



**TRƯỜNG ĐẠI HỌC VINH**  
**VIỆN KỸ THUẬT VÀ CÔNG NGHỆ**

## **CHƯƠNG 4**

# **CÁC KIỂU DỮ LIỆU CÓ CẤU TRÚC**

**Ngệ An, 2019**

# Chương 4:

## CÁC KIỂU DỮ LIỆU CÓ CẤU TRÚC



### NỘI DUNG GIẢNG DẠY:

4.1. Kiểu strings

4.2. Kiểu lists

4.3. Kiểu Tuples

4.4. Kiểu Dictionary

# Chương 4:

## KIẾN THỨC CHƯƠNG 3

### (LẬP TRÌNH HÀM TRONG PYTHON)



#### NỘI DUNG:

3.1. Định nghĩa hàm trong Python

3.2. Các loại hàm trong Python

3.3. Tham số của hàm

3.4. Hàm vô danh

**3.5. Các loại biến trong Python**

# Chương 4:

## CÁC KIỂU DỮ LIỆU CÓ CẤU TRÚC



### NỘI DUNG GIẢNG DẠY:

#### 4.1. Kiểu Strings

#### 4.2. Kiểu Lists

#### 4.3. Kiểu Tuples

#### 4.4. Kiểu Dictionary

# KIỂU STRINGS

- String là một trong các kiểu phổ biến nhất trong Python. String trong Python là immutable.
- Chúng ta có thể tạo các chuỗi bằng cách bao một text trong một trích dẫn đơn hoặc trích dẫn kép.
- Python coi các lệnh trích dẫn đơn và kép là như nhau.

Ví dụ:

```
var1 = 'Hello World!'
```

```
var2 = "Python Programming"
```

# KIỂU STRINGS

- **Truy cập các giá trị trong String**

Python không hỗ trợ một kiểu chữ cái; chúng được coi như các chuỗi có độ dài là 1. Trong Python, String được lưu giữ dưới dạng các ký tự đơn trong vị trí ô nhớ liên tiếp nhau.

Lợi thế của sử dụng String là nó có thể được truy cập từ cả hai hướng (tiến về trước forward hoặc ngược về sau backward).

Việc lập chỉ mục của cả hai hướng đều được cung cấp bởi sử dụng String trong Python:

- Chỉ mục với hướng forward bắt đầu với 0,1,2,3,...
- Chỉ mục với hướng backward bắt đầu với -1,-2,-3,...

# KIỂU STRINGS

- **Truy cập các giá trị trong String**

Để truy cập các giá trị trong String, bạn sử dụng các dấu ngoặc vuông có chỉ mục ở bên trong.

```
var1 = 'Hello World!'
```

```
var2 = "Python Programming"
```

```
print "var1[0]: ", var1[0]
```

```
print "var2[1:5]: ", var2[1:5]
```

# KIỂU STRINGS

- **Cập nhật String trong Python**

Có thể cập nhật một chuỗi đang tồn tại bằng cách gán (hoặc tái gán) một biến cho string khác.

Giá trị mới có thể liên quan hoặc khác hoàn toàn giá trị trước đó.

Ví dụ:

```
var1 = 'Hello World!'
```

```
print "Chuoi hien tai la :- ", var1[:6] + 'Python'
```

*Khi code trên được thực thi sẽ cho kết quả:*

Chuoi hien tai la :- Hello Python



# KIỂU STRINGS

- **Các toán tử để thao tác với String trong Python**

Có ba kiểu toán tử được hỗ trợ bởi String, đó là:

- Toán tử cơ bản;
- Toán tử membership;
- Toán tử quan hệ.

# KIỂU STRINGS

- **Các toán tử cơ bản để thao tác với String**

- Có hai loại toán tử cơ bản có thể được sử dụng với String, đó là toán tử nối chuỗi + và toán tử lặp chuỗi \*.
- Cả hai toán hạng được truyền cho phép nối chuỗi này phải cùng kiểu, nếu không sẽ tạo một lỗi.

Ví dụ:

```
>>> "hoang" + "nam"
```

*Khi code trên được thực thi sẽ cho kết quả:*

`'hoangnam'`

# KIỂU STRINGS

- **Các toán tử membership để thao tác với String**

- Toán tử in: trả về true nếu một ký tự là có mặt trong chuỗi đã cho, nếu không nó trả về false.
- Toán tử not in: trả về true nếu một ký tự là không tồn tại trong chuỗi đã cho, nếu không nó trả về false.

- **Các toán tử quan hệ để thao tác với String**

- Tất cả các toán tử quan hệ (như  $<$ ,  $>$ ,  $<=$ ,  $>=$ ,  $==$ ,  $!=$ ,  $<>$ ) cũng có thể áp dụng cho các String.
- Các chuỗi được so sánh dựa trên giá trị ASCII hoặc Unicode.

# THẢO LUẬN NHÓM

## **NỘI DUNG:**

1. Các toán tử định dạng chuỗi trong Python.
2. Các phương thức và hàm đã xây dựng sẵn để xử lý chuỗi trong Python.

# THẢO LUẬN NHÓM

## (Hàm và các toán tử cơ bản)



- Một chuỗi được xem như một hàng (tuple) các chuỗi con độ dài 1
  - Trong python không có kiểu kí tự (character)
  - Nội dung của chuỗi không thay đổi được, khi ghép thêm nội dung vào chuỗi thực chất là tạo ra chuỗi mới
- Hàm `len(s)` trả về độ dài (số chữ) của s
- Phép toán với chuỗi:
  - Phép nối chuỗi (+): `s = "Good" + " " + "Morning!"`
  - Phép nhân bản (\*): `s = "AB" * 3` # ABABAB
  - Kiểm tra nội dung: `s in '1ABABABCD'` # True

