1. The Power of Variable Names

Code Complete

# 11.0 Python Keywords

<https://docs.python.org/3/reference/lexical_analysis.html#keywords>

### 2.3.1. Keywords

The following identifiers are used as reserved words, or *keywords* of the language, and cannot be used as ordinary identifiers. They must be spelled exactly as written here:

False class finally is return  
None continue for lambda try  
True def from nonlocal while  
and del global not with  
as elif if or yield  
assert else import pass  
break except in raise

## 11.1 Considerations in Choosing Good Names

● No puede darle un nombre a una variable de la misma manera que le da un nombre a un perro, porque es lindo o tiene un buen sonido. A diferencia del perro y su nombre, que son entidades diferentes, una variable y el nombre de una variable son esencialmente lo mismo.

● Ejemplo de Java de nombres de variables deficientes

Example 11-1.

x = x - xx;

xxx = fido + SalesTax( fido );

x = x + LateFee( x1, x ) + xxx;

x = x + Interest( x1, x );

What's happening in this piece of code? What do x1, xx, and xxx mean? What does fido mean?

Example 11-2. Java Example of Good Variable

Names balance = balance - lastPayment;

monthlyTotal = newPurchases + SalesTax( newPurchases );

balance = balance + LateFee( customerID, balance ) +

monthlyTotal;

balance = balance + Interest( customerID, balance );

In view of the contrast between these two pieces of code, a good variable name is readable, memorable, and appropriate.

**The Most Important Naming Consideration**

● La consideración más importante al nombrar una variable es que el nombre describa de manera completa y precisa la entidad que representa la variable. Una técnica eficaz para encontrar un buen nombre es expresar con palabras lo que representa la variable. A menudo, esa declaración en sí misma es el mejor nombre de variable. Es fácil de leer porque no contiene abreviaturas crípticas y no es ambiguo. Debido a que es una descripción completa de la entidad, no se confundirá con otra cosa. Y es fácil de recordar

**Problem Orientation**

● Un buen nombre mnemónico generalmente habla del problema más que de la solución. Un buen nombre tiende a expresar el qué más que el cómo. En general, si un nombre se refiere a algún aspecto de la informática más que al problema, es un cómo más que un qué. Evite ese nombre en favor de un nombre que se refiera al problema en sí. Un registro de datos de empleados podría denominarse inputRec o employeeData. inputRec es un término informático que se refiere a la computación de ideas: entrada y registro. employeeData se refiere al dominio del problema más que al universo informático.

**Optimum Name**

● Longitud: la longitud óptima de un nombre parece estar entre las longitudes de x y maximumNumberOfPointsInModernOlympics. Los nombres que son demasiado cortos no transmiten suficiente significado.

● Gorla, Benander y Benander descubrieron que el esfuerzo requerido para depurar un programa se minimizaba cuando las variables tenían nombres con un promedio de 10 a 16 caracteres (1990). Los programas con nombres con un promedio de 8 a 20 caracteres eran casi tan fáciles de depurar.

Table 11-2. Variable Names That Are Too Long, Too Short, or Just Right

Too long:

numberOfPeopleOnTheUsOlympicTeam

numberOfSeatsInTheStadium

maximumNumberOfPointsInModernOlympics

Too short:

n, np, ntm

n, ns, nsisd

m, mp, max, points

Just right:

