gridspec import GridSpec
fig = plt.figure()
gs = GridSpec(1, 1)
ax = fig.add_subplot(gs[0,0])
plt.show()
```



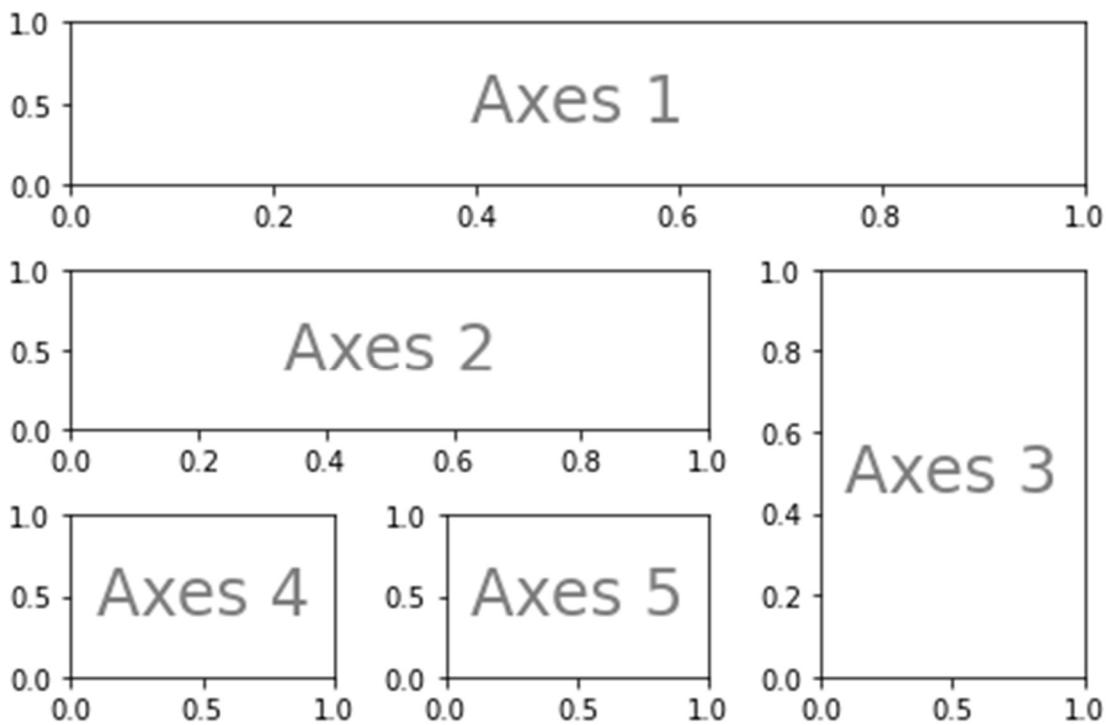
Chúng ta có thể sử dụng một số thông số của Gridspec, ví dụ: Chúng ta có thể xác định rằng biểu đồ của chúng ta sẽ bắt đầu ở mức 20% từ đáy và 15% ở phía bên trái của vùng hình có sẵn:

```
import matplotlib.pyplot as plt
from matplotlib.gridspec import GridSpec
fig = plt.figure()
gs = GridSpec(1, 1,
              bottom=0.2,
              left=0.15,
              top=0.8)
ax = fig.add_subplot(gs[0,0])
plt.show()
```



Ví dụ 1: Thiết kế lưới phức tạp hơn:

```
import matplotlib.gridspec as gridspec
import matplotlib.pyplot as pl
pl.figure(figsize=(6, 4))
G = gridspec.GridSpec(3, 3)
axes_1 = pl.subplot(G[0, :])
pl.text(0.5, 0.5, 'Axes 1', ha='center', va='center', size=24, alpha=.5)
axes_2 = pl.subplot(G[1, :-1])
pl.text(0.5, 0.5, 'Axes 2', ha='center', va='center', size=24, alpha=.5)
axes_3 = pl.subplot(G[1:, -1])
pl.text(0.5, 0.5, 'Axes 3', ha='center', va='center', size=24, alpha=.5)
axes_4 = pl.subplot(G[-1, 0])
pl.text(0.5, 0.5, 'Axes 4', ha='center', va='center', size=24, alpha=.5)
axes_5 = pl.subplot(G[-1, -2])
pl.text(0.5, 0.5, 'Axes 5', ha='center', va='center', size=24, alpha=.5)
pl.tight_layout()
pl.show()
```



8.6.3. Vẽ một plot bên trong một plot khác

Sử dụng phương thức:

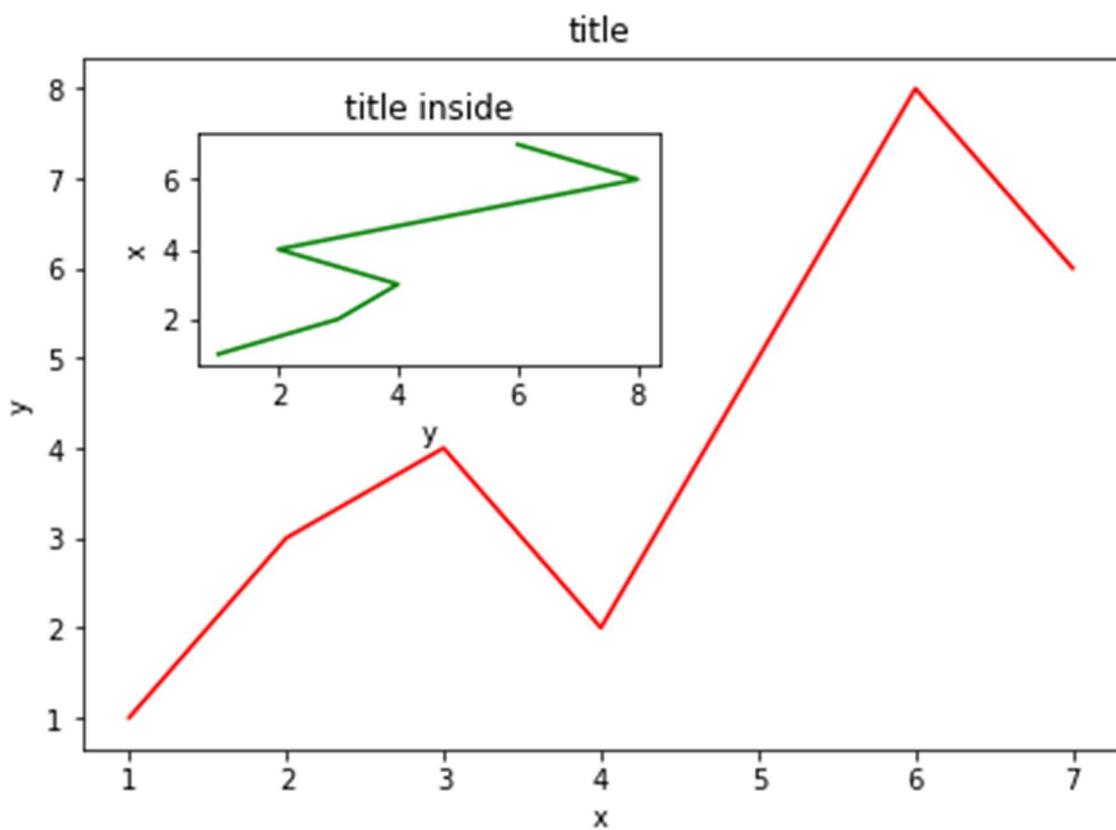
`add_axes`

Thêm một trục tại vị trí rect [trái, dưới, chiều rộng, chiều cao] nơi mà tất cả các số lượng nằm trong các phân số của chiều rộng và chiều cao.

```
import numpy as np
import matplotlib.pyplot as plt
fig = plt.figure()
X = [1, 2, 3, 4, 5, 6, 7]
Y = [1, 3, 4, 2, 5, 8, 6]
axes1 = fig.add_axes([0.1, 0.1, 0.9, 0.9]) # main axes
axes2 = fig.add_axes([0.2, 0.6, 0.4, 0.3]) # inset axes
```

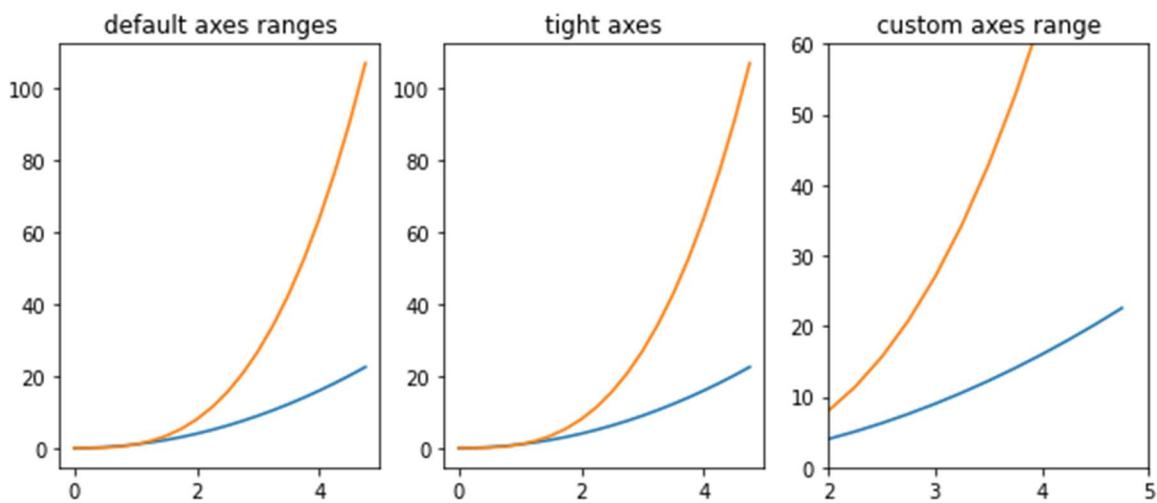
```
# main figure
axes1.plot(X, Y, 'r')
axes1.set_xlabel('x')
axes1.set_ylabel('y')
axes1.set_title('title')

# insert
axes2.plot(Y, X, 'g')
axes2.set_xlabel('y')
axes2.set_ylabel('x')
axes2.set_title('title inside')
plt.show()
```



Có thể định cấu hình các dãy của các trục. Điều này có thể được thực hiện bằng cách sử dụng phương thức `set_ylim()` và `set_xlim()` trong đối象 Axis.

```
import numpy as np
import matplotlib.pyplot as plt
fig, axes = plt.subplots(1, 3, figsize=(10, 4))
x = np.arange(0, 5, 0.25)
axes[0].plot(x, x**2, x, x**3)
axes[0].set_title("default axes ranges")
axes[1].plot(x, x**2, x, x**3)
axes[1].axis('tight')
axes[1].set_title("tight axes")
axes[2].plot(x, x**2, x, x**3)
axes[2].set_ylim(0, 60)
axes[2].set_xlim(2, 5)
axes[2].set_title("custom axes range");
plt.show()
```



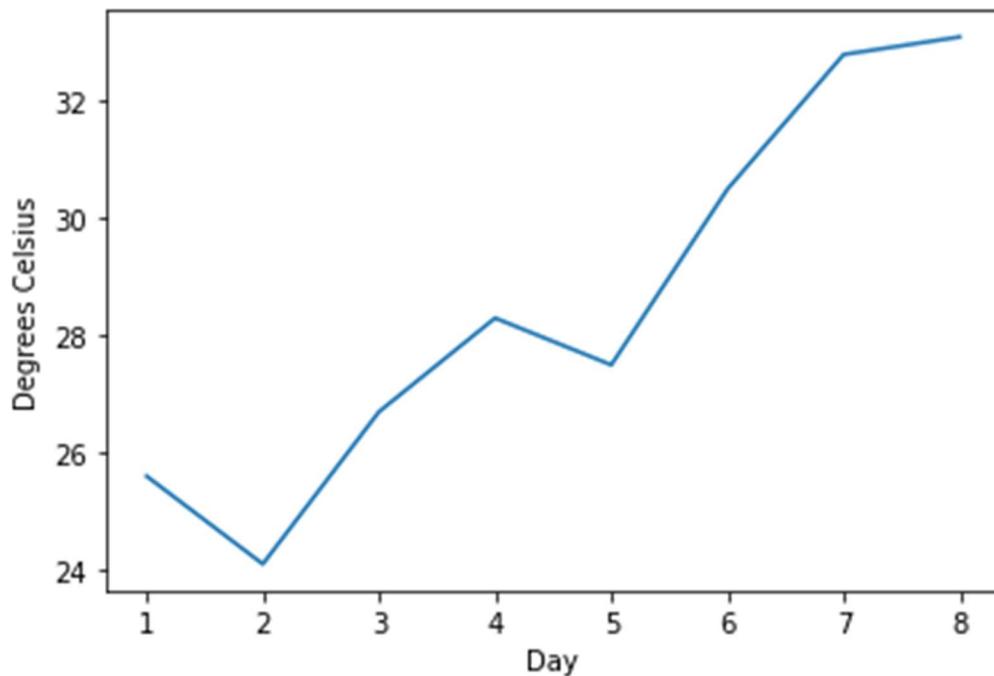
8.7. Cảnh sửa các thành phần của đồ thị

Người dùng có rất nhiều quyền kiểm soát thuộc tính text (kích thước phông chữ, font weight, vị trí và màu sắc, v.v ..) với các giá trị mặc định hợp lý được đặt trong tệp *.rc. Và đáng kể, đối với những người quan tâm đến các hình vẽ mô tả công thức toán học, matplotlib thực hiện một số lượng lớn các biểu tượng toán học và các lệnh của TeX, hỗ trợ các biểu thức toán học bất cứ nơi nào trong hình của bạn. Các phương thức text cơ bản được Axes hỗ trợ: .text() - thêm text tại một vị trí tùy ý để các trục. .set_xlabel() - thêm một nhãn vào trục x. .set_ylabel() - thêm một nhãn vào trục y. .set_title() - thêm tiêu đề cho các trục. .annotate() - thêm một chú thích, với mũi tên tùy chọn, đến các Axes Hoặc được Figure hỗ trợ: .text() - thêm text tại vị trí tùy ý vào hình. .suptitle() - thêm tiêu đề vào hình.

Tìm hiểu chi tiết qua ví dụ dưới đây:

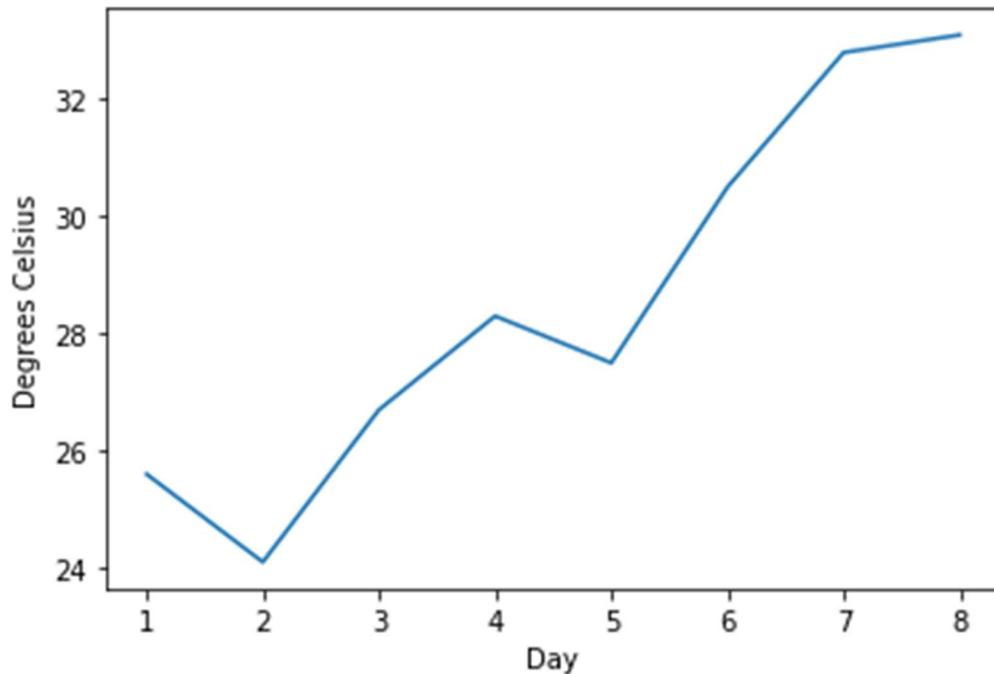
8.7.1. Thêm nhãn cho các trục để mô tả ý nghĩa của trục

```
import matplotlib.pyplot as plt
days = list(range(1,9))
celsius_values = [25.6, 24.1, 26.7, 28.3, 27.5, 30.5, 32.8, 33.1]
fig, ax = plt.subplots()
ax.plot(days, celsius_values)
ax.set_xlabel('Day')
ax.set_ylabel('Degrees Celsius')
plt.show()
```



Ngoài ra, điều này có thể được thực hiện trực tiếp thông qua các phương thức xlabel() và ylabel() của pyplot.

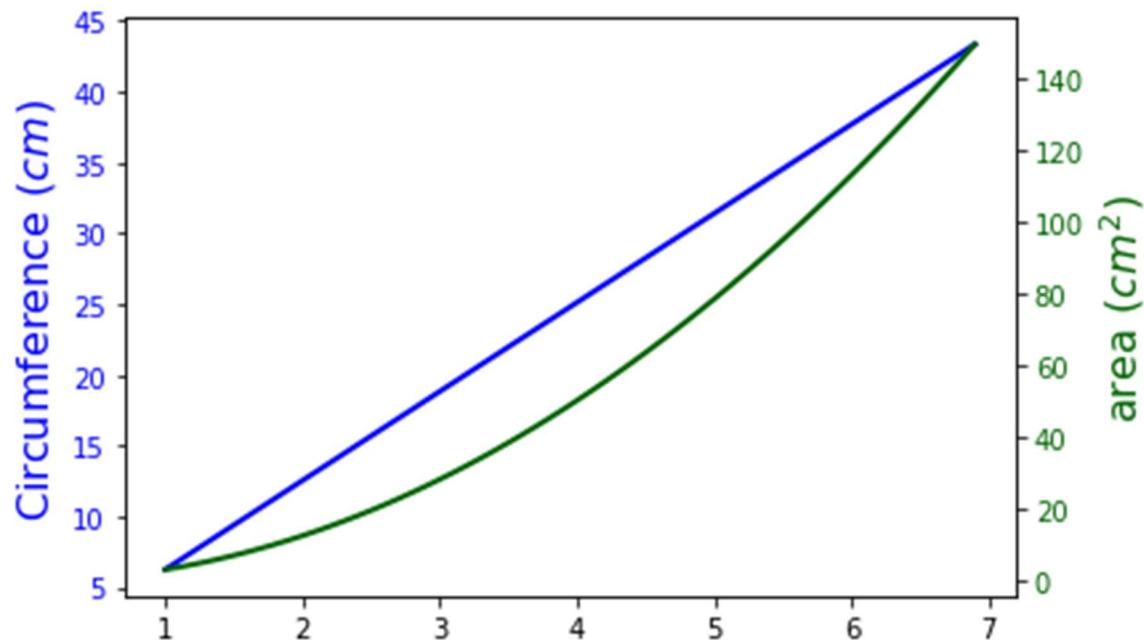
```
import matplotlib.pyplot as plt
days = list(range(1,9))
celsius_values = [25.6, 24.1, 26.7, 28.3, 27.5, 30.5, 32.8, 33.1]
plt.plot(days, celsius_values)
plt.xlabel('Day')
plt.ylabel('Degrees Celsius')
plt.show()
```



Vấn đề tiếp theo là biểu diễn nhiều đường trên hai thang đo khác nhau trên cùng một đồ thị. Trong ví dụ dưới đây: thang đo bên trái là cm và thang đo bên phải cm². Và mầu chốt để thực hiện chính là dòng bôi đen.

