CLASE 4 · UNIDAD I · RA1

Del problema al algoritmo

Diseñar, implementar y validar. Un algoritmo que no se prueba no está terminado: está escrito.

RECONEXIÓN · 18:30

“La clase pasada entregaron un código que funcionaba. ¿Cómo saben que estaba correcto?”

1

Lo ejecutaron una vez

Con los datos que ustedes mismos eligieron, que son justo los que el programa maneja bien.

2

Nadie probó el borde

¿Qué pasa con la lista vacía? ¿Con un cliente repetido? ¿Con un monto negativo?

3

Eso es lo de hoy

El salto de “funciona” a “lo verifiqué”: diseño, implementación y validación.

TRAMO 2 · 18:45 · CIERRE DEL CONTENIDO PENDIENTE

El diccionario como mapa clave-valor

```php
<?php
declare(strict_types=1);

$stock = [
    'A-100' => 12,
    'B-205' => 0,
    'C-330' => 7,
];

// Acceso directo por clave: no recorre nada
echo $stock['B-205'];      // 0

foreach ($stock as $sku => $cant) {
    printf("%-6s %d\n", $sku, $cant);
}
```

La clave es el dato

En una lista, la posición es un número sin significado. En un diccionario, la clave es parte del problema: el SKU, el RUT, la fecha.

Orden de inserción

El array de PHP conserva el orden en que se insertaron las claves. No es automáticamente alfabético ni numérico.

TRAMO 2 · DICCIONARIOS

Tres trampas que hay que ver ejecutadas

1

isset() miente con null

Si el valor guardado es null, isset() dice que la clave no existe.

```
$c = ['debug' => null];

isset($c['debug']);
// false

array_key_exists('debug', $c);
// true
```

2

Clave inexistente

Leer una clave que no está emite un warning y devuelve null. Con ?? se resuelve.

```
echo $c['idioma'];
// Warning + null

echo $c['idioma']
     ?? 'es';
// es
```

3

PHP normaliza las claves

'1', 1 y true son la MISMA clave. Las tres asignaciones se pisan entre sí.

```
$r['1']  = 'a';
$r[1]    = 'b';
$r[true] = 'c';

count($r);
// 1
```

TRAMO 2 · TREINTA SEGUNDOS, NO MÁS

Dos formas de preguntar “¿existe?”

Recorriendo la lista

```
in_array('C-330',
    array_keys($stock), true);
// revisa uno por uno
```

Preguntando por la clave

```
isset($stock['C-330']);

// va directo
```

Las dos devuelven true.

Al final de la clase vamos a medir cuánto cuesta cada una — y por qué esa diferencia crece con la cantidad de datos.

TRAMO 3 · 19:15 · DISEÑO DE ALGORITMOS

Antes de escribir código: tres preguntas

ENTRADA

¿Qué datos recibo y en qué forma vienen?

PROCESO

¿Qué transformación tengo que hacer, paso a paso?

SALIDA

¿Qué debo devolver exactamente, y en qué estructura?

El pseudocódigo no es un trámite

Es el lugar barato para equivocarse. Corregir una decisión en papel cuesta un borrón; corregirla después de escribir cien líneas cuesta media hora.

TRAMO 3 · DEL PAPEL AL CÓDIGO

Caso guiado: total vendido por día

Una tienda registra cada venta con su fecha y su monto. Se necesita el total vendido por día.

PSEUDOCÓDIGO

```
FUNCIÓN totalPorDia(ventas)
  totales ← diccionario vacío
  PARA CADA venta EN ventas
    fecha ← venta.fecha
    SI fecha NO existe EN totales
      totales[fecha] ← 0
    FIN SI
    totales[fecha] ←
      totales[fecha] + venta.monto
  FIN PARA
  DEVOLVER totales
FIN FUNCIÓN
```

PHP 8.5

```php
function totalPorDia(
    array $ventas): array
{
    $totales = [];
    foreach ($ventas as $v) {
        $f = $v['fecha'];
        $totales[$f] =
          ($totales[$f] ?? 0) + $v['monto'];
    }
    return $totales;
}
```

`($totales[$f] ?? 0) + ...` reemplaza al `SI no existe ENTONCES 0` del pseudocódigo: la lógica es la misma, la escritura es más corta.

TRAMO 3 · 19:50 · 25 MINUTOS

Taller · Un problema por persona

Primero el pseudocódigo en papel. Recién cuando esté escrito se abre 3v4l.

1

Control de stock

Lista de movimientos con SKU y cantidad (puede ser negativa).

```
SALIDA:  sku => cantidad final
```

2

Registro de asistencia

Lista de marcas con alumno y estado (presente / ausente).

```
SALIDA:  alumno => [presente, ausente]
```

3

Promedio de notas

Lista de notas por estudiante. Aprobado desde 4,0.

```
SALIDA:  alumno => [promedio, aprobado]
```

Los tres problemas tienen la misma forma

Recorrer una lista y acumular en un diccionario. Si lo descubren solos al comparar sus soluciones, mejor todavía.

BREAK

TRAMO 4 · 20:30 · VALIDACIÓN FUNCIONAL

El código funciona… hasta que no

```php
<?php
declare(strict_types=1);

function promedio(array $notas): float
{
    return array_sum($notas) / count($notas);
}

echo promedio([6.0, 5.5, 4.0]);  // 5.1666...
echo promedio([]);               // ?
```

`DivisionByZeroError`

count([]) es 0. La función nunca contempló ese caso, y nadie lo probó.

Ahora ustedes

¿Cuántos de sus algoritmos del taller anterior sobreviven a una lista vacía? Pruébenlo ahora.

