Softech®
Gather to thrive together

Aptech™
COMPUTER EDUCATION
Unleash your potential

BÀI 06

Dictionaries và Functions Trong Python

Tóm Tắt Nội Dung

Trong bài học này chúng ta sẽ đi tìm hiểu lần lượt các nội dung

1. Tổng quan về kiểu dữ liệu từ điển Dictionaries
2. Tìm hiểu và cách định nghĩa một function (hàm) trong Python
3. Function có chứa tham số, tham số tùy chọn
4. Function có trả về dữ liệu với câu lệnh return
5. Anonymous function (A lambda function) trong Python

6.1 Tổng quan về kiểu dữ liệu từ điển Dictionaries

Khái niệm về Dictionaries

- Trong Python, Dictionaries (Từ điển) là một loại cấu trúc dữ liệu mạnh mẽ và linh hoạt, cho phép bạn lưu trữ và truy xuất dữ liệu theo cặp khóa-giá trị
- Dictionaries được biểu thị bằng dấu ngoặc nhọn "{}" và được sử dụng rộng rãi trong Python để lưu trữ và quản lý các cặp dữ liệu có tính chất "khóa-giá trị".
- **Mỗi cặp trong từ điển bao gồm hai phần:**

1. **Khóa (Key):** Đây là giá trị duy nhất dùng để xác định và truy xuất giá trị tương ứng. Khóa có thể là một đối tượng bất kỳ có thể băm (hashable) như chuỗi (string), số nguyên (integer), bộ (tuple), v.v

2. **Giá trị (Value):** Đây là giá trị được lưu trữ liên kết với khóa tương ứng. Giá trị có thể là bất kỳ đối tượng Python nào

6.1 Tổng quan về kiểu dữ liệu từ điển Dictionaries

Ví dụ

```
person = {
    "name": "John",
    "age": 30,
    "city": "New York"
}
```

Trong ví dụ này, từ điển "**person**" chứa ba cặp khóa - giá trị:

- Khóa **"name"** liên kết với giá trị **"John"**
- Khóa **"age"** liên kết với giá trị **30** (kiểu số nguyên).
- Khóa **"city"** liên kết với giá trị **"New York"**

6.1 Tổng quan về kiểu dữ liệu từ điển Dictionaries

Ví dụ

```
data = {
    "numbers": [1, 2, 3, 4, 5],
    "info": {
            "name": "Alice", "age": 25, "occupation": "Engineer"
            },
    "some_function": lambda x: x ** 2
}
```

- Qua ví dụ này và ví dụ trước bạn thấy **value** có thể có rất nhiều kiểu giá trị

  Kiểu số, chữ, set, function, ….

- Từ điền lồng vào nhau như value của key info ở trên

6.1 Tổng quan về kiểu dữ liệu từ điển Dictionaries

Đặc tính của dữ liệu từ điển

- Từ điển là một cấu trúc dữ liệu **không có thứ tự**, điều này có nghĩa là các phần tử không được sắp xếp theo vị trí thứ tự, mà được truy xuất thông qua khóa của chúng
- Bạn có thể thêm, sửa đổi và xóa các cặp khóa-giá trị trong từ điển một cách linh hoạt
- Không cho phép trùng lặp **key**

```
person = {
    "name": "John",
    "age": 30,
    "city": "New York“,
    "city": “WDC"
}
```

Lỗi vì trùng lặp key **city**

6.2 Phương thức xử lý với từ điển Dictionaries

Phương thức len()

Kiểm tra độ dài của từ điển

```
print(len(person)) # cho kết quả: 3
```

Phương thức type()

Kiểm tra kiểu dữ liệu của từ điển

```
print(type(person)) #cho kết quả <class 'dict'>
```

6.2 Phương thức xử lý với từ điển Dictionaries

Truy cập đến các phần tử của từ điển

- Bạn có thể lấy giá trị (value) của một phần tử dựa vào key của chúng

```
name = person["name"]
print(name)
```

- Ngoài ra bạn có thể sử dụng phương thức **get()** để lấy

```
name = person.get("name")
print(name)
```

- Lấy danh sách key của từ điển với phương thức **keys()**

```
keyList = person.keys()
print(keyList)
```

6.2 Phương thức xử lý với từ điển Dictionaries

Truy cập đến các phần tử của từ điển

- Lấy tất cả các value (giá trị) của từ điển với phương thức **values()**

```
vList = person.values()
print(vList)
```

- Lấy cả key và value của các phần tử trong từ điển với phương thức **items()**

```
itemList = person.items()
print(itemList)
#Kết quả in ra được:
[(‘name', 'Jonh'), (‘age', 30), (‘city', ‘New York’)]
```

Trả về một list các tuple

6.2 Phương thức xử lý với từ điển Dictionaries

Check sự tồn tại của một phần tử trong từ điển

- Ví dụ check city có tồn tại trong từ điển person không

```
if  "city" in person:
  print("Yes, city'  là một key trong từ điển person")
```