numTeamMembers, teamMemberCount

numSeatsInStadium, seatCount

teamPointsMax, pointsRecord

El efecto del alcance en los nombres de variables

* ● El alcance de referencia cruzada se analiza con más detalle en Alcance. ¿Los nombres cortos de variables siempre son malos? No, no siempre. Cuando le da a una variable un nombre corto como i, la longitud en sí dice algo sobre la variable, es decir, que la variable es un valor temporal con un alcance de operación limitado.
* ● Un estudio de W. J. Hansen encontró que los nombres más largos son mejores para variables de uso poco frecuente o variables globales y los nombres más cortos son mejores para variables locales o variables de ciclo (Shneiderman 1980).
* ● Si tiene variables que están en el espacio de nombres global (constantes con nombre, nombres de clases, etc.), considere si necesita adoptar una convención para particionar el espacio de nombres global y evitar conflictos de nombres.
* Calificadores de valor calculado en nombres de variables
* ● Muchos programas tienen variables que contienen valores calculados: totales, promedios, máximos, etc. Si modifica un nombre con un calificador como Total, Suma, Promedio, Máximo, Mínimo, Registro, Cadena o Puntero, coloque el modificador al final del nombre.
* ● Esta práctica ofrece varias ventajas. Primero, la parte más significativa del nombre de la variable, la parte que le da a la variable la mayor parte de su significado, está al principio, por lo que es más prominente y se lee primero. En segundo lugar, al establecer esta convención, evita la confusión que podría crear si usara tanto totalRevenue como incomeTotal en el mismo programa.
* ● Una excepción a la regla de que los valores calculados van al final del nombre es la posición habitual del calificador Num. Colocado al comienzo de un nombre de variable, Num se refiere a un total: numCustomers es el número total de clientes. Colocado al final del nombre de la variable, Num se refiere a un índice: customerNum es el número del cliente actual.
* ● Pero, debido a que el uso de Num crea confusión con tanta frecuencia, probablemente sea mejor evitar todo el problema utilizando Count o Total para referirse a un número total de clientes e Index para referirse a un cliente específico.
* Opuestos comunes en nombres de variables
* Referencia cruzada
* Para obtener una lista similar de opuestos en los nombres de las rutinas, consulte "Usar opuestos con precisión" en Buenos nombres de rutina.
* ● Utilice los opuestos con precisión. El uso de convenciones de nomenclatura para los opuestos ayuda a la coherencia, lo que mejora la legibilidad. Los pares como begin / end son fáciles de entender y recordar. Los pares que se apartan de los opuestos del lenguaje común tienden a ser difíciles de recordar y, por lo tanto, son confusos.

## 11.2 Naming Specific Types of Data

Además de las consideraciones generales al nombrar datos, surgen consideraciones especiales al nombrar tipos específicos de datos. Esta sección describe consideraciones específicas para variables de ciclo, variables de estado, variables temporales, variables booleanas, tipos enumerados y constantes con nombre.

**Naming Loop Indexes**

Referencia cruzada

Para obtener detalles sobre los bucles, consulte el Capítulo 16.

● Han surgido pautas para nombrar variables en bucles porque los bucles son una característica muy común de la programación de computadoras. Los nombres i, j y k son habituales:

Example 11-4. Java Example of a Simple Loop Variable Name

for ( i = firstItem; i < lastItem; i++ )

{ data[ i ] = 0; }

● Si una variable se va a utilizar fuera del ciclo, se le debe dar un nombre más significativo que i, j o k. Por ejemplo, si está leyendo registros de un archivo y necesita recordar cuántos registros ha leído, un nombre como recordCount sería apropiado:

Example 11-5. Java Example of a Good Descriptive Loop Variable Name

recordCount = 0; while ( moreScores() )

{ score[ recordCount ] = GetNextScore(); recordCount++; }

● Debido a que el código se cambia, expande y copia con tanta frecuencia en otros programas, muchos programadores experimentados evitan los nombres como yo por completo.

Example 11-6. Java Example of Good Loop Names in a Nested Loop

for (teamIndex = 0; teamIndex < teamCount; teamIndex++)

{ for (eventIndex = 0; eventIndex < eventCount[teamIndex];

eventIndex++)

{ score[teamIndex][eventIndex] = 0;}

}

● Los nombres elegidos cuidadosamente para las variables de índice de bucle evitan el problema común de la interferencia de índices: decir i cuando te refieres a j y j cuando te refieres a i. También hacen que los accesos a la matriz sean más claros:

score [teamIndex] [eventIndex] es más informativo que score [i] [j].

● Si tiene que usar i, j y k, no los use para nada más que índices de bucle para bucles simples; la convención está demasiado bien establecida y romperla para usarlos de otras formas es confuso.

**Naming Status Variables**

Las variables de estado describen el estado de su programa.

Aquí hay una guía de nomenclatura: piense en un nombre mejor que una bandera para las variables de estado. Es mejor pensar en las banderas como variables de estado. Una bandera nunca debe tener una bandera en su nombre porque eso no le da ninguna pista sobre lo que hace la bandera. Para mayor claridad, a los indicadores se les deben asignar valores y sus valores se deben probar con tipos enumerados, constantes con nombre o variables globales que actúan como constantes con nombre.

Example 11-8.

C++ Examples of Better Use of Status Variables

if ( dataReady ) ...

if ( characterType & PRINTABLE\_CHAR ) ...

if ( reportType == ReportType\_Annual ) ...

if ( recalcNeeded == True ) ...

dataReady = true;

characterType = CONTROL\_CHARACTER;

reportType = ReportType\_Annual;

recalcNeeded = false;

**Naming Temporary Variables**

Las variables temporales se utilizan para contener los resultados intermedios de los cálculos, como marcadores de posición temporales y para mantener los valores de mantenimiento.