# THẢO LUẬN NHÓM

## (Phép cắt chuỗi)

- Dựa trên chỉ mục, phép cắt chuỗi cho phép lấy nội dung bên trong của chuỗi bằng cú pháp như sau
  - `<chuỗi>[vị trí A : vị trí B]`
  - `<chuỗi>[vị trí A : vị trí B : bước nhảy]`
- Giải thích:
  - Tạo chuỗi con bắt đầu từ <vị-trí-A> đến trước <vị-trí-B>
    - Tức là chuỗi con sẽ không gồm vị trí B
  - Nếu không ghi <vị-trí-A> thì mặc định là lấy từ đầu
  - Nếu không ghi <vị-trí-B> thì mặc định là đến hết chuỗi
  - Nếu không ghi <bước-nhảy> thì mặc định bước là 1
  - Nếu <bước-nhảy> giá trị âm thì sẽ nhận chuỗi ngược lại

```
s = '0123456789'
print(s[3:6])           # 345
print(s[3:])            # 3456789
print(s[:6])            # 012345
print(s[-7:-4])         # 345
print(s[-4:-7])         #
print(s[-4:-7:-1])      # 654
print(s[:len(s)])       # 0123456789
print(s[:len(s)-1])     # 012345678
print(s[:])             # 0123456789
print(s[len(s)::-1])    # 9876543210
print(s[len(s)-1::-1])  # 9876543210
print(s[len(s)-2::-1])  # 876543210
```

# THẢO LUẬN NHÓM

## (Định dạng chuỗi)

- Dùng toán tử %: <chuỗi> % (<các tham số>)
  - Bên trong <chuỗi> có các kí hiệu đánh dấu nơi đặt lần lượt các tham số
  - Nếu đánh dấu %s: thay thế bằng tham số dạng chuỗi
  - Nếu đánh dấu %d: thay thế bằng tham số dạng nguyên
  - Nếu đánh dấu %f: thay thế bằng tham số dạng thực
- Ví dụ:

```
"Chao %s, gio la %d gio" % ('txnam', 10)
"Can bac 2 cua 2 = %f" % (2**0.5)
"Can bac 2 cua 2 = %10.3f" % (2**0.5)
"Can bac 2 cua 2 = %10f" % (2**0.5)
"Can bac 2 cua 2 = %.7f" % (2**0.5)
```



# THẢO LUẬN NHÓM

## (Định dạng chuỗi)

- Python cho phép định dạng chuỗi ở dạng f-string

```
myname = 'DHTL'
s = f'This is {myname}.'           # 'This is DHTL.'
w = f'{s} {myname}'               # 'This is DHTL. DHTL'
z = f'{{s}} {s}'                  # '{s} This is DHTL.'
```

- Mạnh mẽ nhất là định dạng bằng format

```
# điền lần lượt từng giá trị vào giữa cặp ngoặc nhọn
'a: {}, b: {}, c: {}'.format(1, 2, 3)
# điền nhưng không lần lượt
'a: {1}, b: {2}, c: {0}'.format('one', 'two', 'three')
'two same values: {0}, {0}'.format(1, 2)
# điền và chỉ định từng giá trị
'1: {one}, 2: {two}'.format(one=111, two=222)
```



# THẢO LUẬN NHÓM

## (Định dạng chuỗi)



### ■ Định dạng bằng format cho phép căn lề phong phú

# căn giữa: ' aaaa '

`'{: ^10}'.format('aaaa')`

# căn lề trái: 'aaaa '

`'{: <10}'.format('aaaa')`

# căn lề phải ' aaaa'

`'{: >10}'.format('aaaa')`

# căn lề phải, thay khoảng trắng bằng -: '-----aaaa'

`'{: ->10}'.format('aaaa')`

# căn lề trái, thay khoảng trắng \*: 'aaa\*\*\*\*\*'

`'{: * <10}'.format('aaaa')`

# căn giữa, thay khoảng trắng bằng +: '+++aaaa+++'

`'{: + ^10}'.format('aaaa')`

# THẢO LUẬN NHÓM

## (Phương thức và hàm có sẵn)



- Các phương thức chỉnh dạng
  - `capitalize()`: viết hoa chữ cái đầu, còn lại viết thường
  - `upper()`: chuyển hết thành chữ hoa
  - `lower()`: chuyển hết thành chữ thường
  - `swapcase()`: chữ thường thành hoa và ngược lại
  - `title()`: chữ đầu của mỗi từ viết hoa, còn lại viết thường
- Các phương thức căn lề
  - `center(width [,fillchar])`: căn lề giữa với độ dài width
  - `rjust(width [,fillchar])`: căn lề phải
  - `ljust(width [,fillchar])`: căn lề trái

# THẢO LUẬN NHÓM

## (Phương thức và hàm có sẵn)



- Các phương thức cắt phần dư
  - `strip([chars])`: loại bỏ những ký tự đầu hoặc cuối chuỗi thuộc vào danh sách [chars], hoặc ký tự trống
  - `rstrip([chars])`: làm việc như strip nhưng cho bên phải
  - `lstrip([chars])`: làm việc như strip nhưng cho bên trái
- Tách chuỗi
  - `split(sep, maxsplit)`: tách chuỗi thành một danh sách, sử dụng dấu ngăn cách sep, thực hiện tối đa maxsplit lần
    - Tách các số nhập vào từ một dòng: `input("Test: ").split(',')`
  - `rsplit(sep, maxsplit)`: thực hiện như split nhưng theo hướng ngược từ phía cuối chuỗi

# THẢO LUẬN NHÓM

## (Phương thức và hàm có sẵn)



- Các phương thức khác
  - `join(list)`: ghép các phần tử trong list bởi phần gạch nối là nội dung của chuỗi, ví dụ: `'-'.join(('1', '2', '3'))`
  - `replace(old, new [,count])`: thế nội dung old bằng nội dung new, tối đa count lần
  - `count(sub, [start, [end]])`: đếm số lần xuất hiện của sub
  - `startswith(prefix)`: kiểm tra đầu có là prefix không
  - `endswith(prefix)`: kiểm tra cuối có là prefix không
  - `find(sub[, start[, end]])`: tìm vị trí của sub (-1: không có)
  - `rfind(sub[, start[, end]])`: như find nhưng tìm từ cuối
  - `islower()`, `isupper()`, `istitle()`, `isdigit()`, `isspace()`, `isalpha()`, `isnumeric()`
  - `index(sub)` giống find, nhưng sinh ngoại lệ nếu không tìm thấy

