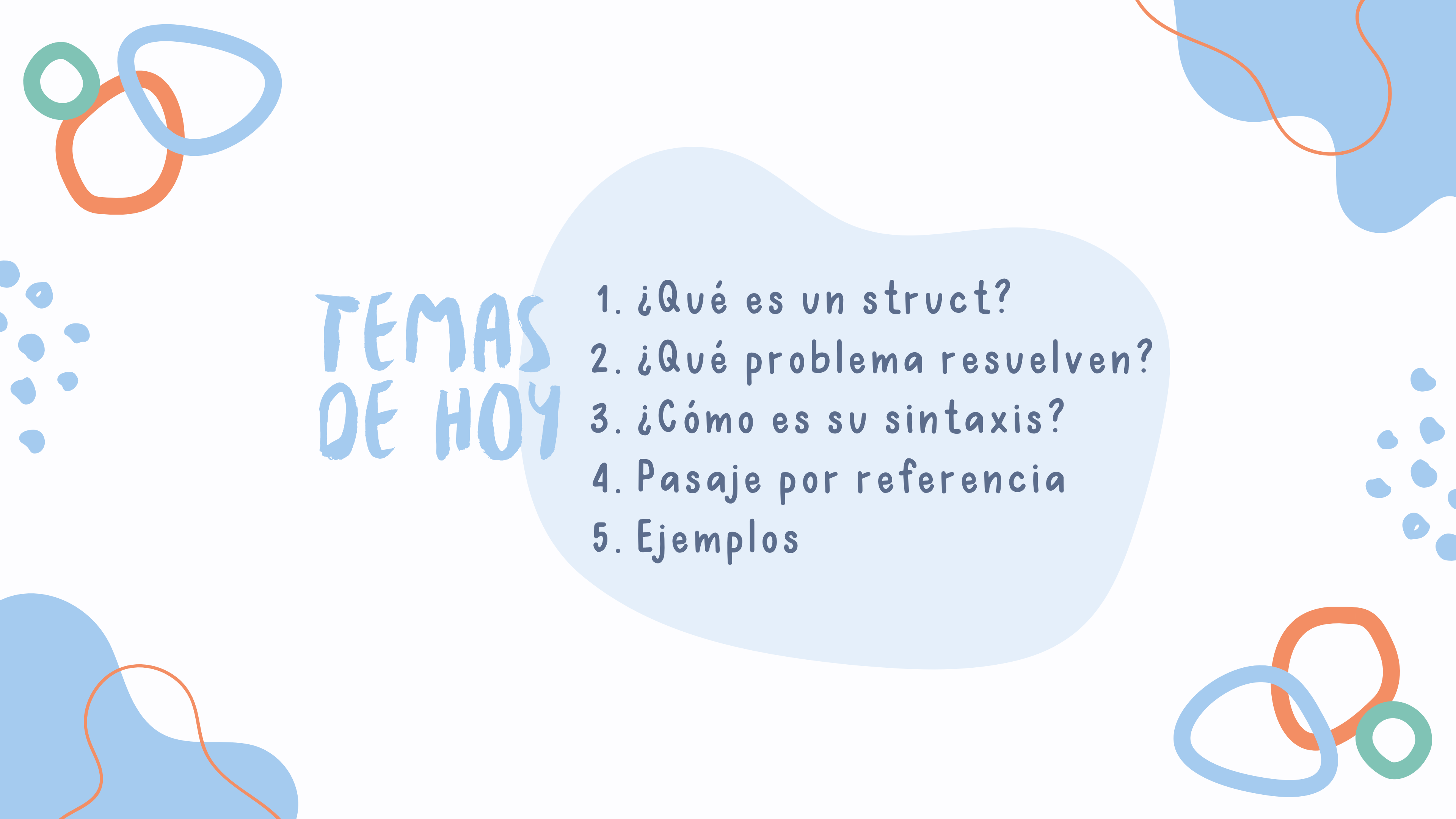




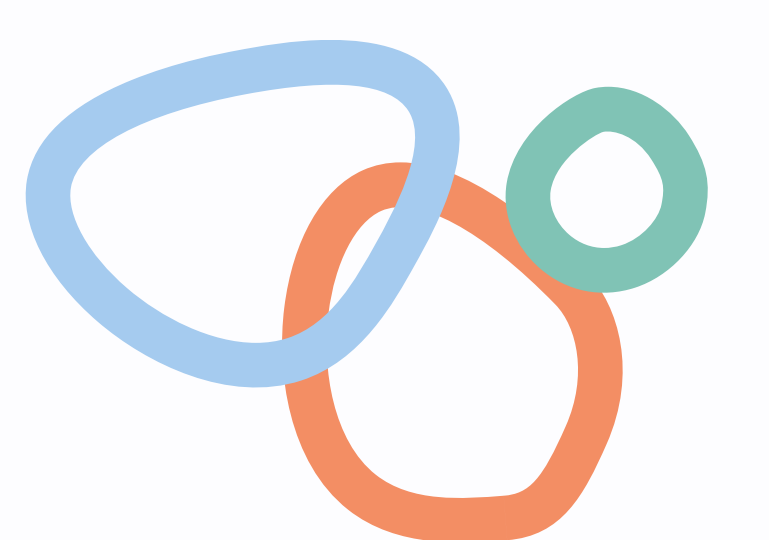
# STRUCTS

¿Qué son y qué resuelven?



# TEMAS DE HOY

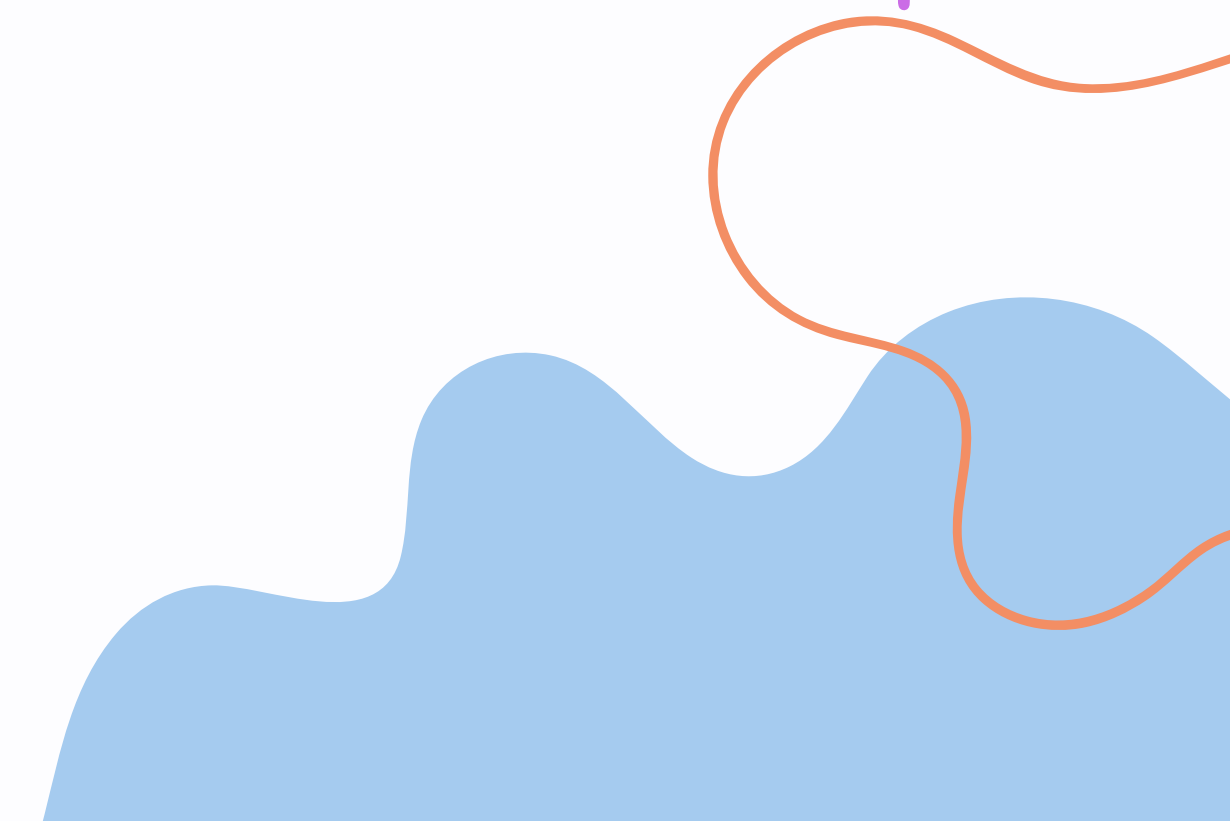