Por lo general, se llaman temp, x, o algún otro nombre vago y no descriptivo. En general, las variables temporales son una señal de que el programador aún no comprende completamente el problema.

Example 11-11. C++

Example with a "Temporary" Variable Name Replaced with a Real Variable

// Compute roots of a quadratic equation.

// This assumes that (b^2-4\*a\*c) is positive.

discriminant = sqrt( b^2 - 4\*a\*c );

root[0] = ( -b + discriminant ) / ( 2 \* a );

root[1] = ( -b - discriminant ) / ( 2 \* a );

This is essentially the same code, but it's improved with the use of an accurate, descriptive variable name.

**Naming Boolean Variables**

● Hecho

Úselo para indicar si se hizo algo. La variable puede indicar si se realiza un bucle o se realiza alguna otra operación. Establezca done en falso antes de que se haga algo, y establezcalo en verdadero cuando se complete algo. error Utilice error para indicar que se ha producido un error.

● encontrado

Utilice encontrado para indicar si se ha encontrado un valor.

● éxito o bien

Utilice éxito u ok para indicar si una operación se ha realizado correctamente.

● Si puede, reemplace el éxito por un nombre más específico que describa con precisión lo que significa tener éxito.

● Dé nombres de variables booleanas que impliquen verdadero o falso. Nombres como done y success son buenos nombres booleanos porque el estado es verdadero o falso; algo está hecho o no está; es un éxito o no lo es. Nombres como status y sourceFile, por otro lado, son nombres booleanos deficientes porque no son obviamente verdaderos o falsos.

● A algunos programadores les gusta poner Is delante de sus nombres booleanos. Entonces el nombre de la variable se convierte en una pregunta: ¿está hecho? isError? ¿es encontrado? isProcessingComplete? Responder la pregunta con verdadero o falso proporciona el valor de la variable. Un beneficio de este enfoque es que no funcionará con nombres vagos: isStatus? no tiene ningún sentido. Un inconveniente es que hace que las expresiones lógicas simples sean menos legibles: if (isFound) es un poco menos legible que if (found).

● Utilice nombres de variables booleanos positivos. Los nombres negativos como notFound, notdone y notSuccessful son difíciles de leer cuando se niegan:

**Naming Enumerated Types**

Referencia cruzada

Para obtener detalles sobre el uso de tipos enumerados, consulte Tipos enumerados.

Cuando utiliza un tipo enumerado, puede asegurarse de que está claro que los miembros del tipo pertenecen todos al mismo grupo mediante el uso de un prefijo de grupo, como Color\_, Planet\_ o Month\_.

Example 11-12.

Visual Basic Example of Using a Prefix Naming Convention for Enumerated Types

Public Enum Color

Color\_Red

Color\_Green

Color\_Blue

End Enum

**Naming Constants**

Referencia cruzada

Para obtener detalles sobre el uso de constantes con nombre, consulte Constantes con nombre.

Al nombrar constantes, nombre la entidad abstracta que representa la constante en lugar del número al que se refiere la constante. CINCO es un mal nombre para una constante (independientemente de si el valor que representa es 5.0). CYCLES\_NEEDED es un buen nombre. CYCLES\_NEEDED puede ser igual a 5.0 o 6.0. CINCO = 6.0 sería ridículo.

## 11.3 The Power of Naming Conventions

Algunos programadores se resisten a los estándares y convenciones, y con razón. Algunas normas y convenciones son rígidas e ineficaces, destructivas para la creatividad y la calidad del programa. Esto es lamentable, ya que los estándares eficaces son algunas de las herramientas más poderosas a su disposición. En esta sección se explica por qué, cuándo y cómo debe crear sus propios estándares para nombrar variables.

**Why Have Conventions?**

● Las convenciones ofrecen varios beneficios específicos: le permiten dar más por sentado. Al tomar una decisión global en lugar de muchas locales, puede concentrarse en las características más importantes del código. Le ayudan a transferir conocimientos entre proyectos. Las similitudes en los nombres le brindan una comprensión más fácil y segura de lo que se supone que deben hacer las variables desconocidas. Le ayudan a aprender código más rápidamente en un nuevo proyecto.

