

Kỹ thuật lập trình:  
**CẤU TRÚC DỮ LIỆU TRONG PYTHON**



# NỘI DUNG

- Làm việc với dữ liệu chuỗi
  - Cấu trúc chuỗi
  - Hàm xử lý chuỗi
- Cấu trúc dữ liệu tập hợp (collection): ***List, Tuple, Set, Dictionary***

# 1. Dữ liệu chuỗi

**Chuỗi** là một *tập hợp các ký tự* được đặt trong cặp *nháy đơn* hoặc *nháy đôi*, chuỗi nhiều dòng được đặt trong cặp 3 nháy đơn hoặc đôi.

```
s1 = "Welcome to UEL"
s2 = 'I Love UEL'
s3 = "Chào mừng bạn đến với Khóa học Python!"
s4 = """Chào mừng bạn đến với Khóa học
Lập trình Python"""

print(s1)
print(s2)
print(s3)
print(s4)
```

```
Welcome to UEL
I Love UEL
Chào mừng bạn đến với Khóa học Python!
Chào mừng bạn đến với Khóa học
Lập trình Python
```

Chuỗi trong Python là một *đối tượng*, các thao tác trên chuỗi có thể được thực hiện thông qua việc gọi các phương thức xử lý theo cú pháp:

**object** . **method name** ( **parameter list** )

## 2. Hàm xử lý chuỗi

### Upper, Lower, Title

**upper:** chuyển đổi chuỗi về chữ in hoa

```
welcome = "Welcome to UEL"  
print(welcome.upper())
```

WELCOME TO UEL

**lower:** chuyển đổi chuỗi về chữ in thường

```
welcome = "Welcome to UEL"  
print(welcome.lower())
```

welcome to uel

**title:** in hoa ký tự đầu mỗi từ

```
welcome = "Welcome to UEL"  
print(welcome.title())
```

Welcome To Uel

Nhóm hàm kiểm tra: **isupper()**, **islower()**, **istitle()**

## 2. Hàm xử lý chuỗi

### »» Capitalize, Swapcase

**capitalize:** in hoa ký tự đầu tiên

```
welcome = "Welcome to UEL"  
print(welcome.capitalize())
```

Welcome to uel

**swapcase:** chuyển đổi hoa → thường, thường → hoa

```
welcome = "Welcome to UEL"  
print(welcome.swapcase())
```

wELCOME TO uel

## 2. Hàm xử lý chuỗi

»»» Căn lề: ljust, rjust, center

```
welcome = "UEL"  
print(welcome.ljust(10))  
print(welcome.ljust(10, "*"))
```

```
UEL  
UEL*****
```

```
welcome = "UEL"  
print(welcome.rjust(10))  
print(welcome.rjust(10, "*"))
```

```
          UEL  
*****UEL
```

```
welcome = "UEL"  
print(welcome.center(10))  
print(welcome.center(10, "*"))
```

```
      UEL  
***UEL***
```

## 2. Hàm xử lý chuỗi

### »» Xóa khoảng trắng thừa: strip, lstrip, rstrip

```
welcome = " Welcome to UEL "  
print(welcome.strip())
```

Welcome to UEL

```
welcome = " Welcome to UEL "  
print(welcome.lstrip())
```

Welcome to UEL

```
welcome = " Welcome to UEL "  
print(welcome.rstrip())
```

Welcome to UEL

### »» Độ dài chuỗi: len(), \_\_len\_\_()

```
welcome = "Welcome to UEL!"  
print(len(welcome)) ➔ 15  
print(welcome.__len__()) ➔ 15
```