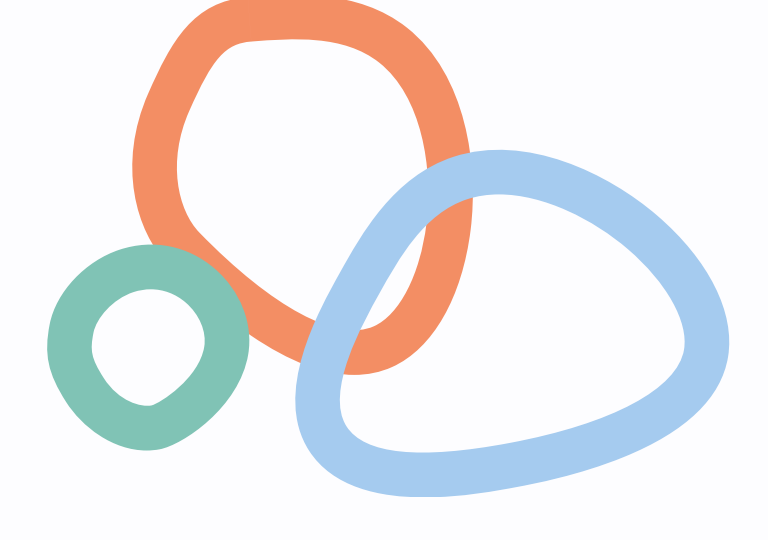
1. ¿Qué es un struct?
2. ¿Qué problema resuelven?
3. ¿Cómo es su sintaxis?
4. Pasaje por referencia
5. Ejemplos



