***GHI CHÉP HỌC PYTHON***

**STRINGS**

|  |  |
| --- | --- |
| astring = "Hello world!"  print(astring.index("o")) | In giá trị vị trí đầu tiên xuất hiện giá trị index |
| print(astring[3:7]) | In giá trị của chuỗi từ vị trí thứ 3 tới vị trí thứ 6 |
| print(astring[3:7:2]) | In từ giá trị ở vị trí thứ 3, tiếp theo cứ cách 2 vị trí lại in tiếp. Chỉ xét trong khoảng từ index3 đến index6 |
| print(astring[::-1]) | Xét cả chuỗi, đầu tiên in giá trị cuối cùng. Sau đó cứ duyệt từ cuối tới đầu chuỗi in ngược lại.  VD: Nếu là -2 thì sẽ in ra "!lo le" |
| print(astring.upper())  print(astring.lower()) | Chuyển đổi toàn bộ thành chữ hoa  chuyển đổi toàn bộ thành chữ thường |
| print(astring.startswith("Hello"))  print(astring.endswith("asdfasdf") | Kiểm tra giá trị bắt đầu và kết thúc chuỗi |
| afewwords = astring.split(" ") | Dự vào giá trị truyền vào hàm split đưa vào mà cát chuỗi đưa vào mảng |
| Note | Nếu nhập vào số dương x, thì khi lấy kí tự x sẽ ở vị trí x+1  Nếu nhập vào số âm x, thì khi lấy kí tự x sẽ ở vị trí x |

%s: Lấy giá trị chuỗi

%d: Lấy giá trị integer

%f: Lấy giá trị số thực

*Lưu ý khi nhập vào 1 string từ bàn phím:*

*- Đối với python2, nếu ta chỉ dùng hàm s=input() --> Thì chương trình sẽ hiểu s là một biến số (có thể là int hay float).*

*- Muốn nhập vào 1 chuỗi trong python2 chúng ta phải dùng hàm s=raw\_input() thì nó mới hiểu s là 1 chuỗi.*

*- Đối với python3 thì chỉ cần dùng hàm s=input(). Mình điền vào kiểu dữ liệu gì nó sẽ hiểu rằng mình đang nhập vào kiểu dữ liệu đó.*

**LỚP PYTHON**

Lớp python được khai báo:

class Class\_name:

… # Thường thì chỗ này sẽ khai báo các thuộc tính cơ bản của lớp

def funtion():....

Note: Các thuộc tính sẽ được biểu diễn thông qua <Class\_object>.<Thuộc tính>

Ví dụ: myobjectx = MyClass()

myobjectx.variable

Các hàm được khai báo trong lớp luôn luôn phải có "self", nó giống như con trỏ trong C. Mặc định trong C, có thể khai báo hoặc không con trỏ này nhưng python thì bắt buộc phải có.

Muốn gọi được funtion của Class thì phải khai báo biế x=Class(), sau đó x.funtion(...)

**DICTIONARY PYTHON**

Khai báo bằng dấu {}:

*<Dictionary>{keys() : values()} Tức là để duyệt tên key ta dùng <Dictionary>. keys(). Tương tự cho values. Và để lấy cả 2 giá trị key và value ta dùng <Dictionary> . values()*

VD: Phonebook{}

for name, number in phonebook.items(): ....

1. print (dict.keys()) # Prints all the keys
2. print (dict.values()) # Prints all the values

Truy cập đến các phần tử dùng: <Dictionary-name>[<name of key>]

Ý nghĩa của hàng này là: In ra giá trị (value) của các key. Key có thể là số hoặc chuỗi.

Gán giá trị: Phonebook["Alice"] = 0969696655

Vẫn chưa hiểu nó điền các biến mình khai baó vào bằng cách nào.

