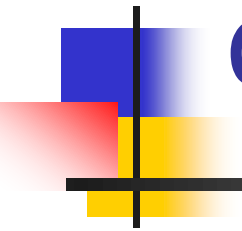
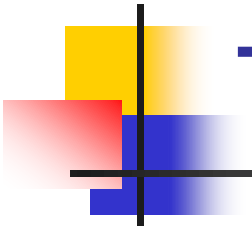


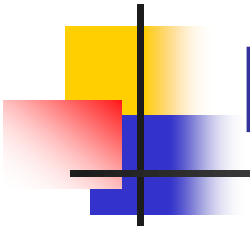
Introducción a los tipos de datos abstractos (TAD)

SINTAXIS Y SEMÁNTICA DEL LENGUAJE



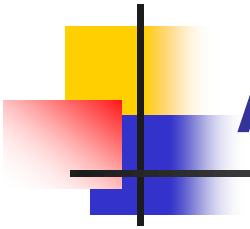
Temario

- Definición del concepto de Abstracción
- Tipos de datos abstractos (TADs)
  - Especificación
  - Diseño de la estructura interna
  - Implementación de operaciones
- TADs simples en Python



Definición de abstracción

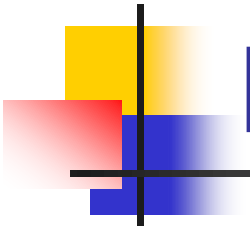
Abstracción: es la capacidad de concentrarse sobre las cualidades o aspectos esenciales de alguna entidad u objeto del mundo real e ignorar las propiedades accidentales.



Abstracción de datos

En una abstracción de datos se piensa en *qué acciones* se pueden aplicar sobre una colección de datos independientemente de *cómo* se llevan a cabo las mismas.

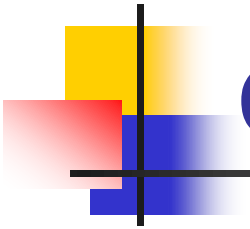
Es la capacidad de encapsular y aislar los datos del diseño, de la implementación y de la ejecución.



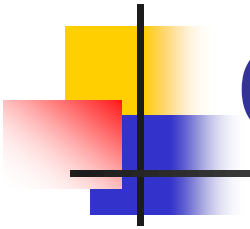
Definición

- Un *TAD* (tipo abstracto de datos) es:
- una *colección de datos*
- acompañada de un *conjunto de operaciones* para manipularlos, de forma tal que quede *oculta tanto la representación interna del nuevo tipo como la implementación de las operaciones*, para todas las unidades de programa que lo utilice.

Evolución hacia el concepto de TAD

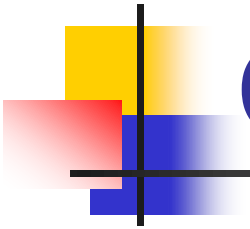


- Lenguaje de máquina→ datos son cadenas de bits + op. de desplazamiento (pto.flot)
- Fortran, Cobol y Algol 60 definen *tipos de datos standard*: entero – real- bool – carácter
- Algol 68 define constructores de tipos o *tipos definidos por el usuario*: alumno, curso



Comparación entre tipos

- Semejanzas entre tipos estándar y def. por usuario:
  - Ambos tienen estructura interna
    - Entero: es una cadena de bits
    - Alumno: es un registro
    - Curso de alumnos: vector de registros
  - Ambos tienen operaciones asociadas
    - Entero: +, -, *, /
    - Alumno: asignar valor a cada campo- imprimir campos



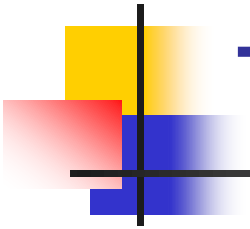
Comparación entre tipos

- Diferencias:
  - Los tipos estándar ocultan su representación interna al programador y no permite que la modifiquemos.
  - Los tipos definidos por el usuario exigen elegir la representación interna al programador y permiten manipularla para asignar valor y modificarlo. Si lo desea, puede modificar dicha estructura interna y en consecuencia modificar las operaciones asociadas.