● Reducen la proliferación de nombres. Sin convenciones de nomenclatura, puede llamar fácilmente a la misma cosa con dos nombres diferentes. Por ejemplo, puede llamar puntos totales tanto pointTotal como totalPoints.

● Compensan las debilidades del idioma. Puede utilizar convenciones para emular constantes con nombre y tipos enumerados. Las convenciones pueden diferenciar entre datos locales, de clase y globales y pueden incorporar tipos.

● Enfatizan las relaciones entre elementos relacionados. Si usa datos de objeto, el compilador se encarga de esto automáticamente. Si su idioma no admite objetos, puede complementarlo con una convención de nomenclatura. Nombres como dirección, teléfono y nombre no indican que las variables estén relacionadas. Pero suponga que decide que todas las variables de datos de empleados deben comenzar con un prefijo de empleado. employeeAddress, employeePhone y employeeName no dejan ninguna duda de que las variables están relacionadas. Las convenciones de programación pueden compensar la debilidad del lenguaje que está utilizando.

● El poder de las convenciones de nomenclatura no proviene de la convención específica elegida, sino del hecho de que existe una convención, lo que agrega estructura al código y le brinda menos cosas de las que preocuparse.

## Cuándo debería tener una convención de nomenclatura

## ● Cuando varios programadores están trabajando en un proyecto.

## ● Cuando planea entregar un programa a otro programador para modificaciones y mantenimiento (lo cual es casi siempre)

## ● Cuando otros programadores de su organización revisan sus programas

## ● Cuando su programa es tan grande que no puede retenerlo todo en su cerebro

## ● Cuándo el programa durará lo suficiente como para dejarlo de lado durante algunas semanas o meses antes de volver a trabajar en él.

## ● Cuando tiene muchos términos inusuales que son comunes en un proyecto y desea tener términos estándar o abreviaturas para usar en la codificación.11.4 Informal Naming Conventions

La mayoría de los proyectos utilizan convenciones de nomenclatura relativamente informales, como las que se describen en esta sección

**Guidelines for a Language-Independent Convention**

A continuación, se muestran algunas pautas para crear una convención independiente del idioma:

● Diferenciar entre nombres de variables y nombres de rutinas. La convención que utiliza este libro es comenzar los nombres de variables y objetos con minúsculas y los nombres de rutinas con mayúsculas: variableName vs. Diferenciar entre clases y objetos.

Example 11-17.

Each of these options has strengths and weaknesses.

Option 1: Differentiating Types and Variables via Initial Capitalization

Widget widget;

LongerWidget longerWidget;

Option 2: Differentiating Types and Variables via All Caps

WIDGET widget;

LONGERWIDGET longerWidget

Option 3: Differentiating Types and Variables via the “t\_” Prefix for

Types

t\_Widget Widget;

t\_LongerWidget LongerWidget;

Option 4: Differentiating Types and Variables via the “a” Prefix for

Variables

Widget aWidget;

LongerWidget aLongerWidget;

6 Option 5: Differentiating Types and Variables via Using More Specific

Names for the Variables

Widget employeeWidget;

LongerWidget fullEmployeeWidget;

● Identificar variables globales. Un problema de programación común es el mal uso de variables globales.

● Identificar las variables de los miembros. Identifica los datos de los miembros de una clase. Aclare que la variable no es una variable local y que tampoco es una variable global.

● Identificar definiciones de tipos. Las convenciones de nomenclatura para los tipos tienen dos propósitos: identifican explícitamente un nombre como un nombre de tipo y evitan los conflictos de nombres con las variables.

Para evitar confusiones, puede prefijar los nombres de los tipos con t\_, como t\_Color y t\_Menu.

● Identificar constantes nombradas. Las constantes con nombre deben identificarse para que pueda saber si le está asignando a una variable un valor de otra variable (cuyo valor podría cambiar) o de una constante con nombre.

En C ++ y Java, la convención es usar todas las letras mayúsculas, posiblemente con guiones bajos para separar palabras, RECSMAX o RECS\_ MAX

● Identificar elementos de tipos enumerados. Los elementos de los tipos enumerados deben identificarse por las mismas razones que las constantes con nombre:

● Identificar parámetros de solo entrada en idiomas que no los aplican. A veces, los parámetros de entrada se modifican accidentalmente.

● Dar formato a los nombres para mejorar la legibilidad. Dos técnicas comunes para aumentar la legibilidad son el uso de mayúsculas y caracteres de espaciado para separar palabras.

otros idiomas también permiten el uso del separador de subrayado (\_).