## 2. Hàm xử lý chuỗi

### »» startswith, endswith

**startswith:** Kiểm tra chuỗi có bắt đầu bằng chuỗi con nào đó hay không. Cú pháp: **startswith(value, start, end)**

**endswith:** Kiểm tra chuỗi có kết thúc bằng chuỗi con nào đó hay không. Cú pháp: **endswith(value, start, end)**

\*Lưu ý: startswith, endswith có phân biệt chữ hoa thường

```
welcome = "Welcome to UEL!"  
print(welcome.startswith("We")) ➔ True  
print(welcome.startswith("We", 3)) ➔ False  
print(welcome.startswith("We", 2, 9)) ➔ False  
print(welcome.endswith("!")) ➔ True  
print(welcome.endswith("L!", -2)) ➔ True
```



## 2. Hàm xử lý chuỗi

### »»» Tìm chuỗi con: **find**, **rfind**

**find**: trả về vị trí đầu tiên tìm thấy chuỗi con, **rfind**: trả về vị trí cuối cùng tìm thấy chuỗi con. Trả về -1 nếu không tìm thấy.

Cú pháp: **find(str, start, end)**, **rfind(str, start, end)**

```
slogan = "No Pain No Gain!"  
print(slogan.find("No")) → 0  
print(slogan.find("No", 2, 6)) → -1  
print(slogan.find("No", 2)) → 8  
print(slogan.rfind("No")) → 8  
print(slogan.rfind("No", -9, -2)) → 8
```

## 2. Hàm xử lý chuỗi

### »»» Tìm chuỗi con: `index`, `rindex`

**`index`**, **`rindex`**: tương tự như **`find`**, **`rfind`**. Tuy nhiên, nếu không tìm thấy chuỗi con sẽ trả về `exception`.

```
slogan = "No Pain No Gain!"  
print(slogan.index("No", 2))  
print(slogan.rindex("No", 5))  
print(slogan.index("Success"))
```

```
8  
8  
Traceback (most recent call last):  
  File "/Volumes/Data/Works/1.UEL/Python/Demo/StringExample/main.py", line 4, in <module>  
    print(welcome.index("Success"))  
ValueError: substring not found
```

## 2. Hàm xử lý chuỗi

### »» Đếm chuỗi con: `count`

**count**: đếm số lần xuất hiện của chuỗi con trong chuỗi gốc, không tìm thấy chuỗi con → trả về 0.

Cú pháp: **count(str, start, end)**

```
slogan = "No Pain No Gain!"  
print(slogan.count("No"))      → 2  
print(slogan.count("No", 3))   → 1  
print(slogan.count("Success")) → 0
```

## 2. Hàm xử lý chuỗi

### »» Cắt chuỗi: substring

```
slogan = "No Pain No Gain!"
```

```
print(slogan[3:7])      Pain
```

```
print(slogan[:7])       No Pain
```

```
print(slogan[None:7])   No Pain
```

```
print(slogan[3:])       Pain No Gain!
```

```
print(slogan[3:None])   Pain No Gain!
```

```
print(slogan[-5:])      Gain!
```

```
print(slogan[:-9])      No Pain
```

## 2. Hàm xử lý chuỗi

### »» Tách chuỗi: split

```
slogan = "No-Pain-No-Gain!"  
s = slogan.split('-')  
print(s)  
for i in s:  
    print(i)
```

```
['No', 'Pain', 'No', 'Gain!']  
No  
Pain  
No  
Gain!
```

## 2. Hàm xử lý chuỗi

### »»» Nối chuỗi: join

```
slogan = "No-Pain-No-Gain!"  
s = slogan.split('-')  
print(s)  
for i in s:  
    print(i, end=' ')  
print('\n-----')  
s1 = ''  
print("Joined string: ", s1.join(s))
```

```
['No', 'Pain', 'No', 'Gain!']
```

```
No Pain No Gain!
```

```
-----
```

```
Joined string:  No Pain No Gain!
```

## 2. Hàm xử lý chuỗi

### »» Thay thế chuỗi: `replace`

**replace**: tìm kiếm và thay thế chuỗi con bằng chuỗi mới.

Cú pháp: **replace(oldStr, newStr, max)**

```
slogan = "No Pain No Gain!"  
print(slogan.replace("No", "o"))  
print(slogan.replace("No", "o", 1))
```

