CHƯƠNG 1. HÀM, DỮ LIỆU VÀ BIẾN

1.1. HIỂN THỊ DỮ LIỆU

1.1. print

Ví dụ 1.1. In dòng chữ Hello world ra màn hình

Python là một ngôn ngữ có cú pháp rất đơn giản. Bắt đầu với chương trình in ra màn hình chữ Hello world. Lệnh đơn giản nhất để in là print.

Users > thinhdoanvu > Documents > DataScience > code_python > hello.py

```
1  print("Hello world")
```

Lưu ý với Python2 hàm print không có dấu ().

Một số lưu ý với hàm print().

Escape sequence: giúp in kí tự đặc biệt ra màn hình

\\: print(" \\ ") → in ra ký tự \

\': print(" \' ") → in ra ký tự '

\n: print(" \n ") → in ra ký tự xuống dòng

\t: print(" \t ") → in ra ký tự tab

""": print(""" """) → In ra màn hình trên nhiều dòng

''': print(''' ''') → In ra màn hình trên nhiều dòng

Ví dụ 1.2. Hãy in dòng sau ra màn hình:

C:\\ some	one "else" '2023

Đáp án:

Users > thinhdoanvu > Documents > DataScience > code_python > hello.py

```
1  print("C:\\\ some \t one \"else\" \'2023")
```

Ví dụ 1.3: Hãy in ra màn hình các dòng sau

Usage: thingy[Options]

-h:	Display usage message

-H Hostname: Hostname be connected to

Đáp án:

Users > thinhdoanvu > Documents > DataScience > code_python > hello.py

```
1  print(""" Usage: thingy[Options]
2      -h:           Display usage message
3      -H Hostname:  Hostname be connected to
4  """)
```

1.1.2. Ghi chú (comment)

Để ghi chú hoặc bình luận trong chương trình và muốn trình thông dịch Python sẽ bỏ qua trong quá trình thực thi, chúng ta sử dụng dấu #

Ví dụ 1.4: Xem xét chương trình sau

Users > thinhdoanvu > Documents > DataScience > code_python > hello.py

```
1  #In ra màn hình dòng chữ Hello world
2  print("Hello world")#Hàm print để in nội dung
3  #Hết
```

1.1.3. Thụt lề (indentation)

Python không sử dụng cặp dấu {...} như trong ngôn ngữ lập trình C hoặc begin...end như trong ngôn ngữ Pascal để đánh dấu cho một khối lệnh mà thay vào đó sử dụng dấu khoảng trắng (tab và space). Theo đó, quy chuẩn của Python sử dụng 4 dấu space.

1.1.4. Một số lưu ý

- Python không sử dụng dấu ; khi kết thúc lệnh như C/Pascal

- Đối với kiểu dữ liệu logic (True/False): lưu ý viết hoa ký tự đầu tiên

- Sử dụng dấu : sau tên của hàm, vòng lặp và cấu trúc điều khiển

Ví dụ 1.5: Xem xét ví dụ sau

```
1  x=1
2  if x==True:
3      #lưu ý dấu và True
4      print("x is 1.")#x là số 1
5  else:
6      print("x is 0.");#x là số 0
```

Kết quả: x is 1.

Tuy nhiên nếu ta gán x khác 1 (x=0, x=2, x=3,...) thì kết quả sẽ cho kết quả: x is 0.

1.2. KIỂU DỮ LIỆU

Trong Python, mọi biến đều là 1 đối tượng nên chúng ta không cần phải khai báo biến trước khi sử dụng như C hoặc Pascal. Có 2 loại biến: (1) biến toàn cục, (2) biến cục bộ.

	Chuỗi kí tự (**str**)	Số nguyên (**int**)	số thực (**float**)
Định nghĩa	các kí tự được đặt trong cặp dấu nháy đơn ' ' hoặc nháy kép " "	gồm các số nguyên dương, nguyên âm và 0	gồm phần nguyên và phần thập phân
Ví dụ	'ký tự' "he's" '1234'	123 -123	123.4 -123.0
Hàm chuyển đổi về kiểu dữ liệu	str(123) ->'123' str(23.5)->'23.5'	int('123')->123 int(12.3)->123	float(123)->123.0 float('12.3')->12.3

Bảng 1: Các kiểu dữ liệu cơ bản trong Python

1.2.1. Kiểu chuỗi

Python cung cấp cho chúng ta nhiều hàm/phương thức (method) để xử lý với chuỗi. Đối với chuỗi trong Python chúng ta có thể truy vấn từ trái qua phải hoặc chiều ngược lại.

0	1	2	3	4	5
p	y	t	h	o	n
-6	-5	-4	-3	-2	-1

Ví dụ 1.6. Xem xét ví dụ sau

```
1  text_1="Python"
2  text_2='NTU2023'
3  print(text_1[0])#output is P
4  print(text_2[-7])#output is N
5  print(text_1[2:])#output is thon
6  print(text_2[:3])#output is NTU
7  print(text_1[-6:-4])#output is Py
8  print(text_2[3:5])#output is 20
```

Lưu ý: Số ký tự cần lấy nằm trong khoảng [start:stop] sẽ trù đi 1 ký tự (ký tự kết thúc)

1.2.1.1. Thao tác với chuỗi

- Thay thế giá trị của ký tự trong chuỗi

Ví dụ 1.7. Hãy thay thế chuỗi Python thành Cython

```
1  source="Python"
2  source[0]='C'
3  print(source)
```

Kết quả:

```
    source[0]='C'
    ~~~~~~^^^
TypeError: 'str' object does not support item assignment
```

Nguyên nhân: Chuỗi trong Python là bất biến nên không thể thay thế ký tự của nó (giá trị của một ô nhớ riêng lẻ) một khi đã khởi tạo.

