Trabajo en clase Macros. Explicar el funcionamiento de los siguientes códigos (1 snippet por diapositiva), identificando tópicos vistos en la clase de hoy. Como arreglos, macros, referencias, etc.

```
%macro print_int 1
   mov eax, 4
   mov ebx, 1
   mov ecx, %1
   mov edx, 4
   int 0x80
%endmacro

section .data
   array dd 1, 2, 3, 4, 5

section .text
   global _start
```

este fragmento de codigo definición de la macro print_int ss una **macro con un argumento** (%1) que representa la dirección del entero que queremos imprimir.
La macro expande instrucciones que realizan una llamada al sistema sys_write y declara un arreglo de 5 enteros.

```
%macro print_int 1
   mov eax, 4
   mov ebx, 1
   mov ecx, %1
   mov edx, 4
   int 0x80
%endmacro
```

En este fragmento se define una **macro** que recibe **1 parámetro**.
Una macro en ensamblador funciona como un *copiar-pegar inteligente*:
cuando la llamas, NASM sustituye esa línea por todas las instrucciones dentro de la macro

Imprime un entero en pantalla usando la llamada al sistema `sys_write`.

```
section .data
   array dd 1, 2, 3, 4, 5
```

Define un **arreglo de 5 enteros (32 bits)** donde cada elemento ocupa **4 bytes** que posteriormente se usará en la siguiente parte del programa para calcular una suma..

dd = *define doubleword* = 4 bytes

Entonces `array` ocupa **5 × 4 = 20 bytes**

Tópicos involucrados

Arreglos en ensamblador
Direcciones consecutivas en memoria
Etiqueta como referencia a dirección base
array es un puntero al primer elemento.

```
_start:
   mov ecx, 0
   mov eax, 0

bucle:
   mov ebx, [array + ecx*4]
   add eax, ebx

   inc ecx
   cmp ecx, 5
   jl bucle

   print_int eax

   mov eax, 1
   xor ebx, ebx
   int 0x80
```

este bloque recorre el arreglo y suma sus elemento.
Es el proceso de inicialización, se lee el elemento array[ecx] y se multiplica ecx *4 porque cada entero ocupa 4 bytes

```
_start:
   mov ecx, 0
   mov eax, 0
```

ecx se usa como índice para recorrer el arreglo y eax se utilizara como acumulador para acumular la suma y empezará en 0

```
bucle:
   mov ebx, [array + ecx*4]
   add eax, ebx
```

en esta parte se lee el elemento array[ecx] y se multiplica ecx *4 porque ocupa 4 bytes y add eax, abx suma el valor leído al acumulador eax.

```
inc ecx
cmp ecx, 5
jl bucle
```

incrementa el indice y compara ecx con el tamaño del arreglo y se repite mientras ecx sea menor a 5, en este bloque se recorre el arreglo y se suman sus elementos.

```
print_int eax

mov eax, 1
xor ebx, ebx
int 0x80
```

este bloque llama a la macro y se intenta imprimir el contenido el registro eax pasando el valor directamente y finalmente se finaliza el programa llamando al sys_exit