INGRESO 2025

Curso de Ingreso

UNIVERSIDAD TECNOLÓGICA NACIONAL

Sistema binario

CURSO COMPLETO

UNIDAD I FUNDAMENTOS LOGICOMATEMÁTICOS

GUIA 1 – Teoría de conjuntos, números y sus tipos

GUIA 2 – Sistema Binario

GUIA 3 – Introducción a la lógica

GUIA 4 – Operaciones aritméticas

GUIA 5 – Números enteros

GUIA 6 – Más de enteros y fracciones

UNIDAD II RESOLUCIÓN DE PROBLEMAS

GUIA 7 – Análisis verbal

GUIA 8 – Método iterativo

GUIA 9 – Analogía y Patrones

GUIA 10 – Divide y conquista

GUIA 11 – Integración

GUIA 12 – Ensayo y Error

2: SISTEMA BINARIO

SISTEMA DE NUMERACION

¿QUE ES?

Es la manera que usamos para representar cantidades o números y que todos podamos entender.

Un sistema de numeración usa símbolos y valores de posición.

En nuestro sistema de numeración, si escribimos “325”, sabemos que significa 3, 2 y 5, y sobre todo, que cada uno tiene un valor posicional 3 centenas, 2 decenas y 5 unidades.

Básicamente, es el lenguaje de los números. ¡Sin él, estaríamos usando palitos, piedras y sumas raras para todo!

ELEMENTOS DE UN SISTEMA DE NUMERACIÓN

BASE: Es la cantidad de dígitos o símbolos que se pueden usar.

Base 10 (decimal): dígitos del 0 al 9.

Base 2 (binario): dígitos 0 y 1.

Base 8 (octal): dígitos 0 al 7.

Base 16 (hexadecimal): dígitos 0 al 9, y 6 letras de la A a la F.

La base dicta cuántos dígitos tenés disponibles.

REGLAS

Nos indican cómo combinar esos dígitos para formar números más grandes.

JERARQUIA POSICIONAL

Cada dígito vale más o menos según la posición que ocupa.

En decimal, “2” en el primer lugar (unidades) vale 2, pero “2” en el segundo lugar (decenas) vale 20.

Dicho en criollo, si tengo 10010 pesos, no es lo mismo el billete de 10000 que el de 10...

DECIMAL Y BINARIO

SISTEMA DECIMAL (base 10): Lo usamos porque tenemos 10 dedos.

Cada posición es una potencia de 10 (10^0, 10^1, 10^2, etc.).

SISTEMA BINARIO (base 2): Solo ceros y unos.

Ideal para las computadoras que están basadas en circuitos electrónicos que distinguen 2 estados: encendido y apagado.

Aunque a primera vista parezca raro, es la esencia de cómo funcionan los dispositivos digitales sin “magia” de por medio.

CARACTERISTICA DEL SISTEMA BINARIO

REPRESENTACION MINIMA: encendido (1) y apagado (0).

Imaginate una llave de luz. Así de simple es la base del binario.

¿Te diste cuenta de que el símbolo universal de prendido y apagado es la combinación de 1 y 0?

RELACION CON LO FISICO/DIGITAL: Todo o nada, sí o no.

Este sistema combina perfecto con la electrónica, donde hay corriente (1) o no hay corriente (0).

En el fondo, toda tu PC, tu celular o tu consola están prendiendo y apagando circuitos a toda velocidad.

NATURALEZA DIGITAL: su relación con circuitos electrónicos.

Los transistores, los diminutos dispositivos dentro de chips, diferencian cuando hay voltaje (1) y cuando no (0).

A partir de eso, se construyen operaciones lógicas: sumas, restas, multiplicaciones, etc., todo usando ceros y unos.

¡Y así tus videojuegos, apps y redes sociales funcionan con millones de “prendidos y apagados” por segundo!

