

Python para Análisis de Datos

Módulo 01

Datos

¿Qué es un dato?

Un dato es una representación simbólica de algún aspecto de la realidad, que debe ser adecuado para su comunicación e interpretación.

Los datos pueden ser cuantitativos o cualitativos, es decir que pueden ser representados por números (y normalmente alguna unidad) en el primer caso, o por palabras que representan alguna cualidad en el segundo. Por ejemplo, que un film dure 2 horas es un dato cuantitativo y que una persona lo califique como 'excelente' es un dato cualitativo.



Estructura

También se puede diferenciar entre **datos estructurados** y **no estructurados**.

Básicamente, entendemos por **datos estructurados** a aquellos que se encuentran **ordenados por tipo de dato** y que son fácilmente accesibles. Ejemplos de este tipo son las bases de datos relacionales, las planillas de cálculo, los csv, los formularios, etc.

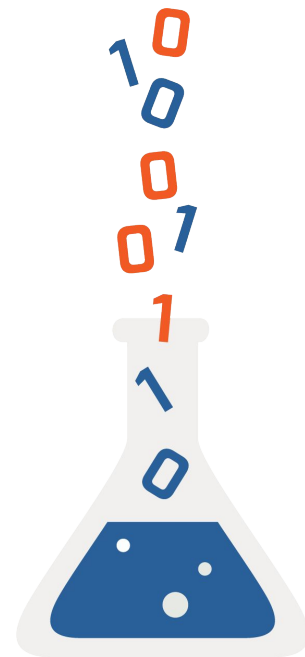
Por su parte, los **datos no estructurados** son aquellos que **carecen de un orden dado por un esquema predefinido**. Esto hace que la información no sea tan fácil de acceder como en los datos estructurados. Ejemplos son archivos de texto, audio, imágenes, video, etc.

Representación de datos

La memoria de una computadora se constituye en una serie de bits, que sólo pueden almacenar ceros y unos. Así que, para poder usar una computadora para analizar datos, hay que representarlos como una combinación de ceros y unos.

Es decir, hay que establecer algún tipo de codificación entre la información que queremos representar y los elementos de la memoria de la máquina para que sea posible tanto codificarlos, como analizarlos y decodificarlos.

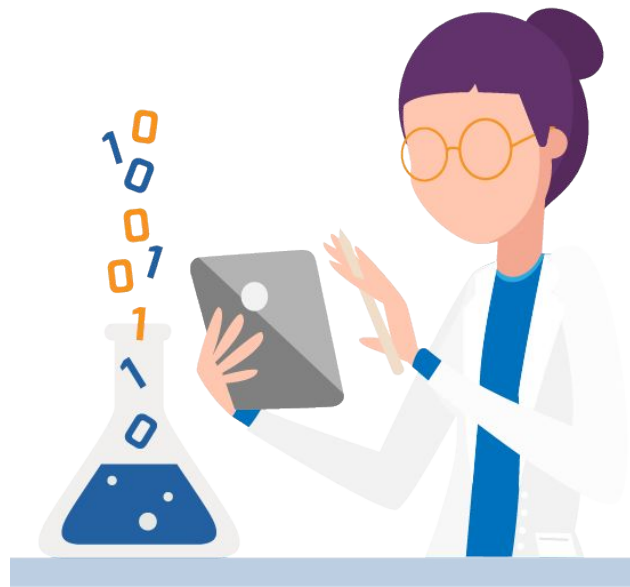
Así es como surge la necesidad de definir distintos tipos de datos.








Enteros

Usualmente escribimos los números en decimal o base diez, por lo que necesitamos diez símbolos (del 0 al 9). Pero, como cada bit sólo puede tener uno de dos valores (0 y 1), en una computadora es necesario representar los números en binario, es decir en base 2.

En este sistema cada posición representa una potencia de 2. Con n bits se pueden representar 2^n combinaciones diferentes de ceros y unos, por lo que hay una cantidad máxima de números que se pueden representar con esa cantidad de bits.



Para representar números enteros sin signo cada posición representa una potencia no negativa de 2.

Posición					
Potencia	4	3	2	1	0
Valor	2^4	2^3	2^2	2^1	2^0
Valor	16	8	4	2	1

Ejemplo

Los bits 1011 pueden representar el número once si trabajamos con enteros sin signo.

Sin embargo, también se puede usar el primer bit para representar el signo del número usándose 0 para los positivos y 1 para los negativos. En ese caso, los mismos bits representarían el número -3.

$$\begin{array}{cccc} 1 & 0 & 1 & 1 \\ \hline 1 \times 2^3 & + & 0 \times 2^2 & + & 1 \times 2^1 & + & 1 \times 2^0 \\ 8 & + & 0 & + & 2 & + & 1 & = & 11 \end{array}$$

$$\begin{array}{cccc} 1 & 0 & 1 & 1 \\ \hline (-1) \times (0 \times 2^2 & + & 1 \times 2^1 & + & 1 \times 2^0) \\ -1 \times (& 0 & + & 2 & + & 1 &) & = & -3 \end{array}$$

Hay muchas formas de representar a los números enteros. Como vimos, se puede representar enteros con o sin signo.

