Tarea Independiente 13/10/2025

David Núñez Franco

October 16, 2025

Inventario de Conceptos Claves

- Promise y la VM de JS
- Problema encadenar promesas (then, catch)
- Iteradores/Iterables en JS
- Symbol, Symbol.iterator
- Iterables, Iteradores, Generadores (function* yield)
- Async/await function como Generador de Promises
- Introducción a Prolog, muy breve historia
- Uso básico de SWI-Prolog, archivos .pl shell swipl
- Cargar un archivo, ejemplo: [holamundo].
- SWI-Prolog es compilado dinámicamente
- LP como una variante de FP usando solo predicados (relaciones).
- Relación como generalización de función (muchos-muchos versus uno-uno)
- Predicado como relación bidireccional
- Datos básicos: átomo (entre comillas simples o sin comillas si es un identificador) versus variable (empieza con mayúscula) en Prolog

Ejercicio 0

> "Despulgue" async.js y haga un caso de prueba para las dos versiones print_json y print_json_async

Solución simplificada

```
// Version con then/catch
function print_json_then(user = 1) {
    fetch(`https://jsonplaceholder.typicode.com/todos/${user}`)
    .then((res) => res.json())
    .then((json) => console.log(json))
    .catch((err) => console.error(err));
}

// Version async/await
async function print_json_async(user = 1) {
    try {
        const res = await fetch(
        `https://jsonplaceholder.typicode.com/todos/${user}`
        );
        const json = await res.json();
        console.log(json);
    } catch (err) {
        console.error(err);
    }
}

// Casos de prueba simples
print_json_then(1);
print_json_async(2);
```

Listing 1: simplifiedVersion.js

Solución depurada con ayuda de IA

```
// print_json con then/catch
export function print_json_then(
user = 1,
{ fetchFn = fetch, logger = console } = {}
) {
    const url = `https://jsonplaceholder.typicode.com/todos/${user}`;
    return fetchFn(url)
    .then((res) => {
        if (!res.ok) throw new Error(`HTTP ${res.status}`);
        return res.json();
    })
    .then((json) => {
        logger.log(json);
        return json;
```

```
        })
        .catch((err) => {
            logger.error(err);
            throw err;
        });
}
```

Listing 2: print_json_then.mjs

```
// version async/await equivalente
export async function print_json_async(
user = 1,
{ fetchFn = fetch, logger = console } = {}
) {
    const url = `https://jsonplaceholder.typicode.com/todos/
       ${user}`;
    try {
        const res = await fetchFn(url);
        if (!res.ok) throw new Error(`HTTP ${res.status}`);
        const json = await res.json();
        logger.log(json);
        return json;
    } catch (err) {
        logger.error(err);
        throw err;
    }
}
```

Listing 3: print_json_async.mjs

```
import { print_json_then } from "./print_json_then.mjs";
import { print_json_async } from "./print_json_async.mjs";

console.log("\n--- Version con then/catch ---");
await print_json_then(1);

console.log("\n--- Version async/await ---");
await print_json_async(2);

console.log("\n--- error esperado, usuario inexistente ---")
   ;
try {
    await print_json_async(9999); // 404
} catch (e) {
    console.log("OK, error capturado:", e.message);
}
```

Listing 4: test_print_json.mjs

Ejercicio 2

Estudie el API de Promise y en particular los combinadores any, Promise.all, Promise.race, Promise.try. Haga ejemplos con all, any, race usando estos tres URIs:

https://jsonplaceholder.typicode.com/posts https://dummyjson.com/products/1 https://dummyjson.com/products