0 Pain 0 Gain!

0 Pain No Gain!

## 2. Hàm xử lý chuỗi

»» Chuyển đổi chuỗi: `maketrans()`, `translate()`

```
slogan = "No Pain No Gain!"  
inputs = "oa"  
outputs = "OA"  
trans = slogan.maketrans(inputs, outputs)  
print(trans)  
print(slogan.translate(trans))
```

```
{111: 79, 97: 65}  
NO PAin NO GAin!
```



## 2. Hàm xử lý chuỗi

»» Các hàm kiểm tra: `isalnum()`, `isalpha()`, `isdigit()`, `isnumeric()`

```
print('Coin68'.isalnum())    True
print('Coin68.com'.isalnum()) False
print('Binance'.isalpha())   True
print('68'.isdigit())         True
print('68.5'.isdigit())       False
print('68abc'.isdigit())      False
print("⑩⑬⑤⑩".isdigit())     False
print('86'.isnumeric())       True
print("⑩⑬⑤⑩".isnumeric())   True
```

### 3. Kiểu dữ liệu tập hợp (collection)

#### ❖ List

##### »» Khái niệm

**List:** là một collection dùng *lưu trữ các phần tử theo thứ tự, các phần tử có thể thuộc nhiều kiểu dữ liệu khác nhau.*

Ví dụ:

```
list_1 = []  
list_2 = ['Python', 'Golang', 'Ruby']  
list_3 = ['Java', 1, 'Swift', 2]  
list_4 = [['React native', 'Ionic', 'Flutter'], [1, 2, 3]]
```

### 3. Kiểu dữ liệu tập hợp (collection)

#### ❖ List

#### »»» Các thao tác trên List

##### Khởi tạo list

<code>list_1 = [1, 2, 3]</code>	<code>[1, 2, 3]</code>
<code>list_2 = list([1, 2, 3])</code> <small>Sử dụng constructor</small>	<code>[1, 2, 3]</code>
<code>list_3 = [i for i in range(4)]</code>	<code>[0, 1, 2, 3]</code>
<code>list_4 = [i for i in range(4) if i % 2 == 0]</code>	<code>[0, 2]</code>
<code>list_5 = [6]*3</code>	<code>[6, 6, 6]</code>

### 3. Kiểu dữ liệu tập hợp (collection)

#### ❖ List

##### »» Các thao tác trên List

##### Truy xuất phần tử của List

```
print('1. Truy xuất phần tử của List theo chỉ mục')
my_list = [1, 2, 3, 4]
print(my_list[2])      3
print('2. Duyệt list theo tập hợp')
for i in my_list:
    print(i, end=' ')  1 2 3 4
print('\n3. Duyệt list theo chỉ mục')
for i in range(len(my_list)):
    print(my_list[i], end=' ')  1 2 3 4
```

### 3. Kiểu dữ liệu tập hợp (collection)

#### ❖ List

##### »» Các thao tác trên List

Truy xuất phần tử của List

```
my_list = ['Python', 'Perl', 'Ruby', 'Golang', 'C#',  
'Java', 'Swift']
```

```
print(my_list[:2])
```

```
print(my_list[None:1])
```

```
print(my_list[3:6])
```

```
print(my_list[5:])
```

```
print(my_list[4:None])
```

```
print(my_list[-2:])
```

```
['Python', 'Perl']
```

```
['Python']
```

```
['Golang', 'C#', 'Java']
```

```
['Java', 'Swift']
```

```
['C#', 'Java', 'Swift']
```

```
['Java', 'Swift']
```