● Trate de no mezclar estas técnicas; eso hace que el código sea difícil de leer.

**Guidelines for Language-Specific Conventions**

En contraste con C y C ++, las convenciones de estilo de Java han estado bien establecidas desde el comienzo del lenguaje: i y j son índices enteros. Las constantes están en ALL\_CAPS separadas por guiones bajos. Los nombres de clases e interfaces escriben con mayúscula la primera letra de cada palabra, incluida la primera palabra, por ejemplo, ClassOrInterfaceName. Los nombres de variables y métodos usan minúsculas para la primera palabra, con la primera letra de cada palabra siguiente en mayúscula, por ejemplo, variableOrRoutineName. El guión bajo no se utiliza como separador dentro de los nombres, excepto para los nombres en mayúsculas. Los prefijos get y set se utilizan para métodos de acceso.

**Sample Naming Conventions**

Las convenciones estándar anteriores tienden a ignorar varios aspectos importantes de la nomenclatura que se discutieron en las últimas páginas, incluido el alcance de variables (privado, de clase o global), la diferenciación entre nombres de clases, objetos, rutinas y variables, y otros problemas.

Tabla 11-3. Convenciones de nomenclatura de muestra para C ++ y Java

## 11.5 Standardized Prefixes

Otras lecturas

Para más detalles sobre la convención de nomenclatura húngara, consulte "La revolución húngara" (Simonyi y Heller 1991).

La estandarización de prefijos para significados comunes proporciona un enfoque conciso pero consistente y legible para nombrar datos. El esquema más conocido para estandarizar prefijos es la convención de nomenclatura húngara, que es un conjunto de pautas detalladas para nombrar variables y rutinas (¡no húngaros!) Que se usó ampliamente en algún momento en la programación de Microsoft Windows.

ya no es de uso generalizado, la idea básica de estandarizar en abreviaturas concisas y precisas sigue teniendo valor. Los prefijos estandarizados se componen de dos partes: la abreviatura del tipo definido por el usuario (UDT) y el prefijo semántico.

**User-Defined Type Abbreviations**

La abreviatura UDT identifica el tipo de datos del objeto o variable que se nombra. Las abreviaturas de UDT pueden referirse a entidades como ventanas, regiones de pantalla y fuentes. Una abreviatura UDT generalmente no se refiere a ninguno de los tipos de datos predefinidos que ofrece el lenguaje de programación.

**Semantic Prefixes**

Los prefijos semánticos van un paso más allá del UDT y describen cómo se usa la variable u objeto. A diferencia de los UDT, que varían de un proyecto a otro, los prefijos semánticos son un estándar en todos los proyectos.

La tabla 11-7 muestra una lista de prefijos semánticos estándar.

Los prefijos semánticos se formatean en minúsculas o en mayúsculas y minúsculas mezclados y se combinan con los UDT y con otros prefijos semánticos según sea necesario.

## 11.6 Creating Short Names That Are Readable

El deseo de utilizar nombres de variables cortos es, en cierto modo, un vestigio de una era anterior de la informática. Los lenguajes más antiguos como ensamblador, básico genérico y Fortran limitaban los nombres de variables a 2–8 caracteres

En lenguajes modernos como C ++, Java y Visual Basic, puede crear nombres de prácticamente cualquier longitud; casi no tiene motivos para acortar nombres significativos.

Es una buena idea estar familiarizado con varias técnicas para abreviar porque ninguna técnica funciona bien en todos los casos.

**General Abbreviation Guidelines**

Aquí hay varias pautas para crear abreviaturas. Algunos de ellos contradicen a otros, así que no intentes usarlos todos al mismo tiempo.

● Utilice abreviaturas estándar (las de uso común, que se enumeran en un diccionario).

● Elimina todas las vocales no principales. (la computadora se convierte en cmptr y la pantalla se convierte en scrn. apple se convierte en aplique y el entero se convierte en intgr.)

● Eliminar artículos: y, o, y así sucesivamente.

● Utilice la primera letra o las primeras letras de cada palabra.

● Truncar consistentemente después de la primera, segunda o tercera (la que sea apropiada) letra de cada palabra.

● Conserve la primera y la última letra de cada palabra.

● Utilice todas las palabras significativas del nombre, hasta un máximo de tres palabras.

● Elimine sufijos inútiles: ing, ed, etc.

● Mantenga el sonido más notorio en cada sílaba.

● Asegúrese de no cambiar el significado de la variable.