Giải pháp: Chúng ta sử dụng toán tử + để nối 2 chuỗi lại với nhau.

```
1  source="Python"
2  source='C'+source[1:]
3  print(source)#cách 1
4  print('C'+source[1:])#cách 2
```

-Toán tử +

Đối với các kiểu dữ liệu dạng số (nguyên, thực) thì đây là phép cộng số học, tuy nhiên với dữ liệu dạng chuỗi thì nó dùng để nối 2 chuỗi lại với nhau.

Ví dụ 1.8. In ra màn hình hello world từ 2 chuỗi khác nhau

```
1  src='hello'
2  dest="world"
3  print(src+" "+dest)
```

Kết quả: hello world

- Toán tử *

Dùng để lặp lại số lần của chuỗi mong muốn

```
1  src="NTU\n"
2  print(src*3)
```

Kết quả:

NTU

NTU

NTU

- Định dạng chuỗi

- Sử dụng định dạng f-string với f đặt trước chuỗi hiển thị và các biến đặt trong dấu ngoặc {}

  Ví dụ 1.9. In ra màn hình chuỗi sau: dai hoc nha trang, 1959

  Đầu tiên bạn sẽ nghĩ đến việc đặt 2 biến như sau:

  ```
  1  school='dai hoc nha trang,'
  2  year=1959
  3  print(school+year)
  ```

  Tất nhiên là chương trình trên không thể hoạt động được, và một lỗi sẽ xuất hiện như sau:

  print(school+year)
  ~~~~~~^~~~~
  TypeError: can only concatenate str (not "int") to str

  Lỗi này xuất hiện do 2 kiểu dữ liệu khác nhau thì không thể sử dụng toán tử +.

  Cách giải quyết: là định dạng year thành chuỗi như sau:

  ```
  1  school='dai hoc nha trang,'
  2  year='1959'
  3  print(school+year)
  ```

  Cách này thì ổn, tuy nhiên trong thực tế không phải lúc nào ta cũng có dữ liệu dạng chuỗi (string) mà rất thường xuyên dữ liệu của chúng ta là dạng số (number) và một số dạng khác nữa (sẽ bàn sau). Do đó, phương án được sử dụng ở đây là chúng ta sẽ tiến hành định dạng về kiểu string: f-string (được hiểu là formatted string)

  ```
  1  my_school='dai hoc nha trang,'
  2  est_year='1959'
  3  print(f"truong cua minh la: {my_school} được thành lập từ {est_year}")
  ```

  Kết quả: truong cua minh la: dai hoc nha trang, được thành lập từ 1959

- Sử dụng hàm str.format() tương tự như f-string

  Ví dụ 1.10. In ra màn hình nội dung của ví dụ 2.4 nhưng với cách tiếp cận format()

```
my_school='dai hoc nha trang,'
est_year='1959'
print("truong cua minh la: {} được thành lập từ {}".format(my_school,est_year))
```

Lưu ý: tên biến không được liệt kê trong {}, phía sau " được sử dụng dấu chấm .format(tên biến1, tên biến 2,...)

- Sử dụng định dạng % tương tự như ngôn ngữ lập trình C

Định dạng	Kiểu dữ liệu	Ý nghĩa
%c	char	ký tự
%s	char *	chuỗi ký tự
%d	int, short	Số nguyên dạng thập phân
%f	float	Số thực
%lf	double	Số thực chính xác gấp đôi

Ví dụ 1.11. In ra màn hình: John Doe is 22 years old.

```
name="John Doe"
age=22
print("%s is %d years old." %(name,age))
```

- Với các kiểu dữ liệu không phải chuỗi, đều có thể chuyển thành kiểu chuỗi với %s

Ví dụ 1.12. In ra màn hình danh sách list=[1,2,3]

```
list=[1,2,3]
print("Du lieu khong phai string: %s" %(list))
print(f"Du lieu khong phai string: {list}")
print("Du lieu khong phai string: {}".format(list))
```

Cả 3 cách viết trên đều cho cùng 1 kết quả hiển thị:

Du lieu khong phai string: [1, 2, 3]

Du lieu khong phai string: [1, 2, 3]

Du lieu khong phai string: [1, 2, 3]

Bài tập áp dụng:

Hãy in ra màn hình chuỗi sau: Hello John Doe. Your current balance is $53.44.

Với data=["John","Doe",53.44]

```
data=["John","Doe",53.44]
print("Hello %s, %s. Your current balance is $%s" %(data[0],data[1],data[2]))#cach 1
print(f"Hello {data[0]}, {data[1]}. Your current balance is ${data[2]}")#cach 2
print("Hello {}, {}. Your current balance is ${}".format(data[0],data[1],data[2]))#cach 3
```

Bạn cũng có thể thay thế bằng 3 biến: f_name="John", s_name="Doe", money=53.44

1.2.1.2. Một số hàm thao tác với chuỗi

Method	Ý nghĩa	Ví dụ
.strip()	loại bỏ tất cả khoảng trắng ở đầu và cuối của chuỗi	$s1.strip()$
.upper()	trả về một chuỗi hoa	$s.upper()$
.lower()	trả về một chuỗi thường	$s.lower()$
.replace()	thay thế một chuỗi cho chuỗi khác	s='Khoa CNTT' s1 = 'Khoa'; s2 = 'Lop' print($s.replace(s1,s2)$)
.split()	dùng để tách chuỗi thành chuỗi con	s='Khoa-CNTT' print($s.split("-")$) #kq: ['Khoa', 'CNTT']
.count()	đếm số lần xuất hiện một chuỗi con trong chuỗi	$s.count('K')$

Bảng 2: Một vài phương thức thao tác với chuỗi

1.2.2. Kiểu danh sách

list giúp ta có thể khai báo, lưu trữ, truy xuất một dãy gồm các đối tượng thuộc bất kì kiểu dữ liệu nào. Bản chất của list rất giống array. Khai báo danh sách như sau:

1.2.2.1. Cú pháp

<ten_danh_sach> = [<ptu1>,<ptu2>,<ptu3>...<ptun>]

Ví dụ: list=['hello',1,3,"xin chao",False,3.1416]

2.2.2. Truy xuất

Các phần tử được truy xuất thông qua chỉ số giống như khi truy xuất mảng (array). Cụ thể:

list[0]=list[-6] → 'hello'

list[1:3] → [1,3]

list[4:] → [False, 3.1416]

1.2.2.3. Các phép toán của danh sách

- Thay đổi giá trị của một phần tử: <tên_danh_sách> [chỉ số] = <giá_trị_mới]
- Thêm phần tử vào cuối danh sách: <tên_danh_sách>.append(giá_trị)
- Thêm phần tử vào vị trí bất kỳ: <tên_danh_sách>.insert(vị_trí,giá_trị)
- Xoá phần tử ở cuối danh sách: <tên_danh_sách>.pop()
- Xoá phần tử ở vị trí bất kỳ: del <tên_danh_sách>[vị_trí]
- Xoá phần tử có giá trị = x: <tên_danh_sách>.remove(x)
- Đếm số lần xuất hiện của x trong danh sách: <tên_danh_sách>.count(x)
- Sao chép danh sách: <tên_danh_sách>.copy()

- Nối 2 danh sách thành 1 danh sách: <tên_danh_sách 1> + <tên_danh_sách 2>
- Kiểm tra một phần tử có giá trị = x: x in <tên_danh_sách>
- Kiểm tra một phần tử không có giá trị = x: x not in <tên_danh_sách>
- Gán danh sách vào danh sách[1]: <tên_danh_sách_mới> = <tên_danh_sách_cũ>[vị trí]
- Nhân danh sách: <tên_danh_sách 1> * số lần
- Số phần tử của danh sách: len(<tên_danh_sách>)
- Phần tử có giá trị lớn nhất: max(<tên_danh_sách >)
- Phần tử có giá trị nhỏ nhất: min(<tên_danh_sách>)
- Truy vấn danh sách dựa vào vị trí của danh sách khác