### 3. Kiểu dữ liệu tập hợp (collection)

#### ❖ List

##### »» Các thao tác trên List

Cập nhật giá trị phần tử

```
list_1 = [1, 2, 3, 4]
list_2 = ["Python", "Angular", "Ruby"]
print(list_1) [1, 2, 3, 4]
print(list_2) ['Python', 'Angular', 'Ruby']
list_1[0] = 9
list_2[1] = "Golang"
print(list_1) [9, 2, 3, 4]
print(list_2) ['Python', 'Golang', 'Ruby']
```

### 3. Kiểu dữ liệu tập hợp (collection)

#### ❖ List

##### »» Các thao tác trên List

Cập nhật giá trị phần tử

```
list_1 = [1, 2, 3, 4]
list_2 = list_1
print(list_1)    [1, 2, 3, 4]
print(list_2)    [1, 2, 3, 4]
list_1[0] = 9
list_1[1] = 8
print(list_1)    [9, 8, 3, 4]
print(list_2)    [9, 8, 3, 4] ?
```

list\_1 →

1	2	3	4
0	1	2	3

list\_2 →

1	2	3	4
0	1	2	3

### 3. Kiểu dữ liệu tập hợp (collection)

#### ❖ List

##### »» Các thao tác trên List

Cập nhật giá trị phần tử

```
list_1 = [1, 2, 3, 4]
```

```
list_2 = list(list_1)
```

```
print(list_1) [1, 2, 3, 4]
```

```
print(list_2) [1, 2, 3, 4]
```

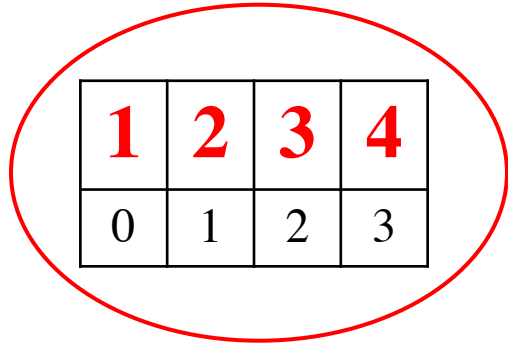
```
list_1[0] = 9
```

```
list_1[1] = 8
```

```
print(list_1) [9, 8, 3, 4]
```

