MỤC LỤC

Chương 1: Một số khái niệm về lập trình và ngôn ngữ lập trình 1
Bài 1. Khái niệm về lập trình 1
a) Thông dịch 2
b) Biên dịch 2
Bài đọc thêm 1 2
Bài 2. Các thành phần của ngôn ngữ lập trình 3
1. Các thành phần cơ bản 3
2. Một số khái niệm 4
Câu hỏi và bài tập 6
Bài đọc thêm 2 7
Chương 2: Chương trình đơn giản 8
Bài 3. Cấu trúc chương trình 8
1. Cấu trúc chung 8
2. Các thành phần của chương trình 8
3. Ví dụ chương trình đơn giản 9
Bài 4. Một số kiểu dữ liệu chuẩn 11
1. Kiểu số 11
2. Kiểu logic 11
Bài 5. Khai báo biến 12
Bài 6. Phép toán, biểu thức, lệnh gán 13
1. Phép toán 13
2. Biểu thức số học 14
3. Hàm số học chuẩn 15
4. Biểu thức quan hệ 15
5. Biểu thức lôgic 16
6. Câu lệnh gán 17
Bài 7. Các hàm chuẩn vào/ra đơn giản 18
1. Nhập dữ liệu vào từ bàn phím 18
2. Đưa dữ liệu ra màn hình 19
Bài 8. Soạn thảo, dịch, thực hiện và hiệu chỉnh chương trình 22
Bài tập và thực hành 1 26
1. Mục đích, yêu cầu 26
2. Nội dung 26
Câu hỏi và bài tập 27
Chương 3. Cấu trúc rẽ nhánh và lặp 28
Bài 9. Cấu trúc rẽ nhánh 28
1. Rẽ nhánh 28

2. Câu lệnh if 29
3. Câu lệnh ghép 30
4. Một số ví dụ 31
Bài 10. Cấu trúc lặp 32
1. Lặp 32
2. Lặp có số lần lặp biết trước và câu lệnh lặp `for` 32
3. Lặp với số lần chưa biết trước và câu lệnh lặp `while` 34
Bài tập và thực hành 2 37
1. Mục đích, yêu cầu 37
2. Nội dung 37
Câu hỏi và bài tập 38
Chương 4. Kiểu dữ liệu có cấu trúc 40
Bài 11. Kiểu danh sách 40
1. Danh sách một chiều 40
2. Kiểu mảng hai chiều 48
Bài tập và thực hành 3 51
1. Mục đích, yêu cầu 51
2. Nội dung 51
Bài tập và thực hành 4 53
1. Mục đích, yêu cầu 53
2. Nội dung 53
Bài 12. Kiểu xâu ký tự 55
1. Khai báo 56
2. Các thao tác cơ bản trên xâu 56
3) Một số ví dụ 61
Bài tập và thực hành 5 64
1. Mục đích, yêu cầu 64
2. Nội dung 64
Câu hỏi và bài tập 65
Chương 5. Tệp và thao tác với tệp 67
Bài 14. Kiểu dữ liệu tệp 67
1. Vai trò kiểu tệp 67
2. Phân loại tệp và thao tác với tệp 67
Bài 15. Kiểu tệp 68
1. Khai báo 68
2. Thao tác với tệp 68
Bài 16. Ví dụ làm việc với tệp 70
Ví dụ 1 70

Ví dụ 2. 71
Câu hỏi và bài tập **72**
Chương 6. Hàm và lập trình có cấu trúc **73**
Bài 17. Chương trình con và phân loại **73**
1. Khái niệm chương trình con 73
2. Phân loại và cấu trúc của chương trình con 75
Bài 18. Ví dụ về cách định nghĩa và sử dụng hàm **77**
1. Cách định nghĩa hàm 77
2. Một số ví dụ về hàm 77
3. Vấn đề tham số của hàm trong Python 79
Bài tập và thực hành 6 ..... **81**
1. Mục đích, yêu cầu 81
2. Nội dung 81
Bài tập và thực hành 7 ..... **82**
1. Mục đích, yêu cầu 82
2. Nội dung 82

Chương 1: Một số khái niệm về lập trình và ngôn ngữ lập trình

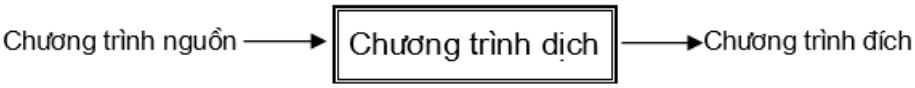
Bài 1. Khái niệm về lập trình

Như ta biết, mọi bài toán có thuật toán đều có thể giải được trên máy tính điện tử. Khi giải bài toán trên máy tính điện tử, sau các bước xác định bài toán và xây dựng hoặc lựa chọn thuật toán khả thi là bước lập trình.

Lập trình là sử dụng cấu trúc dữ liệu và các câu lệnh của ngôn ngữ lập trình cụ thể để mô tả dữ liệu và diễn đạt các thao tác của thuật toán. Chương trình viết bằng ngôn ngữ lập trình bậc cao nói chung không phụ thuộc vào loại máy, nghĩa là một chương trình có thể thực hiện trên nhiều loại máy tính khác nhau. Chương trình viết bằng ngôn ngữ máy có thể được nạp trực tiếp vào bộ nhớ và thực hiện ngay, còn chương trình viết bằng ngôn ngữ lập trình bậc cao phải được chuyển đổi thành chương trình trên ngôn ngữ máy mới có thể thực hiện được.

Chương trình đặc biệt có chức năng chuyển đổi chương trình được viết bằng ngôn ngữ lập trình bậc cao thành chương trình thực hiện được trên máy tính cụ thể được gọi là *chương trình dịch*.

Chương trình dịch nhận đầu vào là chương trình viết bằng ngôn ngữ lập trình bậc cao (chương trình nguồn), thực hiện chuyển đổi sang ngôn ngữ máy (chương trình đích).



Hình 1.1 - Minh họa quá trình chuyển đổi chương trình nguồn thành chương trình đích

Xét ví dụ, bạn chỉ biết tiếng Việt nhưng cần giới thiệu về trường của mình cho đoàn khách đến từ nước Mĩ, chỉ biết tiếng Anh. Có hai cách để bạn thực hiện điều này.

Cách thứ nhất: Bạn nói bằng tiếng Việt và người phiên dịch giúp bạn dịch sang tiếng Anh. Sau mỗi câu hoặc một vài câu giới thiệu trọn một ý, người phiên dịch dịch sang tiếng Anh cho đoàn khách. Sau đó, bạn lại giới thiệu tiếp và người phiên dịch lại dịch tiếp. Việc giới thiệu của bạn và việc dịch của người phiên dịch luân phiên cho đến khi bạn kết thúc nội dung giới thiệu của mình. Cách dịch trực tiếp như vậy được gọi là *thông dịch*.

Cách thứ hai: Bạn soạn nội dung giới thiệu của mình ra giấy, người phiên dịch dịch toàn bộ nội dung đó sang tiếng Anh rồi đọc hoặc trao văn bản đã dịch cho đoàn khách đọc. Như vậy, việc dịch được thực hiện sau khi nội dung giới thiệu đã hoàn tất. Hai công việc đó được thực hiện trong hai khoảng thời gian độc lập, tách biệt nhau. Cách dịch như vậy được gọi là *biên dịch*.

Sau khi kết thúc, với cách thứ nhất không có một văn bản nào để lưu trữ, còn với cách thứ hai có hai bản giới thiệu bằng tiếng Việt và bằng tiếng Anh có thể lưu trữ để dùng lại về sau.

Tương tự như vậy, chương trình dịch có hai loại là *thông dịch* và *biên dịch*.

a) Thông dịch

Thông dịch (interpreter) được thực hiện bằng cách lặp lại dãy các bước sau:

- Kiểm tra tính đúng đắn của câu lệnh tiếp theo trong chương trình nguồn;
- Chuyển đổi câu lệnh đó thành một hay nhiều câu lệnh tương ứng trong ngôn ngữ máy;
- Thực hiện các câu lệnh vừa chuyển đổi được.

Như vậy, quá trình dịch và thực hiện các câu lệnh là luân phiên. Các chương trình thông dịch lần lượt dịch và thực hiện từng câu lệnh. Loại chương trình dịch này đặc biệt thích hợp cho môi trường đối thoại giữa người và hệ thống. Tuy nhiên, một câu lệnh nào đó phải thực hiện bao nhiêu lần thì nó phải được dịch bấy nhiêu lần.

Các ngôn ngữ khai thác hệ quản trị cơ sở dữ liệu, ngôn ngữ đối thoại với hệ điều hành,... đều sử dụng trình thông dịch.

b) Biên dịch

Biên dịch (compiler) được thực hiện qua hai bước:

- Duyệt, kiểm tra, phát hiện lỗi, kiểm tra tính đúng đắn của các câu lệnh trong chương trình nguồn;
- Dịch toàn bộ chương trình nguồn thành một chương trình đích có thể thực hiện trên máy và có thể lưu trữ để sử dụng lại khi cần thiết.

Như vậy, trong thông dịch, không có chương trình đích để lưu trữ, trong biên dịch cả chương trình nguồn và chương trình đích có thể lưu trữ lại để sử dụng vềsau.

Thông thường, cùng với chương trình dịch còn có một số dịch vụ liên quan như biên soạn, lưu trữ, tìm kiếm, cho biết các kết quả trung gian,... Toàn bộ các dịch vụ trên tạo thành một môi trường làm việc trên một ngôn ngữ lập trình cụ thể. Ví dụ, Turbo Pascal 7.0, Free Pascal1.2, Visual Pascal 2.1,... trên ngôn ngữ Pascal, Turbo C++, Visual C++,... trên ngôn ngữ C++.

Các môi trường lập trình khác nhau ở những dịch vụ mà nó cung cấp, đặc biệt là các dịch vụ nâng cấp, tăng cường các khả năng mới cho ngôn ngữ lập trình.

Bài đọc thêm 1

Có thể sử dụng bài viết từ địa chỉ : https://vi.wikipedia.org/wiki/Ngôn ngữ lập trình.

Bài 2. Các thành phần của ngôn ngữ lập trình

1. Các thành phần cơ bản

Mỗi ngôn ngữ lập trình thường có ba thành phần cơ bản là *bảng chữ cái*, *cú pháp* và *ngữ nghĩa*.

a) Bảng chữ cái

Là tập các kí tự được dùng để viết chương trình. Không được phép dùng bất kì kí tự nào ngoài các kí tự quy định trong bảng chữ cái.

Trong Python, bảng chữ cái bao gồm các kí tự:

- Các chữ cái in thường và các chữ cái in hoa của bảng chữ cái tiếng Anh:

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

- 10 chữ số thập phân Ả Rập: `0 1 2 3 4 5 6 7 8 9`
- Các kí tự đặc biệt:

+	-	*	/	=		<	>	[	]	.	,
;	#	^	$	@	&	(	)	{	}	:	'
Dấu cách (mã ASCII là 32)									_		

Bảng chữ cái của các ngôn ngữ lập trình nói chung không khác nhau nhiều. Ví dụ, bảng chữ cái của ngôn ngữ lập trình Python khác Pascal là có sử dụng thêm các kí tự như dấu nháy kép ("), nháy ba ('''), dấu sổ ngược (\), dấu chấm than (!)...

b) Cú pháp

Là bộ quy tắc để viết chương trình. Dựa vào chúng, người lập trình và chương trình dịch biết được tổ hợp nào của các kí tự trong bảng chữ cái là hợp lệ và tổ hợp nào là không hợp lệ. Nhờ đó, có thể mô tả chính xác thuật toán để máy thực hiện.

c) Ngữ nghĩa

Xác định ý nghĩa thao tác cần phải thực hiện, ứng với tổ hợp kí tự dựa vào ngữ cảnh của nó.

Ví dụ

Phần lớn các ngôn ngữ lập trình đều sử dụng dấu cộng (+) để chỉ phép cộng. Xét các biểuthức:

```
A + B     (1)
I + J     (2)
```

Giả thiết *A, B* là các đại lượng nhận giá trị thực và *I, J* là các đại lượng nhận giá trị chuỗi ký tự. Khi đó dấu "+" trong biểu thức (1) được hiểu là cộng hai số thực, dấu "+" trong biểu thức (2) được hiểu là phép nối hai chuỗi. Như vậy, ngữ nghĩa dấu "+" trong hai ngữ cảnh khác nhau là khác nhau.

Tóm lại, cú pháp cho biết cách viết một chương trình hợp lệ, còn ngữ nghĩa xác định ý nghĩa của các tổ hợp kí tự trong chương trình.

Các lỗi cú pháp được chương trình dịch phát hiện và thông báo cho người lập trình biết. Chỉ có các chương trình không còn lỗi cú pháp mới có thể được dịch sang ngôn ngữ máy.

Các lỗi ngữ nghĩa khó phát hiện hơn. Phần lớn các lỗi ngữ nghĩa chỉ được phát hiện khi thực hiện chương trình trên dữ liệu cụ thể.

2. Một số khái niệm

a) Tên (Identifiers)

Mọi đối tượng trong chương trình đều phải được đặt tên theo quy tắc của ngôn ngữ lập trình và từng chương trình dịch cụ thể.

Trong Python, tên là một dãy liên tiếp có số kí tự tùy ý bao gồm chữ số, chữ cái hoặc dấu gạch dưới và bắt đầu bằng chữ cái hoặc dấu gạch dưới.

Ví dụ, trong ngôn ngữ Python:

- Các tên đúng:

```
A
R21
P21_c
_45
```

- Các tên sai:

```
A BC     (chứa kí tự trắng)
6Pq      (bắt đầu bằng chữ số)
X#Y      (chứa kí tự "#" không hợp lệ)
```

Ngôn ngữ Python phân biệt chữ hoa, chữ thường trong tên. Khác với NNLT Pascal không phân biệt chữ hoa, chữ thường. Vídụ, *AB* và *Ab* là hai tên khác nhau trong Python, nhưng lại là cùng một tên trong Pascal.

Nhiều ngôn ngữ lập trình, trong đó có Python, phân biệt hai loại tên:

- Tên dành riêng;
- Tên do người lập trình đặt.

Tên dành riêng

Là loại tên được ngôn ngữ lập trình quy định dùng với ý nghĩa xác định, người lập trình không được sử dụng với ý nghĩa khác. Những tên này được gọi là tên dành riêng (còn được gọi là từ khoá).

Ví dụ. Một số tên dành riêng:

Trong Python: `while, else, import, True, if, def, for`

Trong C++: `main, include, if, while, void`

Tên do người lập trình đặt

Tên do người lập trình đặt được dùng với ý nghĩa riêng, xác định bằng cách khai báo trước khi sử dụng. Các tên này không được trùng với tên dành riêng.

Ví dụ

Tên do người lập trình đặt:

```
A1
DELTA
CT_Vidu
```

b) Hằng và biến

Hằng (constants)

Hằng là các đại lượng có giá trị không thay đổi trong quá trình thực hiện chương trình.

Trong các ngôn ngữ lập trình thường có các hằng số học, hằng lôgic, hằng xâu.

- Hằng số học là các số nguyên hay số thực (dấu phẩy tĩnh hoặc dấu phẩy động), có dấu hoặc không có dấu.
- Hằng lôgic là giá trị đúng hoặc sai tương ứng với *True* hoặc *False*.
- Hằng xâu là chuỗi kí tự trong bảng chữ cái. Khi viết, chuỗi kí tự này được đặt trong cặp dấu nháy (Python dùng dấu nháy đơn hoặc nháy kép, còn C++ dùng dấu nháy kép).

Ví dụ

- Hằng số học:

```
2           0        -5           +18
1.5         -22.36   +3.14159     0.5
-2.236E01   1.0E-6
```

- Hằng lôgic:

+ Trong Python: (chú ý viết in hoa chữ cái đầu)

```
True    False
```

- Hằng xâu:

+ Trong Python:

```
'Informatic'     "TIN HOC"
```

+ Trong C++:

```
"Informatic"     "TIN HOC"
```

Chú ý: Với Python bạn có thể tùy ý viết nháy đơn hoặc nháy kép đều được.

Biến (variables)

Biến là đại lượng được đặt tên, dùng để lưu trữ giá trị và giá trị có thể được thay đổi trong quá trình thực hiện chương trình.

Trong chương trình Python, không yêu cầu bạn khai báo biến và kiểu biến trước khi dùng. Python sẽ xác định kiểu biến ngay sau khi bạn gán giá trị cho biến. Việc sử dụng biến sẽ được trình bày ở các phần sau.

c) Chú thích (comments)

Có thể đặt các đoạn chú thích trong chương trình nguồn. Các chú thích này giúp cho người đọc chương trình nhận biết ngữ nghĩa của chương trình đó dễ hơn. Chú thích không ảnh hưởng đến nội dung chương trình nguồn và được chương trình dịch bỏ qua.

Trong Python có hai loại chú thích: chú thích trên một dòng, chú thích gồm nhiều dòng

- Chú thích trên một dòng được bắt đầu bằng dấu #.

  Ví dụ: Dòng sau đây là chú thích và được trình phiên dịch Python bỏ qua

```
# Hello everybody
```

- Chú thích trên nhiều dòng được bắt đầu và kết thúc bằng cách viết 3 dấu nháy đơn hoặc kép

  Ví dụ: Các dòng sau đây là chú thích và được trình phiên dịch Python bỏ qua

```
""" This is also a
perfect example of
multi-line comments """
```

TÓM TẮT

- Cần có chương trình dịch để chuyển chương trình nguồn thành chương trình đích.
- Có hai loại chương trình dịch: thông dịch và biên dịch.
- Các thành phần của ngôn ngữ lập trình: bảng chữ cái, cú pháp và ngữ nghĩa.
- Mọi đối tượng trong chương trình đều phải được đặt tên:
  - Tên dành riêng: Được dùng với ý nghĩa riêng, không được dùng với ý nghĩa khác.
  - Tên do người lập trình đặt: cần khai báo trước khi sử dụng.
- Hằng: Đại lượng có giá trị không thay đổi trong quá trình thực hiện chương trình.
- Biến: Đại lượng được đặt tên. Giá trị của biến có thể thay đổi trong quá trình thực hiện chương trình.

Câu hỏi và bài tập

1. Tại sao người ta phải xây dựng các ngôn ngữ lập trình bậc cao?

2. Chương trình dịch là gì? Tại sao cần phải có chương trình dịch?

3. Biên dịch và thông dịch khác nhau như thế nào?

4. Hãy cho biết các điểm khác nhau giữa tên dành riêng và tên do người dùng đặt.

5. Hãy tự viết ra ba tên đúng theo quy tắc của Python.

6. Hãy cho biết những biểu diễn nào dưới đây không phải là biểu diễn hằng trong Python và chỉ rõ lỗi trong từng trường hợp:

```
a) 150.0     b) -22       c) 6,23  d) <43>
e) A20       f) 1.06E-15  g) 4+6   h) 'C
i) 'TRUE'
```

Bài đọc thêm 2

Các thầy/cô có thể sử dụng bài viết từ wikipedia theo địa chỉ:

https://vi.wikipedia.org/wiki/Python_(ngôn_ngữ_lập_trình)

Chương 2: Chương trình đơn giản

Bài 3. Cấu trúc chương trình

1. Cấu trúc chung

Nói chung, chương trình được viết bằng một ngôn ngữ lập trình bậc cao thường gồm phần khai báo và phần thân. Phần thân chương trình nhất thiết phải có. Phần khai báo có thể có hoặc không tuỳ theo từng chương trình cụ thể.

Khi diễn giải cú pháp của ngôn ngữ lập trình người ta thường sử dụng ngôn ngữ tự nhiên. Các diễn giải bằng ngôn ngữ tự nhiên được đặt giữa cặp dấu < và >. Các thành phần của chương trình có thể có hoặc không được đặt trong cặp dấu [và].

Với quy ước trên, cấu trúc của một chương trình có thể được mô tả như sau:

```
[< phần khai báo >]
<phần thân>
```

2. Các thành phần của chương trình

a) Phần khai báo

Có thể có các khai báo: thư viện cần dùng, hằng, biến và chương trình con. Tuy nhiên phần này tùy theo từng chương trình mà có hoặc không.

Khai báo thư viện

Mỗi ngôn ngữ lập trình thường có sẵn một số thư viện cung cấp một số chương trình thông dụng đã được lập sẵn. Để sử dụng các chương trình đó cần khai báo thư viện chứa nó.

Ví dụ. Khai báo thư viện

- Trong Python: có một số cách, sau đây là một cách

```
import <math>
```

- Trong C++:

```
#include <cmath >
```

Thư viện *math* trong Python hoặc *cmath* trong C++ cung cấp các chương trình có sẵn để làm việc với các hàm số học. Ví dụ, muốn tính giá trị căn bậc 2 của một số *a*:

- Trong Python, sau khi khai báo thư viện math, ta dùng lệnh:

```
x = math.sqrt(a)
```

- Trong C++, sau khi khai báo thư viện stdio.h, ta dùng lệnh:

```
double x = sqrt(a);
```

Khai báo hằng

Ví dụ. Khai báo hằng

- Trong Python:

```
MaxN = 1000
PI = 3.1416
KQ = 'Ket qua:'
```

- Trong C++:

```
const int MaxN = 1000;
const float PI = 3.1416;
```

Khai báo hằng thường được sử dụng cho những giá trị cố định mà xuất hiện nhiều lần trong chương trình.

Khai báo biến

Tất cả các biến dùng trong chương trình đều phải đặt tên cho chương trình dịch biết để lưu trữ và xử lí. Biến chỉ nhận một giá trị tại mỗi thời điểm thực hiện chương trình được gọi là biến đơn.

Ví dụ

Khi khảo sát phương trình đường thẳng $ax + by + c = 0$, các hệ số a, b, c có thể được khai báo như những biến đơn.

Cách khai báo biến được trình bày riêng trong bài 5.

Khai báo và sử dụng chương trình con được trình bày trong chương 5.

b) Phần thân chương trình

Python là ngôn ngữ thông dịch, chương trình dịch của Python dịch đến đâu thì thực hiện chương trình tới đó. Như vậy không có quy định chặt chẽ phải có phần khai báo và phần thân chương trình như Pascal. Thực tế chương trình Python chỉ là một dãy các dòng lệnh được viết trong một tệp văn bản có đuôi mặc định là ***.py***

Có một điểm cần chú ý, Python không yêu cầu sử dụng dấu ; để chỉ báo điểm kết thúc một câu lệnh như Pascal/C/C++ hoặc một số NNLT khác. Thay vào đó, Python quy định mỗi câu lệnh nên được viết trên một dòng riêng biệt.

3. Ví dụ chương trình đơn giản

Dưới đây xét một vài ví dụ về những chương trình đơn giản.

Ví dụ 1.

Chương trình sau thực hiện việc đưa ra màn hình thông báo "*Xin chao cac ban!*".

Trong Python	Trong C++	Trong Pascal

`print("Xin chao cac ban!")`	`#include <stdio.h>`
	`main()`
	`{`
	`  printf("Xin chao cac ban!");`
	`}`
		`program vi_du;`
		`begin`
		`  writeln('Xin chao cac ban!');`
		`end.`

- Phần khai báo không có - Phần thân chương trình chỉ có một câu lệnh *print* đưa thông báo ra màn hình.	- Phần khai báo chỉ có một câu lệnh *#include* khai báo thư viện *stdio.h*. - Phần thân chương trình chỉ có một câu lệnh *printf* đưa thông báo ra màn hình.	- Phần khai báo chỉ có khai báo tên chương trình gồm tên dành riêng *program* và tên chương trình là *vi_du*. - Phần thân chương trình chỉ có một câu lệnh *writeln*, đưa thông báo ra màn hình.

Ví dụ 2.

Chương trình Python sau đưa các thông báo "*Xin chao cac ban!*" và "*Moi cac ban lam quen voi Python*" ra màn hình.

```
print('Xin chao cac ban!')
print('Moi cac ban lam quen voi Python')
```

Chương trình trên không có phần khai báo. Phần thân chương trình có hai câu lệnh in ra màn hình hai thông báo.

Bài 4. Một số kiểu dữ liệu chuẩn

Các bài toán trong thực tế thường có dữ liệu vào và kết quả ra thuộc những kiểu dữ liệu quen biết như số nguyên, số thực, kí tự,... Khi lập trình cho những bài toán như vậy, khi cần người lập trình sử dụng các kiểu dữ liệu đó thường gặp một số hạn chế nhất định, phụ thuộc vào một số yếu tố như dung lượng bộ nhớ, khả năng xử lí của CPU,...

Vì vậy, mỗi ngôn ngữ lập trình thường cung cấp một số kiểu dữ liệu chuẩn cho biết phạm vi giá trị có thể lưu trữ, dung lượng bộ nhớ cần thiết để lưu trữ và các phép toán tác động lên dữ liệu. Dưới đây xét một số kiểu dữ liệu chuẩn thường dùng cho các biến đơn trong Python.

1. Kiểu số

Python hỗ trợ các kiểu số nguyên, số thực, số phức. Nhưng Python không yêu cầu khai báo kiểu biến từ trước nên khi biến được gán giá trị kiểu nào thì biến sẽ có kiểu đó.

Ví dụ, khi chạy đoạn chương trình sau:

```
x = 5
print(type(x))
x = 1.2345
print(type(x))
```

Ta sẽ nhận được các thông báo dưới đây:

```
<class 'int'>
<class 'float'>
```

Các thông báo này cho biết lần đầu tiên x có kiểu int (nguyên), lần sau x có kiểu float (thực)

Một điều thú vị là kiểu nguyên trong Python cho phép ta tính toán với số cực lớn hàng trăm chữ số. Ví dụ chương trình sau in ra kết quả của 100! (giai thừa)

```
import math
x = math.factorial(100)
print(x)
```

Hàm *factorial()* cho phép tính giai thừa và kết quả in ra là:

```
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000
```

2. Kiểu logic

Python hỗ trợ kiểu logic với hai giá trị đúng (*True*) và sai (*False*). Chú ý viết in hoa chữ đầu.

