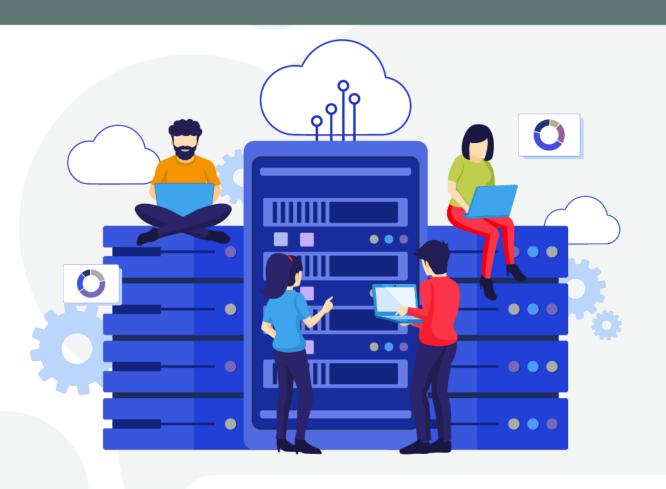


## CICLO 01

Fundamentos de Programación

Programación orientada a objetos
Definición clase vector







## Definición de la clase vector

## **Clase vector**

atributos privados: n, V[] en el constructor se inicializan los valores de  ${f n}$  y del arreglo  ${f V}$ 

En Python el constructor de un objeto de la clase vector se define así:

```
class vector:

    def __init__(self, n):

        self.n = n

        self.V = [0] * (n + 1)
```

Al ejecutar la instrucción

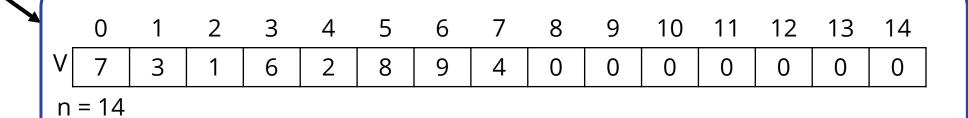
$$a = vector(14)$$

se obtiene un objeto de la clase vector llamado a, el cual se ve como en la figura

a

														13	
٧	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
n	n = 14														

a



Para este objeto a, perteneciente a la clase objeto:

k = a.posicionesUsadas() k valdrá 7

n = a.tamaño() n valdrá 14

b = a.esVacio() b será Falso



atributos privados: n, V[] en el constructor se inicializan los valores de **n** y del arreglo **V** 

En Python la definición de la clase vector con su constructor es así:

```
class vector:
    def init (self, n):
       self.n = n
       self.V = [0] * (n + 1)
def posicionesUsadas(self):
    return self.V[0]
def esVacio(self):
    return self.V[0] == 0
def esLleno(self):
    return self.V[0] == self.n
def tamaño(self):
    return self.n
def imprimeVector(self, mensaje="vector sin nombre: \t"):
    print("\n", mensaje, end=" ")
    for i in range(1, self.V[0] + 1):
        print(self.V[i], end=", ")
    print()
```

```
def agregarDato(self, d):
    if self.esLleno():
       return
    self.V[0] = self.V[0] + 1
    self.V[self.V[0]] = d
def intercambiar(self, a, b):
    aux = self.V[a]
    self.V[a] = self.V[b]
    self.V[b] = aux
def sumarDatos(self):
    s = 0
    for i in range(1, self.V[0] + 1):
        s = s + self.V[i]
    return s
```

```
def burbuja(self):
   for i in range(1, self.V[0]):
       for j in range(1, self.V[0] - i + 1):
            if self.V[j] < self.V[j + 1]:
                self.intercambiar(j, j + 1)
def mayor(self):
    mayor = 1
   for i in range(1, self.V[0] + 1):
        if self.V[i] > self.V[mayor]:
            mayor = i
    return mayor
def menor(self):
    menor = 1
   for i in range(1, self.V[0] + 1):
        if self.V[i] < self.V[menor]:
             menor = i
    return menor
def buscarDato(self, d):
    i = 1
    while i <= self.V[0] and self.V[i] != d:
        i = i + 1
    if i \le self.V[0]:
        return i
    return -1
```

```
def buscarDondeInsertar(self, d):
    i = 1
    while i \le self.V[0] and self.V[i] < d:
        i = i + 1
    return i
def insertar(self, d, i=0):
    if self.esLleno():
        print("\nVector lleno, no se puede insertar")
        return
    if i == 0:
        i = self.buscarDondeInsertar(d)
        for j in range(self.V[0], i - 1, -1):
              self.V[j+1] = self.V[j]
        self.V[i] = d
        self.V[0] = self.V[0] + 1
def borrarDatoEnPosicion(self, i):
    if i \le 0 or i > self.V[0]:
        print("\nParámetro i inválido")
        return
    for j in range(i, self.V[0]):
        self.V[i] = self.V[i + 1]
    self.V[0] = self.V[0] - 1
def borrarDato(self, d):
    i = self.buscarDato(d)
    if i != -1:
        self.borrarDatoEnPosicion(i)
```