```
import numpy as np
import matplotlib.pyplot as plt
fig, ax1 = plt.subplots()
x = np.arange(1,7,0.1)
ax1.plot(x, 2 * np.pi * x, lw=2, color="blue")
ax1.set_ylabel(r"Circumference $(cm)$", fontsize=16, color="blue")
for label in ax1.get_yticklabels():
    label.set_color("blue")
ax2 = ax1.twinx()
ax2.plot(x, np.pi * x ** 2, lw=2, color="darkgreen")
ax2.set_ylabel(r"area $(cm^2)$", fontsize=16, color="darkgreen")
for label in ax2.get_yticklabels():
    label.set_color("darkgreen")
```

```
plt.show()
```



8.7.2. Text, Transforms và Text Position

Tạo ra một graph có thể giúp người đọc thấy được một câu chuyện từ các con số khô khan. Trong một số trường hợp, câu chuyện này có thể được kể một cách hoàn toàn bằng hình ảnh, mà không cần thêm văn bản, nhưng một số khác là cần thiết. Có lẽ các loại chú thích cơ bản nhất bạn sẽ sử dụng là các nhãn và tiêu đề trực, nhưng các tùy chọn vượt xa điều này. Chúng ta hãy cùng xem xét một số dữ liệu và cách chúng ta có thể hình dung và chú thích nó để giúp chuyển tải thông tin thú vị trong bài học này và các bài học tiếp theo. Dữ liệu ảnh hưởng của ngày nghỉ đến sinh nở tại Mỹ.

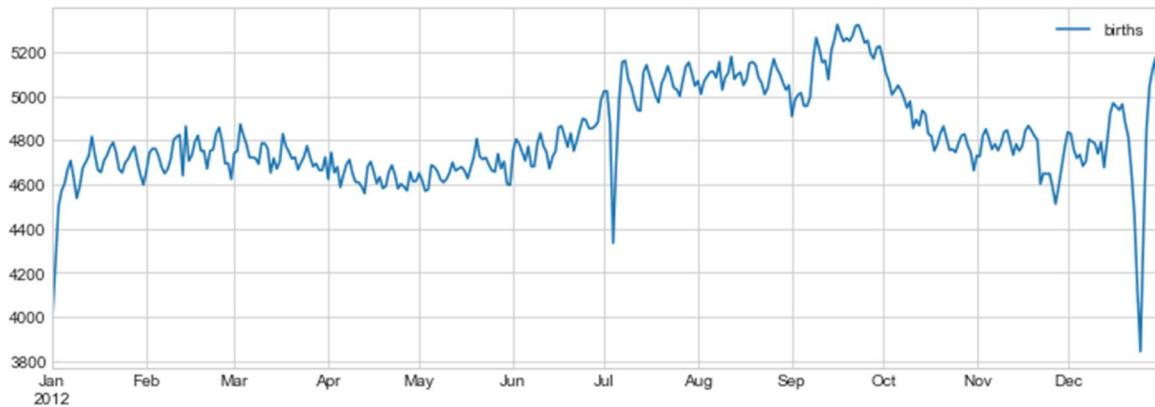
Bước 1: Quá trình clean data

```
import matplotlib.pyplot as plt
import matplotlib as mpl
plt.style.use('seaborn-whitegrid')
import numpy as np
```

```
import pandas as pd  
births = pd.read_csv('https://raw.githubusercontent.com/jakevdp/data-  
CDCbirths/master/births.csv')  
  
quartiles = np.percentile(births['births'], [25, 50, 75])  
mu, sig = quartiles[1], 0.74 * (quartiles[2] - quartiles[0])  
births = births.query('(births > @mu - 5 * @sig) & (births < @mu + 5 * @sig)')  
  
births['day'] = births['day'].astype(int)  
  
births.index = pd.to_datetime(10000 * births.year +  
    100 * births.month +  
    births.day, format='%Y%m%d')  
births_by_date = births.pivot_table('births',  
    [births.index.month, births.index.day])  
births_by_date.index = [pd.datetime(2012, month, day)  
    for (month, day) in births_by_date.index]
```

Bước 2: Vẽ hình

```
fig, ax = plt.subplots(figsize=(12, 4))
births_by_date.plot(ax=ax);
```



Bước 3: Chú thích

```
fig, ax = plt.subplots(figsize=(12, 4))
births_by_date.plot(ax=ax)

# Add labels to the plot
style = dict(size=10, color='gray')

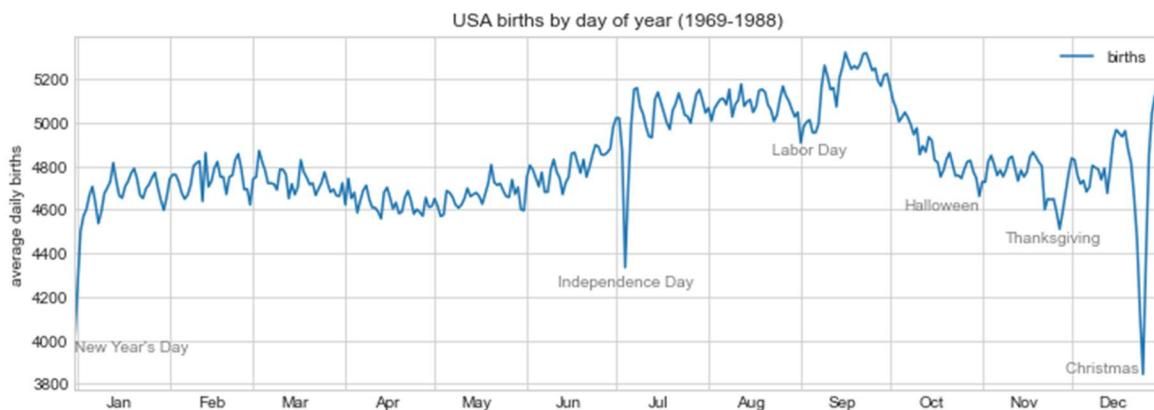
ax.text('2012-1-1', 3950, "New Year's Day", **style)
ax.text('2012-7-4', 4250, "Independence Day", ha='center', **style)
ax.text('2012-9-4', 4850, "Labor Day", ha='center', **style)
ax.text('2012-10-31', 4600, "Halloween", ha='right', **style)
ax.text('2012-11-25', 4450, "Thanksgiving", ha='center', **style)
ax.text('2012-12-25', 3850, "Christmas ", ha='right', **style)

# Label the axes
ax.set(title='USA births by day of year (1969-1988)',
```

```
ylabel='average daily births')
```

Format the x axis with centered month labels

```
ax.xaxis.set_major_locator(mpl.dates.MonthLocator())
ax.xaxis.set_minor_locator(mpl.dates.MonthLocator(bymonthday=15))
ax.xaxis.set_major_formatter(plt.NullFormatter())
ax.xaxis.set_minor_formatter(mpl.dates.DateFormatter('%h'));
```

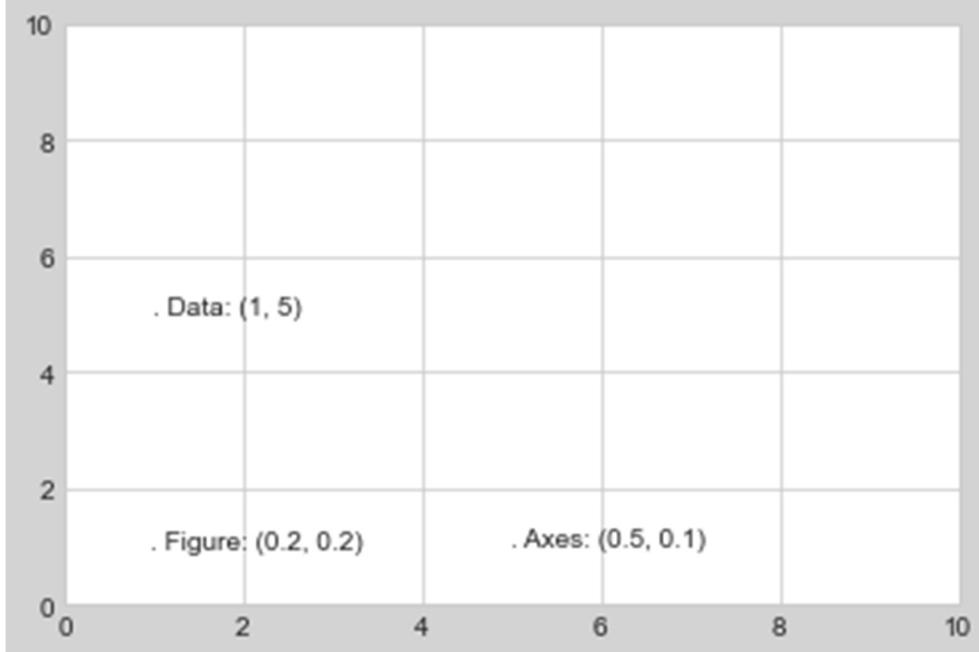


Trong ví dụ trước, chúng ta đã đặt chú thích vào các vị trí dữ liệu. Đôi khi tốt hơn là đặt chú thích vào một vị trí trên các trục hoặc hình, độc lập với dữ liệu. Trong Matplotlib, điều này được thực hiện bằng cách sửa transform trong ax.text()/plt.text(). Bất kỳ display framework nào cũng cần một vài cơ chế để chuyển đổi giữa các hệ tọa độ. Ví dụ, một điểm dữ liệu tại $(x, y) = (1, 1)$ cần phải được đại diện ở một vị trí nhất định trên hình đó, do đó cần phải được biểu diễn bằng các điểm ảnh trên màn hình. Về mặt toán học, các phép biến đổi tọa độ tương đối đơn giản, và Matplotlib có một tập hợp các công cụ phát triển tốt mà nó sử dụng nội bộ để thực hiện chúng (những công cụ này có thể được khám phá trong submodule matplotlib.transforms). Người dùng thông thường hiếm khi cần phải lo lắng về các chi tiết của các biến đổi này, nhưng nó là kiến thức hữu ích khi xem xét việc sắp xếp text trên figure. Có ba biến đổi được xác định trước có thể hữu ích trong tình huống này: + ax.transData: Chuyển đổi kết hợp với dữ liệu tọa độ + ax.transAxes: Chuyển đổi liên kết với các trục (theo đơn vị kích thước trục) +

fig.transFigure: Chuyển đổi kết hợp với hình vẽ (trong các đơn vị kích thước con số) Cùng nhìn nhận qua ví dụ sau:

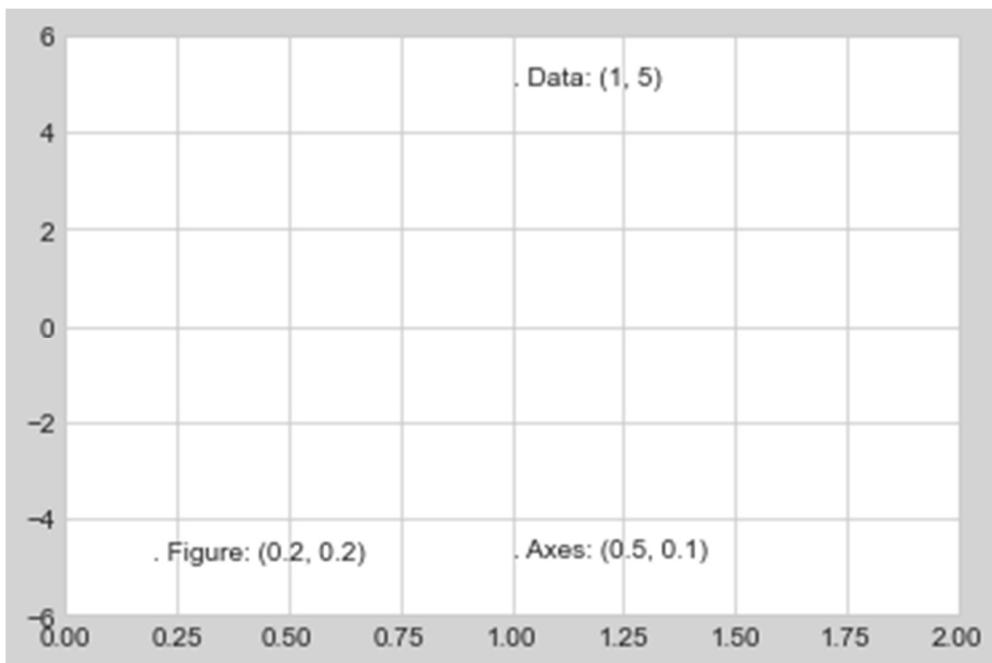
```
fig, ax = plt.subplots(facecolor='lightgray')
ax.axis([0, 10, 0, 10])

# transform=ax.transData is the default, but we'll specify it anyway
ax.text(1, 5, ". Data: (1, 5)", transform=ax.transData)
ax.text(0.5, 0.1, ". Axes: (0.5, 0.1)", transform=ax.transAxes)
ax.text(0.2, 0.2, ". Figure: (0.2, 0.2)", transform=fig.transFigure);
```



Cách giải thích sẽ nhìn theo hướng dữ liệu: + Data(1,5) lựa chọn transform=ax.transData, điểm được biểu diễn tại (1,5). + Axes(0.5,0.1) lựa chọn transform=ax.transAxes, điểm tương ứng theo hướng dữ liệu sẽ là (5,1). Lưu ý: Nếu chúng ta thay đổi các giới hạn trục, thì chỉ có tọa độ transData sẽ bị ảnh hưởng, trong khi các tọa độ khác vẫn đứng yên:

```
ax.set_xlim(0, 2)
ax.set_ylim(-6, 6)
fig
```



Xây dựng chú thích trên đồ thị có mũi tên. Điều này giúp người xem có thể hình dung ra nội dung chính xác và trực quan hơn. Trong nội dung môn học này sử dụng phương thức plt.annotate().

```
%matplotlib inline

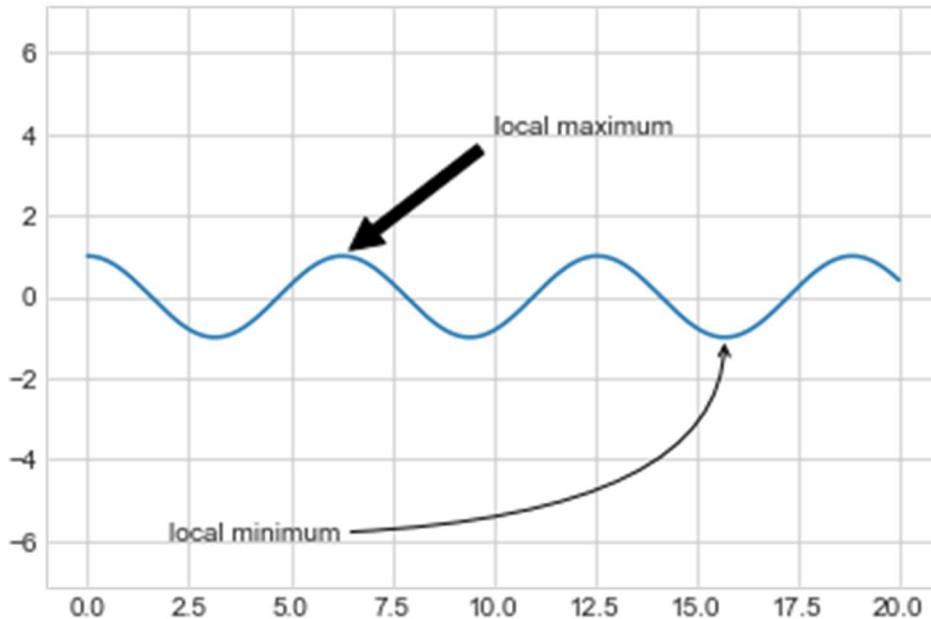
fig, ax = plt.subplots()

x = np.linspace(0, 20, 1000)
ax.plot(x, np.cos(x))
ax.axis('equal')

ax.annotate('local maximum', xy=(6.28, 1), xytext=(10, 4),
            arrowprops=dict(facecolor='black', shrink=0.05))

ax.annotate('local minimum', xy=(5 * np.pi, -1), xytext=(2, -6),
```

```
arrowprops=dict(arrowstyle="->",
    connectionstyle="angle3,angleA=0,angleB=-90"));
```



8.8. Tùy chỉnh spines và ticks của đồ thị

8.8.1. Di chuyển đường biên và đánh bóng các ký hiệu trực

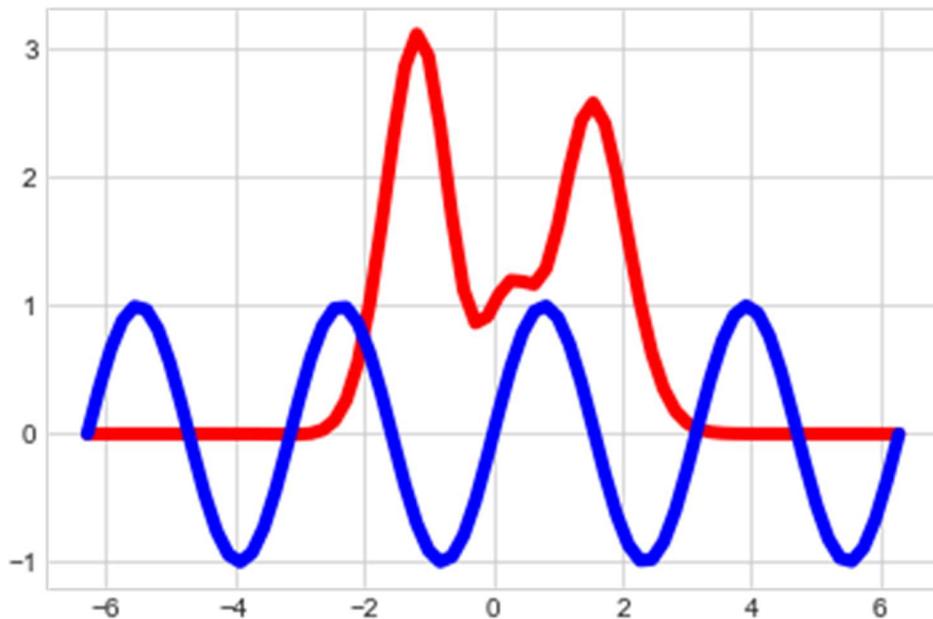
Axis spine - đường ghi nhận ranh giới khu vực dữ liệu. Spine là đường kết nối các dấu ticks trên trực và ghi nhận ranh giới của khu vực dữ liệu. Chúng có thể được đặt tùy ý.

Nhưng trước khi bắt đầu với ví dụ đầu tiên, chúng ta cần xét đến phương thức `gca()`, nó trả về đối tượng `Axes` hiện tại của `figure`.

Ví dụ 1: Không dịch chuyển các trực

```
import numpy as np
import matplotlib.pyplot as plt
X = np.linspace(-2 * np.pi, 2 * np.pi, 70, endpoint=True)
F1 = np.sin(2*X)
F2 = (2*X**5 + 4*X**4 - 4.8*X**3 + 1.2*X**2 + X + 1)*np.exp(-X**2)
```

