

Cohetería Computacional

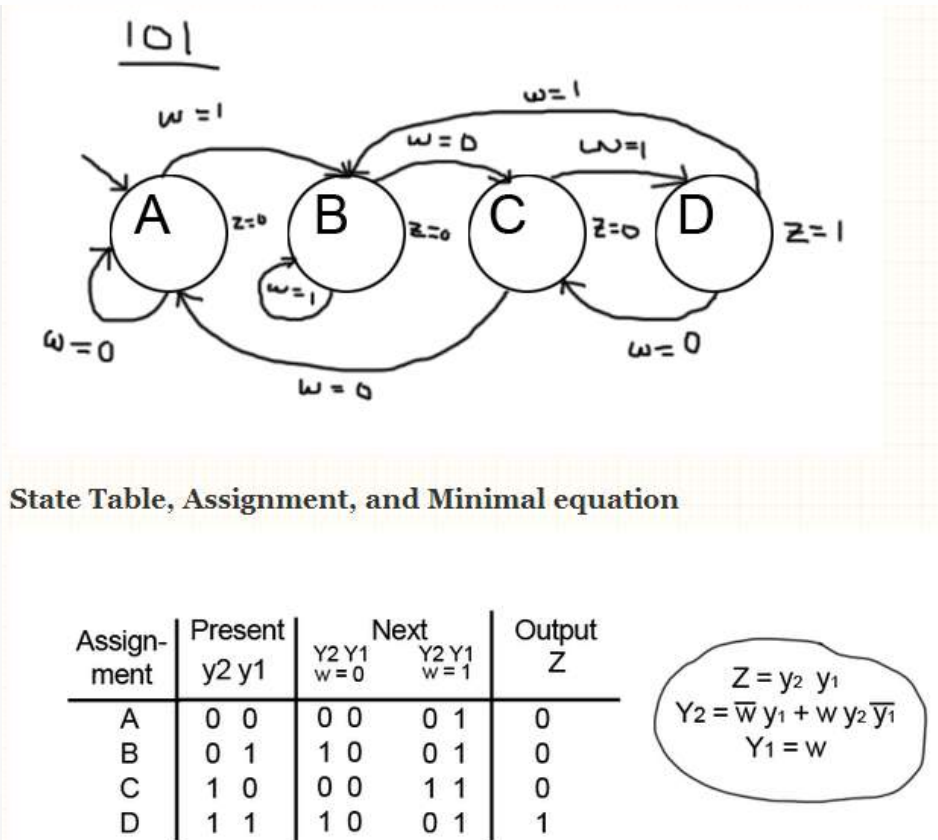
Conocimiento general

- Teoría de computación
- Kahoot
- Ejemplos con Python

