Chương I
Một số khái niệm về lập trình và ngôn ngữ lập trình

BÀI 1. KHÁI NIỆM LẬP TRÌNH VÀ NGÔN NGỮ LẬP TRÌNH

Như đã 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 máy, nghĩa là một chương trình có thể thực hiện trên nhiều máy. 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 trong 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 trong 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).

Chương trình nguồn → Chương trình dịch → 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 (interpreter)

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

① Kiểm tra tính đúng đắn của câu lệnh tiếp theo trong chương trình nguồn;

② Chuyển đổi câu lệnh đó thành một hay nhiều câu lệnh tương ứng trong ngôn ngữ máy;

③ Thực hiện các câu lệnh vừa chuyển đổi được.

Như vậy, quá trình dịch và thực hiện các câu lệnh là tuần tự. 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 (compiler)

Biên dịch được thực hiện qua hai bước:

① Duyệt, kiểm tra, phát hiện lỗi, kiểm tra tính đúng đắn của các câu lệnh trong chương trình nguồn;

② 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 về sau.

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 Pascal 1.2, Visual Pascal 2.1,... trên ngôn ngữ Pascal, C++, Python.

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 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 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 phẩy)
;	#	^	$	@	&	(	)	{	}	:	' (dấu nháy)
dấu cách (mã ASCII 32)									_ (dấu gạch dưới)		

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 biểu thức:

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 nguyên. Khi đó dấu "+" trong biểu thức (1) được hiểu là cộng hai số thực, dấu "+" trong biểu thức (2) được hiểu là cộng hai số nguyên. 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

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.

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

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

Tên dành riêng

Một số 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

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

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.

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

Ví dụ

- Hằng số học:

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

- Hằng logic:

```
True        False
```

- Hằng xâu:

+ Trong Pascal: `'Informatic'` `'TIN HOC'`

+ Trong PYTHON: `'Informatic'` `"TIN HOC"`

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 gồm 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 chuẩn.

5. Hãy tự viết ra ba tên đúng theo quy tắc của Python và có độ dài khác nhau.

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"
```

Chương II
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 tùy 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 tắc 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: Trong Python có một số cách, sau đây là một cách

```
import <math>
```

Thư viện `math` trong Python 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`, sau khi khai báo thư viện `math`, ta dùng lệnh:

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

Khai báo hằng

Ví dụ: Trong Python

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

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 tình dịch biết để lưu trữ và xử lý. 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

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 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 là 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!") - Phần khai báo không có - Phần thân chương trình chỉ có một câu lệnh *print* đưa thông báo ra màn hình	*#include <studio.h>* *{* *Printf ("xin chao cac ban!");* *}* - Phần khai báo chỉ có một câu lệnh *#include* khai báo thư viện *studio.h* - Phần thân chương trình chỉ có một câu lệnh *printf* đưa thông báo ra màn hình	*Program vi_du;* *Begin* *writeln ('xin chao cac ban!');* *End.* - Phần khai báo chỉ có khai báo tên chương trình gồm tên dành riêng *program* và tên chương trình đặt 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 ra 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, người lập trình sử dụng các kiểu dữ liệu đó thường gặp hạn chế nhất định, phụ thuộc vào các 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.

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.

1. Kiểu số

Python hỗ trợ các kiểu số nguyên, số thực, số phức.

Ví dụ 1

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)

Ví dụ 2

Để yêu cầu nhập giá trị nguyên cho x, câu lệnh viết:

```
x = int(input('nhap so nguyen x: '))
```

Để yêu cầ nhập giá trị thực cho x, câu lệnh viết:

```
x = float(input('nhap so thuc x: '))
```

Một điều thú vị là kiểu nguyên trong Python cho phép ta tính toán với sô 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:

```
933262154439441526816992388562667004907159682643816214685929638952
175999932299156089414639761565182862536979208272237582511852109168
640000000000000000000000000
```

2. Kiểu lôgic

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

BÀI 5. BIẾN TRONG PYTHON

Như đã nói ở BÀI 2, 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 là 245)

`Lop, TenHS, Pi = 14, 'Nguyen Van A', 3.14` (lần lượt biến `Lop` là biến nguyên có giá trị `14`, biến `TenHS` là biến xâu có giá trị là `Nguyen Van A`, biến `Pi` là biến thực có giá trị là `3.14`)

Một số chú ý khi sử dụng 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 cần thiết.

  Ví dụ, không nên đặt tên biến là *a, b* 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 số học sinh thì dùng kiểu nguyên, nhưng biến biểu diễn điểm trung bình của học sinh thì dùng kiểu thực.

- 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 gái 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, CÂU 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:

Phép toán	Trong toán học	Trong Python
Các phép toán số học với số nguyên	+ (cộng), - (trừ), . (nhân), : (chia), mod (lấy phần dư), div (chia nguyên), phép lũy thừa	+, -, *, /, %, //, **
Các phép toán số học với số thực	+ (cộng), - (trừ), . (nhân), : (chia), phép lũy thừa	+, -, *, /, **
Các phép toán quan hệ	< (nhỏ hơn), ≤ (nhỏ hơn hoặc bằng), > (lớn hơn), ≥ (lớn hơn hoặc bằng), = (bằng), ≠ (khác)	<, <=, >, >=, ==, !=
Các phép toán lôgic	¬ (phủ định), ∨ (hoặc), ∧ (và)	not, and, or

Chú ý

- Kết quả của các phép toán quan hệ cho giá trị lôgic.
- Một trong những ứng dụng của phép toán lôgic là để tạo ra các biểu thức phức tạp từ các quan hệ đơn giản.
- 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 tròn (và) tạo thành một biểu thức có dạng tương tự như cách viết trong toán học với những quy tắc sau:

- Chỉ dùng cặp ngoặc tròn để xác định trình tự thực hiện phép toán trong trường hợp cần thiết;
- Viết lần lượt từ trái qua phải;
- Các phép toán được thực hiện theo thứ tự:

  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 (div), lấy phần dư (mod) thực hiện trước và các phép toán cộng (+), trừ (-) thực hiện sau.

Ví dụ

Biểu thức trong Toán học	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

- Nếu biểu thức chứa một hằng hay biến kiểu thực thì ta có biểu thức số học thực.
- Trong một số trường hợp nên dùng biến trung gian để có thể tránh được việc tính một biểu thức nhiều lần.
- Theo mặc định Python yêu cầu mỗi lệnh đượ 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 tròn (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 trong Python	Kiểu đối số	Kiểu kết quả
Lũy thừa	x^y	**x**y**	Thực hoặc nguyên	Theo kiểu của đối số
Căn bậc hai	$\sqrt{x}$	**sqrt(**x**)**	Thực hoặc nguyên	Thực
Giá trị tuyệt đối	$\|x\|$	**abs(**x**)**	Thực hoặc nguyên	Theo kiểu của đối số
Lôgarit tự nhiên	Lnx	**ln(**x**)**	Thực	Thực
Luỹ thừa của số e	e^x	**exp(**x**)**	Thực	Thực
Sin	Sinx	**sin(**x**)**	Thực	Thực
Cos	Cosx	**cos(**x**)**	Thực	Thực

Để dùng các hàm số học phải có lệnh khai báo thư viện *math (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*b - 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*R
```

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*. 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 cần phải tách thành phát biểu dưới dạng "$5 \le x$ và $x \le 11$" và được viết như sau:

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

Python còn cho phép viết tắt: `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 - math.sqrt(b*b - 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 THỦ TỤC 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 *thủ tục chuẩn vào/ra đơn giản*.

Trong phần này, ta sẽ xét các *thủ tục 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 được thực hiện bằng thủ tục chuẩn ***input()***

Trong đó, biến sẽ nhận giá trị là kiểu xâu, vì vậy muốn về giá trị kiểu gì ta phải ép kiểu cho giá trị đó.

Ví dụ

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

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

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

Xâu thông báo là xâu 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 xâu thông báo nhưng 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([xâu thông báo]).split())
```

Sau khi hàm *input()* nhận được xâu đầu vào từ bàn phím, nó gọi hàm *split()* để trích xuất xâu dữ liệu đầu vào thành các xâu con, mặc định là căn cứ vào các khoảng trắng (dấu cách). Mỗi khi *split()* trích xuất được xâu con nào thì nó gửi cho hàm *int()* chuyển đổi xâu 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 hoặc số thực dạng dấu phẩy động.

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

```
1 -5 6
```

rồi gõ phím Enter

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

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

Để nhập các giá trị 1, -1.2 và 4 cho các biến thực a, b, c, có thể gõ:

```
1 -1.2 4
```

rồi nhấn phím Enter

2. Đưa dữ liệu ra màn hình

a. Hàm print()

Để đưa dữ liệu ra màn hình, Python cung cấp thủ tục 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: "))
```