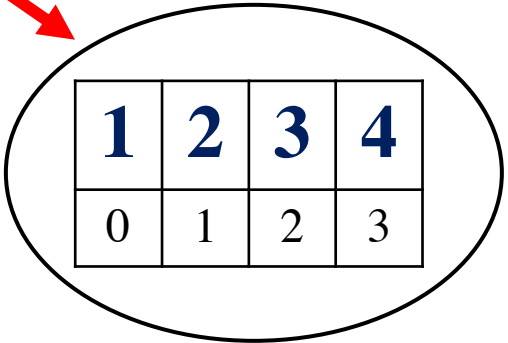
```
print(list_2) [1, 2, 3, 4] ?
```

list\_1



1	2	3	4
0	1	2	3

list\_2



1	2	3	4
0	1	2	3



### 3. Kiểu dữ liệu tập hợp (collection)

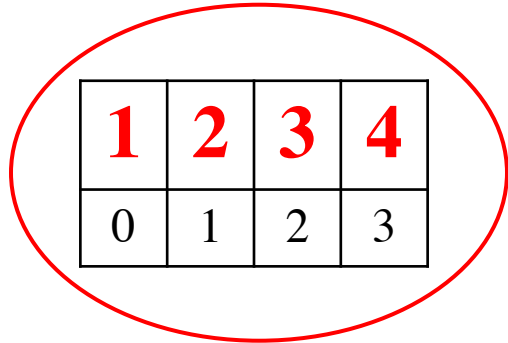
#### ❖ List

##### »» Các thao tác trên List

Cập nhật giá trị phần tử

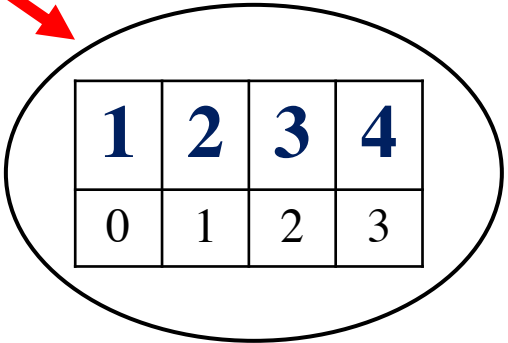
```
list_1 = [1, 2, 3, 4]
list_2 = list_1.copy()
print(list_1)      [1, 2, 3, 4]
print(list_2)      [1, 2, 3, 4]
list_1[0] = 9
list_1[1] = 8
print(list_1)      [9, 8, 3, 4]
print(list_2)      [1, 2, 3, 4] ?
```

list\_1 →



1	2	3	4
0	1	2	3

list\_2 →



1	2	3	4
0	1	2	3

### 3. Kiểu dữ liệu tập hợp (collection)

#### ❖ List

##### »» Các thao tác trên List

Thêm phần tử vào List:

✓ `insert(index, value)`

```
my_list = ["Python", "Angular", "Ruby"]  
my_list.insert(1, "Perl")  
print(my_list) ['Python', 'Perl', 'Angular', 'Ruby']
```

✓ `append(value)`

```
my_list = ["Python", "Ruby"]  
my_list.append(["Perl", "Golang"])  
print(my_list) ['Python', 'Ruby', ['Perl', 'Golang']]
```

### 3. Kiểu dữ liệu tập hợp (collection)

#### ❖ List

##### »» Các thao tác trên List

Thêm phần tử vào List:

✓ `extend(value)`

```
my_list = ["Python", "Ruby"]  
my_list.extend(["Perl", "Golang"])  
print(my_list) ['Python', 'Ruby', 'Perl', 'Golang']
```

✓ `__add__(value)`

```
my_list = ["Python", "Angular", "Ruby"]  
print(my_list.__add__(["Perl"]))  
['Python', 'Angular', 'Ruby', 'Perl']
```

### 3. Kiểu dữ liệu tập hợp (collection)

#### ❖ List

##### »» Các thao tác trên List

**Xóa phần tử khỏi List:**

✓ **pop(index)**

```
my_list = ["Python", "Angular", "Ruby", "Golang"]
my_list.pop()
print(my_list)    ['Python', 'Angular', 'Ruby']
my_list.pop(1)
print(my_list)    ['Python', 'Ruby']
```

✓ **remove(value):** xóa phần tử tìm thấy đầu tiên trong List

```
my_list = ["Python", "Angular", "Ruby", "Angular", "Golang"]
my_list.remove('Angular') # del my_list[1]
print(my_list)    ['Python', 'Ruby', 'Angular', 'Golang']
```

### 3. Kiểu dữ liệu tập hợp (collection)

#### ❖ List

##### »» Các thao tác trên List

Đếm phần tử trên List

```
my_list = [3, 9, 7, 9, 5, 9, 8]  
print(my_list.count(9)) 3
```

Kiểm tra sự tồn tại của phần tử trên List

```
my_list = [3, 9, 7, 9, 5, 9, 8]  
print(my_list.__contains__(2)) False  
print(my_list.__contains__(7)) True
```

```
my_list = [3, 9, 7]  
print(3 in my_list) True  
print(8 in my_list) False
```

### 3. Kiểu dữ liệu tập hợp (collection)

#### ❖ List

##### »» Các thao tác trên List

##### Sắp xếp List

```
my_list = [3, 2, 7, 5, 9, 8]
my_list.sort()
print(my_list)      [2, 3, 5, 7, 8, 9]
my_list_2 = [4, 2, 5, 7, 3, 6]
sorted_list = sorted(my_list_2)
print(sorted_list)  [2, 3, 4, 5, 6, 7]
```