OCTAL Y HEXADECIMAL

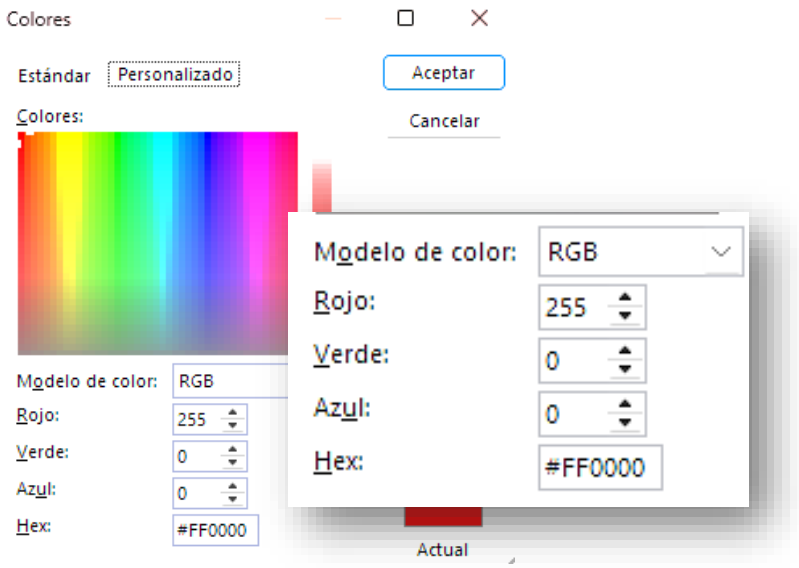
SISTEMA OCTAL (base 8): usa dígitos del 0 al 7.

SISTEMA HEXADECIMAL (base 16): Usa dígitos del 0 al 9 y letras de la A a la F (sí, letras).

Donde se usan:

Direcciones de memoria en programación, manejo interno de ciertos dispositivos, colores en Word como por ejemplo #FF0000 para el rojo.

Aquí te mostramos como podemos configurar color con tres valores decimales, o uno solo hexadecimal



NOTACION PARA DIFERENCIAR SISTEMAS

Usar subíndices para saber en qué base está el número.

Por ejemplo, 101_2 significa que es el “101” en binario, no el “ciento uno” en decimal.

Esto ayuda a evitar confusiones cuando vemos la misma secuencia de dígitos, pero en distintas bases.

¿QUE ES UN BIT?

Es la unidad mínima de información digital.

Es un dígito binario (Binary Digit), es decir, puede ser 0 o 1.

¿Cuántos bits necesito para representar algo? Depende de cuán “grande” sea lo que querés representar.

Por ejemplo:

1 bit es solo 2 valores, 0 y 1.

2 bits son 4 valores, 00, 01, 10 y 11

3 bits son 8 valores, 000, 001, 010, 011, 100, 101, 110 y 111

En resumen, un bit es el ladrillo base con el que se construyen todos los datos en la informática. ¡Sin él, no existiría nada de lo que conocemos en el mundo digital!

BIT / BYTE

Bit: la unidad mínima de información. Son solo dos valores.

Byte: conjunto de 8 bits. Representa 256 valores del 0 al 255.

La mayoría de los tipos de datos ocupan espacios en memoria que se mide en bytes e indica cuantos valores pueden almacenar.

¿CÓMO CONTAR EN BINARIO?

En decimal, con un digito contamos del 0 al 9. Si sumamos uno más, debemos agregar otro dígito: 10.

En binario con un dígito contamos del 0 al 1. Si sumamos uno más, agregamos otro dígito: 10.

Estamos tan acostumbrados al sistema decimal que cuando vemos 10, pensamos en diez. Por eso usamos el subíndice para indicar de que estamos hablando:

$10_2 = 2_{10}$

$101_2 = 5_{10}$

$111_2 = 7_{10}$ y así sucesivamente.

LOS PRIMEROS 16 BINARIOS

Para ver el patrón, listamos del 0 (cero) hasta el 15 (decimal) en binario. Te va a quedar algo así:

SISTEMA DECIMAL	SISTEMA BINARIO
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Fijate cómo vas sumando 1 en binario y cada vez que “colisionan” todos los unos, se agrega un bit extra cuando seguís contando.