Bài 5. Khai báo biến

(Bài 5 nên được chuyển thành một giờ luyện tập về cách khai báo và sử dụng biến)

Như đã nói ở trên, mọi biến dùng trong chương trình Python không cần khai báo từ trước, khi nào cần dùng biến ta sẽ khai báo đồng thời gán cho nó một giá trị để ấn định kiểu của biến.

Ví dụ:

```
x = 1.5  # khai báo biến thực x và gán cho nó giá trị 1.5
y = 245  # khai báo biến nguyên y và gán cho nó giá trị 245
```

Một số chú ý khi khai báo biến:

- Cần đặt tên biến sao cho gợi nhớ đến ý nghĩa của biến đó. Điều này rất có lợi cho việc đọc, hiểu và sửa đổi chương trình khi cầnthiết.

Ví dụ, không nên vì cho ngắn gọn mà đặt tên biến là *d1, d2* mà nên đặt là *dtoan, dtin* gợi nhớ tới ngữ nghĩa của các biến đó là điểm toán, điểm tin của học sinh.

- Không nên đặt tên biến quá ngắn hay quá dài, dễ mắc lỗi khi viết nhiều lần tên biến. Ví dụ, không nên dùng *d1, d2* hay *diemmontoan, diemmontin* cho điểm toán, điểm tin của học sinh.
- Khi khai báo biến cần lưu ý đến kiểu giá trị của nó. Ví dụ, khi khai báo biến biểu diễn giá trị diện tích của hình tròn có thể sử dụng kiểu thực, nhưng biến biểu diễn số học sinh thì dùng kiểu nguyên.
- Biến trong Python được sử dụng cơ động, có thể lúc trước biến *x* là kiểu nguyên sau đó là kiểu thực vẫn được chấp nhận tùy theo giá trị bạn gán cho nó vào thời điểm hiện tại lúc đó.

Ví dụ:

```
x = (4 + 2) * 5  # x có kiểu nguyên
x = 5*5*3.14     # x có kiểu thực
```

Bài 6. Phép toán, biểu thức, lệnh gán

Để mô tả các thao tác trong thuật toán, mỗi ngôn ngữ lập trình đều xác định và sử dụng một số khái niệm cơ bản: phép toán, biểu thức, gán giá trị cho biến.

Dưới đây sẽ xét các khái niệm đó trong Python.

1. Phép toán

Tương tự trong toán học, trong các ngôn ngữ lập trình đều có những phép toán số học như cộng, trừ, nhân, chia trên các đại lượng thực, các phép toán chia nguyên và lấy phần dư, các phép toán quan hệ,… Bảng dưới đây là kí hiệu các phép toán đó trong toán và trong Python:

a. Phép toán số học với số nguyên

Tên phép toán	Trong Python	Trong Pascal	Ví dụ
cộng	+	+	x + y
trừ	-	-	x - y
nhân	*	*	x * y
chia nguyên	//	DIV	x // y
lấy phần dư	%	MOD	x % y
phép lũy thừa	**	không có	x**3 (tức là x^3)

b. Phép toán số học với số thực

Tên phép toán	Trong Python	Trong Pascal	Ví dụ
cộng	+	+	x + y
trừ	-	-	x - y
nhân	*	*	x * y
chia	/	/	x / y
phép lũy thừa	**	không có	x**3 (tức là x^3)

Chú ý phép chia lấy kết quả nguyên và kết quả thực là khác nhau, rất dễ nhầm

c. Phép toán quan hệ

Tên phép toán	Trong Python	Trong Pascal	Ví dụ
so sánh bằng	==	=	x == y
nhỏ hơn	<	<	x < y
lớn hơn	>	>	x > y
nhỏ hơn hoặc bằng	<=	<=	x <= y
lớn hơn hoặc bằng	>=	>=	x >= y
khác	!=	<>	x != y

d. Phép toán logic

Tên phép toán	Trong Python	Trong Pascal	Ví dụ
Phủ định	not	not	not x
hoặc	or	or	x or y
và	and	and	x and y

Chú ý: Trong Python các phép toán logic chỉ được viết chữ in thường.

2. Biểu thức số học

Trong lập trình, biểu thức số học là một biến kiểu số hoặc một hằng số hoặc các biến kiểu số và các hằng số liên kết với nhau bởi một số hữu hạn phép toán số học, các dấu ngoặc đơn (và) tạo thành một biểu thức có dạng tương tự như cách viết trong toán học với những quy tắc sau:

- Chỉ dùng các cặp ngoặc đơn để xác định trình tự thực hiện phép toán trong trường hợp cần thiết;
- Viết lần lượt từ trái qua phải;
- Không được bỏ qua dấu nhân (*) trong tích.

Các phép toán được thực hiện theo thứ tự:

- Ưu tiên thực hiện các phép toán trong ngoặc trước;
- Trong dãy các phép toán không chứa ngoặc thì thực hiện từ trái sang phải, theo thứ tự các phép toán nhân (*), chia (/), chia nguyên (//), lấy phần dư (%) thực hiện trước và các phép toán cộng (+), trừ (-) thực hiện sau.

Ví dụ

Biểu thức trong toán học	Biểu thức trong Python
`5a + 6b`	`5*a + 6*b`
$\frac{xy}{z}$	`x*y/z`
$Ax^2 + Bx + C$	`A*x*x + B*x + C`
$\frac{x+y}{x-\frac{1}{2}} - \frac{x-z}{xy}$	`(x+y)/(x-1/2) - (x-z)/(x*y)`

Chú ý

a) Nếu biểu thức chứa một hằng hay biến kiểu thực thì ta có biểu thức số học thực, giá trị của biểu thức cũng thuộc kiểu thực.

b) Trong một số trường hợp nên dùng biến trung gian để có thể tránh được việc tính một biểu thức nhiều lần.

c) Theo mặc định Python yêu cầu mỗi lệnh được viết trên một dòng riêng biệt, trường hợp câu lệnh quá dài ta có thể viết trên nhiều dòng nhưng cuối mỗi dòng cần có thêm ký hiệu \ , ví dụ:

```
a = 1 + 2 + 3 + \
    4 + 5 + 6 + \
    7 + 8 + 9
```

Trong các phiên bản mới của Python hiện nay, cũng cho phép ta viết như sau:

```
a = (1 + 2 + 3 +
    4 + 5 + 6 +
    7 + 8 + 9)
```

3. Hàm số học chuẩn

Để lập trình được dễ dàng, thuận tiện hơn, các ngôn ngữ lập trình đều có thư viện chứa một số chương trình tính giá trị những hàm toán học thường dùng. Các chương trình như vậy được gọi là các hàm số học chuẩn. Mỗi hàm chuẩn có tên chuẩn riêng. Đối số của hàm là một hay nhiều biểu thức số học và được đặt trong cặp ngoặc đơn (và) sau tên hàm. Bản thân hàm chuẩn cũng được coi là một biểu thức số học và nó có thể tham gia vào biểu thức số học như một toán hạng (giống như biến và hằng). Kết quả của hàm có thể là nguyên hoặc thực hay phụ thuộc vào kiểu của đối số.

Bảng dưới đây cho biết một số hàm chuẩn thường dùng.

Hàm	Biểu diễn toán học	Biểu diễn Python	Kiểu đối số	Kiểu kết quả
lũy thừa	x^y	**pow**(*x*, *y*)	Thực hoặc nguyên	Theo kiểu đối số
căn bậc 2	$\sqrt{x}$	**sqrt**(*x*)	Thực hoặc nguyên	Thực
giá trị tuyệt đối	$\|x\|$	**abs**(*x*)	Thực hoặc nguyên	Theo kiểu đối số
logarit cơ số a của x	$log_a x$	**log**(*x*, *a*)	Thực	Theo kiểu đối số
logarit cơ số 2 của x	$log_2 x$	**log2**(*x*)	Thực	Theo kiểu đối số
lũy thừa cơ số e	e^x	**exp**(*x*)	Thực	Theo kiểu đối số
hàm sin	*sinx*	**sin**(*x*)	Thực	Theo kiểu đối số
hàm cos	*cosx*	**cos**(*x*)	Thực	Theo kiểu đối số
hàm tang	*tg*	**tan**(*x*)	Thực	Theo kiểu đối số

Để dùng các hàm: ***pow(), sqrt(), log(), log2(), exp(), sin(), cos(), tan()*** phải có lệnh khai báo nhập khẩu thư viện *import math* ở đầu chương trình và lời gọi hàm phải có từ *math* đi trước cùng dấu chấm phân cách.

Ví dụ:

Biểu thức toán học $\frac{-b+\sqrt{b^2-4ac}}{2a}$ trong Python có thể viết dưới dạng:

```
(-b + math.sqrt(b**2 - 4*a*c)/(2*a)
```

Hoặc

```
(-b + math.sqrt(b**2 - 4*a*c)/ 2/a
```

Ngoài những hàm số học chuẩn trên, còn có các hàm chuẩn khác được giới thiệu trong những phần sau.

4. Biểu thức quan hệ

Hai biểu thức cùng kiểu liên kết với nhau bởi phép toán quan hệ cho ta một biểu thức quan hệ.

Biểu thức quan hệ có dạng:

```
<biểu thức 1> <phép toán quan hệ> <biểu thức 2>
```

trong đó, *biểu thức 1* và *biểu thức 2* cùng là xâu hoặc cùng là biểu thức số học.

Ví dụ

```
x < 5
i+1 >= 2*j
```

Biểu thức quan hệ được thực hiện theo trình tự:

- Tính giá trị các biểu thức.
- Thực hiện phép toán quan hệ.

Kết quả của biểu thức quan hệ là giá trị lôgic: *True* (đúng) hoặc *False* (sai).

Trong ví dụ trên, nếu *x* có giá trị 3, thì biểu thức *x < 5* có giá trị *True*. Nếu *i* có giá trị 2 và *j* có giá trị 3 thì biểu thức *i + 1 >= 2*j* sẽ cho giá trị *False*.

Ví dụ

Điều kiện để điểm *M* có toạ độ (*x; y*) thuộc hình tròn tâm *I(a; b)*, bán kính *R* là:

```
math.sqrt((x-a)*(x-a) + (y-b)*(y-b)) <= R
```

hoặc

```
(x-a)**2 + (y-b)**2 <= R**2
```

5. Biểu thức lôgic

Biểu thức lôgic đơn giản là biến lôgic hoặc hằng lôgic.

Biểu thức lôgic là các biểu thức lôgic đơn giản, các biểu thức quan hệ liên kết với nhau bởi phép toán lôgic. Giá trị biểu thức lôgic là *True* hoặc *False* (xem phụ lục 4. Bảng giá trị phép toán lôgic). Các biểu thức quan hệ thường được đặt trong cặp ngoặc đơn (và).

Dấu phép toán ***not*** được viết trước biểu thức cần phủ định, ví dụ:

not *(x* < 1*)* thể hiện phát biểu "*x không nhỏ hơn 1*" và điều này tương đương với biểu thức quan hệ *x* >= 1.

Các phép toán ***and*** và ***or*** dùng để kết hợp nhiều biểu thức lôgic hoặc quan hệ, thành một biểu thức thường được dùng để diễn tả các điều kiện phức tạp.

Ví dụ 1

Để thể hiện điều kiện $5 \le x \le 11$, trong Python có thể tách thành phát biểu dưới dạng "$5 \le x$ và $x \le 11$" và được viết như sau:

```
(5 <= x) and (x <= 11)
```

Thậm chí, Python còn cho phép viết như sau:

```
5 <= x <= 11
```

Ví dụ 2

Giả thiết *M* và *N* là hai biến nguyên. Điều kiện xác định *M* và *N* đồng thời chia hết cho 3 hay đồng thời không chia hết cho 3 được thể hiện trong Python như sau:

```
(m % 3 == 0 and n % 3 == 0) or (m % 3 != 0 and n % 3 != 0)
```

6. Câu lệnh gán

Lệnh gán là một trong những lệnh cơ bản nhất của các ngôn ngữ lập trình.

Trong Python câu lệnh gán có dạng:

```
<tên biến> =  <biểu thức>
```

Trong trường hợp đơn giản, tên biến là tên của biến đơn. Kiểu của biến sẽ bị thay đổi theo giá trị của biểu thức.

Chức năng của lệnh gán là đặt cho biến có tên ở vế trái dấu "=" giá trị mới bằng giá trị của biểu thức ở vế phải, đồng thời cũng làm cho biến nhận kiểu giá trị mới.

Ví dụ

```
x1 = (-b - sqrt(b**2 - 4*a*c))/(2*a)
x2 = -b/a - x1
z = z - 1
i = i + 1
```

Trong ví dụ trên, ý nghĩa của lệnh gán thứ ba là giảm giá trị của biến z một đơn vị. Ý nghĩa của lệnh gán thứ tư là tăng giá trị của biến *i* lên một đơn vị.

Ngoài ra Python còn hỗ trợ một số cách viết phép gán như sau:

Biểu thức gán trong Python	Ý nghĩa
`x += 2`	Tăng *x* lên 2 đơn vị
`x -= 2`	Giảm *x* đi 2 đơn vị
`x *= 2`	Tăng *x* lên 2 lần
`x /= 2`	Giảm *x* đi 2 lần
`x %= 2`	Thay *x* bằng phần dư của *x* chia 2
`x //= 2`	Thay *x* bằng phần nguyên của *x* chia 2
`x **= 3`	Nâng *x* lên thành x^3
`a, b = b, a`	Tráo đổi giá trị của *a* và *b* cho nhau
`a = b = 5`	Gán cho cả hai biến *a* và *b* cùng một giá trị 5
`a, b = 2, 3`	Gán *a* = 2 và *b* = 3

Bài 7. Các hàm chuẩn vào/ra đơn giản

Để khởi tạo giá trị ban đầu cho biến, ta có thể dùng lệnh gán để gán một giá trị cho biến. Như vậy, mỗi chương trình luôn làm việc với một bộ dữ liệu vào. Để chương trình có thể làm việc với nhiều bộ dữ liệu vào khác nhau, thư viện của các ngôn ngữ lập trình cung cấp một số chương trình dùng để đưa dữ liệu vào và đưa dữ liệu ra.

Những chương trình đưa dữ liệu vào cho phép đưa dữ liệu từ bàn phím hoặc từ đĩa vào và gán cho các biến, làm cho chương trình trở nên linh hoạt, có thể tính toán với nhiều bộ dữ liệu đầu vào khác nhau. Kết quả tính toán được lưu trữ tạm thời trong bộ nhớ. Những chương trình đưa dữ liệu ra dùng để đưa các kết quả này ra màn hình, in ra giấy hoặc lưu trên đĩa.

Các chương trình đưa dữ liệu vào và ra đó được gọi chung là các *hàm chuẩn vào/ra đơn giản*.

Trong phần này, ta sẽ xét các *hàm chuẩn vào/ra đơn giản* của Python để nhập dữ liệu vào từ bàn phím và đưa thông tin ra màn hình.

1. Nhập dữ liệu vào từ bàn phím

Việc nhập dữ liệu từ bàn phím trong Python được thực hiện bằng một số hàm khác nhau, ở đây ta chọn cách dùng hàm *input()*. Trong Python các hàm nhập dữ liệu hầu như đều nhận giá trị đầu vào ở dạng chuỗi. Vì vậy, sau khi nhận được chuỗi đầu vào ta cần trích xuất và chuyển đổi chuỗi đầu vào thành dạng dữ liệu mong muốn.

Ví dụ:

a) Để nhập vào số nguyên *n* từ bàn phím, ta dùng lệnh sau:

```
n = int(input([chuỗi thông báo]))
```

Ở đây sau khi *input()* nhận được chuỗi đầu vào từ bàn phím (cụ thể là số nguyên *n*) thì gửi cho hàm *int()* chuyển đổi chuỗi đầu vào đó thành một số nguyên và gán cho *n*.

Chuỗi thông báo là chuỗi kí tự sẽ được in ra màn hình trước khi ta nhập dữ liệu từ bàn phím như một lời nhắc nhở người dùng. Có thể không cần chuỗi thông báo như phải có cặp ngoặc đơn sau từ *input*.

b) Để nhập vào ba số nguyên *a, b, c* từ bàn phím, ta có thể dùng lệnh sau:

```
a, b, c = map(int, input([chuỗi thông báo]).split())
```

Sau khi hàm *input()* nhận được chuỗi đầu vào từ bàn phím, nó gọi hàm *split()* để trích xuất chuỗi dữ liệu đầu vào thành các chuỗi con, mặc định là căn cứ vào các khoảng trắng (dấu cách). Mỗi khi *split()* trích xuất được chuỗi con nào thì nó gửi cho hàm *int()* chuyển đổi chuỗi con đó thành một số nguyên, hàm *map()* sẽ gửi các số nguyên đó cho từng biến bên vế trái phép gán một cách lần lượt, số đầu tiên cho *a*, số thứ hai cho *b*, số thứ ba cho *c*.

Khi nhập giá trị cho nhiều biến như ví dụ b) ở trên, những giá trị này phải được gõ cách nhau bởi ít nhất một dấu cách. Các giá trị ứng với biến nguyên phải được biểu diễn dưới dạng nguyên (không có dấu chấm thập phân). Các giá trị ứng với biến thực có thể gõ dưới dạng số nguyên, số thực dạng thông thường hoặc số thực dạng dấu phẩy động.

Ví dụ, để nhập các giá trị 1, -5 và 6 cho các biến thực *a, b, c* trong ví dụ b) ở trên, có thể gõ:

```
1 -5 6
```

rồi gõ phím Enter

2. Đưa dữ liệu ra màn hình

a. Hàm print()

Để đưa dữ liệu ra màn hình, Python cung cấp hàm chuẩn:

```
print(<danh sách kết quả ra>)
```

trong đó, danh sách kết quả ra có thể là tên biến đơn, biểu thức hoặc hằng. Các hằng xâu thường được dùng để tách các kết quả hoặc đưa ra chú thích. Các thành phần trong kết quả ra được viết cách nhau bởi dấu phẩy.

Theo mặc định, sau khi in các kết quả ra màn hình, con trỏ tự động được chuyển xuống dòng tiếp theo. Để giữ cho con trỏ không chuyển xuống đầu dòng tiếp theo ta cần thêm tham số *end=''* vào sau danh sách kết quả ra như sau.

```
print(<danh sách kết quả ra>, end='')
```

Ví dụ 1

Để nhập giá trị nguyên cho biến *M* từ bàn phím, người ta thường dùng lệnh:

```
M = int(input("Hãy nhập giá trị M: "))
```

Khi thực hiện lệnh này, màn hình xuất hiện hộp thoại Python Shell với dòng chữ:

```
Hãy nhập giá trị M:
```

và con trỏ nhấp nháy trong hộp nhập dữ liệu, chờ bạn gõ vào giá trị của *M*.

Ví dụ 2

Sau đây là một chương trình hoàn chỉnh có sử dụng các hàm vào/ra chuẩn của Python.

```
1. N = int(input("Lớp bạn có bao nhiêu người?"))
2. print("Vậy bạn có", N-1, "người bạn trong lớp")
```

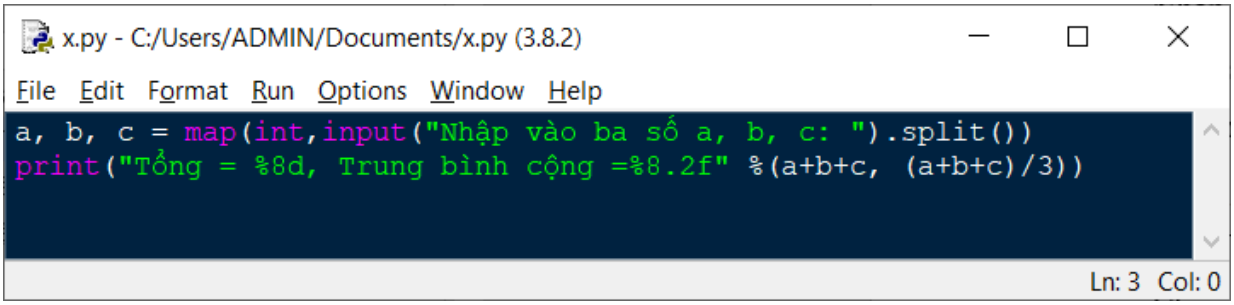
b. Chuỗi định dạng đầu ra

Đôi khi in kết quả ra màn hình, bạn cần định dạng nó theo một số khuôn mẫu, ví dụ như số nguyên phải được in với độ rộng nhất định, số thực phải được in với độ rộng và số chữ số thập phân

nhất định, ... khi đó bạn cần cung cấp cho hàm *print()* một chuỗi thông số gọi là *chuỗi định dạng đầu ra* để yêu cầu hàm in theo định dạng.

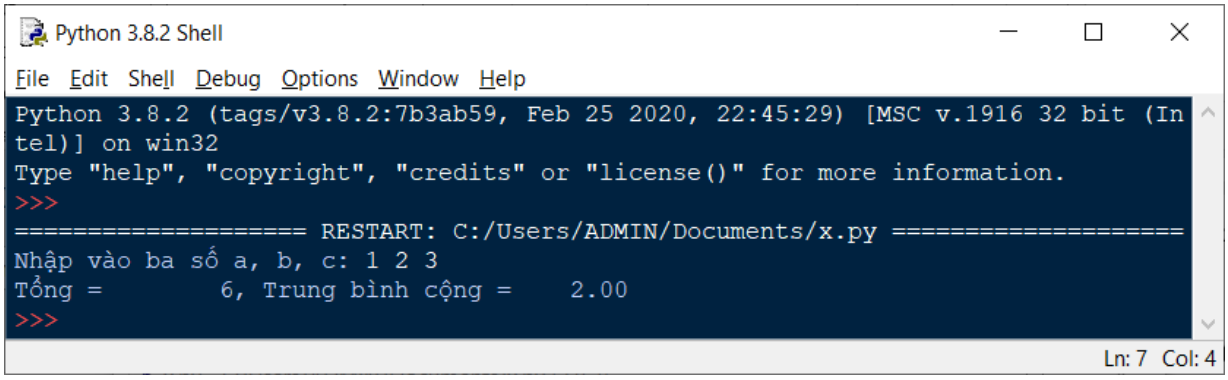
Ví dụ 3

Nhập vào từ bàn phím giá trị 3 số nguyên *a, b, c* và in ra màn hình giá trị tổng và trung bình cộng của 3 số đó. Giá trị tổng phải được in với độ rộng 8, trung bình cộng phải được in với độ rộng 8 và có 2 chữ số thập phân.



Hình 2.1 - Minh họa ví dụ 3 trong cửa sổ Python IDLE

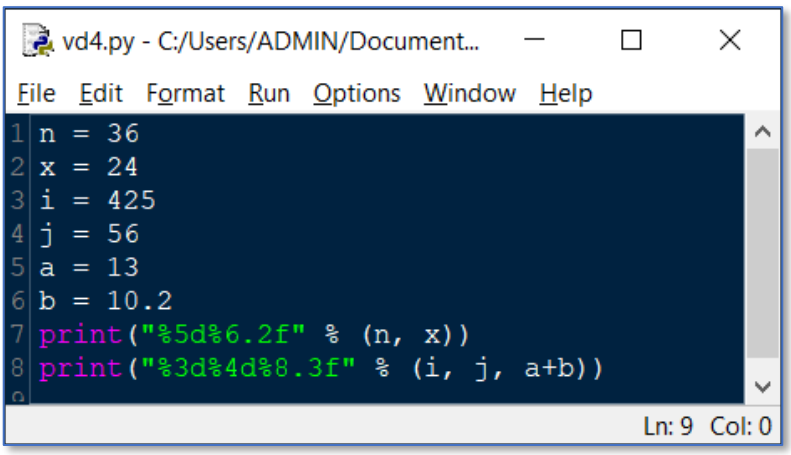
Chạy chương trình, xuất hiện màn hình nhập dữ liệu. Nhập vào 3 giá trị: `1 2 4`, chương trình sẽ in ra màn hình kết quả sau:



Hình 2.2 - Minh họa kết quả ví dụ 3 trong màn hình Python Shell

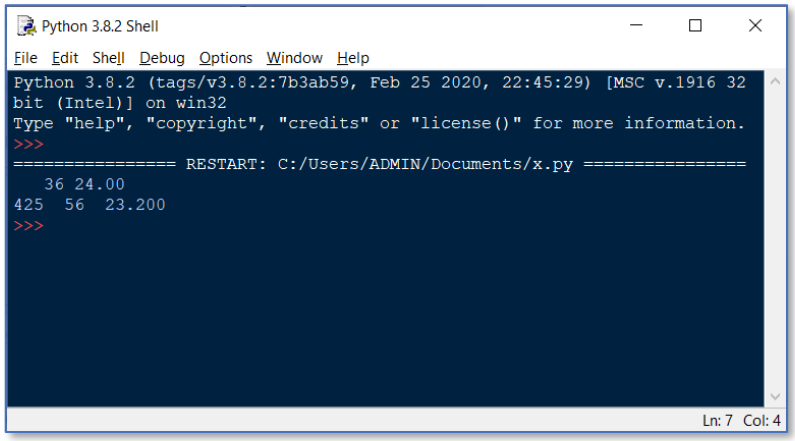
Chuỗi định dạng đầu ra là chuỗi ký tự bắt đầu bằng ký hiệu %, tiếp theo là tham số độ rộng trên màn hình, nếu là vị trí sẽ in ra số thực thì sau tham số độ rộng sẽ có dấu chấm và tiếp đó là số chữ số thập phân đó. Trong chuỗi "*Tổng = %8d, Trung bình cộng =%8.2f*" thì hàm print() in ra một chuỗi thông báo, như xen trong chuỗi đó có hai vị trí chứa chuỗi điều khiển định dạng "*%8d*" cho biết là ở vị trí này sẽ được in ra một số nguyên ở hệ 10 (chữ "d" - decimal), "*%8.2f*" cho biết tham số ở vị trí này phải được in như một số thực (chữ "f" - float) với độ rộng là 8 và có 2 chữ số thập phân. Phần sau của lệnh *print()* phân cách bởi dấu % là phần chỉ định các biểu thức giá trị sẽ được in tại các vị trí đã khai báo định dạng. Số lượng tham số và số lượng các biểu thức giá trị cần in phải bằng nhau. Phần cuối của lệnh *print()* : % (*a+b+c, (a+b+c)/3*) cho biết có hai biểu thức *a + b + c* và *(a+b+c)/3* tương ứng với hai vị trí sẽ được in ra màn hình qua chuỗi định dạng.

