TIN HỌC CƠ SỞ

Phần 2: Ngôn ngữ lập trình Python

CHƯƠNG 11.

KIỂU DỮ LIỆU LIST, TUPLE, SET, DICTIONARY

Tin cơ sở (LT): 010100229802 - DHKL16A1HN, - DHKL16A2HN

GV. Cao Diệp Thắng

Mục tiêu chương

Nắm vững:

- Khái niệm, cú pháp khai báo, khởi tạo các kiểu dữ liệu List, Tuple, Set, Dictionary
- Cách viết code với các kiểu dữ liệu List, Tuple, Set, Dictionary

Nội dung

1. Kiểu dữ liệu tuần tự (sequential data type)
2. List (danh sách) trong Python
3. Tuple (hàng)
4. Set (tập hợp)
5. Dictionary (từ điển)

TIN HỌC CƠ SỞ

Phần 2: Ngôn ngữ lập trình Python

CHƯƠNG 11.

KIỂU DỮ LIỆU LIST, TUPLE, SET, DICTIONARY

Tin cơ sở (LT): 010100229802 - DHKL16A1HN, - DHKL16A2HN

GV. Cao Diệp Thắng

Mục tiêu chương

Nắm vững:

- Khái niệm, cú pháp khai báo, khởi tạo các kiểu dữ liệu List, Tuple, Set, Dictionary
- Cách viết code với các kiểu dữ liệu List, Tuple, Set, Dictionary

Nội dung

1. Kiểu dữ liệu tuần tự (sequential data type)
2. List (danh sách) trong Python
3. Tuple (hàng)
4. Set (tập hợp)
5. Dictionary (từ điển)

11.4. TUPLE (HÀNG)

11.4.1. Khái niệm

- Tuple trong Python là một dạng dữ liệu mà các phần tử trong đó *được xắp xếp theo thứ tự* và **không thể thay đổi được giá trị** sau khi được khai báo.[5,10]
- Giá trị lưu giữ trong tuple có thuộc tính bất biến, có nghĩa là lập trình viên không thể thay đổi hoặc xóa phần tử sau khi tạo tuple.
- Các phần tử trong tuple được đặt giữa cặp dấu **()** và được phân tách với nhau bằng dấu “**,**” hiểu một cách đơn giản hơn, có thể định nghĩa Tuple trong Python là kiểu dữ liệu có phần tử được đặt giữa cặp dấu **()** và không khả năng thay đổi sau khi tạo ra.

Nhận xét:

- Tuple như là 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**
- Tuple khai báo trực tiếp bằng cách liệt kê các phần tử con đặt trong cặp ngoặc tròn **()**, ngăn cách bởi phẩy

Ví dụ 11.35.

`(1, 2, 3, 4, 5)`
`('a', 'b', 'c', 'd')`
`(1, 'one', [2, 'two'])`
`(1,)`
`()`
	`# tuple 5 số nguyên`
	`# tuple 4 chuỗi`
	`# tuple hỗ hợp`
	`# tuple 1 phần tử`
	`# tuple rỗng`

`t0 = 'a', 'b', 'c'`	`# tuple 3 chuỗi`
`t1 = 1, 2, 3, 4, 5`	`# tuple 5 số nguyên`
`t2 = ('a', 'b', 'c')`	`# tuple 3 chuỗi`
`t3 = (1, 'one', [2,'two'])`	`# tuple hỗn hợp`
`t4 = 1, 'one', [2, 'two']`	`# tuple hỗn hợp`
`t5 = ()`	`# tuple rỗng`
`t6 = 'a',`	`# tuple một phần tử`
`t7 = ('a')`	`# KHÔNG phải tuple (là chuỗi)`
`t8 = ('a',)`	`# tuple một phần tử`
`t9 = ('a', )`	`# tuple, không đúng chuẩn python`