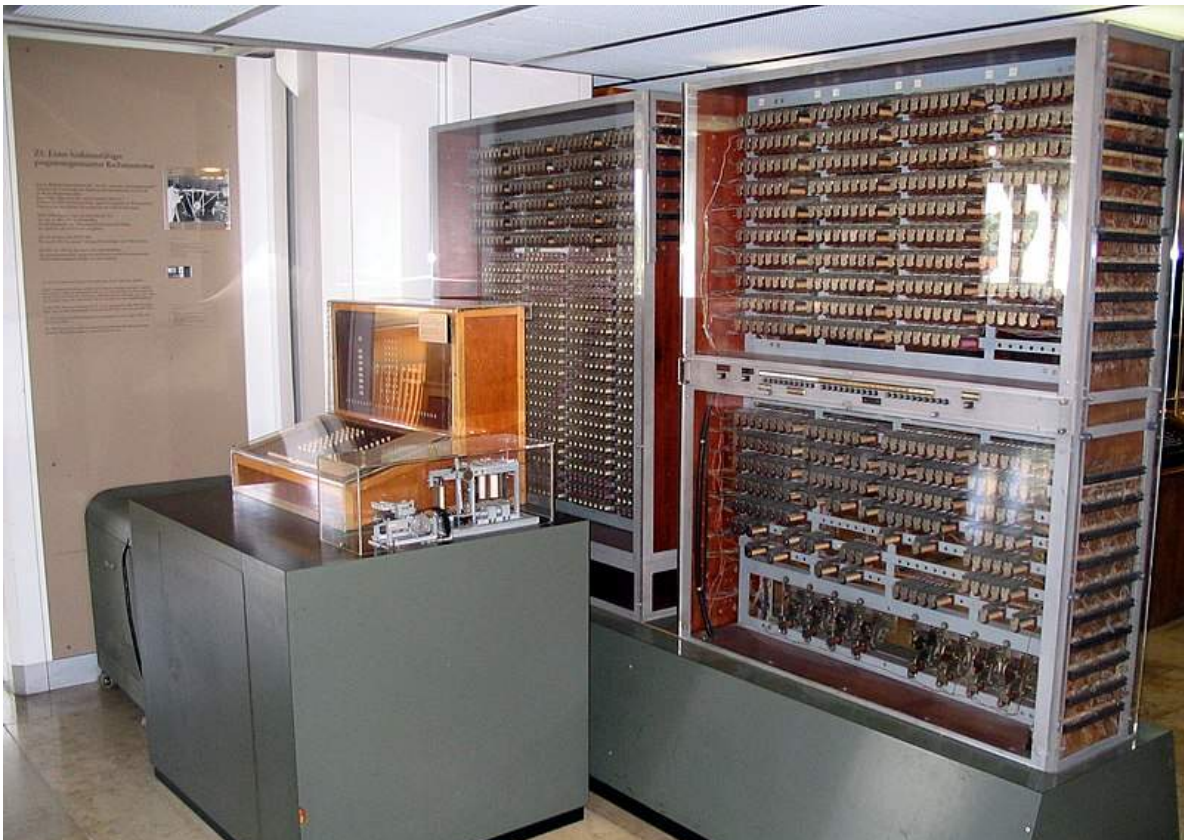
Luis Ross

Lo más básico

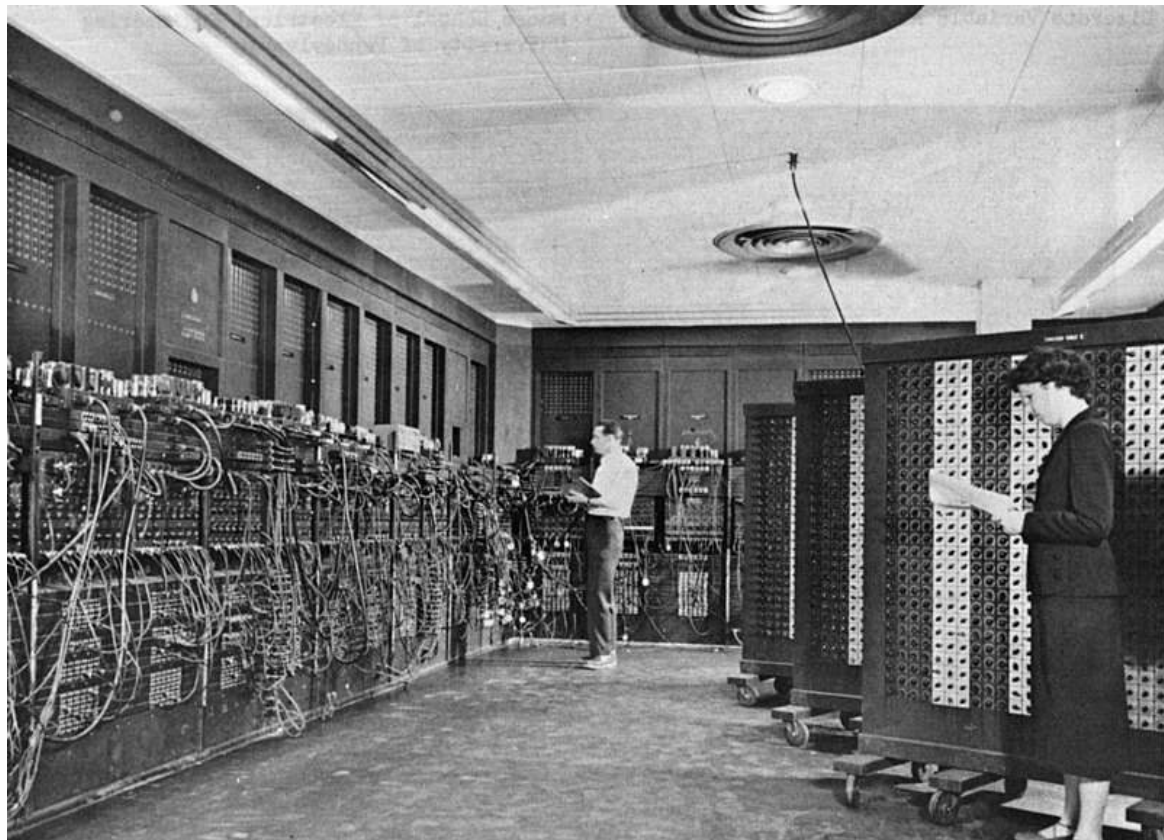
Al inicio de los tiempos



Maquina de Turing



Electromecánicas (Z3)



