**[Biến](https://autohotkey.com/docs/Variables.htm" \l "Intro)**

**A/ Phạm vi biến và khai báo**

Tương tự AutoIt, thường thì không cần khai báo (Local, Global …) trừ những trường hợp riêng hoặc bạn muốn khai báo theo thói quen.

**B/ Tên biến**

Tên biến **không** cần bắt đầu bằng **$** như bên AutoIt.

Tên biến không phân biệt chữ hoa / chữ thường (ví dụ: *CurrentDate* cũng giống như *currentdate* ).

Tên biến có thể dài tối đa 253 ký tự và bao gồm chữ cái, số và các dấu sau đây: **# \_ @ $.**

Tuy nhiên ta nên đặt tên biến theo các chuẩn sau để chương trình vận hành tốt nhất:

* Ta chỉ nên dùng chữ cái, số và dấu \_ để hợp với quy cách chung, **đừng đặt tên biến chứa #@$**
* **Không nên đặt tên biến** **chỉ toàn là số**. Những tên toàn số như vậy không thể được sử dụng trong các [biểu thức](https://autohotkey.com/docs/Variables.htm#Expressions) vì chúng sẽ được xem dưới dạng số chứ không phải là các biến.
* Vì các từ ***AND*** , ***OR*** , và ***NOT*** được sử dụng như các [toán tử](https://autohotkey.com/docs/Variables.htm#Operators) trong các biểu thức nên cũng **đừng đặt tên biến là and, or, not**.
* **Không đặt tên biến** **giống như tên hàm** (cả tên hàm Built-in và hàm tự dựng).
* **Không đặt tên biến trùng với tên biến môi trường (**Environment variables**):** Các biến môi trường được duy trì bởi hệ điều hành. Bạn có thể thấy một danh sách tên các biến môi trường bằng cách Mở Command Promt, gõ lệnh SET và nhấn Enter.
* **Không đặt tên biến trùng với các tên biến Built-in**. Biến Built-in là gì ? Nó giống như Macro bên AutoIt vậy, ví dụ:

AutoIt: @error => AutoHotKey: [ErrorLevel](https://autohotkey.com/docs/misc/ErrorLevel.htm)

AutoIt: @HOUR => AutoHotKey: A\_Hour

AutoIt: @ ScriptDir => AutoHotKey: A\_ScriptDir

\* Full danh sách biến Built-in của AutoHotKey: <https://autohotkey.com/docs/Variables.htm#Special_Characters>

\* Nói chung không cần nhớ nhiều, chỉ cần nhớ là **không đặt tên biến là** **Clipboard**, **ClipboardAll**, **ErrorLevel** hoặc tên biến **bắt đầu với** **A\_**

**C/ Lưu trữ giá trị vào biến**

Để lưu một chuỗi hoặc số trong một biến, có hai phương pháp:

1. **Phương pháp truyền thống**:

+ Sử dụng dấu **=** để gán.

+ Nếu gán chuỗi thì chuỗi đó không cần đặt giữa 2 dấu nháy kép. Ví dụ: MyString = Đây là chuỗi.

+ Nếu gán biến thì tên biến đặt trong 2 dấu **%**. Ví dụ: CopyOfVar = %Var%

1. **Phương pháp biểu thức**:

+ Sử dụng dấu **:=** để gán.

+ Nếu gán chuỗi thì chuỗi đó phải đặt giữa 2 dấu nháy kép **(không thể dùng dấu nháy đơn như bên AutoIt)**. Ví dụ: MyString := "Đây là chuỗi"

+ Nếu gán biến thì tên biến không cần đặt trong 2 dấu % như kiểu truyền thống. Ví dụ: CopyOfVar = Var

**\*** Phương pháp biểu thức được nhiều người ưa thích do tính rõ ràng của nó so với kiểu truyền thống, và bởi vì nó hỗ trợ [cú pháp biểu thức](https://autohotkey.com/docs/Variables.htm#Expressions) gần giống với nhiều ngôn ngữ khác (cả AutoIt). Vì vậy, ta **chỉ nên sử dụng phương pháp Biểu thức** để làm việc với Biến, trừ phi thấy lười lười thì dùng phương pháp truyền thống để gán nhanh số hoặc chuỗi chữ cho Biến thôi.