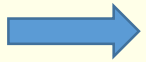
11.4.2. Tạo tuple

Cú pháp: `tên_tuple = (phần tử 1, phần tử 2,…)`

Có thể không dùng cặp đóng mở ngoặc "()" khi tạo tuple

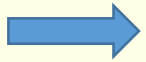
Ví dụ 11.36

```
>>> tup1 = (12,24,21,25,36,32)
>>> print(tup1)
>>> print(type(tup1))
```



```
(12, 24, 21, 25, 36, 32)
<class 'tuple'>
```

```
>>> tup2 = 16,24,23,35,27,21
>>> print(tup2)
>>> print(type(tup2))
```



```
(16,24,23,35,27,21)
<class 'tuple'>
```

```
>>> tup3=(24,)
>>> print(tup3)
>>> print(type(tup3))
```

```
(24,)
<class 'tuple'>
```

11.4.3. Truy xuất phần tử trong tuple

Cú pháp: **tên_tuple[index]**, với index từ 0…chiều dài -1

tên_tuple[begin:end]

*Trong đó **begin>=0, end < chiều dài -1**: Kết quả là các phần tử có index từ begin đến end-1*

Ví dụ 11.37

```
>>> tup1=('one', 'two', 'three', 'four', 'five')
>>> print(tup1)
('one', 'two', 'three', 'four', 'five')
>>>
>>> print(tup1[2])
three
>>>
>>> print(tup1[1:3])
('two', 'three')
>>> print(tup1[2:4])
('three', 'four')
```

Lưu ý:

- *tuple không thay đổi, do đó lập trình viên không thể cập nhật/xóa phần tử trong tuple*
- *Chỉ có thể xóa bỏ cả tuple:*

11.4.4. Xóa bỏ cả tuple:

Cú pháp

```
del(tên_tuple)
```

Ví dụ 11.38

```
>>> tup2=(1,2,3,4,5)
>>> print(tup2)
(1, 2, 3, 4, 5)
>>> del(tup2)
>>> print(tup2)
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    print(tup2)
NameError: name 'tup2' is not defined
```

11.4.5. Các phương thức cơ bản của tuple

Tương tự như các phương thức cơ bản của list nhưng **không có** các phương thức:

`sort, reverse, remove, pop, insert, extend, append`

a. Copy các phần tử của list vào tuple

Cú pháp: **`tên_tuple=tuple(tên_list)`**

Ví dụ 11.39

```
>>> all_list = ['one', 'two', 'three', 'abc', 1,2,3]
>>> tuple_copy = tuple(all_list)
>>> print(tuple_copy)
('one', 'two', 'three', 'abc', 1, 2, 3)
```

- Tuple có thể tạo bằng constructor hoặc generator (bộ sinh) – một cách viết tương tự **list comprehension**
- Tuple hỗ trợ 3 phép toán: +, *, in
- Tuple cho phép sử dụng chỉ mục và cắt

b. 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

Lưu ý: Bộ sinh của tuple chỉ dùng được 01 lần

Ví dụ 11.40

```
#t0 viết giống như bộ suy diễn danh sách/list comprehension
>>> t0 = (c for c in 'Hello!')
>>> t1 = tuple(t0)  #tạo t1 từ t0
>>> t2 = tuple(t0)  #tạo t2 từ t0
>>> print(t0)
 <generator object <genexpr> at 0x0000024BB66677D8>
>>> print(t1)
('H', 'e', 'l', 'l', 'o', '!')
>>> print(t2)
()
```

Như vậy t0 chỉ dùng được một lần

Khác biệt giữa Tuple và List là tuple chiếm ít bộ nhớ hơn và nhanh hơn List.

