

# Tài liệu hướng dẫn thực hành Buổi 1

## Môn : Nguyên Lý Máy Học

Nội dung chính : Giới thiệu ngôn ngữ lập trình Python

### 1. Giới thiệu

Python là một ngôn ngữ lập trình dạng thông dịch do Guido Van Rossum tạo ra năm 1990. Python được viết từ nhiều ngôn ngữ lập trình khác nhau và tạo ra những bản hiện thực khác nhau. Trong đó bản hiện thực chính là CPython được viết bằng C.

Python có tính tương thích lớn trên nhiều môi trường phát triển và cũng được ứng dụng rộng rãi trên nhiều lĩnh vực.

Điểm khác biệt đặc trưng giữa C và Python là Python sử dụng cơ chế cấp phát bộ nhớ tự động và hoàn toàn tạo kiểu động (khác với C phải khai báo biến và cấp phát bộ nhớ tương ứng với kiểu của biến). Điều này giúp tối thiểu hóa số lần gõ phím để viết mã lệnh và giúp cho cấu trúc có hình thức gọn gàng sáng sủa rất thuận tiện cho người mới học lập trình.

### 2. Cài đặt Python và trình soạn thảo

Download Python : Truy cập vào trang web : <https://www.python.org/> để download Python phiên bản 2.7.13

**Chú ý : Cài đặt phiên bản 2.7.13 chứ không cài phiên bản 3.6**


Sau khi cài đặt python ta cần thêm Python vào PATH của hệ thống bằng các bước sau : (chỉ thay đổi khi cài đặt trên môi trường window)

Mở Control Panel

- Chọn System
- Chọn Advanced System Setting
- Chọn tab Advance > Environment Variable
- Trong mục Systeme Variable, tìm chọn variable PATH, copy paste dòng sau vào value của variable này :  
;C:\Python27;C:\Python27\Lib\site-packages;C:\Python27\Scripts\
- Chọn Ok và kết thúc việc thêm python vào PATH của hệ thống

Download trình soạn thảo : Truy cập vào trang web : <https://atom.io/> để download trình soạn thảo Atom

Cách sử dụng Atom để viết một chương trình đơn giản bằng Python :

- Tạo một thư mục trong ổ D và đặt tên là Python
- Mở Atom, từ menu File chọn New File, ta thấy một cửa sổ mới được tạo ra.
- Bấm tổ hợp Ctrl + S để save file đó vào thư mục Python vừa tạo ở ổ đĩa D với tên gọi hello.py
- Trên cửa sổ soạn thảo của Atom viết vào dòng lệnh sau : `print "Hello Python"` và bấm Ctrl + S để save lại
- Mở cửa sổ terminal (bấm tổ hợp  + R, gõ vào `cmd`).
- Trên cửa sổ terminal, gõ vào dòng lệnh `cd D:\Python` để di chuyển đến thư mục mà ta vừa tạo
- (Nếu báo lỗi thì thực hiện theo 2 bước, ta gõ `D:` để di chuyển qua ổ đĩa D, sau đó ta gõ `cd Python` để di chuyển vào thư mục Python)
- Thực thi file hello.py mà ta vừa tạo bằng cách gõ câu lệnh : `python hello.py`

### 3. Các cú pháp cơ bản

Python phân biệt các ký tự thường và hoa, đồng thời tăng cường sử dụng các từ khóa tiếng Anh, hạn chế sử dụng các ký hiệu và cấu trúc cú pháp so với các ngôn ngữ khác

Một số từ khóa thông dụng của Python :

•and	exec	not
•as	finally	or
•assert	for	pass
•break	from	print
•class	global	raise
•continue	if	return
•def	import	try
•del	in	while
•elif	is	with
•else	lambda	yield
•except	continue	

Các toán tử cơ bản :

+ - \* / // (chia làm tròn) % (phần dư) \*\* (lũy thừa)  
~ (not) & (and) | (or) ^ (xor)  
<< (left shift) >> (right shift)  
== (bằng) <= >= != (khác)

Sử dụng ký tự # để chú thích 1 dòng code

Sử dụng ký tự ''' để chú thích 1 đoạn code

Cấu trúc khối lệnh : sử dụng canh lề để bao các khối lệnh của hàm, lớp, hoặc luồng điều khiển.

Số khoảng trắng dùng để canh lề có thể tùy chọn nhưng các lệnh trong cùng một khối phải được canh lề như nhau.