Electrónicas (ENIAC)

Lenguajes para la computadora

```
1000 A9 02  
1002 69 05  
1004 8D A0 0F  
1007 60
```

Lenguaje de máquina 6502

```
; suma 2 + 5 a la memoria  
0005 .BA $1000  
0100 LDA #$02  
0110 ADC #$05  
0120 STA $0FA0  
0130 RTS  
0140 .EN
```

Assembly 6502

Lenguajes de alto nivel (lo común)

```
program main
```

```
  implicit none
```

```
  integer :: valor
```

```
  valor = 2 + 5      ! suma 2 + 5
```

```
end program main
```



Fortran 77+

```
int main()
```

```
{
```

```
  int valor;
```

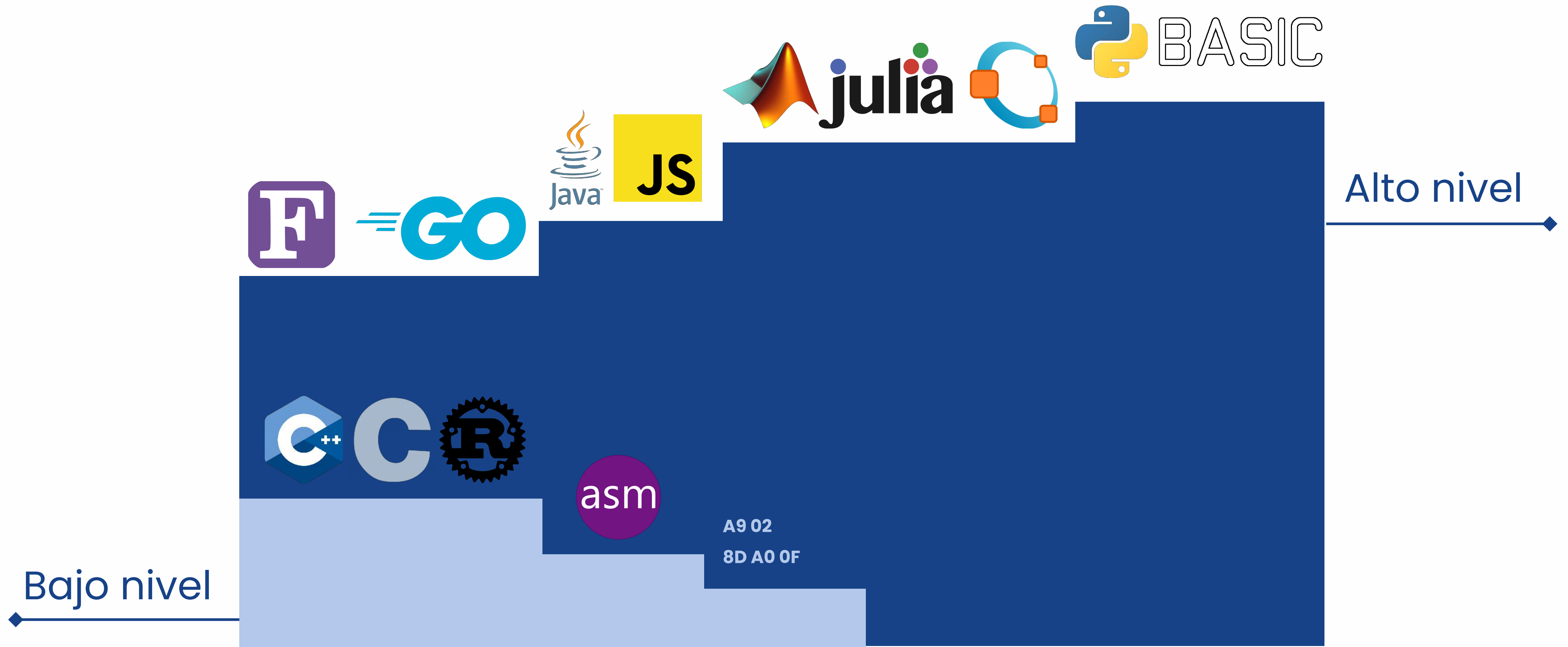
```
  valor = 2 + 5;      /* suma 2 + 5 */
```

```
  return(0);
```

```
}
```