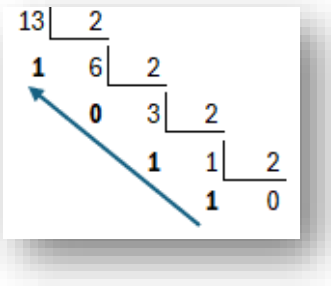
PASAR DE DECIMAL A BINARIO

METODO: División sucesiva por 2.

PASOS:

1. Dividís el número decimal entre 2.
2. Guardás el residuo (0 o 1).
3. Continúa dividiendo el cociente hasta llegar al 0.
4. Escribís los residuos al revés (del último al primero).

EJEMPLO: El 13 en decimal a binario



Dividimos por 2: 13 / 2 = 6 resto 1

Dividimos por 2: 6 / 2 = 3 resto 0

Dividimos por 2: 3 / 2 = 1 resto 1

Dividimos por 2: 1 / 2 = 0 resto 1 Acá terminamos

Leemos los "restos" del último al primero: 13_{10} corresponde a 1101_2

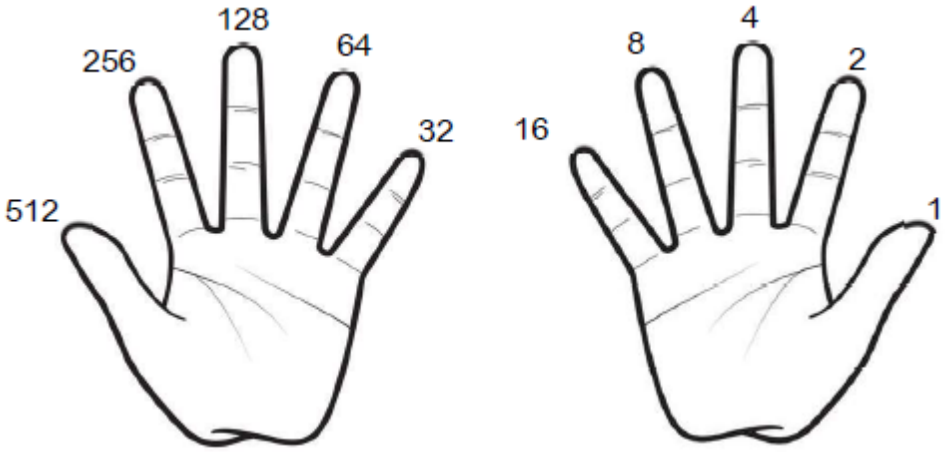
REGLA DE LA MANO PARA POTENCIAS DE 2

Sirve para determinar el valor posicional.

La primera cifra, posición 0, siempre su valor posicional es 1.

La siguiente cifra su valor posicional es 2, y el resto es el doble quedando: 1, 2, 4, 8, 16...

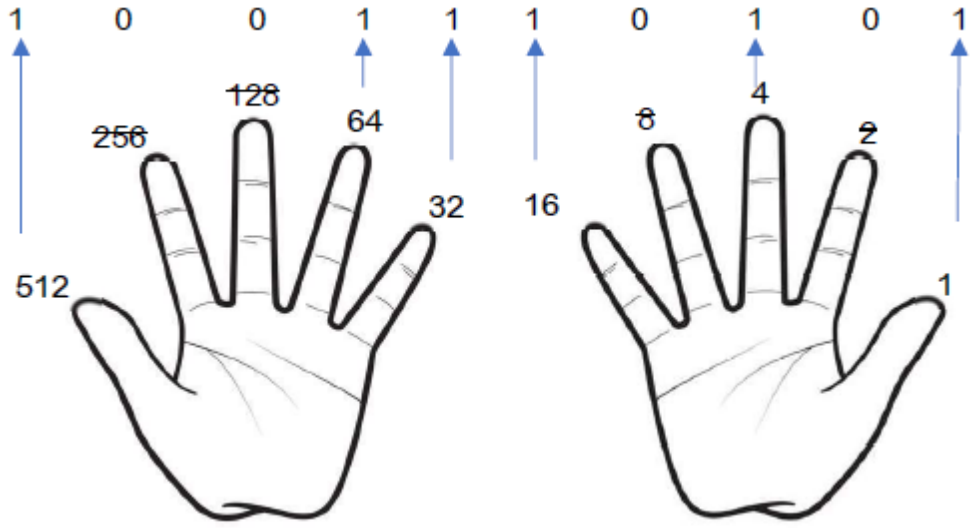
Nuestro primer dedo es la posición 0 y le asignamos valor 1, segundo dedo 2 y sigue...