Thay đổi một phần tử trong từ điển

- Thay đổi giá trị (value) cho một phần tử (key) trong từ điển

```
person[“city”] = “Canada”
```

13.2 Phương thức xử lý với từ điển Dictionaries

Thay đổi một phần tử trong từ điển

- Hoặc bạn có thể sử dụng phương thức **update()**

```
person.update({“city”: “Canada”})
```

Thêm một phần tử vào từ điển

```
person[“email”] = “jonh@gmail.com”
```

Xóa một phần tử trong từ điển

```
person.pop(“email”)
```

Hoặc

```
del person[“email”]
```

6.2 Phương thức xử lý với từ điển Dictionaries

Thay đổi một phần tử trong từ điển

- Xóa phần tử ở cuối từ điển với phương thức **popitem()**

```
person.popitem()
```

Xoá luôn một từ điển

```
del person
```

Xóa tất cả phần tử trong từ điển, trả về từ điển rỗng

```
person.clear()
```

6.2 Phương thức xử lý với từ điển Dictionaries

Vòng lặp for với kiểu dữ liệu từ điển

- Lặp qua từ điển nó trả lại **key** sau mỗi lần lặp

```
for x in person:
   print(x) # Kết quả: name age city
```

- Hoặc bạn dùng phương thức keys()

```
for x in person.keys():
   print(x) # Kết quả: name age city
```

- Qua đó để lấy giá trị của các phần tử sau mỗi lần lặp bạn code như sau:

```
for x in person:
   print(person[x]) # Kết quả: Jonh 30 New york
```

6.2 Phương thức xử lý với từ điển Dictionaries

Vòng lặp for với kiểu dữ liệu từ điển

- Hoặc sử dụng phương thức **values()**

```
for x in person.values():
   print(x) # Kết quả: Jonh 30 New york
```

- Lặp qua lấy cả key và value với phương thức **items()**

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

6.2 Phương thức xử lý với từ điển Dictionaries

Sao chép (copy) một kiểu dữ liệu từ điển

- Sử dụng phương thức **copy()**

```
mydict = person.copy()
print(mydict)
```

- Hoặc một cách khác với phương thức dict()

```
mydict = dict(person)
print(mydict)
```

6.3 Phương thức xử lý với từ điển Dictionaries

Nguồn tham chiếu

- 1. Giới thiệu về kiểu dữ liệu từ điển trong Python – W3School

  https://www.w3schools.com/python/python_dictionaries.asp

- 2. Danh sách các phương thức xử lý từ điển trong Python – W3School

  https://www.w3schools.com/python/python_dictionaries_methods.asp

6.3 Tìm hiểu về Function trong Python

Định nghĩa: function là gì ?

Trong lập trình, **Function** (hàm) là một khối mã có tên được sử dụng để thực hiện một tác vụ cụ thể.

Function giúp chia nhỏ code thành các phần nhỏ hơn, dễ quản lý và sử dụng lại (tái sử dụng)

Cú pháp định nghĩa 1 function

```
def  name_function():
        #Khối lệnh được thực thi khi hàm được gọi
```

Name_function: Đặt theo quy tắc tên biến trong Python

Khối lệnh của hàm: phải thụt vào ít nhất 1 kí tự trắng so với lệnh def

6.3 Tìm hiểu về Function trong Python

Ví dụ về function

Một hàm có thể có hoặc không có giá trị trả về (return value)

```
def my_function():
    print("Lệnh 1")
    print("Lệnh 2")

my_function() #Gọi hàm, cho hàm khởi chạy
```

Trong ví dụ trên hàm chỉ thực thi các lệnh mà không trả lại giá trị gì thì thường được gọi là hàm **void** (Hàm thủ tục)

6.3 function có chứa tham số, tham số tùy chọn

6.3.1 Function có chứa tham số (parameter)

- **Tham số (Parameters):** là các biến được khai báo trong phần khai báo hàm, nhằm đại diện cho các giá trị sẽ được truyền vào khi gọi hàm
- Tham số được đặt tên và có thể có hoặc không có giá trị mặc định
- Khi bạn định nghĩa hàm, bạn có thể khai báo một hoặc nhiều tham số, phân tách chúng bằng dấu phẩy

```
def hello_name(name):
    print("Hello " + name)
```

6.3 function có chứa tham số, tham số tùy chọn

6.3.1 Function có chứa tham số (parameter)

- **Đối số (Arguments):** là các giá trị cụ thể được truyền vào hàm khi bạn gọi nó
- Số lượng và thứ tự của các đối số phải tương ứng với số lượng và thứ tự của các tham số tương ứng trong phần định nghĩa hàm
- Bạn có thể truyền đối số bằng giá trị cụ thể hoặc biến chứa giá trị

```
def hello_name(name):
    print("Hello " + name)

hello_name("Jonh") #Gọi hàm, cho hàm khởi chạy

name = 'Jonh'
hello_name(name) #đối số có thể là biến
```