Los cinco casos que siempre hay que pensar

Vacío · Uno solo · Repetidos · Límite (cero) · Inválido

TRAMO 4 · EL INSTRUMENTO ANTES DE LA HERRAMIENTA

La tabla de casos de prueba

#	CASO	ENTRADA	SALIDA ESPERADA	OBTENIDA	¿OK?
1	Lista vacía	`[]`	`[]`		
2	Un solo elemento	`[ una venta ]`	`esa venta íntegra`		
3	Dos del mismo día	`[ 1000, 500 ]`	`1500 en esa fecha`		
4	Valor cero	`[ monto 0 ]`	`la fecha con 0`		
5	Valor negativo	`[ monto -200 ]`	`total descontado`		

Consigna: cada uno completa esta tabla para su propio algoritmo, con mínimo cuatro casos y el caso vacío obligatorio.

Esta tabla es evidencia del criterio "validación funcional" del programa: casos de prueba, datos variados, verificación de resultados.

TRAMO 4 · 20:50 · TALLER

Un verificador casero, en veinte líneas

```
function verificar(
    string $caso, mixed $esperado, mixed $obtenido): void
{
    $ok = $esperado === $obtenido;
    printf("[%s] %s\n", $ok ? ' OK ' : 'FALLA', $caso);
    if (!$ok) {
        echo '  esperado: ', var_export($esperado, true), "\n";
        echo '  obtenido: ', var_export($obtenido, true), "\n";
    }
}

verificar('lista vacía devuelve arreglo vacío',
    [], totalPorDia([]));

verificar('dos ventas del mismo día se suman',
    ['2026-09-09' => 1500], totalPorDia($dosVentas));
```

`===` y no `==`

Con ==, PHP compara después de convertir tipos: '1' == '01' es true. Un verificador con == puede aprobar código incorrecto.

Y en la Unidad 4…

Esto mismo, hecho por PHPUnit o Pest. Cuando lleguemos ahí no va a ser magia: ya lo escribieron a mano.

Consigna: mínimo 4 casos para su propio algoritmo. Si uno falla, corregir el algoritmo y anotar qué se corrigió.

TRAMO 5 · 21:20 · EFICIENCIA BÁSICA

¿Cuánto cuesta? Primera medición

CÓMO SE MIDE

```php
$inicio = hrtime(true);   // nanosegundos
// ... lo que se quiere medir ...
$fin = hrtime(true);

printf("%.3f ms\n",
    ($fin - $inicio) / 1_000_000);
```

DEMO A · BUSCAR EN 20.000 ELEMENTOS

```php
$lista = range(1, 20_000);
$mapa  = array_flip($lista);
$objetivo = 20_000;   // el peor caso

in_array($objetivo, $lista, true);
isset($mapa[$objetivo]);
```

La lista duplica su tiempo
cada vez que duplicamos los datos

El diccionario no se mueve:
siempre va directo a la clave

Repetir con 20.000, 40.000 y 80.000 elementos, y anotar los tiempos. Eso es evaluación empírica.

TRAMO 5 · DEMO B

La deuda de la clase 2: array_pop vs array_shift

```
$a = range(1, 20_000);
$t0 = hrtime(true);
while ($a) { array_pop($a); }     // saca del final
$t1 = hrtime(true);

$b = range(1, 20_000);
$t2 = hrtime(true);
while ($b) { array_shift($b); }   // saca del inicio
$t3 = hrtime(true);
```

array_pop

Saca el último y no toca nada más.

array_shift

Saca el primero y reindexa todo el arreglo. Hacen lo mismo, pero no cuestan lo mismo.

Conexión con la clase 3: la cola implementada con array_shift paga ese costo cada vez que se atiende a un cliente.

TRAMO 5 · HONESTIDAD METODOLÓGICA

Los números no van a dar iguales dos veces

3v4l es un servidor compartido: la misma medición repetida entrega valores distintos. Eso no invalida la medición, pero obliga a dos reglas.

1 Mirar el orden de magnitud

Ejecutar varias veces y comparar escalas, no decimales. ¿Es el doble? ¿Es cien veces? Esa es la información útil.

2 Contar operaciones

Un contador dentro del ciclo es determinista: da lo mismo el servidor, la hora o la versión. Es la métrica que se puede defender.

```
$ops = 0;
foreach ($lista as $item) { $ops++; if ($item === $objetivo) break; }
echo "Operaciones: $ops\n";   // 20000 en el peor caso, 1 con el diccionario
```

PUESTA EN COMÚN · 22:05

Defiendan su algoritmo

Cinco minutos cada uno. Los otros dos escuchan como revisores.

1. ¿Qué decisiones fueron clave en el diseño de tu algoritmo?
2. ¿Qué dificultades surgieron al pasar del pseudocódigo al código?
3. ¿Cómo se aseguraron de que el algoritmo resolviera correctamente el problema?

Regla para los revisores: cada uno debe aportar un caso de prueba que el expositor no consideró.

CIERRE · 22:20

Un algoritmo tiene tres partes, no una

DISEÑO

Entrada, proceso, salida.

En papel, antes del código.

→

IMPLEMENTACIÓN

La traducción a PHP.

La parte más corta de las tres.

→

VALIDACIÓN

Casos de prueba.

Sin esto, no está terminado.

Evidencias que deben quedar hoy

- La hoja con el pseudocódigo (foto sirve)
- Enlace de 3v4l del algoritmo funcionando
- Enlace de 3v4l del verificador con sus casos
- La tabla de casos de prueba completada

Próxima clase · cierre de la Unidad 1

Optimización y refactorización: tomamos uno de los algoritmos de hoy, buscamos su cuello de botella y lo mejoramos, comparando la versión original con la optimizada.