# Chương 4:

## CÁC KIỂU DỮ LIỆU CÓ CẤU TRÚC



### NỘI DUNG GIẢNG DẠY:

4.1. Kiểu Strings

**4.2. Kiểu Lists**

4.3. Kiểu Tuples

4.4. Kiểu Dictionary

# LIST TRONG PYTHON

- List trong Python là cấu trúc dữ liệu mà có khả năng lưu giữ các kiểu dữ liệu khác nhau.
- List trong Python là thay đổi (mutable), nghĩa là Python sẽ không tạo một List mới nếu bạn sửa đổi một phần tử trong List.
- List là một container mà giữ các đối tượng khác nhau trong một thứ tự đã cho. Các hoạt động khác nhau như chèn hoặc xóa có thể được thực hiện trên List.
- Một List có thể được tạo ra bởi lưu trữ một dãy các kiểu giá trị khác nhau được phân biệt bởi các dấu phẩy.



# LIST TRONG PYTHON

- **Cú pháp để tạo List:**

```
<ten_list>=[giatri1, giatri2, ..., giatriN];
```

Một List trong Python được bao xung quanh bởi các dấu ngoặc vuông [].

Ví dụ:

```
list1 = ['vatly', 'hoahoc', 1997, 2000];
```

```
list2 = [1, 2, 3, 4, 5];
```

```
list3 = ["a", "b", "c", "d"];
```

# LIST TRONG PYTHON

- **Truy cập các giá trị trong List trong Python:**

Tương tự như chỉ mục của chuỗi, chỉ mục của List bắt đầu từ 0.  
Để truy cập các giá trị trong List, sử dụng cú pháp sau:

```
<ten_list>[index]
```

để lấy giá trị có sẵn tại chỉ mục đó.

Ví dụ:

```
list1 = ['vatly', 'hoahoc', 1997, 2000];  
print "list1[0]: ", list1[0]
```

*Kết quả:*

list1[0]: vatly



# LIST TRONG PYTHON

- **Các hoạt động cơ bản trên List trong Python:**

- Có thể thực hiện các hoạt động nối với toán tử + hoặc hoạt động lặp với \* như trong các chuỗi. Điểm khác biệt là ở đây nó tạo một List mới, không phải là một chuỗi.

Ví dụ:

```
list1=[10,20]
```

```
list2=[30,40]
```

```
list3=list1+list2
```

```
print list3
```

*Kết quả:*

```
>>>
```

```
[10, 20, 30, 40]
```

```
>>>
```

# LIST TRONG PYTHON

- **Cập nhật List trong Python:**

- Có thể cập nhật một hoặc nhiều phần tử của List bởi gán giá trị cho chỉ mục cụ thể đó. Cú pháp:

```
<ten_list>[index]=<giatri>
```

```
list = ['vatly', 'hoahoc', 1997, 2000];  
list[2] = 2001;  
print "Gia tri moi tai chi muc thu 2: "  
print list[2]
```

*Kết quả:*

Gia tri moi tai chi muc thu 2 :  
2001

# LIST TRONG PYTHON

- **Xóa phần tử trong List:**

- Để xóa một phần tử trong List, bạn có thể sử dụng lệnh `del` nếu bạn biết chính xác phần tử nào bạn muốn xóa hoặc sử dụng phương thức `remove()` nếu bạn không biết. Ví dụ:

```
list1 = ['vatly', 'hoahoc', 1997, 2000];  
del list1[2];  
print "Cac phan tu cua List sau khi xoa: "  
print list1
```

*Kết quả:*

Cac phan tu cua List sau khi  
xoa:

`['vatly', 'hoahoc', 2000]`

# THẢO LUẬN NHÓM

## NỘI DUNG:

1. Khởi tạo list, các toán tử thực hiện trên list
2. Các hàm và phương thức đã xây dựng sẵn để xử lý List trong Python.

# THẢO LUẬN NHÓM

## (Khởi tạo list)

- List = dãy các đối tượng (một loại array đa năng)
- Các phần tử con trong list không nhất thiết phải cùng kiểu dữ liệu
- Khai báo trực tiếp: liệt kê các phần tử con đặt trong cặp ngoặc vuông (`[]`), ngăn cách bởi dấu phẩy (,)

<code>[1, 2, 3, 4, 5]</code>	<code># list 5 số nguyên</code>
<code>['a', 'b', 'c', 'd']</code>	<code># list 4 chuỗi</code>
<code>[[1, 2], [3, 4]]</code>	<code># list 2 list con</code>
<code>[1, 'one', [2, 'two']]</code>	<code># list hỗn hợp</code>
<code>[]</code>	<code># list rỗng</code>

- Kiểu chuỗi (str) trong python có thể xem như một list đặc biệt, bên trong gồm toàn các str độ dài 1

# THẢO LUẬN NHÓM

## (Khởi tạo list)



- Tạo list bằng constructor

```
l1 = list([1, 2, 3, 4])    # list 4 số nguyên
l2 = list('abc')           # list 3 chuỗi con
l3 = list()                # list rỗng
```

- Tạo list bằng list comprehension: một đoạn mã ngắn trả về các phần tử thuộc list

```
# list 1000 số nguyên từ 0 đến 999
X = [n for n in range(1000)]
# list gồm 10 list con là các cặp [x, x2]
# với x chạy từ 0 đến 9
Y = [[x, x*x] for x in range(10)]
```

# THẢO LUẬN NHÓM

## (Toán tử, chỉ mục và cắt)



- Giữa list và str có sự tương đồng nhất định
  - List cũng hỗ trợ 3 phép toán: ghép nối (+), nhân bản (\*) và kiểm tra nội dung (in)
  - List sử dụng hệ thống chỉ mục và các phép cắt phần con tương tự như str