```
plt.plot(X, F2, '-r', linewidth=5)
plt.plot(X, F1, '-b', linewidth=5)
plt.show()
```

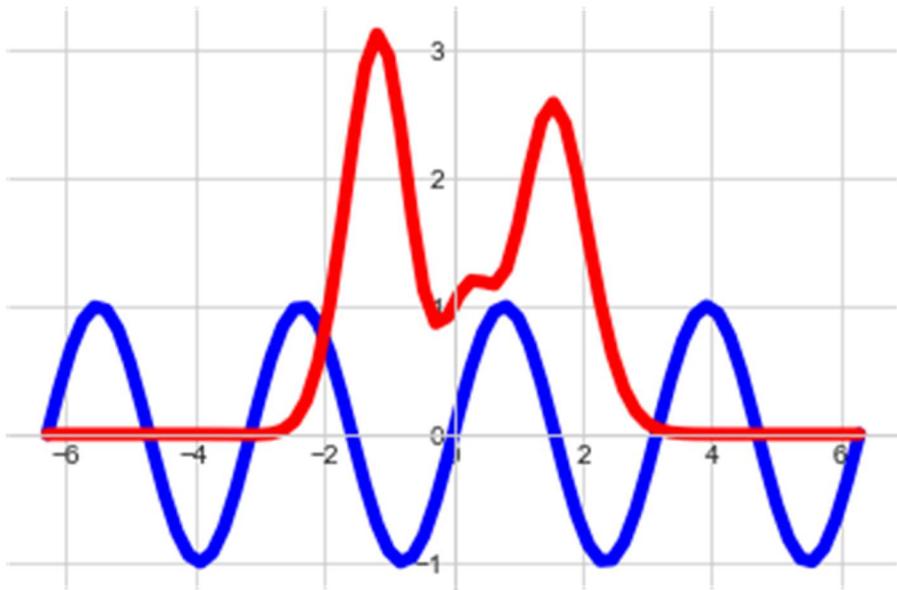


Ví dụ 2: Dịch chuyển các trục

```
import numpy as np
import matplotlib.pyplot as plt
X = np.linspace(-2 * np.pi, 2 * np.pi, 70, endpoint=True)
F1 = np.sin(2* X)
F2 = (2*X**5 + 4*X**4 - 4.8*X**3 + 1.2*X**2 + X + 1)*np.exp(-X**2)
# get the current axes, creating them if necessary:
ax = plt.gca()
# making the top and right spine invisible:
ax.spines['top'].set_color('none')
ax.spines['right'].set_color('none')
```

```
# moving bottom spine up to y=0 position:
ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position((0,0))

# moving left spine to the right to position x == 0:
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position((0,0))
plt.plot(X, F1,'-b',linewidth=5)
plt.plot(X, F2,'-r',linewidth=5)
plt.show()
```

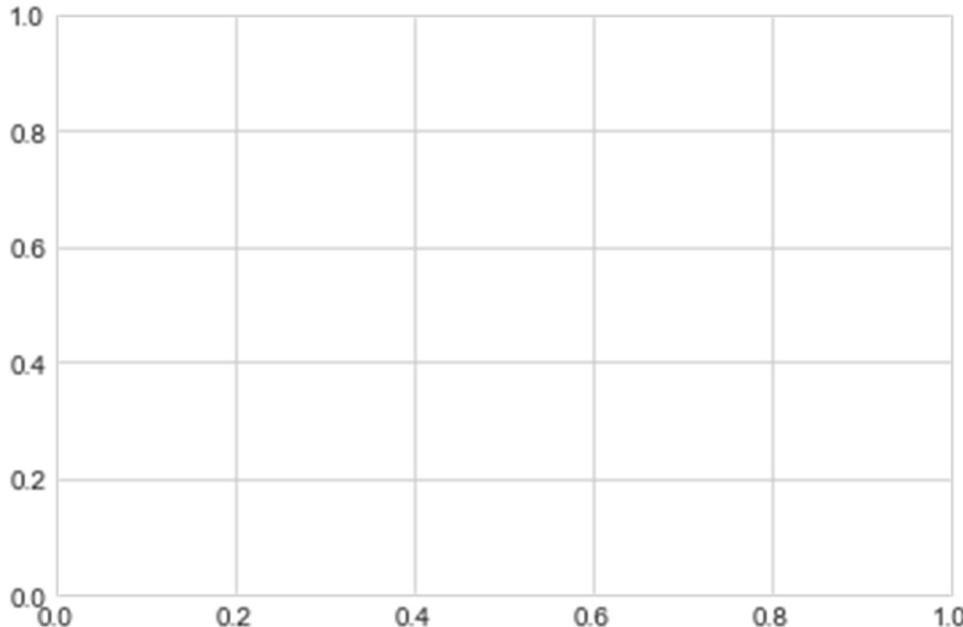


8.8.2. Tùy chỉnh các ticks

Matplotlib tự động điều chỉnh khoảng cách trên trục. `xticks()`,`yticks()` là một phương thức của `pyplot`, có thể được sử dụng để lấy hoặc thiết lập vị trí đánh dấu (ticks) hiện tại và nhãn (label) cho trục x và y tương ứng.

Ví dụ sau mô tả lấy thông số vị trí đánh dấu của hai trục x và y.

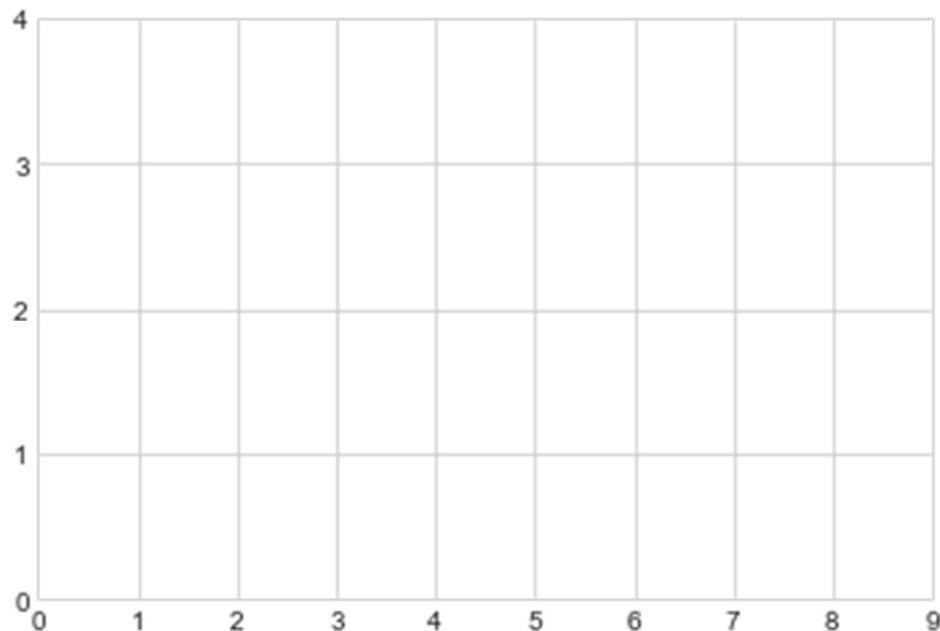
```
import matplotlib.pyplot as plt  
ax = plt.gca()  
locs, labels = plt.xticks()  
print(locs, labels)  
locs, labels = plt.yticks()
```



Chúng ta có thể dùng xticks(), yticks() để thiết lập thông số mới:

```
import numpy as np  
import matplotlib.pyplot as plt  
plt.xticks( np.arange(10) )  
locs, labels = plt.xticks()  
print(locs, labels)  
plt.yticks( np.arange(5) )  
locs, labels = plt.yticks()  
print(locs, labels)
```

```
plt.show()
```



Ví dụ tiếp theo chúng ta sẽ gán đồng thời vị trí và ticks của nhãn.

```
import matplotlib.pyplot as plt  
plt.xticks( np.arange(4),  
            ('Berlin', 'London', 'Hamburg', 'Toronto') )  
plt.show()
```



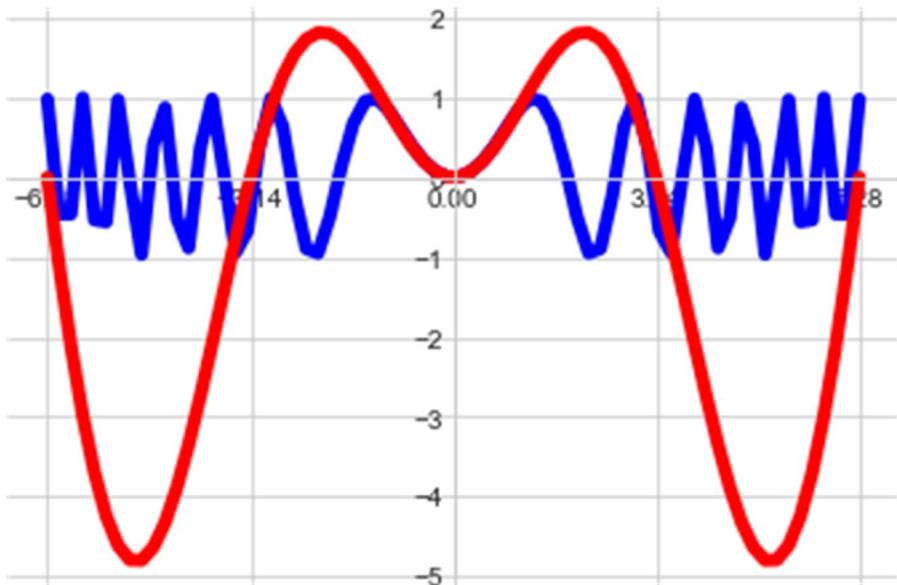
Hãy trở lại ví dụ trước của chúng ta với các hàm lượng giác. Hầu hết mọi người có thể xem xét các yếu tố của Pi để phù hợp hơn cho trục X so với các nhãn nguyên trước đó:

```

import numpy as np
import matplotlib.pyplot as plt
X = np.linspace(-2 * np.pi, 2 * np.pi, 70, endpoint=True)
X = np.linspace(-2 * np.pi, 2 * np.pi, 70, endpoint=True)
F1 = np.sin(X**2)
F2 = X * np.sin(X)
# get the current axes, creating them if necessary:
ax = plt.gca()
# making the top and right spine invisible:
ax.spines['top'].set_color('none')
ax.spines['right'].set_color('none')
# moving bottom spine up to y=0 position:
ax.xaxis.set_ticks_position('bottom')
ax.spines["bottom"].set_position(("data",0))

```

```
# moving left spine to the right to position x == 0:
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position((0,0))
plt.xticks( np.arange(-np.pi * 2,np.pi*2+1,np.pi))
plt.plot(X, F1,'-b',linewidth=5)
plt.plot(X, F2,'-r',linewidth=5)
plt.show()
```



8.8.3. Tùy chỉnh các nhãn của các ticks

Thực hiện đổi tên các xticks thay bằng các ticks tùy chỉnh. Chúng ta sẽ sử dụng phương thức xticks() một lần nữa cho mục đích này như chúng ta đã làm trong các ví dụ trước. Nhưng lần này chúng ta sẽ gọi xticks() với hai tham số: thứ nhất là cùng một danh sách chúng ta đã sử dụng trước đây, nghĩa là các vị trí trên trục x, nơi mà chúng ta muốn đặt các nhãn. Tham số thứ hai là danh sách có cùng kích thước với các dấu LaTeX tương ứng, tức là văn bản mà chúng ta muốn xem thay vì các giá trị. Ký hiệu LaTeX phải là một chuỗi thô trong hầu hết các trường hợp để ngăn chặn cơ chế thoát của Python, bởi vì ký hiệu LaTeX sử dụng rất nhiều và dựa vào dấu gạch chéo ngược.

Xét ví dụ:

```
import numpy as np
import matplotlib.pyplot as plt

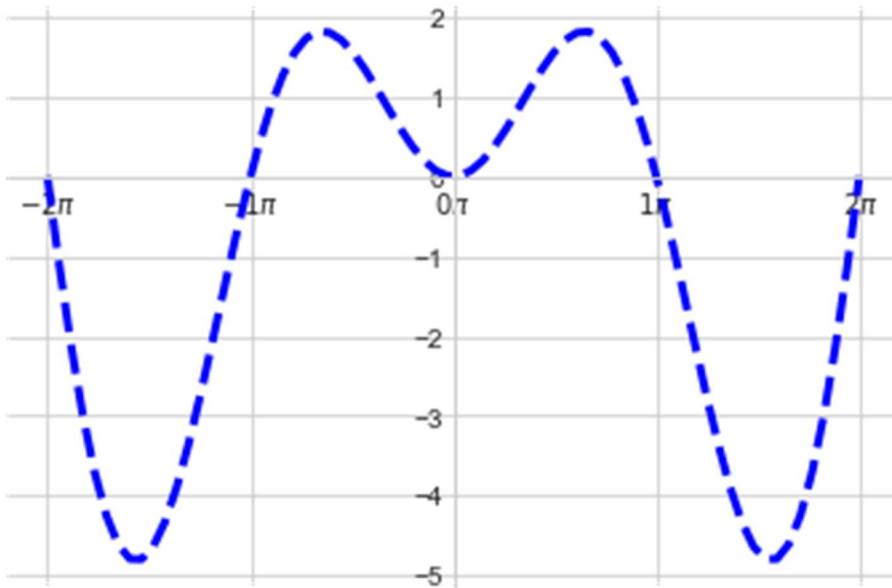
X = np.linspace(-2 * np.pi, 2 * np.pi, 70, endpoint=True)
F1 = X * np.sin(X)

ax = plt.gca()
ax.spines['top'].set_color('none')
ax.spines['right'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position((0,0))
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position((0,0))

xticks = np.arange(-np.pi * 2,np.pi*2+1,np.pi)
replace_x_ticks = [r"$\pi %d $" % (i/np.pi) for i in xticks]
replace_x_ticks

[ '-2\pi', '-1\pi', '0\pi', '1\pi', '2\pi']

# labelling the X ticks:
plt.xticks(xticks,replace_x_ticks)
plt.plot(X, F1,'--b', linewidth=3)
plt.show()
```



8.8.4. Điều chỉnh ticklabels

Chúng ta muốn tăng khả năng dễ đọc của các ticklabels, như tăng kích thước phông chữ, nhưng chúng trên một nền bán trong suốt ‘transparant’.

Tiếp ví dụ trước đó:

```
import numpy as np
import matplotlib.pyplot as plt
X = np.linspace(-2 * np.pi, 2 * np.pi, 70, endpoint=True)
F1 = X * np.sin(X)
ax = plt.gca()
ax.spines['top'].set_color('none')
ax.spines['right'].set_color('none')
ax.xaxis.set_ticks_position('bottom')
ax.spines['bottom'].set_position(('data',0))
ax.yaxis.set_ticks_position('left')
ax.spines['left'].set_position(('data',0))
xticks = np.arange(-np.pi * 2,np.pi*2+1,np.pi)
replace_x_ticks = [r"\$%d\pi\$" % (i/np.pi) for i in xticks]
```

```

replace_x_ticks
['-2\pi', '-1\pi', '0\pi', '1\pi', '2\pi']

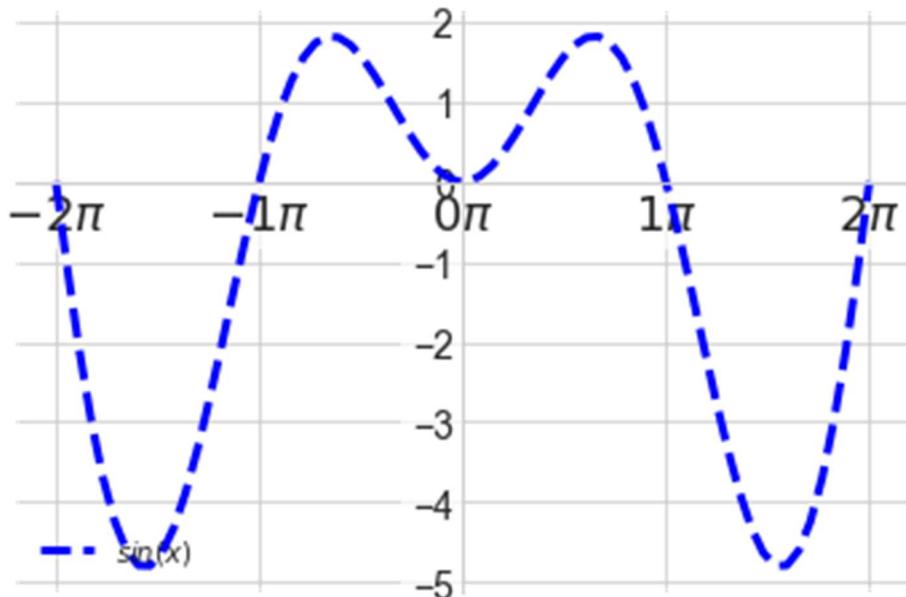
plt.xticks(xticks, replace_x_ticks)

for xtick in ax.get_xticklabels():
    xtick.set_fontsize(18)
    xtick.set_bbox(dict(facecolor='white', edgecolor='None', alpha=0.7))

for ytick in ax.get_yticklabels():
    ytick.set_fontsize(14)
    ytick.set_bbox(dict(facecolor='white', edgecolor='None', alpha=0.7))

plt.plot(X, F1, '--b', linewidth=3, label="$\sin(x)$")
plt.legend(loc='lower left')
plt.show()

```



8.9. Chú thích (Annotation)

Phần này sẽ hướng dẫn cách đánh dấu điểm trên graph qua ví dụ sau: giá trị của $3 * \sin(3 * \pi / 4) = 3/2$ thay vì một số nhầm chán 2.1213203435596428.