Ví dụ 4:



Hình 2.3 - Cửa sổ soạn thảo chương trình ví dụ 4

Chạy chương trình trên ta được màn hình kết quả như sau:



Hình 2.4 - Màn hình kết quả chạy chương trình ví dụ 4

Trong chương trình trên. Dòng lệnh 7, dành 5 vị trí kể từ vị trí con trỏ hiện thời để in ra giá trị của *n*. Nếu *n* có giá trị nguyên ít hơn 5 chữ số hoặc giá trị âm dưới 4 chữ số thì những vị trí đầu sẽ được điền đầy bằng các dấu cách. Tiếp theo là 6 vị trí được dành để in giá trị *x*, trong đó 2 vị trí dành để in ra phần thập phân. Do phần nguyên và phần thập phân được cách nhau bởi dấu chấm nên còn lại 3 vị trí dành cho phần nguyên.

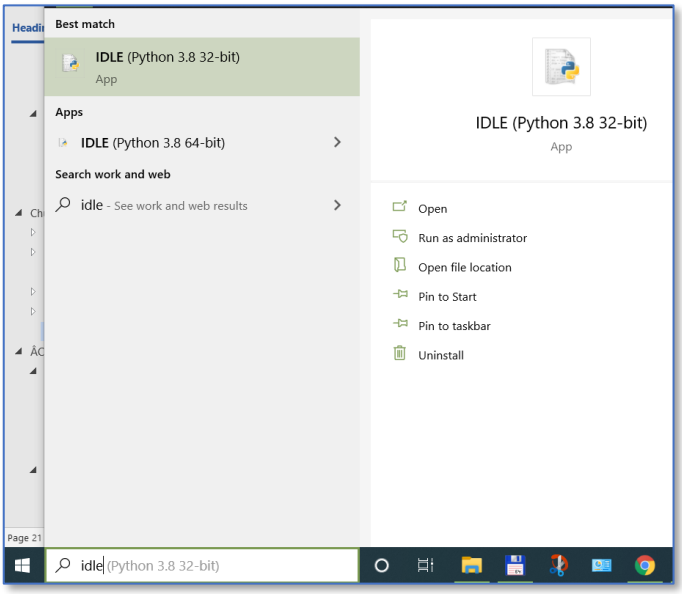
Dòng lệnh 8, đầu tiên là vị trí dành cho biến *i* với 3 chỗ và giá trị của biểu thức *a* + *b* với 8 chỗ, trong đó có 3 chỗ dành cho phần thập phân.

Bài 8. Soạn thảo, dịch, thực hiện và hiệu chỉnh chương trình

Để có thể thực hiện chương trình được viết bằng một ngôn ngữ lập trình, ta cần soạn thảo, sử dụng chương trình dịch để dịch chương trình đó sang ngôn ngữ máy. Các hệ thống lập trình cụ thể thường cung cấp phần mềm phục vụ cho việc soạn thảo, dịch và hiệu chỉnh chương trình.

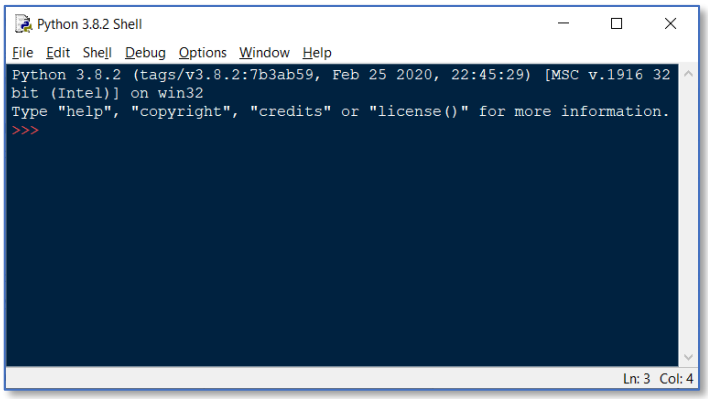
Với ngôn ngữ Python, khi bạn cài đặt phần mềm Python 3.x trên máy tính, thì công cụ lập trình đơn giản nhất gọi là Python IDLE (*Integrated Development and Learning Environment*) đã được cài đặt sẵn. Từ đây cuốn sách này chỉ giới thiệu cách làm việc với Python IDLE.

Sau khi chắc chắn cài đặt Python 3.x trên mày tính cài HĐH Windows, bạn vào chức năng tìm kiếm của windows (nút hình kính lúp cạnh nút *start*) và gõ cụm từ *idle*:



Hình 2.5 - Minh họa màn hình tìm kiếm công cụ Python IDLE

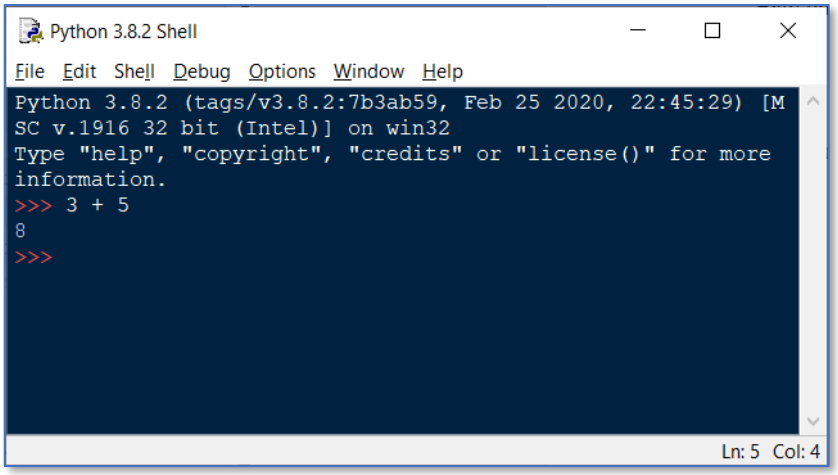
Bạn sẽ nhìn thấy màn hình tương tự như trên. Bạn cũng có thể tìm thấy công cụ Python IDLE trong menu start của windows. Bấm đúp chuột vào công cụ IDLE (hoặc Enter), bạn sẽ thấy màn hình tương tự như sau:



Hình 2.6 - Màn hinh Python Shell

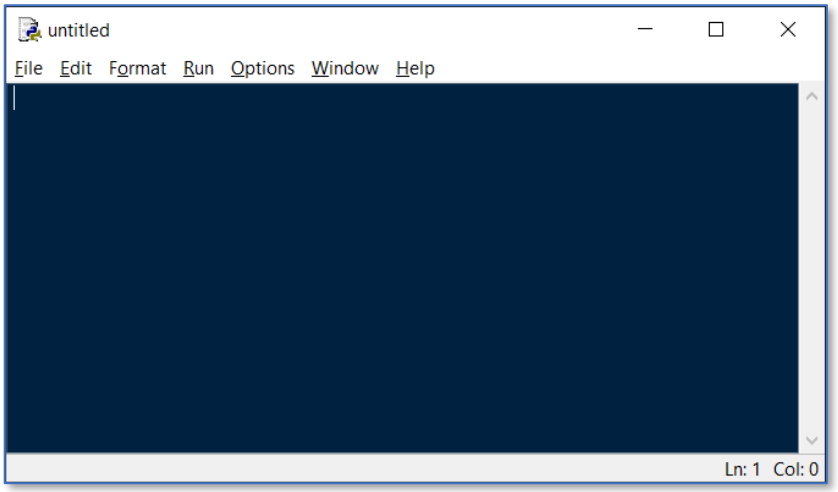
Đây chính là giao diện cửa số lệnh (Python Shell) để bạn có thể chạy các câu lệnh đơn giản của Python ngay từ dấu nhắc >>>.

Ví dụ: Từ dấu nhắc bạn gõ biểu thức toán 3 + 5, Enter thì bạn sẽ nhận được kết quả 8 ở dòng dưới như trong hình. Con trỏ lại xuất hiện ở dòng cuối và chờ bạn ra lệnh tiếp.



Hình 2.7 - Minh họa việc tính toán trực tiếp trên cửa sổ Python Shell

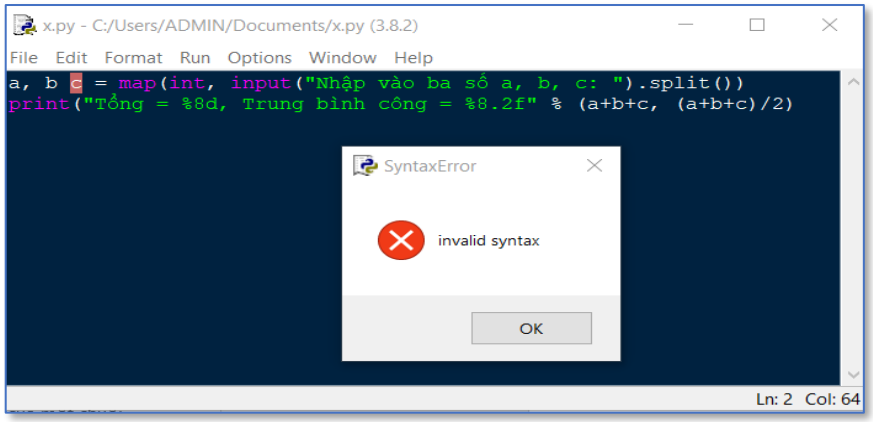
- ***Soạn thảo***: một chương trình, bạn chọn menu *File* → *New File* (*CTRL + N*), xuất hiện màn hình soạn thảo như sau:



Hình 2.8 - Minh họa cửa sổ soạn thảo chương trình của công cụ Python IDLE

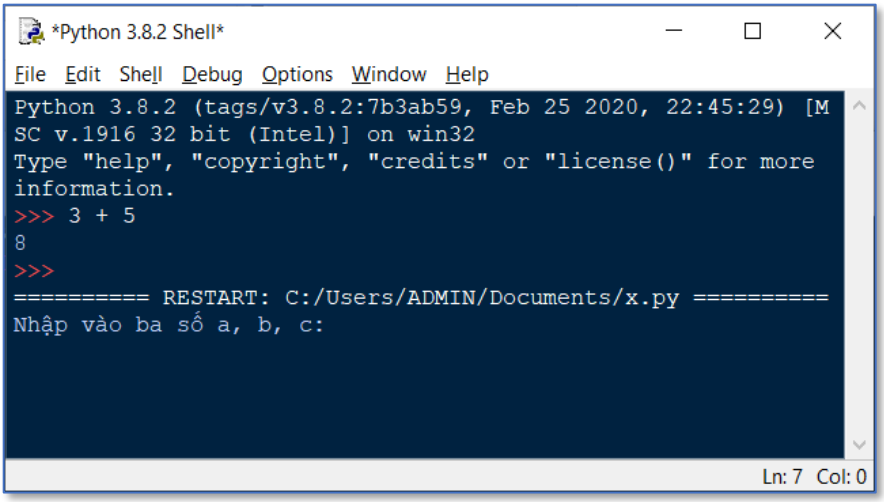
Trong cửa sổ này bạn có thể soạn thảo các lệnh của chương trình như cách soạn thảo văn bản thông dụng và lưu chương trình vào đĩa bằng chức năng menu *File* → *Save* (phím tắt **CTRL + S**) nhập tên chương trình và nhấn **Enter** (phần tên mở rộng ngầm định của tệp chương trình là **.py**). Các chức năng soạn thảo như Copy, Cut, Past, Find, Replace có sẵn và tương tự như trong chương trình soạn thảo Notepad của Windows.

- *Dịch và chạy chương trình*: Sau khi soạn thảo xong chương trình, để dịch và chạy thử chương trình, bạn nhấn phím **F5** hoặc chọn menu **Run → Run Modulle**. Nếu chương trình có lỗi cú pháp, phần mềm sẽ hiển thị thông báo lỗi.



Hình 2.9 - Chương trình trên có lỗi cú pháp

Cần phải sửa lỗi nếu có, lưu lại chương trình rồi tiến hành biên dịch lại cho tới khi không còn lỗi nữa. Nếu không còn lỗi nào nữa thì chương trình sẽ thực thi các câu lệnh một cách tuần tự.



Hình 2.10 - Màn hình giao tiếp của chương trình sau khi đã hết lỗi

- *Đóng cửa sổ chương trình*: Nhấn nút đóng cửa sổ soạn thảo chương trình (Close) hoặc bấm tổ hợp phím ALT + F4.
- *Thoát khỏi môi trường Python IDLE*: Từ cửa sổ lệnh Python Shell bấm nút đóng cửa sổ hoặc bấm ALT + F4 hoặc chọn menu File → Exit hoặc bấm CTRL + Q.

TÓM TẮT

- Dữ liệu của bài toán được biểu diễn thông qua biến trong chương trình theo các quy tắc của ngôn ngữ lập trình cụ thể.
- Kiểu dữ liệu của mọi ngôn ngữ lập trình chỉ cho phép mô tả các đại lượng rời rạc và hữu hạn.

- Một chương trình thường có hai phần: Phần khai báo và phần thân chương trình. Phần khai báo có thể có hoặc không.
- Kiểu dữ liệu chuẩn: Kiểu nguyên, kiểu thực, kiểu chuỗi kí tự, kiểu lôgic.
- Các biến đều phải được khai báo và mỗi biến chỉ khai báo một lần.
- Các phép toán: số học, quan hệ và lôgic.
- Có ba loại biểu thức: số học, quan hệ và lôgic.
- Các ngôn ngữ lập trình có:
- Lệnh gán dùng để gán giá trị của biểu thức cho biến.
- Các thủ tục chuẩn dùng để đưa dữ liệu vào và ra.

Bài tập và thực hành 1

1. Mục đích, yêu cầu

- Giới thiệu một chương trình Python hoàn chỉnh đơn giản;
- Làm quen với một số dịch vụ cơ bản của Python IDLE trong việc soạn thảo, lưu trữ, dịch và thực hiện chương trình.

2. Nội dung

a) Gõ chương trình sau:

```
1. # Giải phương trình bậc 2 #
2.
3. import math
4. a, b, c = map(int, input('a, b, c: ').split())
5. d = b**2 - 4*a*c
6. x1 = (-b - math.sqrt(d))/(2*a)
7. x2 = -b/a - x1
8. print("x1 = %6.2f x2 = %6.2f" % (x1, x2))
```

Chú ý

- Mỗi câu lệnh được viết trên một dòng riêng biệt và không có dấu chấm phảy
- Khai báo sử dụng hàm số học (math) và cách gọi hàm tính căn bậc 2

b) Nhấn phím *CTRL + S* và lưu chương trình với tên là PTB2.PY lên đĩa.

c) Nhấn tổ hợp phím *F5* để dịch và sửa lỗi cú pháp (nếu có).

d) Khi hết lỗi tiếp tục nhấn phím *F5* để thực hiện chương trình. Nhập các giá trị 1; -3 và 2. Quan sát kết quả hiển thị trên màn hình *(x1 = 1.00 x2 = 2.00*).

e) Nhấn phím *F5* rồi nhập các giá trị 1 0 -2. Quan sát kết quả hiển thị trên màn hình (*x1 = -1.41 x2 = 1.41*).

f) Sửa lại chương trình trên sao cho không dùng biến trung gian *d*. Thực hiện chương trình đã sửa với các bộ dữ liệu trên.

g) Sửa lại chương trình nhận được ở mục c bằng cách thay đổi công thức tính *x2* (có hai cách để tính *x2*).

h) Thực hiện chương trình đã sửa với bộ dữ liệu 1; -5; 6. Quan sát kết quả trên màn hình (*x1 = 2 x2 = 3*).

i) Thực hiện chương trình với bộ dữ liệu 1; 1 ; 1 và quan sát kết quả trên mànhình.

Câu hỏi và bài tập

1. Hãy cho biết sự khác nhau giữa hằng có đặt tên và biến.

2. Tại sao phải dùng biến?

3. Trong Python, để một biến có kiểu nguyên thì phải làm thế nào?

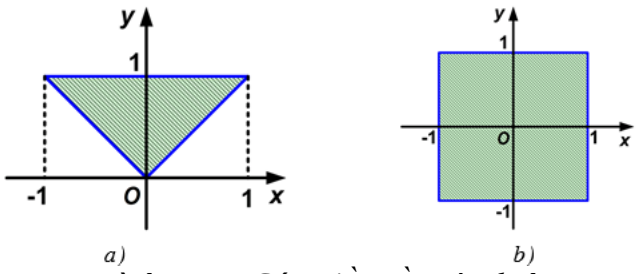
4. Hãy viết biểu thức toán học dưới đây trong Python:

$$(1+z)\frac{x+\frac{y}{z}}{a-\frac{1}{1+x^3}}$$

5. Hãy chuyển các biểu thức trong Python dưới đây sang biểu thức toán học tươngứng:

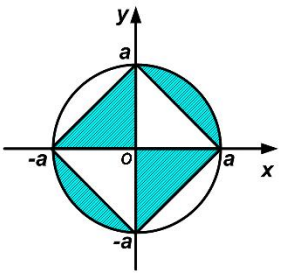
```
a) a/b*2;      b) a*b*c/2;      c) 1/a*b/c;    d) b/sqrt(a*a+b)
e) a**2 + 2*a*b + b**2          f) (x + y)**3  g) exp(5)
```

6. Hãy viết biểu thức lôgic cho kết quả *True* khi toạ độ (*x;y*) là điểm nằm trong vùng gạch chéo kể cả biên của các hình 2.11a và 2.11b.



Hình 2.11 - Các miền cần xác định

9. Hãy viết chương trình nhập số *a* (*a* > 0) rồi tính và đưa ra diện tích phần được gạch chéo trong hình 2.12 (kết quả làm tròn đến bốn chữ số thập phân).



Hình 2.12

10. Lập trình tính và đưa ra màn hình vận tốc *v* khi chạm đất của một vật rơi từ độ cao *h*, biết rằng $v = \sqrt{gh}$, trong đó *g* là gia tốc rơi tự do và *g* = 9,8m/s2. Độ cao *h* (m) được nhập vào từ bàn phím.

Chương 3. Cấu trúc rẽ nhánh và lặp

Bài 9. Cấu trúc rẽ nhánh

1. Rẽ nhánh

Có rất nhiều việc chỉ được thực hiện khi một điều kiện cụ thể nào đó được thoả mãn.

Ví dụ, Châu và Ngọc thường cùng nhau chuẩn bị các bài thực hành môn Tinhọc. Một lần Châu hẹn với Ngọc: "*Chiều mai nếu trời không mưa thì Châu sẽ đến nhà Ngọc*".

Một lần khác, Ngọc nói với Châu: "*Chiều mai nếu trời không mưa thì Ngọc sẽ đến nhà Châu, nếu mưa thì sẽ gọi điện cho Châu để trao đổi*".

Câu nói của Châu cho ta biết một việc làm cụ thể (*Châu đến nhà Ngọc*) sẽ được thực hiện nếu một điều kiện cụ thể (*trời không mưa*) thoả mãn. Ngoài ra không đề cập đến việc gì sẽ xảy ra nếu điều kiện đó không thoả mãn (trời mưa).

Cách diễn đạt như vậy ta nói thuộc dạng mệnh đề thiếu:

Nếu... thì…

Câu nói của Ngọc khẳng định một trong hai việc cụ thể (*Ngọc đến nhà Châu* hay *Ngọc gọi điện cho Châu*) chắc chắn sẽ xảy ra. Tuy nhiên, việc nào trong hai việc sẽ được thực hiện thì tuỳ thuộc vào điều kiện cụ thể (*trời không mưa*) thoả mãn hay không.

Cách diễn đạt như vậy ta nói thuộc dạng mệnh đề đủ:

Nếu… thì…, nếu không thì…

Từ đó có thể thấy, trong nhiều thuật toán, các thao tác tiếp theo sẽ phụ thuộc vào kết quả nhận được từ các bước trước đó.

Cấu trúc dùng để mô tả các mệnh đề có dạng như trên được gọi là cấu trúc rẽ nhánh.

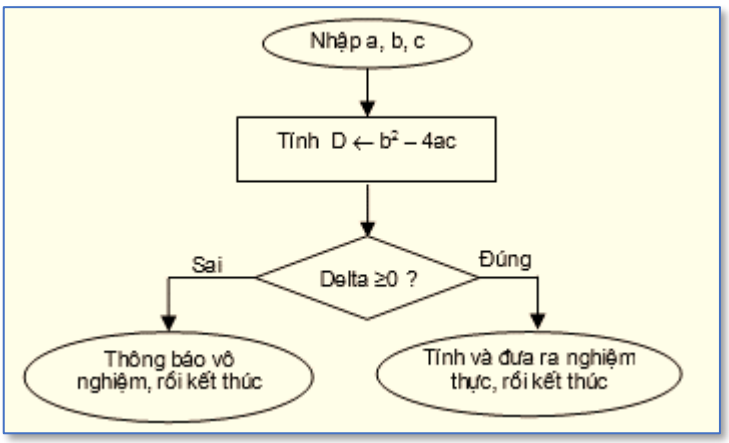
Ví dụ, để giải phương trình bậc hai:

ax2 + bx + c = 0, (a 0)

trước tiên ta tính biệt số delta $D = b^2 – 4ac$.

Nếu *D* không âm, ta sẽ đưa ra các nghiệm. Trong trường hợp ngược lại, ta phải thông báo là phương trình vô nghiệm.

Như vậy, sau khi tính *D*, tuỳ thuộc vào giá trị của *D*, một trong hai thao tác sẽ được thực hiện (hình 4). Mọi ngôn ngữ lập trình đều có các câu lệnh để mô tả cấu trúc rẽ nhánh.



Hình 3.1 - Sơ đồ thể hiện cấu trúc rẽ nhánh

2. Câu lệnh if

Để mô tả cấu trúc rẽ nhánh, Python dùng câu lệnh ***if***. Tương ứng với hai dạng mệnh đề thiếu và đủ nói ở trên, Python có hai dạng câu lệnh ***if***:

a) Dạng thiếu

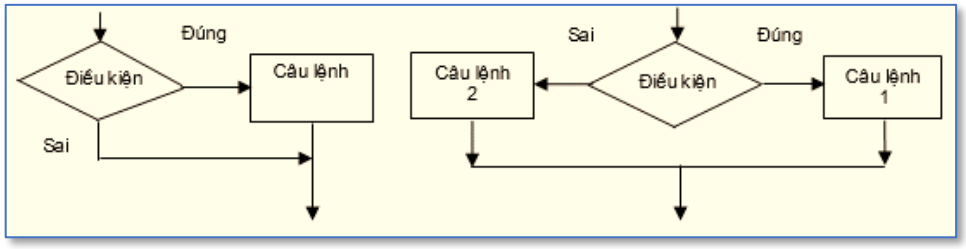
```
if <điều kiện> :
    <câu lệnh >
```

b) Dạng đủ

```
if <điều kiện> :
    <câu lệnh 1>
else:
    <câu lệnh 2>
```

trong đó:

- *Điều kiện*: Biểu thức quan hệ hoặc lôgic có kết quả dạng *True/False*.
- Câu lệnh, câu lệnh 1, câu lệnh 2 là một câu lệnh Python.



Hình 3.2 - Lưu đồ biểu diễn hai dạng lệnh if

- *Ở dạng thiếu*: *điều kiện* sẽ được tính và kiểm tra. Nếu *điều kiện* đúng (có giá trị *True*) thì câu lệnh sẽ được thực hiện, ngược lại thì câu lệnh sẽ bị bỏ qua (hình 3.2 trái).
- *Ở dạng đủ*: *điều kiện* cũng được tính và kiểm tra. Nếu *điều kiện* đúng thì câu lệnh 1 sẽ được thực hiện, ngược lại thì câu lệnh 2 sẽ được thực hiện (hình 3.2 phải).

Ví dụ 1.

```
1. if Delta < 0 :
2.     print('Phuong trinh vo nghiem.')
```

Ví dụ 2.

```
1. if a % 3 == 0:
2.     print('a chia het cho 3')
3. else :
4.     print('a khong chia het cho 3')
```

Ví dụ 3.

Để tìm số lớn nhất *max* trong hai số *a* và *b*, có thể thực hiện bằng hai cách sau:

- Dùng câu lệnh gán *max* = *a* và lệnh *if* dạng thiếu:

```
1. if b > a :
2.     max = b
```

- Dùng một lệnh *if* dạng đủ:

```
1. if b > a :
2.     max = b
3. else:
4.     max = a
```

3. Câu lệnh ghép

Theo cú pháp, sau một số từ khoá (như *if* và *else*) phải là một câu lệnh. Nhưng trong nhiều trường hợp, các thao tác sau những tên dành riêng đó khá phức tạp, đòi hỏi không phải chỉ một mà là nhiều câu lệnh để mô tả. Trong các trường hợp như vậy, ngôn ngữ lập trình cho phép gộp một dãy câu lệnh thành một câu lệnh ghép (hay câu lệnh hợp thành). Sau một mệnh đề điều khiển nào đó của Python, câu lệnh ghép có dạng được viết thụt lề so với mệnh đề đó:

Câu lệnh, Câu lệnh 1, Câu lệnh 2 trong các cú pháp *if* ở trên có thể là câu lệnh ghép.