# LO QUE SABÍAMOS HASTA AHORA...



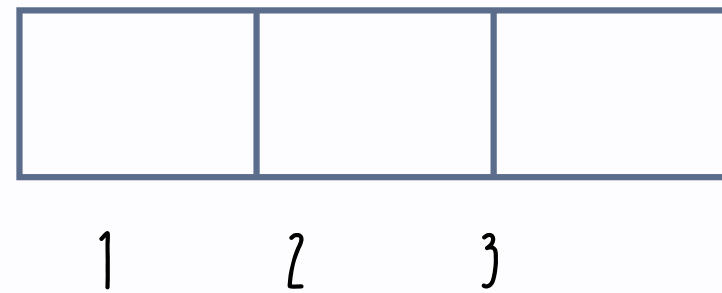
Con variables sueltas podíamos guardar un dato a la vez.  
Con **vectores**, podíamos guardar muchos **datos del mismo tipo**.



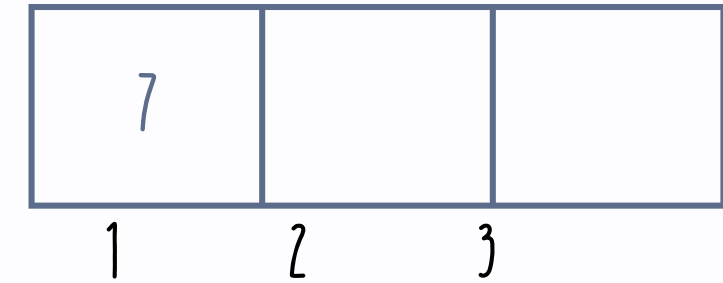
# LO QUE SABIAMOS HASTA AHORA...

Por ejemplo,  
si quisieramos guardar las notas de un alumno:

INT NOTAS[3]



NOTAS[0] = 7



Un vector agrupa **datos**, pero todos  
tienen que ser del **mismo tipo**.

# PROBLEMA

¿Qué pasa si quiero combinar diferentes tipos de datos?

Si yo quisiera guardar en un solo lugar no solo la nota de un Alumno, sino también su padrón, promedio..