Repita estas técnicas hasta que reduzca el nombre de cada variable entre 8 y 20 caracteres o el número de caracteres al que su idioma limita los nombres de las variables.

**Phonetic Abbreviations**

Algunas personas abogan por crear abreviaturas basadas en el sonido de las palabras en lugar de su ortografía. Así, el patinaje se convierte en sk8ing, el realce se convierte en hilite, antes se convierte en b4, la ejecución se convierte en xqt, y así sucesivamente.

**Comments on Abbreviations**

Puede caer en varias trampas al crear abreviaturas. Aquí hay algunas reglas para evitar trampas:

● No abrevie quitando un carácter de una palabra.

● Abreviar de forma coherente. Utilice siempre la misma abreviatura. Por ejemplo, use Num en todas partes o No en todas partes, pero no use ambos.

● Cree nombres que pueda pronunciar. Utilice xPos en lugar de xPstn y needsComp en lugar de ndsCmptg.

● Cree nombres que pueda pronunciar. Utilice xPos en lugar de xPstn y needsComp en lugar de ndsCmptg. Aplique la prueba del teléfono: si no puede leer su código a alguien por teléfono, cambie el nombre de sus variables para que sean más distintivas (Kernighan y Plauger 1978).

● Evite combinaciones que resulten en mala lectura o mala pronunciación. Para referirse al final de B, prefiera ENDB sobre BEND.

● Utilice un diccionario de sinónimos para resolver conflictos de nombres.

● Documente nombres extremadamente cortos con tablas de traducción en el código. En los idiomas que solo permiten nombres muy cortos, incluya una tabla de traducción para proporcionar un recordatorio del contenido mnemónico de las variables. Incluya la tabla como comentarios al comienzo de un bloque de código.

● Documentar todas las abreviaturas en un documento de "Abreviaturas estándar" a nivel de proyecto.

El problema general ilustrado por esta guía es la diferencia entre la conveniencia del tiempo de escritura y la conveniencia del tiempo de lectura. Este enfoque crea claramente un inconveniente en el tiempo de escritura, pero los programadores durante la vida útil de un sistema pasan mucho más tiempo leyendo código que escribiendo código.

Recuerde que los nombres le importan más al lector del código que al escritor. Lea su propio código que no haya visto durante al menos seis meses y observe dónde tiene que trabajar para comprender el significado de los nombres.

**Kinds of Names to Avoid**

* ● Evite nombres o abreviaturas engañosas.
* FALSE suele ser lo opuesto a TRUE y sería una abreviatura incorrecta de "Fig y
* Temporada de almendras ".
* ● Evite los nombres con significados similares.
* input y inputValue, recordNum y numRecords, y fileNumber y fileIndex son
* o semánticamente similar
* ● Evite nombres como clientRecs y clientReps. Son solo una letra diferente entre sí, y la letra es difícil de notar.
* ● Evite nombres que suenen similares, como wrap y rap.
* ● Evite los números en los nombres.
* Si los números de un nombre son realmente significativos, utilice una matriz en lugar de variables independientes. Si una matriz es inapropiada, los números lo son aún más. Por ejemplo, evite file1 y file2, o total1 y total2.
* ● La falta de ortografía de resaltado como hilite para guardar tres caracteres hace que sea endiabladamente difícil para un lector recordar cómo se escribió mal resaltado.
* ● Evite las palabras que suelen estar mal escritas en inglés.
* ● No distinga los nombres de las variables únicamente por el uso de mayúsculas. Si está programando en un lenguaje que distingue entre mayúsculas y minúsculas, como C ++,
* ● Evite múltiples lenguajes naturales. En proyectos multinacionales, aplique el uso de un solo lenguaje natural para todo el código, incluidos los nombres de clases, nombres de variables, etc.
* ● Ocurre un problema más sutil en las variaciones del inglés. Si un proyecto se lleva a cabo en varios países de habla inglesa, estandarice en una versión de inglés
* ● Evite los nombres de tipos, variables y rutinas estándar.
* ● No use nombres que no tengan ninguna relación con lo que representan las variables. Esparcir nombres como margaret y pookie en todo el programa garantiza virtualmente que nadie más podrá entenderlo. Evite el nombre de su novio, el nombre de su esposa, el nombre de su cerveza favorita,
* ● Evite los nombres que contengan caracteres difíciles de leer. Tenga en cuenta que algunos personajes se parecen tanto que es difícil distinguirlos.