Thuật ngữ câu lệnh được hiểu chung cho câu lệnh đơn và câu lệnh ghép.

Ví dụ

```
1. if D < 0 :
2.     print('Phuong trinh vo nghiem.')
3. else:
4.     x1 = (-b – sqrt(b*b – 4*a*c))/(2*a);
5.     x2 = -b/a-x1
```

Trong ví dụ trên lệnh sau mệnh đề *else* là một lệnh ghép gồm hai lệnh đơn được viết thụt lề so với *else*.

4. Một số ví dụ

Ví dụ 1.

Tìm nghiệm thực của phương trình bậc hai:

$$ax^2 + bx + c = 0 \text{ , với } a \neq 0.$$

Input: Các hệ số *a, b, c* nhập từ bàn phím.

Output: Đưa ra màn hình các nghiệm thực hoặc thông báo "Phương trình vô nghiệm".

```
1. # giải phương trình bậc 2
2.
3. import math
4. a, b, c = map(int, input(' a, b, c: ').split())
5. d = b**2 - 4*a*c
6. if d < 0 :
7.     print('Phương trình vô nghiệm.')
8. else:
9.     x1 = (-b - math.sqrt(d))/(2*a)
10.    x2 = -b/a - x1;
11.    print(' x1 = %8.3f x2 = %8.3f' % (x1, x2))
```

Ví dụ 2.

Tìm số ngày của năm *N*, biết rằng năm nhuận là năm chia hết cho 400 hoặc chia hết cho 4 nhưng không chia hết cho 100. Ví dụ, các năm 2000, 2004 là năm nhuận và có số ngày là 366, các năm 1900, 1945 không phải là năm nhuận và có số ngày là 365.

Input: *N* nhập từ bàn phím.

Output: Đưa số ngày của năm *N* ra màn hình.

```
1. # Tính ngày của năm N
2.
3. n = int(input('Năm: '))
4. sn = 365
5. if (n % 400 == 0) or (n % 4 == 0 and n % 100 != 0):
6.     sn = 366
7. print('Số ngày của năm %d là %d' % (n, sn))
```

Bài 10. Cấu trúc lặp

1. Lặp

Với *a* là số nguyên và *a* > 2, xét các bài toán sau đây:

Bài toán 1. Tính và đưa kết quả ra màn hình tổng

$$S = \frac{1}{a} + \frac{1}{a+1} + \frac{1}{a+2} + \cdots + \frac{1}{a+100}$$

Bài toán 2. Tính và đưa kết quả ra màn hình tổng

$$S = \frac{1}{a} + \frac{1}{a+1} + \frac{1}{a+2} + \cdots + \frac{1}{a+N} + \cdots$$

cho đến khi $\frac{1}{a+N} < 0.0001$

Với cả hai bài toán, dễ thấy cách để tính tổng *S* có nhiều điểm tương tự:

- Xuất phát, *S* được gán giá trị $\frac{1}{a}$
- Tiếp theo, cộng vào tổng *S* một giá trị $\frac{1}{a+N}$ với *N* = 1, 2, 3, 4, 5,… việc cộng này được lặp lại một số lần.

Đối với bài toán 1, số lần lặp là 100 và việc cộng vào tổng *S* sẽ kết thúc khi đã thực hiện việc cộng 100 lần.

Đối với bài toán 2, số lần lặp chưa biết trước nhưng việc cộng vào tổng *S* sẽ kết thúc khi điều kiện $\frac{1}{a+N} < 0.0001$ được thoả mãn.

Nói chung, trong một số thuật toán có những thao tác phải thực hiện lặp đi lặp lại một số lần. Một trong các đặc trưng của máy tính là có khả năng thực hiện hiệu quả các thao tác lặp. Cấu trúc lặp mô tả thao tác lặp và được phân biệt hai loại là *lặp với số lần biết trước* và *lặp với số lần chưa biết trước*.

Các ngôn ngữ lập trình đều có các câu lệnh để mô tả cấu trúc điều khiển lặp.

2. Lặp có số lần lặp biết trước và câu lệnh lặp `for`

Có hai thuật toán *Tong_1a* và *Tong_1b* để giải bài toán 1 như sau:

Thuật toán Tong_1a

```
Bước 1. S ← 1/a; N ← 0;  {Khởi tạo S và N}
Bước 2. N ← N + 1;
Bước 3. Nếu N > 100 thì chuyển đến bước 5;
Bước 4. S ← S + 1/(a + N) rồi quay lại bước 2;
Bước 5. Đưa S ra màn hình, rồi kết thúc.
```

Thuật toán Tong_1b

```
Bước 1. S ← 1/a; N ← 101;    {Khởi tạo S và N}
Bước 2. N ← N – 1;
Bước 3. Nếu N < 1 thì chuyển đến bước 5;
Bước 4. S ← S + 1/(a + N) rồi quay lại bước 2;
Bước 5. Đưa S ra màn hình rồi kết thúc.
```

Lưu ý, số lần lặp của cả hai thuật toán trên là biết trước và như nhau (100 lần).

Trong thuật toán *Tong_1a*, giá trị *N* khi bắt đầu tham gia vòng lặp là 1 và sau mỗi lần lặp *N* tăng lên 1 cho đến khi *N* > 100 (*N* = 101) thì kết thúc lặp (thực hiện đủ 100 lần). Trong thuật toán *Tong_1b*, giá trị *N* bắt đầu tham gia vòng lặp là 100 và sau mỗi lần lặp *N* giảm đi 1 cho đến khi *N* < 1 (*N* = 0) thì kết thúc lặp (thực hiện đủ 100 lần). Ta nói cách lặp trong thuật toán *Tong_1a* là dạng tiến và trong thuật toán *Tong_1b* là dạng lùi.

Để mô tả cấu trúc lặp với số lần biết trước, Python dùng câu lệnh ***for*** cú pháp sau kết hợp với phép toán ***in*** và hàm ***range()*** (một số cú pháp ***for*** khác sẽ được nêu trong bài 11, 12).

```
for <biến đếm> in range([giá trị đầu], <giá trị cuối>, [bước nhảy]):
    <lệnh>
```

Trong đó:

- *biến đếm* là biến đơn, thường có kiểu nguyên;
- *giá trị đầu*, *giá trị cuối* là các biểu thức cùng kiểu với *biến đếm*;
- Nếu *bước nhảy* > 0 thì *giá trị đầu* phải nhỏ hơn *giá trị cuối*. Nếu *giá trị đầu* không nhỏ hơn *giá trị cuối* thì vòng lặp không được thực hiện. Nếu *bước nhảy* < 0 thì *giá trị đầu* phải lớn hơn *giá trị cuối*. Nếu *giá trị đầu* không lớn hơn *giá trị cuối* thì vòng lặp không được thực hiện.

Hoạt động của lệnh lặp ***for*** trong cú pháp trên:

- Câu *lệnh* sau ***for*** được thực hiện tuần tự, với *biến đếm* lần lượt nhận giá trị trong phạm vi [*giá trị đầu, giá trị cuối*) (tức là từ *giá trị đầu* đến *giá trị cuối* -1)
- Sau mỗi lên thực hiện *lệnh* thì *biến đếm* được cộng thêm một giá trị là *bước nhảy*.
- Nếu *bước nhảy* < 0 thì ta hiểu là lặp lùi, nếu *bước nhảy* > 0 thì hiểu là lặp tiến.
- Mặc định nếu không nêu tham số *bước nhảy* thì bước nhảy là 1, và nếu không có tham số *giá trị đầu* thì giá trị đầu bằng 0.

Ví dụ 1.

Sau đây là hai chương trình cài đặt các thuật toán *Tong_1a* và *Tong_1b*.

```
1. # Thuật toán tổng a #
2.
3. a = int(input('Hay nhap gia tri a vao!'))
4. s = 1.0/a                    # Buoc 1
5. for n in range(1, 101):              # Buoc 2, Buoc 3
```

```
6.      s = s + 1.0/(a + n)          # Buoc 4
7.  print('Tong S la: %8.4f' % s)    # Buoc 5
```

```
1.  # Thuật toán tổng b #
2.
3.  a = int(input('Hay nhap gia tri a vao!'))
4.  s = 1.0/a                      # Buoc 1
5.  for n in range(100, 0, -1):      # Buoc 2, Buoc 3
6.      s = s + 1.0/(a + n)          # Buoc 4
7.  print('Tong S la: %8.4f' % s)    # Buoc 5
```

Ví dụ 2.

Chương trình sau thực hiện việc nhập từ bàn phím hai số nguyên dương *M* và *N* (*M* < *N*), tính và đưa ra màn hình tổng các số chia hết cho 3 hoặc 5 trong phạm vi từ *M* đến *N*.

```
1.  # Ví dụ 2 #
2.
3.  print("Nhập số M nhỏ hơn số N")
4.  m = int(input('M = '))
5.  n = int(input('N = '))
6.  t = 0
7.  for i in range(m, n+1):
8.      if (i % 3 == 0)or(i % 5 == 0):
9.          t += i
10. print('Kết quả: ',t)
```

3. Lặp với số lần chưa biết trước và câu lệnh lặp `while`

Có thể xây dựng thuật toán *Tong_2* như sau để giải bài toán 2.

Thuật toán Tong_2

```
Bước 1. S ← 1/a; N ← 0;  {Khởi tạo S và N}
Bước 2. Nếu 1/(a  + N) < 0,0001 thì chuyển đến bước 5;
Bước 3. N ← N + 1;
Bước 4. S ← S + 1/(a + N) rồi quay lại bước 2.
Bước 5. Đưa S ra màn hình, rồi kết thúc
```

Như vậy, việc lặp với số lần chưa biết trước sẽ chỉ kết thúc khi một điều kiện cho trước được thoả mãn.

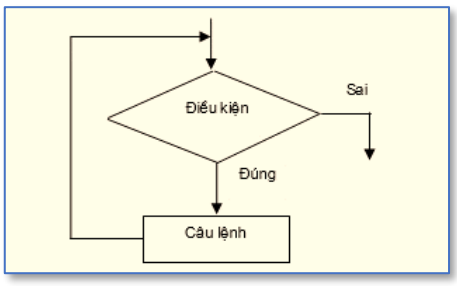
Để mô tả cấu trúc lặp như vậy, Python dùng câu lệnh *while* có dạng:

```
while <điều kiện> :
    <câu lệnh >
```

trong đó:

- *Điều kiện* là biểu thức quan hệ hoặc lôgic có giá trị *True/False*;
- *Câu lệnh* là một câu lệnh của Python.

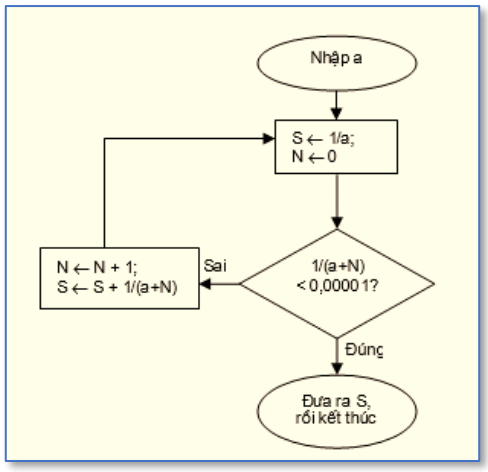
Việc thực hiện lệnh *while* được thể hiện trên sơ đồ ở hình 7.



Hình 3.3 - Sơ đồ lặp với số lần biết trước

Ví dụ 1.

Sau đây là chương trình cài đặt thuật toán Tong_2.



Hình 3.4 - Sơ đồ khối của thuật toán Tong_2

```
1. # Tổng 2 viết bằng while #
2.
3. a = int(input("Hãy nhập giá trị a vào! "))
4. s = 1.0/a                        # Bước 1
5. n = 0
6. while not (1/(a+n) < 0.0001):    # Bước 2
7.     n += 1                       # Bước 3
8.     s += 1.0/(a+n)               # Bước 4
9. print('Tổng S là: %8.4f' % s)    # Bước 5
```

Ví dụ 2.

Tìm ước chung lớn nhất (UCLN) của hai số nguyên dương *M* và *N*.

Có nhiều thuật toán khác nhau tìm UCLN của *M* và *N*. Sau đây là một thuật toán tìm UCLN.

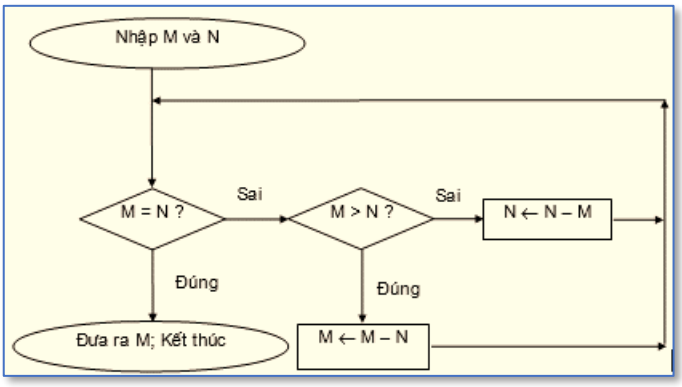
Thuật toán

```
Bước 1.  Nhập M, N;
Bước 2.  Nếu M = N thì lấy M làm UCLN rồi chuyển đến bước5;
Bước 3.  Nếu M > N thì M ơ M - N rồi quay lại bước 2;
Bước 4.  N ơ N - M rồi quay lại bước 2;
Bước 5.  Đưa ra kết quả UCLN rồi kết thúc.
```

Chương trình sau thể hiện thuật toán tìm ước chung lớn nhất.

```
# Tìm ước chung lớn nhất #

m, n = map(int,input("M, N = ").split())
while m != n:
    if m > n:
        m -= n
    else:
        n -= m
print('USCLN = ',m)
```



Hình 3.5 - Sơ đồ khối của thuật toán tìm ước chung lớn nhất

Chú ý

Các câu lệnh trong vòng lặp thường được lặp lại nhiều lần, vì vậy để tăng hiệu quả của chương trình thì những thao tác không cần lặp lại nên đưa ra ngoài vòng lặp.

TÓM TẮT

- Các ngôn ngữ lập trình đều có câu lệnh thể hiện cấu trúc rẽ nhánh và cấu trúc lặp.
- Câu lệnh rẽ nhánh có hai dạng:

  a) Dạng thiếu;

  b) Dạng đủ.
- Có thể gộp dãy câu lệnh thành câu lệnh ghép.
- Các câu lệnh mô tả cấu trúc lặp:

  a) Lặp với số lần biết trước;

  b) Lặp với số lần không biết trước.

Định lí Bohn Jacopini (Bon Ja-co-pi-ni): Mọi quá trình tính toán đều có thể thực hiện dựa trên ba cấu trúc cơ bản là cấu trúc tuần tự, cấu trúc rẽ nhánh và cấu trúc lặp.

Bài tập và thực hành 2

1. Mục đích, yêu cầu

- Xây dựng chương trình có sử dụng cấu trúc rẽ nhánh;
- Làm quen với việc hiệu chỉnh chương trình.

2. Nội dung

Bài toán. Bộ số Pi-ta-go

Biết rằng bộ ba số nguyên dương *a, b, c* được gọi là bộ số Pi-ta-go nếu tổng các bình phương của hai số bằng bình phương của số còn lại. Viết chương trình nhập từ bàn phím ba số nguyên dương *a, b, c* và kiểm tra xem chúng có là bộ số Pi-ta-go hay không.

Ý tưởng: Kiểm tra xem có đẳng thức nào trong ba đẳng thức sau đây được thực hiện hay không:

$$a^2 = b^2 + c^2$$
$$b^2 = a^2 + c^2$$
$$c^2 = a^2 + b^2$$

Những công việc cần thực hiện:

a) Gõ chương trình sau:

```
1. # Bộ số Pi-ta-go #
2.
3. a, b, c = map(int,input("a, b, c: ").split())
4. a2 = a
5. b2 = b
6. c2 = c
7. a2 = a*a
8. b2 = b*b
9. c2 = c*c
10.if a2 == b2 + c2 or b2 == a2 + c2 or c2 == a2 + b2:
11.    print("Ba số đã nhập là bộ số Pi-ta-go")
12.else:
13.    print("Ba số đã nhập không là bộ số Pi-ta-go")
```

Chú ý:

Sau ***else*** có dấu hai chấm (:).

a) Lưu chương trình với tên PITAGO lên đĩa.

b) Nhập các giá trị *a* = 3, *b* = 4, *c* = 5.

c) Lặp lại các bước trên với bộ dữ liệu *a* = 700, *b* = 1000, *c* = 800.

d) Nếu thay dãy lệnh

```
a2 = a
b2 = b
c2 = c
a2 = a*a
```

```
b2 = b*b
c2 = c*c
```

bằng dãy lệnh

```
a2 = a*a
b2 = b*b
c2 = c*c
```

và chạy lại chương trình, thì kết quả có gì thay đổi với bộ dữ liệu cho ở câu c?

e) Nếu thay dãy lệnh

```
a2 = a*a
b2 = b*b
c2 = c*c
```

bằng dãy lệnh

```
a2 = a**2
b2 = b**2
c2 = c**2
```

và chạy lại chương trình, thì kết quả có gì thay đổi với bộ dữ liệu cho ở câu c và d?

Câu hỏi và bài tập

1. Hãy cho biết sự giống và khác nhau của hai dạng câu lệnh ***if*** .

2. Câu lệnh ghép là gì? Tại sao phải có câu lệnh ghép?

3. Có thể dùng câu lệnh while để thay cho câu lệnh for được không? Nếu được, hãy thực hiện điều đó với chương trình *Tong_1a*.

4. Viết câu lệnh ***if*** tính:

a) $z = \begin{cases} x^2 + y^2 & nếu\ x^2 + y^2 \le 1 \\ x + y & nếu\ x^2 + y^2 > 1\ và\ y \ge x \\ 0.5 & nếu\ x^2 + y^2 > 1\ và\ y < x \end{cases}$

b) z = = $\begin{cases} |x| + |y| & nếu\ điểm\ (x,y) thuộc\ hình\ tròn\ bán\ kính\ r\ (r > 0), tâm\ (a;b) \\ x + y & trong\ trường\ hợp\ còn\ lại \end{cases}$

5. Lập trình tính:

a) $Y = \sum_{n=1}^{50} \frac{n}{n+1}$;

b) $e(n) = 1 + \frac{1}{1!} + \frac{1}{2!} + \cdots + \frac{1}{n!} + \cdots$, với *n* lần lượt bằng 3, 4, ..., cho đến khi $\frac{1}{n!} < 2 \times 10^{-6}$. Đưa các giá trị *e(n)* ra màn hình.

6. Lập trình giải bài toán cổ sau:

Vừa gà vừa chó.

Bó lại cho tròn.

Ba mươi sáu con,

Một trăm chân chẵn.

Hỏi bao nhiêu con mỗi loại?

7. Nhập từ bàn phím tuổi của cha và con (tuổi của cha hơn tuổi con ít nhất là25). Đưa ra màn hình bao nhiêu năm nữa thì tuổi cha gấp đôi tuổi con.

8. Một người gửi tiết kiệm không kì hạn với số tiền A đồng với lãi suất 0,2% mỗi tháng. Hỏi sau t tháng, người đó rút tiền thì sẽ nhận được số tiền là bao nhiêu. Biết rằng tiền gửi tiết kiệm không kì hạn không được tính lãi kép.

Chương 4. Kiểu dữ liệu có cấu trúc

Bài 11. Kiểu danh sách

1. Danh sách một chiều

a) Khái niệm kiểu danh sách

Python nguyên bản không hỗ trợ kiểu mảng, thay vào đó người ta dùng kiểu danh sách (list). Trong chương trình Tin 11 có học chút ít về mảng 1 chiều, vì vậy tôi khuyến nghị các bạn nên tìm hiểu về kiểu danh sách của Python thay thế cho kiểu mảng trong SGK Tin 11.

Trong Python, danh sách là một dãy các phần tử không nhất thiết cùng kiểu, có thứ tự (được đánh số từ 0) và có thể thay đổi giá trị. Các phần tử của danh sách được viết phân cách nhau bằng dấu phảy và toàn bộ danh sách được đặt trong cặp dấu ngoặc vuông [].

b) Cách khai báo và khởi tạo danh sách

- Để khai báo một danh sách rỗng ta dùng cú pháp sau:

```
<tên danh sách> = []
```

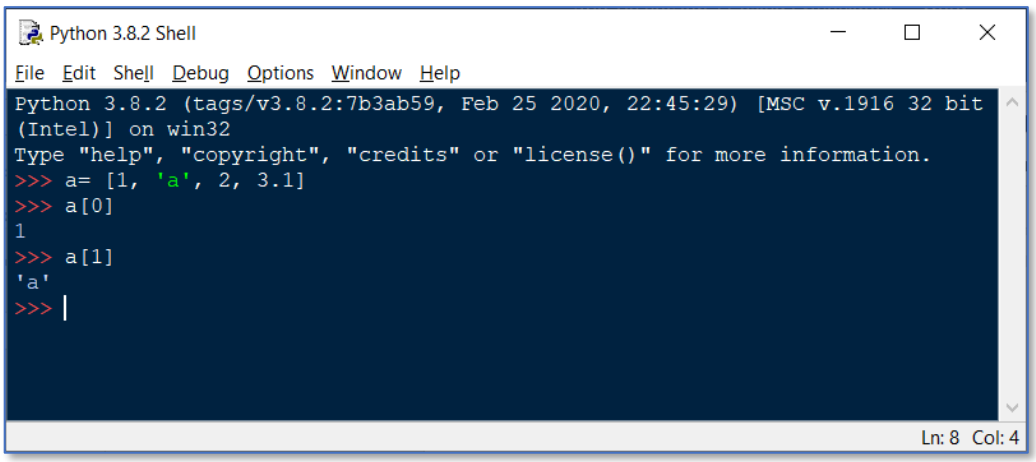
- Để khai báo một danh sách và khởi tạo sắn một số phần tử ta dùng cú pháp sau:

```
<tên danh sách> = [<danh sách phần tử, phân cách bởi dấu phảy>]
```

Ví dụ:

- Khởi tạo một danh sách rỗng *a = []*
- Khởi tạo danh sách có 5 phần tử *b = [1, 2, 3, 4, 5]*
- Khởi tạo danh sách *c* gồm 100 số 0, *c = [0]*100*

Ví dụ trong hình dưới cho thấy phần tử đầu tiên *a[0]* = 1 là một số nguyên, nhưng *a[1]* là chuỗi chứa một chữ cái, *a[2]* là phần tử thứ 3 lại chứa một giá trị thực.



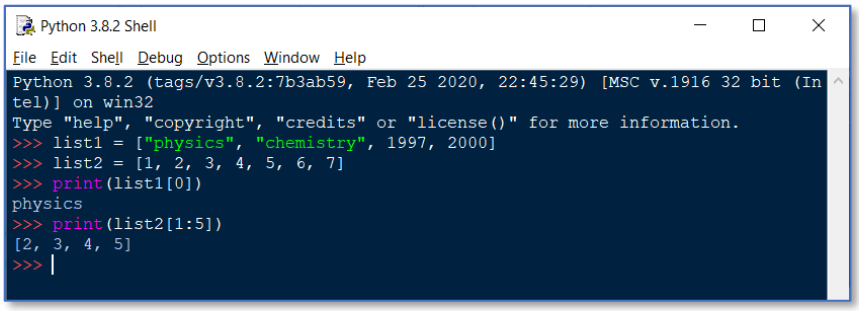
Hình 4.1 - Minh họa việc khai báo và khởi tạo danh sách

c) Các thao tác với danh sách

c.1) Truy xuất phần tử của danh sách

- Để truy cập đến một phần tử ta sử dụng chỉ số đặt trong ngoặc vuông (chú ý phần tử đầu tiên của danh sách có chỉ số là 0). Trong hình 4.1, ta thấy *a[0]* là phần tử đầu tiên của danh sách
- Hãy gõ các dòng lệnh sau vào cửa sổ Python Shell và quan sát kết quả:

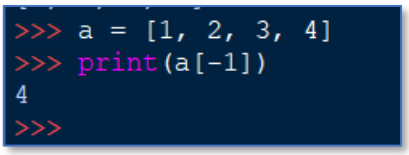
```
list1 = ["physics", "chemistry", 1997, 2000]
list2 = [1, 2, 3, 4, 5, 6, 7]
print(list1[0])
print(list2[1:5])
```



Hình 4.2 - Minh họa các truy xuất phần tử danh sách

Dòng lệnh thứ 3 trong hình in ra phần tử đầu tiên của *list1*, dòng lệnh thứ 4 in ra một đoạn con tronh danh sách *list2*.

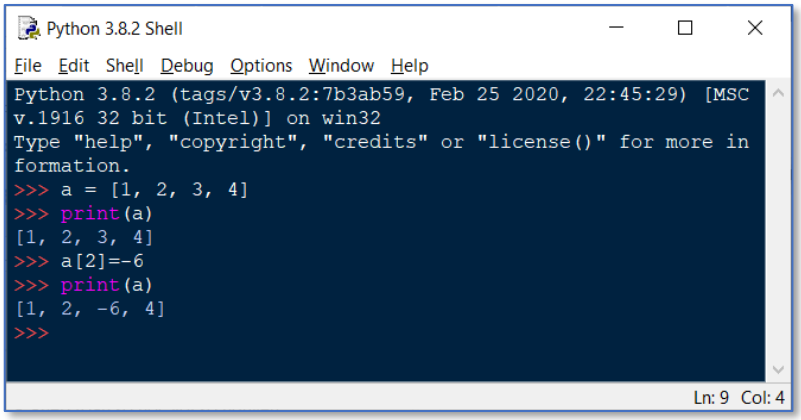
- Đoạn mã dưới cho thấy, Python hỗ trợ sử dụng chỉ số âm để truy cập đến các phần tử của danh sách. Chỉ số -1 trỏ đến phần tử cuối trong danh sách *a*.