```
1   a1=["b",2,3]
2   a2=[1,1.4,"ws",a1]
3   print(a2[3][0])
```

Kết quả: b (vì a2[3] = danh sách a1, a1[0] = 'b')

Ví dụ 1.13. Minh hoạ các chức năng của danh sách

```
list=['hello',1,3,"xin chao",False,3.1416]
#Truy cap phan tu
print("hien thi danh sach: ",list[1:3])
```

→ hien thi danh sach: [1, 3]

```
#Thay doi gia tri
list[2]=2
print("thay doi gia tri: ",list)
```

→ thay doi gia tri: ['hello', 1, 2, 'xin chao', False, 3.1416]

```
#Them ptu vao cuoi danh sach
list.append("end")
print("them vao cuoi ds: ",list)
```

→ them vao cuoi ds: ['hello', 1, 2, 'xin chao', False, 3.1416, 'end']

```
#Them vao vi tri bat ky
list.insert(0,"begin")
print("them vao dau: ", list)
```

→ them vao dau: ['begin', 'hello', 1, 2, 'xin chao', False, 3.1416, 'end']

[1] Việc gán danh sách không làm tạo ra một danh sách mới mà đơn giản chỉ ánh xạ phần tử của danh sách cũ vào trị trí của danh sách mới

```
#Xoa phan tu o cuoi danh sach
print("lay phan tu cuoi danh sach: ",list.pop())
```
→ lay phan tu cuoi danh sach: end
```
print("xoa phan tu cuoi danh sach: ",list)
```
→ xoa phan tu cuoi danh sach: ['begin', 'hello', 1, 2, 'xin chao', False, 3.1416]
```
#Xoa phan tu o vi tri bat ky
del list[2]
print("xoa phan tu o vi tri bat ky: ",list)
```
→ xoa phan tu o vi tri bat ky: ['begin', 'hello', 2, 'xin chao', False, 3.1416]
```
#Xoa phan tu co gia tri = gia tri cho truoc
list.remove(2)
print("Xoa phan tu co gia tri = 2: ",list)
```
→ Xoa phan tu co gia tri = 2: ['begin', 'hello', 'xin chao', False, 3.1416]
```
#Dem so lan xuat hien cua x trong danh sach
count=list.count("begin")
print(count)
```
→ 1
```
#Sao chep danh sach
list_copied=list.copy()
print("danh sach moi: ",list_copied)
```
→ danh sach moi: ['begin', 'hello', 'xin chao', False, 3.1416]
```
#Noi 2 danh sach
list_new = list + list_copied
print("Noi 2 danh sach: ",list_new)
```
→ ['begin', 'hello', 'xin chao', False, 3.1416, 'begin', 'hello', 'xin chao', False, 3.1416]
```
#Kiem tra mot phan tu co gia la x?
print(5 in list)
```
→ False
```
#Kiem tra mot phan tu KHONG co gia la x?
print(5 not in list)
```
→ True

```
#Gan danh sach vao vi tri trong danh sach
new="NTU"
list[0]=new
print("Gan danh sach vao danh sach: ",list)
```

→ Gan danh sach vao danh sach: ['NTU', 'hello', 'xin chao', False, 3.1416]

```
#Nhan danh sach
print("Danh sach sau khi nhan",list*2)
```

→ ['NTU', 'hello', 'xin chao', False, 3.1416, 'NTU', 'hello', 'xin chao', False, 3.1416]

1.2.3. Tuple

1.2.3.1. Khai báo

tên_biến_tuple = (item1, item2,..,itemn)

1.2.3.2. Định nghĩa

- Kiểu dữ liệu tuple là một danh sách có kích thước cố định, các phần tử trong tuple được sắp xếp theo thứ tự và không thể thay đổi đồng nghĩa với việc không được thêm, sửa hoặc xoá các phần tử sau khi được khởi tạo.

- Thường được dùng để lưu trữ các đối tượng cố định.

- Tuple được khai báo bằng dấu ngoặc đơn '()' khác với danh sách '[]'.

- Giống như list thì tuple cũng có thể được lồng vào nhau.

Ví dụ 1.14. Minh hoạ Tuple