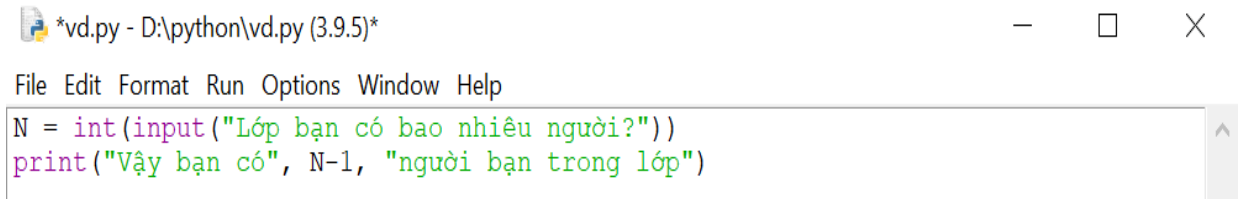
Khi thực hiện lệnh này, màn hình xuất hiện hộp thoại Python Shell với dòng chữ:

```
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 thủ tục vào/ra chuẩn của Python.

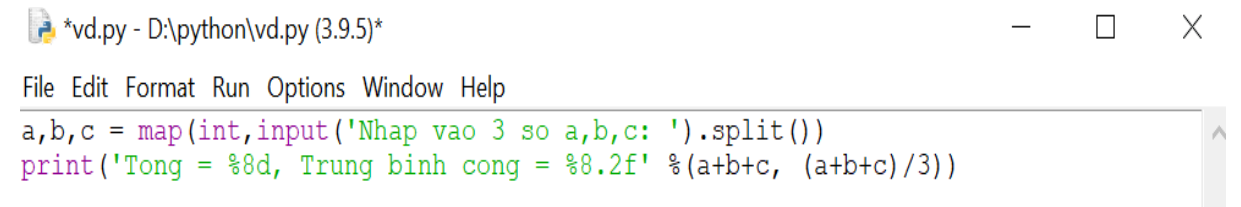


b. Xâu định dạng đầu ra

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 thủ tục *print()* một xâu thông số gọi là *xâu định dạng đầu ra* để yêu cầu thủ tục in theo định dạng.

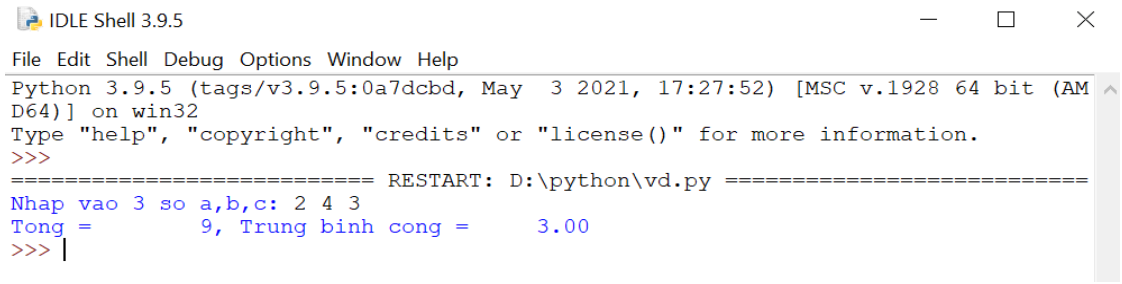
Ví dụ 3

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 được in vơi 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

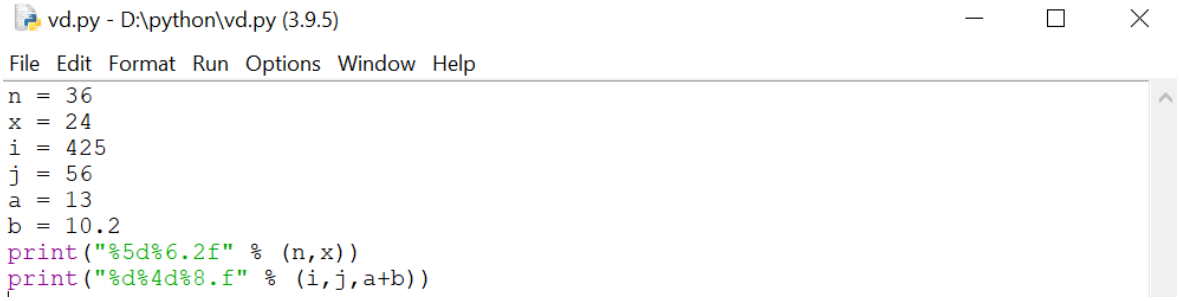
Chạy chương trình, xuất hiện màn hình nhập dữ liệu, nhập vào 3 giá trị: 2 4 3, chương trình sẽ in ra màn hình kết quả như sau:



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

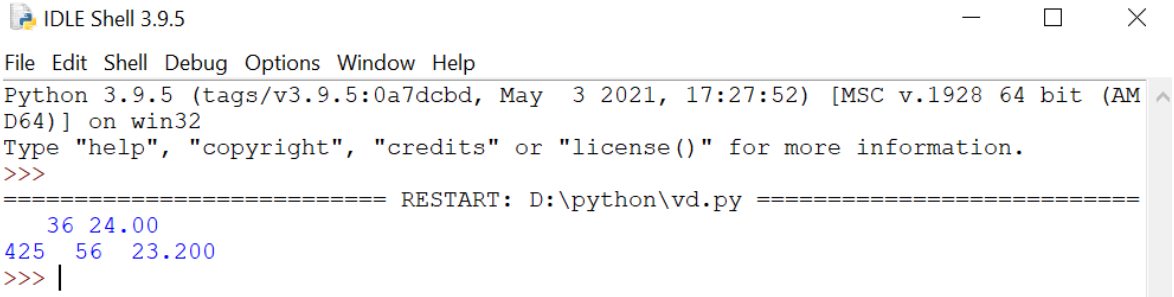
Xâu định dạng đầu ra là xâu ký tự bắt đầu bằng ký hiệu %, tiếp theo là tham số độ rộng trên màn hình, nếu là vị trí sẽ in ra số thực thì sau tham số độ rộng sẽ có dấu chấm và tiếp đó là số chữ số