PESO DEL BINARIO

Como los valores de los dígitos son 1 y 0, y como multiplicar por 1 es el mismo número y multiplicar por 0 es 0, podemos usar mano para encontrar la equivalencia del binario con el decimal.

Ejemplo: **1001110101**



1 + 4 + 16+ 32 + 64 + 512 = **629 en decimal**

PASAR DE BINARIO A DECIMAL

Empezamos de la primera cifra que es equivalente al primer dedo.

Regla: Si en esa posición es un 1 lo consideramos, sino lo descartamos.

Vamos pasando de cifra en cifra viendo los 1

Sumamos el equivalente de cada dedo.

Finalmente volvemos a la primera y sumamos lo que veamos, si es 1 o 0.

Ejemplo 1101_2

1) Empezamos de la primera cifra: 1 0 1 1
2) Lo comparamos con los dedos: 1 2 4 8
3) Como el que coincide con el segundo dedo es 0, queda: 1 0 4 8
4) Los sumamos: 1 + 4 + 8 = 13

En resumen:

Recorremos los bits desde la derecha sumando la potencia de 2 cuando hay un 1.

EJEMPLOS

Pasar de binario a decimal:

101 en binario → (4 + 0 + 1) → 5

111 en binario → (4 + 2 + 1) → 7

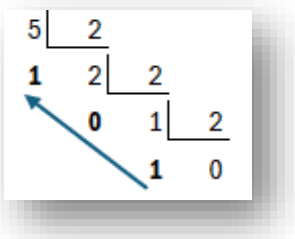
1010 en binario → (8 + 0 + 2 + 0) → 10

101101 en binario → (32 + 0 + 8 + 4 + 0 + 1) → 45

1001001 en binario → (64 + 0 + 0 + 8 + 0 + 0 + 1) → 73

El método es siempre el mismo: sumar potencias de 2 para convertir de binario a decimal.

Pasar de decimal a binario:



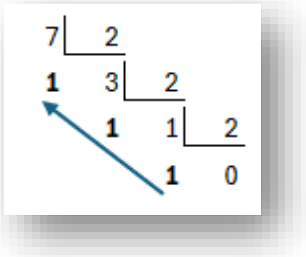
5 en decimal →

5//2 = 2 resto 1

2//2 = 1 resto 0

1//2 = 0 resto 1

→ **101**



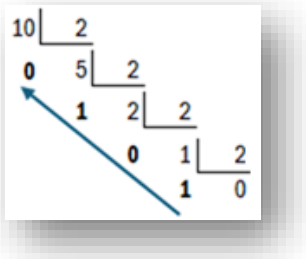
7 en decimal →

7//2 = 3 resto 1

3//2 = 1 resto 1

1//2 = 0 resto 1

→ **111**



10 en decimal →

10//2 = 5 resto 0

5//2 = 2 resto 1

2//2 = 1 resto 0

1//2 = 0 resto 1

→ **1010**

En síntesis: divido por 2, guardo el resto y luego los leemos en orden inverso.

OPERACIONES

SUMA

Cuando sumamos bits en binario y no se produce “acarreo” o carry es súper simple:

0 + 0 = 0 0 + 1 = 1 1 + 0 = 1

Básicamente, si no hay dos 1 al mismo tiempo, la suma como siempre.

SUMA BINARIA CON CARRY

¿Qué pasa si tenemos 1 + 1?

El resultado es 10_2. Esto en decimal sería 2, pero en binario lo representamos como “10”, un 1 con un 0 al lado.

Propagación del acarreo:

Si seguimos sumando 1, ese acarreo puede “viajar” hacia la izquierda.

Por ejemplo, si sumás muchos 1 consecutivos (como 1111 + 1), el acarreo se desplaza un bit en cada paso.

REGLAS DE LA RESTA BINARIA

Cuando restamos en binario, necesitamos ver si hay “préstamo” o borrow.