```
1  a1=("b",2,3)
2  a2=(a1,1,'c',a1)
3  print(a2[0])
```

Kết quả: ('b', 2, 3)

1.2.3.3. Một số hàm thao tác với tuple

- Hàm tuple: được dùng để khai báo kiểu dữ liệu tuple hoặc chuyển dữ liệu dạng danh sách trở thành tuple.

Ví dụ 1.15. Chuyển định dạng danh sách thành tuple

```
1  a1=["b",2,3]
2  a2=(tuple(a1),1,'c',a1)
3  print(type(a2[0]))
```

Kết quả: <class 'tuple'>

- Hàm len(): trả về số phần tử của tuple

- Hàm max(): trả về phần tử có giá trị lớn nhất của tuple

- Hàm min(): trả về phần tử có giá trị bé nhất của tuple

- Hàm .index(x): trả về vị trí của phần tử có giá trị là x trong tuple

- Ngoài ra còn có hàm: +, *, in và not in

Ví dụ 1.15: Một số hàm của tuple

```
1  tuples=(1,3,0,8,6,9)
2  print("so phan tu: ",len(tuples))
3  print("phan tu be nhat: ",min(tuples))
4  print("phan tu lon nhat: ",max(tuples))
5  print("vi tri cua 0: ",tuples.index(0))
```

1.2.4. Set

1.2.4.1. Cú pháp

<tên_biến_set>={item1, item2, ... , item n}

1.2.4.2. Định nghĩa

- Kiểu dữ liệu set - là tập hợp các phần tử không có thứ tự, không thể thay đổi giá trị của các phần tử và không tồn tại chỉ số. Mỗi phần tử trong set là duy nhất (không có trùng lắp hay các giá trị trùng lắp sẽ bị lược bỏ).

- Các phần tử được khai báo trong dấu ngoặc nhọn { }

Ví dụ 1.16: Hiển thị các giá trị trong tập hợp

```
1  a1={1,2,3,2,'hi',"c"}
2  print("cac phan tu trong tap hop: ",a1)
```

Kết quả: cac phan tu trong tap hop: {1, 2, 3, 'hi', 'c'}

1.2.4.3. Một số phương thức xử lý cho tập hợp

- Thêm phần tử cho tập hợp: .add(<giá trị 1>, <giá trị 2>,...)

- Cập nhật giá trị cho tập hợp: .update(<giá trị 1>, <giá trị 2>,...) hoặc .update(<set khác>)

- Xoá giá trị trong tập hợp (mỗi lần chỉ được xoá 1 giá trị): .remove(<giá trị>)

Ví dụ 1.17. Minh hoạ các phương thức xử lý cho tập hợp

```
a1={1,2,3,2,'hi'}
#In noi dung trong tap hop
print("cac phan tu trong tap hop: ",a1)
```

→ cac phan tu trong tap hop: {1, 2, 3, 'hi'}

```
#Them phan tu vao tap hop
a1.add('e')
print("tap hop sau khi them ptu \'e\': ",a1)
```

→ tap hop sau khi them ptu 'e': {1, 2, 3, 'hi', 'e'}

```
a1.add(7)
```

```
print("tap hop sau khi them ptu 7: ",a1)
```

→ tap hop sau khi them ptu 7: {1, 2, 3, 7, 'hi', 'e'}

```
#Cap nhat gia tri cho phan tu
a2={1,2,3}
a2.update([0,4])
print("tap hop sau khi cap nhat gia tri: ",a2)
```

→ tap hop sau khi cap nhat gia tri: {0, 1, 2, 3, 4}

```
#cap nhat gia tri cho set tu 1 set khac
a3={'a','b'}
a2.update(a3)
print("cap nhat gia tri tu 1 set khac: ",a2)
```

→ cap nhat gia tri tu 1 set khac: {0, 1, 2, 3, 4, 'b', 'a'}

```
#Xoa gia tri
a2.remove(4)
print("tap hop a2 sau khi xoa gia tri 4: ",a2)
```

→ tap hop a2 sau khi xoa gia tri 4: {0, 1, 2, 3, 'b', 'a'}

1.2.5. Dictionary

- Từ điển được sử dụng để lưu trữ các giá trị dữ liệu trong các cặp key: value. Từ điển được viết trong dấu ngoặc nhọn {} như đối với set với các khoá và giá trị. Hay nói các khác từ điển là một tập hợp nghĩa là các phần tử có giá trị được sắp xếp theo thứ tự và không thể thay đổi giá trị (từ phiên bản 3.6 trở về trước là không được sắp xếp).

- Các khoá (key) phải là số hoặc chuỗi.

1.2.5.1. Cú pháp

tên_từ_điển = {key1:value1, key2:value2, ..., keyn:valuen}

Ví dụ 1.18. Cấu trúc từ điển

```
1  dict={1: 'hi', 2: 'hello', 4: 1, 3: 2, 5: 'a'}
2  print("tu dien dict la: ", dict)
```

Kết quả: tu dien dict la: {1: 'hi', 2: 'hello', 4: 1, 'key2': 2, 5: 'a'}

1.2.5.2. Một số phương thức xử lý trong từ điển

- Truy cập vào phần tử của từ điển: [] hoặc .get(<key>)

- Số phần tử của từ điển: len(ten_tu_dien)

- Lấy danh sách theo cặp giá trị key:value: .items()

- Xoá một mục của từ điển: del ten_tu_dien[key] hoặc .pop(key)

- Trả về danh sách các key của từ điển: .keys()

- Trả về danh sách các giá trị của từ điển: .values()

- Sắp xếp từ điển theo key → tuple[2]: sorted[3](ten_tu_dien.items(), key=lambda x: x[0])

Ví dụ 1.19. Các hàm trong dictionary

```
dict={1: 'hi', 4: 'hello', 5: 1, 'key2': 2, 2: 'a'}
print("tu dien dict la: ", dict)
```

→ tu dien dict la: {1: 'hi', 4: 'hello', 5: 1, 'key2': 2, 2: 'a'}