Combinando características nacen los TAD's

- Si definimos *nuevos tipos de datos* pero *ocultando tanto su representación interna* como *la implementación de sus operaciones* mediante un mecanismo de *encapsulamiento*, obtenemos un

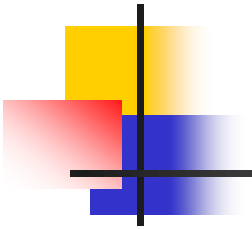
TIPO ABSTRACTO DE DATOS.



TAD's

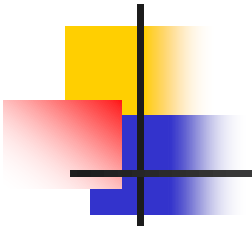
- De esta forma el usuario solo *conoce*
  - *para qué* sirve el nuevo TAD,
  - *qué* operaciones puede aplicarle y
  - *qué* valores puede almacenar

  - pero *no conoce cómo* lo hace: no tiene acceso al almacenamiento ni a la implementación de las operaciones



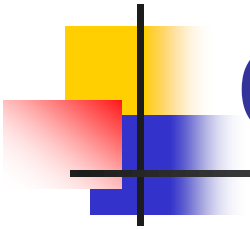
TAD's

- Un TAD se compone entonces de:
  - Una interfaz de usuario o *vista pública:* es lo que conoce el usuario para utilizarlo para programar
    - Especifica para qué sirve el TAD
    - Especifica qué hace cada operación



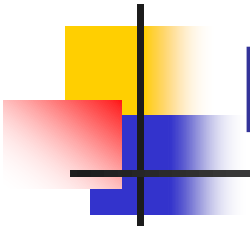
TAD's

- Una implementación o *vista privada,* que solo la conoce el diseñador del TAD
  - Detalla la estructura de datos que soporta el almacenamiento de datos (estructura interna)
  - Codifica la implementación de cada operación



Características de los TAD's

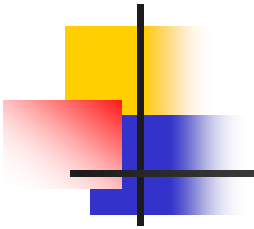
- **Ocultamiento de la información**
  - Es el mecanismo mediante el cual el lenguaje de programación hace inaccesibles tanto la estructura interna como la implementación de las operaciones del TAD. Para acceder a los datos almacenados solo se aplican las operaciones definidas a tal fin.
- **Encapsulamiento**
  - Es el mecanismo mediante el cual tanto la representación interna como la implementación de las operaciones se codifican y guardan juntas, en el lenguaje de programación a utilizar.
- **Generalización**
  - El programador puede definir un TAD cuyos elementos a almacenar sean de tipos genéricos (si el lenguaje lo permite)



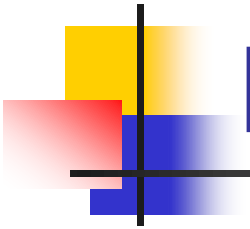
Diseño de TAD's

- Pasos en la creación de un nuevo TAD
  - 1°- Especificación (es la interfaz *pública*)
  - 2°- Diseño de la Representación Interna (*privada*)
  - 3°- Implementación de las Operaciones(*privada*)

Especificación – Interfaz de usuario



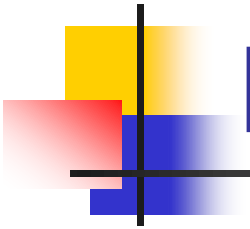
- Se elige el nombre del TAD
- Se explica cuál es la utilidad del TAD en la vida real
- Se definen las operaciones que se pueden aplicar sobre las variables de dicho tipo:
  - Se indican los nombres, tipos y parámetros de dichas operaciones.
  - Se detalla mediante un comentario para qué sirve cada operación del mismo (qué efecto tiene, qué datos recibe y qué datos retorna)



Especificación

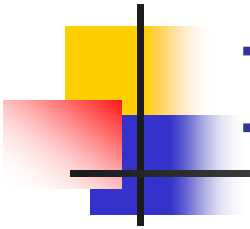
Operaciones básicas de un TAD simple:

- Creación (reserva espacio para almacenar los datos. Puede o no inicializarlos)
- Asignación de valor o modificación de valor
- Consulta de valor
- Destrucción (libera memoria, no siempre se define)



Representación Interna

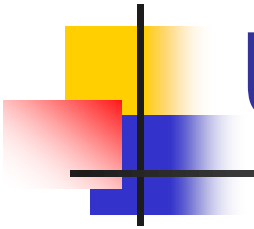
- Elegir la **estructura interna** de la entidad en base a las estructuras o tipos de datos disponibles en el lenguaje de programación elegido, que más se adecúe a los datos que se van a almacenar.