Hình 4.3 - Truy xuất đến phần tử cuối danh sách bằng chỉ số -1

c.2) Thay đổi giá trị phần tử, thêm phần tử vào danh sách Python

- Ta có thể thay đổi giá trị phần tử nào đó của danh sách, hãy xem minh họa dưới đây:

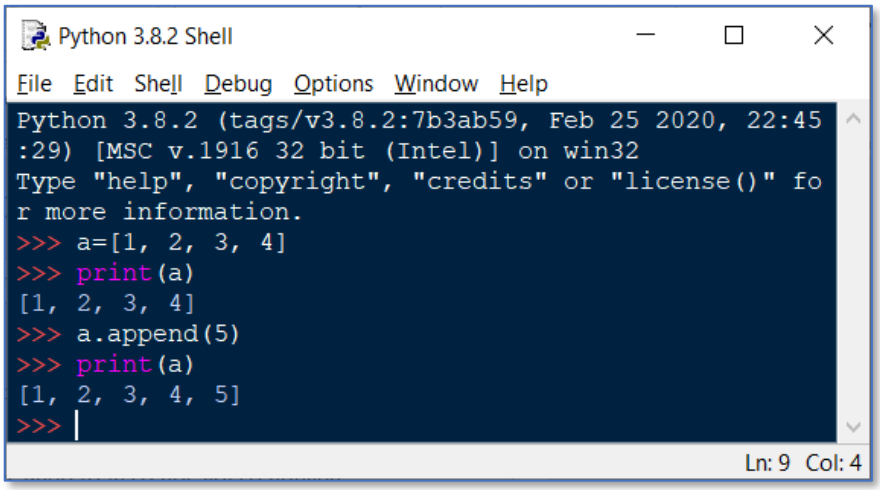


Hình 4.4 - Thay đổi giá trị phần tử thứ 3 trong danh sách a

Dòng lệnh 3 ta đã thay đổi giá trị của phần tử *a*[2] từ 3 thành -6

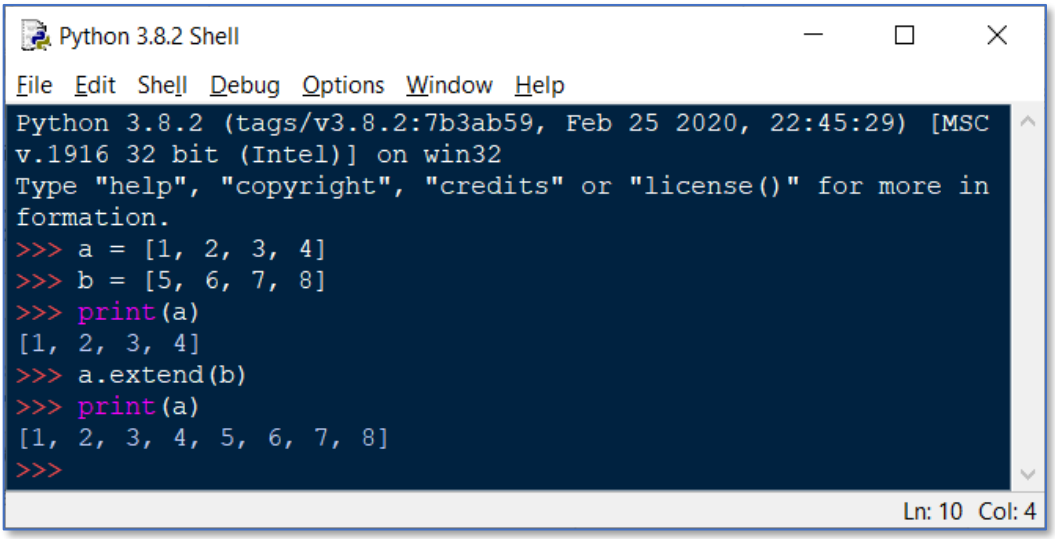
- Thêm phần tử vào cuối danh sách: Danh sách trong Python cho phép thay đổi kích thước, nên trong quá trình thực hiện chương trình, ta có thể thêm phần tử vào cuối danh sách bằng hàm thành viên *append()* của danh sách.

  Ví dụ: Sau khi thực hiện đoạn mã dưới bên trái, ta có kết quả như hình dưới bên phải



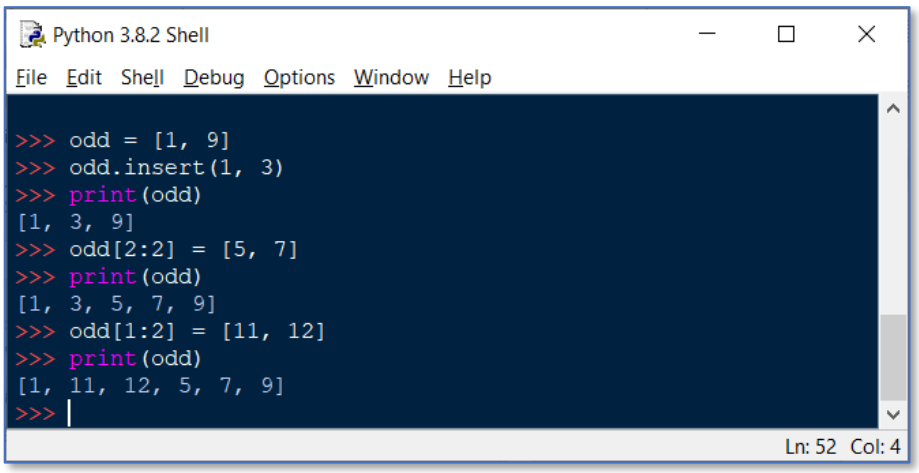
Hình 4.5 - Minh họa việc thêm phần tử vào cuối danh sách bàng lệnh append()

- Ngoài việc thêm từng phần tử vào danh sách, ta còn có thể thêm các phần tử của một danh sách khác vào cuối danh sách hiện tại bằng cách dùng hàm thành viên *extend()*.



Hình 4.6 - Minh họa việc nối danh sách b và cuối danh sách a bằng lệnh extend()

Thậm chí ta có thể chèn thêm một hoặc một số phần tử vào vị trí nào đó trong danh sách bằng hàm thành viên *insert()* hoặc toán tử lát cắt [:] như sau:

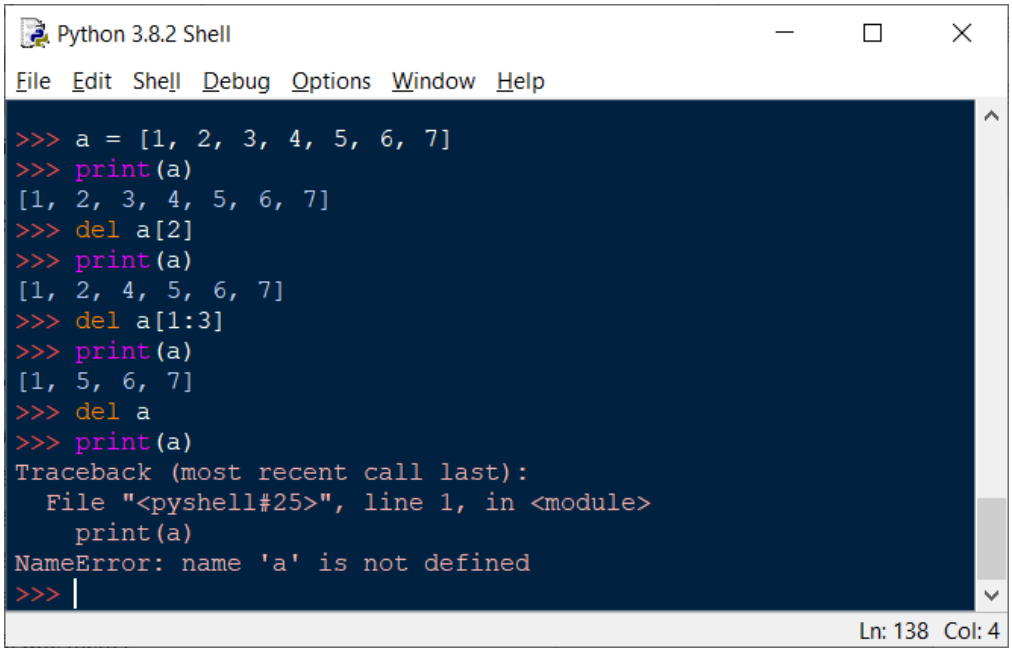


Hình 4.7 - Sử dụng lệnh insert() hoặc toán tử lát cắt để chèn các phần tử vào danh sách

Trong mã trên, dòng thứ 2 gọi hàm *insert(1, 3)* để chèn số 3 vào vị trí 1 trong *odd*. Dòng 4, dùng toán tử lát cắt [2:2] để chèn danh sách [5, 7] vào vị trí 2 trong *odd*. Nhưng dòng 6, toán tử lát cắt [1:2] lại cho phép ghi đè danh sách [11, 12] vào các vị trí 1 và 2 trong danh sách *odd*.

c.3) Xóa hoặc lấy phần tử của danh sách

Để xóa phần tử trong danh sách ta dùng lệnh *del*. Trong hình 4.8, lệnh *del a[2]* xóa phần tử thứ 2 của danh sách *a* hiện tại, lệnh *del a[1:3]* xóa 2 phần tử ở vị trí 1 và 2 trong danh sách *a* hiện tại. Lệnh *del a* xóa tất cả danh sách a, nên lệnh *print(a)* sau đó bị báo lỗi vì *a* không tồn tại nữa.



Hình 4.8 - Minh họa cách sử dụng lệnh del để xóa phần tử và xóa danh sách

Ngoài ra để xóa phần tử cuối danh sách ta gọi hàm thành viên *remove()* hoặc *pop()*. Để xóa tất cả các phần tử trong danh sách ta gọi hàm thành viên *clear()*.

```
>>> a = [1, 2, 3, 4, 5, 6, 7]
>>> print(a)
[1, 2, 3, 4, 5, 6, 7]
>>> a.remove(3)
>>> print(a)
[1, 2, 4, 5, 6, 7]
>>> a.pop(1)
2
>>> print(a)
[1, 4, 5, 6, 7]
>>> a.pop()
7
>>> print(a)
[1, 4, 5, 6]
>>> a.clear()
>>> print(a)
[]
>>>
                                                    Ln: 187
```

Hình 4.9 - Minh họa cách sử dụng lệnh remove(), pop(), clear()

Dòng lệnh *a.remove(3)* xóa phần tử có giá trị 3 đầu tiên tìm thấy tính từ đầu danh sách *a* (nếu không có phần tử nào bằng 3, sẽ có thông báo lỗi). Dòng lệnh *a.pop(1)* lấy ra số 2 ở vị trí 1 trong *a* và đồng thời cũng xóa nó khỏi *a*. Dòng lệnh *a.pop()* lấy ra phần tử ở cuối danh sách *a* và đồng thời cũng xóa phần tử đó khỏi danh sách. Dòng lệnh *a.clear()* xóa tất cả các phần tử của danh sách, kết quả chỉ còn danh sách rỗng *a = []*.

Các bạn có thể tham khảo thêm các hàm khác của danh sách từ internet hoặc từ địa chỉ: https://www.programiz.com/python-programming/methods/list.

Trước khi dừng phần giới thiệu về danh sách Python, các bạn hãy thử chạy đoạn lệnh này và quan sát xem kết quả của nó là gì nhé?

d) Ví dụ

Ví dụ 1. Đo nhiệt độ tuần

```
t1, t2, t3, t4, t5, t6, t7\
= map(int, input("Nhập vào nhiệt độ của 7 ngày: ").split())
tb = (t1 + t2 + t3 + t4 + t5 + t6 + t7)/7
dem = 0
if t1 > tb:
    dem += 1
if t2 > tb:
    dem += 1
if t3 > tb:
    dem += 1
if t4 > tb:
    dem += 1
if t5 > tb:
    dem += 1
if t6 > tb:
    dem += 1
if t7 > tb:
    dem += 1
print("Nhiệt độ trung bình tuần: %4.2f" % tb)
print("Số ngày nhiệt độ cao hơn trung bình: ", dem)
```

Và kết quả chạy thử:

```
Nhập vào nhiệt độ của 7 ngày: 30 32 31 35 30 36 34
Nhiệt độ trung bình tuần: 32.57
Số ngày nhiệt độ cao hơn trung bình:  3
>>>
```

Chú ý: dòng lệnh 1 của mã trên quá dài, ta ngắt lệnh xuống dòng dưới bằng cách thêm dấu \ vào cuối dòng 1 để chỉ báo cho chương trình dịch biết rằng dòng lệnh đó còn tiếp ở dòng sau.

Ví dụ 2. Đo nhiệt độ ngày

```
n = int(input("Nhập số ngày: "))
tong = 0
nhietdo = [0]*n
for i in range(n):
    nhietdo[i] = int(input("Nhập nhiệt độ ngày "+ str(i+1) + ": "))
    tong += nhietdo[i]
dem = 0
tb = tong/n
for i in range(n):
    if nhietdo[i] > tb:
        dem += 1
print("Nhiệt độ trung bình tuần: %4.2f" % tb)
print("Số ngày nhiệt độ cao hơn trung bình: ", dem)
```

Kết quả chạy thử:

```
Nhập số ngày: 5
Nhập nhiệt độ ngày 1: 32
Nhập nhiệt độ ngày 2: 31
Nhập nhiệt độ ngày 3: 34
Nhập nhiệt độ ngày 4: 30
Nhập nhiệt độ ngày 5: 35
Nhiệt độ trung bình tuần: 32.40
Số ngày nhiệt độ cao hơn trung bình:  2
>>>
```

Chú ý: Trong mã trên ở dòng 3 ta khai báo một danh sách *nhietdo* gồm *n* phần tử (nhớ là các phần tử được đánh chỉ số từ 0). Ban đầu, các phần tử đều có giá trị 0.

Trong thực tế lập trình viên Python có thể viết ngắn hơn. Một trong các cách đó có thể như sau:

```
nhietdo = list(map(int,input("Nhập dãy nhiệt độ: ").strip().split()))
n = len(nhietdo)
tb = sum(nhietdo)/n
dem = 0
for x in nhietdo:
    if x > tb:
        dem += 1
print("Nhiệt độ trung bình tuần: %4.2f" % tb)
print("Số ngày nhiệt độ cao hơn trung bình: ", dem)
```

Kết quả chạy thử:

```
*** Remote Interpreter Reinitialized  ***
Nhập dãy nhiệt độ: 30 31 33 24 46
Nhiệt độ trung bình tuần: 32.80
Số ngày nhiệt độ cao hơn trung bình:  2
```

Trong cách viết thứ 2:

+ Dòng 1: Hàm *input()* nhận từ bàn phím một dãy tùy ý các giá trị nhiệt độ từ đầu vào như một chuỗi ký tự phân cách bởi dấu cách, hàm *strip()* sẽ cắt bỏ các dấu trắng hai đầu chuỗi, hàm *split()* sẽ tách chuỗi thành các chuỗi con gồm các chữ số dựa vào dấu hiệu là các khoảng trắng. Hàm *map()* sẽ gửi mỗi chuỗi con sau khi tách cho hàm *int()* chuyển nó thành con số. Cuối cùng hàm *list()* sẽ nhận toàn bộ các số này và biến nó thành một danh sách các số sau đó gán cho biến *nhietdo*. Sau dòng lệnh 1, ta đã có một danh sách các giá trị nhiệt độ lưu trong biến danh sách *nhietdo*.

+ Dòng lệnh 2: hàm *len(nhietdo)* cho ta số phần tử của danh sách và gán vào *n*.

+ Dòng lệnh 3: hàm *sum*(*nhietdo*) cho ta tổng giá trị các phần tử của danh sách *nhietdo*, giá trị này sau khi chia cho *n* sẽ được giá trị nhiệt độ trung bình và được gán cho biến *tb*.

+ Dòng lệnh 5: Sử dụng cách duyệt phần tử bằng lệnh lặp *for* kết hợp với mệnh đề *in*, cách duyệt này ta không cần quan tâm đến chỉ số của danh sách, mỗi phần tử trong danh sách khi được duyệt đến thì được gửi cho *x*, ta kiểm tra thấy *x* thỏa mãn thì tăng biến *dem*.

+ Phần còn lại của mã thì chắc bạn đã hiểu rồi chứ.

Ví dụ 3. Tìm giá trị lớn nhất

Input: Số nguyên dương N ($N \leq 250$) và dãy N số nguyên dương a_1, a_2 ,..., a_N, mỗi số đều không vượt quá 500.

Output: Chỉ số và giá trị của phần tử lớn nhất trong dãy số đã cho (nếu có nhiều phần tử lớn nhất chỉ cần đưa ra một trong số chúng).

```
01 a = list(map(int,input("Nhập các phần tử của dãy: ").strip().split()))
   n = len(a)
   ma = a[0]
   cs = 0
   for i in range(n):
       if ma < a[i]:
           ma = a[i]
           cs = i
   print("Giá trị của phần tử max: ", ma)
10 print("Chỉ số của phần tử max: ", cs)
```

Kết quả chạy thử:

```
Nhập các phần tử của dãy: 23 62 58 91 37
Giá trị của phần tử max:  91
Chỉ số của phần tử max:  3
>>>
```

Chú ý: chỉ số của phần tử *max = 91* trong dãy trên là *3* vì số thứ tự phần tử trong danh sách luôn được tính từ 0.

Ví dụ 4. Sắp xếp dãy số nguyên bằng thuật toán tráo đổi

Input: Số nguyên dương N ($N \leq 250$) và dãy A gồm N số nguyên dương $a_1, a_2,..., a_N$, mỗi số đều không vượt quá 500.

Output: Dãy số A đã được sắp xếp thành dãy không giảm.

Để giải bài toán này, ta sẽ sử dụng thuật toán tráo đổi như mô tả trong SGK Tin học 10.

```
# Sắp xếp tráo đổi #

n = int(input("Số lượng phần tử của dãy số, N = "))
a = [0]*n
for i in range(n):
    a[i] = int(input("Phần tử thứ " + str(i) + " = "))

for j in range(n-1, 0, -1):
    for i in range(j):
        if a[i] > a[i+1]:
            a[i], a[i+1] = a[i+1], a[i]   # lệnh tráo đổi giá trị
for i in range(n):
    print(a[i],end=' ')
```

Ví dụ 5. Tìm kiếm nhị phân

Input: Dãy A là dãy tăng gồm N ($N \leq 250$) số nguyên dương $a_1, a_2,..., a_N$ và số nguyên k.

Output: Chỉ số i mà $a_i = k$ hoặc thông báo "*Không tìm thấy*" nếu không có số hạng nào của dãy A có giá trị bằng k.

Để giải bài toán này, chúng ta sẽ sử dụng thuật toán tìm kiếm nhị phân được trình bày trong SGK Tin học 10.

Bước 1. Nhập N, các số hạng $a_1, a_2,..., a_N$ và khoá k;

Bước 2. $Dau \leftarrow 1, Cuoi \leftarrow N$;

Bước 3. $Giua \leftarrow \left[\frac{Dau + Cuoi}{2}\right]$;

Bước 4. Nếu $a_{Giua} = k$ thì thông báo chỉ số *Giua*, rồi kết thúc;

Bước 5. Nếu $a_{Giua} > k$ thì đặt $Cuoi = Giua - 1$ rồi chuyển đến bước 7;

Bước 6. $Dau \leftarrow Giua + 1$;

Bước 7. Nếu $Dau > Cuoi$ thì thông báo dãy A không có số hạng có giá trị bằng k, rồi kết thúc;

Bước 8. Quay lại bước 3.

```
# Tìm kiếm nhị phân #

n = int(input("Số lượng phần tử của dãy số, N = "))
a = [0]*n
for i in range(n):
    a[i] = int(input("Phần tử thứ " + str(i) + " = "))
```

```
k = int(input("Số giá trị k = "))
dau = 0
cuoi = n-1
tim_thay = False
while (dau <= cuoi)and(not tim_thay):
    giua = (dau + cuoi)//2
    if a[giua] == k:
        tim_thay = True
    elif a[giua] > k :
        cuoi = giua -1
    else:
        dau = giua + 1
if tim_thay:
    print('Chỉ số tìm được là ', giua)
else:
    print('Không tìm thấy')
```

2. Kiểu mảng hai chiều

Xét bài toán tính và đưa ra màn hình bảng cửu chương.

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90

Bảng cửu chương

Hình 4.10 - Minh họa bảng cửu chương

Ta thấy bảng cửu chương có dạng bảng. Ta có thể biểu diễn bảng cửu chương bằng kiểu dữ liệu danh sách hai chiều.

Danh sách hai chiều là bảng các phần tử không nhất thiết có cùng kiểu.

Nhận xét rằng mỗi hàng của danh sách hai chiều có cấu trúc như một danh sách một chiều không cần có cùng kích thước. Nếu ta coi mỗi hàng của danh sách hai chiều là một phần tử thì ta có thể nói danh sách hai chiều là danh sách một chiều mà mỗi phần tử là một danh sách một chiều.

Như vậy, ta cũng có thể mô tả dữ liệu của bảng cửu chương là kiểu danh sách một chiều gồm 10 phần tử, mỗi phần tử lại là danh sách một chiều có 9 phần tử, mỗi phần tử là một số nguyên.

Tương tự như với kiểu danh sách một chiều, với kiểu danh sách hai chiều, các ngôn ngữ lập trình cũng có các quy tắc, cách thức cho phép xác định:

- Tên danh sách;

- Số lượng phần tử của mỗi chiều;
- Cách khai báo biến;
- Cách tham chiếu đến phần tử.

Ví dụ, biến danh sách hai chiều B lưu trữ bảng cửu chương có thể được khai báo bằng Python như sau:

```
b = [[0 for j in range(10)] for i in range(9)]
```

a) Khai báo và khởi tạo

Trong Python, việc khai báo danh sách thường đi kèm với việc khởi tạo giá trị ban đầu cho danh sách. Cách thông dụng để khai báo và khởi tạo danh sách hai chiều như sau:

```
<lname> = [[<val>]*ccol for <cntr> in range(<crow>)]
```

Hoặc

```
<lname> = [[<val> for cntc in range(ccol)] for <cntr> in
range(<crow>)]
```

Ý nghĩa:

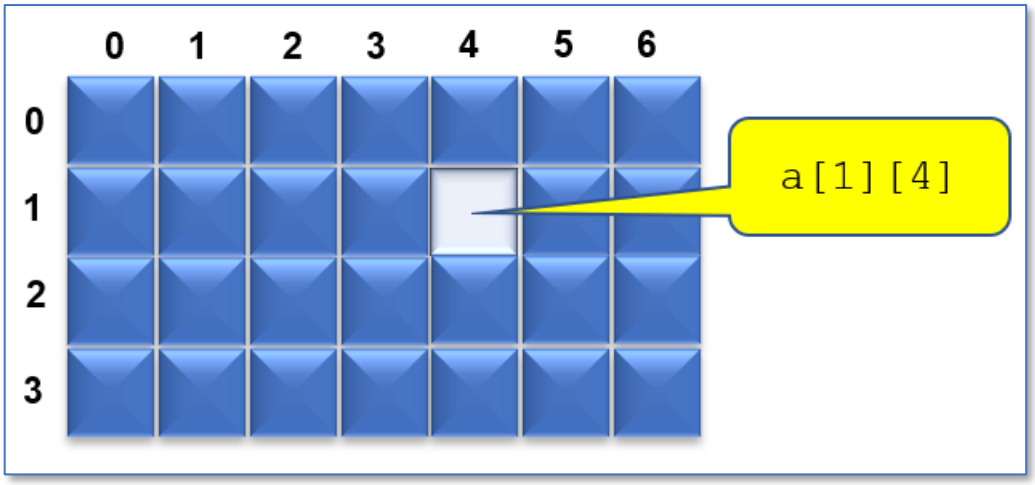
Các cú pháp trên có thể được giải thích như sau, tạo một danh sách 2 chiều với tên là *lname* với kích thước *crow* dòng, *ccol* cột và gán giá trị *val* cho mỗi phần tử trong danh sách. Các dòng, cột đều được đánh số thứ tự từ 0. Các biến *cntr, cntc* là các tên tùy ý chỉ dùng làm biến đếm cho vòng lặp *for*.

Ví dụ. Các khai báo sau đây là hợp lệ

```
a = [[0.0]*200 for i in range(200)]     # bảng số thực
b = [[False]*100 for i in range(100)]   # bảng giá trị logic
```

Việc tham chiếu tới phần tử của danh sách hai chiều được xác định bởi tên danh sách cùng với hai chỉ số, mỗi chỉ số được viết trong một cặp ngoặc [và].

Ví dụ. Tham chiếu tới phần tử ở dòng thứ 5, cột thứ 9 của biến mảng a khai báo trong ví dụ trên là: *a[4][8]* (chú ý dòng thứ 5 có chỉ số là 4, cột thứ 9 có chỉ số là 8).



