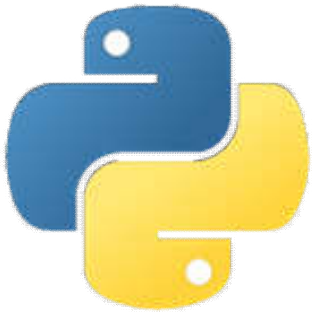

# Dictionary & Function

# Nội dung

1. **Nhắc lại bài cũ**
2. **Dictionary**
3. **Function**
4. **Q & A**

- Kiểu dữ liệu:
  - **List**: Kiểu dữ liệu thay đổi.
    - Cập nhật: Một phần tử hoặc 1 dãy phần tử.
    - Thêm: Append hoặc Insert.
    - Xóa: Del hoặc Remove()
  - **Tuple**: Kiểu dữ liệu bất biến
    - Cập nhật: Một phần tử hoặc 1 dãy phần tử
    - Thêm: Dùng phép toán +
    - Xóa: Dùng lệnh Del

# Nội dung

1. **Nhắc lại bài cũ**
2. **Dictionary**
3. **Function**
4. **Q & A**

# 1.1 Dictionary

- Dictionary (từ điển) là một kiểu dữ liệu, bao gồm nhiều phần tử (element).
- Mỗi phần tử là một cặp khóa và giá trị (Key & value), tương tự với khái niệm ***Map*** trong **Java**.
- Để khai báo một dictionary sử dụng cặp dấu móc { } , các phần tử bên trong phân cách nhau bởi đấu phẩy.
- Mỗi phần tử là một cặp khóa và giá trị ngăn cách nhau bởi dấu hai chấm ( : ).

## ❖Khai báo tổng quát:

```
# Dictionary
_info = {"Name": "Le Van A", "Age": 35, "Country": "Vietnam"}
print(_info)
```

## ❖Khai báo dựa vào hàm dict():

```
# Tạo một dictionary thông qua constructor của lớp dict.
_dict = dict()
_dict["Name"] = "Nguyen Van B"
_dict["Age"] = 25
print("Dictionary: ", _dict)
```

# 1.2 Đặc điểm Key & Value

- Value:
  - Mỗi phần tử (element) của dictionary là một cặp (khóa và giá trị), giá trị có thể là một kiểu bất kỳ (string, number, các kiểu người dùng tự định nghĩa,....), và cho phép trùng lặp.
- Key:
  - Key trong dictionary phải là kiểu bất biến (immutable): string, number, Tuple…
  - Các key trong dictionary không được phép trùng lặp.

```
# Code 1
_dict = dict()
_dict[1] = "One"
_dict[2] = 'Two'
_dict[3] = 'Two'
print("Dictionary: ", _dict)
```

```
# Code 2
_dict = {1: "One", '2': 'Two', 2: "Three"}
print("Dictionary: ", _dict)
```

```
#Code 3
_dict = {1: "One", 2: 'Two', 2: "Three"}
print("Dictionary: ", _dict)
```

# 1.3 Truy cập vào phần tử của Dict

❖Sử dụng cú pháp _dict[key].

```
#Code 1
_dict = {1: "One", '2': 'Two', 2: "Three", 4: 'Four', 5: 'Five'}
print("Dictionary: ", _dict[2])

print("Dictionary: ", _dict['2'])


# Code 2
_dict = {'Name': "Python", "2": 'Two', 2: "Three", 4: 'Four', 5: 'Five'}

print("Dictionary: ", _dict['Name'])
print("Dictionary: ", _dict['2'])
```

# 1.4 Cập nhật Dict

- Dict cho phép cập nhập giá trị ứng với một khóa nào đó.
- Dict sẽ thêm mới một phần tử nếu khóa đó không tồn tại trên dictionary.
- Có thể sử dụng hàm update: _dict.update(dict_element)

```
# Dictionary
_info = {"Name": "Tran", "Age": 37, "Address": "Vietnam"}
# Cập nhập giá trị cho khóa (key) 'Address'
_info["Address"] = "HCM Vietnam"
# Thêm một phần tử mới với khóa (key) là 'Phone'.
_info["Phone"] = "12345"
print("Element count: ", len(_info))

print('New dict: ', _info)
print("_info['Name'] = ", _info["Name"])
print("_info['Age'] = ", _info["Age"])
print("_info['Address'] = ", _info["Address"])
print("_info['Phone'] = ", _info["Phone"])
```

```
_info = {"Name": "Tran", "Age": 37, "Address": "Vietnam"}

_info.update({'key_1': 'value_1', "Name": "Update name"})

print(_info)
```