ANSI C 89+



Nivel de abstracción

```
valor = 2 + 5    # suma 2 + 5
```



```
int main()  
{  
    int valor;  
    valor = 2 + 5;    /* suma 2 + 5 */  
    return(0);  
}
```



Paradigmas de programación

Paradigmas básicos

program main

print *, 'esta linea viene primero'

print *, 'luego esta linea'

100 print *, 'esta linea se repite por siempre'

call sleep(1)

go to 100

end program main



Concurrente

program main

print *, 'esta linea viene primero'

print *, 'luego esta linea'

do while(.TRUE.)

print *, 'esta linea se repite por siempre'

call sleep(1)

end do

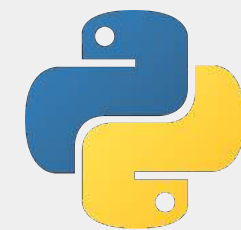
end program main



Estructurado

Funcional (idea general)

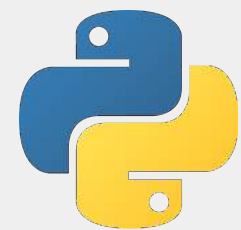
```
# se llama la funcion mayor()  
print (mayor(1,2))
```



```
# se pueden combinar orden de cálculos  
print (mayor(mayor(1,2),5))
```

Usando función pura

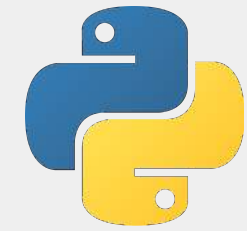
```
def mayor(num1, num2):  
    if num1 > num2:  
        mayor = num1  
    else:  
        mayor = num2  
    return mayor
```



Rutina de la función

Funcional (recursión)

```
def potencia(x, n, cont=0):  
    if n-1 > cont:  
        return x*potencia(x,n,cont+1)  
    elif n < cont:  
        return 1.0/(potencia(x,-n,cont))  
    elif n == 0:  
        return 1  
    else:  
        return x
```



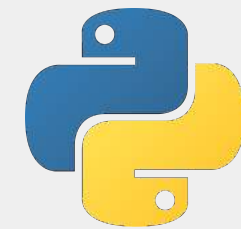
Función recursiva

La función calcula potencia de la forma $y(x,n) = x^n$

No se usan estructuras de ciclo do/for/while

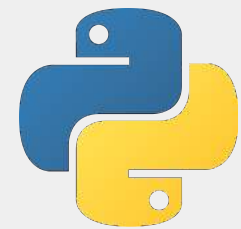
Funcional (de primera clase)

```
def saludo(nombre):  
    return "Hola "+nombre  
def despedida(nombre):  
    return "Adios "+nombre  
def gritar(nombre):  
    return nombre.upper()+"!"  
  
def hablar(funcion, nombre): # funcion parametro  
    print(funcion(nombre))
```



Algunas funciones

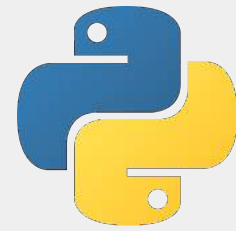
```
hablar(saludo,"Juan")  
hablar(gritar,"Juan")  
hablar(despedida,"Juan")
```



Las funciones pueden ser
pasadas a otra función
porque son ciudadanas de
primera clase

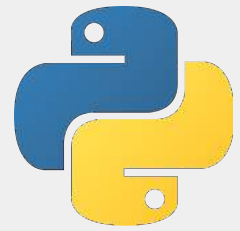
Orientado a objetos

```
# instanciar objeto cuete de la clase cohete
cuete = cohete()
print(cuete.getPeso())      # leer valor inicial
cuete.setPeso(300)         # actualizar valor
print(cuete.getPeso())      # leer valor actualizado
```



Usar métodos de objeto

```
class cohete:
    def __init__(self, peso = 500)
        self.peso = peso
    def setPeso(self, peso)
        self.peso = peso
    def getPeso(self)
        return(self.peso)
```



Clase con métodos

Un poco más de objetos

```
# dos objetos de clases distintas
```

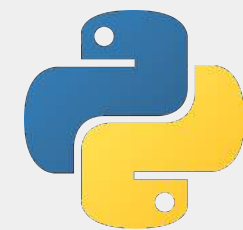
```
Juan = persona()
```

```
Joel = perro()
```

```
# usan el mismo nombre de método
```

```
Juan.hablar()    # dice hola
```

```
Joel.hablar()    # ladra
```