Hình 4.11 - Minh họa danh sách hai chiều

b) Một số ví dụ

Ví dụ 1. Chương trình sau tính và đưa ra màn hình bảng cửu chương.

```
# khai báo và khởi tại giá trị 0
b = [[0]*10 for i in range(9)]

# gán giá trị
for r in range(9):
    for c in range(10):
        b[r][c] = (r+1)*(c+1)

# in giá trị bảng cửu chương
for r in range(9):
    for c in range(10):
        print("%4d" % b[r][c], end='')
    print('')
```

Thậm chí có thể vừa khai báo, vừa gán giá trị luôn, như sau:

```
# khai báo và gán giá trị
b = [[(i+1)*(j+1) for j in range(10)] for i in range(9)]
# in giá trị bảng cửu chương
for r in range(9):
    for c in range(10):
        print("%4d" % b[r][c], end='')
    print('')
```

Ví dụ 2. Chương trình sau nhập vào từ bàn phím các phần tử của mảng hai chiều *B* gồm 5 dòng, 7 cột với các phần tử là các số nguyên và số nguyên *k*. Sau đó, đưa ra màn hình các phần tử của mảng có giá trị nhỏ hơn *k*.

```
# khai báo và gán giá trị
b = [[0]*7 for i in range(5)]

# Nhập giá trị của bảng từ bàn phím
for r in range(5):
    for c in range(7):
        b[r][c] = int(input())

k = int(input("Nhập vào giá trị k = "))

# đếm phần tử nhỏ hơn k
dem = 0;
for r in range(5):
    for c in range(7):
        if b[r][c] < k:
            print(b[r][c],end=' ')
            dem += 1
if dem == 0:
    print("Không có phần tử nào nhỏ hơn ", k)
```

Bài tập và thực hành 3

1. Mục đích, yêu cầu

- Nâng cao kĩ năng sử dụng một số câu lệnh và một số kiểu dữ liệu thông qua việc tìm hiểu, chạy thử các chương trình có sẵn;
- Biết giải một số bài toán tính toán, tìm kiếm đơn giản trên máy tính.

2. Nội dung

Bài 1.

Tạo mảng *A* gồm *n* ($n \leq 100$) số nguyên, mỗi số có trị tuyệt đối không vượt quá 300. Tính tổng các phần tử của mảng là bội số của một số nguyên dương *k* cho trước.

a) Hãy tìm hiểu và chạy thử chương trình sau đây:

```
import random                    # khai báo sử dụng thư viện tạo số ngẫu nhiên
n = int(input("Nhập n = "))      # nhập n
a = [0]*n                        # khai báo danh sách a có n phần tử
for i in range(n):
    a[i] = random.randint(0,300)-random.randint(0,300)
    # hàm random.randint(s, t) tạo số ngẫu nhiên trong đoạn [s, t]

for i in range(n):
    print("%5d" % a[i], end='')
print('')
k = int(input("Nhập k = "))
s = 0
for i in range(n):
    if a[i] % k == 0:
        s += a[i]
print("Tổng cần tính là: %d" % s)
```

Chú ý:

Hàm *random.randint(l, r)* sinh số nguyên ngẫu nhiên trong khoảng [*l*, *r*].

b) Hãy đưa các câu lệnh sau đây vào những vị trí cần thiết nhằm sửa đổi chương trình trong câu a) để có được chương trình đưa ra số các số dương và số các số âm trong mảng.

```
posi = 0
neg = 0
if a[i]> 0:
    posi += 1
elif a[i]<0:
    neg += 1;
print(posi,neg)
```

Bài 2.

Viết chương trình tìm phần tử có giá trị lớn nhất của mảng và đưa ra màn hình chỉ số và giá trị của phần tử tìm được. Nếu có nhiều phần tử như vậy thì đưa ra phần tử có chỉ số nhỏ nhất.

a) Hãy tìm hiểu chương trình sau đây:

```
n = int(input("Nhập n = "))      # nhập n
a = [0]*n                        # khai báo danh sách a có n phần tử
for i in range(n):
    a[i] = int(input("Nhập phần tử thứ "+str(i)+" = "))

j = 0
for i in range(n):
    if a[i] > a[j]:
        j = i
print("Chỉ số %d giá trị %d" % (j, a[j]))
```

b) Chỉnh sửa chương trình trên để đưa ra chỉ số của các phần tử có cùng giá trị lớn nhất.

Bài tập và thực hành 4

1. Mục đích, yêu cầu

- Biết nhận xét, phân tích đề xuất thuật toán giải bài toán sao cho chương trình chạy nhanh hơn;
- Làm quen với dữ liệu có cấu trúc và bài toán sắp xếp.

2. Nội dung

Bài 1.

a) Hãy tìm hiểu và chạy thử chương trình thực hiện thuật toán sắp xếp dãy số nguyên bằng tráo đổi với các giá trị khác nhau dưới đây. Qua đó, nhận xét về thời gian chạy của chương trình.

```
import random
n = int(input("Nhập n = "))      # nhập n
a = [0]*n                        # khai báo danh sách a có n phần tử
for i in range(n):
    a[i] = random.randint(0,300) - random.randint(0,300)
for i in range(n):
    print("%d " % a[i],end='')
print('')

for j in range(n-1, 0, -1):
    for i in range(j):
        if a[i] > a[i+1]:
            a[i], a[i+1] = a[i+1], a[i] # trao giá trị a[i] và a[i+1]
print("Dãy số được sắp xếp")
for i in range(n):
    print("%d " % a[i],end='')
```

b) Khai báo thêm biến nguyên *Dem* và bổ sung vào chương trình những câu lệnh cần thiết để biến *Dem* tính số lần thực hiện tráo đổi trong thuật toán. Đưa kết quả tìm được ra màn hình.

Bài 2.

a) Hãy đọc và tìm hiểu những phân tích để viết chương trình giải bài toán:

Cho mảng *A* gồm *n* phần tử. Hãy viết chương trình tạo mảng *B[1..n]*, trong đó *B[i]* là tổng của *i* phần tử đầu tiên của *A* (Dãy *B* còn có tên là dãy tổng tiền tố của dãy *A*)

Thoạt đầu có thể viết chương trình sau để giải bài toán này:

```
1. # Tổng tiền tố #
2.
3. import random
4. n = int(input("Nhập n = "))
5. a = [0]*n
6. for i in range(n):
7.     a[i] = random.randint(0,300) - random.randint(0,300)
8. for i in range(n):
9.     print(a[i],end=' ')
10.print('')
11.b = [0]*n
12.# Bắt đầu
13.for i in range(n):
```

```
    for j in range(i+1):
        b[i] += a[j]
# Kết thúc
for i in range(n):
    print(b[i],end=' ')
```

Để ý rằng ta có các hệ thức sau:

```
B[1]= A[1]
B[i]= B[i-1]+A[i], 1 < i ≤ n
```

Phân tích đó cho phép thay đoạn chương trình từ chỗ *#Bat dau* đến *#Ket thuc* bởi hai lệnh sau:

```
b[0] = a[0]
for i in range(1,n):
    b[i] = b[i-1] + a[i]
```

Với hai lệnh này, máy chỉ phải thực hiện *n - 1* phép cộng, trong khi với đoạn chương trình trên máy phải thực hiện $\frac{n(n+1)}{2}$ phép cộng.

Nhờ việc phân tích ta có thể tiết kiệm được một lượng tính toán đáng kể.

Tuy tốc độ tính toán của máy tính nhanh nhưng có giới hạn. Do đó, trong khi viết chương trình, ta nên tìm cách viết sao cho chương trình thực hiện càng ít phép toán càng tốt.

Bài 12. Kiểu xâu ký tự

Dữ liệu trong các bài toán không chỉ thuộc kiểu số mà cả kiểu phi số - dạng kítự. Dữ liệu kiểu xâu là dãy các kí tự.

Ví dụ. Các xâu kí tự đơn giản:

'Bach khoa' 'KI SU' '2007 la nam Dinh Hoi'

Xâu (chuỗi) là dãy các kí tự trong bảng mã ASCII hoặc UNICODE, mỗi kí tự được gọi là một phần tử của xâu. Số lượng kí tự trong một xâu được gọi là độ dài của xâu. Xâu có độ dài bằng 0 gọi là xâu rỗng.

Các ngôn ngữ lập trình đều có quy tắc, cách thức cho phép xác định:

- Kiểu biến xâu;
- Số lượng kí tự của xâu;
- Các phép toán thao tác với xâu;
- Cách tham chiếu tới phần tử xâu.

Biểu thức gồm các toán hạng là biến xâu hoặc hằng xâu được gọi là biểu thức xâu. Với dữ liệu kiểu xâu có thể thực hiện phép toán ghép xâu và các phép toán quan hệ.

Có thể xem xâu là danh sách một chiều mà mỗi phần tử là một kí tự. Trong Python, các kí tự của xâu được đánh số thứ tự bắt đầu từ 0.

Tương tự như với danh sách, tham chiếu tới phần tử của xâu được xác định bởi tên biến xâu và chỉ số đặt trong cặp ngoặc vuông [và].

Ví dụ, giả sử có biến *Hoten = 'Nguyen Le Huyen'* thì *Hoten[5]* cho ta chữ '*n*' là chữ thứ 6 trong xâu *Hoten*.

Trong Python, xâu là một dãy các ký tự Unicode. Bảng mã Unicode được sử dụng để biểu diễn mọi ký tự trong tất cả các ngôn ngữ và mang lại sự đồng nhất về mã hóa trên toàn cầu. Bạn có thể tìm hiểu về Unicode từ Python Unicode (*Vì vậy, khi lập trình bằng Python bạn có thể thoải mái gõ tiếng Việt Unicode* ☺).

Xâu ký tự Python được bao bọc bởi cặp dấu nháy đơn hoặc nháy kép, thậm chí nháy ba. Ví dụ, hai chuỗi "*Hello*" và '*Hello*' là như nhau.

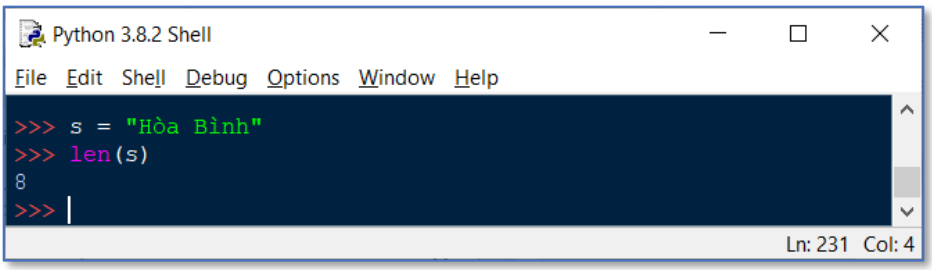
Khác với Pascal/C++ và một số NNLT khác, xâu ký tự trong Python là ***bất biến*** / không thể thay đổi (immutable) giá trị các phần tử trong xâu, nhưng có thể thay đổi giá trị của xâu.

Dưới đây trình bày cách khai báo kiểu dữ liệu xâu, các thao tác xử lí xâu và một số ví dụ sử dụng kiểu xâu trong Python.

1. Khai báo

Giống như các kiểu dữ liệu khác trong Python, kiểu xâu không cần khai báo trước, và cũng không cần xác định độ dài tối đa từ trước. Bất cứ khi nào bạn cần dùng chỉ việc khởi tạo cho biến một giá trị xâu nào nào đó là biến xâu được hình thành. Độ dài của biến xâu là bất kỳ.

Ví dụ: Lệnh dưới đây có dùng hàm *len(s)* cho biết độ dài xâu *s*



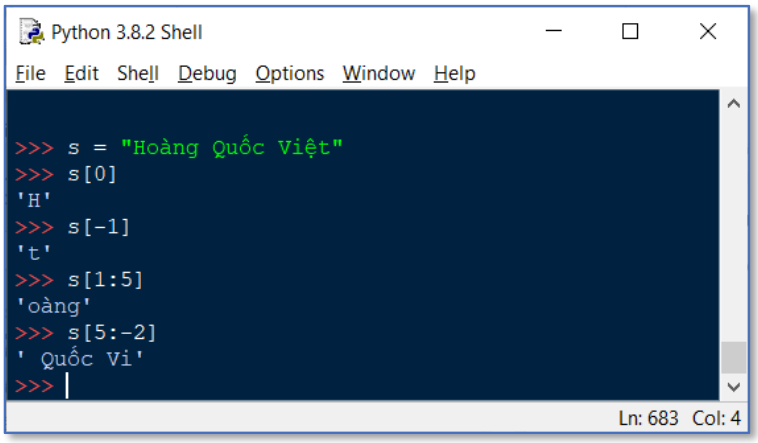
Hình 4.10 - Minh họa việc khai báo và khởi tạo xâu

2. Các thao tác cơ bản trên xâu

a) Truy cập đến phần tử trong xâu, toán tử cắt trích []

Chúng ta có thể truy cập các phần tử riêng lẻ bằng cách sử dụng chỉ số và truy cập đến một xâu con bằng cách sử dụng toán tử cắt trích. Chỉ số trong xâu bắt đầu từ 0. Việc cố gắng truy cập một phần tử ngoài phạm vi chỉ số sẽ phát sinh một lỗi *IndexError*. Các chỉ số phải là số nguyên. Chúng ta không thể sử dụng kiểu *float* hoặc các kiểu khác, việc đó sẽ dẫn đến lỗi *TypeError*.

Python cho phép sử dụng chỉ số âm trong xâu. Chỉ số -1 trỏ đến ký tự cuối cùng của xâu, -2 trỏ đến ký tự liền trước ký tự cuối cùng, v.v. Chúng ta có thể truy cập một xâu con trong một xâu bằng cách sử dụng toán tử cắt trích []. Ví dụ, chạy đoạn mã trong hình trái dưới, ta nhận được kết quả như hình bên phải dưới:



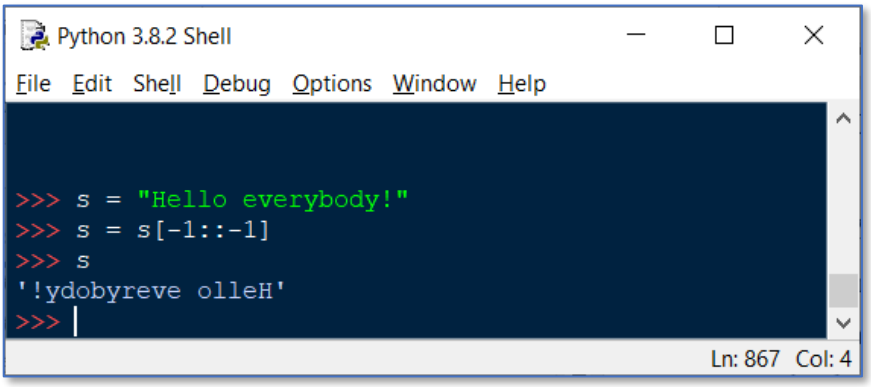
Hình 4.11 - Minh họa cách truy xuất phần tử trong xâu

Toán tử cắt trích [] còn cho phép ta cắt trích một đoạn (phạm vi) trong xâu thao cú pháp.

```
[start : end : step]
```

Ý nghĩa là, lần lượt duyệt và trích các phần tử thuộc phạm vi từ vị trí *start* đến vị trí *end-1* sau mỗi lần lấy một phần tử trong phạm vi thì di chuyển con trỏ với bước nhảy có độ rộng *step*. Các tham số trên có thể vắng mặt: nếu thiếu *start* thì mặc định là bắt đầu từ đầu xâu, nếu thiếu *end* thì mặc định đến hết xâu, nếu thiếu *step* thì mặc định là 1.

Dưới đây là một ví dụ về cách tạo xâu ngược bằng toán tử cắt trích.

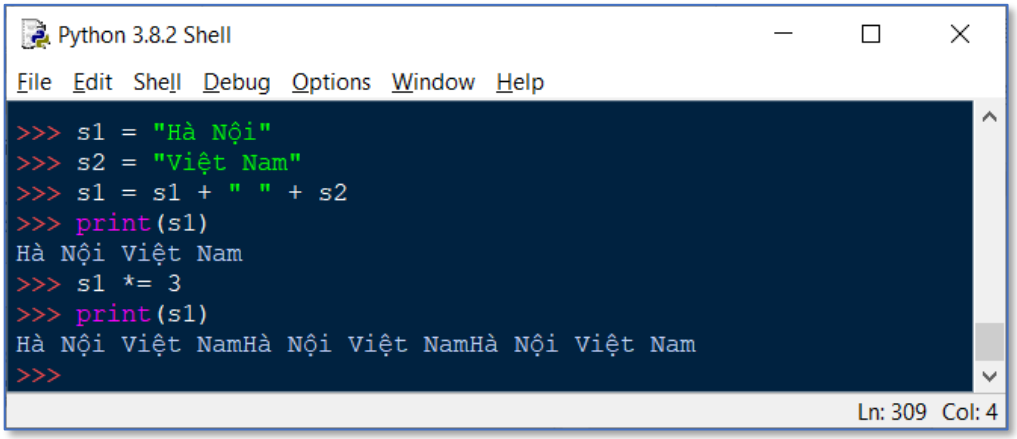


Hình 4.12 - Đảo ngược xâu

b) Ghép xâu

Nối hai hoặc nhiều xâu thành một xâu được gọi là ghép xâu. Toán tử + thực hiện điều này trong Python (cũng như Pascal và C++). Đơn giản chỉ cần viết phép toán + giữa hai xâu là có thể nối chúng. Ngoài ra, trong Python còn hỗ trợ toán tử lặp xâu * được sử dụng để lặp lại xâu với một số lần nhất định.

Ví dụ:



Hình 4.13 - Minh họa phép ghép và phép lặp xâu

Lệnh *s1 = s1 + " " + s2* đã ghép ba xâu *s1*, *" "*, *s2* thành một xâu và đặt trong *s1*. Còn dòng lệnh s1*= 3 đã lặp lại việc ghép xâu *s1* ba lần và đặt lại vào trong *s1*.

c) Các phép so sánh xâu

Các phép so sánh bằng (==), khác (!=), nhỏ hơn (<), lớn hơn (>), nhỏ hơn hoặc bằng (<=), lớn hơn hoặc bằng (>=) có thứ tự ưu tiên thực hiện thấp hơn phép ghép xâu và thực hiện việc so sánh hai xâu theo các quy tắc sau:

- Xâu A là lớn hơn xâu B nếu như kí tự đầu tiên khác nhau giữa chúng kể từ trái sang trong xâu A có mã UNICODE lớn hơn.
- Nếu A và B là các xâu có độ dài khác nhau và A là đoạn đầu của B thì A là nhỏ hơn B.

  Ví dụ

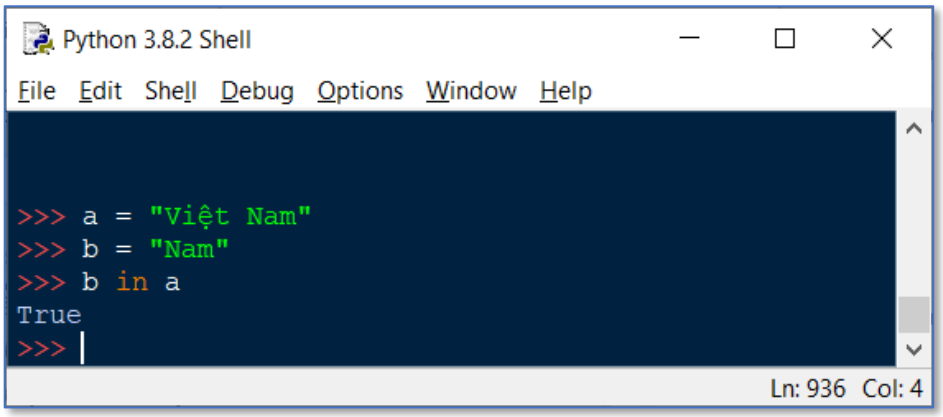
  'May tinh' < 'May tinh cua toi'

- Hai xâu được coi là bằng nhau nếu như chúng giống nhau hoàn toàn.

  Ví dụ

  'TIN HOC' == 'TIN HOC'

Ngoài ra, để kiểm tra một xâu *a* có là xâu con của xâu *b* hay không ta dùng phép toán logic ***in*** với cú pháp: *a* ***in*** *b*, nếu kết quả phép toán là *True* thì *a* là xâu con của *b*, ngược lại thì kết quả là *False*.

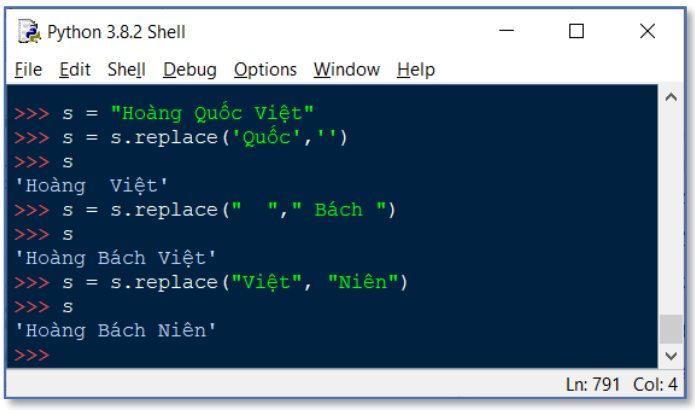


Hình 4.14 - Minh họa phép toán ***in*** *trên xâu*

c) Biến đổi, chèn hoặc xóa xâu con trong một xâu

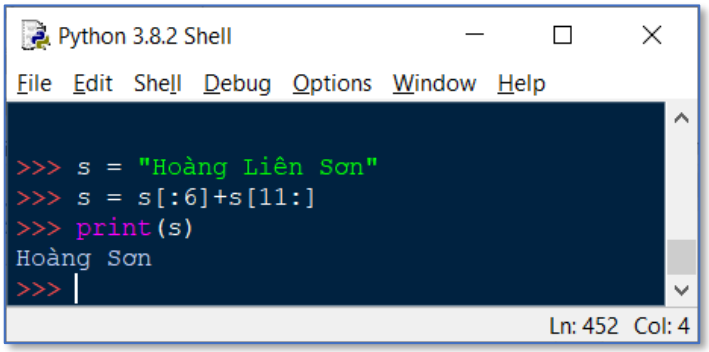
Như đã nói ở trên, các phần tử trong xâu Python là bất biến, không thể sửa hay xóa chúng như trong các ngôn ngữ Pascal/C++, nhưng ta có thể sử dụng một số phương thức thay thế.

***Cách 1*:** Dùng hàm thành viên *replace*(*olds*, *news*) để thay thế phần xâu con *olds* bằng một xâu mới *news* rỗng. Ví dụ:



Hình 4.15 - Minh họa dùng hàm replace() chèn, xóa hoặc thay thế một xâu con trong xâu hiện tại

***Cách 2*:** Dùng phép toán cắt trích *[:]* để tạo ra xâu mới thay thế và bỏ đi phần xâu con muốn xóa. Ví dụ, để xóa đi chữ "*Bình*" trong xâu *s = "Hòa Bình"*, ta làm như sau:

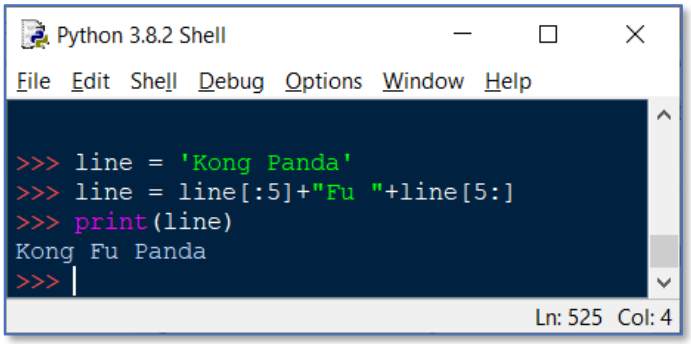


Hình 4.16 - Dùng toán tử cắt trích để xóa xâu con

Trong hình 4.13, ta dùng phép toán cắt trích để thay xâu *s="Hoàng Liên Sơn"* bằng xâu mới "*Hoàng Sơn*" là một phần của xâu *s* cũ. Việc này cũng tương đương với việc xóa một xâu con trong xâu đã cho. Cụ thể, lấy ra và ghép hai xâu con của xâu cũ, xâu con thứ nhất gồm các ký tự từ vị trí 0 đến vị trí 5 của xâu cũ, xâu con thứ hai gồm các ký tự thứ 11 đến cuối trong xâu cũ.

d) Chèn xâu

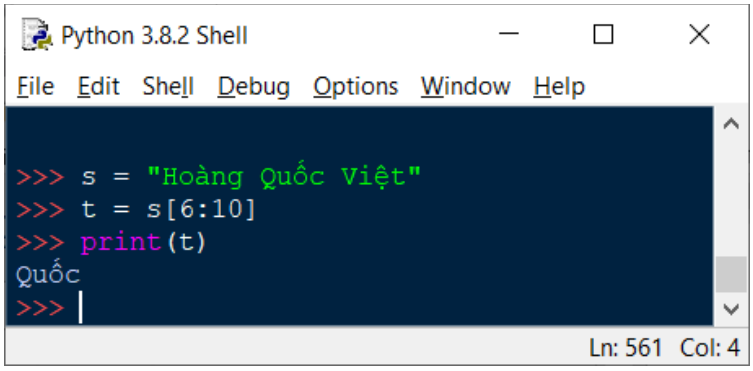
Tương tự như việc xóa và thay thế xâu, việc chèn xâu trực tiếp là không thể thực hiện trong Python mà chỉ có thể thay thế bằng cách dùng phép toán cắt trích. Ví dụ:



Hình 4.17 - Sử dụng phép toán cắt trích để thực hiện chèn xâu

e) Sao chép xâu

Để sao chép một xâu con trong một xâu cho trước ta cũng sử dụng toán tử cắt trích để lấy ra xâu con mong muốn. Ví dụ:



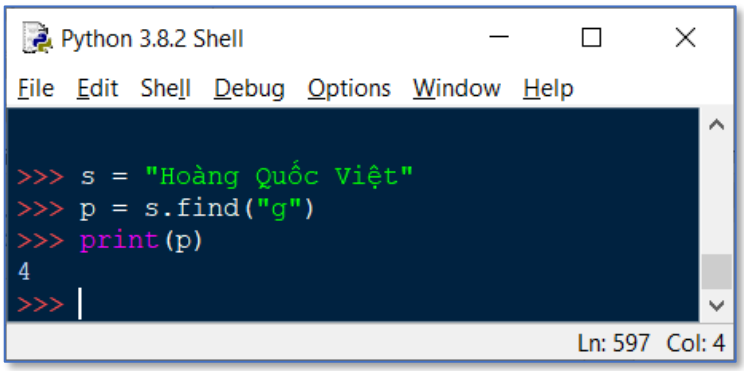
Hình 4.18 - Sử dụng phép toán cắt trích để sao chép xâu con

f) Tính độ dài xâu

Để tìm độ dài xâu ta có thể dùng hàm *len()*, xem ví dụ hình 4.10.

g) Tìm kiếm vị trí xâu con trong một xâu

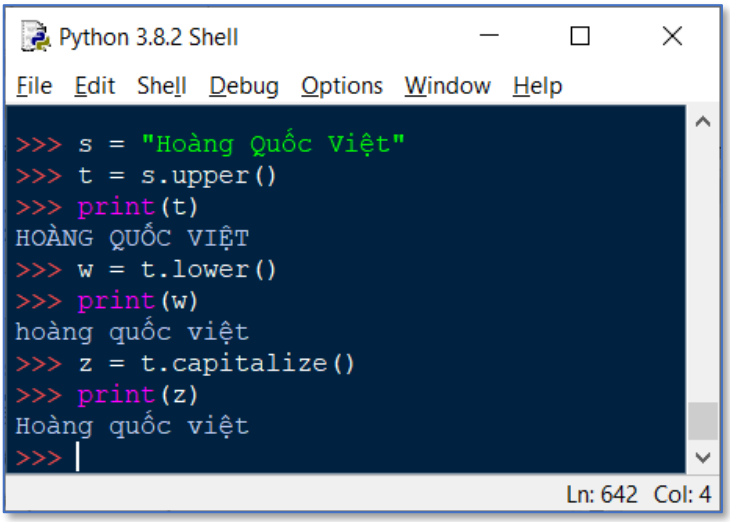
Để tìm kiếm vị trí xuất hiện lần đầu tiên của một xâu con trong một xâu cho trước, ta sử dụng hàm thành viên *find(sub, [start], [end])*, với ý nghĩa tìm xâu con *sub* trong xâu hiện tại, vị trí bắt đầu tìm kiếm là *start* và vị trí kết thúc tìm kiếm là *end*, nếu vắng *start* thì mặc định bắt đầu tìm là vị trí 0 (cận trái) và nếu vắng *end* thì mặc định vị trí kết thúc tìm kiếm là cuối xâu. Nếu việc tìm kiếm không có kết quả thì hàm trả về giá trị -1. Ví dụ:



Hình 4.19 - Tìm kiếm vị trí xâu con

h) Một số hàm định dạng xâu

- Hàm *upper(), lower()* tạo xâu in hoa/in thường toàn bộ từ xâu hiện tại. Hàm *capitalize()* viết hoa chữ đầu tiên trong xâu, các chữ còn lại viết thường.



