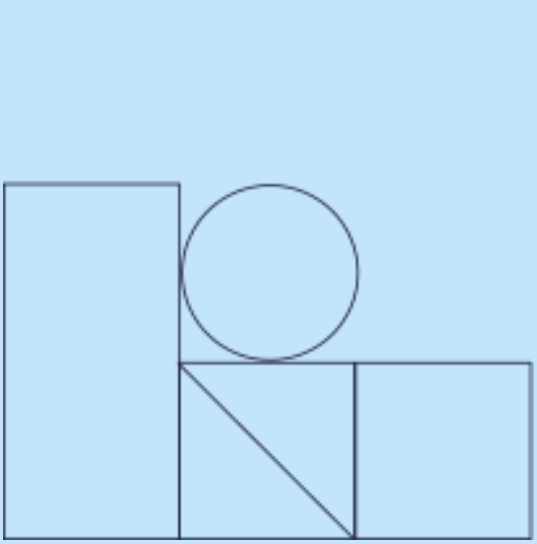
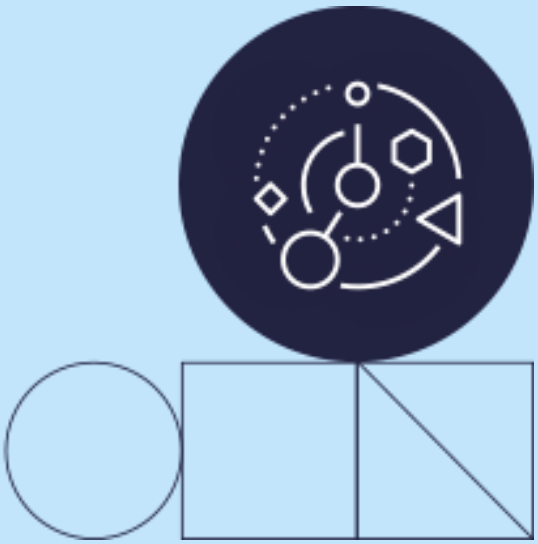
IBM SkillsBuild

IBM

Conceptos básicos de programación

Cuáles son los tipos de datos que existen en programación





Índice

Introducción

Se denomina dato a cualquier objeto manipulable por el ordenador. Un dato puede ser un carácter leído de un teclado, información almacenada en un disco, un número que se encuentra en la memoria central, etc. Los distintos tipos de datos se representan en diferentes formas en el ordenador: por ejemplo, no se almacena internamente de la misma manera un número entero que un carácter. Aunque los lenguajes de alto nivel permiten en alguna medida ignorar la representación interna de los datos, es preciso conocer algunos conceptos mínimos.

A nivel de máquina todos los datos se representan utilizando una secuencia finita de bits. De este hecho ya se deduce que no todos los datos son representables en un ordenador. La definición de un tipo de dato incluye la definición del conjunto de valores permitidos y las operaciones que se pueden llevar a cabo sobre estos valores. Cuando se utiliza un dato en un programa es preciso que esté determinado su tipo para que el traductor sepa cómo debe tratarlo y almacenarlo.

Dependiendo del lenguaje puede o no ser preciso declarar expresamente en el programa el tipo de cada dato.

No todos los tipos de datos existen en todos los lenguajes de programación. Hay lenguajes más ricos que otros en este sentido.

Los tipos de datos en un lenguaje de programación pueden ser muy variados, así que es difícil preguntarte cuantos tipos de datos hay en un lenguaje, ya que incluso podemos crear los nuestros propios, mediante enumeraciones o estructuras.

En general todos los lenguajes de programación de alto nivel trabajan con los mismos tipos de datos básicos. Si bien es cierto que cambia la sintaxis y el modo de crear variables dependiendo de si son de tipado débil o fuerte.

El tipado se refiere a cómo declaramos los tipos de variables. Por ejemplo, algunas las declaramos como enteras, algunas otras como cadena, flotantes, etcétera. Y en algunos lenguajes, no necesitamos declarar el tipo, pues éste se adivina.

Por otro lado, el tipado fuerte no permite hacer operaciones entre objetos de distintos tipos. No podemos sumar una cadena y un entero. En cambio, en los débilmente tipados sí.

