

MỤC LỤC

Bài 1: CÂU LỆNH CƠ BẢN.....	7
1. Cài đặt Python.....	7
2. Viết chương trình.....	8
3. Câu lệnh print.....	9
4. Biến và kiểu dữ liệu.....	10
5. Hiển thị chuỗi theo định dạng.....	14
6. Phép toán.....	14
7. Câu lệnh nhập từ bàn phím.....	16
BÀI TẬP CHƯƠNG 1.....	18
Bài 2: BIỂU THỨC ĐIỀU KIỆN.....	19
1. Phép toán so sánh.....	19
2. Phép toán and và or.....	20
3. Độ ưu tiên toán tử.....	21
4. Điều kiện if.....	21
5. Điều kiện if ... else.....	22
6. Điều kiện if ... elif ... else.....	23
7. Biểu thức điều kiện.....	24
8. Cấu trúc try ... catch.....	24
BÀI TẬP CHƯƠNG 2.....	27
Bài 3: XÂY DỰNG HÀM.....	30
1. Định nghĩa hàm.....	30
2. Hàm không trả về kết quả.....	31
3. Hàm trả về kết quả.....	32
4. Thuộc tính __name__.....	33
5. Hàm đệ quy.....	34
BÀI TẬP CHƯƠNG 3.....	35
Bài 4: VÒNG LẶP.....	37
1. Vòng lặp while.....	37

2. Vòng lặp for.....	38
3. Vòng lặp lồng nhau Vòng lặp lồng nhau của while.....	42
4. Lệnh break trong vòng lặp Lệnh break là lệnh ngừng vòng lặp.....	43
5. Lệnh continue trong vòng lặp.....	45
6. Lệnh pass.....	45
BÀI TẬP CHƯƠNG 4.....	47
ÔN TẬP GIỮA KỲ.....	50
Bài 5: Danh sách.....	52
Bài tập chương 5.....	58
Bài 6: Chuỗi.....	60
Bài tập chương 6.....	68
Bài 8: TỪ ĐIỂN.....	70
1. Khai báo dữ liệu.....	70
2. Kiểu dữ liệu động.....	71
3. Tạo từ điển từ kiểu dữ liệu khác.....	71
4. Truy xuất phần tử.....	72
5. Thay đổi giá trị.....	72
6. Duyệt trong từ điển.....	73
7. Thêm, xóa phần tử.....	73
BÀI TẬP CHƯƠNG 8.....	74
Bài 9: BỘ (TUPLE).....	77
1. Khai báo dữ liệu.....	77
2. Kiểu dữ liệu cố định.....	78
3. Truy xuất phần tử.....	78
4. Duyệt với vòng lặp.....	78
5. Thêm, xóa phần tử.....	79
6. Các hàm của Bộ.....	79
BÀI TẬP CHƯƠNG 9.....	80
Bài 10: TẬP TIN.....	83
1. Giới thiệu tập tin.....	83
1. Trình tự thao tác với tập tin.....	83
2. Mở tập tin.....	84

3.	Đọc tập tin.....	85
4.	Ghi tập tin.....	86
5.	Đóng tập tin.....	87
6.	Xóa tập tin.....	87
BÀI TẬP CHƯƠNG 10.....		88
TÀI LIỆU THAM KHẢO		92

GIỚI THIỆU

Python là ngôn ngữ lập trình cấp cao, hỗ trợ người dùng ở các dạng: tương tác dòng lệnh (command line); script và hướng đối tượng. Chương trình của Python có thể được viết bằng bất kỳ trình soạn thảo văn bản nào (Notepad, Notepad++, IDE Python, PyCharm, ...) và có phần mở rộng là *.py. Trong khi các chương trình viết bằng ngôn ngữ khác như C/C++; VB, .NET phải thực hiện biên dịch (compiler), thì chương trình viết bằng Python chỉ thực thi thông qua trình thông dịch (interpreter).

Các thư viện trong Python khá đa dạng, từ các thư viện như NumPy được sử dụng trong lĩnh vực tính toán ma trận, thống kê, phân tích dữ liệu, đến các thư viện hỗ trợ lập trình ứng dụng mạng như Socket, Webservice, lập trình thiết kế web với framework Flask hay Django. Mặc dù là ngôn ngữ lập trình cấp cao, nhưng cách tiếp cận của Python theo ngôn ngữ kịch bản (script) nên rất phù hợp với tất cả mọi người khi mới bước chân vào môi trường lập trình.

Python được phát triển bởi Guido Van Rossum vào cuối những năm 1980 đầu những năm 1990 tại Viện Nguyên cứu toán và khoa học máy tính ở Hà Lan, và được phát hành lần đầu tiên vào tháng 2 năm 1991. Nó được xây dựng trên nhiều nền tảng khác nhau gồm: ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages. Hiện tại, Python được lưu hành theo giấy phép mã nguồn mở GNU - GPL.

Một số đặc điểm của Python:

- **Dễ học:** từ khóa Python tương đối ít, cấu trúc đơn giản và cú pháp rõ ràng. Phù hợp với các bạn mới tiếp cận học ngôn ngữ lập trình.
- **Dễ đọc:** Câu lệnh của python rõ ràng và tường minh, dễ đọc và dễ viết hơn các ngôn ngữ lập trình khác như C/C++, Java, ...
- **Dễ bảo trì:** Mã nguồn python bảo trì dễ dàng.
- **Thư viện chuẩn linh hoạt:** Phần lớn thư viện của python được tương thích với các môi trường Unix, Windows và Macintosh. Chương trình viết bằng Python có thể thực thi trên các nền tảng khác mà không có bất kỳ thay đổi nào. Nó chạy liền mạch trên hầu hết tất cả các nền tảng như Windows, macOS, Linux.
- **Chế độ tương tác (Interactive mode):** Python hỗ trợ chế độ tương tác cho phép lập trình viên có thể kiểm tra và debug chương trình.
- **Lập trình giao diện (GUI):** Python hỗ trợ lập trình giao diện, người sử dụng có thể thiết kế các chương trình ứng dụng khác nhau.
- **OOP (Object Oriented Programming):** Mọi thứ trong Python đều là hướng đối tượng. Lập trình hướng đối tượng (OOP) giúp giải quyết những vấn đề phức tạp một cách trực quan. Với OOP, bạn có thể phân chia những vấn đề phức tạp thành những tập nhỏ hơn bằng cách tạo ra các đối tượng.
- **Mã nguồn mở:** Python có một cộng đồng rộng lớn cùng xây dựng và phát triển, người dùng có thể sử dụng và phân phối Python.
- **Khả năng mở rộng và có thể nhúng:** Giả sử một ứng dụng đòi hỏi sự phức tạp rất lớn, bạn có thể dễ dàng kết hợp các phần code bằng C, C++ và những ngôn ngữ khác (có thể gọi được từ C) vào code Python. Điều này sẽ cung cấp cho ứng dụng của bạn những tính năng tốt hơn cũng như khả năng scripting mà những ngôn ngữ lập trình khác khó có thể làm được.
- **Ngôn ngữ thông dịch cấp cao:** Không giống như C/C++, với Python, bạn không phải lo lắng những nhiệm vụ khó khăn như

quản lý bộ nhớ, dọn dẹp những dữ liệu vô nghĩa, ... Khi chạy code Python, nó sẽ tự động chuyển đổi code sang ngôn ngữ máy tính có thể hiểu. Bạn không cần lo lắng về bất kỳ hoạt động ở cấp thấp nào.

- **Thư viện tiêu chuẩn lớn để giải quyết những tác vụ phổ biến:**
Python có một số lượng lớn thư viện tiêu chuẩn giúp cho công việc lập trình của bạn trở nên dễ thở hơn rất nhiều, đơn giản vì không phải tự viết tất cả code. Ví dụ: Bạn cần kết nối cơ sở dữ liệu MySQL trên Web server? Bạn có thể nhập thư viện MySQLdb và sử dụng nó. Những thư viện này được kiểm tra kỹ lưỡng và được sử dụng bởi hàng trăm người. Vì vậy, bạn có thể chắc chắn rằng nó sẽ không làm hỏng code hay ứng dụng của mình.
- **Hướng đối tượng:** Mọi thứ trong Python đều là hướng đối tượng. Lập trình hướng đối tượng (OOP) giúp giải quyết những vấn đề phức tạp một cách trực quan. Với OOP, bạn có thể phân chia những vấn đề phức tạp thành những tập nhỏ hơn bằng cách tạo ra các đối tượng.

Bài 1: CÂU LỆNH CƠ BẢN

Mục tiêu:

- Làm quen với giao diện Python
- Có thể thực hiện thao tác nhập/xuất
- Có thể thực hiện các phép tính cơ bản

Nội dung chính:

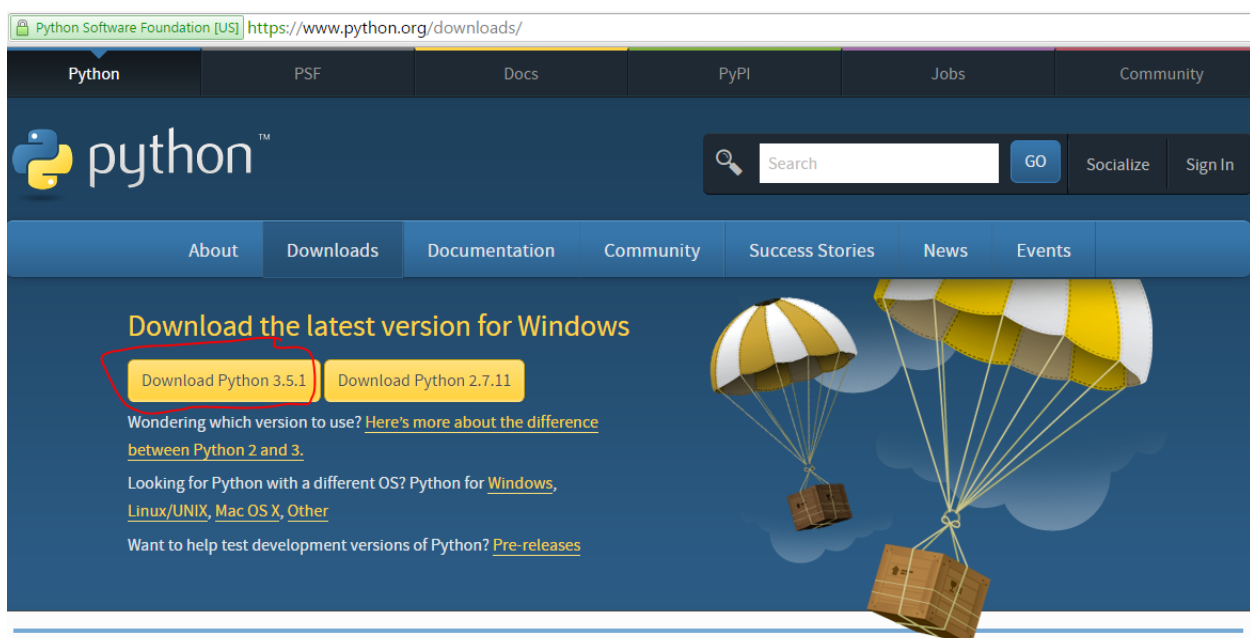
- Cài đặt Python
- Thực hành câu lệnh `print`
- Thực hành phép toán cơ bản
- Thực hành câu lệnh `raw_input`

1. Cài đặt Python

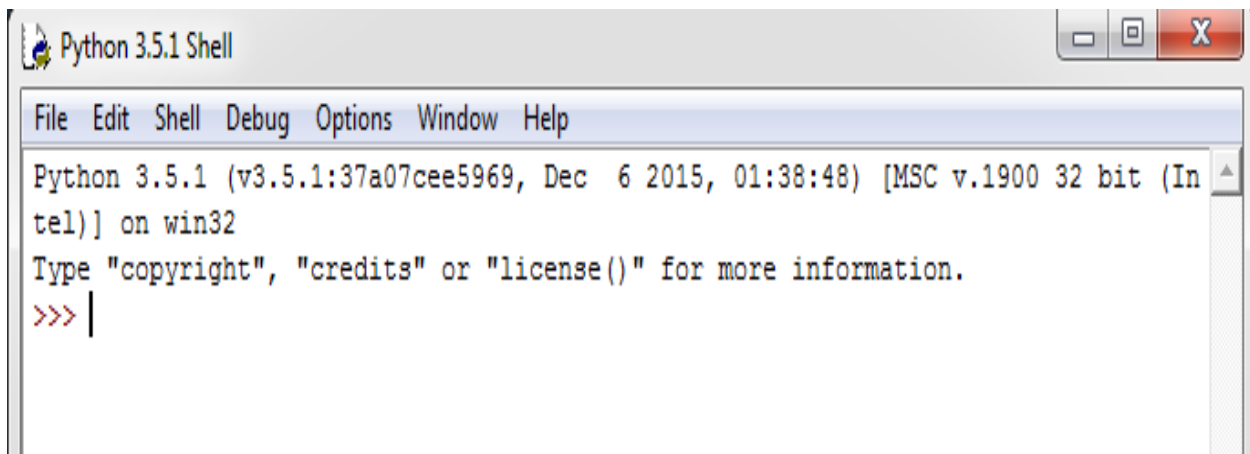
Python có thể chạy trên hệ điều hành Window/Linux, Trong phần thực hành này sinh viên sẽ thực hiện cài đặt Python trên hệ điều hành Window.

Các bước thực hiện:

Bước 1: Download Python tại địa chỉ www.python.org/download

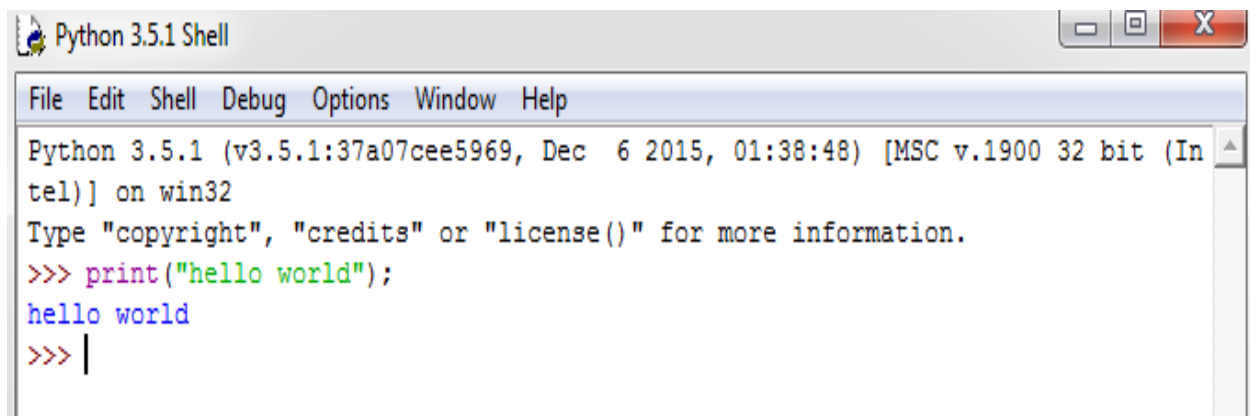


Bước 2: Mở ứng dụng Python Shell (vào menu start > python > python 3.5.1. shell)



Dấu >>>: dấu nhắc lệnh chờ thực hiện.