- Điểm khác biệt: nội dung của list có thể thay đổi

# khởi tạo list ban đầu

```
l1 = list([1, 2, 3, 4])
```

# thay đổi giá trị của phần tử cuối cùng

```
l1[-1] = list('abc')
```

# in nội dung list: [1, 2, 3, ['a', 'b', 'c']]

```
print(l1)
```



# THẢO LUẬN NHÓM

## (Phương thức, hàm có sẵn)



- Một số phương thức thường hay sử dụng
  - `count(sub, [start, [end]])`: đếm số lần xuất hiện của sub
  - `index(sub[, start[, end]])`: tìm vị trí xuất hiện của sub, trả về `ValueError` nếu không tìm thấy
  - `clear()`: xóa trắng list
  - `append(x)`: thêm x vào cuối list
  - `extend(x)`: thêm các phần tử của x vào cuối list
  - `insert(p, x)`: chèn x vào vị trí p trong list
  - `pop(p)`: bỏ phần tử thứ p ra khỏi list (trả về giá trị của phần tử đó), nếu không chỉ định p thì lấy phần tử cuối



# THẢO LUẬN NHÓM

## (Phương thức, hàm có sẵn)



- Một số phương thức thường hay sử dụng
  - `copy()`: tạo bản sao của list (tương tự `list[:]`)
  - `remove(x)`: bỏ phần tử đầu tiên trong list có giá trị x, báo lỗi `ValueError` nếu không tìm thấy
  - `reverse()`: đảo ngược các phần tử trong list
  - `sort(key=None, reverse=False)`: mặc định là sắp xếp các phần tử từ bé đến lớn trong list bằng cách so sánh trực tiếp giá trị

```
x = "Trương Xuân Nam".split()
x.sort(key=str.lower)
print(x)
```

# THẢO LUẬN NHÓM (Examples)



```
# tạo và in nội dung của một list
danh sach = ["apple", "banana", "cherry"]
for x in danh sach:
    print(x)

# thêm một phần tử vào cuối list và lại in ra
danh sach.append("orange")
print(danh sach)

# thêm một phần tử vào giữa list và in ra
danh sach.insert(1, "orange")
print(danh sach)

# sắp xếp lại danh sách và in ra
danh sach.sort()
print(danh sach)
```

# THẢO LUẬN NHÓM

## (Some Operations on list)



1. Tạo một list
2. Duyệt list
3. Truy cập phần tử theo chỉ số
4. Thay đổi nội dung phần tử
5. Cắt list
6. Thêm phần tử vào list
7. Bỏ phần tử khỏi list
8. Tìm kiếm phần tử trong list
9. Sắp xếp các phần tử trong list
10. List lồng ghép

# BÀI TẬP

## NỘI DUNG:

1. Tạo một biến `my_string` và gán cho nó một chuỗi nội dung bất kỳ, sau đó print ra chiều dài chuỗi đó, cuối cùng là print ra chuỗi đó đã được chuyển thành viết hoa hoàn toàn.

2. Nhập  $m, n$  để tạo 2 List số nguyên ngẫu nhiên  $A[n]$  và  $B[m]$  có giá trị khoảng  $\{-100, +100\}$ . Ghép  $A$  và  $B$  thành List  $C[p=m+n]$ . In ra List  $A, B$  và  $C$ .

- Sắp xếp List  $C$  tăng dần.
- Nhập số nguyên  $x$ , kiểm tra  $x$  có xuất hiện trong List  $C$  hay không, nếu có thì xuất hiện mấy lần, ở vị trí đầu tiên nào.
- Xóa các phần tử trùng nhau của List  $C$ , in ra List  $C$  mới.

# BÀI TẬP

## CHUẨN BỊ CHO BUỔI HỌC TIẾP THEO:

1. Đọc các tài liệu về nội dung mục 4.3; 4.4.
2. Các hàm được xây dựng sẵn cho Tuple, Dictionary trong Python.

# Chương 4:

## CÁC KIỂU DỮ LIỆU CÓ CẤU TRÚC

### NỘI DUNG GIẢNG DẠY:

4.1. Kiểu Strings

4.2. Kiểu Lists

**4.3. Kiểu Tuples**

4.4. Kiểu Dictionary

# KIỂU TUPLES

- Một tuple là một dãy các đối tượng không thay đổi (immutable) trong Python, vì thế tuple không thể bị thay đổi.
- Các tuple cũng là các dãy giống như List.
- Không giống List sử dụng các dấu ngoặc vuông, thì tuple sử dụng các dấu ngoặc đơn.
- Các đối tượng trong tuple được phân biệt bởi dấu phẩy và được bao quanh bởi dấu ngoặc đơn ().
- Giống như chỉ mục của chuỗi, chỉ mục của tuple bắt đầu từ 0.

# KIỂU TUPLES

**Ví dụ:**

```
fruitTuple = ("apple", "apricot", "banana", "coconut", "lemon")
```

```
otherTuple = (100, "one", "two", 3)
```

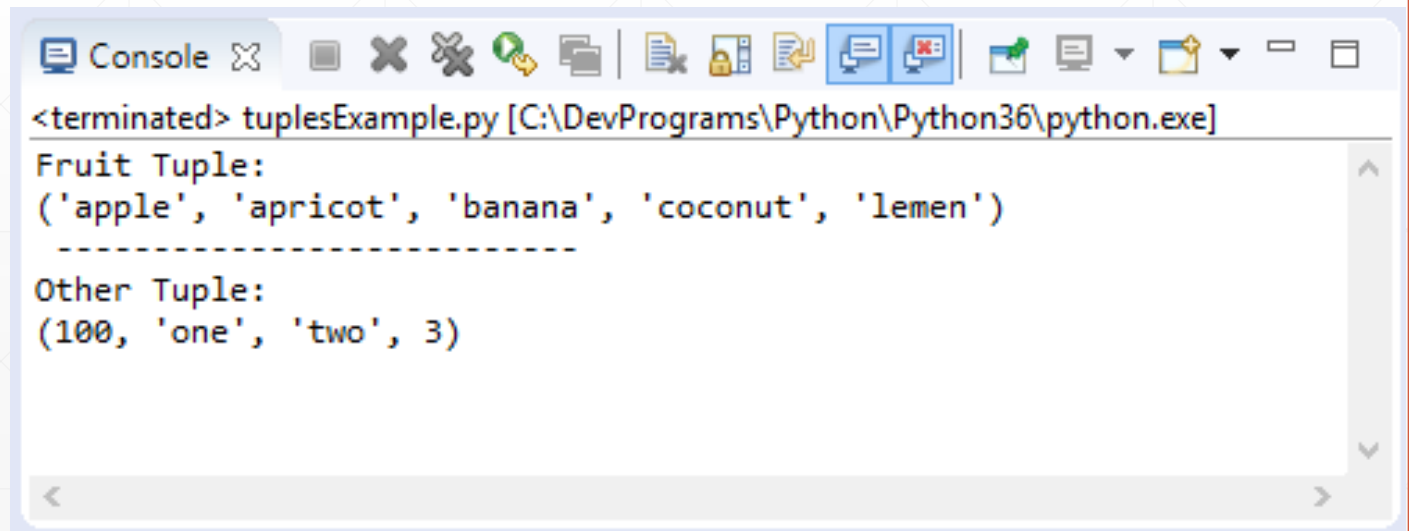
```
print ("Fruit Tuple:")
```

```
print (fruitTuple)
```

```
print (" ----- ")
```

```
print ("Other Tuple:")
```

```
print (otherTuple)
```