1. **Một số lưu ý khi gán giá trị cho biến:**

+ Gán 1 con số: Cả 2 phương pháp đều sử dụng như nhau: Var = 123 hoặc Var := 123

+ Gán giá trị rỗng:

Kiểu truyền thống: Var =

Kiểu biểu thức: Var := ""

+ Gán True False: **Phải** sử dụng kiểu Biểu thức để gán: Var := True , Var := False

**D/ Truy xuất nội dung của biến**

Giống như hai phương pháp lưu trữ giá trị vào biến, truy xuất nội dung biến cũng có hai phương pháp để truy xuất là truyền thống và biểu thức:

* **Phương pháp truyền thống** : Yêu cầu mỗi tên biến được đặt trong 2 dấu hiệu % để lấy nội dung của nó. Ví dụ:

Var = 123 ; hoặc Var := 123

MsgBox Giá trị trong biến có tên Var là %Var%

* **Phương pháp biểu thức** : tên biến không cần đặt trong dấu %, nhưng các chuỗi chữ phải trong dấu nháy kép **(không thể dùng dấu nháy đơn như bên AutoIt).** Ví dụ:

Var = 123 ; hoặc Var := 123

MsgBox **%** "Giá trị trong biến có tên Var là" **.** Var

**Lưu ý:**

1. Ở phương pháp biểu thức , để nối 2 chuỗi hoặc chuỗi với biến thì dùng dấu **. (dấu chấm)** (còn trong AutoIt thì dùng dấu **&**)
2. Ở phương pháp biểu thức , nếu trong chuỗi có dấu **"** thì ta phải double dấu nháy kép đó lên, ví dụ: MyString := "trong chuỗi có dấu **""** thì phải double nó lên"
3. Trong dòng MsgBox ở ví dụ về phương pháp biểu thức trên, **một dấu phần trăm và khoảng trống** được sử dụng ngay đằng trước tham số của lệnh/hàm để thay đổi tham số từ chế độ truyền thống sang biểu thức.

**\* Điều này là cần thiết vì phương pháp truyền thống** được sử dụng theo **mặc định** bởi tất cả các lệnh/hàm**,** trừ những trường hợp riêng có những lệnh/hàm truy xuất biến theo kiểu truyền thống hay biểu thức đều được, không cần thêm **%** (thường là những lệnh/hàm chỉ yêu cầu cung cấp 1 tham số), ví dụ như lệnh Sleep ta để biến kiểu nào cũng được:

MillisecondsToWait = 1000

Sleep MillisecondsToWait

Sleep %MillisecondsToWait%

Sleep % MillisecondsToWait

**\*** Ở đây mình xin nhắc lại một điều đã nói ở Bài 1 (Sử dụng MsgBox) là: **sau tên Lệnh/hàm, mình để dấu phẩy (,) hay không để đều được**:

Sleep MillisecondsToWait

Sleep**,** MillisecondsToWait

1. Nếu **biến chưa được định nghĩa/gán giá trị trước khi được truy xuất** thì AutoHotKey sẽ tự động gán cho biến đó là rỗng (chứ không báo lỗi như bên AutoIt)

**E/ Dung lượng và bộ nhớ biến**

* Mỗi biến có thể chứa tới 64 Mb text theo mặc định. Giới hạn này có thể được điều chỉnh với lệnh [#MaxMem](https://autohotkey.com/docs/commands/_MaxMem.htm), chứa tối đa lên đến **4095** Mb bằng cách thêm vào đầu code

#MaxMem Dung lượng Mb

Ví dụ: #MaxMem 256

* Khi một biến chứa một chuỗi mới dài hơn nội dung hiện tại của nó, bộ nhớ hệ thống bổ sung sẽ được phân bổ tự động.
* Bộ nhớ chiếm bởi một biến lớn có thể được giải phóng bằng cách đặt nó bằng rỗng, ví dụ: var := ""
* Không có giới hạn đối với số lượng biến mà một kịch bản có thể tạo ra. Chương trình này được thiết kế để hỗ trợ ít nhất vài triệu biến mà không làm giảm hiệu suất đáng kể.
* Con số tối đa mà biến có thể lưu là từ - 0x7FFFFFFFFFFFFFFF đến 0x7FFFFFFFFFFFFFFF.

**[Biểu thức So sánh – Mệnh đề Điều kiện](https://autohotkey.com/docs/Variables.htm" \l "Expressions)**

- **Lưu trữ kết quả của một biểu thức** : Để gán một kết quả cho một biến, sử dụng [**:=**](https://autohotkey.com/docs/commands/SetExpression.htm)

Điều này đã nói rõ ở phần BIẾN, mục Lưu trữ giá trị vào biến, phần Phương pháp Biểu thức.