### 3. Kiểu dữ liệu tập hợp (collection)

#### ❖ List

##### »» Các thao tác trên List

##### Đảo ngược List

```
my_list = [3, 2, 7, 5, 9, 8]
my_list.sort()
my_list.reverse()
print(my_list)      [9, 8, 7, 5, 3, 2]
my_list_2 = [4, 2, 5, 7, 3, 6]
my_list_2.sort(reverse=True)
print(my_list_2)    [7, 6, 5, 4, 3, 2]
```

```
my_list = ['Python', 'Perl', 'Ruby', 'Golang', 'C#', 'Java', 'Swift']
print(my_list[::-1])
['Swift', 'Java', 'C#', 'Golang', 'Ruby', 'Perl', 'Python']
```

### 3. Kiểu dữ liệu tập hợp (collection)

#### ❖ List

##### »» Các thao tác trên List

Chuyển đổi string  $\leftrightarrow$  list

```
s1 = 'madam'
print(f"s1: {s1}")    s1: madam
s_list = list(s1)
print(s_list)         ['m', 'a', 'd', 'a', 'm']
s2 = ''.join(s_list)
print("s2: {0}".format(s2))    s2: madam
```



### 3. Kiểu dữ liệu tập hợp (collection)

#### ❖ List

##### »» Các thao tác trên List

Tạo list đa chiều (ma trận)

```
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]  
print(matrix) [[1, 2, 3], [4, 5, 6], [7, 8, 9]]  
print(matrix[0]) [1, 2, 3]  
print(matrix[1]) [4, 5, 6]  
print(matrix[2]) [7, 8, 9]
```

```
row = 3  
col = 4  
matrix = [[0]*col]*row  
print(matrix)  
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```

### 3. Kiểu dữ liệu tập hợp (collection)

#### ❖ Tuple

##### »» Khái niệm

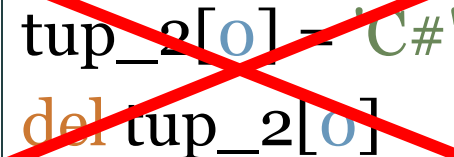
**Tuple:** là một collection tương tự như List. Tuy nhiên, *giá trị của các phần tử không được phép thay đổi.*

Ví dụ:

```
tup_1 = ()  
tup_2 = ('Python', 'Golang', 'Ruby')  
tup_3 = ('Java', 1, 'Swift', 2)  
tup_4 = (['React native', 'Ionic', 'Flutter'], [1, 2, 3])
```

##### »» Các thao tác trên Tuple

Tương tự như trên List. Lưu ý, *không thể cập nhật giá trị hay xóa phần tử.*



```
tup_2[0] = 'C#'  
del tup_2[0]
```

### 3. Kiểu dữ liệu tập hợp (collection)

#### ❖ Set

**Set:** là một collection không có thứ tự, không có chỉ mục, loại bỏ các giá trị trùng lặp. Các phần tử có thể là số, chuỗi, tuple nhưng không thể chứa list.

Ví dụ:

```
set_1 = {1, 6, 2, 8, 3}
```

```
print(set_1) {1, 2, 3, 6, 8}
```

```
set_2 = {1, 6, 2, 6, 8, 2, 8}
```

```
print(set_2) {8, 1, 2, 6}
```

```
set_3 = {1, 5, 'Python', 2, 9, (1, "Golang")}
```

```
print(set_3)
```

```
{1, 2, 5, 9, (1, 'Golang'), 'Python'}
```

### 3. Kiểu dữ liệu tập hợp (collection)

#### ❖ Set

Thêm phần tử

```
set_1 = {6, 5, 8, 3}
set_1.add(4)
print(set_1)    {3, 4, 5, 6, 8}
```

Xóa phần tử