```
#Truy cap vao gia tri cua tu dien
print("phan tu co gia tri la key: ",dict[1])
```

→ phan tu co gia tri la key: hi

```
print("phan tu co gia tri la key: ",dict.get('key2'))
```

→ phan tu co gia tri la key: 2

```
#So phan tu cua tu dien
print("so phan tu cua tu dien: ",len(dict))
```

→ so phan tu cua tu dien: 5

```
#Hien thi theo tung cap key:value
print("danh sach gia tri key:value ",dict.items())
```

→ danh sach gia tri key:value dict_items([(1, 'hi'), (4, 'hello'), (5, 1), ('key2', 2), (2, 'a')])

```
#Xoa phan tu la key trong tu dien bang del
del dict[5]
print("xoa phan tu trong tu dien bang del: ",dict)
```

→ xoa phan tu trong tu dien bang del: {1: 'hi', 4: 'hello', 'key2': 2, 2: 'a'}

```
#Xoa phan tu la key trong tu dien bang pop
dict.pop('key2')
print("xoa phan tu trong tu dien bang pop: ",dict)
```

→ xoa phan tu trong tu dien bang pop: {1: 'hi', 4: 'hello', 2: 'a'}

```
#tra ve danh sach cac key cua tu dien
print("danh sach cac key: ",dict.keys())
```

→ danh sach cac key: dict_keys([1, 4, 2])

```
#tra ve danh sach cac gia tri cua tu dien
print("danh sach cac gia tri: ",dict.values())
```

[2] Sau đó, sử dụng hàm dict() để chuyển từ tuple sang dictionary

[3] key=lambda x: x[0] sử dụng lambda function để xác định rằng sắp xếp được thực hiện dựa trên key của các cặp key-value trong dictionary (x[0] là key)

→ danh sach cac gia tri: dict_values(['hi', 'hello', 'a'])

```
#sap xep theo key cua tu dien
print("tu dien sau khi sap xep: ",sorted(dict.items(),
key=lambda x: x[0]))
```

→ tu dien sau khi sap xep: [(1, 'hi'), (2, 'a'), (4, 'hello')]

	ordered	changeable	Allows duplicate	indexed
List	√	√	√	√
Tuple	√	×	√	√
Set	×	×	×	×
Dictionary	√ (Python version ≥ 3.7)	√	×	√

Bảng 3: Four collection data types in the Python

1.3. BIẾN

1.3.1. Quy tắc đặt tên cho biến

- Chỉ được chứa các chữ cái 'a' - 'z', 'A' - 'Z', các số '0'-'9' và dấu gạch nối '-'
- Không được bắt đầu bằng ký tự số
- Không có khoảng cách trong tên biến
- Tên biến phải gợi nhớ

Ví dụ một số tên biến: my_Name, Ballance, opt, _source,...

1.3.2. Gán giá trị cho biến

- Mỗi biến gán 1 giá trị (mỗi biến nằm trên 1 dòng):

  a=3

  pi=3.1416

- Có thể gán nhiều biến với mỗi biến 1 giá trị trên 1 dòng

Ví dụ 1.20. Xem xét ví dụ sau

```
1  heso, pi, chuoi = 2, 3.1416, 'Dai hoc Nha Trang'
2  print(heso, chuoi, pi)
```

Kết quả:

2 Dai hoc Nha Trang 3.1416

1.3.3. Kiểm tra kiểu dữ liệu của biến

Trong một số trường hợp ta cần kiểm tra kiểu dữ liệu của biến ta sử dụng hàm type(tên_biến)

Ví dụ 1.21. Hãy cho biết các kiểu dữ liệu của ví dụ 1.20

```
heso, pi, chuoi = 2, 3.1416, 'Dai hoc Nha Trang'
print(type(heso))
print(type(pi))
print(type(chuoi))
```

Kết quả:

<class 'int'>

<class 'float'>

<class 'str'>

3.4. Nhập dữ liệu từ bàn phím cho biến

Để đọc giá trị từ bàn phím chúng ta sử dụng hàm input(). Kết quả của hàm này là luôn luôn trả về dữ liệu dạng chuỗi.

Ví dụ 1.21. Nhập vào 2 số a và b, và cho biết kiểu dữ liệu của chúng

```
print("Nhap gia tri cua a = ")
a=input()
print("Nhap gia tri cua b = ")
b=input()
c=a+b
print("kieu du lieu cua a la", type(a))
print("kieu du lieu cua b la", type(b))
print("ket qua c = ",c)
print("kieu du lieu cua c la", type(c))
```

Kết quả:

Nhap gia tri cua a =

2

Nhap gia tri cua b =

3.14

kieu du lieu cua a la <class 'str'>

kieu du lieu cua b la <class 'str'>

ket qua c = 23.14

kieu du lieu cua c la <class 'str'>

Câu hỏi đặt ra là làm thế nào để cộng 2 số a và b trong khi dữ liệu nhập vào là kiểu string?

1.4. CÁC PHÉP TOÁN VỚI THƯ VIỆN MATH

Các phép toán +, -, *, /, % cũng giống với các phép toán của các ngôn ngữ lập trình khác như C hoặc Pascal hoặc mở rộng hơn có // (phép chia lấy phần nguyên được làm tròn xuống). Tuy nhiên Python cung cấp nhiều công cụ toán học hơn. Để sử dụng thư viện math chúng ta cần import vào Python thông qua lệnh import math

Ví dụ 1.22. Sử dụng các hàm toán học của Python

```
1  import math
2  x=math.pi
3  print(x)
4  print(math.sin(x/2))#sin 90 độ
5  print(math.tan(x/4))#tan 45 độ
6  print(math.sqrt(2))#căn bậc 2 (2)
7  print(math.ceil(x))#cận trên của x
8  print(math.floor(x))#cận dưới của x
```

Kết quả:

3.141592653589793

1.0

0.9999999999999999

1.4142135623730951

4

3

1.5. MỘT SỐ HÀM TRONG PYTHON

1.5.1. Cú pháp

def ten_ham(<tham_so_1>,<tham_so_2>,...,<tham_so_n>):

    <cau_lenh>

    return <ket_qua_tra_ve>

- Một số lưu ý: tham số trong hàm của python không bắt buộc, có thể có hoặc không có giá trị trả về.

- Hàm không được gọi thì không được thực thi.

5.2. Hàm có không có tham số truyền vào

Ví dụ 1.23. Hàm msg_box