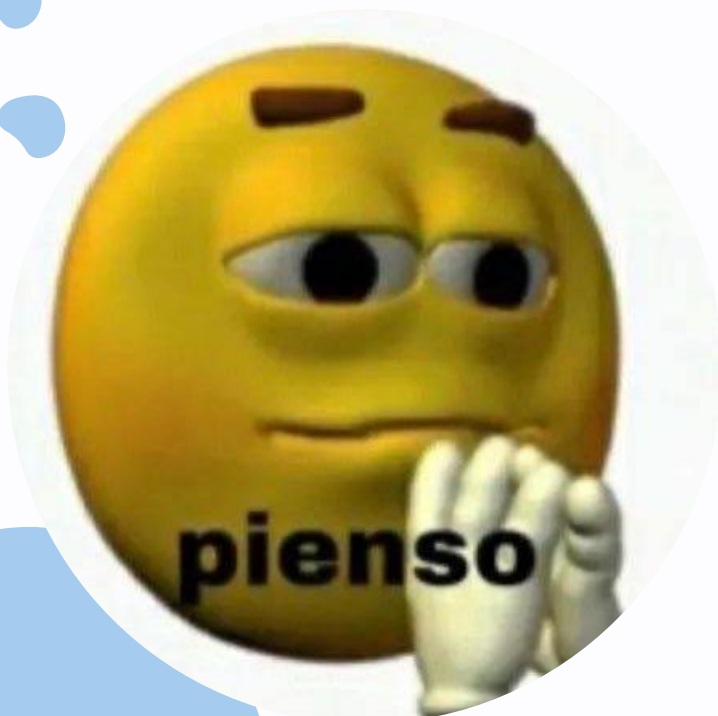
¿puedo usar un vector?

|   |        |      |      |
|---|--------|------|------|
| 7 | 114888 | 7.75 | True |
| 1 | 2      | 3    | 4    |

# PROBLEMA

¿Qué pasa si quiero combinar diferentes tipos de datos?

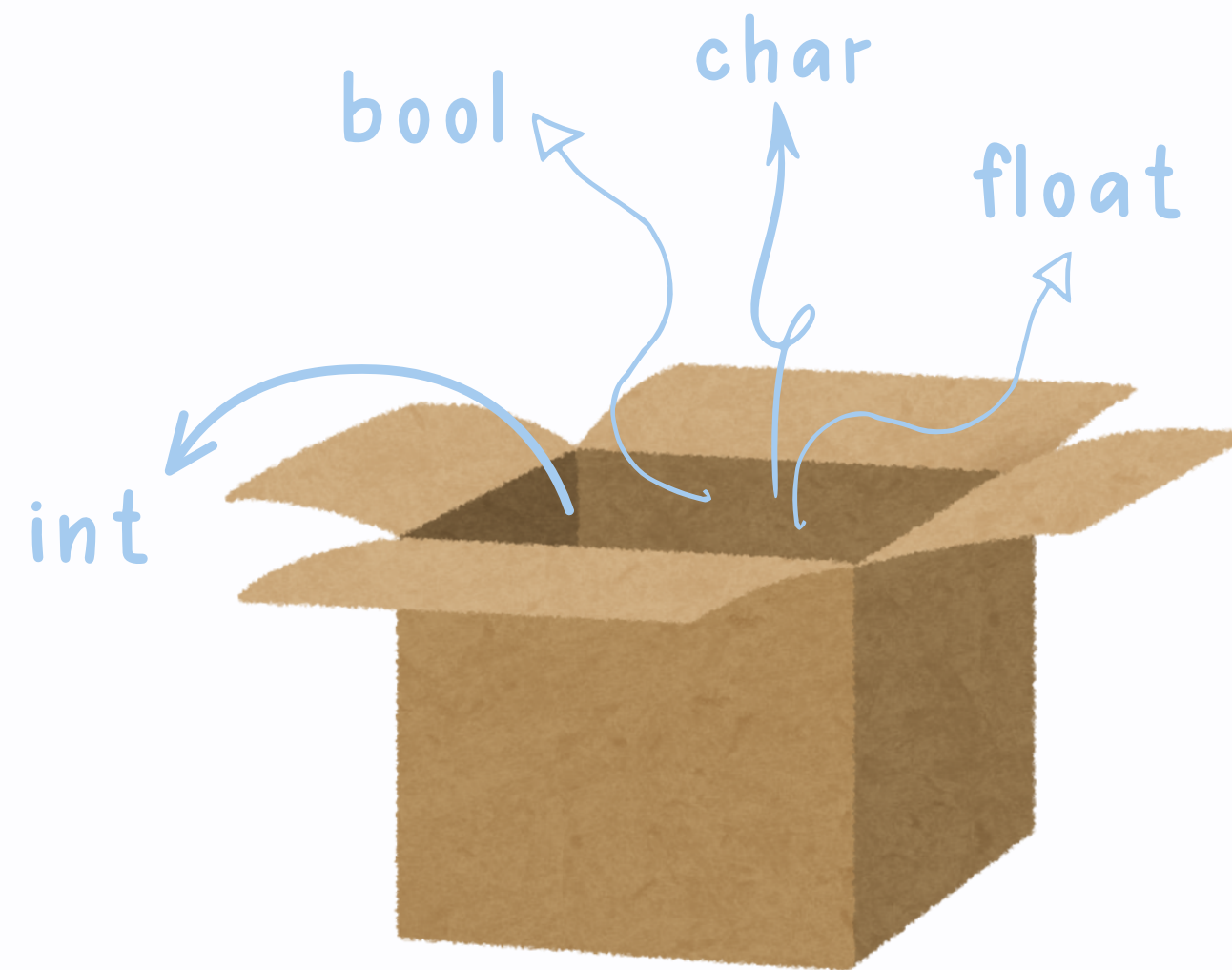
|   |        |      |      |
|---|--------|------|------|
| 7 | 114888 | 7.75 | True |
| 1 | 2      | 3    | 4    |