11.5. SET (TẬP HỢP)

11.5.1. Khái niệm

Tập hợp (set) là kiểu dữ liệu đặc sắc trong Python, Set trong python được tạo ra để xử lý tập hợp mà chúng ta được học trong toán học. Python dùng set để so sánh và tìm hiểu quan hệ giữa các tập hợp, thực hiện các phép toán đặc trưng của tập hợp như **giao, hiệu, phần bù** v.v..[5,6,7,10,11]

- Các đối tượng con đôi một khác nhau: nếu đưa các đối tượng giống nhau vào tập hợp, Python sẽ chỉ giữ lại một đối tượng.
- Không có tính thứ tự: không thể truy cập đến phần tử con thông qua hệ thống chỉ mục hay nói khác đi: các phần tử trong set không theo thứ tự thêm vào, không sử dụng index
- Không phải dữ liệu nào cũng đưa được vào tập hợp: dữ liệu con bắt buộc phải là dạng bất biến (immutable)
- Python sử dụng cấu trúc dữ liệu bảng băm (hashtable) cho tập hợp, đây chính là lý do dữ liệu con phải bất biến để tránh việc dữ liệu bị thay đổi một cách không lường được.

Khởi tạo set

Tương tự như danh sách và hàng, khởi tạo tập hợp đơn giản nhất bằng cách liệt kê các phần tử con:

Cú pháp: `tên_set = {value1, value1, value3,…}`

```
>>> set_nums = {1,5,8,3,9}
>>> print(set_nums)
{1, 3, 5, 8, 9}
>>> type(set_nums)
<class 'set'>
>>>
>>> set_string = {"Kim", "Moc", "Thuy", "Hoa", "Tho"}
>>> print(set_string)
{'Tho', 'Kim', 'Thuy', 'Moc', 'Hoa'}
>>> basket = {'apple', 'orange', 'apple', 'pear'}
>>> print(basket)
{'orange', 'pear', 'apple'} # xóa phần tử trùng nhau
```

Tạo set

Khi khởi tạo set ban đầu, chưa có phần tử,

Cú pháp: **tên_set = set()**

Ví dụ 11.42

```
>>> set_fruits = set()
>>> print(type(set_fruits))
<class 'set'>
```

Ví dụ 11.42

```
s1 = set([1, 2, 3, 4])          # {1, 2, 3, 4} – copy từ list
s2 = set((1, 1, 1))             # {1} – copy từ tuple, bỏ lặp
s3 = s1 – s2                    # {2, 3, 4} – hiệu của hai tập
s4 = set(range(1, 100))         # {1, 2, 3,…, 98, 99}
s5 = set()                      # {} – tập rỗng
```

11.5.3. Cập nhật giá trị/thêm mới phần tử vào set

Cú pháp: **`tên_set.add(element)`**

Ví dụ 11.41.

```
>>> set_fruits = set()
>>> set_fruits.add("orange")
>>> set_fruits.add("lemon")
>>> print(set_fruits)
{'orange', 'lemon'}
```

11.5.4. Xóa phần tử trong set

Sử dụng cả phương thức **discard**() và **remove**() đều có thể xóa phần tử trong set. Tuy nhiên, dùng **discard**() thì nếu không có phần tử cần xóa thì không xóa và cũng không thông báo lỗi còn nếu dùng **remove**() thì nếu không tìm thấy phần tử cần xóa sẽ báo lỗi.

Cú pháp:

```
tên_set.discard("value")

tên_set.remove("value")
```

Ví dụ 11.42

```
>>> set_string = {"Kim", "Moc", "Thuy", "Hoa", "Tho"}
>>> print("Before:", set_string)
Before: {'Kim', 'Moc', 'Hoa', 'Tho', 'Thuy'}
>>>
>>> set_string.discard("Thuy")
>>> print("After:", set_string)
After: {'Kim', 'Moc', 'Hoa', 'Tho'}
>>> set_string.remove("Thuy")
Traceback (most recent call last):
File "<pyshell#14>", line 1, in <module>
set_string.remove("Thuy")
KeyError: 'Thuy
```

11.5.5. Xóa toàn bộ các element trong set

Cú pháp:

```
tên_set.clear()
```

Ví dụ 11.43

```
>>> set_string = {"Kim", "Moc", "Thuy", "Hoa", "Tho"}
>>> set_string.clear()
>>> print(set_string)
set()
```

