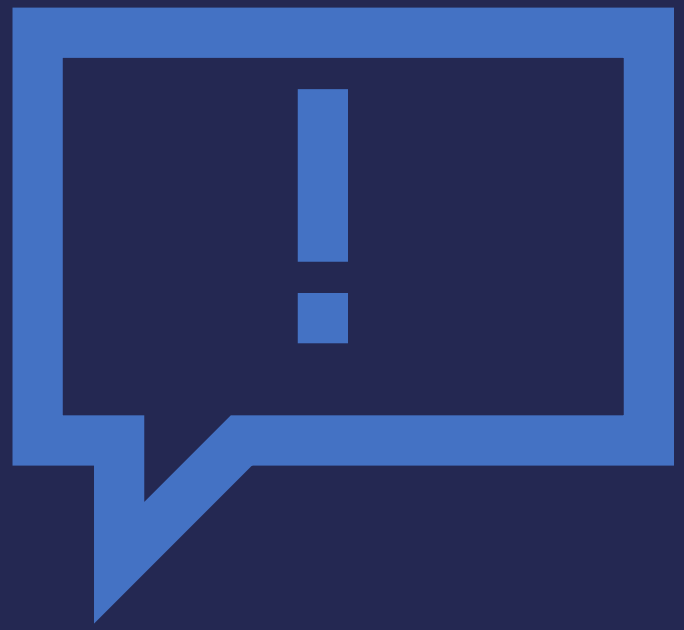
TEMA 9 - PARADIGMAS DE PROGRAMACIÓN

PARADIGMA

- Todo aquel modelo, patrón o ejemplo que debe seguirse en determinada situación.
- En un sentido amplio, se refiere a una teoría o conjunto de teorías que sirve de **modelo** a seguir para resolver problemas o situaciones determinadas que se planteen.