```

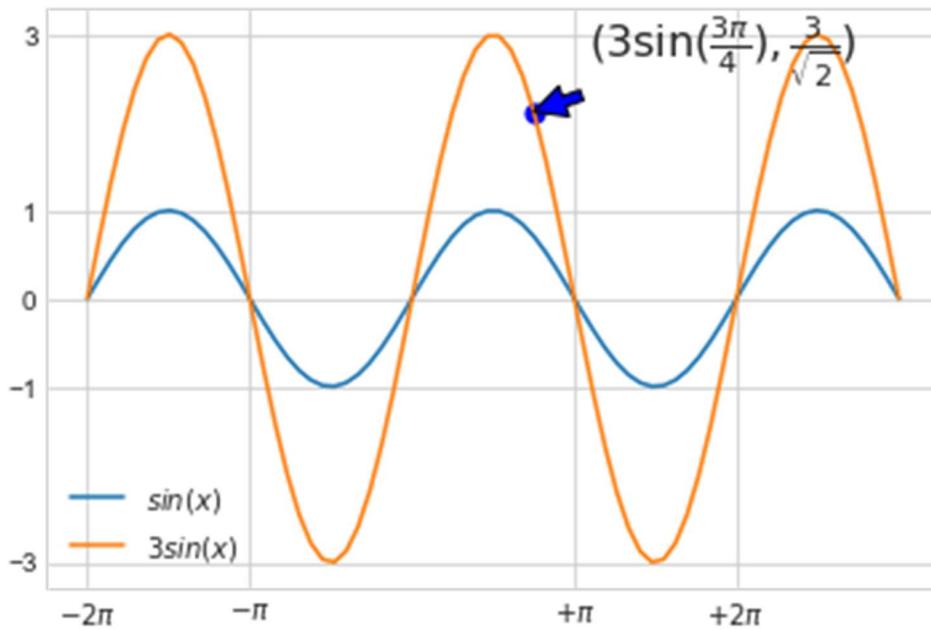
import numpy as np
import matplotlib.pyplot as plt

X = np.linspace(-2 * np.pi, 3 * np.pi, 70, endpoint=True)
F1 = np.sin(X)
F2 = 3 * np.sin(X)

ax = plt.gca()
plt.xticks([-6.28, -3.14, 3.14, 6.28],
           [r'$-2\pi$', r'$-\pi$', r'$+\pi$', r'$+2\pi$'])
plt.yticks([-3, -1, 0, +1, 3])

x = 3 * np.pi / 4
plt.scatter([x],[3 * np.sin(x)], 50, color ='blue')
plt.annotate(r'$3\sin(\frac{3\pi}{4})$',
             xy=(x, 3 * np.sin(x)),
             xycoords='data',
             xytext=(+20, +20),
             textcoords='offset points',
             fontsize=16,
             arrowprops=dict(facecolor='blue'))
plt.plot(X, F1, label="$\sin(x)$")
plt.plot(X, F2, label="$3 \sin(x)$")
plt.legend(loc='lower left')
plt.show()

```



Chúng ta phải cung cấp một số thông tin cho các tham số của chú thích, chúng ta đã sử dụng trong ví dụ trước.

Ý nghĩa của thông số

Xy: Tọa độ của mũi tên.

xytext: Tọa độ của vị trí văn bản

Vị trí xy và xytext trong ví dụ của chúng tôi nằm trong tọa độ dữ liệu. Có những hệ tọa độ khác mà chúng ta có thể lựa chọn. Hệ tọa độ của xy và xytext có thể được xác định qua chuỗi được gán cho xycoords và textcoords. Giá trị mặc định là 'data':

Chuỗi giá trị cho hệ thống tọa độ.

figure points: các điểm từ góc dưới bên trái của hình

figure pixels: các pixels từ góc dưới bên trái của hình

figure fraction: Chữ số 0,0 là thấp hơn bên trái của hình và 1,1 là trên bên phải

axes points: các điểm trục từ góc dưới bên trái của các trục

axes pixels: các pixels từ từ góc dưới bên trái của các trục

axes fraction: 0,0 là phần dưới bên trái của trục và 1,1 phía trên bên phải

data: sử dụng hệ trục tọa độ dữ liệu

Ngoài ra, chúng ta cũng có thể chỉ định các thuộc tính của mũi tên (arrow). Để làm như vậy, chúng ta phải cung cấp một dictionary các thuộc tính của tham số của mũi tên (arrowprops):

width: Chiều rộng của mũi tên.

frac: Phần của chiều dài mũi tên chiếm bởi phần đầu

headwidth: Chiều rộng của đáy của mũi tên

shrink: Di chuyển đầu và căn cứ vào phần trống đi từ điểm chú thích và text.

**kwargs Bất kỳ key của matplotlib.patches.Polygon, ví dụ, facecolor

8.10. Legends

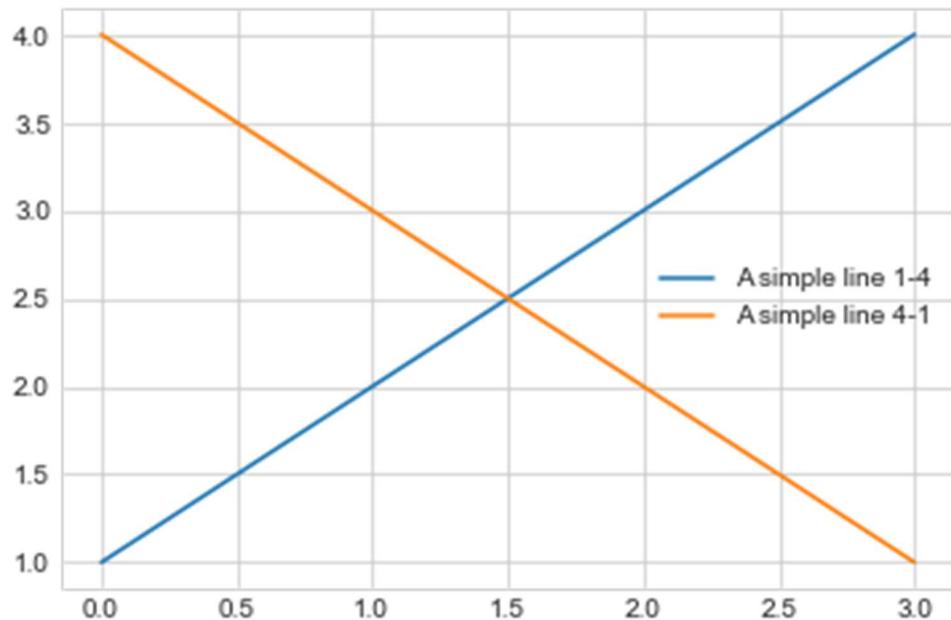
Nếu nhìn vào các line graphs của các ví dụ ở các bài trước, chúng ta nhận ra rằng, chúng ta phải xem xét mã của chúng để hiểu chức năng được mô tả như thế nào. Thông tin này nên được nhúng sẵn trong graph để thuận tiện. Legends được sử dụng cho mục đích này. Từ này bắt nguồn từ tiếng Latinh và mean "to be read". Vì vậy, nó "has to be read" để hiểu được đồ thị. Trước khi Legends được sử dụng trong đồ thị toán học, chúng đã được sử dụng trong bản đồ. Legends - như được tìm thấy trong bản đồ - mô tả hình ảnh ngôn ngữ hoặc biểu tượng của bản đồ. Các Legends được sử dụng trong biểu đồ đường để giải thích chức năng hoặc các giá trị nằm dưới các dòng khác nhau của đồ thị. Chúng ta sẽ chứng minh trong ví dụ đơn giản sau đây làm thế nào chúng ta có thể đặt một Legends trên đồ thị. Một Legends chứa một hoặc nhiều mục. Mỗi mục bao gồm một khóa và một nhãn. Hàm được định nghĩa trong pyplot như sau:

```
legend(*args, **kwargs)
```

Tất cả những gì chúng ta phải làm để tạo ra một Legend cho các line, đã tồn tại trên các trục, là chỉ cần gọi hàm "legend" với một danh sách các chuỗi, mỗi chuỗi cho mỗi Legend.

Ví dụ:

```
import numpy as np
import matplotlib.pyplot as plt
ax = plt.gca()
ax.plot([1, 2, 3, 4])
ax.plot([4,3,2,1])
ax.legend(['A simple line 1-4', 'A simple line 4-1'])
plt.show()
```

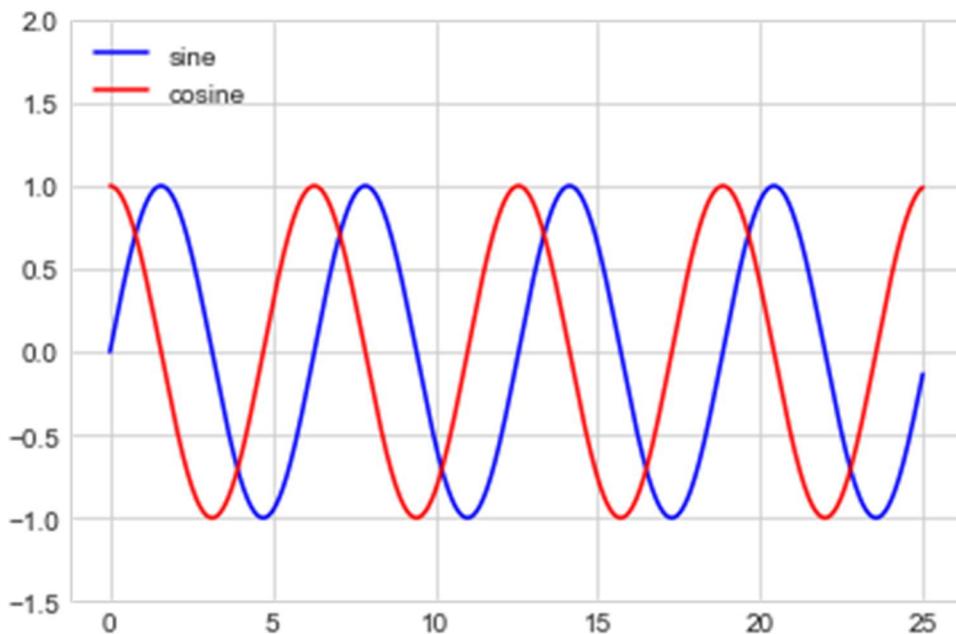


Nếu chúng ta thêm một nhãn vào hàm `plot()`, giá trị sẽ được sử dụng như nhãn trong lệnh `legend()`. Đối số duy nhất hàm `legend()` cần là đối số "loc" để chỉ rõ legend sẽ được đặt ở đâu.

Cùng xem ví dụ sau:

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 25, 1000)
y1 = np.sin(x)
y2 = np.cos(x)
```

```
plt.plot(x, y1, '-b', label='sine')
plt.plot(x, y2, '-r', label='cosine')
plt.legend(loc='upper left')
plt.ylim(-1.5, 2.0)
plt.show()
```



Mã vị trí là:

'best' : 0, (only implemented for axes legends)

'upper right' : 1,

'upper left' : 2,

'lower left' : 3,

'lower right' : 4,

'right' : 5,

'center left' : 6,

'center right' : 7,

'lower center' : 8,

'upper center' : 9,

'center' : 10,

Các bạn nên thử lại ví dụ phía trên bằng cách thay đổi “upper left” trong đoạn mã plt.legend(loc='upper left') bằng các mã mà tôi đã liệt kê phía bên trên để thấy được sự thay đổi tương ứng của legend trên đồ thị.

8.10.1. Legend for Size of Points

Đôi khi các giá trị mặc định của legend không đáp ứng đủ cho việc hiển thị. Ví dụ: có lẽ bạn đang sử dụng kích thước điểm để đánh dấu các tính năng nhất định của dữ liệu và muốn tạo một legend phản ánh điều này. Dưới đây là một ví dụ mà chúng ta sẽ sử dụng kích thước của các điểm để chỉ ra các cụm dân số của các thành phố ở California. Chúng tôi muốn một legend chỉ định quy mô của các kích thước của các điểm và chúng tôi sẽ thực hiện việc này bằng cách vẽ một số dữ liệu được dán nhãn không có mục:

Bước 1: Trích xuất dữ liệu

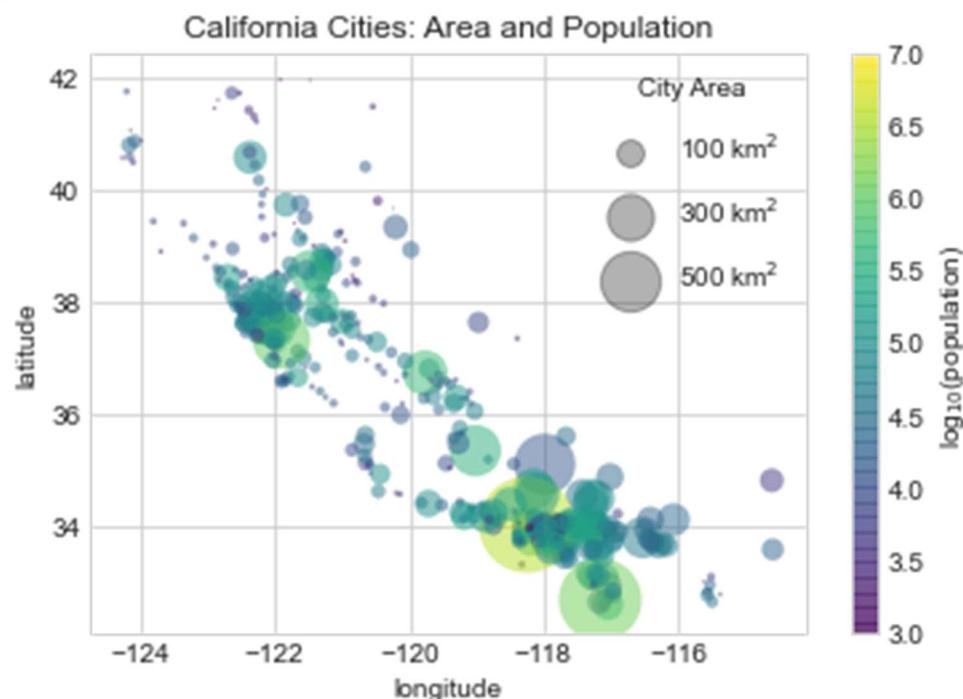
```
import pandas as pd
import io
import requests
url="https://raw.githubusercontent.com/zedeklabs/data-science-ml-
foundation/master/data/california_cities.csv"
s=requests.get(url).content
cities = pd.read_csv(io.StringIO(s.decode('utf-8')))
lat, lon = cities['latd'], cities['longd']
population, area = cities['population_total'], cities['area_total_km2']
```

Bước 2: vẽ các điểm rời rạc (Scatter, sử dụng size and color và không cung cấp thông tin label).

```
plt.scatter(lon, lat, label=None, c=np.log10(population), cmap='viridis', s=area,
            linewidth=0, alpha=0.5)
plt.axis(aspect='equal')
plt.xlabel('longitude')
plt.ylabel('latitude')
plt.colorbar(label='log$_{10}$(population)')
plt.clim(3, 7)
```

Bước 3: Tại đây, chúng ta tạo ra một legend, vẽ các danh sách trống với kích thước và nhãn mong muốn.

```
for area in [100, 300, 500]:
    plt.scatter([], [], c='k', alpha=0.3, s=area, label=str(area) + ' km$^2$')
plt.legend(scatterpoints=1, frameon=False, labelspacing=1, title='City Area')
plt.title('California Cities: Area and Population');
plt.show()
```



Legend sẽ luôn đề cập đến một số đối tượng trên plot, vì vậy nếu chúng ta muốn hiển thị một hình dạng cụ thể, chúng ta cần vẽ ra nó. Trong trường hợp này, các đối tượng chúng ta muốn (các vòng màu xám) không nằm trong plot, vì vậy chúng tôi giả mạo chúng bằng cách vẽ ra các danh sách trống. Chú ý rằng các legend chỉ liệt kê các phần tử trong plot mà có một nhãn được chỉ định. Kỹ thuật giả mạo này giúp ta linh hoạt biểu diễn, chủ thích cho đồ thị.

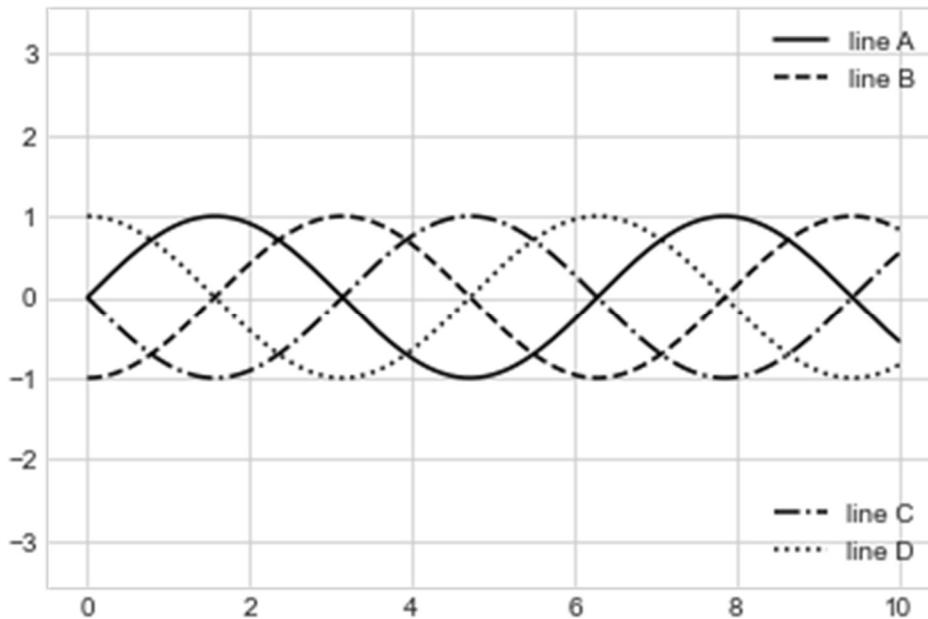
8.10.2. Multiple Legends

Đôi khi bạn muốn thêm nhiều legend vào cùng một trực trong thiết kế một plot. Thật không may, Matplotlib không làm điều này dễ dàng: thông qua giao diện truyền thống tiêu chuẩn, nó chỉ có thể tạo ra một legend duy nhất cho toàn bộ plot. Nếu bạn cố gắng tạo ra một legend thứ hai bằng cách sử dụng plt.legend () hoặc ax.legend (), nó sẽ chỉ đơn giản ghi đè lên cái đầu tiên. Chúng ta có thể làm việc này bằng cách tạo ra legend artist mới từ đầu, và sau đó sử dụng phương pháp ax.add_artist () cấp dưới để tự thêm legend thứ hai vào plot:

```
fig, ax = plt.subplots()
lines = []
styles = [':', '--', '-.', ':']
x = np.linspace(0, 10, 1000)
for i in range(4):
    lines += ax.plot(x, np.sin(x - i * np.pi / 2),
                      styles[i], color='black')
ax.axis('equal')
# specify the lines and labels of the first legend
ax.legend(lines[:2], ['line A', 'line B'],
          loc='upper right', frameon=False)

# Create the second legend and add the artist manually.
from matplotlib.legend import Legend
leg = Legend(ax, lines[2:], ['line C', 'line D'],
             loc='lower right', frameon=False)
```

```
ax.add_artist(leg);
```



8.11. Màu sắc

Sự lựa chọn màu sắc trong quá trình data visualization không chỉ đơn thuần là sự lựa chọn thẩm mỹ, mà còn là một công cụ quan trọng để chuyển tải thông tin định lượng. Lựa chọn đúng màu truyền tải dữ liệu chính xác, ngược với nhiều màu thường có thể bóp méo các mối quan hệ giữa các giá trị dữ liệu. Sử dụng màu sắc hợp lý cũng cho phép nhiều bộ dữ liệu được xếp lớp lại với nhau, giúp tạo ra graphs có thể chỉ ra “nguyên nhân và kết quả”. Có nhiều lý thuyết về sử dụng màu trong việc hình dung, và hiển thị các phương pháp để lựa chọn các bảng màu hiệu quả và các bạn có thể tìm đâu đó trên mạng để có được cách chọn màu hợp lý cho dữ liệu của bạn.

Trong bài viết này tôi muốn giới thiệu các loại màu và bảng màu mà matplotlib hỗ trợ. Điều này rất quan trọng giúp bạn chọn lựa chính xác màu trong matplotlib.

Matplotlib nhận dạng các định dạng sau để chỉ định màu:

Bộ ký tự RGB hoặc RGBA có các giá trị thực nằm trong đoạn [0, 1] (ví dụ, (0.1, 0.2, 0.5) hoặc (0.1, 0.2, 0.5, 0.3));

Một chuỗi RGB hoặc RGBA hex (ví dụ: '# 0F0F0F' hoặc '# 0F0F0F0F').

Biểu diễn chuỗi giá trị thực trong [0, 1] bao gồm cả cấp độ màu xám (ví dụ: '0.5').

Một trong số {b ',' g ',' r ',' c ',' m ',' y ',' k ',' w '}.

Tên màu X11 / CSS4. (sẽ được nhắc lại)

Tên từ <https://xkcd.com/color/rgb/> thêm tiền tố 'xkcd:' (ví dụ: 'xkcd:purple') (sẽ được nhắc lại)

Là một trong những tab {'tab:blue', 'tab:orange', 'tab:green', 'tab:red', 'tab:purple', 'tab:brown', 'tab:pink', 'tab:gray', 'tab:olive', 'tab:cyan'} đến từ phân loại của Tableau Colors.

Thông số màu "CN", tức là 'C' theo sau bởi một chữ số đơn, là một chỉ mục vào chu trình thuộc tính mặc định (matplotlib.rcParams['axes.prop_cycle']); lập chỉ mục xảy ra trong thời gian tạo tác của artist và mặc định là màu đen nếu chu kỳ không bao gồm màu.