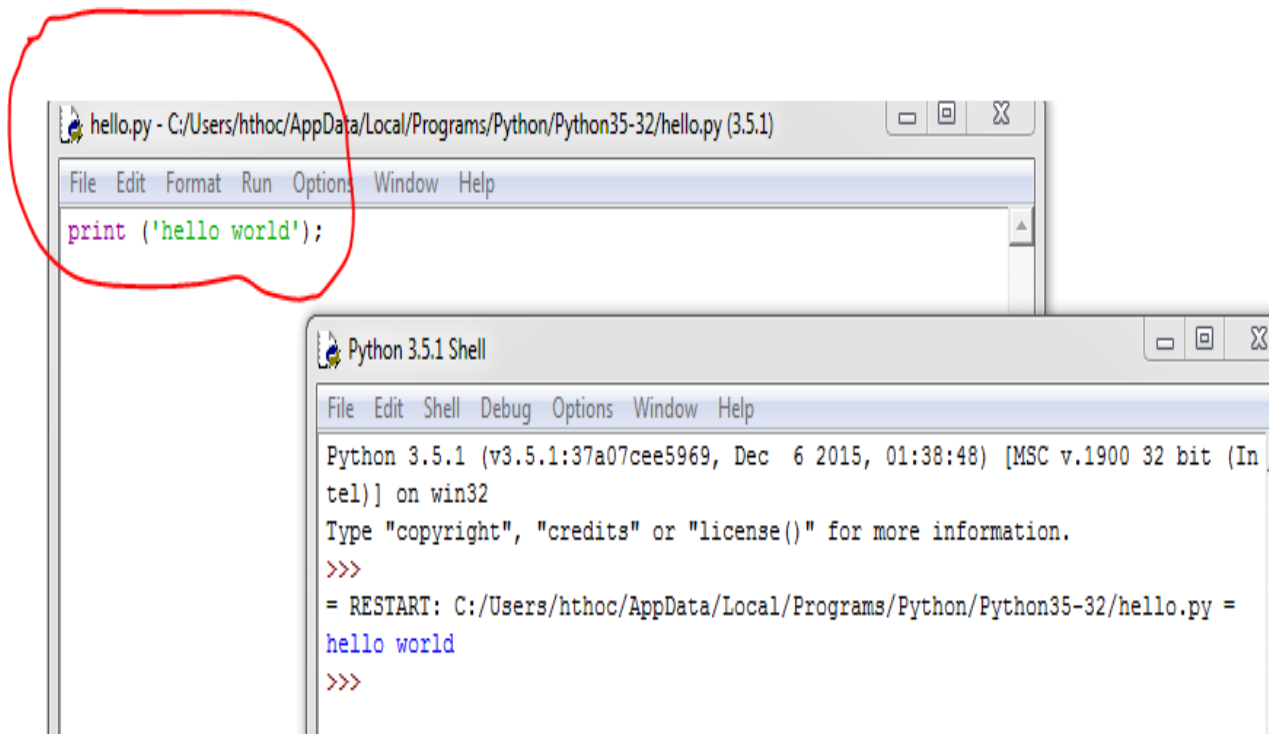
Thí dụ 1: sử dụng câu lệnh print để hiển thị chuỗi "hello world"



2. Viết chương trình

Ghi câu lệnh trực tiếp ngay tại dấu nhắc lệnh (của chương trình Python shell) là cách thông thường để thực hiện những thao tác đơn giản, nhưng sẽ ít được dùng khi chương trình có nhiều câu lệnh.

Trong trường hợp này, cách phù hợp nhất là sử dụng trình soạn thảo (vào menu File của Python shell > new File) để viết chương trình, lưu chương trình lại với định dạng *.Py, chương trình này còn được gọi là Script.



3. Câu lệnh print

Được dùng để hiển thị dữ liệu

Cú pháp:

Cú pháp 1:

```
print ("chuỗi hiện thị")
```

Thí dụ 2:

```
print ("This line will be printed.")
```

Kết quả hiển thị:

This line will be printed.

Cú pháp 2:

```
print ("chuỗi hiện thị", biểu thức/gia trị, tham số)
```

Thí dụ 3:

```
print ("2 + 2 is", 2 + 2)
print ("3 * 4 is", 3 * 4)
print ("100 - 1 is", 100 - 1)
print ("(33 + 2) / 5 + 11.5 is", (33 + 2) / 5 + 11.5)
```

Kết quả hiển thị:

```
2 + 2 is 4
3 * 4 is 12
100 - 1 is 99
(33 + 2) / 5 + 11.5 is 18.5
```

Lưu ý: Với phiên bản Python 2.7, cú pháp hiển thị như sau:

```
print "chuỗi hiển thị"
```

4. Biến và kiểu dữ liệu

Biến: Biến (variable) là tên được đặt trong bộ nhớ máy tính, người lập trình có thể sử dụng biến để lưu trữ dữ liệu và để truy vấn dữ liệu khi cần thiết.

Kiểu dữ liệu: kiểu dữ liệu (datatype hay type) là một cách phân loại dữ liệu cho trình biên dịch (compiler) hoặc thông dịch (interpreter) hiểu người lập trình muốn sử dụng dữ liệu. Hầu hết các ngôn ngữ hỗ trợ nhiều kiểu dữ liệu khác nhau như kiểu số thực, kiểu số nguyên, kiểu luận lý (Boolean), kiểu chuỗi, kiểu danh sách, ... Trong bài này chúng ta sẽ tìm hiểu kiểu dữ liệu số và chuỗi. Trong các bài tiếp theo chúng ta sẽ

có dịp tìm hiểu và sử dụng các kiểu dữ liệu Boolean, danh sách và các kiểu dữ liệu khác.

Kiểu số:

Python hỗ trợ hai kiểu số cơ bản: Số nguyên (integer) và số thực (floating point)

Thí dụ 4: Số nguyên

```
myint = 7  
  
print (myint)
```

Trong thí dụ trên, **myint** là biến và **7** là giá trị được gán vào cho biến **myint**. Như vậy, khi người lập trình muốn sử dụng giá trị **7** thì có thể gọi thông qua biến **myint**.

Thí dụ 5: Số thực

```
myfloat = 7.0  
  
print (myfloat)  
  
myfloat = float (7) //ép số nguyên 7 về số thực 7.0  
  
print (myfloat)
```

Kiểu chuỗi:

Chuỗi là một tập hợp gồm nhiều "ký tự" liên tiếp nhau, được thể hiện/mô tả trong dấu nháy đơn hoặc dấu nháy kép

Thí dụ 6:

```
mystring = 'hello world' //dấu nháy đơn  
  
print(mystring)  
  
Kết quả: hello world  
  
mystring = "hello world" // dấu nháy kép
```

```
print(mystring)

Kết quả: hello world

mystring = "Don't worry about apostrophes"

print (mystring)

Kết quả: Don't worry about apostrophes

mystring = 'Don\'t worry about apostrophes'

print (mystring)

Kết quả: Don't worry about apostrophes
```

* Lưu ý

1. Để sử dụng ký tự đặc biệt (dấu nháy đơn, dấu nháy kép, ...), ta dùng dấu \ và ký tự đặc biệt đó. Xem bảng tóm tắt dưới đây:

Ký hiệu	Ý nghĩa
\n	Dòng mới
\t	Phím tab
\\	\
\'	'
\"	"

2. Trường hợp muốn xóa một biến không còn dùng nữa ta có thể dùng lệnh del như thí dụ dưới đây:

```
1. myfloat = 7.0

2. print (myfloat)

3. del myfloat

4. print (myfloat)
```

Kết quả:

7.0

```
Traceback (most recent call last):  
  File  
"C:\Users\hohuynh\AppData\Local\Programs\Python\Python37  
\test.py", line 4, in <module>  
    print (myfloat)  
NameError: name 'myfloat' is not defined
```

Câu lệnh ở dòng số 4 sẽ báo lỗi "name 'myfloat' is not defined". Lý do là ta đã xóa biến myfloat ở dòng lệnh số 3.

3. Muốn xem một biến thuộc vào kiểu dữ liệu nào, ta sử dụng hàm **type**(biến) như sau:

Thí dụ:

```
>>> i = 42  
  
>>> type(i)  
<type 'int'>  
  
>>> f = float(i)  
  
>>> type(f)  
<type 'float'>
```

4. Kiểm tra vùng lưu trữ giá trị của biến: Để hiểu rõ hơn bản chất dữ liệu của biến, thí dụ biến số thực và biến số nguyên, ta có thể kiểm tra vùng lưu trữ giá trị của biến dựa theo thí dụ dưới đây:

```
import sys  
  
print(sys.int_info)    #Thông tin chi tiết của số nguyên  
print(sys.float_info) #Thông tin chi tiết của số thực
```

Kết quả:

```
sys.int_info(bits_per_digit=30, sizeof_digit=4)

sys.float_info(max=1.7976931348623157e+308,
max_exp=1024, max_10_exp=308, min=2.2250738585072014e-
308, min_exp=-1021, min_10_exp=-307, dig=15,
mant_dig=53, epsilon=2.220446049250313e-16, radix=2,
rounds=1)
```

5. Hiện thị chuỗi theo định dạng

Python sử dụng định dạng chuỗi theo chuẩn C (chuẩn ngôn ngữ lập trình C) để hiện thị chuỗi.

Hiện thị chuỗi: %s

Hiện thị số: %d

Thí dụ 7:

```
name = "John"

print ("My name is, %s!" % name)

age = 18

print ("I am %d years old" % age)
```

Thí dụ 8:

```
name = "John"

age = 23

print ("%s is %d years old." % (name, age))
```

6. Phép toán**a. Phép toán số học**

Phép toán	Toán tử	Ví dụ (python3.x)
Lũy thừa	<code>**</code>	<code>5 ** 2 == 25</code>
Nhân	<code>*</code>	<code>2 * 3 == 6</code>
Chia	<code>/</code>	<code>14 / 3 == 4.666666666666667</code>
Chia lấy phần nguyên	<code>//</code>	<code>14 // 3 == 4</code>
Chia lấy phần dư	<code>%</code>	<code>14 % 3 == 2</code>
Cộng	<code>+</code>	<code>1 + 2 == 3</code> <code>"hello" + " " + "world" =</code> <code>hello world</code>
Trừ	<code>-</code>	<code>4 - 3 == 1</code>

b. Phép gán

Toán tử	Ý Nghĩa	Ví dụ (python3)
<code>=</code>	Gán giá trị bên vế phải cho vế trái	<code>x = 5</code>
<code>+=</code>	Cộng và gán	<code>x+=5</code> <code>(x = x + 5)</code>
<code>-=</code>	Trừ và gán	<code>x-=5</code> <code>(x = x - 5)</code>
<code>*=</code>	Nhân và gán	<code>x*=5</code> <code>(x = x * 5)</code>
<code>/=</code>	Chia và gán	<code>x/=5</code> <code>(x = x/5)</code>

//=	Chia và gán lấy nguyên	x//=5 (x = x // 5)
%=	Chia và gán lấy dư	X%=5 (x = x % 5)
=	Lấy lũy thừa và gán	X=5 (x = x ** 5)

7. Câu lệnh nhập từ bàn phím

Python 2.x:

```
raw_input (prompt)
```

Python 3.x:

```
input (prompt)
```

Hàm `raw_input ()` hoặc `input ()` luôn trả về dữ liệu kiểu chuỗi. Nghĩa là nếu bạn muốn nhập một số (không phải là chuỗi) bạn phải gọi đến hàm chuyển đổi kiểu số nguyên hoặc số thực tương ứng

Thí dụ 9:

```
name = input ("What's your name? ")
print ("Welcome, %s!" % name)
age = input ("How old are you? ")
print ("Next year you are %d" % int(age) + 1)
```


Lưu ý:

`int (chuỗi):` chuyển từ chuỗi sang số nguyên

`float (chuỗi):` chuyển từ chuỗi sang số thực

BÀI TẬP CHƯƠNG 1

Bài 1.1 Viết chương trình in ra màn hình dòng chữ Hello World

Bài 1.2 Viết chương trình nhập nhiệt độ F và chuyển sang nhiệt độ C theo công thức:

$C = 5 * (F - 32) / 9$, với C : nhiệt độ C ; F : nhiệt độ F

Bài 1.3 Viết chương trình nhập vào 2 số nguyên, in ra màn hình tổng bình phương của 2 số nguyên đó.

Công thức: $S = a*a + b*b$

Bài 1.4 Viết chương trình nhập vào 2 số nguyên, in ra màn hình hiệu bình phương của 2 số nguyên đó.

Công thức: $S = a*a - b*b$

Bài 1.5 Viết chương trình nhập một số gồm 2 chữ số, in ra màn hình số hàng đơn vị và số hàng chục

Thí dụ số nhập vào là 13, in ra màn hình số hàng đơn vị là 3 và số hàng chục là 1

Bài 1.6 Viết chương trình nhập một số gồm 2 chữ số, in ra màn hình số đảo ngược

Thí dụ số nhập vào là 13, in ra màn hình số đảo ngược là 31

Bài 1.7 Viết chương trình nhập một giá trị tùy ý (chuỗi, số nguyên, số thực), sau đó in ra màn hình giá trị nhập được và cho biết kiểu dữ liệu của nó

Thí dụ nhập vào số 50, in ra màn hình số vừa nhập là 50 và kiểu dữ liệu là số nguyên.