Tipado débil

Lenguajes de programación de tipado débil son aquellos en los que no indicamos el tipo de variable al declararla. Ya se encarga el propio lenguaje de "adivinar" de qué tipo es nuestra variable.

La verdadera diferencia es que podemos asignar, por ejemplo, un valor entero a una variable que anteriormente tenía una cadena. Más adelante veremos con detenimiento los tipos de datos que existen.

También podemos operar aritméticamente con variables de distintos tipos. Por ejemplo, sumar "x" + 5.

Ventajas

- Nos olvidamos de declarar el tipo
- Podemos cambiar el tipo de la variable sobre la marcha. Por ejemplo, asignarle un string a un int, es decir, cambiar entre letras y números.
- Escribimos menos código

Desventajas

- Al hacer operaciones, a veces éstas salen mal. Por ejemplo, puede que intentemos sumar 500 + "400.00" + 10, cosa que será errónea ya que al estar "400.00" definido entre comillas, se considera un string, es decir, una cadena de texto, no numérica. Veremos este concepto más adelante.
- Hay que castear muchas veces. En ocasiones, tendremos que castear forzosamente las variables para que se comporten como queremos y no generen errores como los mencionados arriba.

  Nota: Hacer un casting consiste en cambiar forzosamente el tipo de una variable. Por ejemplo, diciéndole a nuestro programa que una variable cuyo valor es numérico esa tratada como alfanumérica, es decir, como cadena de caracteres.
- Código menos expresivo. Al declarar los argumentos de una función no sabemos si ésta espera un flotante (un número decimal), un entero, un string (cadena de caracteres)...etc. Tenemos que ir a la función, ver lo que hace e inferir el tipo de variable que espera.
- Inseguridad: existe la posibilidad de que un atacante descubra una vulnerabilidad donde nosotros esperemos una variable de determinado tipo pero se reciba otra.

Lenguajes que lo usan

- PHP
- Javascript

Ejemplo:

Veamos lo que pasa en JavaScript cuando hacemos la siguiente operación:

```
let resultado = "x" + 5;
```

En un lenguaje fuertemente tipado daría un error, pero en JavaScript no pasa nada:

```
> let resultado = "x" + 5;
<· undefined
> resultado
<· "x5"
```

Lo que hemos hecho en este ejemplo es crear una variable a la que hemos llamado "resultado" y le hemos dado como valor la suma de "x+5". En los lenguajes de tipado fuerte esto habría dado un error, ya que no se pueden sumar letras y números. En el caso de Javascript, al ser un lenguajes de tipado débil, nos permite hacerlo.

Tipado fuerte

Aquí es en donde indicamos el tipo de dato al declarar la variable. Dicho tipo no puede ser cambiado nunca. Y no podemos operar entre distintos tipos. Es decir, si declaramos por ejemplo una variable como numérica, nuca podremos meter dentro un dato que no sea numérico.

Ventajas

- Código expresivo: ahora sí sabremos de qué tipo espera un argumento una función.
- Menos errores: Nos olvidaremos de ver el tipo de variable antes de hacer operaciones con ésta.

Desventajas

- Escribir más código: tenemos que especificar el tipo de variable al declararla.

Lenguajes que lo usan

Por mencionar algunos...

- C
- C#
- Java
- Ruby
- Python

Ejemplo

Intentemos realizar la operación de “x” + 5 en Python, y veamos lo que pasa:

```
>>> resultado = "x" + 5
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: must be str, not int
>>>
```

Nos está lanzando un error en el que nos especifica que no se pueden sumar caracteres alfanuméricos y números.

En general, los lenguajes de tipado fuerte son más seguros, pues no permiten hacer operaciones con variables de distintos tipos.

Ponemos tres *ejemplos* de tipos de datos básicos que son los más utilizados: los **números**, los **textos** y las **fechas**.

Si estamos trabajando con **números**, podemos sumar, restar, multiplicar, dividir... y muchas operaciones más, como comparar.

Si estamos con **textos**, podemos comparar (si son iguales o no), podemos sustituir parte del texto, añadir texto a uno existente, etc.

Con **fechas** podemos también sumar o restar fechas (o días, meses, años), compararlas, etc.