Ví dụ: NetPrice := Price \* (1 - Discount / 100)

- Biểu thức So sánh thì thường đi kèm với mệnh đề điều kiện IF, cái này không cần bàn cãi nên bài viết bên dưới sẽ là sự pha trộn giữa So Sánh và IF. **Vì IF có khá nhiều cái để nói nên mình chỉ lướt nhanh qua trong bài viết này và sẽ đi kỹ hơn ở các bài viết sau**.

- Tên biến trong một biểu thức **bên trái dấu so sánh** không được đặt giữa 2 dấu % , ngoại trừ 1 số trường hợp như biến Mảng-Pseudo ( Array%i% ). Ví dụ:

FoundColor = Blue

SearchColor = Red

If FoundColor <> %SearchColor% ; => Đúng cú pháp

If %FoundColor% <> %SearchColor% ; => Sai cú pháp

- Nếu là Lệnh/hàm thì ta dùng dấu **%** đặt sau Tên lệnh/hàm để chuyển đổi giữa 2 kiểu khai báo truyền thống và biểu thức (đã nhắc nhiều ở phần BIẾN, ví dụ: MsgBox % "Chuỗi là " . var ). Còn trong mệnh đề IF thì để chuyển đổi giữa 2 phương pháp ta thêm vào **cặp dấu ngoặc đơn**.

- Ví dụ:

Kiểu truyền thống:

FoundColor = Red

If FoundColor <> Blue

MsgBox Sai màu

Kiểu biểu thức:

FoundColor = Red

If **(** FoundColor <> **"**Blue**"** **)**

MsgBox Sai màu

- Theo lời khuyên từ Tài liệu AutoHotKey thì **nên sử dụng kiểu biểu thức trong mệnh đề IF** để tránh việc chồng chéo giữa tên biến và chuỗi chữ dẫn đến khó kiểm tra, dễ gây lỗi.

- Bất kỳ **biểu thức không hợp lệ** như (x +\* 3) **sẽ trả về một chuỗi rỗng**.

**[Một số lưu ý khác khi sử dụng biến và biểu thức](https://autohotkey.com/docs/Variables.htm" \l "Operators)**

- Giá trị phi số tham gia vào một phép toán **không** phải là **0**. Thay vào đó, nó được coi là một lỗi, làm cho giá trị của biểu thức là **một chuỗi rỗng**. Ví dụ, nếu biến X là trống, biểu thức X + 1 cho ra một giá trị trống thay vì 1.

- Vì lý do lịch sử, các chuỗi tuy chỉ chứa toàn số như "123", "0x123", "0.123" luôn luôn được coi là **không phải số** khi được sử dụng trực tiếp trong một biểu thức, trừ phi chúng đã được gán vào một biến và biến đó nằm trong biểu thức hoặc được xử lý qua các hàm trung gian như [Round( )](https://autohotkey.com/docs/Functions.htm#Round) chẳng hạn. Do đó các tập lệnh nên tránh sử dụng dấu nháy kép xung quanh chữ số trừ phi chúng thật sự là một chuỗi chữ.

- Ví dụ:

MsgBox % "123" + 1 ;=> 1231

var := "123"

MsgBox % var + 1 ;=> 124

**[Toán tử Biểu thức](https://autohotkey.com/docs/Variables.htm" \l "Expression_Operators_in_descending_precedence_order)**