Ví dụ :

```
a = 5
b = 3
if a > b :
    a = a * 2 + 3
    b = b - 6
    c = a/b
    print c
```

Dòng lệnh dài viết trên nhiều dòng sử dụng ký tự \

Ví dụ :

```
c = a + b + \
    10*a - b/4 - \
    5 + 3*a
print c
```

Lệnh nằm trong các cặp dấu ngoặc : [] {} () không cần sử dụng ký tự \ để tiếp tục dòng

Dấu ; để cách nhiều lệnh trên cùng 1 dòng

#### 4. Lệnh và cấu trúc điều khiển

Lệnh if :

```
if biểu_thức_điều_kiện:
    # lệnh...

if biểu_thức_điều_kiện:
    # lệnh...
else:
    # lệnh...
```

Ví dụ :

```

a = 5
b = 3
if a > b :
    print "true"
    print a
else :
    print "false"
    print b

```

Lệnh while :

```

while biểu_thức_đúng:
    # lệnh...

```

Ví dụ :

```

a = 2
b = 6
while a < b :
    a+=1
    print a

```

Lệnh for :

```

for phần_tử in dãy:
    # lệnh...

```

Ví dụ :

```

for i in range (1,10):
    print i

```

Khai báo hàm :

```

def tên_hàm (tham_biến_1, tham_biến_2, tham_biến_n):
    # lệnh...
    return giá_trị_hàm

```

Ví dụ :

```

def binhphuong(number):
    return number * number

print binhphuong(3)

```

## 5. Các kiểu dữ liệu

Python gồm các kiểu dữ liệu cơ bản như sau :

Kiểu số – Number : Kiểu dữ liệu numbers chứa giá trị là một con số, nó được tạo ra khi ta gán các giá trị là một con số cho biến đó.

Python hỗ trợ 4 kiểu dữ liệu numbers khác nhau:

- int
- long
- float
- complex

Ví dụ :

```

a = 5
b = -7
c = 1.234

```

Kiểu chuỗi – String : Kiểu dữ liệu strings được xác định khi giá trị được gán cho nó nằm giữa cặp dấu ' ' hoặc " "

Ví dụ :

```

str1 = "Hello"

```

```
str2 = "welcome"
str3 = "abcdef12345"
```

Kiểu danh sách – List : Một list trong Python là một nhóm có thứ tự các đối tượng. Điều quan trọng cần phải nhấn mạnh là những đối tượng đó không cần phải là cùng loại. Nó có thể là một hỗn hợp tùy ý của các đối tượng như số, chuỗi hoặc có thể là một danh sách khác.

Dưới đây là một số thuộc tính của list:

- List là một tập hợp có thứ tự.
- Dữ liệu trong một list có thể thuộc nhiều kiểu khác nhau.
- Các phần tử trong list có thể được truy xuất thông qua chỉ số của chúng.
- Kích thước của một list có thể thay đổi.
- Chúng ta có thể thay đổi một list, ví dụ như việc thêm mới, xóa hoặc cập nhật phần tử trong list.

Ví dụ : Khai báo một danh sách gồm 5 phần tử kiểu chuỗi :

```
cats = ['Tom', 'Snappy', 'Kitty', 'Jessie', 'Chester']
```

Chúng ta có thể truy xuất các phần tử của list thông qua chỉ số của chúng như sau:

```
print cats[2] # in ra phần tử có chỉ số là 2
// Kitty

print cats[-1] # in ra phần tử cuối cùng của list
// Chester

print cats[1:3] # in ra các phần tử có chỉ số từ 1 đến 3
//[ 'Snappy', 'Kitty']
```

Vì list trong Python có thể thay đổi nên chúng ta có thể thêm mới, sửa hoặc xóa bỏ phần tử trong list:

```
print cats
['Tom', 'Snappy', 'Kitty', 'Jessie', 'Chester']

cats.append('Jerry') # thêm mới phần tử vào list
print cats
['Tom', 'Snappy', 'Kitty', 'Jessie', 'Chester', 'Jerry']

cats[-1] = 'Jerret Cat' # gán lại giá trị cho phần tử cuối cùng của list
print cats
['Tom', 'Snappy', 'Kitty', 'Jessie', 'Chester', 'Jerret Cat']

del cats[1] # xóa bỏ phần tử có số thứ tự 1 khỏi list
print cats
['Tom', 'Kitty', 'Jessie', 'Chester', 'Jerret Cat']
```

Bảng dưới đây liệt kê các hàm dùng để làm việc với list:

- `cmp(list1, list2)`: So sánh phần tử của 2 list
- `len(list)`: Lấy số lượng phần tử trong list.
- `max(list)`: Trả về phần tử có giá trị lớn nhất trong list.
- `min(list)`: Trả về phần tử có giá trị nhỏ nhất trong list.
- `list.append(obj)`: Thêm một phần tử vào list.
- `list.count(obj)`: Đếm số lần xuất hiện của phần tử obj trong list