```
set_1 = {6, 5, 8, 3, 7, 19}
set_1.pop()    {19, 5, 6, 7, 8}
print(set_1)
set_1.discard(8)
print(set_1)    {19, 5, 6, 7}
set_1.remove(5)
print(set_1)    {19, 6, 7}
```

### 3. Kiểu dữ liệu tập hợp (collection)

#### ❖ Set

Toán tử: -, |, &, ^

```
set_1 = {3, 9, 7}
```

```
set_2 = {6, 3, 8}
```

```
print(set_1 - set_2) {9, 7}
```

```
print(set_1 | set_2) {3, 6, 7, 8, 9}
```

```
print(set_1 & set_2) {3}
```

```
print(set_1 ^ set_2) {6, 7, 8, 9}
```

### 3. Kiểu dữ liệu tập hợp (collection)

#### ❖ Dictionary

**Dictionary:** là một collection *không có thứ tự*, mỗi phần tử được định nghĩa bởi cặp *khóa* và *giá trị* (key: value).

Ví dụ:

```
dict1 = {}  
print(dict1) {}  
dict2 = {1: 'Python', 2: 'Ruby'}  
print(dict2) {1: 'Python', 2: 'Ruby'}  
dict3 = {'name': 'Huynh Giao', 'age': 20}  
print(dict3) {'name': 'Huynh Giao', 'age': 20}  
dict4 = dict({'id': 1, 'productName': 'Iphone'})  
print(dict4) {'id': 1, 'productName': 'Iphone'}  
dict5 = dict([(1, 'Python'), (2, 'Ruby')])  
print(dict5) {1: 'Python', 2: 'Ruby'}
```

## 4. Dictionary

### »» Các thao tác trên Dictionary

#### Khởi tạo dictionary với Comprehension

```
d1 = {x: x**2 for x in range(5)}
print(d1) {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
d2 = {y: y+1 for y in range(10) if y % 2 == 0}
print(d2) {0: 1, 2: 3, 4: 5, 6: 7, 8: 9}
```

#### Duyệt các phần tử

```
d = {1: 'Python', 'pythonApp': 'Game, Web',
      2: "Java", "javaApp": "Web, Mobile"}
for i in d:
    print(d[i])
```

```
Python
Game, Web
Java
Web, Mobile
```

### 3. Kiểu dữ liệu tập hợp (collection)

#### ❖ Dictionary

Truy xuất phần tử

```
dict1 = {1: 'Python', 2: 'Ruby', 3: 'Perl', 4: 'Golang'}  
print(dict1[1]) Python  
print(dict1[3]) Perl
```

Cập nhật giá trị phần tử

```
dict1 = {1: 'Python', 2: 'Ruby', 3: 'Perl', 4: 'Golang'}  
dict1[2] = 'Java'  
dict1[4] = 'C#'  
print(dict1)  
{1: 'Python', 2: 'Java', 3: 'Perl', 4: 'C#'}
```



### 3. Kiểu dữ liệu tập hợp (collection)

#### ❖ Dictionary

Thêm phần tử

```
dict1 = {1: 'Python', 2: 'Ruby', 3: 'Perl'}
```

```
dict1[5] = 'Java'
```

```
print(dict1)
```

```
{1: 'Python', 2: 'Ruby', 3: 'Perl', 5: 'Java'}
```

### 3. Kiểu dữ liệu tập hợp (collection)

#### ❖ Dictionary

##### Xóa phần tử

```
dict1 = {1: 'Python', 2: 'Ruby', 3: 'Perl', 4: 'Golang'}  
print(dict1.pop(3)) Perl  
print(dict1) {1: 'Python', 2: 'Ruby', 4: 'Golang'}  
del dict1[2]  
print(dict1) {1: 'Python', 4: 'Golang'}  
print(dict1.popitem()) (4, 'Golang')  
print(dict1) {1: 'Python'}  
dict1.clear()  
print(dict1) {}  
# del dict1  
# print(dict1) --> Error
```

**Q & A**