thập phân đó. Trong xâu "*Tổng = %8d, Trung bình cộng =%8.2f*" thì hàm print() in ra một xâu thông báo, như xen trong xâu đó có hai vị trí chứa xâu 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 xâu 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.

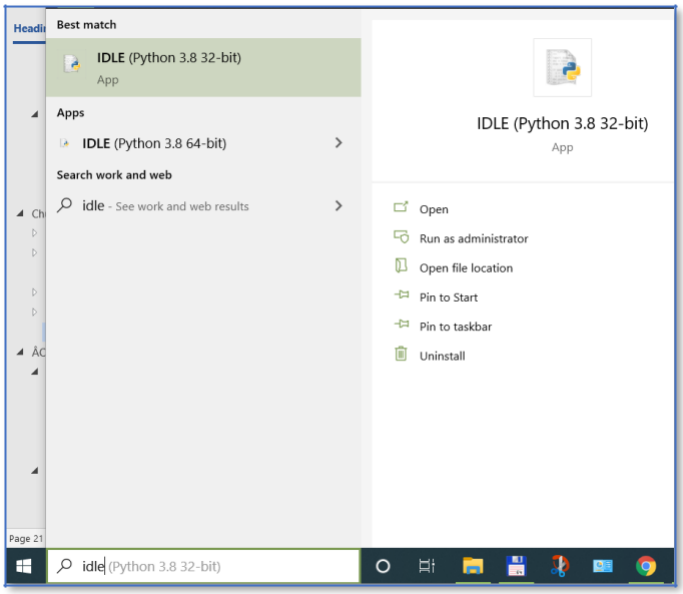
Dòng lệnh 8, đầu tiên là vị trí dành cho biến *i* với 3 chỗ và giá trị của biểu thức *a + b* với 8 chỗ, trong đó có 3 chỗ dành cho phần thập phân.

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.

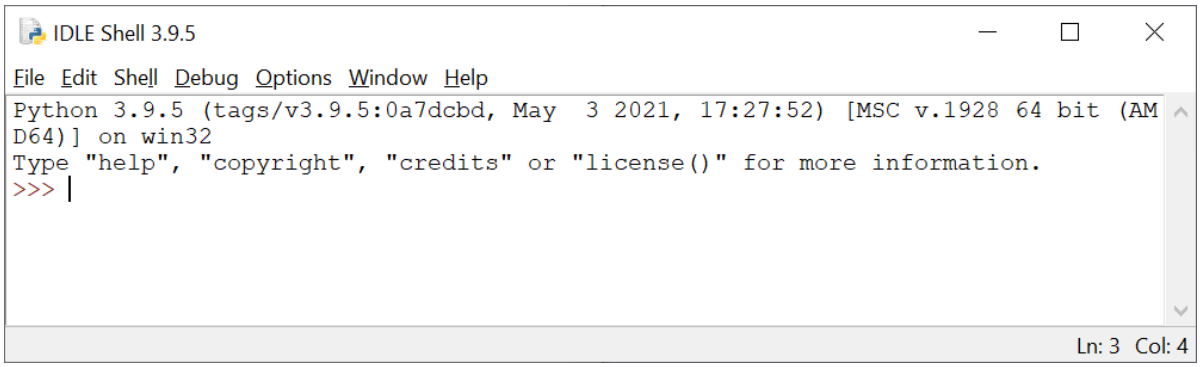
Với ngôn ngữ Python, khi bạn cài đặt phần mềm Python 3.x trên máy tính, thì công cụ lập trình đơn giản nhất gọi là Python IDLE (*Integrated Development and Learning Environment*) đã được cài đặt sẵn. Từ đây cuốn tài liệu này chỉ giới thiệu cách làm việc với Python IDLE.

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

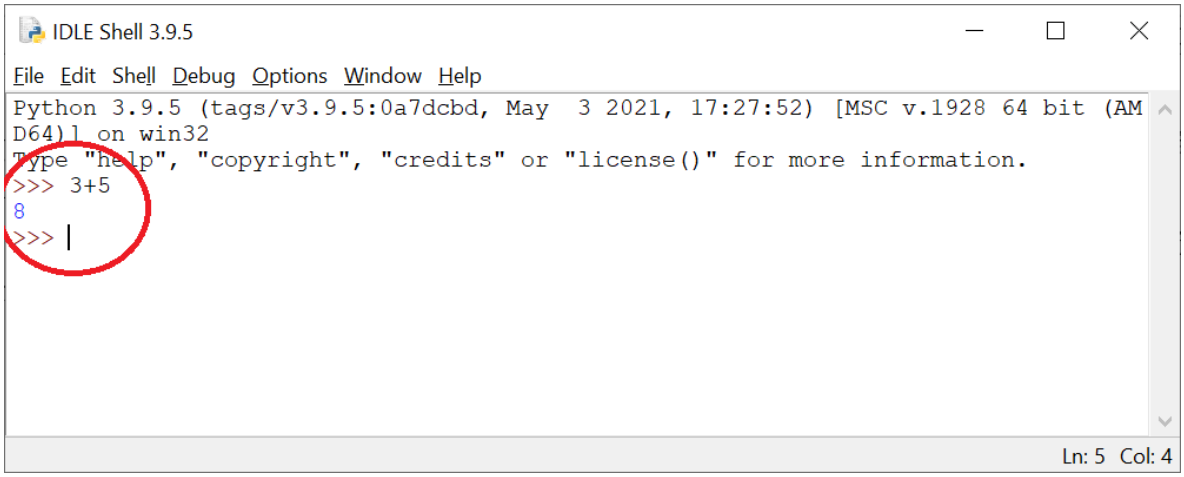
Bạn sẽ nhìn thấy màn hình tương tự như trên. Bạn cũng có thể tìm thấy công cụ Python IDLE trong menu start của windows. Bấm đúp chuột vào công cụ IDLE (hoặc Enter), bạn sẽ thấy màn hình tương tự như sau:



Hình 2.6 - Màn hinh Python Shell

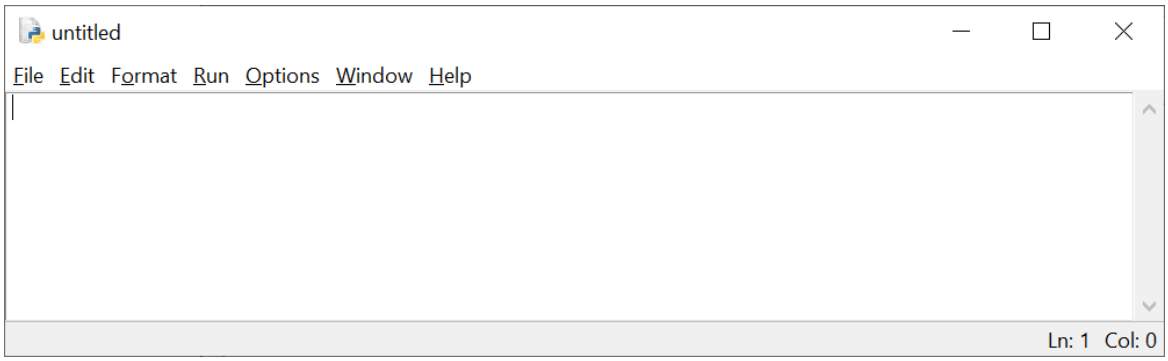
Đây chính là giao diện cửa số lệnh (Python Shell) để bạn có thể chạy các câu lệnh đơn giản của Python ngay từ dấu nhắc >>>.

Ví dụ: Từ dấu nhắc bạn gõ biểu thức toán *3 + 5*, sau đó nhấn 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

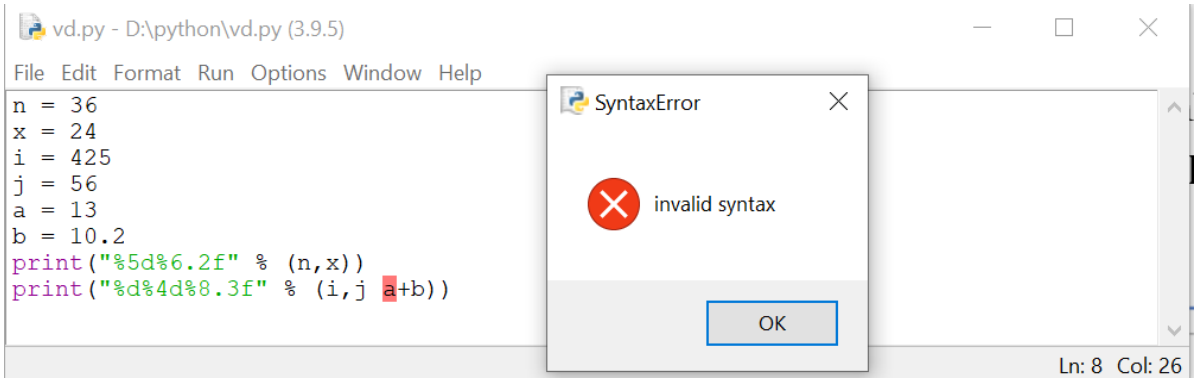
Soạn thảo: một chương trình, bạn chọn menu *File* → *New File* (*CTRL + N*), xuất hiện màn hình soạn thảo như sau:



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

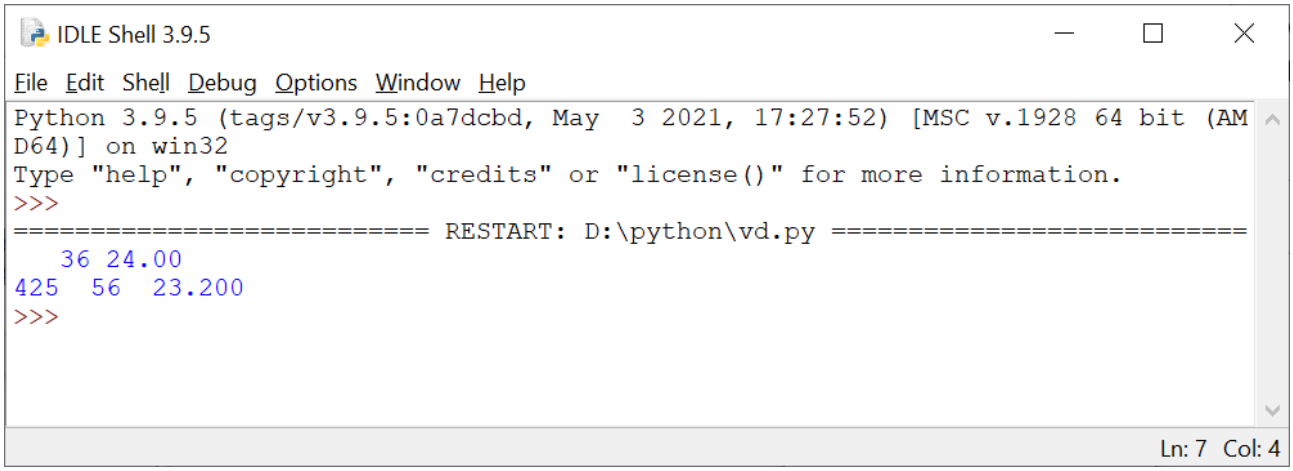
Trong cửa sổ này bạn có thể soạn thảo các lệnh của chương trình như cách soạn thảo văn bản thông dụng và lưu chương trình vào đĩa bằng chức năng menu *File* → *Save* (phím tắt **CTRL + S**) nhập tên chương trình và nhấn **Enter** (phần tên mở rộng ngầm định của tệp chương trình là **.py**). Các chức 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 trong việc soạn thảo, lưu trữ, dịch và thực hiện chương trình.

2. Nội dung

a. Gõ chương trình sau:

```
#GIẢI PHƯƠNG TRÌNH BẬC 2
a = int(input('a= '))           #1
b = int(input('b= '))           #2
c = int(input('c= '))           #3
import math                     #4
d = b*b-4*a*c                   #5
x1 = (-b-math.sqrt(d))/(2*a)    #6
x2 = -b/a-x1                    #7
print ('x1= ',x1)               #8
print ('x2= ',x2)               #9
```

Chú ý

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

b. Nhấn Ctrl + S và lưu chương trình với tên là ptb2.py lên đĩa

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

d. Khi hết lỗi cú 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=  1.0
x2=  2.0
```

e. Nhấn phiếm F5 rồi nhập các giá trị 1, 0, -2. Quan sát kết quả hiển thị trên màn hình

```
x1=  -1.4142135623730951
x2=  1.4142135623730951
```

f. Sửa lại chương trình:

```
#1,2,3:  a,b,c = map(int,input('a,b,c: ').split())
#8,9:    print('x1 = %6.2f x2 = %6.2f'%(x1,x2))
```

Thực hiện lại chương trình đã sửa với hai bộ dữ liệu ở câu d (1;-3;2) và câu e (1;0;-2). Quan sát kết quả hiển thị trên màn hình

g. Thực hiện chương trình với bộ dữ liệu (1;1;1). Quan sát kết quả, giải thích?

Chương III
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 Tin học.

Một lần Châu hẹn với Ngọc: "*Chiều mai nếu trời không mưa thì Châu sẽ đến nhà Ngọc*".

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 thuộc dạng mệnh đề thiếu:

Nếu... thì…

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 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 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:

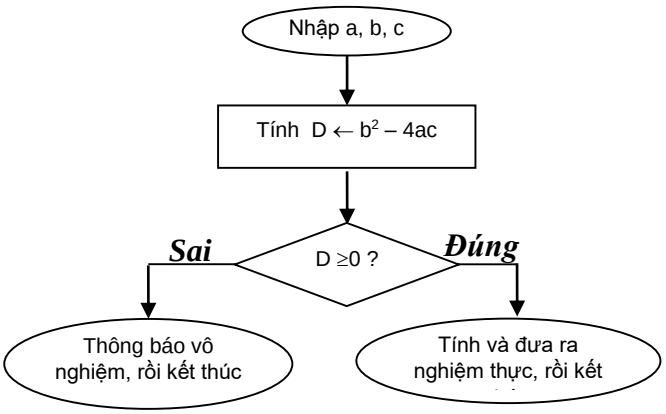
$$ax^2 + bx + c = 0, (a \neq 0)$$

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 (h. 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>
```

Ví dụ

```
a = int(input())
if a%2==0 :
    print(a,"chan")
```

b) Dạng đủ

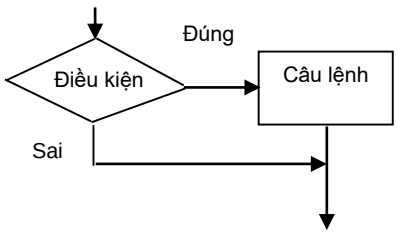
```
if <Điều kiện> :
    <Câu lệnh 1>
else:
    <Câu lệnh 2>
```

Ví dụ

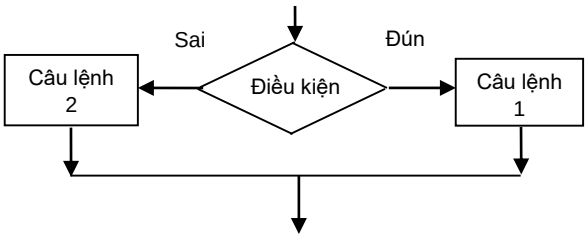
```
a = int(input())
if a%2==0 :
    print(a,"chan")
else:
    print(a,"le")
```

trong đó:

- *Điều kiện:* Biểu thức quan hệ hoặc lôgic.
- *Câu lệnh, câu lệnh 1, câu lệnh 2* là một câu lệnh của CodeBlocks.



Hình 3.2a



Hình 3.2b

- 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 3.2a*).
- 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 3.2b*).

Ví dụ 1

```
if D < 0:
    print('phương trinh vo nghiem')
```

Ví dụ 2

```
if a%3 == 0:
    print ('a chia het cho 3')
else:
    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:

```
max = a
if max < b:
     max = b
```

Dùng một lệnh if dạng đủ:

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

```
if D<0:
     print('Phuong trinh vo nghiem')
else:
     x1=(-b-math.sqrt(b*b-4*a*c))/(2*a)
     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: $ax^2 + bx + c = 0$, 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"