8.11.1. Thông số màu "CN"

Color có thể được xác định bởi một chuỗi khớp với regex C[0-9]. Điều này có thể được thông qua bất kỳ nơi mà một màu sắc hiện đang được chấp nhận và có thể được sử dụng như một 'kí tự màu' như 'k'.

Các chữ là chỉ số của matplotlib.rcParams ['axes.prop_cycle']. Nếu chu trình thuộc tính không bao gồm 'color' thì màu đen sẽ được trả về. Màu sắc được đánh giá khi artist được tạo ra. Ví dụ, dù là hai cách gán màu khác nhau nhưng đều cho cùng một kết quả. 'C1' sẽ tương ứng với giá trị u'#55A868' trong mpl.rcParams['axes.prop_cycle'].

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
```

```

th = np.linspace(0, 2*np.pi, 128)

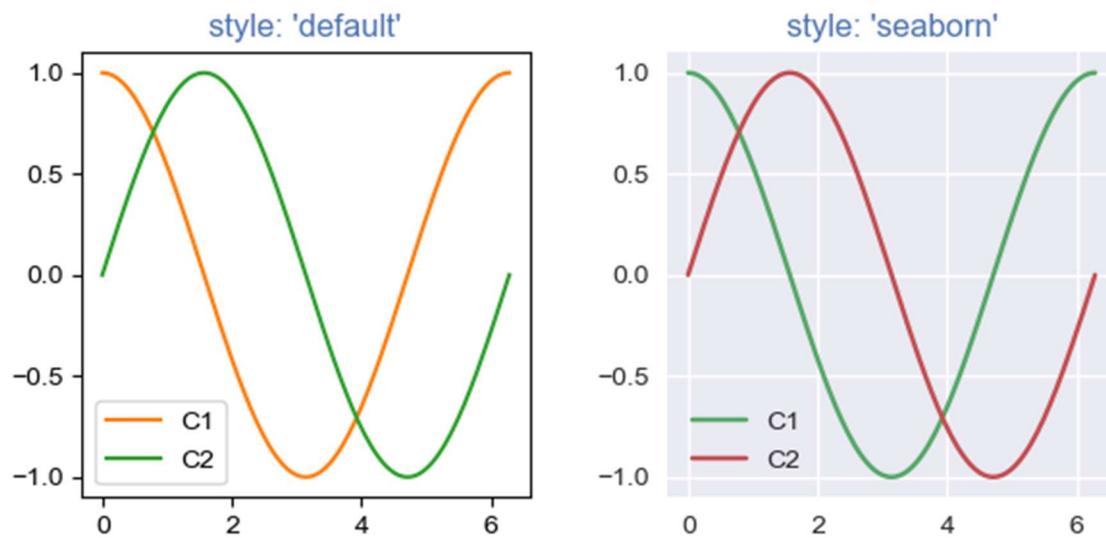
def demo(sty):
    mpl.style.use(sty)
    fig, ax = plt.subplots(figsize=(3, 3))

    ax.set_title('style: {!r}'.format(sty), color='C0')

    ax.plot(th, np.cos(th), 'C1', label='C1')
    ax.plot(th, np.sin(th), 'C2', label='C2')
    ax.legend()

demo('default')
demo('seaborn')

```



6.11.2. Thông số màu X11 / CSS4 và xkcd

Có 95 (trong số 148 màu trong danh sách màu css) xung đột giữa tên CSS4 / X11 và tên xkcd. CSS4 / X11 cho rằng là tên màu chuẩn của web, matplotlib nên theo họ. Do đó, các tên màu xkcd được đặt trước bằng 'xkcd:', ví dụ như bản đồ 'xanh' tới '# 0000FF' trong đó ở dạng 'xkcd: blue' bản đồ là '# 0343DF'. Vì bảng khá là rộng, nên tôi khuyến khích các bạn tự chạy đoạn mã sau để thấy được chi tiết xung đột này.

```
import matplotlib._color_data as mcd
import matplotlib.patches as mpatch

overlap = {name for name in mcd.CSS4_COLORS
           if "xkcd:" + name in mcd.XKCD_COLORS}

fig = plt.figure(figsize=[4.8, 16])
ax = fig.add_axes([0, 0, 1, 1])

for j, n in enumerate(sorted(overlap, reverse=True)):
    weight = None
    cn = mcd.CSS4_COLORS[n]
    xkcd = mcd.XKCD_COLORS["xkcd:" + n].upper()
    if cn == xkcd:
        weight = 'bold'

    r1 = mpatch.Rectangle((0, j), 1, 1, color=cn)
    r2 = mpatch.Rectangle((1, j), 1, 1, color=xkcd)
    txt = ax.text(2, j+.5, ' ' + n, va='center', fontsize=10,
                  weight=weight)
    ax.add_patch(r1)
```

```
ax.add_patch(r2)
ax.axhline(j, color='k')

ax.text(.5, j + 1.5, 'X11', ha='center', va='center')
ax.text(1.5, j + 1.5, 'xkcd', ha='center', va='center')
ax.set_xlim(0, 3)
ax.set_ylim(0, j + 2)
ax.axis('off')
```



Màu sắc là một thành phần quan trọng khi vẽ hình, nó không chỉ giúp cho các thành phần của đồ thị rõ ràng mà còn giúp cho người đọc có thể tập trung vào một vùng nào đó trên hình từ đó có thể dễ nhận ra insight trong dữ liệu. Bài học này đã giới thiệu khá đầu đủ về các loại màu và bảng màu mà matplotlib hỗ trợ dù đó có thể hướng dẫn người đọc chọn được màu thích hợp với hình vẽ của mình.

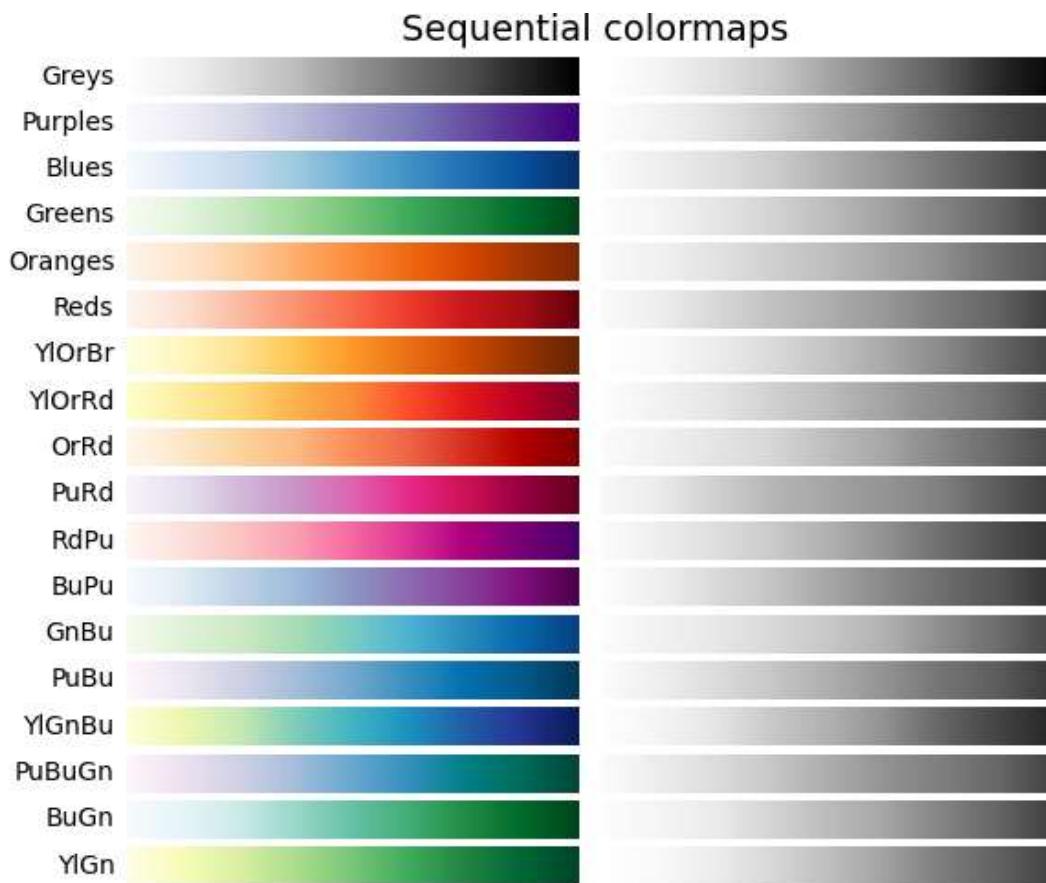
8.12. Colormaps

Đối với nhiều hình vẽ, trong đó các mức tương ứng trong dữ liệu được hiểu là các mức tương đương trong không gian màu, do nhận thức dựa trên colormap là sự lựa chọn tốt nhất. Ví dụ, các nhà nghiên cứu đã phát hiện ra rằng bộ não con người nhận thức sự thay đổi tham số về độ sáng (lightness parameter) - cái phản ánh những thay đổi trong dữ liệu tốt hơn nhiều so với sự thay đổi về màu sắc (hue). Do đó, các bản đồ màu có độ sáng đơn điệu tăng lên thông qua bảng màu sẽ thể hiện tốt hơn cho người xem.

Colormaps thường được chia thành nhiều loại dựa trên chức năng của chúng:

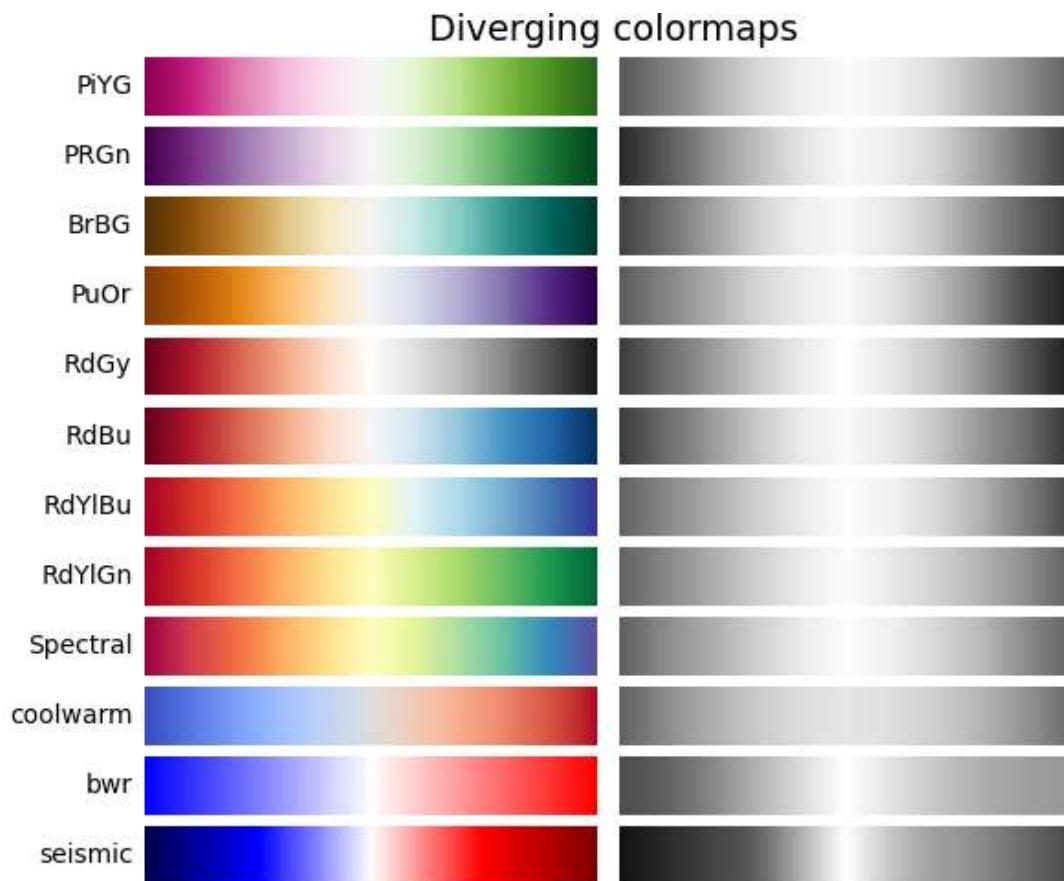
8.12.1. Sequential

Thay đổi độ sáng và thường bao hòa màu sắc từng bước, thường sử dụng một màu duy nhất; nên được sử dụng để đại diện cho thông tin có trật tự (e.g., binary or viridis).



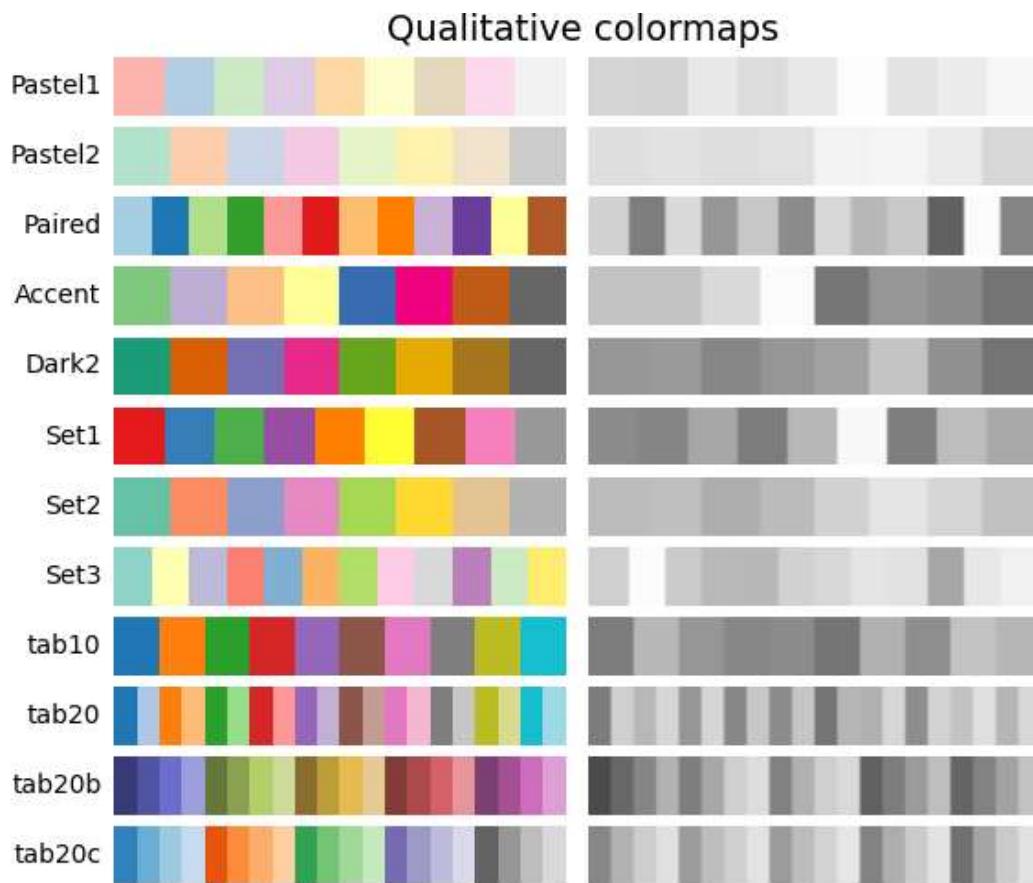
8.12.2. Diverging

Thay đổi độ sáng và có thể bão hòa hai màu khác nhau gặp nhau ở giữa với màu không bão hòa; nên được sử dụng khi thông tin được vẽ có giá trị trung bình quan trọng, chẳng hạn như địa hình (topography) hoặc khi dữ liệu dao động quanh điểm trung bình (e.g., RdBu or PuOr).



8.12.3. Qualitative

Thường là màu sắc hỗn hợp; nên được sử dụng để đại diện cho thông tin không có trật tự hoặc các mối quan hệ (e.g., rainbow or jet).



Đoạn mã sau sẽ giúp mọi người có thể nhìn thấy toàn bộ bảng màu thuộc các loại khác nhau.

```

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import LinearSegmentedColormap
cmaps = [('Perceptually Uniform Sequential', [
    'viridis', 'plasma', 'inferno', 'magma']),
    ('Sequential', [
        'Greys', 'Purples', 'Blues', 'Greens', 'Oranges', 'Reds',

```

```
'YlOrBr', 'YlOrRd', 'OrRd', 'PuRd', 'RdPu', 'BuPu',  
  
'GnBu', 'PuBu', 'YlGnBu', 'PuBuGn', 'BuGn', 'YlGn']),  
  
('Sequential (2)', [  
  
'binary', 'gist_yarg', 'gist_gray', 'gray', 'bone', 'pink',  
  
'spring', 'summer', 'autumn', 'winter', 'cool', 'Wistia',  
  
'hot', 'afmhot', 'gist_heat', 'copper']),  
  
('Diverging', [  
  
'PiYG', 'PRGn', 'BrBG', 'PuOr', 'RdGy', 'RdBu',  
  
'RdYlBu', 'RdYlGn', 'Spectral', 'coolwarm', 'bwr', 'seismic']),  
  
('Qualitative', [  
  
'Pastel1', 'Pastel2', 'Paired', 'Accent',  
  
'Dark2', 'Set1', 'Set2', 'Set3'  
  
]),  
  
('Miscellaneous', [
```

```

'flag', 'prism', 'ocean', 'gist_earth', 'terrain', 'gist_stern',
'gnuplot', 'gnuplot2', 'CMRmap', 'cubehelix', 'brg', 'hsv',
'gist_rainbow', 'rainbow', 'jet', 'nipy_spectral', 'gist_ncar'])]

gradient = np.linspace(0, 1, 256)
gradient = np.vstack((gradient, gradient))

def grayscale_cmap(cmap):
    """Return a grayscale version of the given colormap"""

    cmap = plt.cm.get_cmap(cmap)
    colors = cmap(np.arange(cmap.N))
    RGB_weight = [0.299, 0.587, 0.114]
    luminance = np.sqrt(np.dot(colors[:, :3] ** 2, RGB_weight))
    colors[:, :3] = luminance[:, np.newaxis]
    return LinearSegmentedColormap.from_list(cmap.name + "_gray", colors,
cmap.N)

def plot_color_gradients(cmap_category, cmap_list, nrows):
    fig, axes = plt.subplots(ncols=2, nrows=nrows, figsize=(15, 15))
    fig.subplots_adjust(top=0.95, bottom=0.01, left=0.12, right=0.99)
    axes[0][0].set_title(cmap_category + ' colormaps', fontsize=15)
    axes[0][1].set_title(cmap_category + ' colormaps with grayscale', fontsize=15)
    for ax, name in zip(axes, cmap_list):
        ax[0].imshow(gradient, aspect='auto', cmap=plt.get_cmap(name))
        ax[1].imshow(gradient, aspect='auto',
cmap=grayscale_cmap(plt.get_cmap(name)))
        pos = list(ax[0].get_position().bounds)
        x_text = pos[0] - 0.02
        y_text = pos[1] + pos[3]/2.

```

```

fig.text(x_text, y_text, name, va='center', ha='right', fontsize=15)