Bài 1.8 Viết chương trình nhập vào tiền lương chính, phụ cấp và số ngày đi làm trong tháng, in ra màn hình Lương chính nhận được theo công thức:

Lương chính nhận được =	Tiền lương chính + Phụ cấp (nếu có))	X Số ngày công đi làm
	22	

Bài 2: BIỂU THỨC ĐIỀU KIỆN

Mục tiêu:

- Làm quen với Biểu thức điều kiện 1 chiều, 2 chiều
- Làm quen phép toán so sánh
- Làm quen với cách viết Script theo biểu thức điều kiện
- Cấu trúc bất ngoại lệ

Nội dung chính:

- Điều kiện *if*
- Điều kiện *if ... else*
- Điều kiện *if ... elif ... else*
- Biểu thức điều kiện
- *try ... except*

1. Phép toán so sánh

Phép toán	Ký hiệu	Sử dụng	Kết quả
So sánh bằng	<code>==</code>	<code>a == b</code>	True/False
So sánh khác	<code>!=</code>	<code>a != b</code>	True/False
So sánh nhỏ hơn	<code><</code>	<code>a < b</code>	True/False
So sánh nhỏ hơn hay bằng	<code><=</code>	<code>a <= b</code>	True/False
So sánh lớn hơn	<code>></code>	<code>a > b</code>	True/False
So sánh lớn hơn hay bằng	<code>>=</code>	<code>a >= b</code>	True/False
IS	<code>is</code>	<code>a is b</code>	Trả về true nếu các biến ở hai bên toán tử cùng trỏ tới một đối tượng (hoặc cùng giá trị), ngược lại là false
IS Not	<code>is not</code>	<code>a is not b</code>	Ngược lại với is

Thí dụ 1:

Thí dụ	Kết quả
5 == 4	False
5 != 4	True
5 < 4	False
5 <= 4	False
5 > 4	True
5 >= 4	True

Thí dụ 2:

```
x=5  
y=5  
print(x is y)  
Kết quả: True
```

Thí dụ 3:

```
x=5  
y=5  
print(x is not y)  
Kết quả: False
```

2. Phép toán and và or

Phép toán	Giá trị
True and True	True
True and False	False
False and True	False
False and False	False
True or True	True
True or False	True
False or True	True
False or False	False

3. Độ ưu tiên toán tử

Python có ràng buộc thứ tự ưu tiên của các toán tử. Bảng dưới đây trình bày thứ tự ưu tiên từ cao đến thấp.

Thứ tự ưu tiên	Toán tử	Miêu tả
1	**	Toán tử mũ
2	* / % //	Phép nhân, chia, lấy phần dư và phép chia lấy phần nguyên
3	+ -	Toán tử Cộng, Trừ
4	<= < > >=	Các toán tử so sánh
5	<> == !=	Các toán tử so sánh
6	= %= /= //= -= += *= **=	Các toán tử gán
7	is, is not	Các toán tử so sánh
8	not, or, and	Các toán tử Logic

Lưu ý: Trong thực tế khi lập trình, để câu lệnh tường minh, rõ ràng, lập trình viên thường dùng cặp ngoặc tròn () để nó rõ nghĩa hơn.

4. Điều kiện if

if Biểu thức điều kiện:

Câu lệnh*

Câu lệnh* được thực hiện khi "Biểu thức điều kiện" có giá trị True.

Thí dụ 2:

```
x = 5

if x < 10:

    print ("Smaller")

if x > 20:

    print ("Bigger")

print ("Finis")
```

Kết quả:

```
Smaller

Finis
```

5. Điều kiện if ... else

```
if Biểu thức điều kiện:

    Câu lệnh 1

else:

    Câu lệnh 2
```

Câu lệnh 1 được thực hiện khi “Biểu thức điều kiện” có giá trị True, **Câu lệnh 2** được thực hiện khi “Biểu thức điều kiện” có giá trị False.

Thí dụ 3:

```
x = 5

if x < 2:

    print ("Smaller")

else:
```

```
print ("Bigger")

print ("All done")
```

6. Điều kiện if ... elif ... else

```
if Biểu thức điều kiện 1:

    Câu lệnh 1

elif Biểu thức điều kiện 2:

    Câu lệnh 2

...

elif Biểu thức điều kiện n

    Câu lệnh n

else:

    Câu lệnh else
```

Thí dụ 4:

```
x = int (input ('Nhập một số nguyên: '))

if x < 2:

    print 'small'

elif x < 10:

    print 'Medium'

else:

    print 'LARGE'

print 'All done'
```

7. Biểu thức điều kiện

Với biểu thức điều kiện if ... else như thí dụ sau:

```
if a > b:
    max=a
else:
    max=b
```

Bạn có thể viết ngắn gọn như sau:

```
max = a if (a > b) else b
```

Lưu ý: Nếu bạn đang sử dụng các ngôn ngữ lập trình như C/C++; VB, .NET hay Java thì cú pháp trên sẽ là:

```
max = (a > b) ? a : b
```

Thí dụ 5:

```
>>> max = 3 if (3 > 5) else 5
```

Kết quả: 5

8. Cấu trúc try ... catch

Đôi khi câu lệnh xảy ra lỗi trong lúc thi thực như thí dụ dưới đây:

```
astr = input ('Nhập số nguyên: ')
istr = int(astr)
print ('Số vừa nhập ', istr)
```


Nếu script trên khi thực thi, giá trị nhập vào không phải là số thì sẽ báo lỗi như sau:

```
== RESTART: C:/Users/hohuynh/AppData/Local/Programs/Python/Python37/test.py ==  
Nhập số nguyên: hello  
Traceback (most recent call last):  
  File "C:/Users/hohuynh/AppData/Local/Programs/Python/Python37/test.py", line 2  
    , in <module>  
      istr = int(astr)  
ValueError: invalid literal for int() with base 10: 'hello'  
>>>
```

Lý do xảy ra lỗi: Trong thí dụ trên, chuỗi **hello** nhập từ bàn phím được gán cho biến `astr`. Ở dòng tiếp theo, biến `astr` được chuyển thành giá trị số nguyên thông qua hàm `int(astr)`. Tuy nhiên trong trường hợp này biến `astr` không thể nào chuyển sang dạng số được vì giá trị đang là **hello**. Nên khi chương trình thực hiện đến dòng đó sẽ báo lỗi `Invalid literal for int()`, lỗi giá trị chuỗi không hợp lệ để chuyển sang định dạng số.

Vậy khi nào không bị lỗi: Bạn sẽ không bị lỗi trong trường hợp bạn phải nhập đúng giá trị dạng số như minh họa dưới đây khi người dùng nhập vào cho chuỗi `astr` là 5.

```
== RESTART: C:/Users/hohuynh/AppData/Local/Programs/Python/Python37/test.py ==  
Nhập số nguyên: 5  
Số vừa nhập 5
```

Tuy nhiên, chương trình viết như trên sẽ không phù hợp với người dùng, chương trình phải đảm bảo xử lý được tất cả các trường hợp mà có thể xảy ra khi người dùng sử dụng, nghĩa là phải xem xét đến các khả năng mà người dùng xử lý sai và đưa ra các chỉ dẫn phù hợp.

Để xử lý cho trường hợp này ta có thể dùng khối lệnh `try ... except` được hỗ trợ trong python.

Cú pháp như sau:

```
try:  
    Câu lệnh có khả năng sinh ra các ngoại lệ  
  
except:  
    Câu lệnh xử lý trong trường hợp lỗi xảy ra
```

Trở lại thí dụ, giả sử ta mong muốn xử lý lỗi theo hướng mặc định gán giá trị là -1 khi người dùng nhập vào phát sinh lỗi, cú pháp xử lý như sau:

```
astr = input ('Nhập số nguyên: ')\n\ntry:\n    istr = int(astr)\n\nexcept:\n    istr = -1\n\nprint 'Done', istr
```

BÀI TẬP CHƯƠNG 2

Bài 2.1 Không lập trình, hãy xem đoạn script sau và cho biết kết quả:

```
a = 5
b = 7

print ('a > b', a > b)
print ('a < b', a < b)
print ('a >= b', a >= b)
print ('a <= b', a <= b)
print ('a == b', a == b)
print ('a < b or a != b', a < b or a != b)
print ('a > b and a != b', a > b and a != b)
```

Bài 2.2 Cho đoạn script sau:

```
x = a

if x > 2:
    print 'Bigger than 2'
    print 'Still bigger'

print 'Done with 2'
```

a. Theo bạn, với giá trị a bằng bao nhiêu thì đoạn script trên in ra màn hình kết quả:

Bigger than 2

Still bigger

Done with 2

b. Theo bạn, với giá trị a bằng bao nhiêu thì đoạn script trên in ra màn hình kết quả:

Done with 2

Bài 2.3 Viết chương trình nhập vào hai số a, b; giải và biện luận phương trình $ax + b = 0$

Gợi ý: sử dụng hàm `int (string)` để chuyển dữ liệu từ định dạng chuỗi sang định dạng số.

Thí dụ: `a = int (input('Nhập a: '))`

Bài 2.4 Viết chương trình nhập vào ba số a, b, c; giải và biện luận phương trình $ax^2 + bx + c = 0$

Bài 2.5 Viết chương trình nhập vào lương cơ bản, số năm công tác, in ra tiền lương hàng tháng có tính số % phụ cấp thâm niên theo công thức:

Lương hàng tháng = lương cơ bản + phụ cấp thâm niên

Nếu số năm công tác dưới 5 năm: phụ cấp thâm niên 0%

*Nếu số năm công tác trên 5 năm, phụ cấp thâm niên được tính theo % số năm công tác. Thí dụ người A có 6 năm thâm niên, vậy phụ cấp thâm niên của A là 6% * lương cơ bản.*

Bài 2.6 Viết chương trình nhập vào năm sinh, in ra tuổi. Lưu ý nếu năm sinh nhập vào không hợp lệ (giá trị âm, hoặc lớn hơn năm hiện tại, hoặc giá trị nhập vào là chuỗi) thì hiện thị ra thông báo đồng thời gán giá trị mặc định là năm hiện tại.

Bài 2.7 Giá cước taxi được tính theo công thức: 1km đầu tiên tính từ lúc lên xe: 13000 đồng. Mỗi km tiếp theo được tính theo đơn giá là 12000 đồng. Từ km 30 trở lên, mỗi km được tính là 9000 đồng.

Hãy viết chương trình nhập vào số km đã đi và in ra giá tiền cần phải thanh toán.

Bài 2.8 Tổng số giờ làm việc được quy định cho một nhân viên trong tuần là 40 giờ, nếu người lao động làm nhiều hơn số giờ quy định, thì mỗi giờ hơn sẽ được tính là 1.5 giờ. Hãy nhập vào tiền lương của 1 giờ, số giờ làm việc trong tuần, in ra tiền lương sẽ được trả trong tuần.

Bài 2.9 Viết chương trình nhập vào năm sinh (namsinh), in ra tuổi (tuoi). Lưu ý nếu năm sinh nhập vào không hợp lệ (giá trị âm, hoặc lớn hơn năm hiện tại, hoặc giá trị nhập vào là chuỗi) thì hiện thị ra thông báo đồng thời gán giá trị mặc định là năm hiện tại.

Gợi ý: Cần phải bổ sung thư viện thời gian tại đoạn đầu của Script theo cú pháp.

```
import time

today = time.localtime()

tuoi = namsinh - today[0]
```

Bài 3: XÂY DỰNG HÀM

Mục tiêu:

- Làm quen với khai báo và sử dụng hàm
- Làm quen với hàm main

Nội dung chính:

- Định nghĩa hàm
- Xây dựng và sử dụng hàm không trả về kết quả
- Xây dựng và sử dụng hàm có trả kết quả về
- Hàm main
- Hàm đệ quy

1. Định nghĩa hàm

Hàm (Function) là **đoạn chương trình** độc lập, hoàn chỉnh, có thể sử dụng nhiều lần. Hàm cho phép người lập trình (developer) cấu trúc chương trình thành các phân đoạn khác nhau để dễ dàng trong việc sử dụng và chỉnh sửa.

Thí dụ: Viết chương trình nhập vào 4 số nguyên a, b, c, d các giải phương trình sau:

$$ax + b = 0$$

$$cx + d = 0$$

Trường hợp không định nghĩa hàm, để giải 2 phương trình $ax + b = 0$ và $cx + d = 0$ ta phải đi lần lượt giải cho từng phương trình.

```
1 a=int(input("Nhap so a: "))
2 b=int(input("Nhap so b: "))
3 c=int(input("Nhap so c: "))
4 d=int(input("Nhap so d: "))
5 #Giai phuong trinh ax + b = 0
6 if a != 0:
7     x = -float(b)/a
8     print (x)
9 elif b != 0:
10    print ('Phuong trinh vo nghiem')
11 else:
12    print ('Phuong trinh vo so nghiem')
13
14 #Giai phuong trinh cx + d = 0
15 if c != 0:
16     x = -float(d)/c
17     print (x)
18 elif d != 0:
19     print ('Phuong trinh vo nghiem')
20 else:
21     print ('Phuong trinh vo so nghiem')
```

Như hình bên trên, từ dòng 6 đến dòng 12, dùng để giải phương trình $ax + b = 0$; dòng 15 đến dòng 21 giải phương trình $cx + d = 0$.

Nhược điểm:

1. Ta phải lập lại nhiều lần một đoạn chương trình để giải quyết cho cùng 1 dạng của bài toán.
2. Khi cần phải chỉnh sửa vì một số lý do (thí dụ phương pháp giải chưa chính xác chẳng hạn) thì ta cần phải điều chỉnh cho tất cả các đoạn chương trình đó.
3. Số câu lệnh được dùng trong chương trình sẽ tăng lên đáng kể.

2. Hàm không trả về kết quả

```
def TenHam (Danh sách tham số nếu có):  
    Câu lệnh
```

Thí dụ 1:

```
def thing ():  
    print ('Hello')  
    print ('Fun')  
  
thing ()  
  
print ('Zip')  
  
thing ()
```

Kết quả:

```
Hello  
  
Fun  
  
Zip  
  
Hello  
  
Fun
```

3. Hàm trả về kết quả

```
def TenHam (Danh sách tham số nếu có):  
    Câu lệnh  
    return Ketqua
```

Thí dụ 2:

```
def add (x, y):  
    return x + y  
  
z = add (3,5)  
  
print (z)  
  
Kết quả:
```


8

Thí dụ 3:

```
def add (x, y=4):
    return x + y
```

```
z = add (3)
```

```
t = add (3,6)
```

```
print (z)
```

```
print (t)
```

Kết quả:

7

9

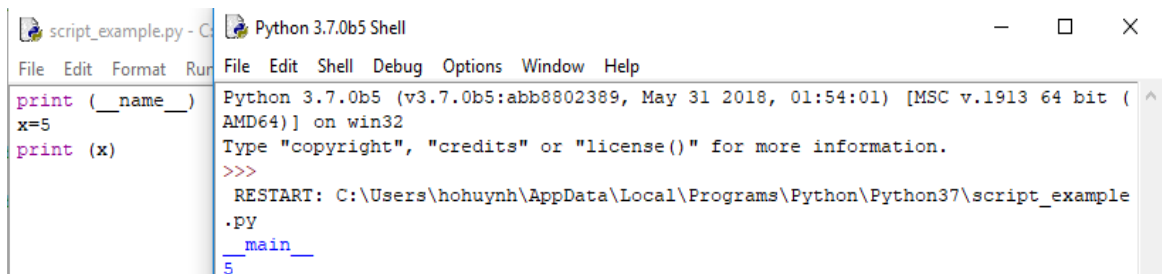
4. Thuộc tính `__name__`

`__name__`: là một thuộc tính mặc định của chương trình python, mỗi một Script python luôn tồn tại thuộc tính `__name__`.

