Datos

¿De qué nos ocuparemos?
La definición de dato, los diferentes tipos de datos desde simples caracteres y números hasta colecciones y cómo definir un tipo de datos propio.

Dato es uno de esos términos que todos usan pero pocos entienden. Mi diccionario lo define como:

> Dato: "hecho o valor a partir del cual se puede inferir una conclusión; información"

No es mucho mejor, pero al menos nos da un punto de partida.

Los datos son aquello que un programa manipula. Sin datos un programa no funcionaría correctamente. Los programas manipulan datos de manera muy diferente según el *tipo* de dato del que se trate. Y hay varios de estos tipos:

Tipos de Datos

Cadenas de Caracteres

Ya hemos visto este tipo de datos. Son literalmente cualquier cadena o secuencia de caracteres que puedan imprimirse en la pantalla. (De hecho pueden ser también ciertos *caracteres de control* que no son imprimibles).

En Python las cadenas pueden representarse de varias formas:

Con comillas simples:

```
'Esta es una cadena'
```

Con comillas dobles:

```
"Esta otra es una cadena muy similar"
```

Con tres comillas dobles:

```
""" Esta es una cadena muy larga que puede
    ocupar varias líneas si así lo deseamos y
    Python la mantendrá del mismo modo en que
    nosotros la tipeamos..."""
```

Un uso especial de esta última forma puede verse en la generación de la documentación de las funciones de Python creadas por nosotros mismos, algo que veremos más adelante.

Podemos acceder a los caracteres individuales de una cadena al tratarla como una matriz de caracteres (ver 'matrices' más adelante). Usualmente hay varias operaciones que el lenguaje de programación provee para ayudarnos a manipular las cadenas, tales como buscar una subcadena dentro de otra, unir dos cadenas, copiar una cadena en otra, etc.

Enteros

Números enteros desde un valor negativo alto hasta otro valor positivo alto. El valor máximo se conoce como *MAXINT* y depende de la cantidad de bits utilizados en la computadora para representar un número. En la mayor parte de las computadoras actuales esta cantidad es de 32 bits, lo que implica que MAXINT se acerce a los dos billones.

También podemos utilizar *enteros sin signo* lo que incluye números positivos y el cero. De esta manera el número máximo alcanzable equivale a dos por MAXINT, o cuatro billones en una computadora de 32 bits.

Dado que el tamaño de los enteros está restringido a MAXINT, cuando sumamos dos enteros cuyo total es mayor que MAXINT, el resultado obtenido es incorrecto. En algunos lenguajes y sistemas el resultado incorrecto igual se devuelve (usualmente con algún tipo de aviso secreto que uno puede revisar si cree que pudo haber habido algún problema). Normalmente en estos casos se produce un error, el cual será manejado por el programa o directamente éste finalizará. Pithon utiliza este último sistema, mientras que TCL ha adoptado el primero. BASIC produce un error pero no provee ningún método para tratarlo (al menos yo no sé cómo).

Números Reales

Estos son las fracciones. Pueden representar números muy altos, más altos que MAXINT pero con menor precisión. Esto quiere decir que dos números reales que deberían ser idénticos pueden no serlo cuando son examinados por la computadora. Esto se debe a que la computadora trabaja por aproximación en los más mínimos detalles. De esta forma 4,0 podría ser representado como 3,9999999.... o 4,00000001. Estas aproximaciones son lo suficientemente precisas para la mayor parte de nuestros objetivos, pero ocasionalmente pueden ser importantes para alguna tarea específica. Recordá esto si obtenés un resultado extraño al utilizar números reales.

Números Complejos o Imaginarios

Si tenés una formación científica o matemática seguramente los conocerás muy bien. Si este no es tu caso, lo más probable es que ni siquiera hayas escuchado hablar de los números complejos. De todos modos, algunos lenguajes de programación -Fortran, por ejemplo- permiten trabajar con números complejos. La mayor parte del resto, como Python, proveen una librería de funciones que permiten operar con números complejos. Y antes de que preguntes, lo mismo se aplica para las matrices.

