



CICLO 1

[FORMACIÓN POR CICLOS]

Fundamentos de **PROGRAMACIÓN**



Ingeni@
Soluciones TIC



UNIVERSIDAD
DE ANTIOQUIA

Facultad de Ingeniería



Lectura

CLASE

vector



Con los siguientes ejercicios se pretende que usted use los conocimientos adquiridos en el curso para construir métodos con diferentes condiciones y características. De esta manera pondrá a prueba sus capacidades y destrezas en este campo.

Ya hemos definido los atributos de la clase y el constructor. Pasemos a definir las operaciones que podrán realizar los objetos de la clase vector. Veremos cuáles son las operaciones que se necesitan efectuar. Luego, para cada operación definiremos un método. Recuerde que en el contexto de POO dichas operaciones se definen como métodos.

1. Un método que retorne la capacidad del arreglo **V**. **tamano()**: retorna el valor del dato privado **n**.
2. Un método que retorne el número de elementos en el vector. **posicionesUsadas()**: retorna el número de posiciones utilizadas, es decir, retorna **V[0]**.
3. Un método para actualizar el número de elementos del vector. **asignaNumeroElementos(m)**: actualiza el contenido de **V[0]**.
4. Un método que retorne el dato correspondiente a una posición dada. **retornaDatos(i)**: retorna el dato que se halla en la posición **i** de **V**.
5. Un método que actualice el dato correspondiente a una posición **dada**. **asignaDatos(d, i)**: almacena el dato **d** en la posición **i** de **V**.
6. Un método para imprimir los datos del vector. **imprimeVector()**: recorre **V** mostrando el contenido de cada posición.
7. Un método para determinar si el vector está vacío o no. **esVacio()**: retorna verdadero si **V[0]** es cero, falso de lo contrario.
8. Un método para determinar si el vector está lleno o no. **esLleno()**: retorna verdadero si **V[0]** es igual a **n**, falso de lo contrario.
9. Un método para insertar un dato al final del vector **agregaDatos(d)**: inserta el dato **d** a continuación del último dato en **V**.



10. Un método que retorne la posición en la cual se halla un dato en el vector. **buscarDato(d):** retorna la posición en la cual se halla el dato **d** en **V**. Si no encuentra el dato, retorna **-1**.

11. Un método para buscar dónde insertar un dato en un vector ordenado ascendente. **buscaDondeInsertar(d):** retorna la posición en la cual debe quedar el dato **d** para que los datos de **V** continúen ordenados en forma ascendente.

12. Un método para insertar un dato en una posición específica. **insertar(d, i):** inserta el dato **d** en la posición **i** de **V**. Recuerde que esto implica mover los datos de **V**.

13. Un método para borrar un dato que está en una posición dada del arreglo **V**. **borrarDatoEnPosicion(i):** elimina el dato que está en la posición **i** de **V**.

14. Un método para borrar un dato sin conocer en cuál posición se halla. **borrarDato(d):** busca el dato **d** en el vector y si lo encuentra lo elimina.

15. Un método que retorne la posición en la cual se halla el mayor dato del arreglo **V**. **mayor():** retorna la posición en la cual es halla el mayor dato del vector.

16. Un método que retorne la posición en la cual se halla el menor dato del arreglo **V**. **menor():** retorna la posición en la cual es halla el menor dato del vector.

17. Un método que me permita intercambiar dos datos cualesquiera del arreglo **V**. **intercambiar(i, j):** intercambia el dato que está en la posición **i** del vector con el que está en la posición **j**.

18. Un método para ordenar los datos del vector en forma **ascendente**. **selección():** ordena los datos de **V** en forma ascendente.