Polimorfismo

```
class pesona:
```

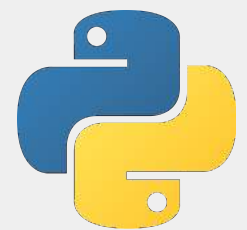
```
    def saludar(self)
```

```
        print("Hola")
```

```
class perro:
```

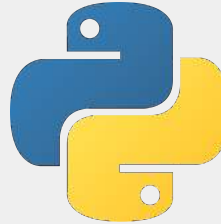
```
    def saludar(self)
```

```
        print("**ladra**")
```

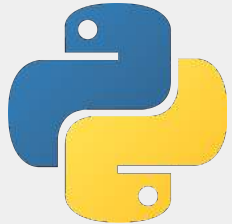


Clases distintas

Orientado a objetos (más)

```
class motor:  
    def __init__(self, propelente="no asignado"):   
        self.propelente = propelente  
    def setPropelente(self, propelente):  
        self.propelente = propelente  
    def getPropelente(self):  
        return self.propelente
```

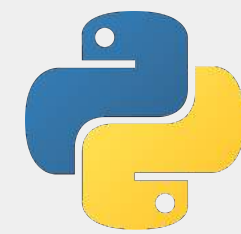
Clase padre

```
class cohete(motor):   
    def __init__(self, propelente="no asignado",  
nombre="sin nombre"):  
        motor.__init__(self, propelente)  
        self.nombre = nombre  
    def setNombre(self, nombre):  
        self.nombre = nombre  
    def getNombre(self):  
        return self.nombre
```

Clase con herencia

Orientado a objetos (ya)

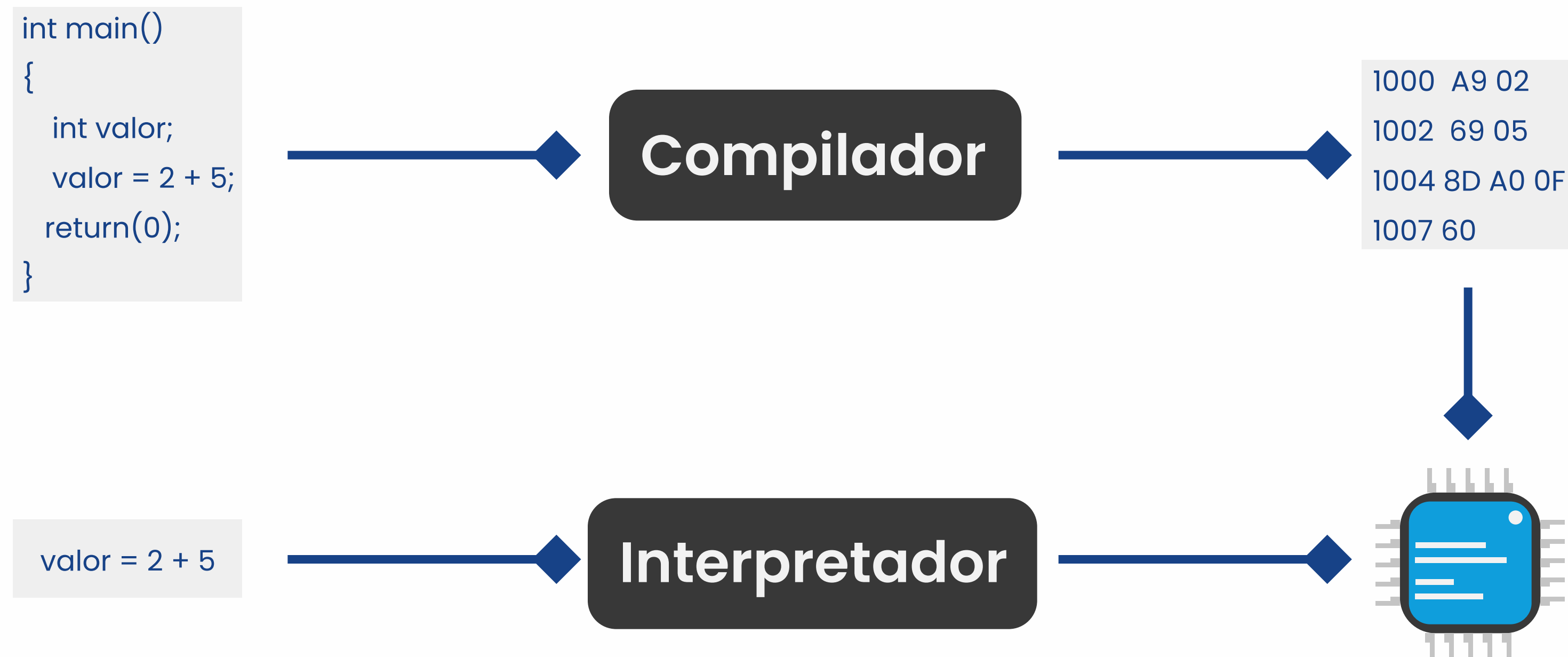
```
cohete1 = cohete(propelente="proplelente1")  
print(cohete1.getPropelente())  
cohete1.setPropelente("propelente2")  
print(cohete1.getPropelente())  
print(cohete1.getNombre())  
cohete1.setNombre("Miravalles 1")  
print(cohete1.getNombre())
```