# 1.5 Xóa phần tử

- Có 2 cách để loại bỏ một phần tử ra khỏi một dictionary.
  - Sử dụng toán tử **del**
  - Sử dụng phương thức **__delitem__(key)**

```
# (Key,Value) = (Tên, Số điện thoại)
contacts = {"An": "01217000111", "Binh": "01234000111", "Tam": "01217000222"}

print("Contacts: ", contacts)
print("Delete key = 'An' ")
# Xóa một phần tử ứng với khóa 'An'
del contacts["An"]
print("Contacts (After delete): ", contacts)
print("Delete key = 'Binh' ")
# Xóa một phần tử ứng với khóa 'Binh'
contacts.__delitem__("Binh")
print("Contacts (After delete): ", contacts)
print("Clear all element")
# Xóa toàn bộ các phần tử.
contacts.clear()
print("Contacts (After clear): ", contacts)
# Xóa luôn dictionary 'contacts' khỏi bộ nhớ
del contacts
# Lỗi xẩy ra khi truy cập vào biến không tồn tại trên bộ nhớ
print("Contacts (After delete): ", contacts)
```

# 1.6 Hàm thao tác với Dict

| Hàm | Mô tả |
|---|---|
| len(dict) | Trả về số phần tử của dict. |
| str(dict) | Hàm chuyển đổi ra một string biểu diễn dictionary. |
| *type(variable)* | *Trả về kiểu của biến được truyền vào. Nếu biến truyền vào là một dictionary, thì nó sẽ trả về đối tượng đại diện lớp **'dict'**.* |
| *dir(clazzOrObject)* | *Trả về danh sách các thành viên của lớp (hoặc đối tượng) được truyền vào. Nếu truyền vào là lớp **dict**, nó sẽ trả về danh sách các thành viên của lớp **dict**.* |

```
# (Key,Value) = (Tên, Số điện thoại)
_contacts = {"An": "01217000111", "Binh": "01234000111", "Tam": "01217000222"}
print("contacts: ", _contacts)

print("Element count: ", len(_contacts))

_contactsAsString = str(_contacts)
print("str(contacts): ", _contactsAsString)
print("str(contacts) - From 0 - 6: ", _contactsAsString[:6])
print("str(contacts) - 0: ", _contactsAsString[0])
# Một đối tượng đại diện lớp 'dict'.

aType = type(_contacts)
print("type(contacts): ", aType)
```

# 1.7 Phương thức cho Dict (1)

| STT | Phương thức và Miêu tả |
|---|---|
| 1 | **Phương thức dict.clear()**<br>Xóa tất cả phần tử của *dict* |
| 2 | **Phương thức dict.copy()**<br>Trả về bản sao của *dict* |
| 3 | **Phương thức fromkeys(seq,value1)/ fromkeys(seq)**<br>Được sử dụng để tạo một Dictionary mới từ dãy seq và value1. Trong đó dãy seq tạo nên các key và tất cả các key chia sẻ các giá trị từ value1. Trong trường hợp value1 không được cung cấp thì value của các key được thiết lập là None |
| 4 | **Phương thức dict.get(key, default=None)**<br>Trả về giá trị của key đã cho. Nếu key không có mặt thì phương thức này trả về None |
| 5 | **Phương thức dict.has_key(key)**<br>Trả về true nếu key là có mặt trong Dictionary, nếu không là false |

# 1.7 Phương thức cho Dict (2)

| STT | Phương thức và Miêu tả |
|---|---|
| 6 | **Phương thức dict.items()**<br>Trả về tất cả các cặp (key-value) của một Dictionary |
| 7 | **Phương thức dict.keys()**<br>Trả về tất cả các key của một Dictionary |
| 8 | **Phương thức dict.setdefault(key, default=None)**<br>Tương tự get(), nhưng sẽ thiết lập dict[key]=default nếu *key* là không tồn tại trong dict |
| 9 | **Phương thức dict.update(dict2)**<br>Được sử dụng để thêm các item của dictionary 2 vào Dictionary đầu tiên |
| 10 | **Phương thức dict.values()**<br>Trả về tất cả các value của một Dictionary |