Caso sin préstamo:

1 – 0 = 1 1 – 1 = 0

Nada complicado:

Si tenés 1 y le restás 1, queda 0

Si tenés 1 y le restás 0, queda 1

Caso con préstamo:

Cuando intentás hacer 0 – 1 no podés porque 0 es menos que 1. Entonces, le “pedís prestado” un 1 a la cifra de la izquierda.

Eso convierte tú 0 en un 10, y ya podés restar.

Es similar a lo que hacemos en decimal cuando le pedís prestado al dígito de la izquierda.

INTRODUCCION AL COMPLEMENTO A 2

A veces restar en binario puede ser un poco enredado cuando tenés que hacer varios préstamos. ¿Cómo no morir en el intento?

Acá entra el complemento a 2 es un método sencillo que transforma la resta en una suma. ¿Mágico? Más bien lógico.

¿Cómo funciona?

1) Asegurate de que ambos números tenga la misma cantidad de bits, ES MUY IMPORTANTE.
2) Invertís todos los bits del número que vas a restas (0 → 1, 1 → 0).
3) Sumás 1 en binario al resultado invertido.
4) Sumás ambos números y listo: Tenés el mismo resultado que si hubieras restado directamente.

Aunque parezca raro, ¡funciona! En informática, este método se utiliza constantemente para representar números negativos además de evitar restas en el sentido que estamos acostumbrados.

EJERCICIO COMPLEMENTO A 2

$$101 - 11$$

$$101_2 = 5_{10}$$

$$11_2 = 3_{10}$$

Si hacemos la resta en decimal es 5 – 3 = 2. Si usamos complemento a 2 y sumamos, ¡El resultado en binario también será 2!

Paso 1: Igualamos la cantidad de bits de ambos números. En este caso usaremos 4, se pone un bit de más.

$$101_2 = 0101_2$$

$$11_2 = 0011_2$$

Paso 2: Invertimos todos los bits (0 → 1, 1 → 0) del que vas a restar:

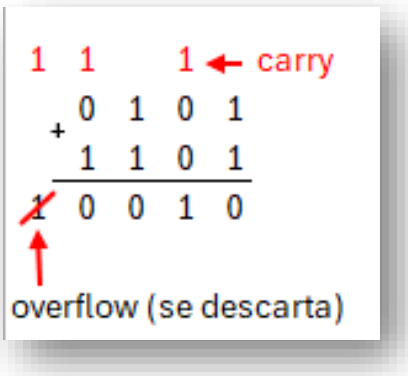
$$0011_2 \rightarrow 1100_2$$

Paso 3: Sumás 1 al invertido:

$$1100_2 \rightarrow 1101_2$$

Paso 4: Luego, sumamos

$$\mathbf{0101_2 + 1101_2}$$



1 + 1 = 0 (con carry)

0 + 0 + sumamos el carry = 1 (sin carry)

1 + 1 = 0 (con carry)

0 + 1 + sumamos el carry = 0 se produce un overflow y el 1 se descarta.

Resultado final = **0010_2** que equivale a 2_{10}

¡FUNCIONA!

INTRODUCCION A LA MULTIPLICACION

Multiplicar en binario es como sumar repetidas veces como la multiplicación en decimal, pero con ceros y unos.

Reglas básicas:

$1 * 1 = 1$ $0 * 1 = 0$

$1 * 0 = 0$ $0 * 0 = 0$

La operatoria es igual a multiplicar en el sistema decimal con varias cifras. Lo haces cifra a cifra, colocando el resultado desplazado una posición. Luego sumás todo.

Ejemplo: 101 x 11

$101_2 = 5_{10}$ $11_2 = 3_{10}$

En binario, hacés:

101 x 11

PASO 1: Multplicás por la primera cifra

$$101 * 1 = 101$$

PASO 2: Luego por la segunda que también es un 1.

$$101 * 1 = 101$$

PASO 3: Sumás los resultados parciales

*		**1**	**0**	**1**
			1	**1**
+		1	0	1
	1	0	1	
	1	**1**	**1**	**1**

$$\mathbf{1111_2 = 15_{10}}$$

MEMORIA BINARIA

CONCEPTO

Imaginate una tira de 8 lámparas. Algunas encendidas, otras apagadas como un bit con 0 y 1.