6.3 function có chứa tham số, tham số tùy chọn

6.3.1 Function có chứa tham số tùy chọn/mặc định

- Giá trị mặc định chỉ được sử dụng nếu không có đối số tương ứng được truyền vào khi gọi hàm

```
def hello_name(name = ‘Guest’):
      print(“Hello “ + name)

hello_name(“Jonh”) #Kết quả: Hello Jonh
hello_name() #Kết quả: Hello Guest
```

Trong ví dụ trên, tham số name của hàm hello_name() có giá trị mặc định là "Guest".
Nếu không có đối số được truyền, giá trị mặc định "Guest" sẽ được sử dụng

6.3 function có chứa tham số, tham số tùy chọn

6.3.1 Function có nhiều tham số

```
def hello_name(fname, lname, age = 18):
    print(fname + " " + lname , age)
```

- Khi bạn định nghĩa hàm, bạn có thể khai báo một hoặc nhiều tham số, phân tách chúng bằng dấu phẩy
- Tham số mặc định luôn luôn được định nghĩa sau cùng

Ví dụ gọi hàm **add_numbers()** và sử dụng giá trị trả về:

6.3 function có trả về return

Cách định nghĩa

```
def sum(a, b):
    sum = a + b
    return sum
```

- Hàm nhận a, b là tham số đầu vào
- Hàm trả về (return) giá trị tổng của 2 số a và b

Ví dụ gọi hàm sum() và sử dụng giá trị trả về:

```
result = sum(5, 3)
print(result) # Kết quả: 8
```

6.3 Phạm vi sử dụng biến trong Function

Local Scope (Phạm vi cục bộ)

Biến được khai báo bên trong khối lệnh của hàm thì chỉ sử dụng được bên trong phạm vi khối hàm đó, kể cả hàm con của nó.

```
def myFunction():
        x =  3;
        print(x)
        def subFunction():
                print(x)
```

Qua ví dụ trên biến **x** có thể sử dụng được cả trong hàm con subFunction

6.3 Phạm vi sử dụng biến trong Function

Global Scope (Phạm vi toàn cục)

Ví dụ thứ 2: Biến toàn cục với từ khóa global

```
def myFunction():
    global x
    x = 3;
    print(x)

print(x)
```

Qua ví dụ trên biến **x** dùng được trong hàm myFunction, vừa dùng được bên ngoài hàm myFunction

6.3 Phạm vi sử dụng biến trong Function

Global Scope (Phạm vi toàn cục)

Để thay đổi giá trị của một biến toàn cục bên trong một hàm, hãy tham chiếu đến biến đó bằng cách sử dụng từ khóa toàn cục:

```
x =  2;
def myFunction():
      global x
      x =  3;
      print(x)

print(x)
```

Qua ví dụ trên biến **x** cuối cùng được in ra là x = 3

6.3 Hàm nặc danh (lambda function)

Trong Python, Lambda function (còn được gọi là hàm vô danh) là một loại hàm đặc biệt mà không cần định nghĩa bằng từ khóa def như các hàm thông thường. Lambda function được tạo ra bằng cú pháp ngắn gọn và linh hoạt, thường được sử dụng trong các trường hợp đơn giản khi bạn chỉ cần một hàm nhỏ mà không cần đặt tên

Cú pháp

lambda arguments: expression

- **lambda** là từ khóa để định nghĩa Lambda function
- **arguments** là danh sách các tham số (có thể không có hoặc có nhiều tham số)
- **expression** là biểu thức mà Lambda function sẽ thực hiện và trả về kết quả, và chỉ duy nhất 1 dòng lệnh

6.3 Hàm nặc danh (lambda function)

Ví dụ về lambda function

lambda với một tham số a, và khi đó a + 10 chính là return của hàm

```
x = lambda a : a + 10
print(x(5))
```

lambda với hai tham số a, b – các tham số cách nhau bằng dấu phẩy

```
x = lambda a, b : a * b
print(x(5, 6))
```

6.3 Hàm nặc danh (lambda function)

Ví dụ về lambda function

lambda bên trong một function khác

```
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)
print(mydoubler(12))
```

Qua ví vụ này chúng ta phải gọi 2 lần để thực thi được chương trình

6.3 Hàm nặc danh (lambda function)

Nguồn tham chiếu

- Tổng quan về Function

  https://www.w3schools.com/python/python_functions.asp

- Hàm nặc danh Lambda

  https://www.w3schools.com/python/python_lambda.asp

- Danh mục các hàm dựng sẵn - Built in Functions

  https://www.w3schools.com/python/python_ref_functions.asp

Tổng kết lại bài

1. Nắm được thế nào là kiểu dữ liệu từ điển Dictionaries
2. Nắm được các phương thức xử lý với từ điển Dictionaries
3. Nắm được tổng quan về Hàm, các loại hàm, cách sử dụng hàm