Hình 4.20 - Minh họa cách dùng hàm upper(), lower(), capitalize()

Ngoài các chức năng trên, Python còn rất nhiều tiện ích trên kiểu xâu, các bạn có thể tự tra cứu trên Internet (ví dụ trang https://www.programiz.com/python-programming/methods/string)

3) Một số ví dụ

(các ví dụ này được viết dựa theo sách Tin 11, các bạn có thể tìm thấy nhiều ví dụ hấp dẫn hơn mang dáng dấp của Python trên Internet ☺)

Ví dụ 1

Chương trình dưới đây nhập họ tên của hai người vào hai biến xâu và đưa ra màn hình xâu dài hơn, nếu bằng nhau thì đưa ra xâu nhập sau.

```
a = input("Nhập họ tên người thứ nhất: ")
b = input("Nhập họ tên người thứ hai: ")
if len(a) > len(b):
    print(a)
else:
    print(b)
```

Ví dụ 2

Chương trình dưới đây nhập hai xâu từ bàn phím và kiểm tra kí tự đầu tiên của xâu thứ nhất có trùng với kí tự cuối cùng của xâu thứ hai không.

```
a = input("Nhập xâu thứ nhất: ")
b = input("Nhập xâu thứ hai: ")
if a[0] == b[-1]:
    print("Trùng nhau")
else:
    print("Khác nhau")
```

Ví dụ 3

Chương trình sau nhập một xâu vào từ bàn phím và đưa ra màn hình xâu đó nhưng được viết theo thứ tự ngược lại.

```
a = input("Nhập xâu: ")
n = len(a)
for i in range(n-1, -1, -1):
    print(a[i],end='')
```

Ví dụ 4

Chương trình sau nhập một xâu vào từ bàn phím và đưa ra màn hình xâu thu được từ nó bởi việc loại bỏ các dấu cách.

Cách 1: Tương tự cách trong NNLT Pascal trong sách Tin 11

```
a = input("Nhập xâu: ")
n = len(a)
b = ""      # khởi tạo xâu rỗng b
for i in range(n):
    if a[i]!=' ':
        b +=a[i]
print("Kết quả: ", b)
```

Cách 2: Dùng hàm *replace()* thay thế các xâu dấu cách bằng các xâu rỗng

```
a = input("Nhập xâu: ")
n = len(a)
b = a.replace(" ",'')
print("Kết quả: ", b)
```

Ví dụ 5

Chương trình sau nhập vào từ bàn phím xâu kí tự *s*1, tạo xâu *s*2 gồm tất cả các chữ số có trong *s*1 (giữ nguyên thứ tự xuất hiện của chúng) và đưa kết quả ra màn hình.

Cách 1: Tương tự cách trong NNLT Pascal trong sách Tin 11

```
a = input("Nhập xâu: ")
n = len(a)
b = ""      # khởi tạo xâu rỗng b
for i in range(n):
    if a[i] >='0' and a[i]<='9':
        b +=a[i]
print("Kết quả: ", b)
```

Cách 2: Dùng hàm *isdigit()* để kiểm tra các phần tử của xâu là chữ số

```
a = input("Nhập xâu: ")
n = len(a)
b = ""      # khởi tạo xâu rỗng b
for i in range(n):
    if a[i].isdigit():
        b +=a[i]
print("Kết quả: ", b)
```

Bài tập và thực hành 5

1. Mục đích, yêu cầu

Làm quen với việc tìm kiếm, thay thế và biến đổi xâu.

2. Nội dung

Bài 1.

Nhập vào từ bàn phím một xâu. Kiểm tra xâu đó có phải là xâu đối xứng, tức là đọc nó từ phải sang trái cũng như từ trái sang phải hay không (các xâu có tính chất này được gọi là palindrome).

a) Hãy chạy thử chương trình sau:

```
a = input("Nhập xâu: ")
n = len(a)
b = ""      # khởi tạo xâu rỗng b
for i in range(n-1, -1, -1): # tạo xâu b ngược với xâu a
    b +=a[i]
if a == b:
    print("Xâu là palindrome")
else:
    print("Xâu không là palindrome")
```

b) Hãy viết lại chương trình trên trong đó không cần có biến xâu *p*.

```
a = input("Nhập xâu: ")
n = len(a)
ok = True
for i in range(n//2):
    if a[i] != a[n-i-1]:
        ok = False
        break
if ok == True:
    print("Xâu là palindrome")
else:
    print("Xâu không là palindrome")
```

Bài 2.

Viết chương trình nhập từ bàn phím một xâu kí tự *S* và thông báo ra màn hình số lần xuất hiện của mỗi chữ cái tiếng Anh trong *S* (không phân biệt chữ hoa hay chữ thường).

```
a = input("Nhập xâu: ")
n = len(a)
dem = [0]*26
a = a.upper() # thay hết các chữ in thường thành in hoa
for i in range(n):
    dem[ord(a[i])-ord('A')] += 1
for i in range(26):
    if dem[i] > 0 :
        print("Chữ",chr(ord('A')+i),"xuất hiện", dem[i],"lần")
```

Chú ý: Python cũng hỗ trợ các hàm *ord()* và *chr()* với ý nghĩa tương tự như NNLT Pascal.

TÓM TẮT

- Kiểu dữ liệu có cấu trúc được xây dựng từ những kiểu dữ liệu đã có theo quy tắc, khuôn dạng do ngôn ngữ lập trình cung cấp.
- Danh sách
  - Danh sách là dãy hữu hạn các phần tử không nhất thiết có cùng kiểu.
  - Khai báo và khởi tạo danh sách.
  - Tham chiếu phần tử danh sách: *tên danh sách [chỉ số phần tử]*
- Kiểu dữ liệu xâu
  - Xâu là dãy các kí tự trong bảng mã UNICODE
  - Các thao tác xử lí thường sử dụng:
    - Phép ghép xâu;
    - Phép so sánh;
    - Các thao tác khác trên xâu.

Câu hỏi và bài tập

1. Tại sao danh sách là kiểu dữ liệu có cấu trúc?

2. Có cần thiết phải khai báo kích thước của danh sách trước khi dùng không?

3. Các phần tử của danh sách có thể có những kiểu gì?

4. Tham chiếu đến phần tử của danh sách bằng cách nào?

5. Viết chương trình nhập từ bàn phím số nguyên dương N ($N \leq 100$) và dãy A gồm N số nguyên $A_1, A_2,..., A_N$, có giá trị tuyệt đối không lớn hơn 1000. Hãy cho biết dãy A có phải là một cấp số cộng hay không và thông báo kết quả ra mànhình.

6. Viết chương trình nhập từ bàn phím số nguyên dương N ($N \leq 100$) và dãy A gồm N số nguyên $A_1, A_2,..., A_N$, có trị tuyệt đối không lớn hơn 1000. Hãy đưa ra những thông tin sau:

a) Số lượng số chẵn và số lẻ trong dãy;

b) Số lượng số nguyên tố trong dãy;

7. Dãy F là dãy Fi-bô-na-xi nếu: $F0 = 1; F1 = 1; F2 = 2; FN = FN-1 + FN-2$ với $N > 2$.

Viết chương trình nhập từ bàn phím số nguyên dương N và đưa ra màn hình số hạng thứ N của dãy Fi-bô-na-xi. Chương trình của em thực hiện được với giá trị lớn nhất của N là bao nhiêu?

8. Chương trình sau đây thực hiện những gì?

```
NMax = 50
a = [[0.0]*NMax for i in range(NMax-1)]

n = int(input("Nhập N = "))
for i in range(1, n+1):
    for j in range(n):
        a[i][j] = float(input("A["+str(i)+","+str(j)+"]= "))

for i in range(1, n+1):
    for j in range(n):
        a[i][j], a[n-i+1][j] = a[n-i+1][j], a[i][j]

for i in range(1, n+1):
    for j in range(n):
        print("%10.2f" % a[i][j],end='')
    print('')
```

9. Cho mảng hai chiều kích thước $n \times n$ với các phần tử là những số nguyên. Tìm trong mỗi dòng phần tử lớn nhất rồi đổi chỗ nó với phần tử có chỉ số dòng bằng chỉ số cột.

Chương trình sau đây giải bài toán trên:

```
a = [[0]*15 for i in range(15)]

n = int(input("Nhập N = "))
for i in range(n):
    for j in range(n):
        a[i][j] = int(input("A["+str(i)+","+str(j)+"]= "))

for i in range(1, n+1):
    Max = a[i][0]
    ind = 0
    for j in range(1,n):
        if a[i][j] > Max:
            Max = a[i][j]
            ind = j
    a[i][i], a[i][ind] = a[i][ind], a[i][i]

for i in range(n):
    for j in range(n):
        print("%10d" % a[i][j],end='')
    print('')
```

10. Viết chương trình nhập từ bàn phím xâu kí tự *S* có độ dài không quá 100. Hãy cho biết có bao nhiêu chữ số thập phân xuất hiện trong xâu *S*. Thông báo kết quả ra màn hình.

Chương 5. Tệp và thao tác với tệp

Bài 14. Kiểu dữ liệu tệp

1. Vai trò kiểu tệp

Tất cả các dữ liệu có các kiểu dữ liệu đã xét đều được lưu trữ ở bộ nhớ trong (RAM) và do đó dữ liệu sẽ bị mất khi tắt máy. Với một số bài toán, dữ liệu cần được lưu trữ để xử lí nhiều lần và với khối lượng lớn cần có kiểu dữ liệu tệp (file).

Kiểu dữ liệu tệp có những đặc điểm sau:

- Được lưu trữ lâu dài ở bộ nhớ ngoài (đĩa từ, CD,…) và không bị mất khi tắt nguồnđiện;
- Lượng thông tin lưu trữ trên tệp có thể rất lớn và chỉ phụ thuộc vào dung lượngđĩa.

2. Phân loại tệp và thao tác với tệp

Xét theo cách tổ chức dữ liệu, có thể phân tệp thành hai loại:

- *Tệp văn bản* là tệp mà dữ liệu được ghi dưới dạng các kí tự theo mã ASCII/UNICODE. Trong tệp văn bản, dãy kí tự kết thúc bởi nhóm kí tự xuống dòng hay kí tự kết thúc tệp.

Các dữ liệu dạng văn bản như sách, tài liệu, bài học, giáo án, các chương trình nguồn viết bằng ngôn ngữ bậc cao,... thường được lưu trữ dưới dạng tệp văn bản.

- *Tệp có cấu trúc* là tệp chứa dữ liệu được tổ chức theo một cách thức nhất định. Dữ liệu ảnh, âm thanh,... thường được lưu trữ dưới dạng tệp có cấu trúc.

Xét theo cách thức truy cập, có thể phân tệp thành hai loại:

- *Tệp truy cập tuần tự* cho phép truy cập đến một dữ liệu nào đó trong tệp chỉ bằng cách bắt đầu từ đầu tệp và đi qua lần lượt tất cả các dữ liệu trước nó.
- *Tệp truy cập trực tiếp* cho phép tham chiếu đến dữ liệu cần truy cập bằng cách xác định trực tiếp vị trí (thường là số hiệu) của dữ liệu đó.

Khác với mảng, số lượng phần tử của tệp không xác định trước.

Hai thao tác cơ bản đối với tệp là ghi dữ liệu vào tệp và đọc dữ liệu từ tệp.

Thao tác đọc/ghi với tệp được thực hiện với từng phần tử của tệp.

Để thao tác với kiểu dữ liệu tệp, người lập trình cần tìm hiểu cách thức NNLT cung cấp cách:

- Khai báo biến tệp;
- Mở tệp;
- Đọc/ghi dữ liệu;
- Đóng tệp.

Bài 15. Kiểu tệp

Trong mục này ta chỉ xét cách khai báo, thao tác với tệp văn bản trong Python.

1. Khai báo

Cũng như các biến kiểu khác, Python không yêu cầu khai báo biến tệp từ trước, chỉ khi nào bạn cần mở tệp thì mới khai báo biến tệp và đồng thời sẽ thực hiện thao tác mở tệp để thực hiện các tác vụ đọc/ ghi/ bổ sung dữ liệu gắn với biến tệp đó.

2. Thao tác với tệp

Trong Python, hoạt động với tệp diễn ra theo thứ tự sau:

- Mở tệp
- Đọc hoặc ghi dữ liệu
- Đóng tệp

Python có một hàm tích hợp để mở một tệp là *open()*. Hàm này trả về một biến tệp, và nó được sử dụng để đọc hoặc sửa đổi tệp ở từng chế độ mở phù hợp.

a) Chế độ mở tệp để đọc dữ liệu

Cú pháp:

```
<tên biến tệp> = open(<tên tệp>, ["r"], [endcoding = "xxx"])
```

Ý nghĩa:

Sau lệnh trên, mọi thao tác với tệp đều gắn với *tên biến tệp*, tham số "*r*" chỉ báo mở tệp để đọc (tham số này là mặc định và có thể vắng mặt), tham số *endcoding* cũng là tùy chọn, nếu vắng nó tệp được mở theo bảng mã mặc định ASCII. Nếu muốn mở tệp văn bản tiếng Việt Unicode thông dụng hiện nay, thì tham số này viết là *endcoding = "utf-8"*.

Ví dụ.

Lệnh sau đây sẽ gắn tệp *paths.cpp* chứa trong thư mục *C++* của ổ đĩa *F:* với biến tệp *f* và tệp được mở ở chế độ đọc dữ liệu.

```
f = open("f:\c++\paths.cpp", "r")
```

Chú ý:

Trong chế độ mở tệp để đọc, nếu tệp được chỉ định không tồn tại thì bạn sẽ nhận được thông báo lỗi: *FileNotFoundError: [Errno 2] No such file or directory: '.....'*.

b) Đọc dữ liệu từ tệp đang mở ở chế độ đọc

Có 3 cách đọc dữ liệu từ một tệp văn bản:

- Cách 1: đọc một lần toàn bộ văn bản trong tệp bằng lệnh *read()*. Cú pháp:

```
<biến xâu> = <biến tệp>.read()
```

- Cách 2: đọc mỗi lần một dòng từ tệp bằng lệnh *readline()*. Cú pháp:

```
<biến xâu> = <biến tệp>.readline()
```

- Cách 3: đọc một lần được một danh sách, mỗi phần tử của danh sách là một dòng trong tệp bằng lệnh *readlines()*. Cú pháp:

```
<biến danh sách xâu> = <biến tệp>.readlines()
```

c) Mở tệp để ghi dữ liệu

Cú pháp:

```
<tên biến tệp> = open(<tên tệp>,"w",[endcoding = "xxx"] )
```

Ý nghĩa:

Việc mở tệp để ghi dữ liệu chỉ khác với việc mở để đọc ở tham số thứ 2 trong hàm *open()*, đó là chữ "*w*" (write). Nếu tệp này chưa tồn tại thì nó sẽ được tạo mới trên đĩa, còn nếu tệp đã tồn tại thì nó bị ghi đè.

d) Ghi dữ liệu vào tệp đang mở ở chế độ ghi

Python hỗ trợ 2 cách ghi dữ liệu vào một tệp văn bản:

- Cách 1: Ghi một xâu văn bản vào tệp bằng lệnh *write()*. Cú pháp:

```
<biến tệp>.write(<biểu thức xâu>)
```

- Cách 3: Ghi một lần cả một danh sách các xâu văn bản vào tệp bằng lệnh *writelines()*. Cú pháp:

```
<biến tệp>.writelines(<danh sách xâu>)
```

e) Đóng tệp đang mở

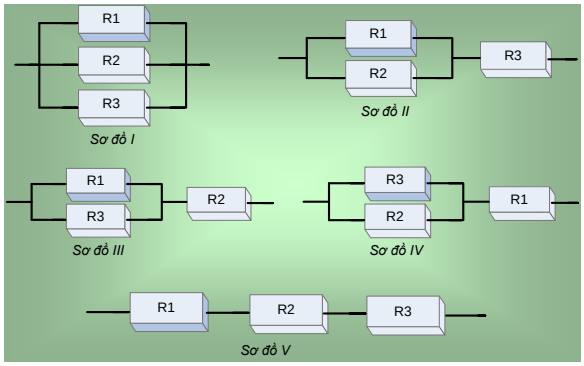
Sau khi kết thúc phiên làm việc với tệp thì ta cần đóng tệp và giải phóng tài nguyên. Giả sử tệp được mở đang gắn với biến tệp *f* thì ta chỉ cần sử dụng lệnh *f.close()* để đóng tệp.

Bài 16. Ví dụ làm việc với tệp

Ví dụ 1

Tính điện trở tương đương.

Cho ba điện trở *R1, R2, R3*. Sử dụng cả ba điện trở ta có thể tạo ra năm điện trở tương đương bằng cách mắc các sơ đồ nêu ở hình 5.1.



Hình 5.1 - Sơ đồ mắc điện trở

Mỗi cách mắc sẽ cho một điện trở tương đương khác nhau. Ví dụ, nếu mắc theo sơ đồ I thì điện trở tương đương sẽ là:

$$R = \frac{R_1 * R_2 * R_3}{R_1 * R_2 + R_1 * R_3 + R_2 * R_3}$$

Còn nếu mắc theo sơ đồ V thì $R = R_1 + R_2 + R_3$.

Cho tệp văn bản RESIST.DAT gồm nhiều dòng, mỗi dòng chứa ba số thực R1, R2 và R3, các số cách nhau một dấu cách, $0 < R1, R2, R3 \leq 10^5$.

Chương trình sau đọc dữ liệu từ tệp RESIST.DAT, tính các điện trở tương đương và ghi kết quả ra tệp văn bản RESIST.EQU, mỗi dòng ghi năm điện trở tương đương của ba điện trở ở dòng dữ liệu vào tương ứng.

```
# Mở tệp để đọc dữ liệu
f1 = open("resist.dat")
# Mở tệp để ghi dữ liệu
f2 = open("resist.equ", "w")
# Khai báo danh sách có 5 phần tử
a = [0]*5
# Lặp vô hạn, dừng bằng lệnh break
while True:
    # Đọc một dòng từ tệp vào biến xâu s, cắt bỏ các dấu cách thừa
    s = f1.readline().strip()
    # Nếu s rỗng thì dừng vòng lặp while
    if not s :
        break
    # tách xâu s và đổi thành 3 số thực và gán cho r1, r2, r3
    r1, r2, r3 = map(float, s.split())
    a[0] = r1*r2*r3/(r1*r2 + r1*r3 + r2*r3)
    a[1] = r1*r2/(r1+r2) + r3
    a[2] = r1*r3/(r1+r3) + r2
```

```
    a[3] = r3*r2/(r3+r2) + r1
    a[4] = r1 + r2 + r3
    # ghi danh sách a[i] ra tệp với định dạng số thực
    for i in range(5):
        f2.write("%9.3f" % a[i])
    # xuống dòng trong tệp
    f2.write("\n")
# đóng các tệp
f1.close()
f2.close()
```

Ví dụ 2.

Thầy hiệu trưởng tổ chức cho giáo viên và học sinh của trường đi cắm trại, sinh hoạt ngoài trời ở vườn quốc gia Cúc Phương. Để lên lịch đến thăm khu trại các lớp, thầy Hiệu trưởng cần biết khoảng cách từ trại của mình (ở vị trí có toạ độ (0;0)) đến trại các giáo viên chủ nhiệm lớp. Mỗi lớp có một khu trại, vị trí trại của mỗi thầy chủ nhiệm đều có tọa độ nguyên (x ; y), được ghi vào tệp văn bản TRAI.TXT (như vậy, tệp TRAI.TXT chứa các cặp số nguyên liên tiếp, các số cách nhau bởi dấu cách và không kết thúc bằng dấu xuống dòng).

Chương trình sau sẽ đọc từ tệp TRAI.TXT các cặp toạ độ, tính và đưa ra màn hình khoảng cách (với độ chính xác hai chữ số sau dấu chấm thập phân) từ trại mỗi giáo viên chủ nhiệm lớp đến trại của thầy hiệu trưởng.

```
import math

f = open('trai.txt')
while True:
    # Đọc từng dòng dữ liệu trong tệp vào chuỗi s
    # dùng hàm strip() cắt bỏ các dấu cách thừa hai đầu xâu
    s = f.readline().strip()
    # nếu s rỗng (hết tệp) thì dừng lặp
    if not s :
        break
    # tách s thành các chuỗi con phân cách bằng dấu cách
    # chuyển các chuỗi con thành một danh sách số nguyên
    l = list(map(int, s.split()))
    n = len(l)
    # duyệt danh sách l lấy ra các cặp x, y để tính khoảng cách
    for i in range(0, n, 2):
        x = l[i]
        y = l[i+1]
        d = math.sqrt(x**2 + y**2)
        print("%10.2f" % d, end ='')
f.close()
```

TÓM TẮT

- Việc trao đổi dữ liệu với bộ nhớ ngoài được thực hiện thông qua kiểu dữ liệu tệp;
- Để có thể làm việc với tệp cần phải gắn tệp với biến tệp;
- Mỗi ngôn ngữ lập trình đều có các chương trình chuẩn để làm việc với tệp;
- Các thao tác với tệp văn bản:
  - Cách khai báo biến tệp, mở tệp và đóng tệp.
  - Đọc/ghi: tương tự như làm việc với bàn phím và màn hình.

Câu hỏi và bài tập

1. Nêu một số trường hợp cần phải dùng tệp.
2. Khi cần nhập dữ liệu từ tệp phải dùng những thao tác nào?
3. Tại sao cần phải mở tệp trước khi đọc/ghi tệp?
4. Tại sao phải dùng câu lệnh đóng tệp sau khi đã kết thúc ghi dữ liệu vào tệp?

Chương 6. Hàm và lập trình có cấu trúc

Bài 17. Chương trình con và phân loại

1. Khái niệm chương trình con

Các chương trình giải các bài toán phức tạp thường rất dài, có thể gồm hàng trăm, hàng nghìn lệnh. Khi đọc những chương trình dài, rất khó nhận biết được chương trình thực hiện các công việc gì và việc hiệu chỉnh chương trình cũng khó khăn. Vì vậy, vấn đề đặt ra là phải cấu tạo chương trình như thế nào để cho chương trình dễ đọc, dễ hiệu chỉnh, nâng cấp.

Mặt khác, việc giải quyết một bài toán phức tạp thường đòi hỏi và nói chung có thể phân thành các bài toán con.

Xét bài toán tính tổng bốn luỹ thừa:

$TLuythua = a^n + b^m + c^p + d^q$

Bài toán đó bao gồm bốn bài toán con tính a^n, b^m, c^p, d^q, có thể giao cho bốn người, mỗi người thực hiện một bài. Giá trị *TLuythua* là tổng kết quả của bốn bài toán con đó. Với những bài toán phức tạp hơn, mỗi bài toán con lại có thể được phân chia thành các bài toán con nhỏ hơn. Quá trình phân rã làm "mịn" dần bài toán như vậy được gọi là cách thiết kế từ trên xuống.