```
1  def msg_box():
2      print("hello world")
3
4  msg_box()
```

Kết quả: hello world

1.5.3. Hàm có số lượng tham số truyền vào cố định

Ví dụ 1.24. Viết chương trình tính tổng 2 số a, b. Trong đó a, b là 2 tham số truyền vào

```
1  def tong(a, b):
2      return a+b
3  x=3
4  y=5
5  print("ham tinh tong cua {} + {} = {} ".format(x,y,tong(x,y)))
```

Kết quả: ham tinh tong cua 3 + 5 = 8

1.5.4. Hàm chưa biết trước số lượng đối số truyền vào

Ví dụ 1.25. Viết chương trình tính tổng 2 số

```
1  def tong(*n):
2      return n[0]+n[1]
3  x=3
4  y=5
5  print("ham tinh tong cua {} + {} = {} ".format(x,y,tong(x,y)))
```

Kết quả: ham tinh tong cua 3 + 5 = 8

Trong trường hợp chưa biết trước số lượng tham số cần truyền chúng ta có thể sử dụng dấu * ngay trước tham số duy nhất.

1.5.5. Package

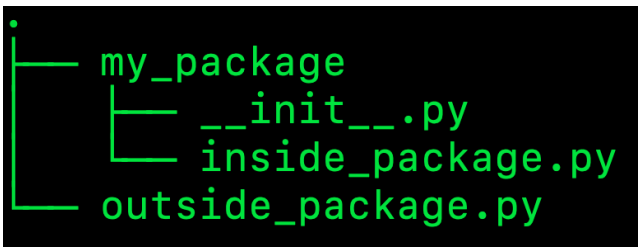
1.5.5.1. Tạo mới package

- Tạo mới folder chứa package: my_ package

- Tạo mới tập tin: __init__.py[4]

- Tạo mới các module (tập tin chứa mã nguồn) bên trong package: inside_package.py

Cấu trúc của package như sau:



Nội dung của các tập tin:

__init__.py: bỏ trống

insdide_package.py:

```
1  def msg_hello():
2      print("day la thong bao ben TRONG package")
```

[4] Tập tin này có thể để trống hoặc chứa các khai báo để được thực hiện khi package được import. Tập tin `__init__.py` làm cho thư mục trở thành một package Python.

1.5.5.2. Sử dụng package

Bằng cách import nó trong các tập tin Python khác.

```
from my_package import inside_package
def msg_hello():
    print("day la thong bao ben NGOAI package")
msg_hello()
```

Kết quả:

day la thong bao ben TRONG package

day la thong bao ben NGOAI package

Tại sao?

Khi chạy tập tin outside_package.py trình thông dịch của Python sẽ thực thi lệnh import inside_package.py nên lời gọi hàm msg_hello() ở đó cũng được thực thi. Sau đó, lời gọi hàm ở tập tin msg_hello() lại được thực hiện tiếp theo.

Giải pháp?

Để tránh trường hợp này, ta sử dụng thuộc tính __name__ có giá trị là __main__ như sau:

- Tập tin inside_package.py

Users > thinhdoanvu > Documents > DataScience > code_python > my_package > inside_package.py

```
def msg_hello():
    print("day la thong bao ben TRONG package")
if __name__ == '__main__':
    msg_hello()
```

- Tập tin outside_package.py

Users > thinhdoanvu > Documents > DataScience > code_python > outside_package.py

```
from my_package import inside_package
def msg_hello():
    print("day la thong bao ben NGOAI package")
if __name__ == '__main__':
    msg_hello()
```

Kết quả:

day la thong bao ben NGOAI package

1.6. PHẠM VI HOẠT ĐỘNG CỦA BIẾN

Giống như đa số ngôn ngữ lập trình hướng đối tượng khác, chúng ta có biến toàn cục, biến cục bộ và thời gian hoạt động của chúng.

1.6.1. Biến cục bộ

Biến cục bộ chỉ có tác dụng bên trong hàm hoặc trong khố (sau dấu :) và chỉ có tác dụng trong bản thân hàm hoặc khối đó. Biến cục bộ sẽ bị xóa khỏi bộ nhớ khi kết thúc khối đó.

Ví dụ 1.26. Ví dụ về biến cục bộ

```
1  def bien_cuc_bo():
2      x=7
3      print(f"gia tri cua bien cuc bo = {x}")
4  bien_cuc_bo()
5
6  print("gia tri cua bien = {}".format(x))
```

Kết quả:

gia tri cua bien cuc bo = 7

NameError: name 'x' is not defined

1.6.2. Biến toàn cục

- Có tác dụng lên toàn bộ chương trình.

Ví dụ 1.27. Ví dụ về biến toàn cục

```
1  x=7
2  def display():
3      print("gia tri cua bien toan cuc: {}".format(x))
4
5  display()
```

Kết quả: gia tri cua bien toan cuc: 7

- Biến toàn cục cũng có thể được khai báo trong hàm khác: global <ten_bien>

Ví dụ 1.28. Khai báo biến toàn cục trong chương trình/hàm khác

```
1  def bien_cuc_bo():
2      global x
3      x=7
4      print(f"gia tri cua bien cuc bo = {x}")
5  bien_cuc_bo()
6
7  print("gia tri cua bien = {}".format(x))
```

Kết quả:

gia tri cua bien cuc bo = 7

gia tri cua bien = 7

CHƯƠNG 2. CẤU TRÚC ĐIỀU KHIỂN

2.1. TOÁN TỬ

Cũng giống như đa số ngôn ngữ lập trình khác, Python cũng hỗ trợ các toán tử so sánh thông dụng: bằng (==), không bằng (!=), nhỏ hơn (<), nhỏ hơn hoặc bằng (<=), lớn hơn (>), lớn hơn hoặc bằng (>). Ngoài ra, người lập trình còn có thể kết hợp với các phép toán luận lý như: và (and), hoặc (or), phủ định (not)...

2.2. CÂU LỆNH RẼ NHÁNH

2.2.1. if...else

- Cú pháp:

```
if (dieu_kien):
        lenh_thuc_thi_khi_dieu_kien_dung
else:
        lenh_thuc_thi_khi_dieu_kien_sai
```

- Lưu ý:

Python sử dụng ký tự tab hoặc 4 ký tự trắng liên tiếp làm tiêu chuẩn cho bắt đầu và kết thúc một khối lệnh.

Ví dụ 2.1. Nhập 2 số từ bàn phím và in ra màn hình số lớn nhất

```
1  print("nhap so a = ")
2  a=input()
3  print("nhap so b = ")
4  b=input()
5  if(a>b):
6      print("so lon nhat la {}".format(a))
7  else:
8      print("so lon nhat la {}".format(b))
```

Kết quả:

nhap so a =

3

nhap so b =

5

so lon nhat la 5

2.2.2. if...elif...else

Trong trường hợp bài toán có nhiều hơn 2 trường hợp, chúng ta sẽ dùng câu lệnh elif để rẽ nhánh sang các trường hợp khác.

- Cú pháp:

```
if (dieu_kien_1):
        cau_lenh_1
elif (dieu_kien_2):
        cau_lenh_2
elif (dieu_kien_3):
        cau_lenh_3
...
else:
        cau_lenh_n
```

Ví dụ 2.2. Viết chương trình nhập vào 1 số nguyên từ bàn phím và cho biết số có mấy chữ số (số nguyên trong phạm vi 0 - 1000).

```
1  print("nhap 1 so trong pham vi 0-1000")
2  number=input()
3  if (int(number)<10):
4      print("so co 1 chu so")
5  elif (int(number)>10 and int(number)<100) :
6      print("so co 2 chu so")
7  elif (int(number)>100 and int(number)<1000):
8      print("so co 3 chu so")
9  else:
10     print("so nhap vao khong hop le")
```

Hoặc

```
1  print("nhap 1 so trong pham vi 0-1000")
2  number=input()
3  if (len(number)==1):
4      print("so co 1 chu so")
5  elif (len(number)==2) :
6      print("so co 2 chu so")
7  elif (len(number)==3):
8      print("so co 3 chu so")
9  else:
10     print("so nhap vao khong hop le")
```

Kết quả:

```
nhap 1 so trong pham vi 0-1000
123
so co 3 chu so
```

Lưu ý:

- Khi nhập dữ liệu từ bàn phím, Python sẽ coi đó là ký tự (dù nhập dạng số) nên cần chuyển định dạng sang số nguyên (hàm int(number)) nếu cần thiết.

- Một cách khác, vì định dạng của dữ liệu là dạng chuỗi nên chúng ta có thể sự dụng hàm len(number) để xác định chiều dài của chuỗi (hay số ký tự của số vừa nhập).

- Trong trường hợp câu lệnh if thoả điều kiện nhưng không làm gì cả thì cần sử dụng pass để bỏ qua.

Ví dụ 2.3. Bỏ qua điều kiện

```
print("nhap 1 so nguyen: ")
a=input()
print("nhap 1 so nguyen khac: ")
b=input()
if(a==b):
    pass
else:
    print("a # b")
```

2.2.3. switch...case

Câu lệnh switch ... case không tồn tại trong Python, cách để thể hiện điều này thông qua kiểu dữ liệu dictionary hoặc if..elif..else.

Ví dụ 2.4. Hiển thị các thứ trong tuần

```
def day_of_week(number):
    dow = {2: 'Monday', 3: 'Tuesday', 4: 'Wednesday', 5: 'Thursday',
           6: 'Friday', 7: 'Saturday', 0: 'Sunday'}
    result = dow.get(number, 'Khong ton tai thu trong tuan')
    print(result)

day_of_week(1)
day_of_week(5)
```

Kết quả:

Khong ton tai thu trong tuan

Thursday

2.3. Câu lệnh lặp

2.3.1. while

- Cú pháp:

```
while (dieu_kien):
    cau_lenh_thoa_dieu_kien
```

Với cú pháp này vòng lặp sẽ không thể dừng cho đến khi điều kiện sai.

Ví dụ 2.5. In ra màn hình các số nguyên trong khoảng từ 1 - n (với n nhập từ bàn phím)

```
print("Nhap n = ")
n=int(input())
i=1
while(i<=n):
    print(i, end=" ")
    i+=1
```

Kết quả:

Nhap n = 5

1 2 3 4 5

2.3.2. break

break được sử dụng để kết thúc vòng lặp khi thoả điều kiện mà không cần chờ đợi cho đến khi thực hiện đủ số vòng lặp định trước.

Ví dụ 2.6. Nhập vào 1 số n từ bàn phím thoả điều kiện (5<n<10), in ra các số 1-n

Cách 1

```
print("Nhap n = ")
n=int(input())
while(n<5 or n>10):
    print("Nhap n = ")
    n=int(input())
print("so vua nhap: %d" %n)
```

Nhap n = 11

Nhap n = 1

Nhap n = 6

so vua nhap: 6

Cách 2

```
while(1):
    print("Nhap n = ")
    n=int(input())
    if(n>5 and n<10):
        break
print("so vua nhap: %d" %n)
```

Nhap n = 3

Nhap n = 11

Nhap n = 7

so vua nhap: 7

Ví dụ 2.7. Viết chương trình tìm số nguyên n bé nhất sao cho 1/n < a, với a là số thực do người dùng nhập vào từ bàn phím