Valores Booleanos - Verdadero y Falso

Como indica el encabezado, este tipo presenta sólo dos valores: *verdadero* o *falso*. Algunos lenguajes manipulan los valores booleanos directamente, mientras que otros usan una convención por medio de la cual un valor numérico (en general 0) representa 'falso' y otro (1 o -1) equivale a 'verdadero'.

En general se conoce a los valores booleanos como "valores de verdad" debido a que son utilizados para comprobar si algo es verdadero o falso. Por ejemplo, si escribimos un programa que realice backups de todos los archivos en un directorio, lo que debemos hacer es copiar un archivo y luego preguntarle al sistema operativo por el nombre del siguiente archivo. Si no hay más archivos responderá con una cadena vacía, entonces podremos comprobar que la cadena está vacía y guardar el resultado como un valor booleano (verdadero si está vacía). Pronto verás cómo utilizaremos este resultado más adelante en este curso.

Colecciones

Las ciencias de la Computación han creado una disciplina en sí misma para estudiar las colecciones y sus diversos comportamientos. Algunos de los nombres que podrás encontrar son:

Matrices o vectores
: Una lista de items que pueden ser indexados para una recuperación sencilla y rápida. Usualmente es necesario aclarar de entrada cuántos items deseamos guardar en la matriz. Por ejemplo, si tenemos un vector llamado A, podemos recuperar su tercer item escribiendo A[3]. (En realidad, generalmente los vectores comienzan en la posición 0, por lo cual deberíamos escribir A[2]). Las matrices o vectores son fundamentales en BASIC ya que son la única colección predeterminada. En Python las matrices se simulan por medio de listas (ver más abajo) y Tcl implementa las matrices como diccionarios (ver más abajo).

Listas
: Una lista es una secuencia de items. La diferencia con los vectores es que una lista puede seguir creciendo al agregársele un nuevo item. En general no están indexadas, por lo cual se hace necesario buscar el item requerido recorriendo desde el principio al fin toda la lista y evaluando cada elemento para ver si es el que nosotros buscamos. Tanto Python como Tcl trabajan con listas, mientras que en BASIC debemos utilizar algunos trucos para simularlas. Los programadores de BASIC en general crean matrices muy grandes para superar esta debilidad. Python permite indexar las listas -en realidad estrictamente no maneja vectores, pero combina en un mismo elemento la posibilidad de indexación de los vectores con la habilidad para crecer de las listas. Como veremos pronto esta característica es realmente muy útil.

Pilas
: Pensá en una pila de bandejas en un restaurant: un asistente coloca una pila de bandejas limpias sobre las que ya había antes, y estas son tomadas una por una por los clientes. De esta manera, las bandejas que quedan abajo de todo son las menos utilizadas (y a veces no les llega nunca la oportunidad de ser usadas). Las pilas de datos funcionan del mismo modo: se agrega un dato a la pila o se retira uno de ella, pero el dato retirado es siempre el último que se colocó en la pila. Esta propiedad de las pilas se denomina con frecuencia *First In Last Out ("el primero es el último")* o *FILO*. Una característica útil de las pilas es que se puede revertir una lista de items al colocarla en la pila y luego retirarla. El resultado será una lista inversa respecto de la original. Las pilas no son tipos predeterminados en Python, Tcl o Basic, por lo que es necesario crear una función para implementarlas. Sin embargo, las listas son en general el mejor punto de partida, ya que pueden crecer -igual que las pilas- según sea necesario.

Bolsas
: Una bolsa es una colección de items sin un orden específico y que puede contener duplicados. Las bolsas tienen en general operadores que nos permiten agregar, buscar y borrar los items. En Python y en Tcl las bolsas son simplemente listas. En BASIC es necesario crear la bolsa a partir de una matriz grande.

Conjunto
: Un conjunto tiene la propiedad de guardar únicamente un miembro de cada item. En estos, se puede comprobar si un item pertenece al conjunto (pertenencia), y agregar, remover u obtener items o unir dos conjuntos según la teoría matemática de los conjuntos (unión, intersección, etc.). Ninguno de los lenguajes estudiados aquí implementa conjuntos directamente, pero pueden ser utilizados en Python y en Tcl gracias al tipo de datos llamado diccionario.