# 1.8 Luyện tập

# Nội dung

1. **Nhắc lại bài cũ**
2. **Dictionary**
3. **Function**
4. **Q & A**

# 2.1 Function

- Hàm, là một khối code được tổ chức và có thể tái sử dụng, để thực hiện một hành động hay một nhiệm vụ nào đó. Có rất nhiều hàm cung cấp sẵn trong Python, điển hình như hàm print() được dùng để đưa dữ liệu ra console.
- Tuy nhiên lập trình viên cũng có thể tạo riêng cho mình một hàm với cách định nghĩa và kiểu giá trị cho riêng. Các hàm này được gọi là user-defined function.

## ❖Cú pháp:

```
def ten_ham( cac_tham_so ):
        "function_doc_string"
        function_suite
        return [bieu_thuc]
```

```
# Định nghĩa một hàm: truyền vào tên người và trả về một chuỗi.
def sayHello(name):
    'Hàm truyền vào tên người và trả về một chuỗi'
    return "Hello " + name

# Sử dụng hàm sayHello
text = sayHello("Nguyen Van A!")
print(text)
```

- Khi định nghĩa các hàm cần theo các qui tắc sau:
  - Từ khóa **def** được sử dụng để bắt đầu phần định nghĩa hàm. Def xác định phần bắt đầu của khối hàm.
  - Cú pháp khai báo hàm: **def** ten_ham():
  - Các tham số được truyền vào bên trong các dấu ngoặc đơn. Ở cuối là dấu hai chấm.
  - Trước khi viết một code, một độ thụt dòng được cung cấp trước mỗi lệnh. Độ thụt dòng này nên giống nhau cho tất cả các lệnh bên trong hàm đó.
  - Documentation String của một hàm được khai báo đầu tiên trong ‘’ hoặc “”. Sau đó là các lệnh để thực thi.

❖Để thực thi một hàm, bạn cần gọi hàm đó. Cú pháp như sau:

```
ten_ham( cac_tham_so )
```

```
# Phan dinh nghia ham
def printme(str):
    "Chuoi nay duoc truyen vao trong ham"
    print(str)
    return

# Goi ham printme
printme("Loi goi dau tien toi custom func!")
printme("Loi goi thu 2 toi custom func.")
```

# 2.2 Phạm vi biến

- Tất cả các biến trong một chương trình không phải có thể truy cập tại tất cả vị trí ở trong chương trình đó. Điều này phụ thuộc vào nơi lập trình viên đã khai báo biến đó.
- Phạm vi biến quyết định nơi nào của chương trình có thể truy cập, sử dụng biến đó. Trong Python, có hai khái niệm về phạm vi biến là:
  - Biến toàn cục
  - Biến cục bộ

# 2.2.1 Biến cục bộ

- Các biến được khai báo bên trong một thân hàm là biến cục bộ.
- Các biến cục bộ này chỉ có thể được truy cập ở bên trong hàm mà đã được khai báo, không thể được truy cập ở bên ngoài thân hàm đó.

```
def msg():
    # Bien cuc bo
    a = 10
    print("Gia tri cua a la", a)
    return


msg()
print (a)  # Xay ra error vi bien day la cuc bo cua ham msg
```

# 2.2.2 Biến toàn cục

- Biến được định nghĩa bên ngoài hàm được gọi là biến toàn cục.
- Biến toàn cục có thể được truy cập bởi tất cả các hàm ở khắp nơi trong chương trình. Do đó phạm vi của biến toàn cục là rộng nhất.

```
b = 20

def msg():
    a = 10
    print("Gia tri cua a la", a)
    # Bien toan cuc co the duoc su dung ben trong ham
    print("Gia tri cua b la", b)
    return

msg()
print(b)
```

# 2.3 Tham số của hàm

- Python hỗ trợ các kiểu tham số chính như sau:
  - Tham số bắt buộc
  - Tham số mặc định
  - Tham số từ khóa (tham số được đặt tên)
  - Tham số thay đổi