# Turn off *all* ticks & spines, not just the ones with colormaps.

for ax in axes:
    ax[0].set_axis_off()
    ax[1].set_axis_off()

for cmap_category, cmap_list in cmaps:
    nrows = len(cmap_list)
    plot_color_gradients(cmap_category, cmap_list, nrows)
plt.show()

```

Nếu các bạn đã chạy đoạn mã phía trên, các bạn sẽ hỏi tại sao tôi lại phải chuyển đổi các bảng màu sang dạng đen trắng. Chuyển đổi sang màu xám (Grayscale conversion) là rất quan trọng để chú ý đến các ánh phảm in có các ô màu. Nếu điều này không được chú ý đến trước thời gian, độc giả có thể không thể giải thích được hình vẽ vì màu xám thay đổi không thể đoán trước trong colormap.

Một số quan sát thú vị sau:

Chúng ta thấy rằng các sequential colormaps có thể chuyển đổi hợp lý sang màu xám. Một số bảng màu thuộc Sequential2 thể hiện rất tốt khi chuyển sang màu xám, mặc dù (autumn, spring, summer, winter) có sự thay đổi màu xám rất ít. Nếu một bảng màu như thế này đã được sử dụng trong một hình vẽ và sau đó hình vẽ được in với màu xám, rất nhiều thông tin từ colormap có thể ánh xạ tới cùng một giá trị màu xám.

Các diverging colormaps chủ yếu khác nhau từ màu xám đậm trên các cạnh bên ngoài để trắng ở giữa. Một số (PuOr và seismic) có màu xám sẫm màu ở một bên hơn và do đó không đối xứng. Coolwarm có rất ít phạm vi màu xám và sẽ in ra một hình vẽ đồng nhất hơn, gây mất rất nhiều chi tiết. Lưu ý rằng, các contours được gắn nhãn có thể giúp phân biệt giữa một mặt của bảng màu với màu khác vì không thể sử dụng màu sắc khi hình vẽ được in bằng màu xám.

Nhiều bảng màu thuộc Qualitative and Miscellaneous colormaps, chẳng hạn như Accent, hsv và jet, thay đổi từ tối sang sáng và trở lại màu xám đậm trong suốt bản đồ màu. Điều này sẽ làm cho người xem không thể diễn giải thông tin trong hình vẽ khi nó được in bằng màu xám.

Tham khảo thêm các loại colormap tại link:

<https://matplotlib.org/tutorials/colors/colormaps.html#sphx-glr-tutorials-colors-colormaps-py>

8.13. Stylesheets

Matplotlib vừa mạnh mẽ vừa phức tạp: có thể điều chỉnh mọi khía cạnh của hình vẽ, nhưng thường mất thời gian và phức tạp để tạo ra một hình vẽ tuyệt đẹp. Việc phát hành Matplotlib 1.5 làm cho việc điều chỉnh này dễ dàng hơn để đạt được hình có tinh thẩm mỹ nhất định bằng cách kết hợp bộ các styles trong matplotlib.

Trong bài viết này, tôi sẽ giới thiệu về cách thiết lập một style, thể hiện một số styles khác nhau trong hoạt động và cho thấy cách matplotlibs dễ dàng thay đổi các thiết lập để phù hợp với sở thích của chính bạn.

Mỗi style tạo ra một cái nhìn chung mà có thể dễ dàng áp dụng cho tất cả các loại hình vẽ khác nhau. Họ thay đổi tất cả các khía cạnh thị giác chính của hình vẽ như xticks, legends và nhãn.

Chúng ta có thể kiểm tra các kiểu mặc định thông qua lệnh sau:

```
from matplotlib import pyplot as plt
print(plt.style.available)
```

Kết quả:

```
['Solarize_Light2', '_classic_test_patch', 'bmh', 'classic', 'dark_background', 'fast',
'fivethirtyeight', 'ggplot', 'grayscale', 'seaborn', 'seaborn-bright', 'seaborn-
colorblind', 'seaborn-dark', 'seaborn-dark-palette', 'seaborn-darkgrid', 'seaborn-
deep', 'seaborn-muted', 'seaborn-notebook', 'seaborn-paper', 'seaborn-pastel',
```

```
'seaborn-poster', 'seaborn-talk', 'seaborn-ticks', 'seaborn-white', 'seaborn-whitegrid', 'tableau-colorblind10']
```

Những loại trên có thể nhóm thành nhóm như sau:

seaborn-*: Đây là một bộ styles từ Seaborn, nó là một bộ sung cho Matplotlib, cung cấp các tính năng bổ sung và cải thiện tính thẩm mỹ cho matplotlib. Mặc định seaborn-deep, seaborn-pastel và seaborn-white là những bộ styles ưu thích.

dark_background: dark_background là một chuẩn của matplotlib với màu sắc thay đổi cho độ tương phản cao.

bmh: style này xuất phát từ sách Bayesian Methods for Hackers. Tôi tìm thấy nó đặc biệt phù hợp với đồ thị khoa học bằng cách hiển thị độ chính xác của hình vẽ.

ggplot: Style này xuất phát từ hệ thống vẽ cùng tên cho ngôn ngữ R: phải mất rất nhiều bài học về việc trình bày dữ liệu, tập trung vào sự đơn giản.

fivethirtyeight: Style này mô phỏng giao diện của nhà phân tích dữ liệu nổi tiếng Nate Silver's site fivethirtyeight.com.

Khi bạn chạy đoạn mã dưới đây các bạn sẽ nhìn thấy sự khác biệt khi ta lần lượt thay đổi style của hình vẽ: 'seaborn-white', 'dark_background', 'bmh', 'ggplot', 'fivethirtyeight'.

```
import matplotlib.pyplot as plt
style = 'fivethirtyeight' # 'seaborn-white' 'dark_background', 'bmh', 'ggplot',
'fivethirtyeight'
plt.style.use(style)
plt.rcParams.update({'font.family': 'serif',
                    'font.serif': 'Ubuntu',
                    'font.monospace': 'Ubuntu Mono',
```

```
'font.size':10,  
  
'axes.labelsize':10,  
  
'axes.labelweight':'bold',  
  
'axes.titlesize':10,  
  
'xtick.labelsize':8,  
  
'ytick.labelsize':8,  
  
'legend.fontsize':10,  
  
'figure.titlesize':12})  
  
width, height = plt.figaspect(2)  
  
fig = plt.figure(figsize=(width,height), dpi=250)  
  
fig.suptitle("%s style example" % style)  
  
# Product sales plot  
  
ax1 = plt.subplot(221)  
  
ax1.bar([1, 2, 3, 4], [125, 100, 90, 110], label="Product", width=0.8,  
align='center')
```

```
plt.xticks([1, 2, 3, 4], ['Q1', 'Q2', 'Q3', 'Q4'])

plt.xlabel('Time (FY)')

plt.ylabel('Sales')

# Font style isn't accessible through rcParams

ax1.set_title("Product sales", fontstyle='italic')

# Opportunities by age plot

ax2 = plt.subplot(222)

population_ages = [10,11,13,14,14,
                   20,20,21,21,22,22,22,23,25,25,25,25,25,27,27,
                   30,30,30,31,32,32,32,33,33,33,33,34,34,34,34,34,34,34,36,37,38,39,
                   41,41,42,42,42,43,45,45,49,
                   55,57,59,
                   72,]

bins = [10,20,30,40,50,60]

ax2.hist(population_ages, bins, histtype='bar', rwidth=0.8)

plt.xlabel('Age (days)')

plt.ylabel('Closed sales')
```

```

ax2.set_title('Opportunities age', fontstyle='italic')

# Marketing channels line plot

ax3 = plt.subplot(2,2,(3,4))

y_series = [1,2,3,4,5]

x_1 = [5,9,9,12,9]

x_2 = [4,3,12,8,14]

x_3 = [1,3,4,2,5]

ax3.plot(y_series, x_1, linewidth=2, linestyle=':', marker='o', label='facebook aid')

ax3.plot(y_series, x_2, linewidth=2, linestyle='--', marker='v', label='TV')

ax3.plot(y_series, x_3, linewidth=2, linestyle='-.', marker='s', label='sales team')

plt.xlabel('Months')

plt.ylabel('Marketing leads')

plt.xticks([1,2,3,4,5], ['Oct', 'Nov', 'Dec', 'Jan', 'Feb'])

leg=plt.legend(loc='best', numpoints=1, fancybox=True)

# Axes alteration to put zero values inside the figure Axes

# Avoids axis white lines cutting through zero values - fivethirtyeight style

xmin, xmax, ymin, ymax = ax3.axis()

ax3.axis([xmin-0.1, xmax+0.1, ymin-0.1, ymax+0.4])

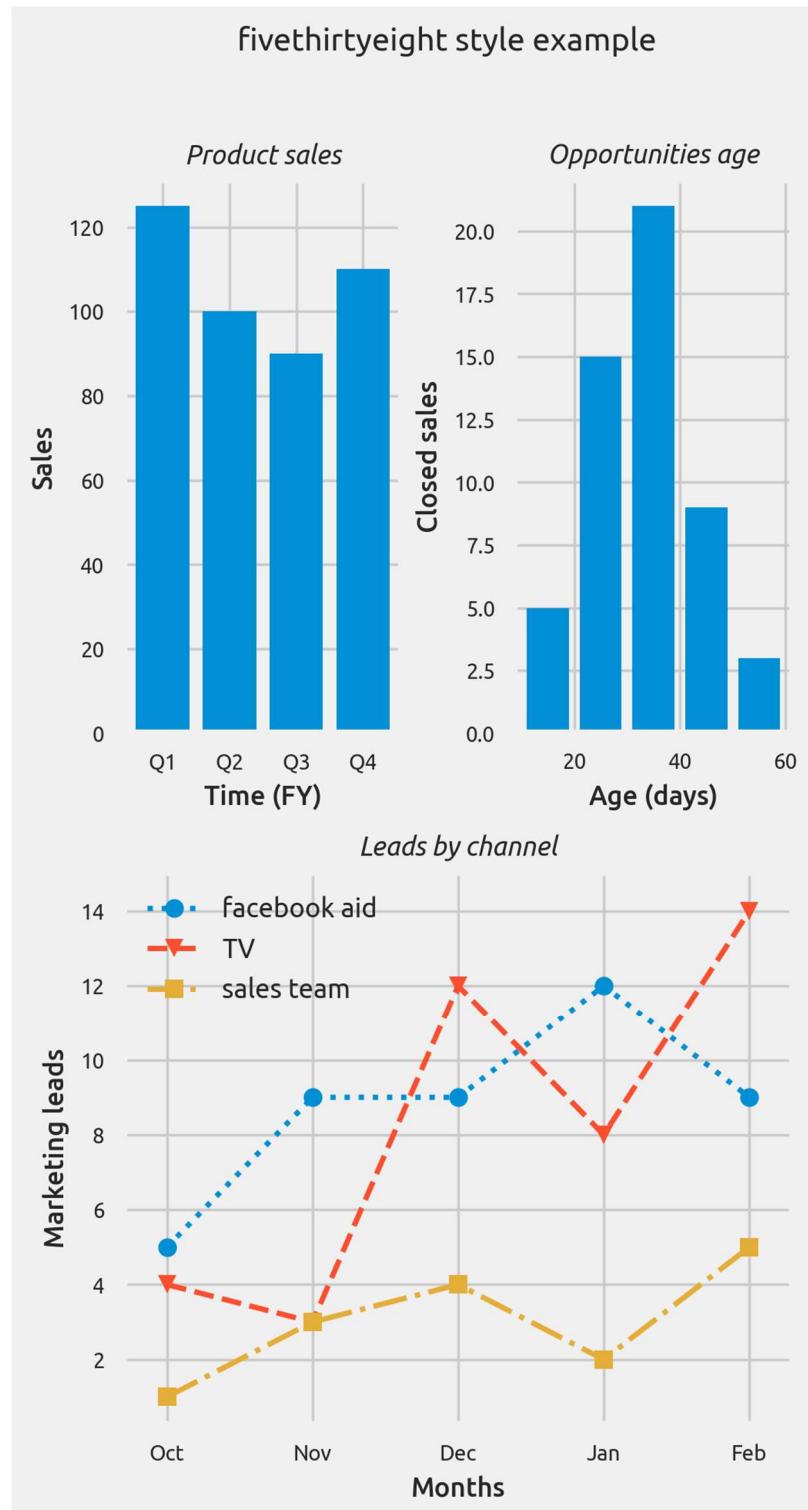
ax3.set_title('Leads by channel', fontstyle='italic')

# Space plots a bit

plt.subplots_adjust(hspace=0.25, wspace=0.40)

plt.show()

```



8.14. Contour

Một đường vạch hoặc isoline của một hàm của hai biến là một đường cong dọc theo hàm đó có một giá trị không đổi.

Nó là một mặt cắt ngang của đồ thị ba chiều của hàm $f(x, y)$ song song với mặt phẳng x, y .

Đường viền được sử dụng, ví dụ, trong địa lý và khí tượng học.

Trong bản đồ, một đường viền nối các điểm có độ cao (chiều cao) tương đương, đường đồng mức, chẳng hạn như mực nước biển trung bình.

Chúng ta cũng có thể nói một cách tổng quát hơn rằng một đường viền của một hàm có hai biến là một đường cong kết nối những điểm có cùng giá trị.

Bước 1: tạo một ‘meshgrid’ (ý nghĩa xem tại: <https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.meshgrid.html>)

```
#bước 1
import numpy as np
xlist = np.linspace(-3.0, 3.0, 3)
ylist = np.linspace(-3.0, 3.0, 4)
X, Y = np.meshgrid(xlist, ylist)
print(xlist)
print(ylist)
print(X)
print(Y)
```

Output:

```
[-3. 0. 3.]
[-3. -1. 1. 3.]
[[-3. 0. 3.]]
```

```

[-3. 0. 3.]
[-3. 0. 3.]
[-3. 0. 3.]]
[[-3. -3. -3.]
 [-1. -1. -1.]
 [ 1. 1. 1.]
 [ 3. 3. 3.]]

```

Bước 2: tính giá trị Z

```

#bước 2
Z = np.sqrt(X**2 + Y**2)
print(Z)

```

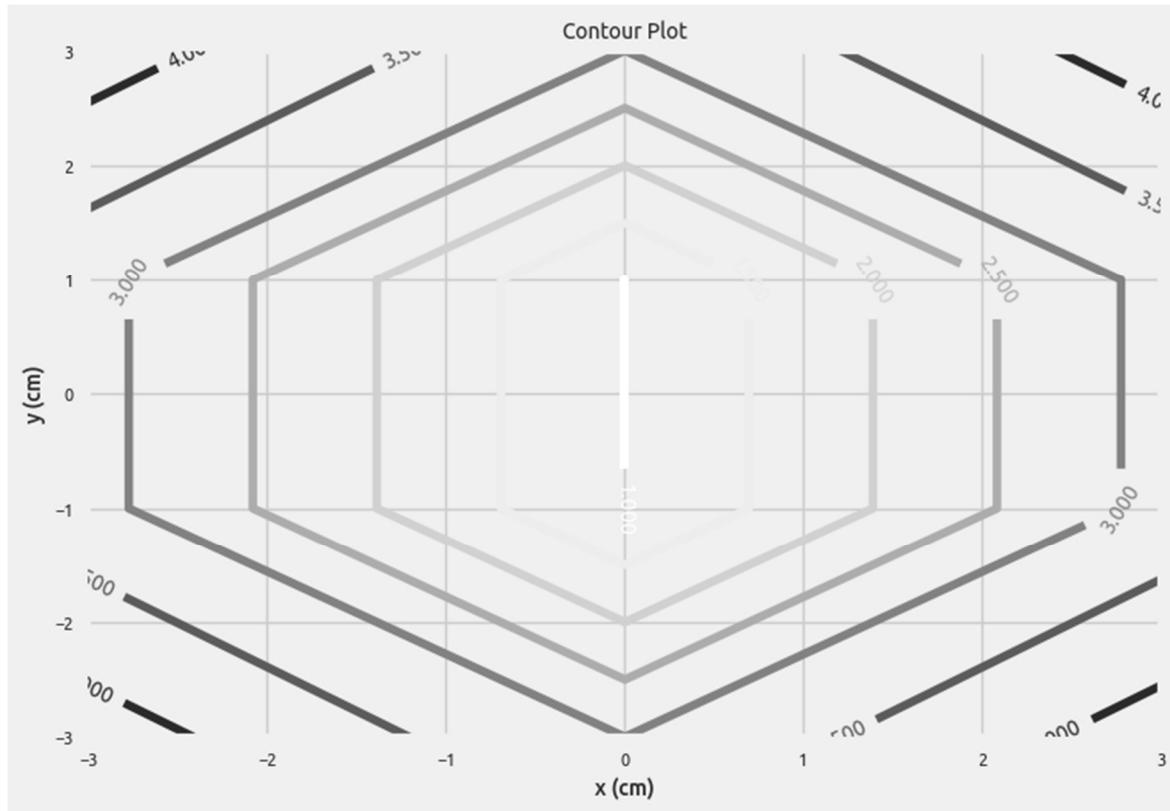
Output:

```

[[4.24264069 3.      4.24264069]
 [3.16227766 1.      3.16227766]
 [3.16227766 1.      3.16227766]
 [4.24264069 3.      4.24264069]]