```
#GIẢI PHƯƠNG TRÌNH BẬC 2
a,b,c = map(int,input('a,b,c: ').split())
import math
d = b*b-4*a*c
if d < 0:
  print('Phương trình vô nghiệm')
else:
  x1 = (-b-math.sqrt(d))/(2*a)
  x2 = -b/a-x1
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.

```
#TÍNH NGÀY CỦA NĂM N
N = int(input('năm: '))
sn = 365
if (N%400==0) or (N%4==0 and N%100!=0):
      sn = 366
print('số ngày của năm %d la %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

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

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

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

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 $N = 1, 2, 3, 4, 5,\ldots$ 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 `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*** như sau:

```
for <biến đếm> in range([<giá trị đầu>],<giá trị cuối>,[<bước nhảy>]):
    <câu 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 lần thực hiện *câu 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 hàm range

range (10) → 0; 1; 2; 3; 4; 5; 6; 7; 8; 9

range(1, 10) → 1; 2; 3; 4; 5; 6; 7; 8; 9

range(1, 10, 2) → 1; 3; 5; 7; 9

range(10, 0, -1) → 10; 9; 8; 7; 6; 5; 4; 3; 2; 1

range(10, 0, -2) → 10; 8; 6; 4; 2

Ví dụ 1

```
for i in range(1,10):
    print(i)
```

Ý nghĩa đoạn lệnh trên: .....

.....

Ví dụ 2

```
s=0
for i in range(2,15,2):
    s = s+i
```

Ý nghĩa đoạn lệnh trên: .....

.....

Kết quả `s` cuối cùng: .....

Ví dụ 3

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

```
#THUẬT TOÁN Tong_1a
a = int(input('a = '))
S = 1/a
for N in range(1,101):
     S = S+1/(a+N)
print('Tong can tim là: %8.4f'%S)

#THUẬT TOÁN Tong_1b
a = int(input('a = '))
S = 1/a
for N in range(100,0,-1):
     S = S+1/(a+N)
print('Tong can tim là: %8.4f'%S)
```

Ví dụ 4. 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*.

```
#TỔNG CÁC SỐ CHIA HẾT CHO 3 HOẶC 5 TỪ M ĐẾN N
print('Nhập số M nhỏ hơn số N')
M=int(input('M= '))
N=int(input('N= '))
tong = 0
for i in range(M,N+1):
    if (i%3 ==0) or (i%5== 0):
     tong = tong + i                #hoặc tong +=i
print('Kết quả: ',tong)
```

3. Lặp với số lần chưa biết trước và câu lệnh 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:

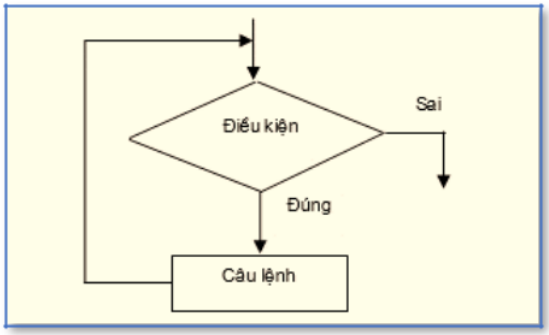
```
while <điều kiện>:
     <câu lệnh>
```

trong đó

điều kiện là biểu thức quan hệ hoặc logic có giá trị *True/False*;

câu lệnh là một câu lệnh của Python

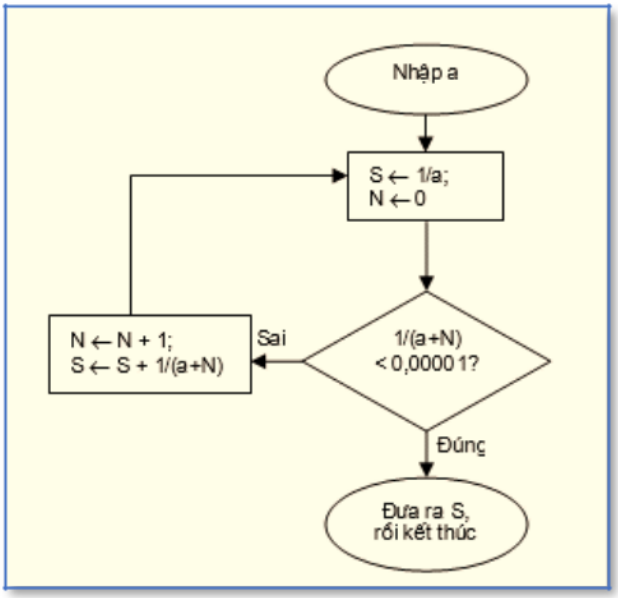
Việc thực hiện lệnh *while* được thể hiện trên sơ đồ:



Hình 3.3 – Sơ đồ lặp với số lần chưa biết trước

Ví dụ 1

Thuật toán *Tong_2* dạng sơ đồ khối:



Hình 3.4 – Sơ đồ khối của thuật tón Tong_2

Sau đây là chương trình cài đặt thuật toán *Tong_2*

```
#THUẬT TOÁN Tong_2
a = int(input('a= '))
S = 1/a
N = 0
while (1/(a+N) >=0.0001):
     N = N+1                    #hoặc N+=1
     S=S+1/(a+N)                #hoặc S+=1/(a+N)
print('Tong S la: %8.4f' %S)
```