Tương tự, khi lập trình để giải bài toán trên máy tính có thể phân chia chương trình (gọi là chương trình chính) thành các khối (môđun), mỗi khối bao gồm các lệnh giải một bài toán con nào đó. Mỗi khối lệnh sẽ được xây dựng thành *một chương trình con*. Sau đó, chương trình chính sẽ được xây dựng từ các chương trình con này. Có thể xem chương trình con cũng là một chương trình và nó cũng có thể được xây dựng từ các chương trình con khác.

Cách lập trình như vậy dựa trên phương pháp lập trình có cấu trúc và chương trình được xây dựng gọi là chương trình có cấu trúc.

Chương trình con là một dãy lệnh mô tả một số thao tác nhất định và có thể được thực hiện (được gọi) từ nhiều vị trí trong chương trình.

Ví dụ, Mỗi môn Văn, Toán, Anh có một số con điểm. Bạn cần viết chương trình nhập vào các con điểm, tính và in ra điểm trung bình của mỗi môn. Điểm của mỗi môn được nhập trên một dòng và cách nhau bởi dấu cách. Thứ tự, dòng thứ nhất là điểm Văn, dòng thứ hai là điểm Toán, dòng thứ ba là điểm Anh. Chương trình viết bằng Python có thể như sau:

```
1. van = list(map(float, input('Nhập các điểm văn: ').split()))
2. m = len(van)
3. toan = list(map(float, input('Nhập các điểm toán: ').split()))
4. n = len(toan)
5.
6. anh = list(map(float, input('Nhập các điểm anh: ').split()))
```

```
7. k = len(anh)
8.
9. tvan = 0.0
10.for i in range(m):
11.    tvan += van[i]
12.
13.ttoan = 0.0
14.for i in range(n):
15.    ttoan += toan[i]
16.
17.tanh = 0.0
18.for i in range(k):
19.    tanh += anh[i]
20.
21.print("%6.1f%6.1f%6.1f" % (tvan/m, ttoan/n, tanh/k))
```

Trong chương trình trên có ba đoạn lệnh tương tự nhau (dòng 9-11, dòng 13-15, dòng 17-19) việc lặp lại những đoạn lệnh tương tự nhau làm cho chương trình vừa dài vừa khó theo dõi. Để nâng cao hiệu quả lập trình, các ngôn ngữ lập trình bậc cao đều cung cấp khả năng xây dựng chương trình con dạng tổng quát "đại diện" cho nhiều đoạn lệnh tương tự nhau, chẳng hạn để tính điểm trung bình của một môn học nào đó ta viết một hàm *Average(l)* và gửi cho hàm danh sách điểm cần tính *l*, hàm sẽ tính và trả về giá trị trung bình cho ta. Cụ thể chương trình trên có thể viết lại như sau:

```
1. def Average(l):
2.     m = len(l)
3.     t = 0.0
4.     for i in range(m):
5.         t += l[i]
6.     print("%6.1f" % (t/m))
7.
8. van = list(map(float, input('Nhập các điểm văn: ').split()))
9. toan = list(map(float, input('Nhập các điểm toán: ').split()))
10.anh = list(map(float, input('Nhập các điểm anh: ').split()))
11.Average(van)
12.Average(toan)
13.Average(anh)
```

Lợi ích của việc sử dụng chương trình con

- Tránh được việc phải viết lặp đi lặp lại cùng một dãy lệnh nào đó tương tự như trong ví dụ tính trung bình ở trên. Ngôn ngữ lập trình cho phép tổ chức dãy lệnh đó thành một chương trình con (hàm). Sau đó, mỗi khi chương trình chính cần đến dãy lệnh này chỉ cần gọi thực hiện chương trình con đó.
- Khi phải viết chương trình lớn hàng nghìn, hàng vạn lệnh, cần huy động nhiều người tham gia, có thể giao cho mỗi người (hoặc mỗi nhóm) viết một chương trình con, rồi sau đó lắp ghép chúng lại thành chương trình chính. Ví dụ, với các bài toán mà việc tổ chức dữ liệu vào và ra không đơn giản thường người ta chia bài toán thành ba bài toán con như nhập, xử lí và xuất dữ liệu, rồi viết các chương trình con tương ứng.

- Phục vụ cho quá trình trừu tượng hoá: Người lập trình có thể sử dụng các kết quả được thực hiện bởi chương trình con mà không phải quan tâm đến việc các thao tác này được cài đặt như thế nào. Trừu tượng hoá là tư tưởng chủ đạo để xây dựng chương trình nói chung và chương trình có cấu trúc nói riêng.
- Các ngôn ngữ lập trình thường cung cấp phương thức đóng gói các chương trình con nhằm cung cấp như một câu lệnh mới (tương tự như các lệnh gọi thực hiện các hàm và thủ tục chuẩn) cho người lập trình sử dụng mà không cần biết mã nguồn của nó như thế nào.
- Thuận tiện cho phát triển, nâng cấp chương trình. Do chương trình được tạo thành từ các chương trình con nên chương trình dễ đọc, dễ hiểu, dễ kiểm tra và hiệu chỉnh. Việc nâng cấp, phát triển chương trình con nào đó, thậm chí bổ sung thêm các chương trình con mới nói chung không gây ảnh hưởng đến các chương trình con khác.
- Hiện nay, ngày càng có nhiều thiết bị kĩ thuật số tiện ích như máy quay phim, máy ảnh, máy ghi âm, các thiết bị âm thanh, màn hình màu độ phân giải cao,... có thể được kết nối với máy tính. Việc thiết kế những chương trình con thực hiện các giao tiếp cơ bản với các thiết bị như vậy là rất cần thiết và giúp mở rộng khả năng ứng dụng của ngôn ngữ.

2. Phân loại và cấu trúc của chương trình con

a) Phân loại

Trong NNLT Python, chương trình con chính là hàm, hàm có thể chia làm hai loại:

- *Hàm có kết quả* (Fruitful functions) là loại hàm thực hiện một số thao tác nào đó và trả về một số giá trị theo sau lệnh ***return***. Ví dụ hàm toán học hay hàm xử lí xâu:

  sin(x) nhận vào giá trị thực *x* và trả về giá trị *sinx*,

  sqrt(x) nhận vào giá trị *x* trả về giá trị căn bậc hai của *x*,

  len(x) nhận vào xâu hoặc danh sách *x* và trả về độ dài của xâu hoặc danh sách *x*,...

- *Hàm không có kết quả* (Void functions) là hàm thực hiện một số thao tác nhất định nhưng không trả về giá trị nào. Ví dụ hàm *print(), write()*...

b) Cấu trúc của hàm

Hàm có cấu trúc tương tự chương trình, nhưng nhất thiết phải có tên, phần đầu dùng để khai báo tên, phần thân chứa các lệnh:

```
<phần đầu>
<phần thân>
```

Phần đầu là khai báo tên hàm và danh sách tham số hình thức.

Phần thân có thể có khai báo biến cho dữ liệu vào, các hằng và biến dùng trong hàm. Thành phần chủ yếu của thân hàm là dãy các câu lệnh thực hiện để từ những dữ liệu vào ta nhận được dữ liệu ra hay kết quả mong muốn, và nếu là hàm có kết quả thì nó chứa lệnh return trả về kết quả đó.

Tham số hình thức

Các biến được khai báo cho dữ liệu vào/ra được gọi là *tham số hình thức* của hàm. Các biến được khai báo để dùng riêng trong thân hàm được gọi là *biến cục bộ*.

Ví dụ, trong hàm *Average(l)* ở trên thì *l* là các tham số hình thức và *m, t* là các biến cục bộ.

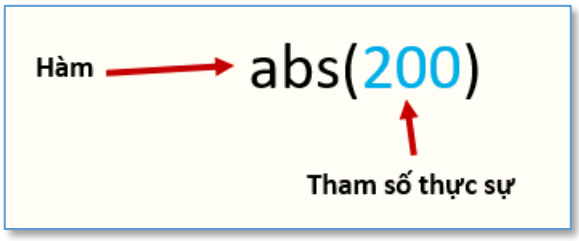
Nói chung, chương trình chính và các hàm khác không nhìn thấy các biến cục bộ của một hàm nhưng mọi hàm đều sử dụng được các biến của chương trình chính. Do vậy, các biến của chương trình chính được gọi là *biến toàn cục*. Ví dụ, các biến *van, toan, anh* khai báo trong chương trình chính ở ví dụ trên là biến toàn cục.

Hàm có thể có hoặc không có tham số hình thức và biến cục bộ.

c) Thực hiện hàm

Tham số thực sự

Để thực hiện (gọi) một hàm, ta cần phải có lệnh gọi nó, tương tự lệnh gọi các hàm dựng sẵn của Python, bao gồm tên hàm với tham số (nếu có) là các hằng và biến chứa dữ liệu vào và ra tương ứng với các tham số hình thức đặt trong cặp ngoặc (và). Các hằng và biến này được gọi là các *tham số thực sự*.



Hình 6.1 - Minh họa về hàm và tham số thực sự

Khi thực hiện hàm, các tham số hình thức để nhập dữ liệu vào sẽ nhận giá trị của tham số thực sự tương ứng, còn các tham số hình thức để lưu trữ dữ liệu ra sẽ trả giá trị đó cho tham số thực sự tương ứng.

Ví dụ, khi thực hiện tính trung bình cần ba lần gọi hàm *Average(l)* với tham số (*van*), (*toan*), (*anh*) và các tham số này là tham số thực sự tương ứng với tham số hình thức (*l*).

Sau khi hàm kết thúc, lệnh đứng ngay sau lệnh gọi hàm sẽ được thực hiện.

Bài 18. Ví dụ về cách định nghĩa và sử dụng hàm

Các ngôn ngữ lập trình đều có các quy tắc viết và sử dụng chương trình con. Trong mục này sẽ xét cách viết và sử dụng hàm trong Python.

1. Cách định nghĩa hàm

Định nghĩa hàm có cú pháp như sau:

```
def <tên hàm>([danh sách tham số]) :
    [<dãy các lệnh>]
    [return <dãy giá trị trả về>]
```

Dòng đầu tiên của định nghĩa hàm là từ khóa ***def***, tiếp theo là tên hàm, sau đó là cặp ngoặc đơn và kết thúc là dấu hai chấm (:). Danh sách tham số (nếu có) được liệt kê trong cặp ngoặc đơn và phân cách bằng dấu phảy.

Phần thân hàm là dãy các câu lệnh được viết thụt lề so với dòng đầu của định nghĩa hàm.

Chú ý

- Dùng định nghĩa hàm có tham số hình thức hay không thì cặp ngoặc đơn nhất thiết phải có và cuối dòng đầu tiên phải có dấu hai chấm.
- Các lệnh trong thân hàm phải viết thụt lề (thường là 3-4 dấu cách hoặc 1 phím tab)
- Nếu là hàm có kết quả thì trong thân hàm sẽ có lệnh ***return*** và theo sau là dãy giá trị trả về. Khác với NNLT Pascal/C++, Python cho phép hàm trả về nhiều giá trị thuộc kiểu bất kỳ.

2. Một số ví dụ về hàm

a) Ví dụ hàm không có kết quả

Xét ví dụ vẽ hình chữ nhật có dạng sau:

```
* * * * * * *
*           *
* * * * * * *
```

Ta có thể vẽ hình chữ nhật trên với ba câu lệnh:

```
print('* * * * * * *')
print('*           *')
print('* * * * * * *')
```

Như vậy, trong một chương trình, mỗi khi cần vẽ một hình chữ nhật như trên cần phải đưa vào ba câu lệnh này.

Trong chương trình sau, ta đưa ba câu lệnh trên vào một hàm có tên là *Ve_Hcn* (vẽ hình chữ nhật). Mỗi khi cần vẽ một hình chữ nhật ta cần đưa vào một câu lệnh gọi hàm đó. Chương trình gọi hàm *Ve_Hcn()* ba lần để vẽ ba hình chữ nhật.

Hàm Ve_Hcn() chỉ đơn thuần là in ra 3 dòng trên màn hình và hàm không trả về giá trị nào.

```
1. def Ve_hcn():
2.     print('* * * * * * *')
3.     print('*           *')
4.     print('* * * * * * *')
5.
6. Ve_hcn()
7. print('\n\n')
8. Ve_hcn()
9. print('\n\n')
10.Ve_hcn()
```

Hàm *Ve_Hcn()* trong ví dụ trên chỉ vẽ được hình chữ nhật với kích thước cố định là 7 × 3. Giả sử chương trình cần vẽ nhiều hình chữ nhật với kích thước khác nhau. Để hàm *Ve_Hcn()* có thể thực hiện được điều đó, cần có hai tham số vào là chiều dài, chiều rộng và dòng đầu của định nghĩa hàm được viết lại như sau:

```
def Ve_Hcn(dai, rong):
```

Khai báo này có nghĩa hàm *Ve_Hcn()* sẽ được thực hiện để vẽ hình chữ nhật có kích thước tuỳ theo giá trị của các tham số *dai*, *rong* và giá trị của các tham số *dai* và *rong* là nguyên.

Trong chương trình sau đây mô tả đầy đủ hàm *Ve_Hcn()* với các tham số *dai*, *rong* và sử dụng hàm này để vẽ các hình chữ nhật có kích thước khác nhau.

```
1. def Ve_Hcn(dai, rong):
2.     for i in range(dai):
3.         print('*',end=' ')
4.     print('')
5.     for i in range(rong-2):
6.         print('*', end=' ');
7.         for j in range(dai-2):
8.             print(' ', end=' ')
9.         print('*')
10.    for i in range(dai):
11.        print('*',end=' ')
12.    print('')
13.
14.Ve_Hcn(25, 10)
15.print('\n\n')
16.Ve_Hcn(5, 10)
17.print('\n\n')
18.a = 4
19.b = 2
20.for i in range(4):
21.    Ve_Hcn(a,b)
22.    a *= 2
23.    b *= 2
```

Các tham số *dai*, *rong* của hàm *Ve_Hcn()* là tham số. Trong lệnh gọi hàm *Ve_Hcn(5, 3)* (vẽ hình chữ nhật kích thước 5 × 3) tham số *dai* được thay bởi số nguyên 5, tham số *rong* được thay bởi số nguyên 3.

Còn trong lời gọi hàm *Ve_Hcn(a, b)* vẽ hình chữ nhật kích thước *a* × *b*, tham số *dai* được thay bởi giá trị hiện thời của biến *a*, tham số *rong* được thay bởi giá trị hiện thời của biến *b*.

b) Ví dụ hàm có kết quả

Ví dụ 1

Xét chương trình thực hiện việc rút gọn một phân số, trong đó có sử dụng hàm tính ước chung lớn nhất (UCLN) của hai số nguyên.

```
1. # Phần định nghĩa hàm
2. def UCLN(x, y):
3.     while y != 0:
4.         du = x % y
5.         x = y
6.         y = du
7.     return x
8.
9. def DC(x, y):
10.    return y, x
11.
12.# Phần chương trình chính
13.tu, mau = map(int, input("Nhập tử số và mẫu số: ").split())
14.a = UCLN(tu, mau)
15.if a > 1:
16.    tu //= a
17.    mau //= a
18.print("%5d %5d" % (tu, mau))
```

*Chú ý: Trong chương trình này, các biến **tu, mau** và **a** là các biến toàn cục, còn biến **du** là biến cục bộ.*

Vì hàm *UCLN()* có trả về kết quả, nên khi gọi hàm ta thường đặt hàm vào vế phải của phép gán để gán giá trị trả về của hàm cho một biến nào đó. Ví dụ:

```
a = 6*UCLN(tu, mau) + 1
```

Ví dụ 2

Chương trình sau cho biết chu vi và diện tích của một hình chữ nhật có hai cạnh *a, b*. Trong đó có sử dụng hàm tính chu vi và diện tích hình chữ nhật.

```
1. # định nghĩa hàm
2. def chuvi_dientich(x, y):
3.     return (x + y)*2, x*y
4.
5. # chương trình chính
6. a, b = map(int, input("Nhập độ dài hai cạnh hình chữ nhật: ").split())
7. cv, dt = chuvi_dientich(a, b)
8. print("%8d%8d" % (cv, dt))
```

Khi gọi hàm ta thấy vế trái có 2 biến *cv* (chu vi) và *dt* (diện tích) đều nhận được hết quả trả về, đây là điểm khác biệt của NNLT Python so với Pascal và C++.

3. Vấn đề tham số của hàm trong Python

Các tham số hàm trong Python được chia làm hai loại: *bất biến* (immutable), *khả biến* (mutable).

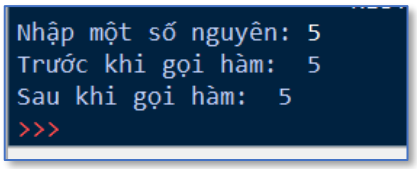
- Kiểu tham số bất biến gồm các kiểu: *số nguyên, số thực, chuỗi* và *bộ* (tuble): Các tham số có kiểu này khi được gửi cho hàm thì các tác động trong thân hàm lên chúng không làm thay đổi giá trị của chúng sau khi hàm kết thúc.
- Kiểu tham số khả biến gồm các kiểu: *danh sách* (list), *tập hợp* (set), *từ điển* (dict): Các tham số kiểu này khi được gửi cho hàm thì các tác động trong thân hàm lên chúng sẽ làm thay đổi giá trị của chúng sau khi hàm kết thúc.

Ví dụ 1: Minh họa về tham số bất biến

Nhập vào một số nguyên, gửi số đó cho hàm để nhân đôi giá trị của biến và quan sát xem biến có bị thay đổi giá trị sau khi gọi hàm hay không.

```
1. def gap_doi(x):
2.     x *= 2
3.
4. a = int(input("Nhập một số nguyên: "))
5. gap_doi(a)
6. print(a)
```

Kết quả chạy chương trình với giá trị nhập vào bằng 5 như sau:



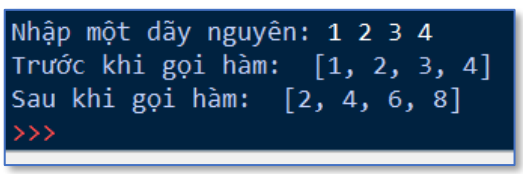
Ta thấy, giá trị biến *a* sau khi hàm *gap_doi()* tác động không hề thay đổi.

Ví dụ 2: Minh họa về tham số khả biến

Gửi cho hàm một dãy số đó để nhân đôi giá trị của mỗi phần tử của dãy và quan sát xem dãy có bị thay đổi giá trị sau khi gọi hàm hay không.

```
1. def gap_doi(x):
2.     for i in range(len(x)):
3.         x[i] *= 2
4.
5. a = list(map(int,input("Nhập một dãy nguyên: ").split()))
6. print("Trước khi gọi hàm: ", a)
7. gap_doi(a)
8. print("Sau khi gọi hàm: ", a)
```

Kết quả chạy chương trình với giá trị dãy nhập vào là : 1 2 3 4, như sau:



Ta thấy, giá trị các phần tử của danh sách *a* sau khi hàm *gap_doi()* tác động đã thay đổi.

Bài tập và thực hành 6

1. Mục đích, yêu cầu

- Rèn luyện các thao tác xử lí xâu;
- Nâng cao kĩ năng viết, sử dụng hàm.

2. Nội dung

a) Tìm hiểu việc xây dựng hai hàm sau đây

- Hàm *CatDan(S)* nhận đầu vào là xâu *s* gồm không quá 79 kí tự, tạo ra xâu kết quả *t* thu được từ xâu *s* bằng việc chuyển kí tự đầu tiên của *s* xuống cuối xâu. Ví dụ nếu *s = 'abcd'* thì *t = 'bcda'*.

```
1. def CatDan(s):
2.     t = s[1:]+s[0]
3.     return t
```

- Hàm *CanGiua(s)* nhận đầu vào là xâu *s* gồm không quá 79 kí tự, bổ sung vào đầu *s* một số dấu cách để khi đưa ra màn hình xâu kí tự trong *s* ban đầu được căn giữa dòng gồm 80 kí tự.

```
1. def CanGiua(s):
2.     n = len(s)
3.     n = (80-n)//2
4.     t = ' '*n + s
5.     return t
```

b) Theo dõi cách sử dụng hai hàm trên, ta có thể viết chương trình sau đây để nhập một xâu kí tự từ bàn phím và đưa xâu đó ra màn hình có dạng dòng chữ chạy giữa màn hình văn bản 25×80.

```
1. # Bài thực hành 6
2. import time
3. def CatDan(s):
4.     t = s[1:]+s[0]
5.     return t
6.
7. def CanGiua(s):
8.     n = len(s)
9.     n = (80-n)//2
10.    t = ' '*n + s
11.    return t
12.
13.s = input("Nhập xâu s: ")
14.s = CanGiua(s)
15.try:
16.    while True:
17.        print(s)
18.        time.sleep(0.5)   # Dừng 0.5 giây
19.        s = CatDan(s)
20.        print('\n'*30)
21.except KeyboardInterrupt: # Dừng khi bấm CTRL + C
22.    pass
```

Hãy chạy thử chương trình trên với dòng chữ:

'*... Mừng nghìn năm Thăng Long - Hà Nội!...* '

c) Hãy viết hàm *ChuChay(s, dong)* nhận đầu vào là xâu *s* gồm không quá 79 kí tự và biến nguyên *dong*, đưa ra xâu *s* có dạng chữ chạy ở dòng *dong*. Viết và chạy chương trình có sử dụng hàm này.

Bài tập và thực hành 7

1. Mục đích, yêu cầu

- Nâng cao kĩ năng viết, sử dụng hàm;
- Biết cách viết một chương trình có cấu trúc để giải một bài toán trên máy tính.

2. Nội dung

a) Tìm hiểu việc xây dựng các hàm và thủ tục thực hiện tính độ dài các cạnh, chu vi, diện tích, kiểm tra các tính chất đều, cân, vuông của tam giác được trình bày dưới đây.

Giả thiết tam giác được xác định bởi toạ độ của ba đỉnh. Ta xây dựng các hàm:

- Hàm nhận dữ liệu vào là biến mô tả tam giác R và đầu ra là độ dài của ba cạnh a, b, c:

  `DaiCanh(R)`
- Hàm tính chu vi của tam giác R:

  `Chuvi(R)`
- Hàm tính diện tích của tam giác R:

  `Dientich(R)`
- Hàm nhận đầu vào là biến mô tả tam giác R và đầu ra là tính chất của tam giác (Deu hay Can hay Vuong):

  `Tinhchat (R)`
- Hàm hiển thị toạ độ ba đỉnh tam giác lên màn hình:

  `Hienthi(R)`
- Hàm tính khoảng cách giữa hai điểm P, Q:

  `Kh_cach (P, Q)`

b) Tìm hiểu chương trình nhập vào toạ độ ba đỉnh một tam giác và sử dụng các hàm được xây dựng dưới đây để khảo sát các tính chất của tam giác.

```
1.   import math
2.
3.   def Kh_Cach(p, q):
4.       return math.sqrt((p[0]-q[0])**2 + (p[1]-q[1])**2)
5.
6.   def DaiCanh(r):
```

```
7.        a = Kh_Cach(r[1], r[2])
8.        b = Kh_Cach(r[0], r[2])
9.        c = Kh_Cach(r[0], r[1])
10.       return a, b, c
11.
12.   def ChuVi(r):
13.       a, b, c = DaiCanh(r)
14.       return a + b + c
15.
16.   def DienTich(r):
17.       a, b, c = DaiCanh(r)
18.       p = (a + b + c) / 2
19.       return math.sqrt(p*(p-a)*(p-b)*(p-c))
20.
21.   def HienThi(r):
22.       print("Tọa độ 3 đỉnh của tam giác là:")
23.       print("- Đỉnh A(%.3f,%.3f" % (r[0][0], r[0][1]))
24.       print("- Đỉnh B(%.3f,%.3f" % (r[1][0], r[1][1]))
25.       print("- Đỉnh C(%.3f,%.3f" % (r[2][0], r[2][1]))
26.
27.   def TinhChat(r):
28.       a, b, c = DaiCanh(r)
29.       tl = ''
30.       if a == b == c:
31.           t += "đều"
32.       if a == b or b == c or a == c:
33.           tl += "cân "
34.       if a**2 == b**2 + c**2 or \
35.          b**2 == a**2 + c**2 or \
36.          c**2 == a**2 + b**2:
37.           tl += "vuông"
38.       return tl
39.
40.   print("Nhập tam giác")
41.   x1, y1 = map(float, input("Nhập tọa độ đỉnh A: ").split())
42.   x2, y2 = map(float, input("Nhập tọa độ đỉnh B: ").split())
43.   x3, y3 = map(float, input("Nhập tọa độ đỉnh C: ").split())
44.   tg = ((x1, y1), (x2, y2), (x3, y3))
45.   HienThi(tg)
46.   print("Diện tích: %9.3f" % DienTich(tg))
47.   print("Chu vi: %9.3f" % ChuVi(tg))
48.   print("Tam giác có tính chất là tam giác "+TinhChat(tg))
```

c) Viết chương trình sử dụng các hàm và thủ tục xây dựng ở trên để giải bài toán:

Cho tệp dữ liệu TAMGIAC.DAT có cấu trúc như sau:

- Dòng đầu tiên chứa số *N*;
- N dòng tiếp theo, mỗi dòng chứa sáu số thực *xA, yA, xB, yB, xC, yC* là toạ độ ba đỉnh *A(xA, yA), B(xB, yB), C(xC, yC)* của tam giác *ABC*.

Hãy nhập dữ liệu từ tệp đã cho. Trong số *N* tam giác đó, đưa ra tệp TAMGIAC.OUT ba dòng:

- Dòng đầu tiên là số lượng tam giác đều;
- Dòng thứ hai là số lượng tam giác cân (nhưng không là đều);
- Dòng thứ ba là số lượng tam giác vuông.