- `list1.extend(list2)`: Thêm tất cả các phần tử của `list2` vào `list1`
- `list.index(obj)`: Trả về chỉ số bé nhất mà `obj` xuất hiện trong `list`
- `list.insert(index, obj)`: Thêm phần tử `obj` vào vị trí `index` trong `list`.
- `list.pop(index)`: Xóa và trả về phần tử có chỉ số `index` trong `list`.
- `list.remove(obj)`: Xóa phần tử trong `list`.
- `list.reverse()`: Đảo ngược các phần tử trong `list`.
- `list.sort()`: Sắp xếp các phần tử trong `list`.

Kiểu bộ – Tuple : Một Tuple có thể được hiểu là một danh sách không thay đổi, điều này có nghĩa là bạn không thể thay đổi một tuple một khi nó đã được tạo ra. Một tuple được định nghĩa tương tự như list ngoại trừ việc tập hợp các phần tử được đặt trong dấu ngoặc đơn thay vì dấu ngoặc vuông như list, ngoài ra quy tắc về chỉ số phần tử của tuple cũng tương tự như list.

Vậy tại sao Python lại định nghĩa một kiểu dữ liệu tương tự như list? Dưới đây là các ưu điểm của tuple:

- Tốc độ truy xuất, xử lý dữ liệu của tuple nhanh hơn so với list.
- Trong lập trình, có những dữ liệu không được thay đổi, trong trường hợp đó bạn nên dùng tuple thay vì list, điều này sẽ đảm bảo được dữ liệu của bạn, chống lại những sự thay đổi ngẫu nhiên đối với dữ liệu đó.

Bảng dưới đây liệt kê các hàm dùng để làm việc với list:

- `cmp(tuple1, tuple2)`: So sánh phần tử của 2 tuple
- `len(tuple)`: Lấy số lượng phần tử trong tuple.
- `max(tuple)`: Trả về phần tử có giá trị lớn nhất trong tuple.
- `min(tuple)`: Trả về phần tử có giá trị nhỏ nhất trong tuple.
- `tuple(list)`: Chuyển một list sang tuple.

Kiểu từ điển – Dictionary : Một Dictionary trong Python có thể được hiểu là một tập các cặp key-value. Mỗi key được phân cách với giá trị của nó bằng dấu hai chấm (:), các phần tử trong dictionary được phân cách bằng dấu phẩy. Một dictionary rỗng, không chứa bất kỳ phần tử nào được định nghĩa bằng hai dấu ngoặc nhọn như sau: `{}`.

Key trong dictionary phải là duy nhất nhưng điều này không cần thiết với các value, nghĩa là trong một dictionary có thể có nhiều key có cùng một value. Các value trong dictionary có thể thuộc bất kỳ một kiểu nào còn các key thì chỉ được cố định trong một số kiểu như: string, number hoặc tuple.

Dictionary không dùng chỉ số của các phần tử để truy xuất dữ liệu, thay vào đó, chúng ta có thể truy xuất dữ liệu thông qua key như ví dụ sau:

```
dict1 = {'Name' : 'Zyra', 'Age' : 7, 'Class' : 'A5'} # định nghĩa 1 dictionary
print dict['Name'] # lấy giá trị có key = 'Name'
'Zyra'
print dict['Age'] # lấy giá trị có key = 'Age'
7
```

Chúng ta có thể chỉnh sửa một Dictionary như việc thêm mới, xóa, hoặc chỉnh sửa phần tử như ví dụ dưới đây:

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
dict['Age'] = 8 # cập nhật một phần tử trong dict
```