|  |  |
| --- | --- |
| **++**  **--** | **Tăng/giảm trước, Tăng/giảm sau**.  - Cộng hoặc trừ đi 1 đơn vị của một biến. Toán tử có thể xuất hiện trước hoặc sau tên của biến.  Ví dụ, Var := ++X hoặc Var := X++.  - Có gì khác nhau giữa 2 thứ tự xuất hiện ? Trong tài liệu tiếng Anh có nói hơi khó hiểu (đối với người chưa từng tiếp cận với ngôn ngữ C ). Mình nói đơn giản theo cách hiểu như sau:  Ví dụ:  X := 1 , Var := **++**X  MsgBox Giá trị của X là %X% và Var là %Var%  => Chạy code ta sẽ thấy X lúc này sẽ có giá trị là 2, và Var = X cũng = 2 luôn  X := 1 , Var := X**++**  MsgBox Giá trị của X là %X% và Var là %Var%  => Chạy code ta sẽ thấy X lúc này sẽ có giá trị là 2, còn Var = X trước khi tăng tức là = 1  => Tóm lại:  Đặt **++** **trước** tên biến\_1 thì sẽ tăng biến\_1 lên 1 đơn vị, biến\_2 được chỉ định lưu giá trị biến\_1 sẽ lưu giá trị hiện tại của biến\_1.  Đặt **++** **sau** tên biến\_1 thì sẽ tăng biến\_1 lên 1 đơn vị, biến\_2 được chỉ định lưu giá trị biến\_1 sẽ lưu giá trị hiện tại của biến\_1 trừ đi 1 (hay nói cách khác: lưu giá trị trước đó của biến\_1)  Tương tự, **--** là trừ đi 1 đơn vị của một biến. Cũng có thể đặt **--** ở trước hoặc sau biến và có quy luật gán giá trị như phép **++** |
| **\*\*** | **Luỹ thừa**.  - Khác với AutoIt dùng dấu **^** để thực hiện phép tính luỹ thừa thì AutoHotKey dùng **\*\***  - Lưu ý duy nhất cho phép tính này là ví dụ ta có **-2\*\*2** thì nó sẽ = **-(2\*\*2)** = **-4** chứ không phải bằng 4. Vì vậy khi thực hiện phép luỹ thừa thì nên bỏ giá trị cơ số trong đóng mở ngoặc đơn để chắc chắn cho code chạy chính xác. |
| **+**  **-**  **\*  /** | **Các phép toán cộng, trừ, nhân, chia**.  Nếu thực hiện biểu thức với cùng 1 biến duy nhất, ta có thể viết tắt các biểu thức này như sau:  X := X + 2 => X +=2  X := X – 2 => X -=2  X := X \* 2 => X \*= 2  X := X / 2 => X /= 2  **\*** Phép chia cho số 0 sẽ ra rỗng.  **\*** Phép chia sẽ trả về một con số có 6 số thập phân, ví dụ: 2 / 2 sẽ không bằng 1 mà bằng 1.000000 |
| **//** | **Chia và lấy phần nguyên.**  - Ví dụ: 5 // 2 = 2.  - Ta cũng có thể viết tắt khi thực hiện 1 biểu thức cùng 1 biến duy nhất: X := X // 2 => X //= 2  \* Nếu chia và lấy phần dư thì phải dùng hàm Mod( ) |
| **<<  >>** | **Dịch Bit trái (<<)** và **phải (>>)**  Mình không rành vụ dịch Bit này lắm, mà code cũng ít khi sử dụng đến nên mình cho qua, ai thích thì tự tham khảo trên Google về dịch Bit nhé. |
| **~**  **&  ^  |** | **Các phép toán trên Bit: NOT (~), AND (&), XOR (^), OR (|)**  - Lưu ý nhắc lại để không nhầm lẫn với các phép toán này:  + Trong AutoIt để nối chuỗi thì dùng dấu **&** , còn trong AutoHotKey thì dùng dấu chấm (**.**)  + Trong AutoIt phép luỹ thừa thì dùng dấu **^** , còn trong AutoHotKey thì dùng dấu **\*\***  - **3 phép toán AND, OR, XOR** cũng có thể viết tắt khi biểu thức có cùng 1 biến duy nhất:  X = X & 2 => X &= 2  X = X ^ 2 => X ^= 2  X = X | 2 => X |= 2  - Ví dụ về các phép toán trên Bit:  2d = 00000010b suy ra !2d = 11111101b = 4294967293d (d, b là biểu diễn dec và bin)  3d & 2d = 00000011b & 00000010b = 00000010b = 2d  Mấy cái này ai có học về CNTT đều biết, còn ai không biết thì khỏi tìm hiểu cũng được vì cũng ít áp dụng, thích thì đọc thêm tài liệu ^^ |