Solución

```
const URLS = [
"https://jsonplaceholder.typicode.com/posts",
"https://dummyjson.com/products/1",
"https://dummyjson.com/products",
];

// funcion auxiliar que encapsula el patron repetido de hacer una peticion
// fetch y convertir la respuesta a JSON, con manejo basico de errores HTTP
const fetchJson = (url) =>
fetch(url).then((r) => {
    if (!r.ok) throw new Error(`HTTP ${r.status} @ ${url}`);
    return r.json();
});

// cuando se usan combinadores como Promise.any o Promise.race, solo se
// recibe el valor de la promesa ganadora. COn labeled se sabe de que
// URL vino el resultado
const labeled = (url) => fetchJson(url).then((data) => ({ url, data }));

// ALL: espera a todas, rechaza si una falla.
export async function demoAll() {
    const [posts, product1, products] = await Promise.all(URLS.map(fetchJson));
    console.log("ALL ok:", {
        posts: posts.length,
        product1: product1.id,
        products: products.products?.length ?? products.length,
    });
}

```

```
30          // ANY: primera que cumpla, ignora rechazos hasta que una
                cumpla
31          export async function demoAny() {
32              const first = await Promise.any(URLS.map(labeled));
33              console.log("ANY first ok from", first.url);
34          }
35
36          // RACE: primera que se resuelva o rechace
37          export async function demoRace() {
38              try {
39                  const first = await Promise.race(URLS.map(labeled));
40                  console.log("RACE first settled ok from:", first.url
                        );
41              } catch (e) {
42                  console.log("RACE rejected first:", e.message);
43              }
44          }
45
46          // "PROMISE.try" equivalente
47          export function promiseTry(fn) {
48              return Promise.resolve().then(fn);
49          }
50
51          // Uso de "try"
52          export async function demoTry() {
53              const val = await promiseTry(() => JSON.parse('{"x":1}')
                    );
54              console.log("TRY ok:", val);
55              try {
56                  await promiseTry(() => JSON.parse("{bad}"));
57              } catch (e) {
58                  console.log("TRY error capturado:", e.message);
59              }
60          }
61
62          await demoAll();
63          await demoAny();
64          await demoRace();
65          await demoTry();
```

Ejercicio 3

Escriba un función delayedApply(f, sec) que retorna una lambda g talque g(x) es una promesa que se resolverá no antes de sec segundos en el valor f(x).

Solución

```
export function delayedApply(f, sec) {
    return function (x) {
        return new Promise((resolve, reject) => {
            try {
                setTimeout(() => resolve(f(x)), sec * 1000);
            } catch (e) {
                reject(e);
            }
        });
    };
}

const double = (n) => n * 2;
const g = delayedApply(double, 2);

console.log("Inicio...");
g(5).then((v) => console.log("Resultado:", v));
```

Ejercicio 4

Escriba en JavaScript (imperativo) un generador asíncrono complaint(sec) que espera sec segundos antes de producir la primera hilera "Ay" y luego, cada sec segundos, produce una nueva hilera con una "y" más al final de la anterior: "Ayy", "Ayyy", "Ayyyy", etc. El generador debe producir infinitamente (hasta que el consumidor lo detenga).

Solución

```
async function* complaint(sec) {
    let s = "Ay";
    while (true) {
        await new Promise((r) => setTimeout(r, sec * 1000));
        yield s;
        s += "y";
    }
}

async function main(max = 10) {
    const gen = complaint(2);
    for await (const s of gen) {
        console.log(s);
        if (s.length > max) break;
    }
```

```
        }

        main();
```

Ejercicio 5

```
En Prolog: Indique en cada caso si es un átomo o una variable o algo distinto de ambas
a) Hola
b) _Hola
c) '_Hola'
d) '"hola que tal"'
e) _xyz
```

Solución

- a) Hola = Variable (Empieza con mayuscula)
- b) _Hola = Variable (Empieza con guion bajo)
- c) '_Hola' = Átomo (Entre comillas simples -¿ siempre átomo)
- d) '"hola que tal"' = Átomo (Entre comillas simples)
- e) _xyz = Variable (Empieza con guion bajo)