Cola
: Una cola es similar a una pila excepto que el primer elemento de una cola es el primero en ser retirado. Esto se conoce como *First In First Out ("el primero es el primero")* o *FIFO*.

Diccionarios
: Un diccionario combina las propiedades de las listas, los conjuntos y los vectores. Es posible seguir agregando elementos (como en las listas) y también acceder a los items mediante una clave provista en el punto de inserción (como con los vectores). Debido a que el acceso a los datos se realiza por medio de una clave, ésta debe ser necesariamente única ya que si no se perdería la referencia (como en los conjuntos). Los diccionarios son estructuras inmensamente útiles y son tipos de datos predeterminados tanto en Python como en Tcl. En BASIC son muy poco utilizados dada la dificultad de implementarlos de manera eficiente. Podemos utilizar los diccionarios de muchas maneras y más adelante veremos varios ejemplos. Por ahora, veamos cómo crear un diccionario en Python, cómo agregar algunos datos y cómo recobrarlos:

```
>>> dict = {}
>>> dict['booleano'] = "Un dato cuyo valor puede ser verdadero o falso"
>>> dict['entero'] = "Un número entero"
>>> print dict['booleano']
```

Sencillo ¿no?

Hay muchos otros más pero estos son los principales con los que trabajaremos en este curso (de hecho, sólo utilizaremos algunos de estos).

Archivos

Como usuario de computadoras sabrás todo acerca de los archivos, la base de prácticamente todo lo que hacemos con una computadora. No es sorprendente entonces que la mayor parte de los lenguajes de programación incluyan un tipo especial de datos llamado *archivo*. Dado que los archivos y su procesamiento son tan importantes, dejaré para más adelante su discusión en una sección especial.

Fecha y Hora

La fecha y la hora a veces se incluyen como predeterminados en algunos lenguajes. En otros casos son representados simplemente por un número alto (típicamente el número de segundos a partir de una determinada fecha u hora). Para otros lenguajes este tipo se representa de manera compleja, como veremos en la próxima sección. Este procedimiento vuelve más sencillo recuperar el mes, el día o la hora.

Tipos Complejos/Definidos por el Usuario

Muchas veces los tipos básicos descriptos más arriba no son adecuados para determinada tarea aunque los combinemos por medio de colecciones. En ocasiones deseamos agrupar varios datos juntos y tratarlos como si fueran un solo elemento. Un ejemplo de esta situación podría ser la descripción de una dirección postal: la calle, el número, la ciudad y el código postal. La mayor parte de los lenguajes nos permiten agrupar estos datos en un *registro* o *estructura*.

En BASIC un registro de este tipo se realiza así:

```
Type Direccion
     Numero_Casa AS INTEGER
     Calle AS STRING
     Ciudad AS STRING
     Cod_Postal AS STRING
End Type
```

En Python es algo diferente:

```
class Direccion:
    def __init__(self, Casa, Calle, Ciudad, Codigo):
        self.Numero_Casa = Casa
        self.Calle = Calle
        self.Ciudad = Ciudad
        self.Cod_Postal = Codigo
```

Puede parecerte medio esotérico, pero no te preocupes que pronto tendrá su sentido.

Veremos cómo utilizar estas estructuras en nuestra próxima sección dedicada a las variables.

Variables

Los datos son almacenados en la memoria de la computadora. Podemos comparar este proceso con las casillas de correo donde se colocan las cartas. Uno podría colocar una carta en cualquier casilla, pero si estas no tienen una etiqueta que las identifique, resultará prácticamente imposible recuperar la carta. Para seguir con la comparación, las variables son las etiquetas de las casillas en la memoria de la computadora.