Implementación

- Por último se **desarrollan** los algoritmos correspondientes a **las operaciones**, **en el código del lenguaje elegido** y de acuerdo a la representación interna definida en el paso previo.

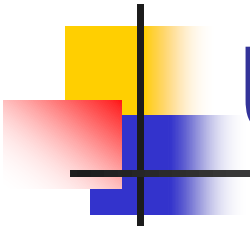
- Dependiendo del lenguaje elegido el TAD se implementa con diferentes herramientas.



Uso del TAD

- Una vez que el *diseñador* termina de implementar el TAD, lo guarda (implanta) en el ambiente de desarrollo y le entrega al usuario solo la interfaz pública.

- El *usuario* podrá utilizar el TAD en sus programas de aplicación definiendo variables de dicho tipo y manipulándolas a través de las operaciones definidas en dicha interfaz.



Uso de un TAD SIMPLE

- Supongamos tener definido en Python el TAD horario que almacena la hora, los minutos y los segundos de un horario.
- Sus operaciones básicas son:
  - Crear un horario (vacío)
  - Cargar datos de un horario
  - Modificar los datos de un horario
  - Consultar los datos de un horario

Interfaz de usuario

crearHorario():
#crea y retorna un horario sin datos

cargarHorario (horar, ho, mi, se):
#Carga los datos de un horario

verHora(horar):
#Retorna la hora

verMin(horar):
#Retorna los minutos

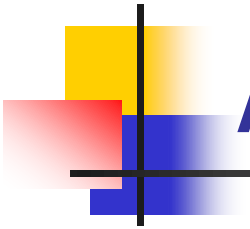
verSeg(horar):
#Retorna los segundos

modiHora(horar, otraH):
#Modifica la hora

modiMin(horar, otroM):
#Modifica los minutos

modiSeg(horar, otroS):
#Modifica los segundos

asignarHorario(h1,h2):
#Copia los datos del horario h2 en el horario h1



APLICACIÓN

- Desarrollar una aplicación que cree y cargue 1 horario y luego lo imprima expresado en segundos.

- Observación: El TAD horario está definido e implementado en el módulo TadHorario.py

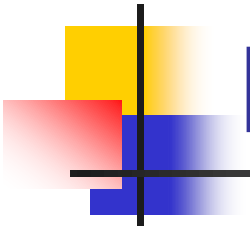
APLICACIÓN

`from TadHorario import *` → suponemos creado y guardado en el ambiente Python el módulo correspondiente al TAD horario. Así se pueden importar todas sus operaciones

```
#Crea y carga los datos de un horario
hour=crearHorario()
ho=int(input("Ingrese una hora HH"))
minu=int(input("Ingrese los minutos MM"))
sec=int(input("Ingrese los segundos SS"))
cargarHorario(hour,ho,minu,sec)

#Pasa el horario a segundos y lo imprime
totalseg= verHora(hour)*3600 + verMin(hour)*60 +
verSeg(hour)
print(totalseg)
```

Usamos el TAD horario sin saber su implementación.