Ejercicio 6.a

```
Problema: "Contar cuantas cuentas tiene una persona dada"
Añada y trate de entender esto a nuestro holamundo.pl:

/* Obtiene la lista de cuentas de la persona Person */
accountsOf(Person, Accounts) :- findall(tuple(Person, Account), person_account(Person,
/*
Donde en SWI-Prolog
findAll( Tuple, Query, ListOfTuples) encuentra la lista ListOfTuples de todos los tupl
Piense como en un select Tuple from Query.
Las listas en Prolog usan la misma notación de arrays en JS. Ejemplos
[juan, maria, pedro] es una lista de tres átomos.
[[juan, 20], [maria, 21], [pedro, 15]] es una lista de 3 listas de largo dos c/u.
[] es la lista vacía
*/
Definamos
/* Cuenta cuántas cuentas N tiene la Persona */
```

```
howManyAccounts(Person, N) :- accounts(Person, Accounts), length(Accounts, N).
/*
length(L, N) es para relacionar una lista A y su largo N.
La coma es un AND
*/

Pruebe en el shell de swipl (CMD en la carpeta donde está holamundo.pl) ese código.
```

Solución

```
/*
@author: david
*/

% Juan puede tener mas de una cuenta bancaria (relacion 1-N)
% Maria comparte algunas cuentas con Juan
person_account(juan, '200-ABC').
person_account(juan, '200-QWR').
person_account(juan, '100-ABC').
person_account(maria, '200-ABC').
person_account(maria, '100-RST').

/* Obtiene la lista de cuentas de la persona Person */
accountsOf(Person, Accounts) :-
    findall(tuple(Person, Account), person_account(Person,
       Account), Accounts).

/* Cuenta cuantas cuentas N tiene la Persona */
howManyAccounts(Person, N) :-
    accountsOf(Person, Accounts),
    length(Accounts, N).
```

Listing 5: holamundo.pl

```
?- [holamundo].
true.

?- accountsOf(juan, A).
A = [tuple(juan, '200-ABC'), tuple(juan, '200-QWR'), tuple(
   juan, '100-ABC')].

?- howManyAccounts(juan, N).
N = 3.

?- howManyAccounts(maria, N).
N = 2.
```

Listing 6: Pruebas en swipl

Ejercicio 6.b

Reto: Dos personas P y Q son, coowners(P, Q), si existe una cuenta A que es cuenta de
Escriba ese predicado. Siga este seudo código:

coowners(P, Q) :- P tiene una cuenta A y Q tiene esa misma cuenta A y P y Q no son la

En Prolog la coma es el AND, el punto y coma es el OR.
En Prolog el operador de negación es \= (equivale != de JS/JAVA).

Solución

```
1       /* Dos personas P y Q son coowners si comparten al menos una
            cuenta */
2       coowners(P, Q) :-
3       person_account(P, A),
4       person_account(Q, A),
5       P \= Q.
```

Listing 7: holamundo.pl

```
1       ?- coowners(P, Q).
2       P = juan,
3       Q = maria .
```

Listing 8: output

Ejercicio 6.c

Añada un predicado gender(P, G) para indicar que la persona P tiene género G (male, fe

Solución

```
1       /* Indicar el genero de cada persona */
2       gender(juan, male).
3       gender(maria, female).
```

Listing 9: holamundo.pl

```
?- [holamundo].
true.

?- gender(juan, G).
G = male.

?- gender(maria, G).
G = female.

?- gender(P, female).
P = maria.
```

Listing 10: output

Ejercicio 6.d

Añada balance(Acc, Amount) para decir que la cuenta Acc tiene un balance Amount(este

Solución

```
/* Indica el balance (monto) de cada cuenta */
balance('200-ABC', 1500).
balance('200-QWR', 3200).
balance('100-ABC', 800).
balance('100-RST', 1200).
```

Listing 11: holamundo.pl

```
?- balance('200-ABC', Monto).
Monto = 1500.

?- balance(Cuenta, 1200).
Cuenta = '100-RST'.

?- findall(C, balance(C, _), Lista).
Lista = ['200-ABC', '200-QWR', '100-ABC', '100-RST'].

?- findall(C, balance(C, 1500), Lista).
Lista = ['200-ABC'].

?- findall(C, balance(C, 800), Lista).
Lista = ['100-ABC'].

?- findall(C, balance(C, 3200), Lista).
```

```
17          Lista = ['200-QWR'].
18
19          ?- findall(C, balance(C, 0), Lista).
20          Lista = [].
```

Listing 12: output