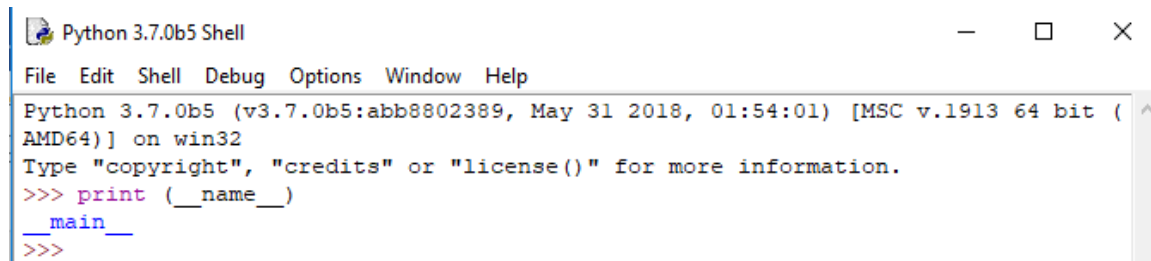
'`__main__`': là một giá trị của thuộc tính `__name__`, giá trị `__main__` mô tả phạm vi mà ở đó các câu lệnh sẽ được thực thi.

Thuộc tính `__name__` được gán là `__main__` trong các trường hợp:

- Đọc từ thiết bị nhập chuẩn (bàn phím)
- Đọc từ script



- Đọc từ dấu nhắc lệnh (interactive prompt)



```
Python 3.7.0b5 Shell
File Edit Shell Debug Options Window Help
Python 3.7.0b5 (v3.7.0b5:abb8802389, May 31 2018, 01:54:01) [MSC v.1913 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print (__name__)
__main__
>>>
```

```
if __name__ == "__main__":
    # execute only if run as a script
    main ()
```

5. Hàm đệ quy

Một số ngôn ngữ lập trình Python, C/C++, ... cho phép một hàm được gọi tới chính nó. Kỹ thuật này được gọi là Đệ qui (Recursion), trong toán học gọi là quy nạp. Trong đệ qui, một hàm a có thể gọi trực tiếp chính hàm a.

Thí dụ 4: Trong toán học để tính giai thừa n, ta có định nghĩa như sau:

$$N! = N * (N - 1)!, \text{ với } 1! = 1; 0! = 1$$

Áp dụng quy tắc này, ta có thể lập trình bằng Python hàm tính N! như sau:

```
def giai_thua(n):
    # điều kiện thoát đệ quy
    if n==0 or n==1:
        return 1
    return n * giai_thua(n-1) # gọi đệ quy
print (giai_thua(5))
```

BÀI TẬP CHƯƠNG 3

Bài 3.1 Cho biết kết quả hiện thị của Script sau và giải thích.

```
x = 5

print 'Hello'

def print_lyrics ():
    print "I'm a lumberjack, and I'm okay."
    print 'I sleep all night and I work all day.'

print 'Yo'

x = x + 2

print x
```

Bài 3.2 Viết hàm với tham số truyền vào là năm sinh, sử dụng hàm vừa cài đặt, nhập vào năm sinh và in ra tuổi:

Thí dụ nhập 1984 in ra: Ban sinh năm 1984, vay ban 19 tuoi.

Bài 3.3 Viết hàm với tham số truyền vào là nhiệt độ F, trả về kết quả nhiệt độ C theo công thức. Sử dụng hàm vừa cài đặt, nhập vào độ F và in ra màn hình độ C.

$C = 5 * (F - 32) / 9$, với C: nhiệt độ C; F: nhiệt độ F

Bài 3.4 Viết hàm với tham số truyền vào là một tháng và trả về mùa tương ứng trong năm. Sử dụng hàm vừa cài đặt, nhập vào một tháng và in ra màn hình mùa trong năm.

Thí dụ: Người dùng nhập vào tháng 2, in ra màn hình là mùa Xuân.

Từ tháng 1 đến tháng 3: Mùa Xuân

Từ tháng 4 đến tháng 6: Mùa Hạ

Từ tháng 7 đến tháng 9: Mùa Thu

Từ tháng 10 đến tháng 12: Mùa Đông

Bài 3.5 Viết hàm tìm số lớn nhất của hai số nguyên a và b; sử dụng hàm vừa cài đặt, nhập vào 3 số nguyên a, b, c và tìm số lớn nhất trong 3 số đó.

Thí dụ: Người dùng nhập vào ba số 5, 9, 4. In ra màn hình số lớn nhất là 9.

Bài 3.6 Viết hàm tính diện tích hình tròn với tham số truyền vào là bán kính; sử dụng hàm vừa cài đặt, nhập vào bán kính và in ra màn hình diện tích hình tròn.

Công thức tính diện tích hình tròn:

$$S = \text{PI} * R * R$$

Với `PI = math.pi` (sử dụng thư viện `math` bằng cách gọi lệnh `import math` ở đầu script)

Bài 4: VÒNG LẶP

Mục tiêu:

- Làm quen với các vòng lặp
- Vùng dữ liệu range
- Làm quen với lệnh *break*, *continue*

Nội dung chính:

- Vòng lặp *while*
- Vòng lặp *for* với *range*
- Vòng lặp *for* và mảng
- Vòng lặp lồng nhau
- Lệnh *break* trong vòng lặp
- Lệnh *continue* trong vòng lặp
- Lệnh *else* với vòng lặp

1. Vòng lặp *while*

a. Cấu trúc vòng lặp *while*

```
while điều kiện:  
    khối lệnh
```

Thí dụ 1:

```
#Tạo một biến x và gán giá trị cho x là 1  
x=1  
#Nếu x < 10 thì thực hiện khối lệnh  
while(x < 10) :  
    print("x=",x)
```

```
x = x+1 # tăng biến đếm  
print("Lệnh nằm ngoài vòng lặp while")
```

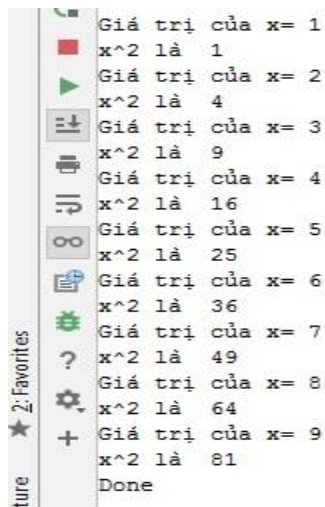
b. Vòng lặp while và lệnh else

Vòng lặp **while** có thể kết hợp với lệnh **else**. Lệnh **else** sẽ được thực thi nếu vòng lặp while chạy và kết thúc bình thường, không bị ngắt bởi lệnh **break**

Thí dụ 2:

```
x=1  
while x< 10:  
    print("Giá trị của x=",x)  
    print("x^2 là ",x*x)  
    x=x+1  
else:  
    print("Done")
```

Kết quả:



```
Giá trị của x= 1  
x^2 là 1  
Giá trị của x= 2  
x^2 là 4  
Giá trị của x= 3  
x^2 là 9  
Giá trị của x= 4  
x^2 là 16  
Giá trị của x= 5  
x^2 là 25  
Giá trị của x= 6  
x^2 là 36  
Giá trị của x= 7  
x^2 là 49  
Giá trị của x= 8  
x^2 là 64  
Giá trị của x= 9  
x^2 là 81  
Done
```

2. Vòng lặp for

a. Cấu trúc vòng lặp for

```
for biến_vòng_lặp in dãy:  
    khối_lệnh
```

Thí dụ 3:

```
for i in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]:  
    print("Hello World!")
```

Thí dụ 4:

```
for char in "Hello Python":  
    print(char)
```

b. Hàm range

Hàm **range** được sử dụng để tạo ra một danh sách chứa các dãy số.

Cú pháp:

```
range([ star ], stop, [ step ] )
```

star : số bắt đầu của chuỗi

stop: là giới hạn mà chuỗi đạt tới (không bao gồm số này)

step: là số bước tăng lên của một số trong chuỗi so với số trước đó

Thí dụ 5:

```
range(4)          # [ 0,1,2,3 ]  
range(4,7)        # [4,5,6]  
range(3,20,2)     #[3,5,7,9,11,13,15,17,19 ]
```

c. Vòng lặp for và hàm range**Thí dụ 6:**

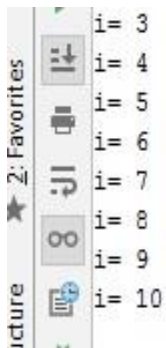
```
for i in range(5):  
    print("i=",i)
```

Kết quả

```
>>> runfile('C:/Users/Administrator/Pyt  
i= 0  
i= 1  
i= 2  
i= 3  
i= 4
```

Thí dụ 7:

```
for i in range(3,11):  
    print("i=",i)
```

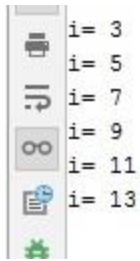
Kết quả

```
i= 3  
i= 4  
i= 5  
i= 6  
i= 7  
i= 8  
i= 9  
i= 10
```

Thí dụ 8:

```
for i in range(3,15,2):  
    print("i=",i)
```

Kết quả



```
i= 3
i= 5
i= 7
i= 9
i= 11
i= 13
```

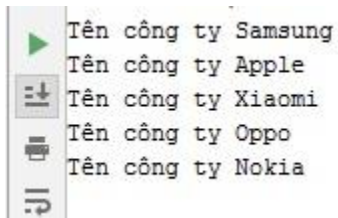
d. Vòng lặp for và mảng

Thí dụ 9:

```
#Khai báo một mảng
congty = ["Samsung", "Apple", "Xiaomi", "Oppo", "Nokia"]

for i in congty:
    print("Tên công ty", i)
```

Kết quả:



```
Tên công ty Samsung
Tên công ty Apple
Tên công ty Xiaomi
Tên công ty Oppo
Tên công ty Nokia
```

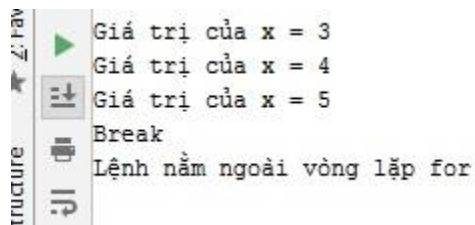
e. Vòng lặp for và lệnh else

```
for x in range(3,7):
    print("Giá trị của x =", x)
    if(x==5):
        print("Break")
        break

else:
    print("Lệnh else sẽ không được thực hiện")
    # do lệnh break trong vòng lặp for nên lệnh else
    không được thực hiện
```

```
print("Lệnh nằm ngoài vòng lặp for")
```

Kết quả:



```
Giá trị của x = 3
Giá trị của x = 4
Giá trị của x = 5
Break
Lệnh nằm ngoài vòng lặp for
```

3. Vòng lặp lồng nhau

Vòng lặp lồng nhau của while

```
while điều kiện:
    while điều kiện:
        Các lệnh
    Các lệnh
```

Vòng lặp lồng nhau của for

```
for biến_vòng_lặp in dãy:
    for biến_vòng_lặp in dãy:
        Các lệnh
    Các lệnh
```

Thí dụ 10:

```
for i in range(0,10):
    for j in range(i,10):
        print(j, end = " ")
    print ("")
```

Kết quả:



```

0 1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
2 3 4 5 6 7 8 9
3 4 5 6 7 8 9
4 5 6 7 8 9
5 6 7 8 9
6 7 8 9
7 8 9
8 9
9

```

Thí dụ 11:

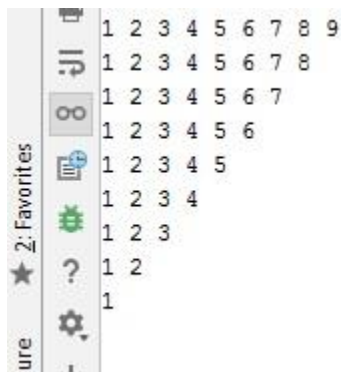
```

i = 1

while(i <= 10):
    j = 1
    while (j <= 10 - i):
        print(j, end = " ")
        j += 1
    print("")
    i += 1

```

Kết quả:



```

1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7
1 2 3 4 5 6
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1

```

4. Lệnh break trong vòng lặp

Lệnh break là lệnh ngừng vòng lặp

Sử dụng lệnh break trong vòng lặp while

```
while điều_kiện
```

```
#code trong vòng lặp while

if điều_kiện

    break

# code trong vòng lặp while

#code bên ngoài vòng lặp while
```

Sử dụng lệnh break trong vòng lặp for

```
for biến_đếm in dãy:

    #code trong vòng lặp

    if điều_kiện

        break

    #code trong vòng lặp

#code ngoài vòng lặp
```

Thí dụ 12:

Sử dụng lệnh **break** để kết thúc vòng lặp ở một thời điểm nào đó. Vòng lặp sẽ bị dừng lại khi biến *i* lớn hơn 7

```
for i in range(1,10):
    if i > 7:
        break
    print(i)
```

Kết quả:



5. Lệnh continue trong vòng lặp

Lệnh continue để bỏ qua các câu lệnh còn lại trong khối lệnh (block) và kiểm tra lại điều kiện trước khi thực thi lại khối lệnh

Thí dụ 13:

Python sẽ bỏ qua vòng lặp khi gặp lệnh continue. Kết quả sẽ hiển thị 0,1,2,3,4

```
i = 0
while i < 10:
    if (i == 5):
        continue
    print(i)
    i += 1
```

Kết quả:



6. Lệnh pass

Lệnh pass là lệnh Null (không làm gì cả), sử dụng để đánh dấu, nhắc nhở thêm code sau. Khác biệt giữa chú thích

(comment) và lệnh pass trong Python là bộ thông dịch sẽ bỏ qua toàn bộ chú thích, còn với lệnh pass thì không bỏ qua. Tuy vậy không có gì xảy ra khi lệnh pass được thực thi.