Veamos cómo queda el archivo Python en el que definimos la clase vector con sus métodos.



```

import random
class vector:

    def __init__(self, n):
        self.n = n
        self.V = [0] * (n + 1)

    def construyeVector(self, m, r):
        self.V[0] = m
        for i in range(1, m + 1):
            self.V[i] = random.randint(1, r)

    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 agregarDatos(self, d):
        if self.esLleno():
            return
        self.V[0] = self.V[0] + 1
        self.V[self.V[0]] = d

    def asignaDatos(self, d, i):
        self.V[i] = d

    def retornaDatos(self, i):
        return self.V[i]

    def intercambiar(self, a, b):
        aux = self.V[a]
        self.V[a] = self.V[b]
        self.V[b] = aux

    def seleccion(self):
        for i in range(1, self.V[0]):
            k = i
            for j in range(i + 1, self.V[0] + 1):
                if self.V[j] < self.V[k]:
                    k = j
            self.intercambiar(k, i)

    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 <= self.V[0]:
        return i
    return -1

def borrarDatoEnPosicion(self, i):
    if i <= 0 or i > self.V[0]:
        print("\nParámetro i inválido")
        return
    for j in range(i, self.V[0]):
        self.V[j] = self.V[j + 1]
    self.V[0] = self.V[0] - 1

def borrarDato(self, d):
    i = self.buscarDato(d)
    if i != -1:
        self.borrarDatoEnPosicion(i)

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 tamagno(self):
    return self.n

def asignaNumeroElementos(self, m):
    self.V[0] = m

def buscaDondeInsertar(self, d):
    i = 1
    while i <= 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.buscaDondeInsertar(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 sumaDatos(self):
    s = 0
    for i in range(1, self.V[0] + 1):
        s = s + self.V[i]
    return s

```

Al método que imprime vector lo hemos definido **imprimeVector(self, mensaje = "vector sin nombre: \t")**. El parámetro mensaje tiene asignada la leyenda “vector sin nombre:”, lo cual significa que el parámetro mensaje es opcional. Si este método se invoca sin ningún mensaje, la función **imprimeVector()** escribirá el mensaje “vector sin nombre”.

Algo similar ocurre con la función **insertar(self, d, i = 0)**. Esta función se puede invocar de dos maneras:

1. vec.insertar(dato, 8)
2. vec.insertar(25)

En la primera forma insertará el dato en la posición 8 del vector.

En la segunda forma busca la posición donde debe insertar el dato y lo inserta en esa posición. Esta segunda forma se utiliza perfectamente cuando se desee insertar un dato en un vector cuyos datos están ordenados ascendentemente y se desea que se conserve esa propiedad.

Esta es una forma de **polimorfismo**: un método que actúa diferente dependiendo del número de parámetros que tenga.

Es importante notar el uso de la palabra **self**. La palabra **self** se utiliza para referirse al objeto que invoca el método. Es indispensable siempre que queramos referirnos a alguno de los atributos de la clase y cuando se desee invocar otro método de la misma clase.

En los métodos **esVacio()** y **esLleno()** se utilizan las siguientes instrucciones:

```
return self.V[0] == 0  
y  
return self.V[0] == self.n
```

las cuales son una forma abreviada de escribir:

```
if self.V[0] == 0:  
    return True  
else:  
    return False  
  
y  
if self.V[0] == self.n:  
    return True  
else:  
    return False
```

El método **mayor()** retorna la posición en la cual se encuentra el mayor dato del vector. En caso de que el mayor dato esté varias veces en el vector, nuestro método retorna la posición de la primera ocurrencia de ese dato mayor. Por ejemplo, si el mayor dato es 88 y se encuentra en las posiciones 3, 16, 28, 39 y 77, nuestro algoritmo retorna 3 ya que es en esa posición donde aparece el 88 la primera vez. Esto funciona así porque la instrucción en la cual se comparan los datos se utiliza el operador mayor que (**>**). Si en vez de mayor que utilizamos el operador mayor o igual que (**>=**), entonces el método retornará la última posición donde aparece el mayor dato, es decir, la posición 77.

Recomendamos al lector entender y racionalizar la clase **vector** y todos los métodos definidos aquí.