Ahora ya conocemos qué son los datos, pero ¿qué podemos hacer con ellos? Desde el punto de vista de la programación podemos crear *instancias* de los datos (organizados en objetos) y *asignarlas* a variables. Una variable es una referencia a un área específica de la memoria de la computadora donde se guardan los datos. En algunos lenguajes de programación la variable debe coincidir con el tipo de dato al cual apunta. En BASIC, por ejemplo, declaramos una variable de cadena agregándole el signo $ al final del nombre:

```
DIM MICADENA$
MICADENA$ = "Esta es una cadena"
```

En este ejemplo `DIM MICADENA$` crea la referencia y especifica el espacio para almacenar la cadena (y sabemos que se trata de una cadena por el signo $). La línea `MICADENA$ = "Esta es..."` define los datos y los coloca en el espacio de la memoria denominado `MICADENA$`.

De forma similar declaramos un entero mediante el signo % al final del nombre:

```
DIM MIENTERO%
MIENTERO% = 7
```

En Python y en Tcl una variable adquiere el tipo de datos que se le asigna por primera vez y lo mantendrá durante el programa, avisándonos si intentamos mezclar los datos de manera extraña, tal como sumar una cadena a un número (¿Recordás el ejemplo que vimos del mensaje de error?). Es posible cambiar el tipo de datos de una variable en Python reasignando la variable.

```
>>> q = 7
>>> print 2*q
14
>>> q = "Siete"
>>> print 2*q
SieteSiete
```

Notemos que inicialmente la variable q apuntaba al número 7 y mantuvo este valor hasta que reasignamos la variable con el valor "Siete". De esta manera las variables en Python mantienen cualquier tipo de datos siendo posible modificar la referencia hacia otro tipo de datos simplemente reasignando la variable. Cuando se produce la reasignación el dato original se pierde y Python lo borrará de la memoria (salvo que se lo recupere en otra variable). A esto se denomina "*recolección de basura*". (Esto puede compararse con el empleado del correo que cada tanto revisa las casillas y retira aquellos paquetes que carecen de información de destino. Si nadie los reclama ni es posible encontrar a su dueño, los paquetes son incinerados.)

BASIC no permite realizar esto. Si una variable es una variable de cadena (terminada en $) jamás podremos asignar a ella un número. De forma similar, es imposible asignar una cadena a una variable de enteros (terminada en %). Por otra parte, BASIC permite 'variables anónimas' (no tienen ningún identificador después del nombre). Sin embargo, este tipo de variables sólo puede contener números enteros o reales.

Un ejemplo final con variables enteras en BASIC:

```
i% = 7
PRINT 2 * i%
i% = 4.5
PRINT 2 * i%
```

Notemos que la asignación de 4.5 en la variable i% parece funcionar, sólo que en realidad lo único que fue asignado fue la parte entera del valor. Esto nos recuerda la forma en que Python trata la división de enteros. Todos los lenguajes de programación tienen sus pequeñas idiosincracias como esta.

Acceso a Tipos Complejos

Podemos también asignar un tipo de datos complejo a una variable, pero para acceder a cada uno de los *campos* individuales del tipo de datos deberemos utilizar algún mecanismo de acceso, que será definido por el propio lenguaje. Usualmente se trata de un punto.

Considerando el caso del tipo "dirección" que hemos definido antes, en BASIC realizaríamos lo siguiente:

```
DIM Direc AS Direccion
Direc.Numero = 7
Direc.Calle = "Los Rosales"
Direc.Ciudad = "Cualquiera"
Direc.Codigo = "123 456"
PRINT Direc.Numero," ",Direc.Calle
```

Y en Python:

```
Direc = Direccion(7,"Los Rosales","Cualquiera","123 456")
print Direc.Numero, Direc.Calle
```

Ahora que ya sabemos qué son las variables y cómo crearlas vamos a ver qué cosas podemos hacer con ellas.

Para recordar
• Los datos se organizan en tipos, y las operaciones que pueden realizarse sobre ellos dependen en gran medida del tipo de datos que se utilice. • Los tipos de datos simples incluyen a las cadenas de caracteres, los números y los valores de verdad o booleanos. • Los tipos de datos complejos incluyen a las colecciones, los archivos, las fechas y los tipos definidos por el usuario.

Anterior Siguiente Contenido

Si tenés preguntas o sugerencias, enviáme un e-mail a alan.gauld@btinternet.com