# 2.3.1 Tham số bắt buộc

❖Các tham số bắt buộc là các tham số được truyền tới một hàm theo một thứ tự chính xác. Tức là, số tham số trong lời gọi hàm sẽ kết nối chính xác với phần định nghĩa hàm.

```
# Phan dinh nghia cua ham sum
def sum(_a, _b):
    "Ham tinh tong hai tham so"
    _c = _a + _b
    print(_c)


sum(5, 10)
# Error
sum(20)
```

- Trong trường hợp đầu tiên, khi hàm sum() được gọi đã được truyền hai giá trị là 5và 10, đầu tiên Python so khớp với phần định nghĩa hàm, sau đó 5 và 10 được gán tương ứng cho a và b. Do đó hàm sum được tính toán và in ra kết quả.
- Trong trường hợp thứ hai, khi chỉ truyền cho hàm sum() một giá trị là 20, giá trị này được truyền tới phần định nghĩa hàm. Tuy nhiên phần định nghĩa hàm chấp nhận hai tham số trong khi chỉ có một giá trị được truyền, do đó sẽ tạo ra một lỗi.

# 2.3.2 Tham số mặc định

❖Tham số mặc định là tham số mà cung cấp các giá trị mặc định cho các tham số được truyền trong phần định nghĩa hàm, trong trường hợp mà giá trị không được cung cấp trong lời gọi hàm.

```
# Phan dinh nghia ham
def msg(_id, _name, _age=23):
    "In gia tri da truyen"
    print('Id:', _id)
    print('Ten:', _name)
    print('Tuoi:', _age)
    return
# Goi ham
#msg(100, 'Hoang', 20)
msg(_id=100, _name='Hoang', _age=20)
print('*******')
msg(_id=101, _name='Thanh')
#msg(101,'Thanh')
```

- Trong trường hợp đầu tiên, khi hàm msg() được gọi đang truyền ba giá trị là 100, Hoang, và 20, thì các giá trị này sẽ được gán tương ứng cho các tham số và do đó chúng được in ra tương ứng.
- Trong trường hợp thứ hai, khi chỉ truyền hai tham số cho hàm msg() được gọi là 101 và Thanh, thì các giá trị này được gán tương ứng cho _id và _name. Không có giá trị nào được gán cho tham số thứ ba trong lời gọi hàm, và vì thế hàm sẽ lấy giá trị mặc định là 23.

# 2.3.3 Tham số từ khóa

❖Sử dụng tham số từ khóa, tham số được truyền trong lời gọi hàm được kết nối với phần định nghĩa hàm dựa trên tên của tham số. Với trường hợp này, vị trí của các tham số trong lời gọi hàm là tùy ý.

```
def msg(_id, _name):
    "In gia tri da truyen"
    print('Id: ', _id)
    print('Tên: ', _name)
    return


msg(_id=100, _name='Hoang')
msg(_name='Thanh', _id=101)
```

❖Trong trường hợp đầu tiên, trong lời gọi hàm msg() truyền hai giá trị và truyền vị trí giống trong phần định nghĩa hàm. Sau khi so khớp với phần định nghĩa hàm, thì các giá trị này được truyền tương ứng với các tham số trong phần định nghĩa hàm. Điều này được thực hiện dựa trên tên tham số.

❖Trong trường hợp thứ hai, trong lời gọi hàm msg(), cũng truyền hai giá trị nhưng với vị trí khác với phần định nghĩa hàm. Thì ở đây, dựa vào tên của tham số, các giá trị này cũng được truyền tương ứng cho các tham số trong phần định nghĩa hàm.

# 2.3.4 Tham số thay đổi

- Trong trường hợp cần xử lý một hàm mà có số tham nhiều hơn là số tham số đã xác định trong khi định nghĩa hàm. Những tham số này được gọi là các tham số có số tham số thay đổi (variable-length args) và không được đặt tên trong định nghĩa hàm, nó không giống như các tham số bắt buộc và tham số mặc định.
- Cú pháp:

```
def ten_ham([tham_so_chinh_thuc,] *var_args_tuple ):
                "function_docstring"
               function_suite
               return [bieu_thuc]
```

```
# Phan dinh nghia ham o day
def printinfo(arg1, *var):
    "In mot tham so da truyen"
    print("Ket qua la: ")
    print(arg1)
    for v in var:
        print(v)
    return


# Bay gio ban co the goi ham printinfo
# Goi lan 1
printinfo(10)
# Goi lan 2
printinfo(70, 60, 50)
```

# 2.4 Luyện tập


Q
&
A