11.5.6. Xóa set

Cú pháp: `del(tên_set)`

Ví dụ 11.44

```
>>> set_string = {"Kim", "Moc", "Thuy", "Hoa", "Tho"}
>>> del(set_string)
>>> print(set_string)
Traceback (most recent call last):
File "<pyshell#21>", line 1, in <module>
print(set_string)
NameError: name 'set string' is not defined
```

11.5.7. Lấy phần tử (element) ra khỏi set

Kết quả trả về là là element được lấy ra, báo lỗi nếu set không có element nào

Cú pháp:

```
tên_set.pop()
```

Ví dụ 11.45

```
>>> set_fruits = {"apple", "pear", "orange", "grape"}
>>> print(set_fruits)
{'pear', 'grape', 'apple', 'orange'}
>>>
>>> fruit = set_fruits.pop()
>>> print(fruit)
pear
>>> print(set_fruits)
{'grape', 'apple', 'orange'}
```

Các phương thức cơ bản

a. Chiều dài, max, min, sum của set

Cú pháp:

len(tên_set)/max(tên_set)/min(tên_st)

Ví dụ 11.46

```
set_dest = {1, 2, 3, 4, 5}
>>> print(set_dest)
{1, 2, 3, 4, 5}
>>> print("len: ", len(set_dest))
len: 5
>>> print("max: ", max(set_dest))
max: 5
>>> print("min: ", min(set_dest))
min: 1
>>> print("sun: ", sum(set_dest))
sum: 15
```

11.5.8. Tạo bản sao của set

Cú pháp:

```
tên_set_đích = tên_set_nguồn.copy()
```

Ví dụ 11.47

```
>>> set_source = {1,4,3,2,5}
>>> set_dest = set_source.copy()
>>> print("set dest:", set_dest)
set dest: {1, 2, 3, 4, 5}
```

11.5.9. Duyệt set:

Tương tự như duyệt list

Ví dụ 11.48

```
>>> set_nums = {1,4,3,2,5}
>>> print(set_nums)
{1, 2, 3, 4, 5}
>>> for num in set_nums:
print("Num:", num,',', end = ' ')
Num: 1 , Num: 2 , Num: 3 , Num: 4 , Num: 5 ,
```

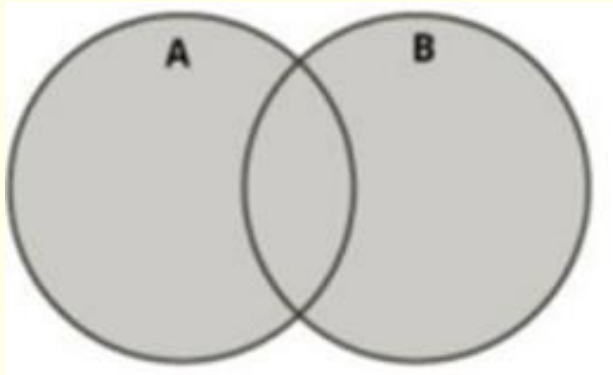
11.5.10. Set Union (hợp 2 tập hợp)

Trả về tất cả các element không trùng nhau của các set

Cú pháp: `set_union = set1|set2|...`

Ví dụ 11.49

```
>>> setA = {1,2,6,3,7}
>>> setB = {1,4,5,8,9}
>>> setC = setA | setB
>>> print(setC)
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```



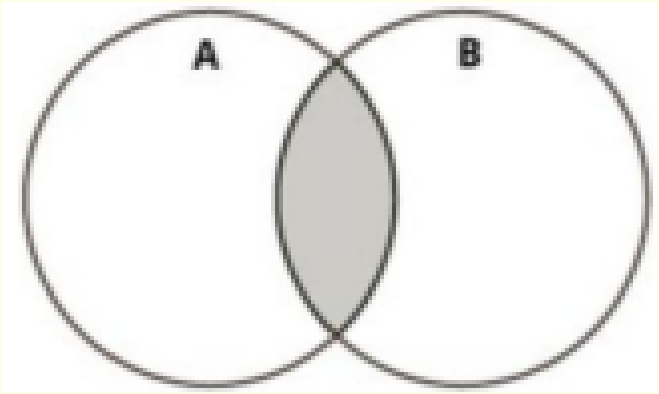
11.5.11. Set Intersection (giao 2 tập hợp)

Trả về các element cùng xuất hiện trong các set

Cú pháp: `set_intersection = set1 & set2 &...`

Ví dụ 11.50