A screenshot of a Python console window titled 'Console'. The window shows the execution of a script named 'tuplesExample.py'. The output displays the 'fruitTuple' as a tuple of strings: ('apple', 'apricot', 'banana', 'coconut', 'lemon'), followed by a separator line of dashes, and then the 'otherTuple' as a tuple containing an integer and two strings: (100, 'one', 'two', 3).

```
<terminated> tuplesExample.py [C:\DevPrograms\Python\Python36\python.exe]
Fruit Tuple:
('apple', 'apricot', 'banana', 'coconut', 'lemon')
-----
Other Tuple:
(100, 'one', 'two', 3)
```



# KIỂU TUPLES

- **So sánh List và Tuple**

List và Tuple đều là một dãy (sequence) các phần tử. Chúng có các khác biệt sau:

- Khi viết một List bạn sử dụng cặp dấu ngoặc vuông [ ], trong khi viết một Tuple bạn sử dụng dấu ngoặc tròn ( ).
- List là kiểu dữ liệu có thể biến đổi (mutable), bạn có thể sử dụng phương thức như `append()` để thêm phần tử vào List, hoặc sử dụng phương thức `remove()` để xóa các phần tử ra khỏi List mà không làm tạo ra thêm một thực thể 'List' khác trên bộ nhớ.

# KIỂU TUPLES

- **Truy cập các phần tử của Tuples**

- Có thể truy cập vào các phần tử của Tuple thông qua chỉ số.
- Các phần tử của Tuple được đánh chỉ chỉ từ trái sang phải, bắt đầu từ 0.
- Có thể truy cập vào các phần tử của Tuple theo chỉ số âm (Negative index), các phần tử được đánh chỉ số từ phải sang trái với các giá trị -1, -2, ...

# KIỂU TUPLES

- **Truy cập các phần tử của Tuples**

Truy cập các giá trị trong tuple tương tự như khi truy cập các phần tử trong List.

Ví dụ:

```
tup1 = ('vatly', 'hoahoc', 1997, 2000);
```

```
tup2 = (1, 2, 3, 4, 5, 6, 7 );
```

```
print "tup1[0]: ", tup1[0]
```

```
print "tup2[1:5]: ", tup2[1:5]
```

*Kết quả:*

```
tup1[0]: vatly
```

```
tup2[1:5]: [2, 3, 4, 5]
```

# KIỂU TUPLES

- **Các hoạt động cơ bản trên tuple trong Python**

Giống như String và List, chúng ta cũng có thể sử dụng toán tử nối + và toán tử lặp \* với tuple. Điểm khác biệt là nó tạo ra một tuple mới, không tạo ra một chuỗi hay list.

Ví dụ:

```
data1=(1,2,3,4)
```

```
data2=('x','y','z')
```

```
data3=data1+data2
```

```
print data3
```

*Kết quả:*

```
>>>
```

```
(1, 2, 3, 4, 'x', 'y', 'z')
```

```
>>>
```

# KIỂU TUPLES

- **Xóa các phần tử của tuple trong Python**

Xóa các phần tử đơn của tuple là điều không thể. Chúng ta chỉ có thể xóa toàn bộ tuple với lệnh del.

Ví dụ: Chú ý rằng sẽ có một exception được tạo ra, đó là bởi vì sau khi xóa thì tuple này không tồn tại nữa.

```
data=(10,20,'hoang',40.6,'z')
```

```
print data
```

```
del data          # Sẽ xóa dữ liệu của tuple
```

```
print data        # Sẽ hiển thị một error bởi vì tuple đã bị xóa
```

# THẢO LUẬN NHÓM

## **NỘI DUNG:**

Các hàm và phương thức đã xây dựng sẵn để xử lý Tuple trong Python.

# THẢO LUẬN NHÓM

- Tuple = dãy các đối tượng (list), nhưng không thể bị thay đổi giá trị trong quá trình tính toán
- Như vậy str giống tuple nhiều hơn list
- Khai báo trực tiếp bằng cách liệt kê các phần tử con đặt trong cặp ngoặc tròn (), ngăn cách bởi phẩy

(1, 2, 3, 4, 5)

# tuple 5 số nguyên

('a', 'b', 'c', 'd')

# tuple 4 chuỗi

(1, 'one', [2, 'two'])

# tuple hỗn hợp

(1,)

# tuple 1 phần tử

()

# tuple rỗng



# THẢO LUẬN NHÓM

- Tuple hỗ trợ 3 phép toán: +, \*, in
- Tuple cho phép sử dụng chỉ mục và cắt
- Các phương thức thường dùng của tuple
  - `count(v)`: đếm số lần xuất hiện của v trong tuple
  - `index(sub[, start[, end]])`: tương tự như str và list
- Tuple khác list ở điểm nào?
  - Chiếm ít bộ nhớ hơn
  - Nhanh hơn