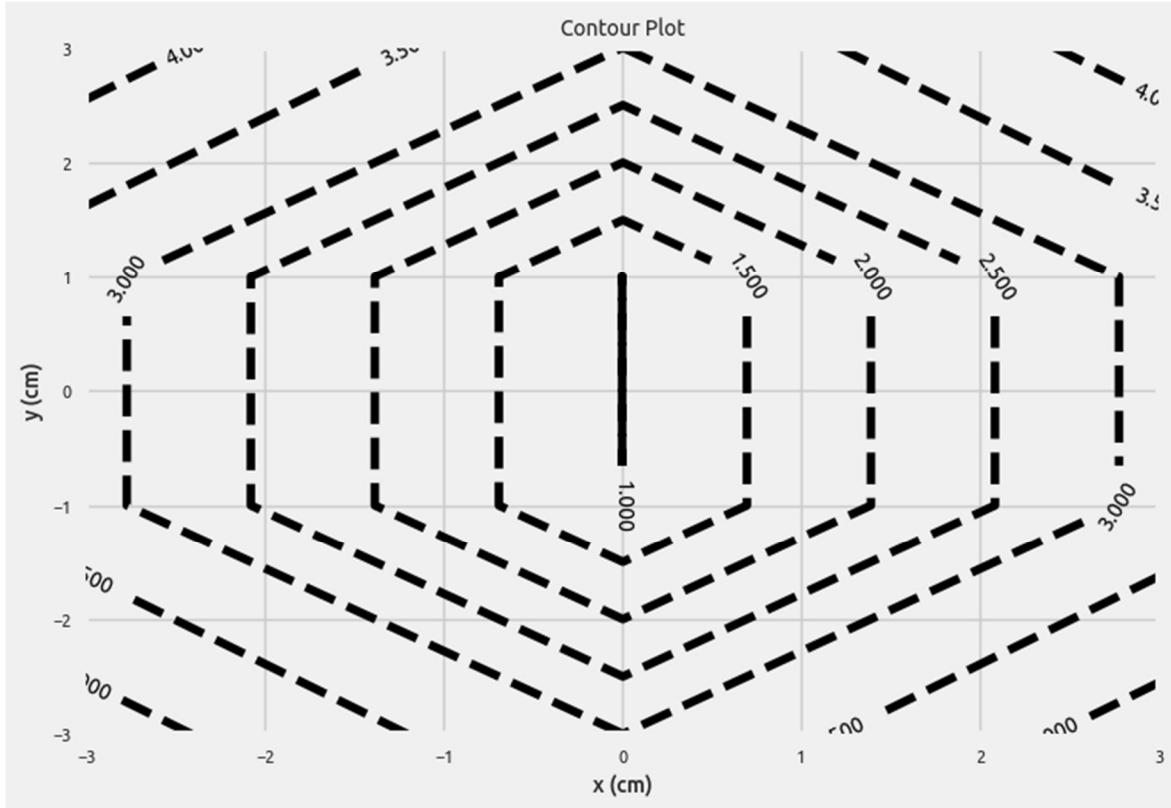
```

Bước 3: Dùng phương thức contour() của pyplot để vẽ một contour mong muốn từ X,Y,Z



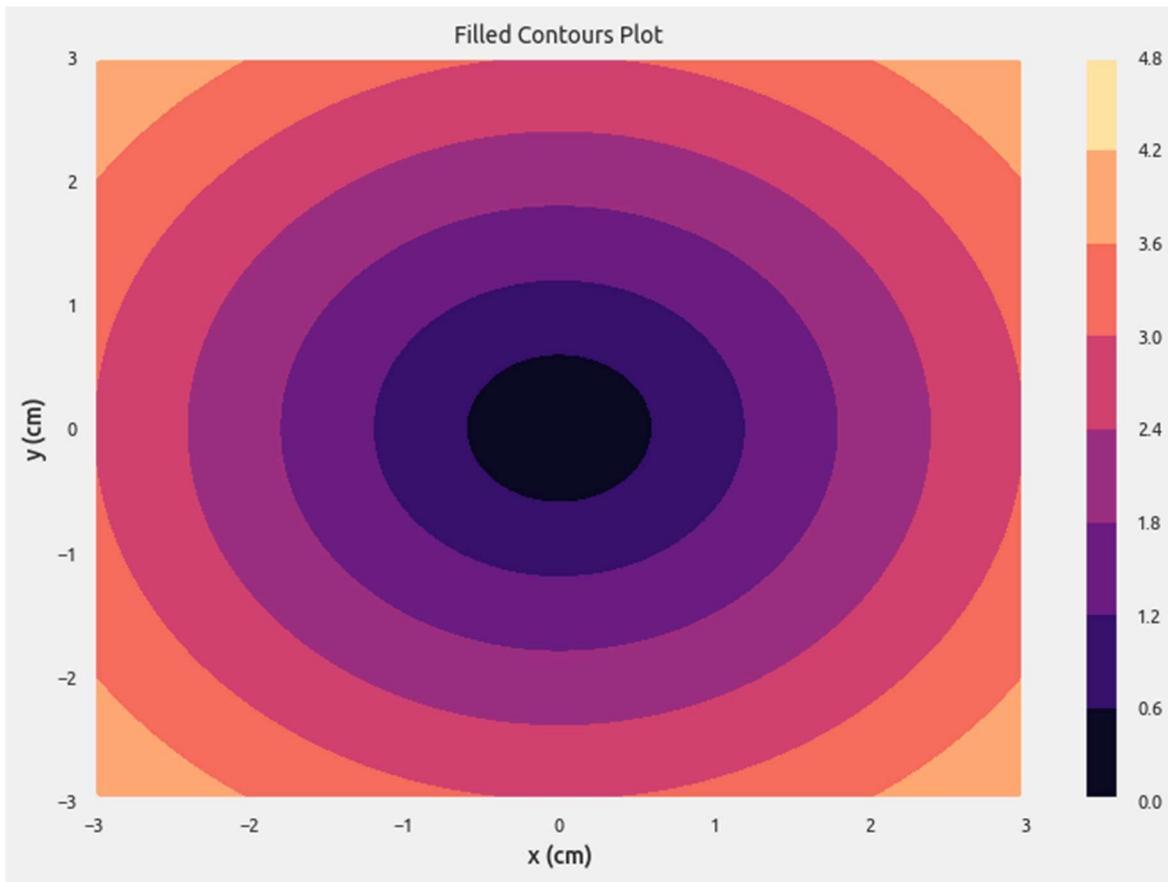
Ví dụ: thay đổi kiểu đường.

```
import matplotlib.pyplot as plt
plt.figure()
cp = plt.contour(X, Y, Z, colors='black', linestyles='dashed')
plt.clabel(cp, inline=True, fontsize=10)
plt.title('Contour Plot')
plt.xlabel('x (cm)')
plt.ylabel('y (cm)')
plt.show()
```



Ví dụ: tô màu cho các contours

```
import numpy as np
import matplotlib.pyplot as plt
xlist = np.linspace(-3.0, 3.0, 100)
ylist = np.linspace(-3.0, 3.0, 100)
X, Y = np.meshgrid(xlist, ylist)
Z = np.sqrt(X**2 + Y**2)
plt.figure()
cp = plt.contourf(X, Y, Z,cmap='magma')
plt.colorbar(cp)
plt.title('Filled Contours Plot')
plt.xlabel('x (cm)')
plt.ylabel('y (cm)')
plt.show()
```



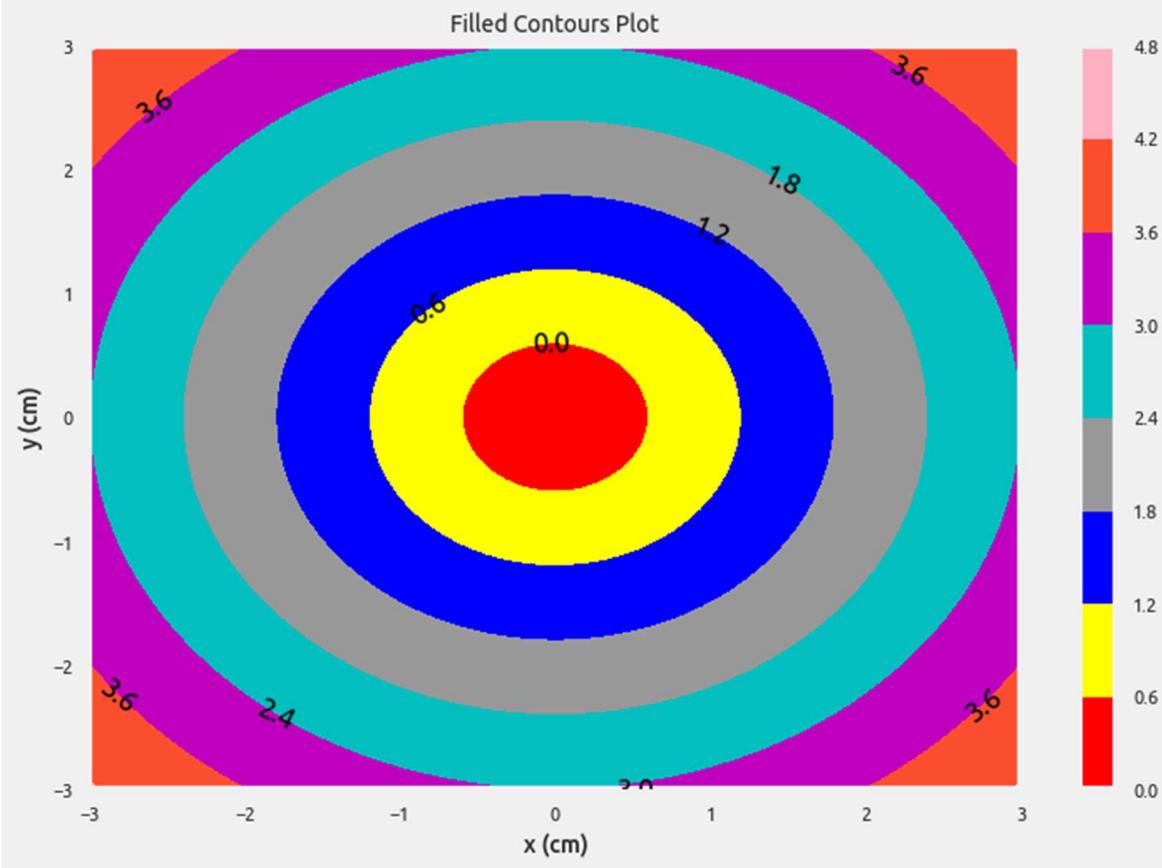
Thay vì dùng cmap, các mức khác nhau sẽ được vẽ bằng các màu khác nhau theo thứ tự được chỉ định qua đối số ‘colors’.

```
import numpy as np
import matplotlib.pyplot as plt
xlist = np.linspace(-3.0, 3.0, 100)
ylist = np.linspace(-3.0, 3.0, 100)
X, Y = np.meshgrid(xlist, ylist)
Z = np.sqrt(X**2 + Y**2)
plt.figure()
contour = plt.contourf(X, Y, Z)
```

```

plt.clabel(contour, colors = 'k', fmt = '%.2f', fontsize=12)
c = ('#ff0000', '#ffff00', '#0000FF', '0.6', 'c', 'm','C1','xkcd:soft pink')
contour_filled = plt.contourf(X, Y, Z, colors=c)
plt.colorbar(contour_filled)
plt.title('Filled Contours Plot')
plt.xlabel('x (cm)')
plt.ylabel('y (cm)')
plt.show()

```



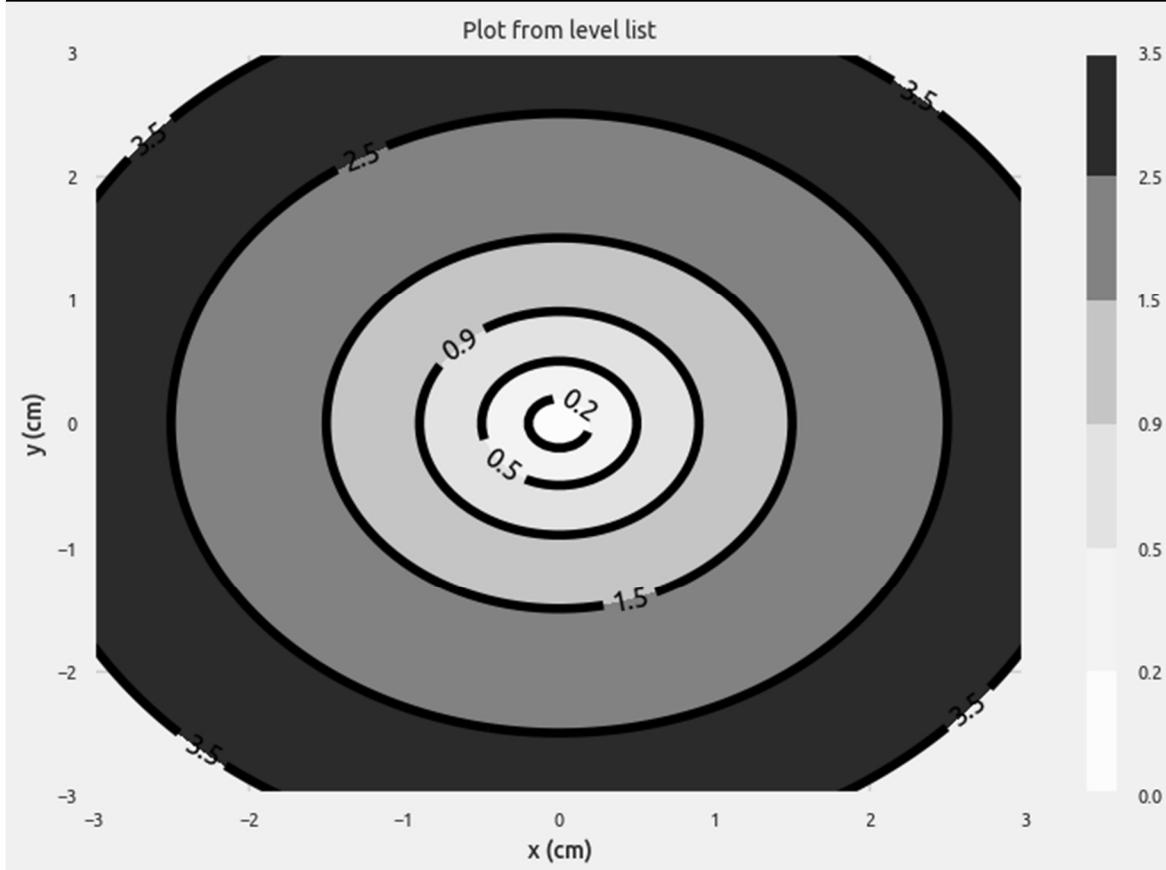
Chúng ta có thể tự định nghĩa bằng các levels, bằng cách cung cấp một danh sách các levels như một tham số thứ tư. Đường viền sẽ được vẽ cho mỗi giá trị trong danh sách, nếu chúng ta sử dụng `contour()`. Đối với `contourf()`, sẽ có tô màu các khu vực giữa các giá trị trong danh sách.

```
import matplotlib.pyplot as plt
```

```

xlist = np.linspace(-3.0, 3.0, 100)
ylist = np.linspace(-3.0, 3.0, 100)
X, Y = np.meshgrid(xlist, ylist)
Z = np.sqrt(X ** 2 + Y ** 2 )
plt.figure()
levels = [0.0, 0.2, 0.5, 0.9, 1.5, 2.5, 3.5]
contour = plt.contour(X, Y, Z, levels, colors='k')
plt.clabel(contour, colors = 'k', fmt = '%.2f', fontsize=12)
contour_filled = plt.contourf(X, Y, Z, levels)
plt.colorbar(contour_filled)
plt.title('Plot from level list')
plt.xlabel('x (cm)')
plt.ylabel('y (cm)')
plt.show()

```



Qua bài học này, bạn đọc đã học được cách sử dụng contour để mô tả dữ liệu của mình. Một loại đồ thị khá thú vị mà ta thường bắt gặp trong các bài học về địa lý khi còn học phổ thông.

6.15. Biểu diễn lỗi qua errorbars

Trong các báo cáo khoa học, số liệu thường có kèm theo các thanh sai số (errorbars). Việc hiển thị thông tin này hiệu quả sẽ truyền tải thông tin đầy đủ hơn.

Pyplot cung cấp một phương thức giúp chúng ta làm việc này một cách hiệu quả và đơn giản:

```
matplotlib.pyplot.errorbar(x, y, yerr=None, xerr=None, fmt="", ecolor=None,
elinewidth=None, capsize=None, barsabove=False, lolims=False, uplims=False, xlolims=False,
xuplims=False, errorevery=1, capthick=None, hold=None, data=None, **kwargs).
```

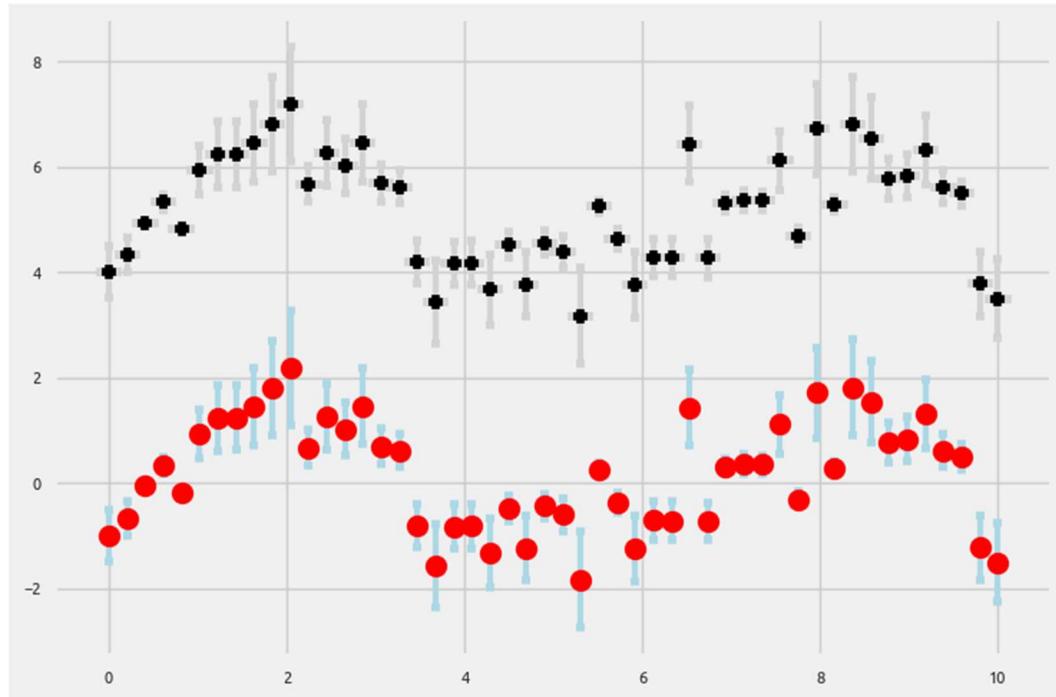
Tham số x,y, xerr, yerr có thể nhận dữ liệu là một số hoặc dữ liệu kiểu mảng. Trong đó fmt là một mã để điều khiển diện mạo của đường kẻ và điểm.

Ngoài các tùy chọn cơ bản như: x,y, xerr, yerr, fmt. errorbar còn có nhiều lựa chọn để tinh chỉnh các kết quả đầu ra. Sử dụng các tùy chọn bổ sung này bạn có thể dễ dàng tùy chỉnh tính thẩm mỹ của graph. Nó thực sự hữu ích, đặc biệt là khi ta phải biểu diễn nhiều dữ liệu trên một hình vẽ, ví dụ như làm cho các thanh lỗi màu nhẹ hơn các điểm chính:

Cùng đi qua một số ví dụ sau để có thể hiểu rõ hơn về các tham số.

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 10, 50)
y = np.sin(x) + 0.8 * np.random.randn(50)
dy = y*0.5
plt.errorbar(x, y, yerr=dy, fmt='o', markeredgewidth=4,
color='red', ecolor='lightblue', elinewidth=3, capsiz=2);
```

```
plt.errorbar(x, y + 5,xerr = 0.1, yerr=dy, fmt='+' ,markeredgecolor ='black', ecolor='lightgray', elinewidth=3, capsized=2);
plt.show()
```



Tham khảo thêm các loại biểu diễn Errorbars tại:

<https://jakevdp.github.io/PythonDataScienceHandbook/04.03-errorbars.html>

8.16. Histogram

Thật khó tưởng tượng rằng bạn mở một tờ báo hoặc tạp chí mà không thấy một số biểu đồ cho bạn biết về số người hút thuốc trong các nhóm tuổi nhất định, số ca sinh trong một năm và v.v. Đó là một cách tuyệt vời để miêu tả các sự kiện mà không cần phải sử dụng quá nhiều từ, nhưng về mặt hạn chế, chúng cũng có thể được sử dụng để thao túng hoặc nói dối qua thống kê.

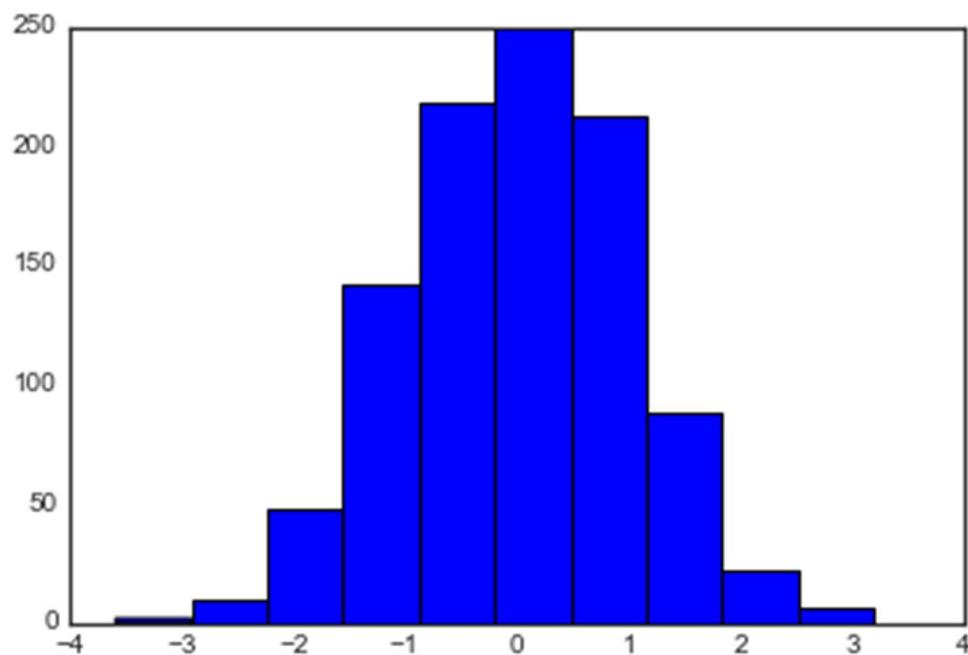
Histogram là gì? Một định nghĩa chính thức có thể là: Đây là một đại diện đồ họa của một phân bố tần suất của một số dữ liệu số. Hình chữ nhật có kích thước bằng nhau theo hướng ngang có chiều cao tương ứng với tần suất.