Tipos de datos para variables

Cada lenguaje de programación puede trabajar con muchos tipos de datos.

Pero de todos ellos, siempre tendremos los **tipos primitivos de datos**.

Éstos están incorporados al lenguaje de programación, y nos sirve para poder hacer cosas más complicadas.

Vamos a hablar de los tipos de datos más comunes, que nos podemos encontrar en la mayoría de los lenguajes de programación, como puede ser Java, C o C++.

Tipo de datos número: int

Los números suelen ser representados en un lenguaje de programación de maneras diferentes, ya que importa decidir una serie de cuestiones:

- Cual va ser el tamaño que vamos a usar.
- Si va a tener números decimales o no.
- Si va ser negativo.

Empezaremos con el tipo primitivo `int`. Este tipo de datos representa cualquier número sin decimales, ya sea positivo o negativo.

Aunque es habitual encontrarlo escrito en el código fuente como `int`, hay otros lenguajes, como visual basic, que se escribe `integer`.

Tipos de datos número real: double o float

Si nos interesa utilizar un número con decimales, solemos encontrar el tipo de datos `double` o `float`.

A esto lo llamamos un número de punto flotante. Declararlo como un tipo u otro dependerá de la precisión en decimales que quieras tener. Los `float` se usan para números no muy grandes con decimales y el tipo `double` para números muy grandes con decimales.

¿Como podemos distinguir una variable si se ha declarado como `int` o como `float`? Por ejemplo, si nos encontramos un número con un punto decimal (**3.14**), se deduce que es de tipo `float`. Aunque podemos definir un número sin decimales como float, con lo que el programa le añadirá los decimales. Por ejemplo, si definimos una variable como `float` y le damos el valor de 456, nuestro programa la leerá como 456.00

También puedes ver un número con la letra F o con la letra D, para distinguir si es un `float` o un `double`. Por ejemplo **3.56F**.

Tipos de datos cadena: char o string

Suele ser un valor alfanumérico. Si es un sólo carácter individual, tenemos el tipo `char`.

Un `char` es un carácter Unicode, y solemos escribirlo entre comillas simples (‘ ’).

Pero si es una cadena de caracteres, es decir, caracteres seguidos unos detrás de otro formando una secuencia (una palabra o una frase), lo solemos encontrar como `string`.

El tipo `string` debemos escribirlo entre comillas dobles (“ ”) para diferenciarlo del `char`, aunque puede ser diferente, dependiendo del lenguaje de programación.

En el siguiente código veremos un uso del tipo de datos `string`:

```
miVariable = "Hola Mundo";
```

Hemos declarado una variable en la que le hemos introducido como valor la frase “Hola mundo”, por lo que dicha variable es de tipo `string`.

Tipo	Tamaño en bytes	Ejemplo
Int	2	1, 55, 73, 1500
Float	4	4.33, 5.92, 75.22, $5e^{-2}$
Double	8	7.5138, 9.513, $7e^{-5}$
Char	1	T, Y, %, ¡, #, 52
Void	0	No hay valor

Tipo de datos booleano: boolean

Los valores lógicos son representados por el tipo primitivo `boolean`. Representa si una condición se cumple o no se cumple.

Suelen tener dos valores identificados, `true` (verdadero) o `false` (falso). En algunos lenguajes puede equivaler a los números **0** y **1**.

Una variable puede usarse para representar cualquiera de los dos **valores**.

Por ejemplo, podríamos hablar de verdadero o falso, encendido o apagado, activo o no activo...etc.

Tipos de datos abstractos

Ahora que ya conocemos los tipos de datos primitivos, podemos saber **que significa tipos de datos abstractos**.

Los **tipos de datos en programación orientada a objetos** suelen ser los básicos que hemos visto antes, pero podemos encontrarnos con tipos de datos que sean clases u objetos. Esto se verá con profundidad cuando tratemos la programación orientada a objetos.

Los **tipos de datos abstractos** aumentan y extienden la complejidad, ya que tiene un *conjunto de valores* y unas *operaciones* asociadas a ellos.

Luego entraríamos en que estos datos están *encapsulados*, en la herencia para aprovechar mejor las operaciones que hemos codificado y el *polimorfismo*, pero esos son conceptos que serán explicados más adelante.