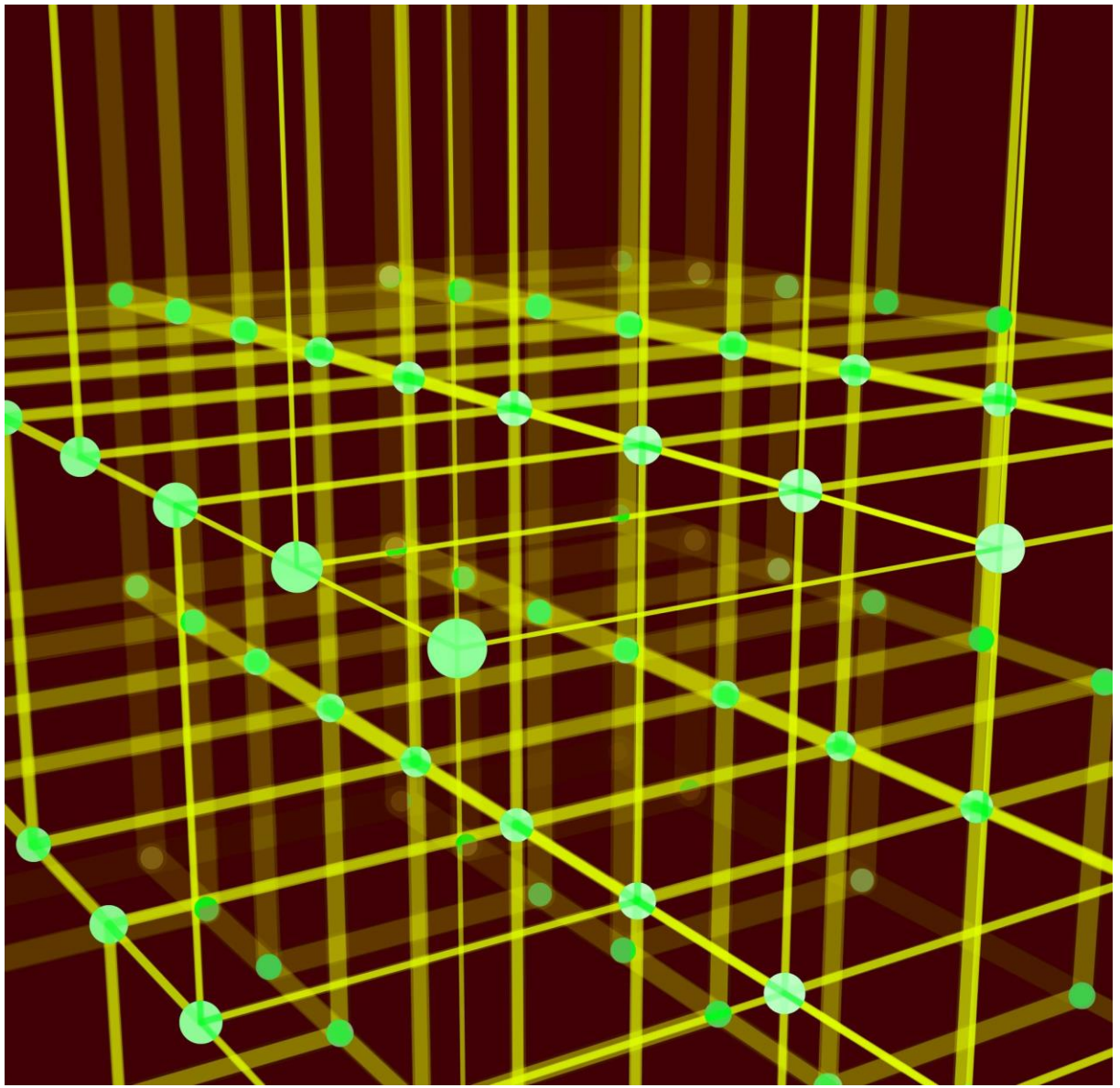


ALGUNOS PARADIGMAS DE PROGRAMACIÓN

- Programación imperativa (procedimental o por procedimientos)
- Programación estructurada
- Programación modular
- Programación orientada a objetos
- Programación en paralelo
- Programación multiparadigma

PARADIGMAS DE PROGRAMACIÓN

- La programación procedimental, estructurada y modular son paradigmas de programación
- La programación procedimental implica un enfoque lineal y de arriba hacia abajo para la codificación
- La programación estructurada enfatiza el uso de subrutinas
- La programación modular implica dividir el código en partes fácilmente manejables



PROGRAMACIÓN IMPERATIVA/PROCEDIMENTAL (PROCEDURAL PROGRAMMING)

- Un programa es en realidad un conjunto de declaraciones que se ejecutan **secuencialmente**.
- La única opción que tiene un programa, en términos de capacidad de administración, es dividir el programa en pequeños módulos.
- "C", por ejemplo, es un lenguaje de procedimiento.
- Python admite programación de procedimientos.

PROGRAMACIÓN IMPERATIVA/PROCEDIMENTAL

```
#Ejemplo: Ordenamiento de listas

import random

tam = int(input("Cantidad de elementos de  la lista? "))
numeros = list()
for i in range(tam):
    #Numeros del 1 al 100 los agrega
    numeros.append(random.randint(1,100))
print("Lista no ordenada:\n",numeros)
numeros.sort()
print("Lista ordenada:\n",numeros)
```

- Todo el código está seguido, en el flujo principal del programa.
- Se ejecuta en su totalidad, de principio a fin.
- Aprovecha condicionales (if, elif, else) y ciclos.

PROGRAMACIÓN ESTRUCTURADA

```
#Ejemplo: Ordenamiento de listas

import random

def Burbuja(numeros):
    numeros.sort()

tam = int(input("Cantidad de elementos de  la lista? "))
Numeros = list()
Numeros2 = list()
for i in range(tam):
    Numeros.append(random.randint(1,100))
    Numeros2.append(random.randint(1,100))
print("Lista no ordenada:\n",Numeros)
Burbuja(Numeros)
Burbuja(Numeros2)
print("Lista ordenada:\n",Numeros)
```

- Tiene como opción el encapsular el código en bloques llamados procedimientos o funciones, las cuales se identifican como las subrutinas del programa.
- Las funciones solo se ejecutan si son llamadas
- Utiliza la sintáxis:

```
def NombreFuncion(parámetros):

        #Código
```