Thao tác với dictionary:

Xoá 1 item: **<Dictionary-name>.pop("name of key")** or **del <Dictionary-name>["Name of key"]**

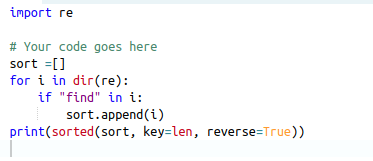
Xoá cả dictionary: **del <Dictionary-name>**

Xoá tất cả các phần tử mà không xoá dictionary: **<dictionary-name>.clear()**

Cập nhật dictionary**: <Dictionary-name>={key: value}** : Câu lệnh này sẽ cập nhật lại giá trị cho các key nếu key đã tồn tại sẵn, còn nếu nó chưa tồn tại thì item này sẽ được thêm vào.

Thêm 1 item vào dic: **<Dicyionary-name>["name of key"] = value**

**WRITING MODULES:**



Hàm append(): Thêm phần tử trong ngoặc vào mảng, có thể số hoăc chuôix kí tự.

sorted: Dùng để sắp xếp chuỗi chúng ta cần theo ý muốn:

Cú pháp: sorted(name\_of\_List, key=None, reverse=False): Hai thông số key và reverse mặc định không khai báo chương trình sẽ hiểu là sắp xếp chuỗi theo a,b,c còn đối với số thì sẽ sắp xếp từ nhỏ đén lớn(Khai báo: sorted(name\_of\_list))

nếu reverse=True: Sắp xếp theo chiều ngược lại.

Xét chuồi trong vd: Sắp xếp theo độ dài, nếu độ dài càng lớn thì sẽ được sắp xếp trước. Nếu cũng độ dài thì xếp theo an pha bê.

*Import các lớp con trong thư viện:*

Cú pháp: **import <Cha>.<Con>**

hoặc: **From <Cha> import <Con>**

Hình dung cách để biết trong thư viện cha và con có gì thông qua ví dụ sau:

*Vd: Trong thư viện của re có lớp côn là split:*

**print(dir(re))**

**print(dir(re.split))**

*Hàm init trong python??*

\_\_init\_\_ là hàm dựng hay constructor của một class. Khi một thực thể (instance) của một class được tạo ra thì hàm này sẽ được thực thi đầu tiên và một cách tự động

*Tại sao các hàm trong class cứ phải có self?*

self giống như this trong các ngôn ngữ hướng đối tượng khác. Đối với các ngôn ngữ khác thì mình không cần phải truyền this hoặc self vào. Nhưng Python yêu cầu phải như thế.

self trong Python không nhất thiết phải là self mà có thể là dat hoặc dnh cũng được. Nhưng người ta sử dụng self để thể hiện đây là thực thể của object. Trong tiếng Anh self có thể hiểu là bản thân nó. Điều này giúp việc đọc code dễ hơn.

**GENERATOR**

Những đối tượng "iterable" có thể được duyệt qua các phần tử, bởi vì chúng được cài đặt phương thức \_\_iter\_\_. Phương thức này sẽ trả về một đối tượng iterator. Đối tượng này cần phải hỗ trợ giao thức iteration (sẽ được nói đến sau). Nếu một đối tượng "iterable" có nhiều kiểu duyệt phần tử khác nhau, có thể chúng ta sẽ cần thêm các xử lý để xác định iterator. (Ví dụ một đồ thị có thể duyệt theo chiều rộng và theo chiều sâu.)

Với đối tượng iterator, nó cần phải được cài đặt hai phương thức sau, và bộ hai phương thức này được gọi là giao thức iteration.