# THẢO LUẬN NHÓM

- Chúng ta đã làm quen với range khi dùng vòng for
  - `range(stop)`: tạo miền từ 0 đến stop-1
  - `range(start, stop[, step])`: tạo miền từ start đến stop-1, với bước nhảy là step
    - Nếu không chỉ định thì `step = 1`
    - Nếu `step` là số âm sẽ tạo miền đếm giảm dần (`start > stop`)
- Vậy range khác gì một tuple đặc biệt
  - Range chỉ chứa số nguyên
  - Range nhanh hơn rất nhiều
  - Range chiếm ít bộ nhớ hơn
  - Range vẫn hỗ trợ chỉ mục và cắt (nhưng khá đặc biệt)

# Chương 4:

## CÁC KIỂU DỮ LIỆU CÓ CẤU TRÚC



### NỘI DUNG GIẢNG DẠY:

4.1. Kiểu Strings

4.2. Kiểu Lists

4.3. Kiểu Tuples

**4.4. Kiểu Dictionary**

4.5. Bài tập

# KIỂU DICTIONARY

- Dictionary trong Python là một tập hợp các cặp key và value không có thứ tự.
- Là một container mà chứa dữ liệu, được bao quanh bởi các dấu ngoặc móc đơn `{}`.
- Mỗi cặp key - value được xem như là một item. Key mà đã truyền cho item đó phải là duy nhất, trong khi đó value có thể là bất kỳ kiểu giá trị nào.
- Key phải là một kiểu dữ liệu không thay đổi (immutable) như chuỗi, số hoặc tuple.

# KIỂU DICTIONARY

- Key và value được phân biệt riêng rẽ bởi một dấu hai chấm (:).
- Các item phân biệt nhau bởi một dấu phẩy (,).
- Các item khác nhau được bao quanh bên trong một cặp dấu ngoặc móc đơn tạo nên một Dictionary trong Python.

Ví dụ:

```
data={100:'Hoang' ,101:'Nam' ,102:'Binh'}
```

```
print data
```

*Kết quả là:*

```
{100: 'Hoang', 101: 'Nam', 102: 'Binh'}
```

# KIỂU DICTIONARY

- **Các thuộc tính của key trong Dictionary:**

Không có hạn chế nào với các value trong Dictionary, tuy nhiên với key thì bạn cần chú ý các điểm sau:

**(a)** Nhiều hơn một entry cho mỗi key là không được phép. Nghĩa là không cho phép bản sao các key được xuất hiện. Khi bắt gặp nhiều bản sao key trong phép gán, thì phép gán cuối cùng được thực hiện.

Ví dụ:

```
dict = {'Ten': 'Hoang', 'Tuoi': 7, 'Ten': 'Nam'};  
print "dict['Ten']: ", dict['Ten']
```

Kết quả là:

dict['Ten']: Nam

# KIỂU DICTIONARY

- **Các thuộc tính của key trong Dictionary:**

**(b)** Key phải là immutable. Nghĩa là bạn chỉ có thể sử dụng chuỗi, số hoặc tuple làm key của Dictionary. Ví dụ:

```
dict = {'Ten': 'Hoang', 'Tuoi': 7};  
print "dict['Ten']: ", dict['Ten']
```



# KIỂU DICTIONARY

- **Truy cập các giá trị trong Dictionary trong Python:**

Khi chỉ mục không được định nghĩa với Dictionary, thì các giá trị trong Dictionary có thể được truy cập thông qua các key của chúng.

**Cú pháp:**

```
<ten_dictionary>[key]
```

**Ví dụ:**

```
data={'Id':100, 'Ten':'Thanh', 'Nghenghiep':'Developer'}
```

```
print "Id của nhan vien la: ",data['Id']
```

```
print "Ten của nhan vien la:",data['Ten']
```

# KIỂU DICTIONARY

- **Cập nhật Dictionary trong Python:**

- Item (cặp key-value) có thể được cập nhật bằng cách thêm một entry mới hoặc một cặp key-value mới, sửa đổi một entry đã tồn tại, hoặc xóa một entry đang tồn tại .

**Ví dụ:**

```
data={'Id':100, 'Ten':'Thanh', 'Nghenghiep':'Developer'}
```

```
data['Nghenghiep']='Manager'
```

```
data['Mucluong']=12000000
```

```
print data1
```

# KIỂU DICTIONARY

- **Xóa phần tử từ Dictionary trong Python:**

- Với Dictionary, bạn có thể xóa một phần tử đơn hoặc xóa toàn bộ nội dung của Dictionary đó. Bạn sử dụng lệnh `del` để thực hiện các hoạt động này.

- Cú pháp để xóa một item từ Dictionary:

```
del ten_dictionary[key]
```

Để xóa cả Dictionary, sử dụng cú pháp:

```
del ten_dictionary
```

# THẢO LUẬN NHÓM

## **NỘI DUNG:**

1. Các hàm và phương thức đã được xây dựng sẵn cho Dictionary trong Python.
2. Đặc điểm của các kiểu dữ liệu có cấu trúc.

# THẢO LUẬN NHÓM

- Từ điển là một danh sách các từ (key) và định nghĩa của nó (value)
  - Yêu cầu các key không được trùng nhau, như vậy có thể xem từ điển như một loại set

- Từ điển có thể khai báo theo cú pháp của set

```
>>> dic = {1:'one', 2:'two', 3:'three'}
```

```
>>> print(dic[1])
```

```
'one'
```

```
>>> dic[4]='four'
```

```
>>> print(dic)
```

```
{1: 'one', 2: 'two', 3: 'three', 4: 'four'}
```

# THẢO LUẬN NHÓM

- Chú ý: chỉ những loại dữ liệu immutable (không thể thay đổi) mới có thể dùng làm key của từ điển

```
dic = { (1,2,3):"abc", 3.1415:"abc" }
```

```
dic = { [1,2,3]:"abc" }           # lỗi
```