Un **vector** no puede **mezclar** diferentes tipos de datos como booleanos, enteros, decimales..

# SOLUCIÓN: STRUCTS

Un struct en C es una estructura de datos que **permite agrupar diferentes tipos de datos** bajo un mismo nombre.



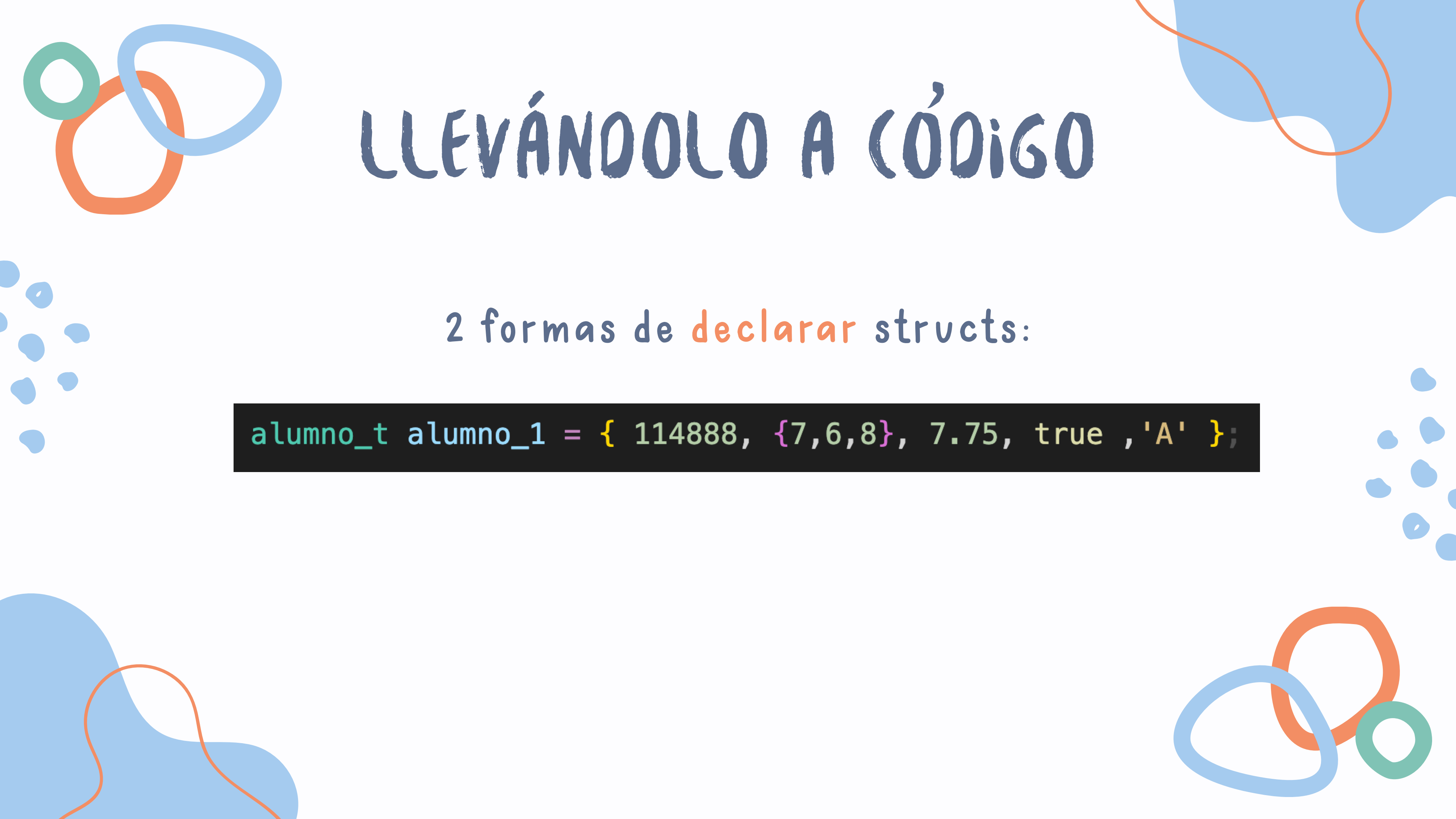
# LLEVÁNDOLO A CÓDIGO

```
#include <stdio.h>
#include <stdbool.h>

typedef struct alumno{
    int legajo;
    int notas[3];
    float promedio;
    bool trabaja;
    char division;
} alumno_t;
```