Se pueden reutilizar métodos y atributos sin tener que especificar más datos

Interpretados y compilados

Diferencia fundamental



Diferencias específicas

Interpretado

- sintaxis más simple
- mayor portabilidad
- menor rendimiento

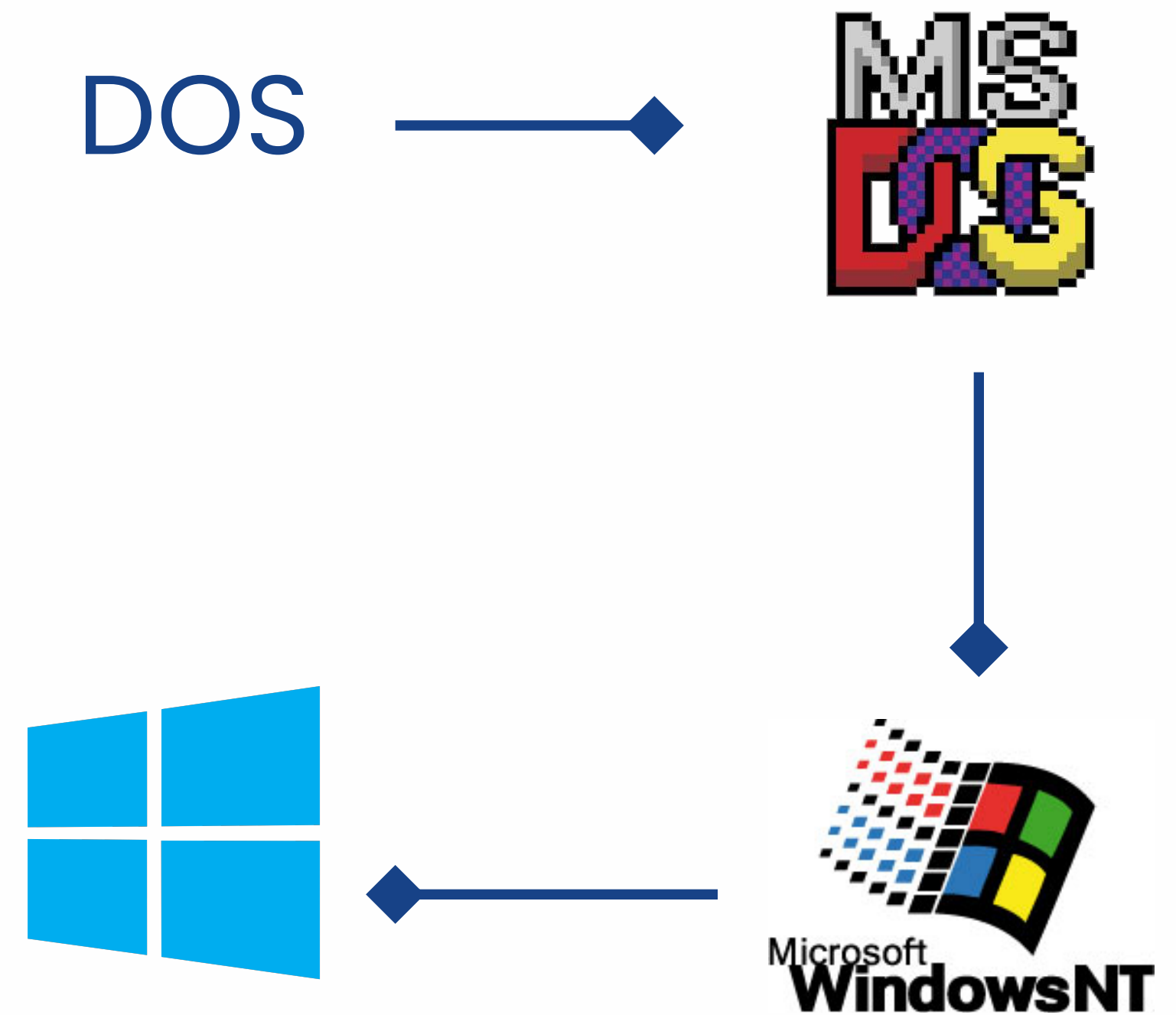
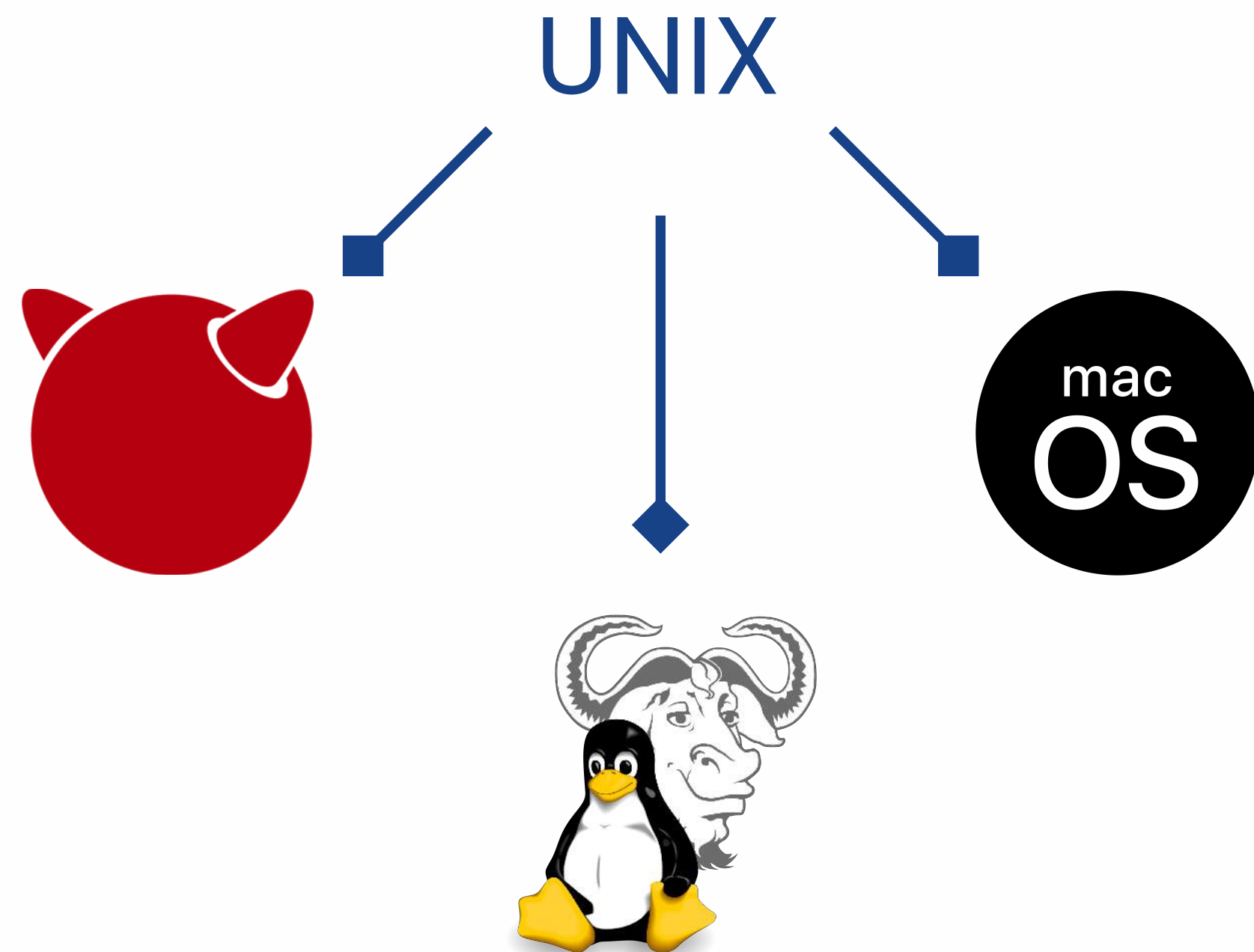
Compilado

