Sintaxis y Semántica del Lenguaje

TAD Simple implementación

```
def crearEstudiante():
    est=["", "", 0, 0]
    return est
```

```
def cargarEstudiante(est, n, a, l, p):
    est[0]=n
    est[1]=a
    est[2]=l
    est[3]=p
```

Si se fijan bien, al cargar el estudiante, se está asociando cada posición a un atributo particular. Esto debe tenerse en cuenta a la hora de crear los ver y los modificar.

TAD Simple implementación

```
def verNombre(est):
    return est[0]

def modNombre(est, nNom):
    est[0] = nNom

def verApellido(est):
    return est[1]

def modApellido(est, nApe):
    est[1] = nApe

def verLegajo(est):
    return est[2]

def modLegajo(est, nLeg):
    est[2] = nLeg

def verPromedio(est):
    return est[3]

def modPromedio(est, nProm):
    est[3] = nProm
```

Recordando que:

0 → Nombre
1 → Apellido
2 → Legajo
3 → Promedio

TAD Simple implementación

```
def asignarEstudiante(est1, est2):
    est2[0] = est1[0]
    est2[1] = est1[1]
    est2[2] = est1[2]
    est2[3] = est1[3]
```

Al entregar el TAD, este debe estar completo, ninguno de los métodos implementados en esta filmina puede faltar

TAD Compuesto

En muchas ocasiones, al desarrollar aplicaciones, tendremos que guardar los tad simples en algún contenedor que los agrupe. Los alumnos se anotan a cursos, los autos se venden en concesionarias, una clínica tiene pacientes...

Ante esta necesidad, utilizamos ***TAD Compuesto***. Estos TAD son del tipo ***conjunto de datos***.

TAD Compuesto - Métodos

```
#crearCurso():
        #crea y retorna un curso vacío

#agregarEstudiante(cur, est):
        #agrega un estudiante al curso

#eliminarEstudiante(cur, est):
        #elimina un estudiante del curso

#recuperarEstudiante(cur, i):
        #recupera el iesimo estudiante del curso

#tamanio(cur):
        #retorna la cantidad de estudiantes del curso
```

TAD Compuesto - Ejercicio

1. Crear un curso. Crear y agregar cuatro estudiantes.
    a. Una vez cargados, listar los datos de los cuatro estudiantes.

1. Asumir cargado un curso completo
    a. Recuperar el alumno con legajo 33928 e imprimir su nombre y apellido.
    b. Eliminar al alumno recuperado en el inciso anterior.

Conjuntos de datos - Pila y Cola

De entre los conjuntos de datos, tenemos dos estructuras con un comportamiento particular.

- Pila
- Cola

¿Que recuerdan de estas estructuras?

TAD Pila

Las pilas son tipos de datos genéricos, es decir, pueden almacenar cualquier tipo de dato. Estas son una estructura de tipo “LIFO” (*Last Input, First Output*), el último elemento colocado es el primero en sacarse.

Sus operaciones básicas implican apilar (push) y desapilar (pop). Un ejemplo cotidiano de este tipo de estructura es la pila de platos.

TAD Pila - Especificación

```
def crearPila():
    #Crea una pila vacia

def esVacia(pila):
    #Retorna Verdadero si la pila no
tiene elementos

def apilar(pila,elem):
    #Agrega un elemento al final de la
pila

def desapilar(pila):
    #Retorna y elimina el ultimo
elemento de la pila

def tamanio(pila):
    #Retorna la cantidad de elementos
de la pila

def copiarPila(pila1,pila2):
    #Copia los datos de una pila a otra
```

TAD Cola

Al igual que las pilas, son tipos de datos genéricos. Sin embargo, estas estructuras son del tipo “FIFO” (*First Input, First Output*), el primer elemento colocado es el primero en sacarse. Sus operaciones básicas son encolar (push o enqueue) y desencolar (pop o dequeue).

Un ejemplo cotidiano podrían ser las personas que esperan para ingresar al banco.

TAD Cola - Especificación

```
def crearCola():
    #Crea una cola vacia

def esVacia(cola):
    #Retorna Verdadero si la cola no tiene elementos

def encolar(cola, elem):
    #Agrega un elemento al final de la cola

def desencolar(cola):
    #Retorna y elimina el primer elemento de la cola

def tamanio(cola):
    #Retorna la cantidad de elementos de la cola

def copiarCola(cola1,cola2):
    #Copia los datos de una cola a otra
```