Si tenemos n bits para representar números sin signo, podemos cubrir el rango que va de 0 a $2^n - 1$. Pero si esos misma cantidad de bits la usamos para representar números con signo, podemos cubrir el rango de $\pm 2^{n-1} - 1$ (el cero se representa dos veces, $+0$ y -0). Si queremos representar números por fuera de ese rango, se produce lo que se llama **overflow**.

Otras variantes para representar números incluyen el **big-endian** o **little-endian**, es decir, si el bit más significativo va al final o al comienzo, y otra codificación llamada **complemento a 2** que no duplica la representación del cero.



Flotantes

Para representar números racionales, con parte decimal, se utiliza lo que se llama representación de números de coma flotante. Esta representación debe poder manejar también potencias negativas de 2 ($2^{-1}=0.5$, $2^{-2}=0.25$, $2^{-3}=0.125$, etc.).

Se utiliza algo similar a la notación científica, con una parte para representar las cifras significativas y otra para representar el exponente al que se eleva la base.

Existen flotantes de distinta precisión: simple (32 bits), doble (64 bits) y cuádruple (128 bits).

Texto

Vimos que podemos usar una serie de bits para representar números. Pero también debemos ser capaces de codificar otros tipos de datos en ceros y unos, por ejemplo, los caracteres de texto.

Existen muchas codificaciones de caracteres en números binarios, por ejemplo: ASCII, Unicode, UTF-8, etc. Todas estas codificaciones establecen un mapeo entre caracteres y números binarios de distinta longitud. ASCII usa 8 bits, pudiendo codificar 256 caracteres y Unicode utiliza 16 bits alcanzando 65536 caracteres.

h o l a
01101000 01101111 01101100 01100001 00100000
m u n d o
01101101 01110101 01101110 01100100 01101111

Algunos de los caracteres ASCII y su codificación en binario:

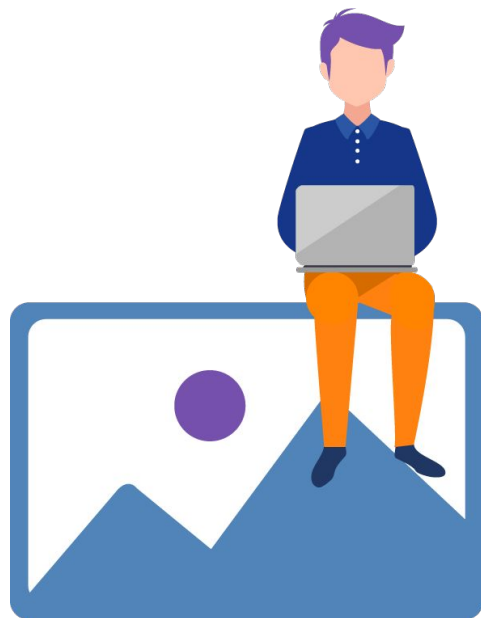
Dec	Symbol	Binary	Dec	Symbol	Binary
65	A	0100 0001	83	S	0101 0011
66	B	0100 0010	84	T	0101 0100
67	C	0100 0011	85	U	0101 0101
68	D	0100 0100	86	V	0101 0110
69	E	0100 0101	87	W	0101 0111
70	F	0100 0110	88	X	0101 1000
71	G	0100 0111	89	Y	0101 1001
72	H	0100 1000	90	Z	0101 1010
73	I	0100 1001	91	[0101 1011
74	J	0100 1010	92	\	0101 1100
75	K	0100 1011	93]	0101 1101
76	L	0100 1100	94	^	0101 1110
77	M	0100 1101	95	_	0101 1111
78	N	0100 1110	96	`	0110 0000
79	O	0100 1111	97	a	0110 0001
80	P	0101 0000	98	b	0110 0010
81	Q	0101 0001	99	c	0110 0011
82	R	0101 0010	100	d	0110 0100



Imágenes

Las pantallas muestran imágenes por píxeles, la menor unidad de representación de una imagen. Cuantos más píxeles, más resolución. Cada pixel está compuesto por tres colores: rojo, verde y azul. Cada color tiene $2^8 = 256$ intensidades posibles dando un total de $256^3 = 16777216$ combinaciones de colores. Por lo tanto para codificar la intensidad de cada color se utiliza un byte, y cada pixel se representa con 3 bytes.

Para una imagen en blanco y negro se puede usar un bit por pixel y si es en escala de grises su puede usar un byte por pixel.



Tipos básicos en Python

Python tiene los tipos de dato entero, flotante y complejo. Los **enteros** son de precisión ilimitada, lo que significa son creados dinámicamente y con la cantidad de bytes necesarios para representarlos. Los **flotantes** son de 64 bits y los **complejos** tienen parte real e imaginaria de 64 bits cada una.

Los **strings** son secuencias inmutables de caracteres Unicode y pueden tener cualquier longitud. En Python los **booleanos** son un tipo de dato que hereda de los enteros y que, para cualquier operación aritmética, tienen los valores `False=0` y `True=1`.

Cada uno de estos tipos de dato tiene un constructor: **`int()`**, **`float()`**, **`complex()`**, **`str()`** y **`bool()`**.



¡Muchas gracias!

¡Sigamos trabajando!