| **!**  **&&**  **||** | **Phép toán Logic: NOT (!), AND (&&) và OR (||)**  NOT : Đảo ngược lại  AND : Các phép toán trước và sau AND trong biểu thức đều phải đúng  OR : Các phép toán trước và sau OR trong biểu thức chỉ cần đúng 1 phép toán (trước hoặc sau)  - Ví dụ:  X > 3 && y < 1 => x phải lớn hơn 3 và y phải bé hơn 1 thì biểu thức mới trả về 1 (True)  X > 3 || y < 1 => chỉ cần x lớn hơn 3 hoặc y bé hơn 1 là biểu thức đã trả về 1 (True) rồi.  - Dùng dấu ngoặc đơn để sắp xếp thứ tự ưu tiên khi biểu thức có nhiều phép toán Logic. |
| **> <  >=**  **<=**  **<>**  **!=** | **Lớn hơn (>)** , **Bé hơn (<)** , **Lớn hơn hoặc bằng (>=)** , **Bé hơn hoặc bằng (<=)** , **Khác ( <> hoặc != đều được)**  Cái này thì đơn giản rồi nên không cần giải thích nhiều. |
| **=  ==** | **Cả 2 đều là so sánh Bằng.**  Nhiều bạn không hiểu đều là so sánh bằng thì người ta phân biệt 2 thằng để làm gì. Nói nôm na là khi so sánh 2 chuỗi chữ với nhau, nếu dùng dấu = thì so sánh không phân biệt hoa thường, còn == thì phân biệt hoa thường. Ví dụ:  "chuẩn" = "Chuẩn" => Kết quả là 1 (True) => Bằng, tức là 2 chuỗi giống nhau  "chuẩn" == "Chuẩn" => Kết quả là 0 (False) => Không bằng, tức là 2 chuỗi được coi như khác nhau (do khác chữ C có in hoa và không in hoa) |
| **MỘT SỐ CÁCH RÚT GỌN BIỂU THỨC CẦN BIẾT** | |
| **A ? B : C** | **Ternary**.  Toán tử này là một sự thay thế viết tắt cho câu lệnh [if-else](https://autohotkey.com/docs/commands/IfExpression.htm) . Nếu biểu thức A đúng thì trả về B, nếu sai thì trả về C. Ví dụ:  If var > 3  X = 1  Else  X = 2  Viết gọn lại là:  X := var > 3 ? 1 : 2 |
| , | **Dấu phẩy đa tuyên bố**.  - Dấu phẩy có thể được sử dụng để viết nhiều dòng biểu thức đơn giản thành một dòng.  - Lưu ý: Nếu dòng gán giá trị cho biến đang là kiểu truyền thống phải chuyển về kiểu biểu thức.  - Ví dụ:  X **=** 2  Y = 3  Z := X / 2  T := Round(X / 3, 2)  Có thể viết gọn lại là:  X **:=** 2 , Y **:=** 3 , Z := X / 2, T := Round(X / 3, 2)  Theo tài liệu thì khi chuyển từ nhiều hàng sang 1 hàng, code có thể hoạt động nhanh hơn tới 35% |
| **:=** | **Gán nhiều biến cùng giá trị.**  Chỉ áp dụng khi gán số hoặc một lệnh/hàm trả về số. Không áp dụng cho chuỗi.  Ví dụ:  X **=** 2  Y = 2  Có thể viết lại là:  X :**=** Y := 2 |
| **~=** | **RegExp.**  - Viết tắt cho [RegExMatch](https://autohotkey.com/docs/commands/RegExMatch.htm). Lượng kiến thức về Regexp là rất lớn nên bạn tự tìm hiểu về nó nếu chưa biết cách sử dụng nhé.  - Cấu trúc: "Chuỗi chứa nội dung cần tìm" **~=** "Patern RegExp"  \* Lưu ý: Cấu trúc ~= **chỉ trả về vị trí đầu tiên của chuỗi tìm thấy** chứ không phải trả về chuỗi tìm thấy.  - Ví dụ: "abc123" ~= "\d" sẽ trả về **4**  Var := ("abc123" ~= "\d")  MsgBox % Var |
| **.=** | **Nối chuỗi tắt.**  - Khác AutoIt dùng & để nối chuỗi thì trong AutoHotKey, dấu chấm được dùng để nối chuỗi. Ví dụ:Var := "abc" **.** "123"  - Ta có thể viết tắt khi biểu thức sử dụng một biến duy nhất để nối vào chính biến đó như sau:  A := "abc"  A := A . "123"  Có thể viết lại thành:  A := "abc"  A **.=** "123" |

**Nói sơ trước về Mảng trong AutoHotKey**

- Cách khai báo. Ví dụ: A := ["one", "two", 3]

- Khác với AutoIt mảng bắt đầu từ phần tử 0, còn AutoHotKey mảng bắt đầu bằng phần tử 1.

- Ví dụ:

A := ["one", "two", 3]

MsgBox % A[1]