- sintaxis más completa
- mejor rendimiento
- debe considerar la plataforma deseada

*Manejo de memoria

Plataformas (portabilidad 1)

so (os) de escritorio



Navegadores web



Chromium



WebKit



Gecko

JS

SO Móviles

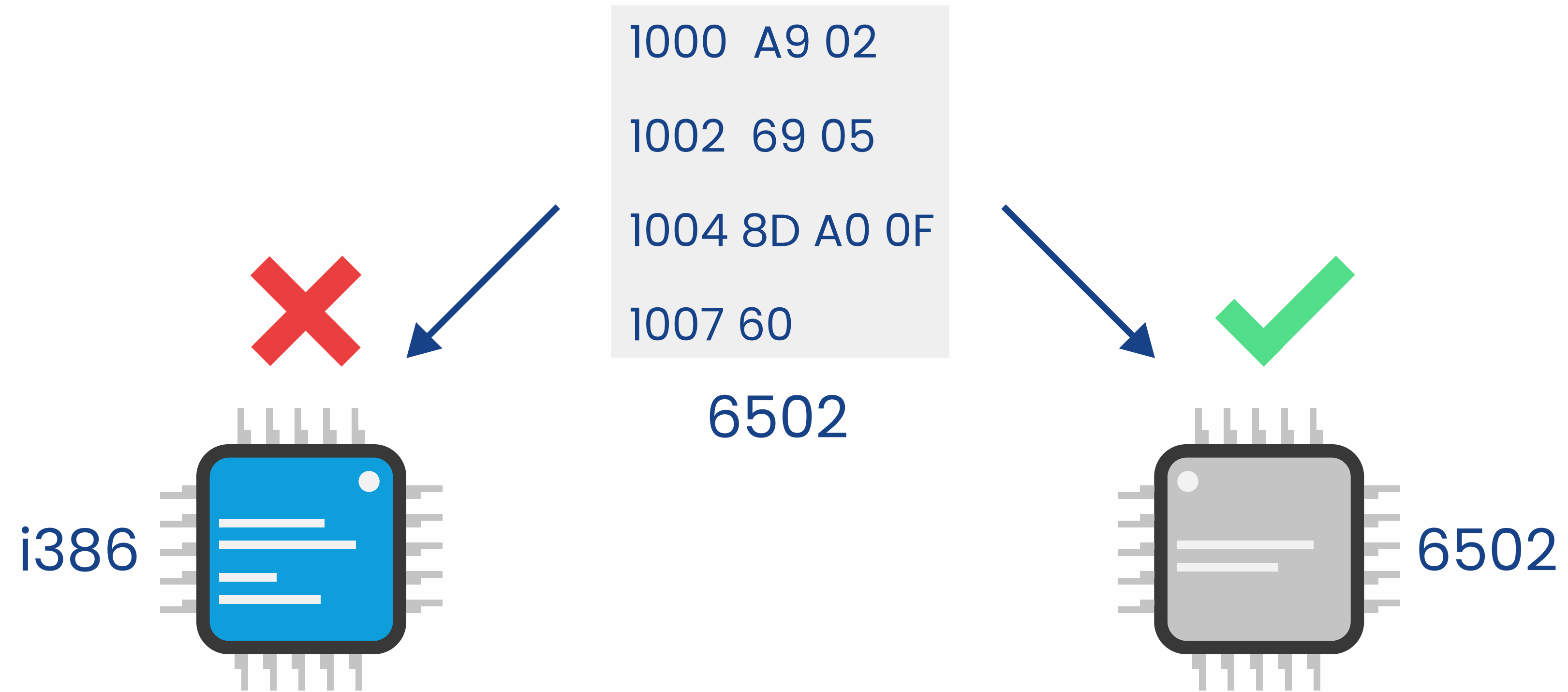


android



Arquitecturas (portabilidad 2)

Set de instrucciones



Algunas arquitecturas

6502

PowerPC

x86

x64

ARM

Atmel AVR



Próxima clase, taller de Python

Revisar

- Instalar python
python.org/downloads
- Editar archivo del ejemplo para que guarde la velocidad y el peso en un archivo de texto
(revisar documentos de la clase en Teams)