Ahora tenemos bajo una misma estructura toda la información que queremos sobre un Alumno. Pero...  
*¿cómo accedo a cada dato?*

OBS: el `typedef` sirve para darle un "alias" al struct alumno, en este caso `alumno_t`



# LLEVÁNDOLO A CÓDIGO

2 formas de **declarar** structs:

```
alumno_t alumno_1 = { 114888, {7,6,8}, 7.75, true , 'A' };
```

# LLEVÁNDOLO A CÓDIGO

nombre\_variable.campo

```
alumno_t alumno_1;  
alumno_1.legajo = 114888;  
alumno_1.notas[0] = 7;  
alumno_1.notas[1] = 6;  
alumno_1.notas[2] = 8;  
alumno_1.promedio = 7.75;  
alumno_1.trabaja = false;  
alumno_1.division = 'A'
```

Al declarar `alumno_t alumno_1;` estoy declarando una variable como cualquier otra. Para acceder a los campos de la misma simplemente uso `.`

OBS: `alumno_T` va a funcionar como cualquier otro tipo de dato.

# LEYENDO LOS DATOS

```
printf("El alumno se encuentra en la división: %c \n",
      alumno_1.division);

printf("Las notas del alumno dentro del cuatrimestre
fueron: ");

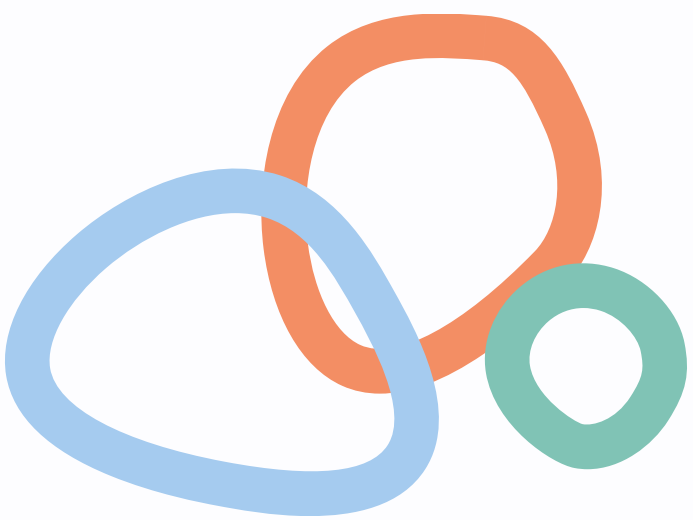
for (int i = 0; i < MAX_NOTAS; i++) {
    printf("%i \n", alumno_1.notas[i]);
}
```



## acceso a campos

- variable: `alumno.promedio`
- puntero : `alumno->promedio`

## paso por valor vs. referencia

- valor: copia
  - referencia: **modifica** el original
- 

# PASAJE POR REFERENCIA

```
void calcular_promedio(alumno_t *alumno) {  
    int suma = 0;  
    for (int i = 0; i < MAX_NOTAS; i++) {  
        suma += alumno->notas[i];  
    }  
    alumno->promedio = (float) suma / MAX_NOTAS;  
}
```

- Uso del operador `->`: cuando `alumno_t *alumno` es un **puntero**,
- Se usa `alumno->promedio` en lugar de `alumno.promedio`. Esto es equivalente a `(*alumno).promedio`.

# PASAJE POR REFERENCIA

```
calcular_promedio(&alumno_1);
```

- Paso por referencia (&alumno\_1): se envía la dirección de memoria de alumno\_1 (alumno\_t), permitiendo **modificarlo** sin copiar la estructura.



¿PREGUNTAS?

# EJEMPLOS