PROGRAMACIÓN ESTRUCTURADA

```python
#Ejemplo: Ordenamiento de listas

import random

#Flujo principal
print("Hola mundo 1")

def Burbuja(numeros):
    numeros.sort()

#Flujo principal
print("Hola mundo 2")

#Función main
if __name__ == "__main__":
    tam = int(input("Cantidad de elementos? "))
    Numeros = list()
    for i in range(tam):
        Numeros.append(random.randint(1,100))
    print("Lista no ordenada:\n",Numeros)
    Burbuja(Numeros)
    print("Lista ordenada:\n",Numeros)

#Código principal (fin)
print("Fin")
```

Para una major organización, en Python, Podemos aprovechar el concepto de main.

Se compone de 3 secciones:

- Flujo principal.
  - Las líneas de Código están alineadas a la orilla.
  - Se ejecuta siempre
  - Se ejecuta de arriba hacia abajo
  - Si encuentra el "main" y lo puede ejecutar, lo hace y continua.

PROGRAMACIÓN ESTRUCTURADA

```python
#Ejemplo: Ordenamiento de listas

import random

#Flujo principal
print("Hola mundo 1")

def Burbuja(numeros):
    numeros.sort()

#Flujo principal
print("Hola mundo 2")

#Función main
if __name__ == "__main__":
    tam = int(input("Cantidad de elementos? "))
    Numeros = list()
    for i in range(tam):
        Numeros.append(random.randint(1,100))
    print("Lista no ordenada:\n",Numeros)
    Burbuja(Numeros)
    print("Lista ordenada:\n",Numeros)

#Código principal (fin)
print("Fin")
```

- Función main.
  - Se ejecuta si y solo si el archivo donde está es el que se está ejecutando.
  - Si el archive donde esta es importado, no se ejecuta.
  - Sintaxis:

```
if __name__ =
"__main__":

        #Código
```

- Otras funciones. Solo se ejecutan si las llaman

PROGRAMACIÓN MODULAR

```
def Burbuja(numeros):
    numeros.sort()

print("03_Flujo principal del módulo")

#Función principal de este programa
if __name__ == "__main__":
    print("05_Impresión en la función main")

print("04_Flujo principal final del módulo")
```

- Todo archivo, que contiene código Python, es un módulo.
- Cada uno puede contener las 3 partes ya mencionadas: flujo principal, función main y otras funciones.

PROGRAMACIÓN MODULAR

ModBurbuja.py - C:\Users\marle\OneDrive\Clase Programacion\Enero 2020\IP\Programas 3erP\Paradigmas\ModBurbuja.py (3.7.0)

File Edit Format Run Options Window Help

```
def Burbuja(numeros):
    numeros.sort()

#Ejemplo: Ordenamiento
#Llamo otros módulos y si son 2 o más, los separo con comas
import random, ModBurbuja


#Código principal (inicio)
print("01_Hola mundo")

#Función principal de este programa
if __name__ == "__main__":
    tam = int(input("Cantidad de elementos de  la lista? "))
    Numeros = list()
    for i in range(tam):
        Numeros.append(random.randint(1,100))
    print("Lista no ordenada:\n",Numeros)
    ModBurbuja.Burbuja(Numeros)
    print("Lista ordenada:\n",Numeros)

#Código principal (fin)
print("02_Fin")
```

- Para usar ese módulo en otro programa, usamos el código:

`import NombreArchivo`

- NOTA: no se agrega el ".py" al final

PROGRAMACIÓN ORIENTADA A OBJETOS (OBJECT ORIENTED PROGRAMMING - OOP)