```
dict['School'] = "DPS School" # thêm mới một phần tử vào dict
print dict['Age']
// 8
print dict['School']
// DPS School
```

Dưới đây là danh sách các hàm để làm việc với Dictionary trong Python:

- `cmp(dict1, dict2)`: So sánh phần tử của 2 dict
- `len(dict)`: Lấy số lượng phần tử trong dict.
- `str(dict)`: Tạo ra một chuỗi có thể in được của dict
- `type(variable)`: Trả về kiểu của biến được truyền vào, nếu biến được truyền vào là kiểu dictionary hàm sẽ trả về kiểu dictionary.
- `dict.clear()`: Xóa tất cả các phần tử của dict
- `dict.copy()`: Trả về một bản sao của dict
- `dict.get(key, default=None)`: Trả về giá trị tương ứng với key hoặc trả về giá trị default nếu key không tồn tại trong dict
- `dict.has_key(key)`: Trả về true nếu key tồn tại trong dict, ngược lại trả về false.
- `dict.items()`: Trả về một list các cặp key/value của dict.
- `dict.keys()`: Trả về một list các key của dict
- `dict.setdefault(key, default=None)`: Hàm này gần giống với hàm `get()` nhưng sẽ gán `dict[key]=default` nếu key chưa tồn tại trong dict.
- `dict1.update(dict2)`: Thêm các phần tử của dict2 và dict1.
- `dict.values()`: Trả về danh sách value của dict dưới dạng một list.

## 6. Cài đặt các thư viện cần thiết

Python hỗ trợ rất nhiều các gói thư viện hỗ trợ cho nhiều yêu cầu sử dụng khác nhau từ xử lý đồ họa, bảo mật, thiết kế giao diện hay nâng cao hiệu năng tính toán. Trong nội dung môn học Nguyên Lý Máy Học, chúng ta cần cài đặt và sử dụng 3 thư viện sau :

**NumPy** : Đây là một thư viện cơ bản hỗ trợ rất mạnh mẽ cho việc tính toán trên những ma trận dữ liệu đa chiều

**SciPy** : Đây là thư viện tổng hợp chứa các giải thuật, thuật toán và các công cụ giúp cho việc xử lý tín hiệu, tối ưu hóa, thống kê, và nhiều lĩnh vực khác

**Matplotlib** : Đây là thư viện hỗ trợ cho việc hiển thị các biểu đồ, đồ thị trong không gian 2 chiều hoặc 3 chiều.

Để cài đặt được các thư viện này, ta cần cài đặt 2 công cụ là ***pip*** và ***setuptools***.

Ta copy file `get-pip.py` vào thư mục `D://Python`

Từ cửa sổ terminal, ta gõ lệnh `python get-pip.py` để cài đặt công cụ `pip`

Sau đó ta tiếp tục cài đặt công cụ `setuptools` bằng dòng lệnh : (hệ điều hành Window)

```
python -m pip install -U pip setuptools
```

Nếu sử dụng hệ điều hành Linux hay MacOS ta dùng dòng lệnh sau :

```
pip install -U pip setuptools
```

Sau khi cài đặt thành công 2 công cụ hỗ trợ, ta tiến hành cài đặt các thư viện bằng các dòng lệnh sau :

Cài đặt thư viện Numpy	:	<code>pip install numpy</code>
Cài đặt thư viện SciPy	:	<code>pip install scipy</code>
Cài đặt thư viện Matplotlib	:	<code>pip install matplotlib</code>

## 7. Ví dụ minh họa

Ví dụ 1 : Thao tác trên mảng với thư viện NumPy

```

import numpy as np

#tao mang a gom 6 phan tu
a = np.array([0,1,2,3,4,5])

# in mang a
print a
# in so chieu cua mang a
print a.ndim

# in hình dạng của mang a
print a.shape

# in các phần tử trong mang a có giá trị lớn hơn 3
print a[a>3]

# thay đổi các giá trị lớn hơn 3 của mang a thành 10
a[a>3] = 10

# thay đổi hình dạng của mang a
b = a.reshape((3,2))

# in mang b
print b

# in phần tử hàng 3 cột 2 của mang b
print b[2][1]

# thay đổi giá trị phần tử hàng 3 cột 1 của mang b
b[2][0] = 50

# nhân các giá trị của mang b cho 2 và lưu vào mang c
c = b * 2

# in mang c
print c

```

Ví dụ 2 : Đọc và xử lý dữ liệu từ file bên ngoài với thư viện SciPy, hiển thị dữ liệu với thư viện Matplotlib

```

import scipy as sp

# doc du lieu tu file web_traffic.tsv va luu vao bien data
data = sp.genfromtxt("web_traffic.tsv", delimiter = "\t")

# in hinh dang cua du lieu
print data.shape
# (743,2) ==> du lieu gom la 743 dong va 2 cot

# lay du lieu o cot thu nhat gan cho bien x
x = data[:,0]

# lay du lieu o cot thu hai gan cho bien y
y = data[:,1]

# dem xem co bao nhieu hang ko co du lieu trong cot y
print (sp.isnan(y))

# loai bo cac dong ko co du lieu trong ca 2 cot x va y
x = x[~sp.isnan(y)]
y = x[~sp.isnan(y)]

import matplotlib.pyplot as plt

plt.scatter(x,y)
plt.title("Web traffic over the last month")
plt.xlabel("Time")
plt.ylabel("Hits/hour")
plt.xticks([w*7*24 for w in range(10)],
            ['week %i'%w for w in range(10)])
plt.autoscale(tight=True)
plt.grid()
plt.show()

```

Ta thu được kết quả sau :