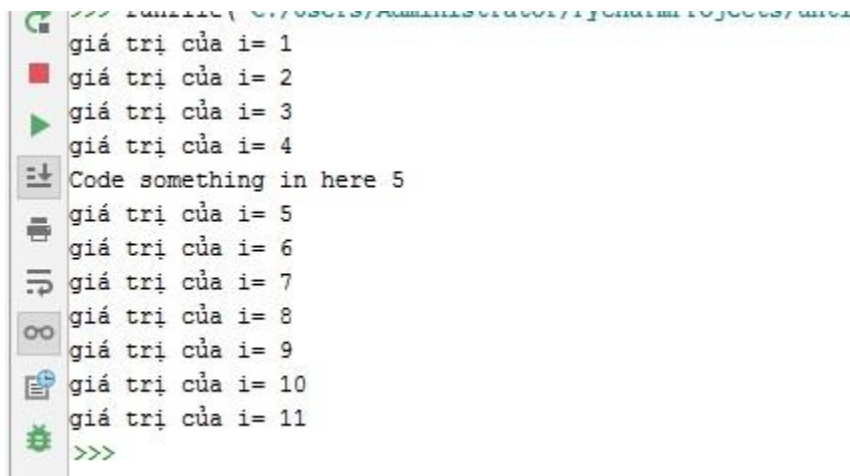
Thí dụ 14:

```
i=0
for i in range(15):
    i=i+1

    if (i==5):
        print("Code something in here", i )
        pass
    print("giá trị của i=", i)

print("Ngoài vòng lặp for")
```

Kết quả:



```
giá trị của i= 1
giá trị của i= 2
giá trị của i= 3
giá trị của i= 4
Code something in here 5
giá trị của i= 5
giá trị của i= 6
giá trị của i= 7
giá trị của i= 8
giá trị của i= 9
giá trị của i= 10
giá trị của i= 11
>>>
```

BÀI TẬP CHƯƠNG 4

Bài 4.1: Sử dụng vòng lặp for để viết lại đoạn code sau

```
a = 0

while a < 100:

    print(a)

    a += 1

print()
```

Bài 4.2: Nhập vào một số, tìm tổng các số lẻ từ 1 đến n

Thí dụ Nhập n=20

In ra tổng số lẻ Sum= 1+2+3+5+ ... + 19

Bài 4.3 Tính tổng các số nguyên tố từ 1 đến 100. Số nguyên tố là số tự nhiên CHỈ chia hết cho 1 và chính nó.

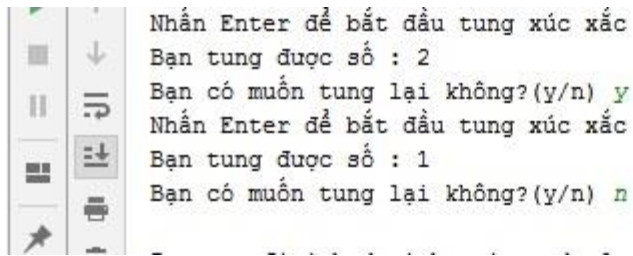
Thí dụ: Số 3 là số nguyên tố, số 9 không phải là số nguyên tố

Sum= 1+3+5+7+11+13+17+ + 97

Bài 4.4: Viết chương trình tung xúc xắc. Đưa ra kết quả ngẫu nhiên (từ 1 đến 6) cho người dùng bằng cách sử dụng vòng lặp while. Vòng lặp sẽ tiếp tục cho đến khi người dùng ngừng lại.

Gợi ý: `num = random.randint(1,6)`

Thí dụ: Nhấn Enter để bắt đầu tung xúc xắc.



```

Nhấn Enter để bắt đầu tung xúc xắc
Bạn tung được số : 2
Bạn có muốn tung lại không?(y/n) y
Nhấn Enter để bắt đầu tung xúc xắc
Bạn tung được số : 1
Bạn có muốn tung lại không?(y/n) n

```

Bài 4.5 Nhập vào hai số x và n . Tính dãy số

$$S(x, n) = x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots + \frac{x^n}{n!}$$

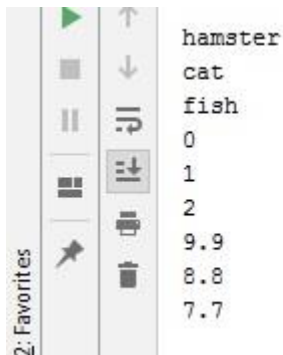
Thí dụ nhập $x=2, n=3$

In ra kết quả của $S(2, 3) = 2 + \frac{2^2}{2!} + \frac{2^3}{3!}$

Bài 4.6 Cho mảng `list_of_lists = [['hamster', 'cat', 'fish'], [0, 1, 2], [9.9, 8.8, 7.7]]`

Sử dụng vòng lặp `for` lồng nhau để in ra từng item

Kết quả:



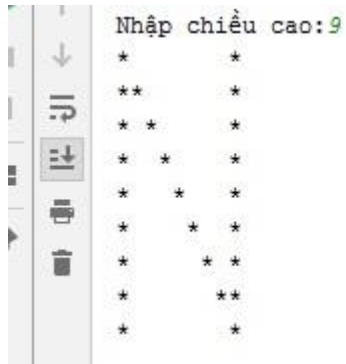
```

hamster
cat
fish
0
1
2
9.9
8.8
7.7

```

Bài 4.7 Nhập vào một số n , sử dụng vòng lặp lồng nhau để in ra chữ N

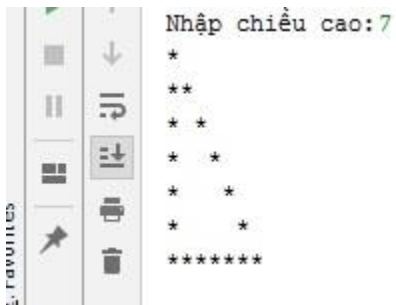
Thí dụ nhập chiều cao là 9. In ra chữ N như sau:



```
Nhập chiều cao: 9
*      *
**     *
* *    *
* *    *
*  *   *
*   *  *
*    * *
*     **
*      *
```

Bài 4.8 Nhập vào số n, viết chương trình in ra hình tam giác

Thí dụ nhập chiều cao là 7, in ra hình tam giác như sau



```
Nhập chiều cao: 7
*
**
* *
*  *
*   *
*    *
*****
```

ÔN TẬP GIỮA KỲ

Bài 1: Viết chương trình nhập vào một số nguyên dương n và đếm số lượng chữ số nguyên dương n

Bài 2: Không sử dụng hàm tính lũy thừa, viết Chương trình nhập vào một số và giá trị lũy thừa, tính lũy thừa theo phương pháp gọi đệ quy

Bài 3: Không sử dụng hàm tính lũy thừa, viết Chương trình nhập vào một số và giá trị lũy thừa, tính lũy thừa theo phương pháp không gọi đệ quy

Bài 4: Viết chương trình nhập vào một số nguyên dương n , tìm chữ số có giá trị lớn nhất của số nguyên dương n .

Bài 5: Viết chương trình nhập vào một số nguyên dương n , đổi sang hệ nhị phân của số nguyên dương n .

Bài 6: Viết chương trình nhập vào một số nhị phân, đổi sang hệ thập phân của số nhị phân vừa nhập.

Bài 7: Viết chương trình nhập vào một số nguyên dương n :

tính $S(n) = 1 + 1/2 + 1/3 + \dots + 1/n$ với $n > 0$

Bài 8: Viết chương trình in ra hình chữ nhật có kích thước $m \times n$ trong hai trường hợp: Hình chữ nhật rỗng (chỉ có đường viền); hình chữ nhật đặc.

Bài 9: Viết chương trình nhập vào số nguyên dương n , hãy đếm chữ số lẻ của số nguyên dương n đó.

Bài 10: Viết chương trình nhập vào số nguyên dương n , hãy đếm chữ số chẵn của số nguyên dương n đó.

Bài 11: Viết chương trình nhập vào số nguyên dương n , viết chương trình tìm số nguyên dương m lớn nhất sao cho:

$$1 + 2 + 3 + \dots + m < n$$

Bài 12: Viết chương trình nhập vào độ y , chuyển sang đơn vị radian theo công thức:

$$x = \frac{\pi}{180}y$$

Sau đó tính $\sin(x)$ với độ chính xác 0.000001 theo công thức:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots + (-1)^n \frac{x^{2n+1}}{(2n+1)!}$$

Bài 13: Viết chương trình nhập vào số nguyên dương n , in ra tất cả các số lẻ nhỏ hơn n trừ ngoại trừ các số 5, 7, 9 nếu có.

Bài 14: Viết chương trình in ra bảng cửu chương 5.

Bài 15: Viết chương trình in ra các ký tự từ A đến Z.

Bài 16: Viết chương trình in ra các ký tự từ a đến z.

Bài 17: Viết chương trình giải phương trình:

$$ax^2 + bx + c = 0$$

Với a, b, c là 3 số thực được nhập vào từ bàn phím.

Bài 18: Viết chương trình nhập vào số nguyên dương n , in ra màn hình tất cả các số là ước số của n .

Bài 19: Viết chương trình nhập vào số nguyên dương n và cho biết các chữ số của số nguyên n có theo thứ tự tăng dần?

Thí dụ: $n = 12345$

Các chữ số của số nguyên n lần lượt là 1, 2, 3, 4, 5 theo thứ tự tăng dần.

Bài 20: Tương tự bài 19, kiểm tra các chữ số của số nguyên n có theo thứ tự giảm dần?

Bài 21: Viết lại tất cả các bài trên theo dạng khai báo hàm.

Bài 5: Danh sách

Mục tiêu:

- *Nắm được khái niệm danh sách trong Python*
- *Biết cách khai báo một danh sách*
- *Các thao tác cơ bản trên danh sách*

Nội dung chính:

- *Khai báo danh sách*
- *Cách thức xử lý danh sách*
- *Các phương thức của danh sách*

1. Tổng quan về danh sách trong Python

Danh sách (List) là một tập hợp các phần tử. Danh sách trong Python có thể chứa các phần tử thuộc các kiểu dữ liệu giống nhau hay khác nhau, nhưng thông thường chúng thuộc cùng một kiểu dữ liệu.

Khai báo danh sách:

Để tạo danh sách, có thể dùng dấu ngoặc vuông

```
List1 = [10, 20, 30, 40]
List2 = ['crunchy frog', 'ram bladder', 'lark vomit']
List3 = ['spam', 2.0, 5, [10, 20]]
```

List1 là danh sách gồm 4 số nguyên.

List2 danh sách gồm 3 chuỗi ký tự

List3 là một danh sách gồm những kiểu dữ liệu khác nhau:

'spam': một chuỗi

2.0: số thực

5: số nguyên

[10,20]: một danh sách.

Một danh sách có thể chứa các phần tử là một danh sách (List3 trong ví dụ trên). Cơ chế này gọi là nested.

Danh sách chứa **0** phần tử gọi là danh sách rỗng. Khai báo danh sách rỗng bằng cặp dấu ngoặc vuông []

```
>>> cheeses = ['Cheddar', 'Edam', 'Gouda']
>>> numbers = [17, 123]
>>> empty = []
>>> print cheeses, numbers, empty
['Cheddar', 'Edam', 'Gouda'] [17, 123] []
```

2. Một số thao tác cơ bản trên danh sách:

- a. Truy xuất đến các phần tử và danh sách con trong một danh sách cho trước

Chỉ số của danh sách hoạt động theo cơ chế sau:

- Chỉ số được biểu diễn bằng một con số nguyên

- Nếu đọc hoặc ghi một phần tử không tồn tại, sẽ thu về kết quả là `IndexError`
- Nếu chỉ số là số âm, nó sẽ đếm ngược từ cuối danh sách

```
>>> squares[0] # trả về phần tử có chỉ số 0
1
>>> squares[-1] # trả về phần tử cuối cùng
25
>>> squares[-3:] # trả về list con, bắt đầu tại vị trí 3 từ cuối list, kết thúc tại cuối list
[9, 16, 25]
>>> squares[:] # trả về tất cả phần tử của list
[1, 4, 9, 16, 25]
```

- b. Kiểm tra một phần tử có trong danh sách hay không:
Dùng toán tử ***in***

```
>>> cheeses = ['Cheddar', 'Edam', 'Gouda']
>>> 'Edam' in cheeses
True
>>> 'Brie' in cheeses
False
```

- c. Cộng (nối) hai list.

```
a = [1, 2, 3]
b = [4, 5, 6]
c = a + b
print c
[1, 2, 3, 4, 5, 6]
```

Tương tự, phép nhân *******, sẽ lặp lại danh sách một số lần.

```
>>> [0] * 4
[0, 0, 0, 0]
>>> [1, 2, 3] * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

- d. Phép gán trong danh sách.

Thay đổi giá trị các phần tử của danh sách

```
>>> cubes = [1, 8, 27, 65, 125]
>>> cubes[3] = 64 # thay giá trị
>>> cubes
```

```
[1, 8, 27, 64, 125]
```

Có thể thực hiện phép gán cho một list con, điều này có thể thay đổi kích thước của list

```
>>> letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>> letters
['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>> # thay đổi một số giá trị
>>> letters[2:5] = ['C', 'D', 'E']
>>> letters
['a', 'b', 'C', 'D', 'E', 'f', 'g']
>>> # xóa chúng
>>> letters[2:5] = []
>>> letters
['a', 'b', 'f', 'g']
>>> # xóa list bằng cách thay tất cả phần tử bằng list rỗng
>>> letters[:] = []
>>> letters
[]
```

e. Chiều dài của list có thể được lấy bằng hàm len()

```
>>> letters = ['a', 'b', 'c', 'd']
>>> len(letters)
4
```

Lưu ý: dù list có thể chứa list, nested list chỉ tính là một phần tử. List sau có bốn phần tử:

```
['spam', 1, ['Brie', 'Roquefort', 'Pol le Veq'], [1, 2, 3]]
```

f. Duyệt list

Để duyệt qua một list, có thể dùng vòng lặp

```
cheeses = ['Cheddar', 'Edam', 'Gouda']
for cheese in cheeses:
    print cheese
```

Để cập nhật nội dung trong list, dùng vòng lặp và chỉ số

```
numbers = [17, 123]
for i in range(len(numbers)):
    numbers[i] = numbers[i] * 2
```

g. Xóa danh sách

Có nhiều cách để xóa.

Xóa một phần tử tại vị trí cho trước:

```
>>> t = ['a', 'b', 'c']
>>> x = t.pop(1)
>>> print t
['a', 'c']
>>> print x
B
```

Xóa và trả về một giá trị tại vị trí cho trước. Nếu hàm pop không truyền vào tham số, phần tử cuối sẽ bị xóa và trả về.

Nếu chỉ muốn xóa mà không cần lấy giá trị trả về:

```
>>> t = ['a', 'b', 'c']
>>> del t[1]
>>> print t
['a', 'c']
```

Nếu biết phần tử muốn xóa mà không biết vị trí:

```
>>> t = ['a', 'b', 'c']
>>> t.remove('b')
>>> print t
['a', 'c']
```

Xóa nhiều phần tử:

```
>>> t = ['a', 'b', 'c', 'd', 'e', 'f']
>>> del t[1:5]
>>> print t
['a', 'f']
```

3. Các phương thức của List

Phương thức	Diễn giải
<code>list.append(x)</code>	Thêm một phần tử vào cuối list
<code>list.extend(iterable)</code>	Thêm các phần tử vào cuối list Ví dụ: <code>x = [1, 2, 3]</code> <code>x.extend([4, 5])</code> <code>print (x)</code> <code>[1, 2, 3, 4, 5]</code>
<code>list.insert(i, x)</code>	Thêm phần tử x vào vị trí i
<code>list.remove(x)</code>	Xóa phần tử đầu tiên trong list có giá trị bằng x. Trả về lỗi nếu không tìm thấy
<code>list.pop([i])</code>	Trả về phần tử tại vị trí i trong list và xóa phần tử đó ra khỏi list
<code>list.clear()</code>	Xóa tất cả phần tử trong list

<code>list.index(x, start, end)</code>	Trả về vị trí của phần tử đầu tiên trong list có giá trị bằng x, trả về lỗi nếu không tìm thấy. Có thể giới hạn phạm vi tìm kiếm bằng cách xác định vị trí bắt đầu start và vị trí kết thúc end
<code>list.count(x)</code>	Trả về số lần xuất hiện của x trong danh sách
<code>list.sort(key=None, reverse=False)</code>	Sắp xếp các phần tử trong list. Mặc định là sắp tăng dần. Có thể sắp xếp giảm dần bằng cách <code>reverse=True</code>
<code>list.reverse()</code>	Đảo ngược các phần tử trong list
<code>list.copy()</code>	Tạo nên bản copy của list

Ví dụ: minh họa cách sử dụng một số phương thức của list

```
>>> fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi',
'apple', 'banana']
>>> fruits.count('apple')
2
>>> fruits.count('tangerine')
0
>>> fruits.index('banana')
3
>>> fruits.index('banana', 4) # Tìm banana tiếp theo bắt đầu
từ vị trí 4
6
>>> fruits.reverse()
>>> fruits
['banana', 'apple', 'kiwi', 'banana', 'pear', 'apple',
'orange']
>>> fruits.append('grape')
>>> fruits
['banana', 'apple', 'kiwi', 'banana', 'pear', 'apple',
'orange', 'grape']
>>> fruits.sort()
>>> fruits
['apple', 'apple', 'banana', 'banana', 'grape', 'kiwi',
'orange', 'pear']
>>> fruits.pop()
'pear'
```

Bài tập chương 5

Bài 5.1

Viết chương trình tính tổng các phần tử trong một danh sách gồm các số thực

Bài 5.2

Viết chương trình tính tổng các số chẵn trong một danh sách gồm các số nguyên

Bài 5.3

Viết chương trình tìm phần tử lớn nhất trong danh sách gồm các số thực

Bài 5.4

Viết chương trình kiểm tra xem 2 hai danh sách có phần tử trùng nhau hay không.

Bài 5.5

Viết chương trình xóa các phần tử trùng trong danh sách

Bài 5.6

Viết chương trình xóa các số chẵn, trong một danh sách các số nguyên

Bài 5.7

Viết chương trình xuất ra các phần tử chung của hai danh sách

Bài 5.8

Viết chương trình để tách một danh sách gồm các số nguyên ra thành hai danh sách: danh sách 1 chứa các số chẵn, và danh sách 2 chứa các số lẻ.

Bài 5.9

Viết chương trình tìm vị trí của một phần tử trong một danh sách cho trước. Nếu phần tử này không có trong danh sách, xuất ra -1

Ví dụ: `a = [1, 3, 5, 7, 9]`

Tìm 3. Xuất ra 1

Tìm 9. Xuất ra 4

Tìm 2. Xuất ra -1

Bài 5.10

Viết chương trình đảo ngược 1 danh sách và xuất ra màn hình.

Ví dụ: `a = [1, 3, 5, 7]`. Output: `[7, 5, 3, 1]`

Bài 6: Chuỗi

Mục tiêu:

- *Làm quen với chuỗi trong python*
- *Hiểu rõ các phương pháp xử lý trên chuỗi.*

Nội dung chính:

- *Khai báo chuỗi*
- *Các thao tác cơ bản trên chuỗi*
- *Các phương thức trên chuỗi*

Kiểu chuỗi đã được sơ lược nhắc đến trong chương 1. Chương này sẽ đi sâu hơn về chuỗi trong Python

1. Cơ bản về chuỗi:

Chuỗi là một dãy các ký tự. Trong Python, chuỗi được khai báo bằng hai cách: nháy đơn hoặc nháy kép.

```
chuoil = 'Hello world'
chuoil2 = "Hello world"
print(chuoil)
print(chuoil2)
```

Kết quả:

```
Hello world
Hello world
```

Để ghi các ký tự đặc biệt, dùng dấu "\" (tham khảo lại chương 1 về các ví dụ)

Ký hiệu	Ý nghĩa
\n	Dòng mới
\t	Phím tab
\\	\
\'	'
\"	"

Nếu không muốn ký hiệu \ thay thế các ký hiệu đặc biệt, có thể dùng r để báo hiệu thuần chuỗi

```
>>> print('C:\some\name') # Ở đây \n nghĩa là xuống dòng
C:\some
ame
>>> print(r'C:\some\name') # Lưu ý cách dung r trước dấu nhắc
C:\some\name
```

Nếu muốn chuỗi trải trên nhiều dòng, dùng 3 dấu nháy: """...""" hay '''...'''

```
print("""\
Usage: thingy [OPTIONS]
    -h                Display this usage message
    -H hostname       Hostname to connect to
""")
```

Có kết quả:

```
Usage: thingy [OPTIONS]
      -h                Display this usage message
      -H hostname       Hostname to connect to
```

Chuỗi có thể được nối với nhau bởi dấu cộng **+**, hoặc lặp lại với dấu nhân *****

```
>>> "Movie: " + "Tora "*3
'Movie: Tora Tora Tora'
```

Hai chuỗi đặt cạnh nhau sẽ tự động ghép lại

```
>>> 'Py' 'thon'
'Python'
```

2. Một số thao tác trên chuỗi.

a. Lấy ký tự trong chuỗi.

Chuỗi là một mảng các ký tự, trong Python chỉ số được đánh dấu từ 0

```
>>>animal = "Tiger"
>>>print(animal[1])
i
>>>print(animal[3])
e
```

b. Chiều dài chuỗi

Để lấy chiều dài chuỗi, dùng hàm **len** có sẵn

```
>>>animal = "Tiger"
>>>len(animal)
5
```

Để lấy phần tử cuối của mảng, lấy chiều dài trừ 1

```
>>>animal = "Tiger"
>>>length = len(animal)
>>>print(animal[length-1])
r
```

Có thể dùng chỉ số âm để lấy phần tử ngược lại từ cuối chuỗi

```
animal = "Tiger"
print(animal[-1]) # Ký tự cuối chuỗi
print(animal[-3]) # Ký tự thứ 3 từ cuối đếm lên
r
g
```

c. Duyệt chuỗi

Có nhiều tính toán liên quan đến việc xử lý từng ký tự trên chuỗi. Có thể dùng vòng lặp while hay for để duyệt từng ký tự

```
animal = "Tiger"
index = 0
while index < len(animal):
    letter = animal[index]
    print (letter)
    index = index + 1
```

Sẽ có kết quả

```
T
i
g
e
r
```

Hoặc

```
animal = "Tiger"
for letter in animal:
    print (letter)
```

Kết quả

```
T
i
g
e
r
```

d. Lấy chuỗi con

[n:m] Lấy chuỗi con bắt đầu từ n đến trước m (không lấy ký tự tại vị trí m)

```
>>>s = "Dinosaurs"
>>>print(s[2:6])
```

```
nosa
```

Nếu bỏ chỉ số *n*, chuỗi con sẽ lấy từ đầu. Nếu bỏ chỉ số sau, chuỗi con sẽ lấy đến cuối

```
>>>s = "Dinosaurs"  
>>>print(s[:3])  
Din  
>>>print(s[4:])  
saurs  
>>>print(s[:])  
Dinosaurs
```

Nếu *n* nhỏ hơn hoặc bằng *m*, chuỗi con sẽ rỗng

```
>>>s = "Dinosaurs"  
>>>print(s[3:3])  
# Rỗng
```

e. Thay đổi nội dung chuỗi

Khi thay đổi nội dung chuỗi tại một vị trí nào đó, không nên dùng toán tử `[]`. Ví dụ:

```
>>> greeting = 'Hello, world!'  
>>> greeting[0] = 'J'  
TypeError: 'str' object does not support item assignment
```

Lỗi này xảy ra do string là đối tượng bất biến, không thể thay đổi nội dung một chuỗi đã tồn tại. Thay vào đó, nên tạo một chuỗi mới:

```
>>> greeting = 'Hello, world!'  
>>> new_greeting = 'J' + greeting[1:]  
>>> print new_greeting  
Jello, world!
```

Ví dụ này tạo nên chuỗi mới, gán ký tự "J" và chuỗi con của chuỗi `greeting` bắt đầu từ 1. Chuỗi gốc không thay đổi.

3. Một số phương thức của chuỗi.

a. Biến thành chữ hoa

Biến một chuỗi thành chuỗi mới viết hoa

```
>>> word = 'banana'  
>>> new_word = word.upper()  
>>> print new_word  
BANANA
```


b. Tìm kiếm chuỗi con

Trả về vị trí xuất hiện của một chuỗi con

```
>>> word = 'banana'
>>> index = word.find('a')
>>> print index
1
>>> word.find('na')
2
```

Có thể quy định bắt đầu tìm kiếm tại vị trí nào bằng tham số thứ 2:

```
>>> word.find('na', 3)
4
```

Và tham số quy định vị trí kết thúc tìm kiếm:

```
>>> name = 'bob'
>>> name.find('b', 1, 2)
-1
```

Ở đây trả về -1, nghĩa là không tìm thấy. Tìm kiếm thất bại và vị trí từ kiểm bắt đầu từ 1, và kết thúc trước 2 (không tính 2)

c. Toán tử in

```
>>> 'a' in 'banana'
True
>>> 'seed' in 'banana'
False
```

Trả về true nếu chuỗi một xuất hiện trong chuỗi hai, và trả về false nếu không xuất hiện.

Ví dụ: đoạn mã sau sẽ in ra tất cả các ký tự trong chuỗi word1 cũng xuất hiện trong chuỗi word2

```
word1 = "apples"
word2 = "oranges"
for letter in word1:
    if letter in word2:
        print letter
```

d. So sánh chuỗi:

Có thể dùng các ký hiệu = < > để so sánh hai chuỗi (thứ tự từ điển)

```
if word == 'banana':
    print 'All right, bananas.'
```

```
if word < 'banana':
    print 'Your word,' + word + ', comes before banana.'
elif word > 'banana':
    print 'Your word,' + word + ', comes after banana.'
else:
    print 'All right, bananas.'
```

Lưu ý: Trong Python, ký hiệu viết hoa sẽ đứng trước (nhỏ hơn) ký hiệu viết thường:

```
Your word, Pineapple, comes before banana.
```

Bảng dưới đây liệt kê các phương thức thường sử dụng trong Python

Phương thức	Diễn giải
string.count(sub, start, end)	Đếm xem trong chuỗi, có bao nhiêu chuỗi con sub . Start: vị trí bắt đầu đếm, mặc định = 0 End: vị trí kết thúc đếm, mặc định = len() VD: string = "conmeoden.com" print(string.count('o')); # Kết quả: 2 print(string.count('o', 3)); # Kết quả: 1
string.find(str, start, end)	Tìm vị trí xuất hiện của chuỗi con str . Trả về - 1 nếu không tìm thấy Start: vị trí bắt đầu tìm, mặc định = 0 End: vị trí kết thúc tìm, mặc định = len()
string.index(str, start, end)	Giống như hàm find, nhưng nếu không tìm thấy sẽ gọi Exception
string.isalnum()	Trả về True, nếu chuỗi chỉ chứa chữ hoặc số. Trả về False nếu có chứa ký hiệu khác

<code>string.isalpha()</code>	Trả về True, nếu chuỗi chỉ chứa chữ. Trả về False nếu có chứa ký hiệu khác
<code>string.isdigit()</code>	Trả về True, nếu chuỗi chỉ chứa số. Trả về False nếu có chứa ký hiệu khác
<code>string.islower()</code>	Trả về True, nếu các ký tự trong chuỗi là chữ thường. Trả về False nếu ngược lại
<code>string.isupper()</code>	Trả về True, nếu các ký tự trong chuỗi là viết hoa. Trả về False nếu ngược lại
<code>string.lower()</code>	Biến chuỗi thành chữ thường
<code>String.upper()</code>	Biến chuỗi thành chữ hoa
<code>string.replace(old,new,max)</code>	Tìm kiếm và thay thế một chuỗi bằng một chuỗi mới
<code>string.split(char, max)</code>	Biến một chuỗi thành một mảng các char

Bài tập chương 6

Bài 6.1

Viết chương trình để tính chiều dài 1 chuỗi.

Bài 6.2

Viết chương trình để lấy về một chuỗi là chuỗi tạo thành bởi hai ký tự đầu tiên và hai ký tự cuối của một chuỗi cho trước. Nếu chuỗi cho trước này có chiều dài nhỏ hơn hai, trả về chuỗi rỗng: Ví dụ: 'w3resource' → 'w3ce', 'w3' → 'w3w3', 'w' → Chuỗi rỗng.

Bài 6.3

Viết chương trình thêm 'ing' vào cuối một chuỗi cho trước. Nếu chuỗi cho trước đã kết thúc bởi 'ing' thêm 'ly' vào cuối. Nếu chuỗi cho trước có độ dài nhỏ hơn 3, để nguyên không thêm gì.

Ví dụ: 'abc' → 'abcing', 'string' → 'stringly'

Bài 6.4

Viết chương trình để xóa ký tự thứ **n** từ một chuỗi không rỗng.

Bài 6.5

Viết chương trình để xóa các ký tự ở vị trí **lẻ**.

Bài 6.6

Viết chương trình để tạo nên một chuỗi mới từ hai chuỗi. Cách nhau bởi khoảng trắng và hoán vị hai vị trí đầu của hai chuỗi.

Ví dụ: input: 'abc', 'xyz', output: 'xyc abz'

Bài 6.7

Cho một danh sách gồm nhiều chuỗi. Xuất ra chuỗi có độ dài lớn nhất.

Bài 6.8

Viết chương trình nhập vào 1 chuỗi. Xuất ra chuỗi đảo ngược.

Ví dụ: 'Orange' → 'egnarO'

Bài 6.9

Viết chương trình xóa tất cả khoảng trắng trong chuỗi

Bài 6.10

Viết chương trình đảo ngược các từ trong chuỗi.

Ví dụ: 'The quick brown fox' → 'fox brown quick The'

Bài 6.11

Viết chương trình xuất ra số ký tự lặp lại trong một chuỗi.

Ví dụ:

Input: thequickbrownfoxjumpsoverthelazydog

Output:

o 4

e 3

u 2

h 2

r 2

t 2

Bài 6.12

Viết chương trình đảo vị trí dấu chấm và dấu phẩy trong câu.

Ví dụ: '32.054,23' → '32,054.23'

Bài 6.13

Viết chương trình, đếm và xuất ra các nguyên âm trong một chuỗi.

Bài 6.14

Viết chương trình để xóa ký tự lặp trong một chuỗi.

Bài 8: TỪ ĐIỂN

Mục tiêu:

- Tổ chức dữ liệu với kiểu dữ liệu Từ Điển
- Thực hiện các thao tác trên Từ Điển
- Vận dụng các hàm xây dựng sẵn của Từ Điển

Nội dung chính:

- Khai báo và sử dụng kiểu dữ liệu Từ Điển
- Chuyển đổi kiểu dữ liệu khác thành Từ Điển
- Duyệt trong Từ Điển
- Các lệnh thường dùng

1. Khai báo dữ liệu

Từ điển là một tập hợp dữ liệu tương tự như kiểu danh sách nhưng chúng không có thứ tự, được đánh chỉ mục và có thể thay đổi dữ liệu bên trong nó. Điều này đồng nghĩa với việc ta có thể chứa và truy xuất dữ liệu bởi các khóa thay vì vị trí như trong danh sách (list).

Mỗi phần tử trong từ điển là một cặp (khóa, giá trị). Khóa được sử dụng để có thể truy xuất giá trị dữ liệu sau này.

Để tạo từ điển, ta đặt các phần tử trong dấu ngoặc nhọn, mỗi phần tử cách nhau bằng dấu phẩy. Ví dụ:

```
mydict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}
```

Trong ví dụ trên ta đã tạo ra một từ điển có tên là mydict chứa 3 phần tử với mỗi phần tử là một cặp khóa:giá trị.

Khóa và giá trị trong từ điển không nhất thiết chỉ là các chuỗi. Khóa có thể là bất kì kiểu nào như số nguyên, số thực,

... Tương tự, giá trị có thể là một đối tượng bất kì trong Python. Điều này làm cho từ điển trở thành một cấu trúc khá linh hoạt.

```
table = {1975: 'Holy Grail',  
         1979: 'Life of Brian', # Keys are integers, not strings  
         1983: 'The Meaning of Life'}
```

2. Kiểu dữ liệu động

Từ điển là một kiểu dữ liệu động. Nó có thể mở rộng thêm các phần tử sau khi đã tạo cũng như xóa bớt phần tử khi cần thiết mà không tạo ra một từ điển mới.

```
D = {} # Assign by keys dynamically  
D['name'] = 'Bob'  
D['age'] = 40
```

Trong ví dụ trên, đầu tiên ta tạo ra một từ điển rỗng. Sau đó, để thêm phần tử vào ta thực hiện phép gán như thể phần tử đã tồn tại. Khi thực hiện lệnh gán, bên dưới đó, từ điển sẽ tự động thêm khóa mới nếu như phần tử đó chưa tồn tại trong từ điển.

3. Tạo từ điển từ kiểu dữ liệu khác

Ta có thể tạo từ điển từ các dữ liệu khác đã tồn tại. Khi nhận một dữ liệu từ kiểu khác, từ điển sẽ tự động phân tích để tìm ra cặp khóa và giá trị, sau đó tiến hành thêm vào từ điển. Để thực hiện việc này, từ điển sử dụng hàm `dict` và nhận vào đối số là các cặp khóa, giá trị hoặc từ một danh sách các bộ khác.

```
dict(name='Bob', age=40) # dict keyword argument form  
  
dict([('name', 'Bob'), ('age', 40)]) # dict key/value tuples form
```

4. Truy xuất phần tử

Các phần tử được truy xuất thông qua khóa của chúng. Cú pháp truy xuất giống như truy xuất trong danh sách (list), chỉ khác là thay vì vị trí trong danh sách thì lúc này sẽ là các khóa.

```
mydict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
x = mydict["model"]    #access item with key "model"  
print(x)               #Ford
```

Ngoài ra, Python còn cho phép chúng ta lấy phần tử thông qua các hàm của từ điển.

```
mydict.get('brand')
```

Một điểm lợi khi dùng hàm là khi khóa không có trong từ điển, nếu theo cách truy xuất thông thường sẽ gây cho chương trình bị lỗi. Khi dùng hàm `get`, ta sẽ nhận giá trị trả về là `None`, chương trình vẫn thực hiện bình thường mà không bị vấn đề.

5. Thay đổi giá trị

Tương tự như cách truy xuất phần tử trong từ điển, để thay đổi giá trị bằng cách thực hiện phép gán.

```
mydict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
  
mydict["year"] = 2018
```

Trong ví dụ, ta đã thay đổi phần tử có khóa "year" với giá trị mới là 2018.

Lưu ý: khi khóa không tồn tại, từ điển sẽ tạo khóa này và bổ sung giá trị vào.

6. Duyệt trong từ điển

Các cú pháp lặp đều có thể sử dụng để duyệt qua các phần tử trong từ điển.

```
for x in mydict:
    print(mydict[x])
```

Câu lệnh trên thực hiện duyệt từng khóa x trong từ điển và in giá trị tương ứng với khóa đó. Ngoài cách duyệt qua từng khóa, ta có thể duyệt qua từng giá trị trong từ điển bằng cách:

```
for x in mydict.values():
    print(x)
```

Như vậy, lúc này x chính là từng giá trị trong từ điển. Trong trường hợp muốn có cả khóa và giá trị, ta có thể sử dụng cú pháp sau:

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

7. Thêm, xóa phần tử

Thêm phần tử vào mảng ta có thể thực hiện tương tự như phép gán giá trị vào một khóa mới. Từ điển sẽ tự làm các công việc còn lại.

Để xóa phần tử khỏi từ điển, ta có thể sử dụng từ khóa del, hàm pop(), hàm clear()...

```
mydict = {
    "brand": "Ford",
    "model": "Mustang",
    "year": 1964
}
del mydict["model"]      #sử dụng từ khóa del
mydict.pop("model")      #sử dụng hàm pop
mydict.clear()           #đưa từ điển về trạng thái rỗng {}
del mydict               #xóa toàn bộ từ điển (mydict sẽ
không tồn tại nữa)
```

BÀI TẬP CHƯƠNG 8

Bài 8.1 Viết một đoạn lệnh cho phép người dùng thêm một khóa vào từ điển. Trong đó khóa và giá trị do người dùng chỉ định:

Ví dụ:

Từ điển trước khi thêm: {0: 10, 1: 20}

Phần tử cần thêm: key = 2, value = 30

Từ điển sau khi thêm (không cần đúng thứ tự như trong ví dụ):

{0: 10, 1: 20, 2: 30}

Bài 8.2 Viết đoạn lệnh để kiểm tra xem một khóa có nằm trong từ điển không. Nếu có, xuất ra "Tồn tại giá trị trong từ điển với giá trị là v" với v là giá trị có trong từ điển. Nếu không, xuất ra là "Không có khóa k trong từ điển" với k là khóa đã nhập.

Ví dụ: {"a": 1, "b": 2}

Tìm "a"

Tồn tại giá trị trong từ điển với giá trị là 1

Bài 8.3 Viết chương trình xóa một phần tử với khóa cho trước khỏi từ điển. Xuất lại từ điển sau khi xóa.

Ví dụ: {"a": 1, "b": 2, "c": 3.5, "d": "hello"}

Xóa phần tử với khóa: "c"

Từ điển sau khi xóa (không cần đúng thứ tự như ví dụ):

{"a": 1, "b": 2, "d": "hello"}

Bài 8.4 Viết chương trình kiểm tra xem từ điển có rỗng không. Nếu có, in ra là "Từ điển rỗng". Nếu không, in ra là "Từ điển không rỗng".

Ví dụ: {"a": 1, "b": 2, "c": 3.5, "d": "hello"}

Từ điển không rỗng

Bài 8.5 Viết chương trình tính tổng các giá trị là số nguyên có trong từ điển.

Ví dụ: {"a": 1, "b": 2, "c": 3.5, "d": "hello"}

Tổng là 3

Bài 8.6 Viết chương trình gộp 2 từ điển lại làm một. Nếu khóa trùng thì giá trị của khóa này trong từ điển mới là tổng của 2 giá trị trong từ điển cũ. In ra từ điển gộp.

Ví dụ:

Từ điển 1: {"a": 1, "b": 2, "c": 3.5, "d": "hello"}

Từ điển 2: {"e": 1, "b": 2, "f": 3.5, "d": "world"}

Từ điển sau khi gộp (không cần đúng thứ tự như ví dụ):

{"a": 1, "b": 4, "c": 3.5, "d": "helloworld", "e": 1, "f": 3.5}

Bài 8.6 Viết chương trình xóa tất cả các phần tử có giá trị trùng nhau trong từ điển. In ra từ điển sau khi xóa.

Ví dụ: {"a": 1, "b": 2, "c": 1, "d": "hello"}

Từ điển sau khi xóa: {"b": 2, "d": "hello"}

Bài 8.7 Viết chương trình xuất ra 3 phần tử có giá trị lớn nhất trong từ điển. Thứ tự in từ giá trị lớn đến nhỏ, nếu bằng nhau thì thứ tự nào cũng được.

Từ điển: {"a": 1, "b": 8, "c": 9, "d": 2.5, "e": 8}

3 phần tử có giá trị lớn nhất là:

b 9

c 8

e 8

Bài 8.8 Viết chương trình chuyển đổi một chuỗi do người dùng nhập thành từ điển với quy ước như sau:

- Mỗi kí tự trong chuỗi là khóa trong từ điển. Có phân biệt hoa thường.*
- Giá trị của nó là số lần xuất hiện của kí tự đó trong chuỗi*

Ví dụ: "ILoveYouVeryMuch"

Từ điển có được là (không cần đúng thứ tự như trong ví dụ):

```
{"I":1, "L":1, "o":2, "v":1, "e":2, "Y":1, "u":2, "V":1, "r":1, "y":1, "M":1, "c":1, "h":1}
```

Bài 8.9 Viết chương trình dùng từ điển để quản lý một danh sách các sinh viên trong lớp với mỗi sinh viên có các thông tin:

- Id: mã sinh viên*
- Họ tên*
- Năm sinh*
- Giới tính*
- Địa chỉ*
- Điểm Toán, Lý, Hóa*

Mỗi sinh viên được quản lý với khóa là mã id của sinh viên và không trùng nhau.

Chương trình cho phép:

- Thêm một sinh viên mới*
- Xóa một sinh viên dựa trên mã sinh viên*
- Tìm sinh viên với họ tên*
- Tính điểm TB cho từng sinh viên*

Bài 9: BỘ (TUPLE)

Mục tiêu:

- Tổ chức dữ liệu với kiểu dữ liệu Bộ (Tuple)
- Thực hiện các thao tác trên Bộ
- Vận dụng các hàm xây dựng sẵn của Bộ

Nội dung chính:

- Khai báo và sử dụng kiểu dữ liệu Bộ
- Duyệt trong Bộ
- Các lệnh thường dùng

1. Khai báo dữ liệu

Kiểu dữ liệu Bộ (tuple) về cơ bản giống với kiểu danh sách (list) nhưng chúng ta không thể thay đổi các phần tử trong nó một cách tự do mà không sinh ra bộ nhớ mới.

Như vậy, về khai báo và khởi tạo giá trị nó tương tự như danh sách.

```
mytuple = ("apple", "banana", "cherry")
```

Trong ví dụ trên ta đã tạo ra một tuple gồm 3 phần tử. Như vậy so với danh sách thì điểm khác nằm ở chỗ dấu ngoặc. Với danh sách, Python dùng dấu ngoặc vuông [], còn với bộ Python dùng dấu ngoặc tròn ().

Tương tự như các kiểu tập hợp khác, kiểu dữ liệu Bộ có thể chứa bất kì kiểu dữ liệu khác.

```
mytuple = ('Bob', 40.5, ['dev', 'mgr'])
```

2. Kiểu dữ liệu cố định

Một khi đã khởi tạo các giá trị cho bộ thì ta không thể thay đổi giá trị của nó. Đồng nghĩa với việc, cũng không thể thêm phần tử mới vào trong bộ đang có sẵn.

```
mytuple = ("apple", "banana", "cherry")
mytuple[1] = "blackcurrant" #error
```

Trong ví dụ trên ta không thể thay đổi giá trị "banana" thành "blackcurrant" được.

Để thay đổi giá trị hay thêm phần tử mới vào Bộ đều phải thực hiện qua phương pháp gián tiếp. Hay nói cách khác, ta phải tạo ra một Bộ mới với các phần tử mà chúng ta muốn giữ lại ở Bộ cũ và bổ sung thêm các dữ liệu mới. Ví dụ:

```
mytuple = (1, 2, 3, 4)
mytuple = mytuple + (5, 6)
```

Lưu ý, biến mytuple của lần gán sau là một bộ hoàn toàn mới.

3. Truy xuất phần tử

Truy xuất các phần tử trong Bộ tương tự như trong danh sách. Nghĩa là chúng ta sử dụng chỉ mục đếm từ 0 để truy xuất vị trí của phần tử trong Bộ.

```
mytuple = ("apple", "banana", "cherry")
print(mytuple[1])
```

Việc truy xuất đến chỉ mục nằm ngoài chỉ mục cho phép của Bộ sẽ gây ra lỗi.

```
mytuple = ("apple", "banana", "cherry")
print(mytuple[9]) #error
```

4. Duyệt với vòng lặp

Ta có thể sử dụng bất kì vòng lặp nào để duyệt qua các phần tử của Bộ. Nói chung là giống với cách chúng ta làm việc với danh sách.

```
mytuple = ("apple", "banana", "cherry")
for x in mytuple:
    print(x)
```

5. Thêm, xóa phần tử

Một lần nữa lưu ý là ta không thể thêm hay xóa một phần tử bất kì trong Bộ. Để thực hiện việc thêm, xóa ta phải thực hiện một cách gián tiếp. Các cách này đều sinh ra một Bộ mới.