* Phương thức \_\_iter\_\_ trả về chính đối tượng iterator. Phương thức này được yêu cầu cài đặt cho cả đối tượng "iterable" và iterator để có thể sử dụng các câu lệnh for và in.
* Phương thức \_\_next\_\_ (ở Python 2 là next) trả về phần tử tiếp theo. Nếu không còn phần tử nào nữa thì StopIteration exception sẽ được raise.
* Iterator object đơn giản chỉ là một đối tượng mà cho phép ta lấy từng giá trị một của nó. ==>Không thể lấy bất kì giá trị nào như ta hay làm với List hay Chuỗi.
* Iterator không có khả năng tái sử dụng trừ một số iterator có phương thức hỗ trợ như file object sẽ có phương thức seek.
* Iterator sử dụng hàm next để lấy từng giá trị một. Và sẽ có lỗi StopIteration khi sử dụng hàm next lên đối tượng đó trong khi nó hết giá trị đưa ra.

generator function cho phép chúng ta tạo ra các iterator theo một cách đơn giản hơn.

Generator giới thiệu câu lệnh yield cho Python. Nó hoạt động giống như return vì nó trả về một giá trị.

Vậy là chúng ta đã **tiết kiệm được trạng thái** của function. Trong lần tiếp theo function được gọi, vẫn tiếp tục từ **nơi dừng lại**, với **cùng các giá trị biến** mà nó đã có trước khi yielding.

**Hoạt động:** Khi hàm generator được gọi, nó trả kết quả là một đối tượng generator và không thực sự gọi và thực thi hàm. Khi phương thức \_\_next\_\_ được gọi, hàm generator sẽ bắt đầu chạy, cho tới khi nó gặp lệnh yield. Giá trị được yield sẽ được trả về cho hàm \_\_next\_\_.

**Note: VD:**

def fib():

a, b = 0, 1

while 1:

yield b **# chỗ này sẽ xác định gặp b thì ngừng và trả về giá trị b.**

a, b = b, a + b

**MULTIPLE FUNTIONS ARGUMENTS**

Lưu ý khi khai báo hàm:

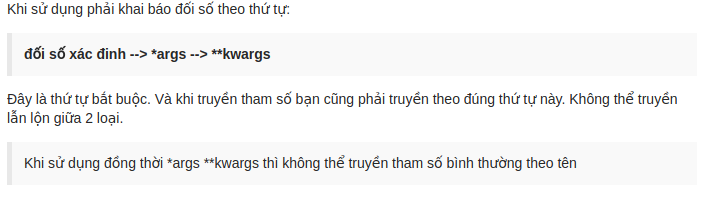
def(a,b,c,\*d): Thì d là phần mở rộng, là biến số của hàm có thể có hoặc không, hoặc có thể coi nó như 1 mẳng các phần tử.

def(a,b,c,\*\*d): Thì d chính là các định nghĩa các trường hopwj thao tác đối với 3 tham số a,b,c chứ nó không phải là biến số của hàm.

* **\*args** nhận các tham số truyền bình thường. Sử dụng **args** như một list.
* **\*\*kwargs** nhận tham số truyền theo tên. Sử dụng **kwargs** như một. Dictionary

Sử dụng dictionary có thể in ra được giá trị của từng key

https://daynhauhoc.com/t/cach-truyen-tham-so-dang-args-va-kwargs-trong-python/7115



**SETS**

**Set** là một container, tuy nhiên không được sử dụng nhiều bằng [LIST](http://www.howkteam.vn/redirect?Id=Y%2fHKZv6fbvNJnzXtx4Ic0lthogs3xmFNd%2fmA%2beIi3HryAF8oSVDXw8tVQcWzs%2bBVJJ65snGjoC37FhTDY7gxAxznBWxrjSCDk2DulPLI3BdPdg%2fXJ8UdJVQVu4nl56z8A2EXGiw2UbVhnUEtCMTcmr%2b8FyVrpkjBD3uTbTHHuPc%3d) hay [TUPLE](http://www.howkteam.vn/redirect?Id=Y%2fHKZv6fbvNJnzXtx4Ic0lthogs3xmFNd%2fmA%2beIi3Ho51JKvj70j%2bmXEz1sb1QQhjX8dEZZ13DfSXFf43aWJ5hKvpoZlLaTD1MDEAdSHiUbI7dCbZqD3YIsdn8NKctzsYytkchWLjGBx%2bPbvAG0WWA%3d%3d).