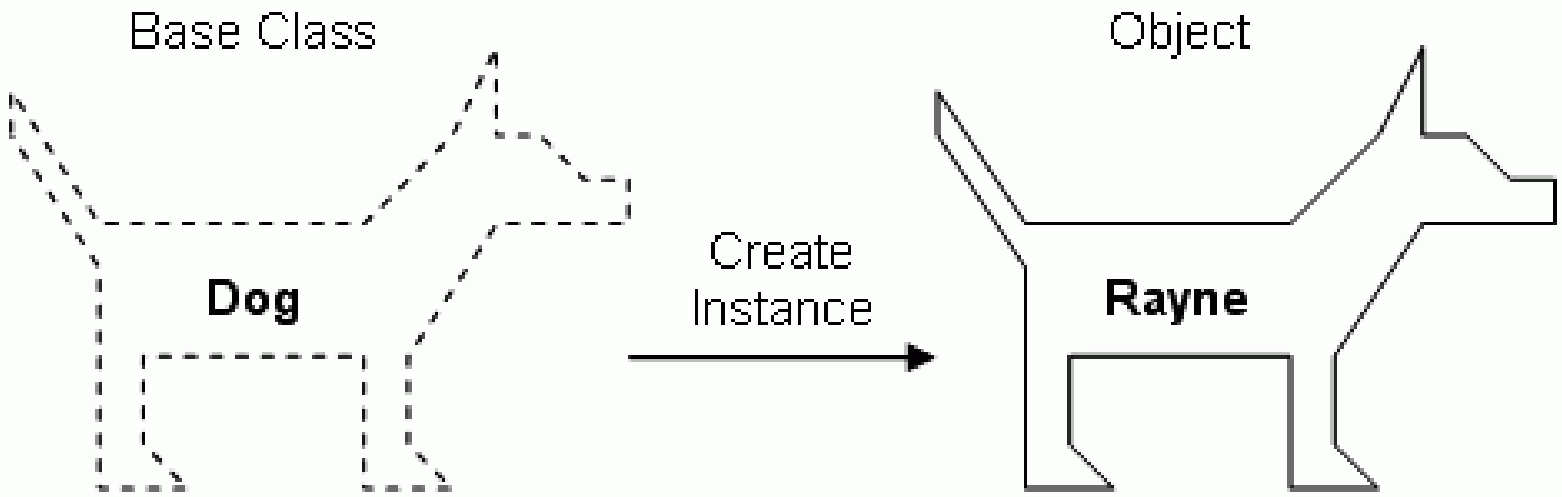
SE CENTRA PRINCIPALMENTE EN LA **INSTANCIA** DE UNA CLASE.

LA INSTANCIA DE UNA CLASE SE LLAMA **OBJETO**.

UNA CLASE ES UNA ENTIDAD REAL O VIRTUAL QUE TIENE UNA IMPORTANCIA PARA EL PROBLEMA EN CUESTIÓN Y TIENE LÍMITE.

POR EJEMPLO, EN UN PROGRAMA QUE SE OCUPA DE LA GESTIÓN DE ESTUDIANTES, "ESTUDIANTE" PUEDE SER UNA CLASE. SE REALIZAN SUS INSTANCIAS Y LA TAREA EN CUESTIÓN SE PUEDE LOGRAR MEDIANTE LA COMUNICACIÓN A TRAVÉS DE MÉTODOS.