- Một số phép toán / phương thức thường dùng
  - `len(d)`: trả về độ dài của từ điển (số cặp key-value)
  - `del d[k]`: xóa key k (và value tương ứng)
  - `k in d`: trả về True nếu có key k trong từ điển
  - `k not in d`: trả về True nếu không có key k trong từ điển
  - `pop(k)`: trả về value tương ứng với k và xóa cặp này đi
  - `popitem()`: trả về (và xóa) một cặp (key, value) tùy ý

# THẢO LUẬN NHÓM

- Một số phép toán / phương thức thường dùng
  - `get(k)`: lấy về value tương ứng với key k
    - Khác phép `[]` ở chỗ `get` trả về `None` nếu `k` không phải là key
  - `update(w)`: ghép các nội dung từ từ điển `w` vào từ điển hiện tại (nếu key trùng thì lấy value từ `w`)
  - `items()`: trả về list các cặp (key, value)
  - `keys()`: trả về các key của từ điển
  - `values()`: trả về các value của từ điển
  - `pop(k)`: trả về value tương ứng với `k` và xóa cặp này đi
  - `popitem()`: trả về (và xóa) một cặp (key, value) tùy ý



# THẢO LUẬN NHÓM

- Dùng zip để ghép 2 list thành tuple

```
>>> l1 = ["a", "b", "c"]
>>> l2 = [1, 2, 3]
>>> c = zip(l1, l2)
>>> for i in c:
...     print(i)
...
('a', 1)
('b', 2)
('c', 3)
```

# BÀI TẬP

## NỘI DUNG:

1. Viết chương trình chấp nhận một chuỗi số, phân tách bằng dấu phẩy từ giao diện điều khiển, tạo ra một danh sách và một tuple chứa mọi số.

2. Viết chương trình sắp xếp tuple (name, age, score) theo thứ tự tăng dần, name là string, age và height là number. Tuple được nhập vào bởi người dùng. Tiêu chí sắp xếp là:

Sắp xếp theo name sau đó sắp xếp theo age, sau đó sắp xếp theo score. Ưu tiên là tên > tuổi > điểm.

# BÀI TẬP



## NỘI DUNG:

3. Định nghĩa một hàm có thể in dictionary chứa các key là số từ 1 đến 20 (bao gồm cả 1 và 20) và các giá trị bình phương của chúng.

4. Với số nguyên  $n$  nhất định, hãy viết chương trình để tạo ra một dictionary chứa  $(i, i*i)$  như là số nguyên từ 1 đến  $n$  (bao gồm cả 1 và  $n$ ) sau đó in ra dictionary này. Ví dụ: Giả sử số  $n$  là 8 thì đầu ra sẽ là:  $\{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64\}$ .

# BÀI TẬP

## CHUẨN BỊ CHO BUỔI HỌC TIẾP THEO:

1. Đọc các tài liệu về nội dung mục 5.1; 5.2.
2. Tìm hiểu về đặc điểm của một số loại module trong lập trình Python.

# Ngoại lệ và xử lý ngoại lệ

## Ngoại lệ là gì?

- Ngoại lệ = lỗi, đúng, nhưng không hẳn
- Thường người ta chia lỗi thành 3 nhóm
  1. **Lỗi khi viết chương trình**: hệ quả là chương trình không chạy được nếu là thông dịch (hoặc không dịch được, nếu là biên dịch)
  2. **Lỗi khi chương trình chạy**: hệ quả là phải thực hiện lại
    - Chẳng hạn như nhập liệu không đúng, thì phải nhập lại
  3. **Ngoại lệ**: vẫn là lỗi, xảy ra khi có một bất thường và khiến một chức năng không thể thực hiện được
    - Chẳng hạn như đang ghi dữ liệu ra một file, nhưng file đó lại bị một tiến trình khác xóa mất

# Ngoại lệ và xử lý ngoại lệ

## Ngoại lệ là gì?

- Ranh giới giữa ngoại lệ và lỗi khá mong manh, thậm chí khó phân biệt trong nhiều tình huống
- Cách chia lỗi thành 3 nhóm có khuynh hướng cho rằng môi trường thực thi của chương trình là thân thiện và hoàn hảo
- Python có xu hướng chia lỗi thành 2 loại
  - **Syntax error**: viết sai cú pháp, khiến chương trình thông dịch không dịch được
  - **Exception**: xảy ra bất thường không như thiết kế
    - Như vậy xử lý exception sẽ khiến chương trình ổn định và hoạt động tốt trong mọi tình huống

# Ngoại lệ và xử lý ngoại lệ

## Ngoại lệ là gì?

- Ví dụ về syntax error:

```
>>> while True print('Hello world')
File "<stdin>", line 1
    while True print('Hello world')
            ^
```

**SyntaxError**: invalid syntax

- Ví dụ về exception:

```
>>> 10 * (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
```

**ZeroDivisionError**: division by zero

- Có vẻ như syntax error cũng chỉ là một exception!!!



# Ngoại lệ và xử lý ngoại lệ

## “xử lý” ngoại lệ

```
while True:
```

Vòng lặp nhập X  
cho đến khi người dùng  
nhập vào đúng giá trị số

```
    try:
```

```
        x = int(input("Nhập số X: "))  
        break
```

Khởi nhập X  
(có thể nhập lỗi)

```
    except ValueError:
```

```
        print("Lỗi, hãy nhập lại.")
```

Xử lý khi lỗi xảy ra

```
print("X =", x)
```



# Ngoại lệ và xử lý ngoại lệ

## Cú pháp try-except-else-finally

- **Cú pháp:**  
`try:`  
`except:`  
`else:`  
`finally:`
- **Công việc của từng khối:**
  - Khối `"try"`: đoạn mã có khả năng gây lỗi, khi lỗi xảy ra, khối này sẽ bị dừng ở dòng gây lỗi
  - Khối `"except"`: đoạn mã xử lý lỗi, chỉ thực hiện nếu có lỗi xảy ra
  - Khối `"else"`: có thể xuất hiện ngay sau khối `except` cuối cùng, đoạn mã sẽ được thực hiện nếu không có `except` nào được thực hiện (đoạn `try` không có lỗi)
  - Khối `"finally"`: còn được gọi là khối `clean-up`, luôn được thực hiện dù có xảy ra lỗi hay không