Si están encendidas de derecha a izquierda la 1, la 3, la 4 y la 7 eso representa el 01001101_2 que equivale a 77_{10}

¿Te animás a calcular el máximo número que puedas guardar en 8 bits?

TIPO INTEGER

Los Integer (o enteros) son números sin parte decimal, tipo 0, 1, 42, -13... Son super comunes en los lenguajes de programación y representan valores dentro de un rango fijo.

Por ejemplo, en 8 bits podés guardar del 0 al 255, porque:

$$2^8 = 256 \text{ valores}$$

Te confunde que llegamos al 255, y es porque consideramos al 0.

¿Cuántos cabrán en un integer de 32 bits?

No respondas todavía, ¡seguí leyendo!

INTEGER SIGNED

Cuando necesitamos guardar negativos o positivos, usamos un bit para indicar el signo.

El bit mas significativo, el de la izquierda, lo define:

0 es positivo

1 es negativo

Ahora, si pensaste que en 32 bits la fórmula a aplicar es 2^{32} te hicimos confundirte, porque usamos un bit para el signo, lo que nos queda para usar 31.

El rango que guarda un integer de 32 bits es:

VALOR MÁX.: 2^{31} - 1 = **2147483647** positivos porque consideramos al 0

VALOR MIN.: 2^{31} = **-21474836478** negativos

El rango total de un integer de 32 bits es:

-2147483648 a 2147483647

TIPO FLOAT

Los float o números de coma flotante son para representar decimales como 3.14159 o -0.007.

-Son útiles en cálculos científicos o medidas precisas.

-Tienen un rango amplio, pero con precisión limitado por bits.

-¡Cuidado! A veces dan resultados “raros” como 0.30000000000000004 al sumar 0.1 tres veces. Esto pasa por cómo se almacenan los números en binario.

CONVERSION DE FRACCIONARIO DECIMAL A BINARIO

Para guardar números decimales como 3.25 en la compu, se usa un formato estándar: el llamado IEEE 754.

IEEE 754

Define cómo transformar la parte entera y la parte fraccionaria en binario.

También especifica un exponente y un bit de signo para manejar números grandes (o muy chiquitos) y negativos.

Vamos a convertir 5.75 a binario

PASO 1: "Tomamos la PARTE ENTERA: 5 y lo convertimos en binario"

Lo dividimos sucesivamente entre 2 anotando el resto (0 o 1), hasta que el cociente sea 0.

5 / 2 = 2 con residuo 1

2 / 2 = 1 con residuo 0

1 / 2 = 0 con residuo 1 (acá terminamos)

Leemos los residuos al revés (del último al primero): 101. Esto significa que la parte entera 5 en binario es:

101

PASO 2: "Tomamos la parte fraccionaria 0.75 y lo convertimos en binario"

Para la fracción 0.75, multiplicamos por 2 y vemos si el resultado es ≥ 1 o no.

Siempre trabajamos con la parte fraccionaria resultante hasta que la misma sea 0.

$$0.75 * 2 = 1.50,$$

¿Es ≥ 1? Sí, entonces tenemos 1. Como queda .50 seguimos

$$0.50 * 2 = 1.00,$$

¿Es ≥ 1? Sí, entonces tenemos 1. Como queda .00 no seguimos

Juntamos los bits, lo que resulta en **11**

$$\mathbf{5.75_{10} = 101.11_2}$$

PASO 3: "Normalizar la mantisa a 1.xxxxx"

La mantisa es la parte que se ajusta para que haya un 1 antes del punto.

Si te quedó algo como 101.11_2, y queremos llegar a $1.xxxx_2$, lo que hacemos es lo que llamamos "flotar el punto" a la izquierda.

Es claro que en decimal cuando movemos el punto, usamos potencias de 10, en binario usamos potencias de 2.