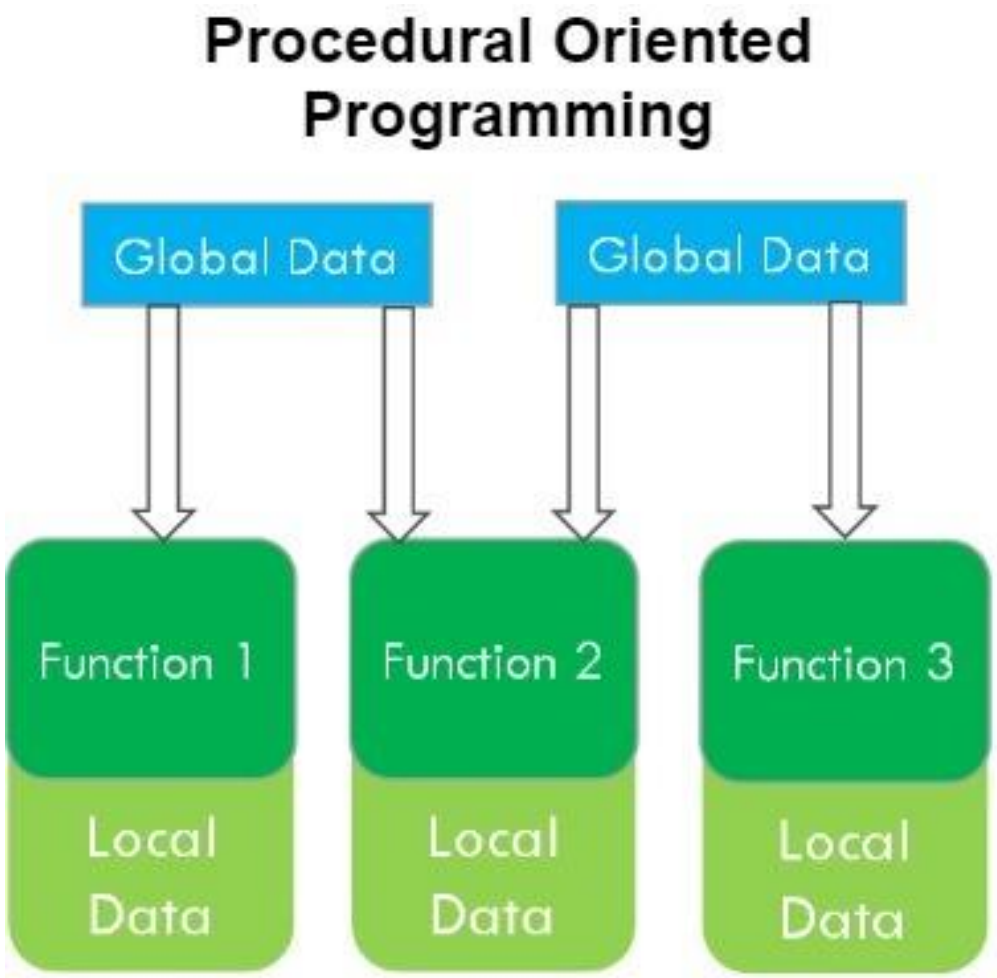
PYTHON ESTÁ ORIENTADO A OBJETOS.