# Ngoại lệ và xử lý ngoại lệ

## Cú pháp try-except-finally

### ■ Chú ý:

- Khối try chỉ có 1 khối duy nhất, phải viết đầu tiên
- Khối finally có thể có hay không, nếu có thì khối này phải viết cuối cùng
- Khối except có thể không viết, có một khối, hoặc nhiều khối except (để xử lý nhiều tình huống lỗi khác nhau)
- Một khối except có thể xử lý một loại lỗi, nhiều loại lỗi hoặc tất cả các loại lỗi
- Nếu không xử lý triệt để lỗi có thể “ném” trả lại lỗi này bằng lệnh “raise”
- Có thể phát sinh một ngoại lệ bằng lệnh “raise <lỗi>”

# Ngoại lệ và xử lý ngoại lệ

## Cú pháp try-except-finally

```
except (NameError, TypeError):    # xử lý 2 loại lỗi
    print("Name or Type error")

except IOError as e:              # lấy đối tượng lỗi, đặt tên là e
    print(e)
    raise                          # trả lại lỗi này

except ValueError:                # xử lý lỗi Value
    print("Value error")

except:                            # xử lý tất cả các lỗi còn lại
    print("An error occurred")
    raise NameError("Ko bit")     # tạo ra một lỗi "Ko bit"

else:                              # thực hiện nếu không có lỗi nào
    print("OK")
```

# Ngoại lệ và xử lý ngoại lệ

## Một số loại exception thường gặp

Exception	Miêu tả
Exception	Lớp cơ sở (base class) của tất cả các ngoại lệ
StopIteration	Được tạo khi phương thức next() của một iterator không trả về bất kỳ đối tượng nào
StandardError	Lớp cơ sở của tất cả exception có sẵn ngoại trừ StopIteration và SystemExit
ArithmeticError	Lớp cơ sở của tất cả các lỗi xảy ra cho phép tính số học
OverflowError	Được tạo khi một phép tính vượt quá giới hạn tối đa cho một kiểu số
FloatingPointError	Được tạo khi một phép tính số thực thất bại
ZeroDivisionError	Được tạo khi thực hiện phép chia cho số 0 với tất cả kiểu số
AssertionError	Được tạo trong trường hợp lệnh assert thất bại

# Ngoại lệ và xử lý ngoại lệ

## Một số loại exception thường gặp

Exception	Miêu tả
AttributeError	Được tạo trong trường hợp tham chiếu hoặc gán thuộc tính thất bại
EOFError	Được tạo khi không có input nào từ hàm <code>raw_input()</code> hoặc hàm <code>input()</code> và tới EOF (viết tắt của end of file)
ImportError	Được tạo khi một lệnh import thất bại
KeyboardInterrupt	Được tạo khi người dùng ngắt việc thực thi chương trình, thường là bởi nhấn Ctrl+c
LookupError	Lớp cơ sở cho tất cả các lỗi truy cứu
IndexError	Được tạo khi một chỉ mục không được tìm thấy trong một dãy (sequence)
KeyError	Được tạo khi key đã cho không được tìm thấy trong Dictionary
NameError	Được tạo khi một định danh không được tìm thấy trong local hoặc global namespace

# Ngoại lệ và xử lý ngoại lệ

## Một số loại exception thường gặp

Exception	Miêu tả
UnboundLocalError	Được tạo khi cố gắng truy cập một biến cục bộ từ một hàm hoặc phương thức nhưng mà không có giá trị nào đã được gán cho nó
EnvironmentError	Lớp cơ sở cho tất cả ngoại lệ mà xuất hiện ở ngoài môi trường Python
IOError	Được tạo khi hoạt động i/o thất bại, chẳng hạn như lệnh print hoặc hàm open() khi cố gắng mở một file không tồn tại
OSError	Được do các lỗi liên quan tới hệ điều hành
SyntaxError	Được tạo khi có một lỗi liên quan tới cú pháp
IndentationError	Được tạo khi độ thụt dòng code không được xác định hợp lý
SystemError	Được tạo khi trình thông dịch tìm thấy một vấn đề nội tại, nhưng khi lỗi này được bắt gặp thì trình thông dịch không thoát ra



# Ngoại lệ và xử lý ngoại lệ

## Một số loại exception thường gặp

Exception	Miêu tả
SystemExit	Được tạo khi trình thông dịch thoát ra bởi sử dụng hàm <code>sys.exit()</code> . Nếu không được xử lý trong code, sẽ làm cho trình thông dịch thoát
TypeError	Được tạo khi một hoạt động hoặc hàm sử dụng một kiểu dữ liệu không hợp lệ
ValueError	Được tạo khi hàm đã được xây dựng sẵn có các kiểu tham số hợp lệ nhưng các giá trị được xác định cho tham số đó là không hợp lệ
RuntimeError	Được tạo khi một lỗi đã được tạo ra là không trong loại nào
NotImplementedError	Được tạo khi một phương thức abstract, mà cần được triển khai trong một lớp được kế thừa, đã không được triển khai thực sự

# Ngoại lệ và xử lý ngoại lệ

## Bài tập

1. Viết chương trình nhập 2 số nguyên  $a$  và  $b$ , sau đó tính và in ra giá trị phân số  $a/b$ . Chú ý xử lý ngoại lệ trong các tình huống dưới đây:
  - Người dùng nhập  $a$  hoặc  $b$  không phải số nguyên
  - Người dùng nhập  $b = 0$
2. Viết chương trình yêu cầu người dùng nhập  $a$ ,  $b$  và  $c$  là độ dài 3 cạnh của một tam giác. Xử lý ngoại lệ trong các tình huống sau:
  - Người dùng nhập  $a$ ,  $b$  hoặc  $c$  không phải là kiểu số
  - Người dùng nhập giá trị 0 hoặc số âm cho  $a$ ,  $b$  hoặc  $c$
  - Người dùng nhập  $a$ ,  $b$ ,  $c$  dương nhưng không thể là cạnh của một tam giác