Nếu chúng ta xây dựng một histogram, chúng ta sẽ bắt đầu phân bố phạm vi của các giá trị x có thể vào các khoảng hoặc thùng chứa thường có kích thước bằng nhau và liền kề.

8.16.1. Histogram một chiều

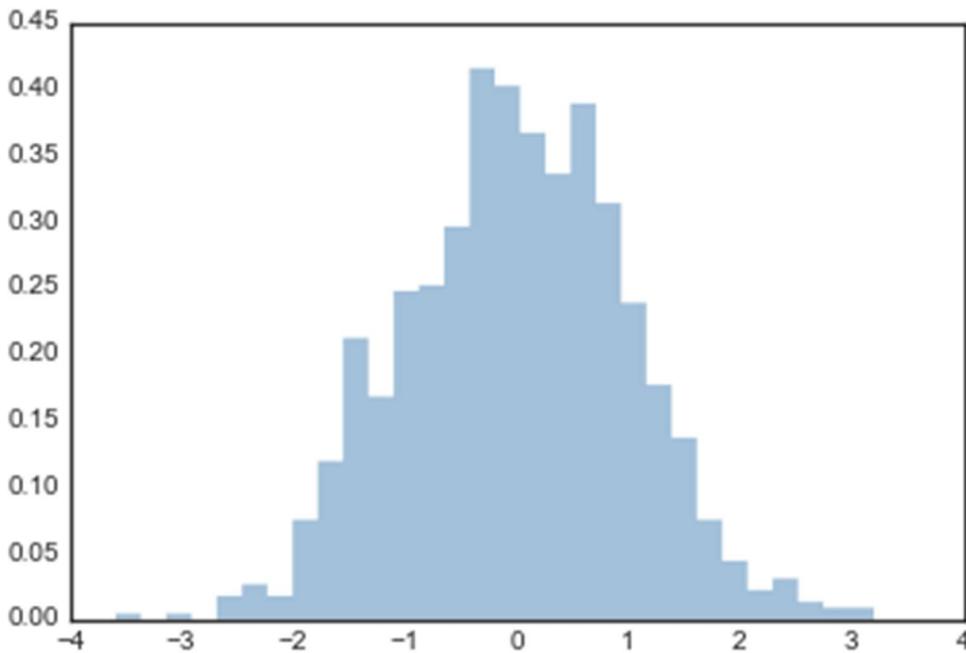
Bây giờ chúng ta bắt đầu với một chương trình Python thực tiễn. Chúng tôi tạo một histogram với các số ngẫu nhiên:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('seaborn-white')
data = np.random.randn(1000)
plt.hist(data);
```



Hàm hist () có nhiều tùy chọn để điều chỉnh cả tính toán và hiển thị; đây là một ví dụ về biểu đồ tùy chỉnh hơn:

```
plt.hist(data, bins=30, normed=True, alpha=0.5,
          histtype='stepfilled', color='steelblue',
          edgecolor='none');
```



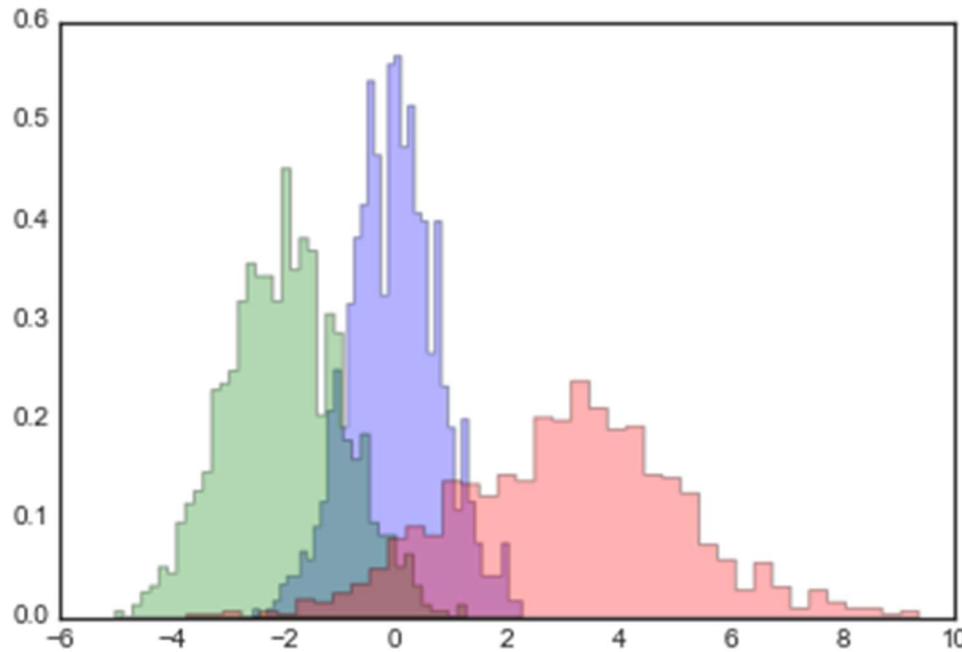
plt.hist docstring có thêm thông tin về các tùy chọn tùy chỉnh khác có sẵn. Nhận thấy sự kết hợp giữa histtype = 'stepfilled' này cùng với một số alpha trong suốt sẽ rất hữu ích khi so sánh biểu đồ của một số bản phân phối:

```
x1 = np.random.normal(0, 0.8, 1000)
x2 = np.random.normal(-2, 1, 1000)
x3 = np.random.normal(3, 2, 1000)

kwargs = dict(histtype='stepfilled', alpha=0.3, normed=True, bins=40)

plt.hist(x1, **kwargs)
plt.hist(x2, **kwargs)
```

```
plt.hist(x3, **kwargs);
```



Nếu bạn chỉ muốn tính toán biểu đồ (nghĩa là đếm số điểm trong một thùng nhất định) và không hiển thị nó, thì hàm np.histogram () có sẵn:

```
counts, bin_edges = np.histogram(data, bins=5)
print(counts)
```

Output:

```
[ 12 190 468 301 29]
```

8.16.2. Histograms và Binnings hai chiều

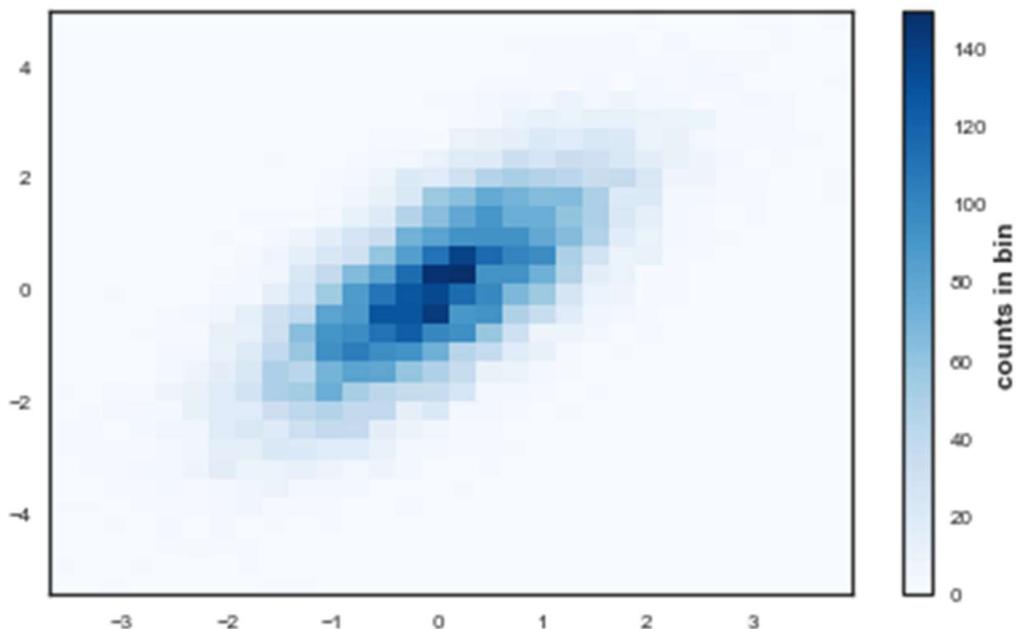
Giống như ta tạo histogram trong một chiều bằng cách chia dữ liệu vào thùng, chúng ta cũng có thể tạo histogram theo hai chiều bằng cách chia điểm giữa các thùng chứa hai chiều. Chúng ta sẽ xem xét một số cách để làm điều này ở đây. Chúng ta sẽ bắt đầu bằng cách xác định một số dữ liệu-mảng x và y được rút ra từ một phân bố Gaussian đa biến:

```
import numpy as np
import matplotlib.pyplot as plt
mean = [0, 0]
```

```
cov = [[1, 1], [1, 2]]
x, y = np.random.multivariate_normal(mean, cov, 10000).T
```

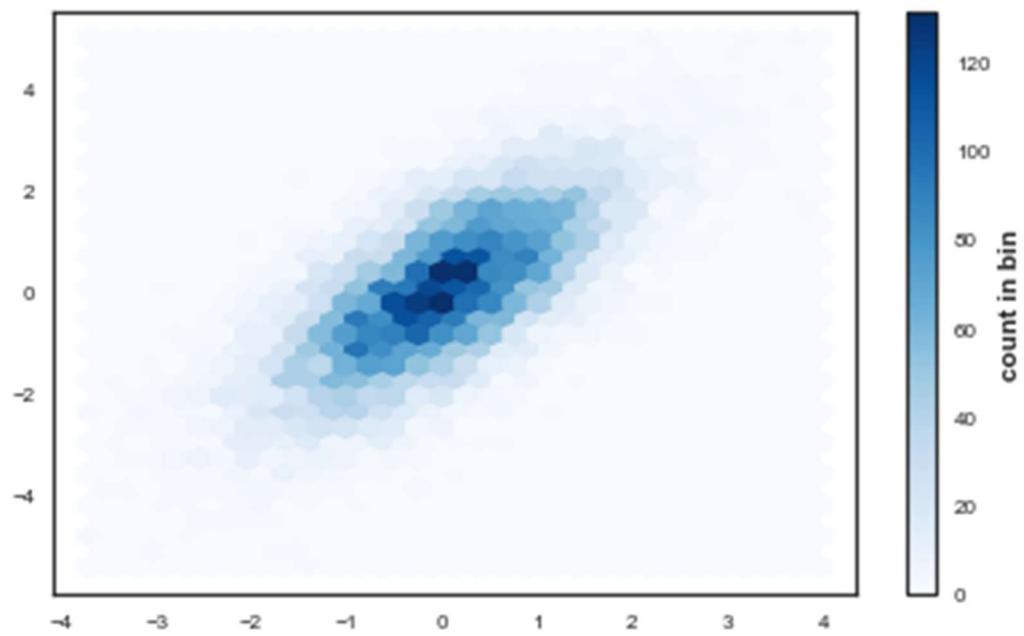
Một cách đơn giản để vẽ histogram hai chiều là sử dụng Matplotlib.

```
plt.hist2d(x, y, bins=30, cmap='Blues')
cb = plt.colorbar()
cb.set_label('counts in bin')
```



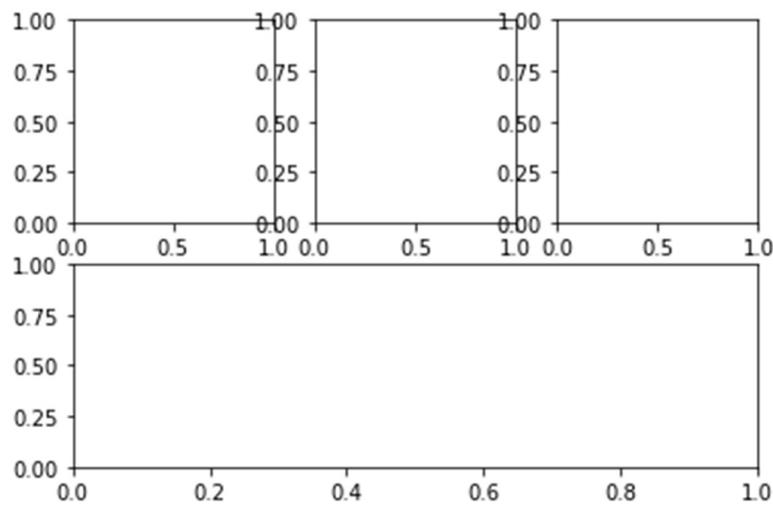
Biểu đồ histogram hai chiều tạo ra một tessellation của hình vuông qua các trục. Một hình dạng tự nhiên khác cho việc tessellation là hình lục giác thông thường. Với mục đích này, Matplotlib cung cấp phương thức plt.hexbin, nó sẽ đại diện cho một bộ dữ liệu hai chiều được chèn trong một ô lưới lục giác:

```
plt.hexbin(x, y, gridsize=30, cmap='Blues')
cb = plt.colorbar(label='count in bin')
plt.show()
```

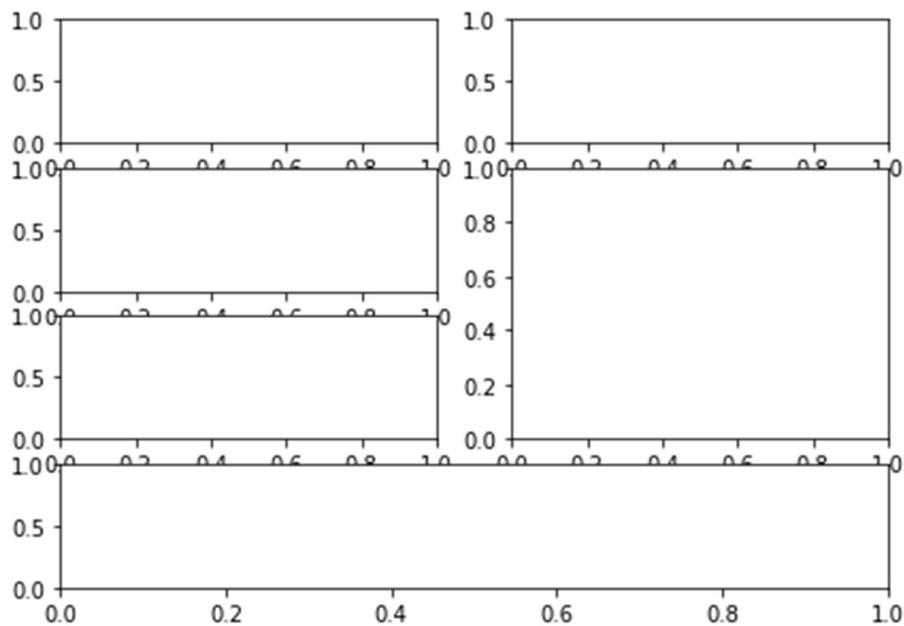


8.17. Bài tập

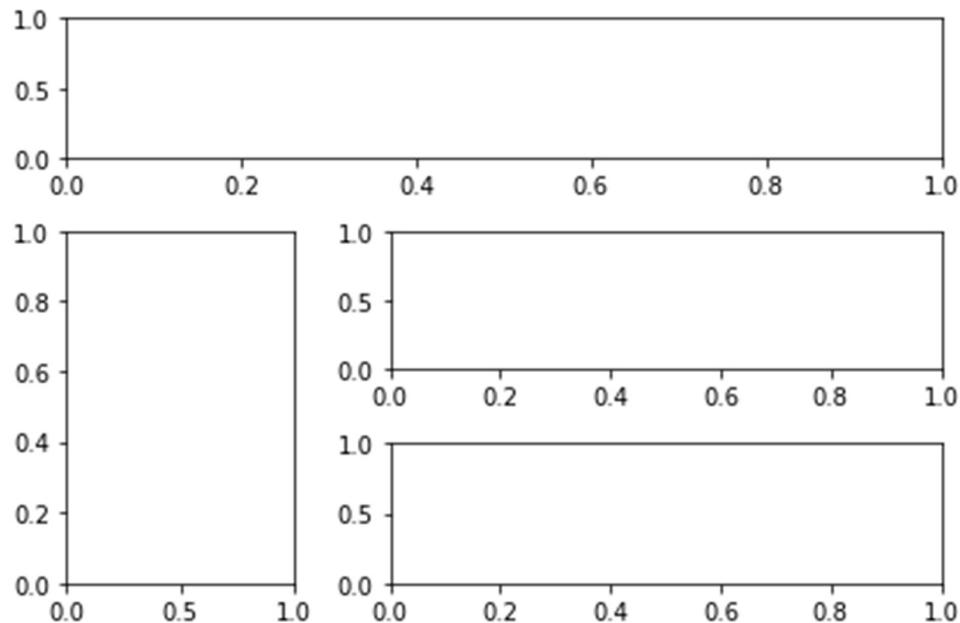
1. Vẽ biểu đồ như hình sau



2. Vẽ biểu đồ như hình sau

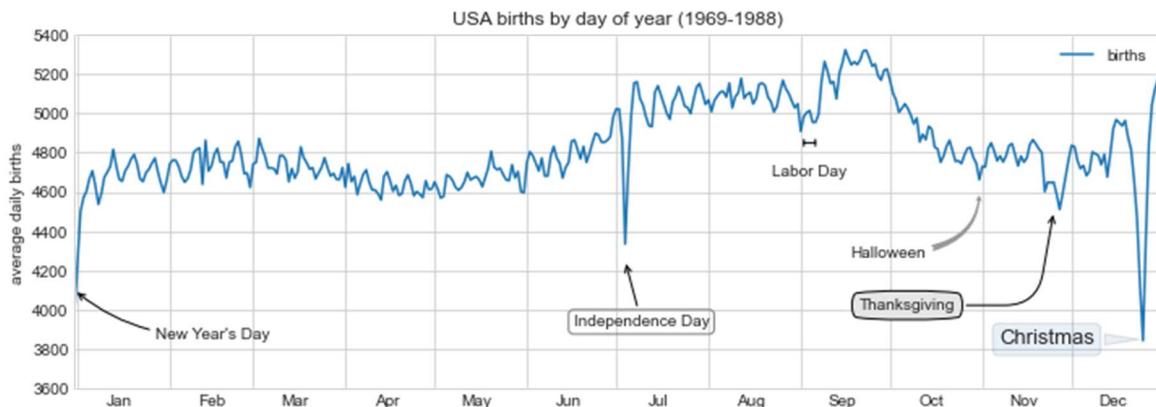


3. Vẽ biểu đồ như hình



4. Vẽ đồ thị như hình bên dưới.

Gợi ý: sử dụng mũi tên được điều khiển thông qua từ điển arrowprops.



5. Vẽ đồ thị như hình. Chú ý phần mũi tên.