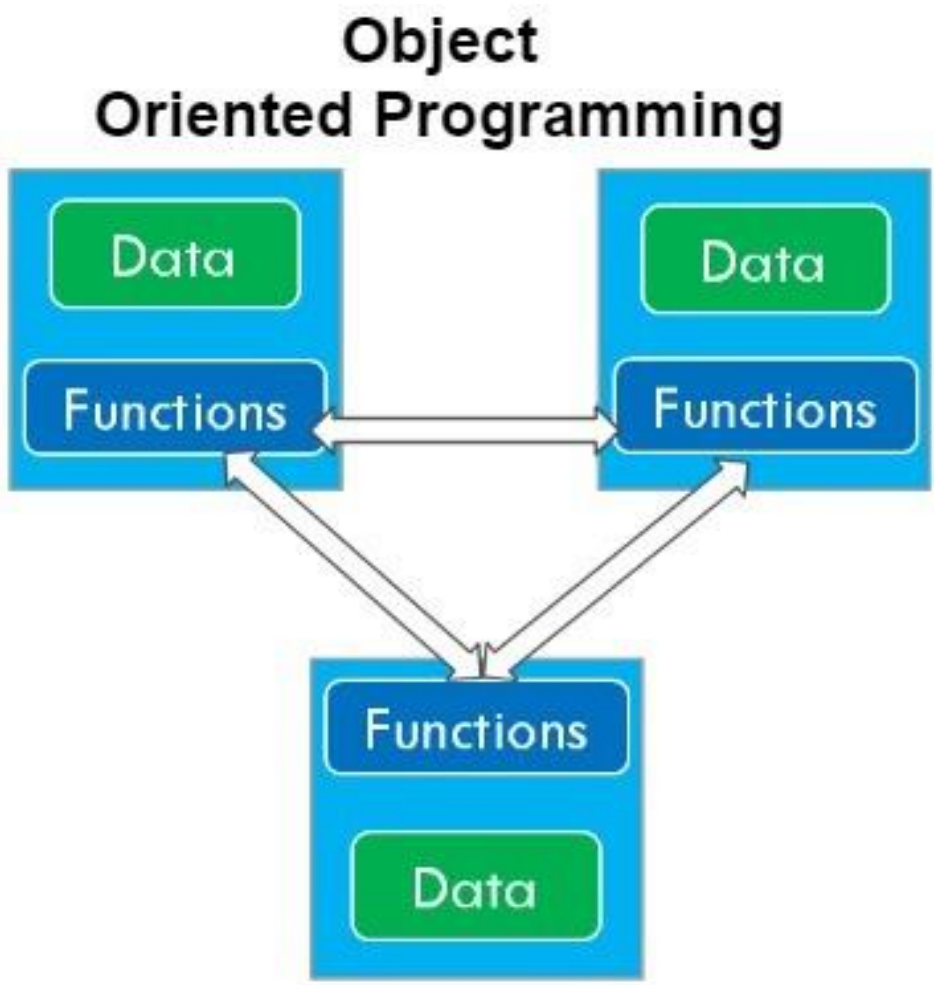


Properties	Methods
Color	Sit
Eye Color	Lay Down
Height	Shake
Length	Come
Weight	

Property values	Methods
Color: Gray, White, and Black	Sit
Eye Color: Blue and Brown	Lay Down
Height: 18 Inches	Shake
Length: 36 Inches	Come
Weight: 30 Pounds	

PROGRAMACIÓN ORIENTADA A OBJETOS (OBJECT ORIENTED PROGRAMMING - OOP)





PROGRAMACIÓN ORIENTADA A OBJETOS (OBJECT ORIENTED PROGRAMMING - OOP)

EJEMPLO DE CLASES EN PYTHON

```
class Empleado:
    empCount = 0

    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
        Empleado.empCount += 1

    def displayCount(self):
      print ("Total Empleado ", Empleado.empCount)

    def displayEmpleado(self):
        print ("Nombre : ", self.name,  ", Salario: ", self.salary)


emp1 = Empleado ("John", 100)
emp1.displayEmpleado()
emp1.displayCount()
```

PROGRAMACIÓN ORIENTADA A OBJETOS

```
#Ejemplo: Ordenamiento
import random

class OrdBurbuja:
    def __init__(self):
        print("Creamos un objeto")

    def Burbuja(self,num):
        for i in range(1,len(num)):
            for j in range(0,len(num)-i):
                if num[j] > num[j+1]:
                    (num[j+1],num[j]) = (num[j],num[j+1])

tam = int(input("Cantidad de elementos de  la lista? "))
Numeros = list()
for i in range(tam):
    Numeros.append(random.randint(1,100))
print("Lista no ordenada:\n",Numeros)
objeto = OrdBurbuja()
objeto.Burbuja(Numeros)
print("Lista ordenada:\n",Numeros)
```

- Incluye el diseño de clases (modelos de entidades de la realidad).
- Las clases tienen la sintaxis:

```
class NombreClase:

        #Constructor

        #Metodos
```

- Una clase tiene nombre, constructor, atributos (variables) y métodos

PROGRAMACIÓN ORIENTADA A OBJETOS - CLASE

```
class Alumno:
    #Constructor
    def __init__(self,nombre,mat,sem):
        self.nombre = nombre
        self.mat = mat
        self.sem = sem

    def Imprimir(self):
        print(self.mat,'\t',self.nombre,'\t',self.sem)

    def PasarSem(self):
        self.sem = self.sem + 1

#Instanciamos un objeto
objAlumno = Alumno("Pancho",1234,2)
objAlumno.Imprimir()
objAlumno.PasarSem()
objAlumno.Imprimir()
```