```
print("Nhap so thuc a = ")
a=float(input())
n=1
while(1/n >a):
    n+=1
print("n= %d, 1/%d <= %f" %(n,n,a))
```

Kết quả:

Nhap so thuc a = 0.125

n= 8, 1/8 <= 0.125000

2.3.3. continue

continue dùng để bỏ qua vòng lặp hiện tại và chuyển sang vòng lặp tiếp theo.

Ví dụ 2.8. In ra màn hình các số lẻ 1 - n, với n được nhập từ bàn phím

```
1  print("Nhap so nguyen n = ")
2  n=int(input())
3  i=0
4  while(i<n):
5      i+=1
6      if(i%2==0):
7          continue
8      print(i,end=" ")
```

Kết quả:

Nhap so nguyen n = 10

1 3 5 7 9

2.3.4. while...else

Không giống với ngôn ngữ lập trình C/Pascal, câu lệnh lặp while có thể kết hợp với else để kết thúc vòng lặp.

Ví dụ 2.9. In ra các số trong khoảng 1-n/2, với n là số nguyên được nhập từ bàn phím

```
1  print("Nhap so nguyen n = ")
2  n=int(input())
3  i=0
4  while(i<n/2):
5      i+=1
6      print(i,end=" ")
7  else:
8     print("\nket thuc vong lap")
```

Kết quả:

Nhap so nguyen n = 10

1 2 3 4 5

ket thuc vong lap

2.3.4. for

Khác với while, vòng lặp for được sử dụng trong trường hợp ta đã biết trước chính xác số lần lặp của đoạn chương trình.

- Cú pháp:

for <biến_chạy> in dãy:

câu_lệnh

Trong đó: dãy có thể là string, set, tubple, dictionary... và biến chạy sẽ nhận các giá trị thành phần có trong dãy.

Ví dụ 2.10. Minh hoạ cho vòng lặp for với các kiểu dữ liệu

`` day={1,2,3,4,5,6} for i in day: print(i, end=" ") ``` kiểu dữ liệu set	``` day=('a','b','c','d','e') for i in day: print(i, end=" ") ``` kiểu dữ liệu array
`` day=[1,2,3,4,5,6] for i in day: print(i, end=" ") ``` kiểu dữ liệu list	``` day="abcde" for i in day: print(i, end=" ") ``` kiểu dữ liệu string
Kết quả: 1 2 3 4 5 6	Kết quả: a b c d e

2.3.4.1. Kiểu dictionary

Với kiểu dữ liệu dạng từ điển (dictionary)

+ Một (1) biến chạy sẽ nhận từng cặp (key,value) của phần tử:

Ví dụ 2.11. Truy xuất phần tử của dictionary

```
dict={1:'mot', 2:'hai', 3:'ba', 4:'bon'}
for i in dict.items():
    print(i,end=' ')
```

Kết quả: (1, 'mot') (2, 'hai') (3, 'ba') (4, 'bon')

+ Hai (2) biến chạy sẽ nhận: (1) key và (2) value của từng phần tử:

Ví dụ 2.12. Truy xuất từng cặp giá trị key và value của dictionary

```
dict={1:'mot', 2:'hai', 3:'ba', 4:'bon'}
for i,j in dict.items():
    print(i," ",j)
```

Kết quả:

1 mot

2 hai

3 ba

4 bon

2.3.4.2 Hàm range trong vòng lặp for

+ hàm range(n) với một tham số truyền vào sẽ trả về một dãy số bắt đầu từ 0 và tăng lên 1 (theo mặc định), sau đó kết thúc tại n – 1.

Ví dụ 2.13. in ra các số nguyên trong khoảng từ 1-n, với n nhập từ bàn phím

```
n=10
for i in range(n):
    print(i,end=" ")
```

Kết quả: 0 1 2 3 4 5 6 7 8 9

+ hàm range(start, end, step) với tham số bắt đầu start, kết thúc end và bước nhảy step (để trống thì mặc định là 1).

Ví dụ 2.14. In ra các số nguyên trong khoảng từ 1-n, với n nhập từ bàn phím thoả (5<n<10)

Phân tích bài toán: **Đầu tiên** cần nhập số từ bàn phím và kiểm tra điều kiện thoả n, trong trường hợp này ta chưa biết trước số lần lặp (số lần mà người dùng nhập n thoả điều kiện): ta dùng vòng lặp while. **Sau đó** tiến hành in các số từ 1 - n, lần in này ta đã biết trước n (đã nhập từ bàn phím và đã thoả điều kiện) nên ta dùng vòng lặp for. Đương nhiên, ta có thể dùng vòng lặp while để thay thế cho vòng lặp for.

```
while(1):
    print("Nhap so nguyen n = ")
    n=int(input())
    if(n>=5 and n<=10):
        break
for i in range(1,10,1):
    print(i, end=" ")
```

Kết quả:

Nhap so nguyen n = 10

1 2 3 4 5 6 7 8 9

2.3.4.3. for...break

Sử dụng break để kết thúc vòng lặp khi chưa thực hiện đủ số lần lặp

Ví dụ 2.15. Hiển thị các giá trị trong khoảng từ 1-n/2

```
n=10
for i in range(n):
    print(i,end=" ")
    if(i>=n/2):
        break
```

Kết quả: 0 1 2 3 4 5

2.3.4.4. for...continue

Sử dụng continue để bỏ qua vòng lặp hiện tại và chuyển sang vòng lặp kế tiếp

Ví dụ 2.16. Hiển thị các số lẻ trong khoảng 1-n

```
n=10
for i in range(n):
    if(i%2==0):
        continue
    print(i,end=" ")
```

Kết quả: 1 3 5 7 9

2.3.4.5. for...else

Giống như vòng lặp while, từ khóa else trong vòng lặp for chỉ định một khối mã sẽ được thực thi khi vòng lặp for kết thúc.

Ví dụ 2.17. Sử dụng else trong vòng lặp for

```
n=10
for i in range(n):
    print(i,end=" ")
else:
    print("\nKet thuc vong lap")
```

Kết quả:

0 1 2 3 4 5 6 7 8 9

Ket thuc vong lap