```
mytuple = (1, 2, 3, 4)
mytuple = mytuple + (5, 6)      # Thêm (5,6) vào mytuple
mytuple = (2,) + mytuple[1:]    # Thay thế 1 bằng giá trị 2
```

Trong ví dụ trên, mỗi mytuple ở vế trái của phép gán là một Bộ hoàn toàn mới.

6. Các hàm của Bộ

Để lấy chiều dài của Bộ, ta có thể dùng hàm len():

```
mytuple = (1, 2, 3, 4)
len(mytuple)      # chiều dài của mytuple là 4
```

Để lấy vị trí (index) của một giá trị có trong Bộ, ta dùng hàm index(). Hay để đếm số lần xuất hiện của một giá trị bất kì, ta dùng hàm count(). Lưu ý là hai hàm này phải xác định đối với Bộ cho trước.

```
mytuple = (1, 1, 3, 4)
mytuple.index(4)    #4 xuất hiện ở vị trí 3
mytuple.count(2)    #số lần xuất hiện của 1 là 2
```

BÀI TẬP CHƯƠNG 9

Bài 9.1 Viết chương trình tạo ra một tuple từ dữ liệu người dùng nhập vào

Ví dụ:

Các giá trị người dùng nhập:

1

2

3

4

Bộ được tạo là: (1,2,3,4)

Bài 9.2 Viết chương trình thêm một phần tử vào cuối của một bộ cho trước.

Ví dụ: ("a", 2, "b", "c")

Thêm: 2.5

Bộ sau khi thêm: ("a", 2, "b", "c", 2.5)

Bài 9.3 Viết chương trình xóa phần tử ở cuối khỏi một bộ cho trước.

Ví dụ: ("a", 2, "b", "c")

Bộ sau khi xóa: ("a", 2, "b")

Bài 9.4 Viết chương trình thêm phần tử vào vị trí bất kì trong bộ cho trước. Giá trị và vị trí do người dùng chỉ định. Vị trí được tính từ 0.

Ví dụ: ("a", 2, "b", "c")

Thêm 8 vào vị trí 2.

Bộ sau khi thêm: ("a", 2, 8, "b", "c")

Bài 9.4 Viết chương trình xóa một phần tử ở vị trí bất kì trong bộ cho trước. Vị trí do người dùng chỉ định và được tính từ 0.

Ví dụ: ("a", 2, "b", "c")

Xóa tại vị trí 1.

Bộ sau khi xóa: ("a", "b", "c")

Bài 9.5 Viết chương trình xóa các phần tử trùng nhau trong bộ cho trước (nếu có phần tử trùng thì chỉ giữ lại 1 phần tử đầu tiên gặp được).

Ví dụ: ("a", 2, "b", 2, "a", "c", "a")

Bộ sau khi xóa: ("a", 2, "b", "c")

Bài 9.6 Viết chương trình tính tổng các số có trong bộ (số nguyên, số thực).

Ví dụ: ("a", 3, "b", 2.5, "a", 8.5, "a")

Tổng là: 14

Bài 9.7 Viết chương trình đảo lại thứ tự của bộ.

Ví dụ: ("a", "b", "c")

Bộ sau khi đảo là: ("c", "b", "a")

Bài 9.7 Viết chương trình sắp xếp bộ chứa các số nguyên lại theo thứ tự tăng dần.

Ví dụ: (8, 2, 9, 15, 2, 1)

Bộ sau khi sắp xếp là: (1, 2, 2, 8, 9, 15)

Bài 9.8 Viết chương trình kiểm tra xem bộ có đối xứng không.

Ví dụ:

(1, 2, 3, 3, 2, 1)

Bộ đối xứng

(1, 2, 3, 2, 1)

Bộ đối xứng

()

Bộ đối xứng

(1, 2, 3, 4, 2, 1)

Bộ không đối xứng

Bài 9.9 Viết chương trình tìm phần tử lớn thứ k trong một bộ các số nguyên. Với k do người dùng chỉ định.

Ví dụ: (8, 2, 9, 15, 2, 1)

k là 4

Phần tử lớn thứ k là 2

Bài 10: TẬP TIN

Mục tiêu:

- Lưu trữ dữ liệu lên tập tin
- Truy xuất dữ liệu trên tập tin
- Sử dụng dữ liệu sau khi đọc

Nội dung chính:

- Đóng mở tập tin
- Ghi dữ liệu lên tập tin
- Đọc dữ liệu từ tập tin
- Thao tác dữ liệu

1. Giới thiệu tập tin

Tập tin được sử dụng để lưu trữ dữ liệu lên các thiết bị như ổ cứng, mạng... Trong máy tính, tập tin được chia làm hai loại: tập tin văn bản và tập tin nhị phân.

Nói một cách vắn tắt, tập tin văn bản chứa nội dung là các kí tự mà con người có thể đọc được. Tập tin nhị phân chứa nội dung là các byte thô mà chỉ có chương trình đặc thù mới hiểu và thể hiện nội dung tương ứng.

Đối với tập tin văn bản, Python hỗ trợ cả chuẩn ASCII và Unicode. Cách truy xuất ở mỗi dạng cũng có phần khác nhau. Để đơn giản và phù hợp với ở mức độ môn học, chúng ta tạm thời gác Unicode qua một bên và tập trung chủ yếu vào ASCII.

Đối với tập tin nhị phân, chúng ta chỉ lướt nhẹ qua về cách mở và cũng không đi sâu vào.

1. Trình tự thao tác với tập tin

Khi thao tác với tập tin, chúng ta thường trải qua các giai đoạn:

- Mở tập tin với một đường dẫn xác định đến nơi đang chứa tập tin và kèm với đó là các mode mà chúng ta muốn thao tác.
- Kế tiếp, ta thực hiện các thao tác mà mình muốn làm với tập tin như đọc, ghi, di chuyển trong tập tin.
- Cuối cùng là đóng tập tin lại để giải phóng tài nguyên cũng như hoàn tất quá trình ghi lên ổ cứng.

Trong 3 thao tác trên, thường lập trình viên quên viết thao tác cuối cùng (đóng tập tin). Python có chức năng phát hiện và tự thực hiện các công việc cần thiết để đóng tập tin khi lập trình viên quên. Điều này giúp cho chương trình tránh các lỗi xảy ra. Tuy nhiên, thói quen lập trình tốt vẫn là nhớ đóng tập tin lại khi xài xong.

2. Mở tập tin

Python hỗ trợ hàm `open()` để mở tập tin:

```
f = open("demofile.txt")
```

Ví dụ trên sẽ mở một file có tên là `demofile.txt` tại nơi đang chứa code python. Khi mở thành công, Python sẽ trả về một đối tượng file và chúng ta đặt tên cho nó là `f`. Ta có thể chỉ định đường dẫn tuyệt đối trong chuỗi tập tin nhưng cách này không khuyến khích.

Với cách mở trên, file được mở mặc định với chế độ mở là văn bản và việc mở để dành cho đọc. Trong trường hợp muốn thực hiện các mục đích khác như mở để ghi, mở tập tin nhị phân, ... ta cần chỉ định thêm đối số thứ hai trong hàm trên. Đối số ở dạng chuỗi.

```
f = open("demofile.txt", "rt")
```

Trong ví dụ, chúng ta chỉ định rõ mở file để đọc và tập tin ở dạng văn bản.

Các cấu hình:

- "r": đọc (mặc định)
- "a": mở ghi tiếp (tạo mới nếu file chưa có)
- "w": viết (tạo mới nếu file chưa có)
- "x": tạo mới (nếu có file thì báo lỗi)

Có thể bổ sung thêm loại tập tin:

- "t": loại tập tin văn bản (mặc định)
- "b": loại tập tin nhị phân

3. Đọc tập tin

Các hàm mà Python hỗ trợ cho việc đọc:

- `read()`: đọc toàn bộ nội dung của file vào một biến chuỗi
- `read(N)`: đọc N kí tự (hoặc byte) vào biến chuỗi
- `readline()`: đọc 1 dòng (bao gồm cả dấu xuống dòng `\n`) vào biến chuỗi.
- `readlines()`: đọc toàn bộ file vào một danh sách các dòng.

Ví dụ:

```
f = open("demofile.txt", "r")
print(f.read())
print(f.read(5))
print(f.readline())
print(f.readlines())
```

Để có thể đọc lần lượt từng dòng cho đến khi hết file, ta có thể làm như sau:

```
f = open('script2.py')
while True:
    line = f.readline()
    if not line: break
    print(line.upper(), end='')
```

Trong ví dụ, ta kiểm tra biến `line` để biết dấu hiệu kết thúc file. Khi đạt đến vị trí cuối, biến `line` sẽ rỗng. Ngoài cách trên, ta có thể dùng cú pháp đơn giản hơn:

```
f = open("demofile.txt", "rt")
for line in f:
    print(line)
```

4. Ghi tập tin

Các hàm hỗ trợ cho việc viết nội dung lên tập tin:

- `write(aString)`: viết một chuỗi lên tập tin
- `writelines(aList)`: viết tất cả các dòng trong danh sách lên tập tin

Ví dụ:

```
f = open("demofile.txt", "w")
f.write("Hello World")
```

Ta có thể viết tiếp một tập tin đang sẵn có:

```
f = open("demofile.txt", "a")
f.write("Now the file has one more line!")
```

Lưu ý: để viết một đối tượng không phải là chuỗi lên file, ta phải chuyển đổi (convert) đối tượng này thành chuỗi trước khi ghi. Phương thức `write()` không tự động làm cho chúng ta các công việc này.

```
X, Y, Z = 43, 44, 45
S = 'Spam'
D = {'a': 1, 'b': 2}
L = [1, 2, 3]

F = open('datafile.txt', 'w')
F.write(S + '\n')
F.write('%s,%s,%s\n' % (X, Y, Z))
F.write(str(L) + '$' + str(D) + '\n')
F.close()
```

Một trường hợp ngoại lệ là khi dùng hàm `writelines()` cho phép ghi một danh sách lên file. Tuy nhiên, mỗi phần tử trong danh sách cũng phải ở dạng chuỗi.

5. Đóng tập tin

Một thói quen lập trình tốt là nên đóng file lại khi không còn sử dụng nữa.

```
myfile = open('myfile.txt', 'w')
myfile.write('hello text file\n')
myfile.close()
```

6. Xóa tập tin

Để xóa tập tin ta phải sử dụng một import một module của hệ thống và chạy thực thi hàm `remove()`:

```
import os
os.remove("demofile.txt")
```

Lưu ý, khi file không tồn tại, lệnh này sẽ phát sinh ra lỗi. Để khắc phục nguy cơ này ta nên kiểm tra file có tồn tại không trước khi xóa.

```
import os
if os.path.exists("demofile.txt"):
    os.remove("demofile.txt")
else:
    print("The file does not exist")
```

Python còn cung cấp cho chúng ta thêm hàm xóa thư mục:

```
import os
os.rmdir("myfolder")
```

BÀI TẬP CHƯƠNG 10

Bài 10.1 Viết chương trình tạo một tập tin văn bản với tên và nội dung do người dùng chỉ định.

Ví dụ:

Tên tập tin: abc.txt

Nội dung: "Hello World"

Bài 10.2 Viết chương trình đọc nội dung của một tập tin văn bản cho trước. Tên tập tin do người dùng chỉ định. Trong tập tin có nhiều dòng chữ.

Ví dụ:

Tên tập tin: abc.txt

Nội dung:

Hello World

I Love You Very Much

I am here. Where are you?

Bài 10.3 Viết chương trình đọc một danh sách các số được ghi trong một tập tin văn bản, với mỗi số cách nhau bằng dấu khoảng trắng. Hiển thị danh sách ra màn hình và tính tổng các số đó.

Ví dụ:

Tên tập tin: abc.txt

Nội dung: 1 2 3 4 5.5 6.5

Tổng là: 22

Bài 10.4 Viết chương trình ghi thêm nội dung vào một tập tin văn bản đang có trên hệ thống.

Ví dụ:

Tên tập tin: abc.txt

Nội dung đang có: "Hello"

Nội dung thêm: "my love"

Bài 10.5 Viết chương trình xóa một tập tin đang có trên hệ thống.

Ví dụ:

Tên tập tin: abc.txt

Đã xóa tập tin

Tên tập tin: xyz.txt

Tập tin không tìm thấy

Bài 10.6 Viết chương trình xóa toàn bộ tập tin trong một thư mục có sẵn. Lưu ý, không xóa thư mục

Ví dụ:

Thư mục: myFolder

Đã xóa tất cả tập tin trong thư mục

Bài 10.6 Viết chương trình nối 2 tập tin văn bản lại thành một tập tin duy nhất.

Ví dụ:

Tập tin: abc.txt và def.txt

Tập tin đã nối là abcdef.txt

Bài 10.6 Viết chương trình tách tập tin văn bản thành các tập tin sao cho mỗi tập tin không quá x kí tự và số lượng tập tin được tạo là ít nhất. Với x là số kí tự do người dùng chỉ định.

Ví dụ:

Tập tin: abc.txt

Tập tin đã nói là a.txt, b.txt, c.txt, d.txt

Bài 10.7 Viết chương trình ghi một danh sách sinh viên lên tập tin sau đó đọc lại vào một danh sách khác. Kiểm tra hai danh sách có giống nhau không.

Thông tin sinh viên bao gồm:

- Mã SV
- Họ tên
- Năm sinh
- Giới tính
- Điểm Toán
- Điểm Lý
- Điểm Hóa

Mỗi sinh viên sẽ cách nhau một dòng trống. Danh sách phải có ít nhất 5 sinh viên.

Ví dụ:

Tên tập tin: abc.txt

Nội dung:

```
Id: 1801
Họ tên: Nguyễn Văn Tèo
Năm sinh: 2000
Giới tính: Nam
Điểm Toán: 8
Điểm Lý: 9
Điểm Hóa: 5

Id: 1802
Họ tên: Lê Thị Béo
Năm sinh: 1989
Giới tính: Nữ
Điểm Toán: 2
Điểm Lý: 3
Điểm Hóa: 4

Id: 1803
Họ tên: Trần Như Trục
Năm sinh: 2030
Giới tính: Không xác định
Điểm Toán: 10
Điểm Lý: 0.05
Điểm Hóa: 8.5
```

TÀI LIỆU THAM KHẢO

https://www.python-course.eu/conditional_statements.php

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

<https://jaxenter.com/differences-python-2-3-148432.html>

<https://jaxenter.com/implement-switch-case-statement-python-138315.html>