- Constructor

```
def __init__(self,parametros):

        #Codigo
```

- Métodos

```
def NombreMet(self,parametros):

        #Codigo
```

PROGRAMACIÓN ORIENTADA A OBJETOS - CLASE

```
class Alumno:
    #Constructor
    def __init__(self,nombre,mat,sem):
        self.nombre = nombre
        self.mat = mat
        self.sem = sem

    def Imprimir(self):
        print(self.mat,'\t',self.nombre,'\t',self.sem)

    def PasarSem(self):
        self.sem = self.sem + 1

#Instanciamos un objeto
objAlumno = Alumno("Pancho",1234,2)
objAlumno.Imprimir()
objAlumno.PasarSem()
objAlumno.Imprimir()
```

- Atributos
  - Al no existir declaraciones, usan directamente
  - Requieren la palabra clave self

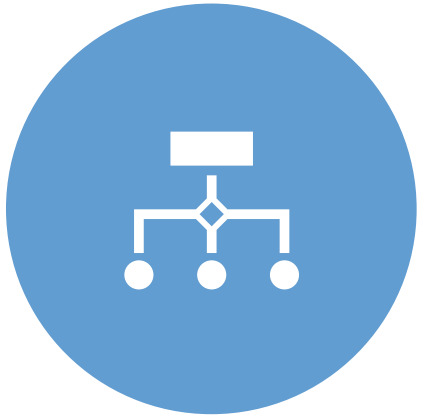
```
self.atributo = parametro

print(self.atributo)
```

- Instanciar objetos

```
nombreObj = Clase(argumentos)

nombreObj.Metodo()
```

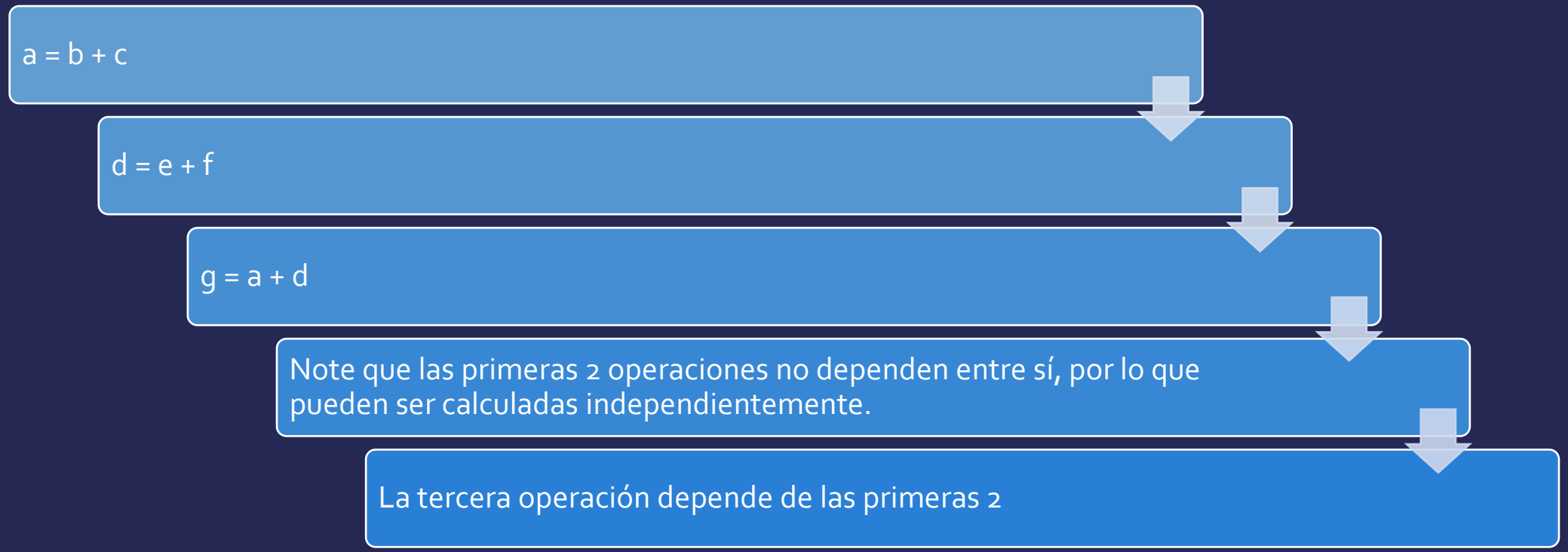


ES UN TIPO DE CÓMPUTO EN EL QUE DIVERSOS PROCESADORES EJECUTAN O PROCESAN UNA APLICACIÓN SIMULTÁNEAMENTE.