Một Set gồm các yếu tố sau:

* Được giới hạn bởi cặp ngoặc **{}**, tất cả những gì nằm trong đó là những phần tử của Set.
* Các phần tử của Set được phân cách nhau ra bởi dấu phẩy (**,**).
* Set không chứa nhiều hơn 1 phần tử trùng lặp

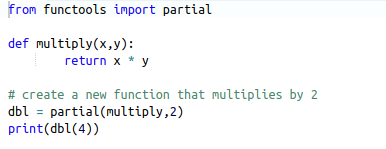
a = set(["Jake", "John", "Eric"])

b = set(["John", "Jill"])

* print(a.intersection(b)): Chỉ in ra phần tử trong a giống với b, không có trả về set()
* print(a.symmetric\_difference(b)): In ra tất cả các phần tử trong a và b khác nhau (Khác thì cho chung 1 set) như vd trên sẽ in ra {'Eric', 'Jill', 'Jake'}
* print(a.difference(b)): Chỉ in ra các phần tử trong a khác trong b
* print(a.union(b)): Gộp 2 set lại với nhau, cái nào trùng thì chỉ giữ lại 1

**PARTIAL FUNTION**

Sử dụng hàm partial trong python có thể đặt các số mặc định khi truyền vào 1 hàm và khi muốn gọi hàm chúng ta chỉ cần truyền thêm các phần tử còn lại.



Note: Khi sử dụng partial sẽ thay thế các phần tử theo thứ tự từ trái sang.

VD:

from functools import partial

def func(u,v,w,x):

return u\*4 + v\*3 + w\*2 + x

dbl = partial(func,5) #**Thay thế u=5. Có thể thay thế nhiều phaàn tử cùng lúc. VD: p = partial(func,5,6,7)**

print(dbl(4,4,20)) #**Khi gọi hàm chỉ cần truyền vào các tham số còn lại.**

**HƯỚNG ĐỐI TƯỢNG TRONG PYTHON**

**CODE INTROSPECTION**

|  |  |
| --- | --- |
| **Hàm** | **Mô tả** |
| *help()* | có thể sử dụng help(tên\_funtion) để biết cách sử dụng cũng như cấu trúc gọi hàm. |
| *dir():* | Liệt kê danh sách các thành viên của lớp hoặc đối tượng. hàm này liệt kê ra danh sách các phương thức, thuộc tính, biến của lớp hoặc của đối tượng. |
| *hasattr()* | hasattr(obj,name): Kiểm tra xem đối tượng này có thuộc tính cho bởi tham số 'name' hay không. |
| getattr() | getattr(obj, name[, default]): Trả về giá trị của thuộc tính, hoặc trả về giá trị mặc định nếu đối tượng không có thuộc tính này. |
| setattr() | setattr(obj,name,value): Sét giá trị vào thuộc tính. Nếu thuộc tính không tồn tại, thì nó sẽ được tạo ra. |
| delattr() | delattr(obj, name): Xoá bỏ thuộc tính |
| **id()** |  |
| **type()** | Trả về kiểu của biến đã được truyền vào |
| **repr()** | repr(object) Hàm chuyển đổi đối tượng x thành một chuỗi biểu thức  *``print`` hiển thị ra những gì mã bên dưới muốn hiển thị.* *``repr`` trả về biểu diễn string dạng chuẩn của object.* |
| **callable()** | Kiểm tra xem đối tượng có phải là 1 hàm hay không |
| **issubclass()** | Hàm **issubclass(sub, sup)** trả về true nếu lớp con **sub** đã cho thực sự là lớp con của lớp cha **sup**. |
| **isinstance()** | Hàm **isinstance(obj, Class)** trả về true nếu **obj** là một instance của lớp **Class** hoặc là một instance của lớp con của Class. |

**CLOSURES - HÀM LỒNG NHAU**