```
>>> setA = {1,2,6,3,7}
>>> setB = {1,4,5,8,9}
>>> setD = setA & setB
>>> print(setD)
{1}
```



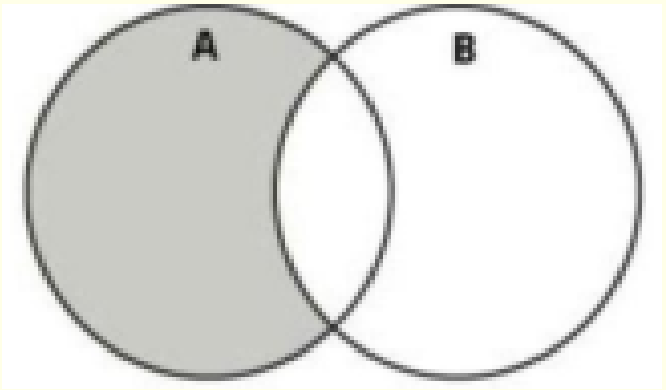
11.5.12. Set Diference (hiệu 2 tập hợp)

Trả về các phần tử (element) chỉ có trong tập hợp (set) này mà không có trong set (tập hợp) kia.

Cú pháp: `set_difference = set1-set2`

Ví dụ 11.51

```
>>> setA = {1,2,6,3,7}
>>> setB = {1,4,5,8,9}
>>> setE = setA - setB
>>> print(setE)
{2, 3, 6, 7}
```



11.5.13. Set Symmetric Difference

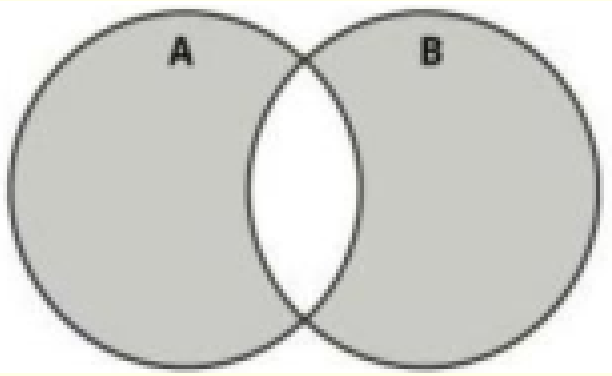
Trả về các phần tử(element) không cùng xuất hiện trong các set

Cú pháp:

```
set_ Symmetric_Difference = set1 ^ set2
```

Ví dụ 11.52

```
>>> setA = {1,2,6,3,7}
>>> setB = {1,4,5,8,9}
>>> setF = setA ^ setB
>>> print(setF)
{2, 3, 4, 5, 6, 7, 8, 9}
```



11.5.14. Sắp xếp tăng dần/giảm dần

Cú pháp sắp xếp tăng: **`sorted(tên_set)`**

Cú pháp sắp xếp giảm: **`sorted(tên_set, reverse = True)`**

Ví dụ 11.53

```
>>>pySet = {'e', 'a', 'u', 'o', 'i'}
>>>print("Set: ", pySet)
Set: {'u', 'e', 'o', 'i', 'a'}
>>>print("Sorted set: ", sorted(pySet))
Sorted set: ['a', 'e', 'i', 'o', 'u']
>>>
>>>print("Sorted reversed set: ",
sorted(pySet, reverse=True))
Sorted reversed set: ['u', 'o', 'i', 'e', 'a']
```

Trong ví dụ 11.34 ở trên, chúng ta sử dụng các vòng lặp for lồng nhau để lặp qua từng hàng và từng cột. Tại mỗi điểm, chúng ta đặt phần tử **X[i][j]** vào trong phần tử **mt_CV**[j][i].

,