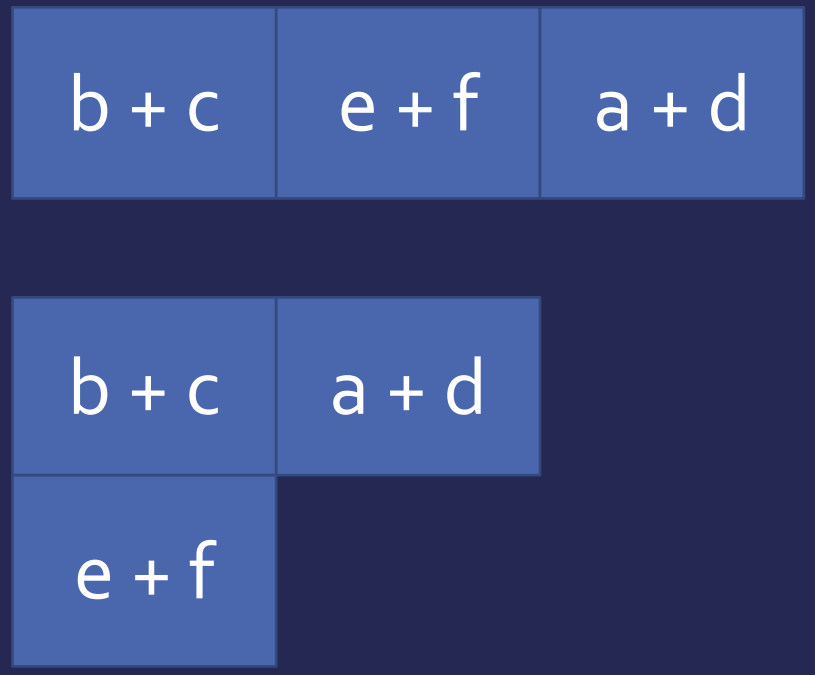
MEJORA LA EFICIENCIA DE COMPUTACIÓN DIVIDIENDO LA CARGA DE TRABAJO ENTRE LOS PROCESADORES.

CÓMPUTO EN PARALELO

EJEMPLO


a = b + c
d = e + f
g = a + d
Note que las primeras 2 operaciones no dependen entre sí, por lo que pueden ser calculadas independientemente.
La tercera operación depende de las primeras 2

EJEMPLO



3 Unidades de tiempo

2 Unidades de tiempo

Unidad de tiempo

```python
def print_cube(num):
        print("Cubo: {}".format(num * num * num))

def print_square(num):
        print("Cuadrado: {}".format(num * num))

if __name__ == "__main__":
##      Creamos 2 hilos (threads)
##       Cada hilo tiene como objetivo una funcion, square y cube
##       Como necesitan argumentos, se envía una tupla
        t1 = threading.Thread(target=print_square, args=(10,))
        t2 = threading.Thread(target=print_cube, args=(10,))

        # Se inician los hilos
        t1.start()
        t2.start()

        # Espera hasta que los hilos 1 y 2 se ejecuten completamente
        t1.join()
        t2.join()

        print("Listo!")
```

```
>>>
 RESTART: D:\Universidad
 General\Material GESG\Ej
ding.py
Cuadrado: 100Cubo: 1000

Listo!
>>>
 RESTART: D:\Universidad
 General\Material GESG\Ej
ding.py
Cuadrado: 100Cubo: 1000

Listo!
>>>
 RESTART: D:\Universidad
 General\Material GESG\Ej
ding.py
Cubo: 1000Cuadrado: 100

Listo!
>>>
 RESTART: D:\Universidad
 General\Material GESG\Ej
ding.py
Cuadrado: 100Cubo: 1000

Listo!
>>>
```

EL PRIMER HILO EN TERMINAR IMPRIME