En este caso debés mover 2 lugares, o sea, lo normalizás a

$$\mathbf{101.11 = 1.0111 * 2^2}$$

La parte “1.0111” es la mantisa; el **exponente es 2.**

NOTA IMPORTANTE

Como en la mantisa siempre queda 1.xxxxx, ¿Para que necesitamos guardar el 1 gastando un bit? se descarta quedándonos con xxxxxx

Asi 1.0111 pasa a **0111**

PASO 4: "El exponente se guarda en complemento a 127"

En formato IEEE 754 de 32 bits (float simple precisión), se usa un exponente con un sesgo de 127.

Si tu exponente en decimal es 2, le sumás 127 que pasa a 129. El 129 lo convertís a binario, por ejemplo, 10000001_2 en 8 bits.

El exponente que guardamos es **10000001**

PASO 5: "Determinar el bit de signo"

El bit de signo indica si tu número es positivo (0) o negativo (1). Ojo, acá positivo es 0.

Ejemplo: si tu número es +5.75, bit de signo = 0. Si fuera -5.75, bit de signo = 1.

PASO FINAL: "Armamos el float de 32 bits"

1 bit para el signo (positivo)

8 bits para el exponente (2 en exceso 127)

23 bits para la mantisa (011 completado el resto con 0. Equivale a 1.011)

0 10000001 01110000000000000000000

EJEMPLOS RAROS: 0.1

Al guardar 0.1 decimal a binario hacemos la conversión del .1 a binario, y pasa algo raro...

un decimal exacto genera un periódico en binario.

Veamos:
Para convertir 0.1 decimal a binario por multiplicaciones sucesivas:

0.1 * 2 = 0.2 → bit 0 (porque es < 1). Queda 0.2, seguimos.

0.2 * 2 = 0.4 → bit 0; queda 0.4

0.4 * 2 = 0.8 → bit 0; queda 0.8

0.8 * 2 = 1.6 → bit 1; queda 0.6

0.6 * 2 = 1.2 → bit 1; queda 0.2

... y vemos que el 0.2 regresa, repitiendo el ciclo 0.2 → 0.4 → 0.8 → 0.6 → 0.2...

Por eso, $0.1_{10} = 0.0001100110011..._2$ es periódico porque 1001 se repite indefinidamente.

NERDEADA EN PYTHON: 0.1 + 0.1 + 0.1 = 0.3?

¿Alguna vez hiciste (0.1 + 0.1 + 0.1) en Python y obtuviste 0.30000000000000004?

Si 0.1 en decimal genera un binario periódico, ¡No se guarda en forma exacta! En un float de 32 bits queda así.

```
0 01111011 00110011001100110011001
```

Así el 0.1 no se guarda exactamente en binario: es una aproximación.

En el caso de Python lo guarda en 64 bits y por supuesto también es truncado:

0011111101110011001100110011001100110011001100110011001100110011010

Si lo queremos convertir a decimal, no resulta en 0.1. Es que perdió precisión por ser periódico y guardarse truncado:

$0.1000000000000000005551115123..._{10}$

¡De ahí que la suma se “desvíe” un poquito!

Al sumarlos resulta en:

0.30000000000000004

ES FUNDAMENTAL ENTENDER CÓMO SE ALMACENAN NUESTROS DATOS PARA EVITAR ERRORES AL UTILIZARLOS COMO CONDICIONES EN NUESTROS PROGRAMAS.

USO DE MEMORIA PARA GUARDAR DATOS

Según el lenguaje y la arquitectura.

INTEGER: Pueden ocupar 2 bytes (16 bits), 4 bytes (32 bits) o incluso más. Cuantos más bytes, mayor el rango de valores.

Con 32 bits podés guardar números tan grande como ±2 mil millones.

Es importante conocer el rango, para evitar **overflow** si tu número crece demasiado.

FLOAT: En muchos lenguajes usan 4 bytes (32 bits), pero también existe el double de 8 bytes (64 bits).

Más bits = más precisión para decimales y un rango exponencial más grande.

Con floats, siempre hay riesgo de error de redondeo, así que hay que tener cuidado cuando sumamos números grandes con pequeños.

OVERFLOW

El overflow ocurre cuando tu número supera el límite máximo (o mínimo) que podés representar con esos bits.