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 liệt kê

Bước 1: Nhập M, N;

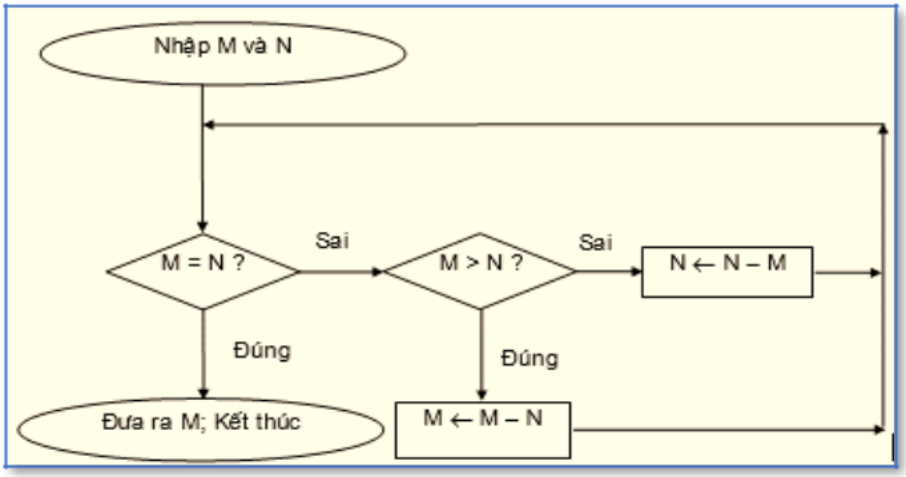
Bước 2: Nếu M = N thì lấy giá trị chung này làm UCLN rồi chuyển đến bước 5;

Bước 3: Nếu M > N thì M ← M - N rồi quay lại bước 2;

Bước 4: N ← N - M rồi quay lại bước 2;

Bước 5: Đưa ra kết quả UCLN rồi kết thúc.

Thuật toán sơ đồ khối



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

Chương trình

```
#TÌM ƯỚC CHUNG LỚN NHẤT
M,N = map(int,input('M,N= ').split())
while M!=N:
     if M>N:
          M=M-N       #hoặc M-=N
     else:
          N=N-M       #hoặc N-=M
print('UCLN= ',M)
```

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:

```
#BỘ SỐ PITAGO
a=int(input('a= '))                                          #1
b=int(input('b= '))                                          #2
c=int(input('c= '))                                          #3
if (a*a==b*b+c*c) or (b*b==a*a+c*c) or (c*c==a*a+b*b):        #4
    print(a,b,c,' là bộ số Pitago')                          #5
else:                                                        #6
    print(a,b,c,' không là bộ số Pitago')                    #7
```

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) Thay dãy câu lệnh #1, #2, #3 bằng 1 câu lệnh

CÂU HỎI VÀ BÀI TẬP

1. Có thể dùng câu lệnh **while** để thay cho câu lệnh **for** được không? Nếu được, hãy thực hiện điều đó với chương trình tính tổng các số chia hết cho 3 hoặc 5 từ M đến N.

2. Viết câu lệnh **if** tính:

a) $$z = \begin{cases} x^2 + y^2 & \text{nếu } x^2 + y^2 \le 1. \\ x + y & \text{nếu } x^2 + y^2 > 1 \text{ và } y \ge x. \\ 0,5 & \text{nếu } x^2 + y^2 > 1 \text{ và } y < x. \end{cases}$$

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

3. Lập trình để 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?

4. Nhập từ bàn phím tuổi của cha và con (tuổi của cha hơn tuổi con ít nhất là 25). Đưa ra màn hình bao nhiêu năm nữa thì tuổi cha gấp đôi tuổi con.

5. 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.

6. Đưa ra màn hình như bên dưới:

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

7. Đưa ra màn hình như bên dưới:

```
1*
2**
3***
4****
5*****
6******
```

8. Đưa ra màn hình tam giác cân (số tầng tùy ý)

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

9. Tính tổng các số nguyên từ 1 đến n với n nhập vào từ bàn phím.

10. Nhập N từ bàn phím (N>0), tính giai thừa của n (n!).

11. Tính tổng các số lẻ trong phạm vi từ 1 đến 20.

12. Tính tổng các số chẵn trong phạm vi từ 1 đến n.

13. Nhập 1 số A từ bàn phím, Kiểm tra A có phải là **số nguyên tố** hay không?

14. Nhập M từ bàn phím (M>0), Tính giá trị của biểu thức **Tong = $1^2 + 2^2 + 3^2 + ... + M^2$**

15. Một người gửi tiết kiệm 100 đồng. Hỏi sau 12 tháng anh ta sẽ thu được số tiền là bao nhiêu? Biết, lãi xuất mỗi tháng của Ngân hàng là 0.15%.

16. Một giáo viên vào lớp và phát kẹo cho các em học sinh theo qui tắc sau: em đầu tiên được nhận 1 viên kẹo, em tiếp theo nhận số kẹo nhiều hơn em trước đó 3 viên. Hỏi nếu lớp học có 20 học sinh thì tổng số kẹo phải có để Giáo viên phát đủ cho cả lớp.