Ejemplo (recordá el rango del integer de 32 bits):

2147483647 + 1 = ¡Salgo del rango!

El resultado no va a ser lo que esperás: se vuelve negativo o salta a un valor loco.

En floats, podés obtener “∞” si te vas demasiado arriba.

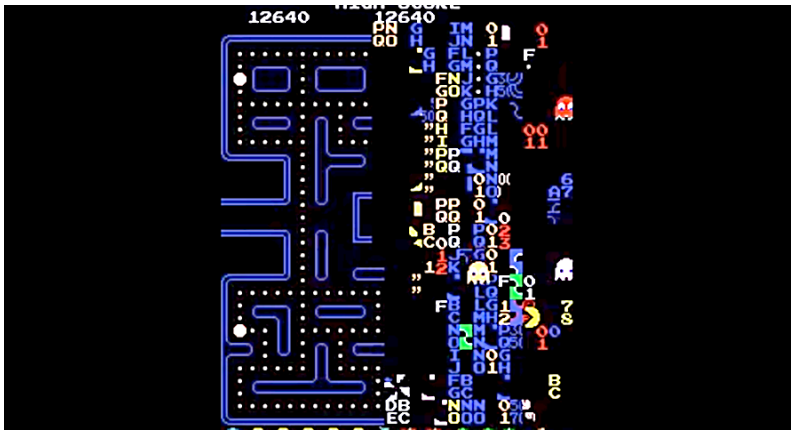
Si intentás provocar overflow en Python compartí el resultado en el foro.

PANTALLA DE LA MUERTE EN PAC-MAN

La famosa “pantalla de la muerte” (o kill screen) del Pac-Man ocurre en el nivel 256. ¿Por qué justo ahí?

Porque el código del juego original usaba una variable de 8 bits para contar los niveles. Eso significa que podía contar desde 0 hasta 255 (en decimal) y, cuando alcanzaba el 256 (que es 1 más que 255), se producía un overflow: la variable volvía a 0, y los gráficos se corrompían por completo.

En lugar de aparecer un nivel normal, el lado derecho de la pantalla se llenaba de símbolos rarísimos, prácticamente imposibles de jugar. Por eso se le dice “pantalla de la muerte”: El juego se vuelve injugable y “muere” en ese punto.



ASCII y ASCII extendido

ASCII (American Standard Code for Information Interchange) es un código que asigna un número a cada carácter. El ASCII clásico define 128 caracteres (de 0 a 127).

La “A” es 65 en decimal, que en binario es 01000001.

Es la base de la codificación de texto en muchas computadoras.

ASCII extendido llega a 256 caracteres (0 a 255) para símbolos adicionales (acentos, ñ, etc.).

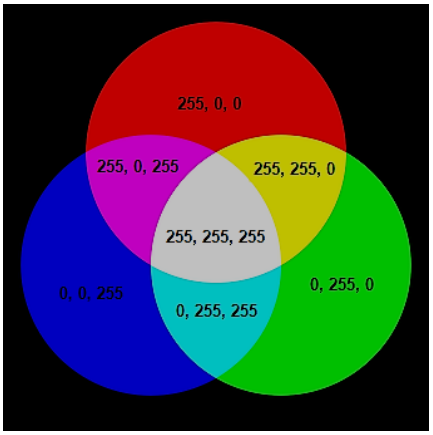
Hoy en día se usa más Unicode (UTF-8), pero el ASCII extendido sigue presente en ciertos contextos.

UN PIXEL EN BINARIO

Una imagen es un arreglo de pixeles.

Un píxel puede representarse con 1 bit (blanco/negro), o con más bits (RGB 24 bits, 32 bits con alfa, etc.).

Básicamente, cada píxel es un conjunto de bits que indican el color que se muestra en pantalla.



Cuantos más bits por píxel, mayor la gama de colores.

Por ejemplo, un monitor gamer utiliza 8 bits por cada canal de color (del 0 al 255) lo que lleva a representar más de 16,7 millones de colores.

Sin embargo, se han anunciado modelos que utilizan 10 bits. Sería el caso del monitor LG UltraGear que es el de mayor resolución (5120 x 2160 pixeles) disponible a enero del 2025.