Diseño de TAD SIMPLE en Python



Ejemplo: TAD Alumno:

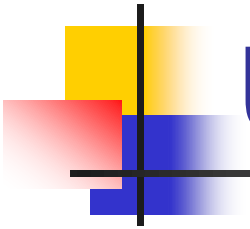
- sirve para almacenar el nombre (string), legajo(entero) y promedio(real) de un alumno
- Operaciones básicas:
  - Crear un alumno
  - Cargar datos del alumno
  - Consultar datos del alumno
  - Modificar datos del alumno

Especificación TAD Alumno

```
crearAlu():
   #crea y retorna un alumno sin datos

cargarAlu (alu, nom, leg, prom):
    #Carga los datos de un alumno

 verNom(alu):
    #Retorna el nombre del alumno

verLeg(alu):
    #Retorna el legajo del alumno

verProm(alu):
    #Retorna el promedio del alumno
```

Especificación TAD Alumno

```
modiNom(alu, otroN):
    #Modifica el nombre del alumno

modiLeg(alu, otroL):
    #Modifica el legajo del alumno

modiProm(alu, otroP):
 #Modifica el promedio del alumno

asignarAlu(alu1, alu2):
    #Copia los datos del alumno 2 en el alumno 1
```



Uso del TAD - APLICACIÓN

- Ejercicio: Sin terminar el diseño, es decir, solo con la interfaz, SIN SABER CUÁL SERÁ LA IMPLEMENTACIÓN NI LA EST. INTERNA, vamos a desarrollar una aplicación que cree 2 alumnos y luego imprima el que tiene menor legajo.

Uso del TAD - APLICACIÓN

`from TadAlu import *` → suponemos creado y guardado en el ambiente Python el módulo correspondiente al TAD alumno. Así se pueden importar todas las operaciones

```
#Crea y carga los datos del alumno1
a1=crearAlu()
n=input("Ingrese un nombre")
l=int(input("Ingrese un legajo"))
p=float(input("Ingrese un promedio"))
cargarAlu(a1,n,l,p)

#Imprime los datos del alumno1
print( verNom(a1))
print (verLega(a1))
print (verProm(a1))
```

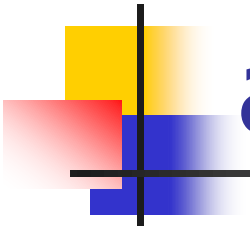
Uso del TAD - APLICACIÓN

```
#Crea y carga los datos del alumno2
a2=crearAlu()
n=input("Ingrese un nombre")
l=int(input("Ingrese un legajo"))
p=float(input("Ingrese un promedio"))
cargarAlu(a2,n,l,p)

#Imprime los datos del alumno1
print (verNom(a2))
print (verLega(a2))
print (verProm(a2))
```

Uso del TAD - APLICACIÓN

```
#Imprime el nombre del alumno de menor legajo

print ("El nombre del alumno de menor legajo
es:")
if (verLega(a1)<verLega(a2)):
    print (verNom(a1))
else:
    print (verNom(a2))
```

Representación interna TAD alumno



Si queremos terminar el diseño del TAD, hay que pasar a la implementación.

En primer lugar debemos pensar:

- Qué estructura de datos nos es más conveniente para almacenar los datos de un alumno?

Representación interna TAD alumno

En nuestro caso como no tenemos el tipo de dato registro en Python, vamos a elegir como estructura interna el tipo de datos lista.

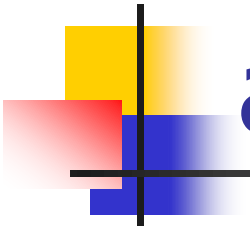
Así entonces se define:

alumno=[“”, “”, “”]

La lista tendrá 3 elementos:

1era posición para el nombre del alumno
2da posición para el legajo
3era posición para el promedio

Implementación del TAD alumno



- Se codifica en Python cada operación especificada, respetando la estructura de datos elegida

- En Python un TAD se implementa con ***un módulo.***

Implementación del TAD alumno

- El módulo **permite encapsular** la estructura interna y la implementación de las operaciones.

  Pero **NO se logra un ocultamiento** completo porque la importación de un módulo permite acceder a la estructura interna del Tad, no así a la implementación de las operaciones.

  Esta forma de encapsular solo permite ***simular.*** Para lograr un ocultamiento y encapsulamiento efectivo deberían usarse clases (POO).

Implementación del TAD alumno

```
def crearAlu():
    #crea y retorna un alumno sin datos
    al=["",0,0]      #inicializa cada posición en base a su futuro contenido
    return al

def cargarAlu (alu, nom, leg, prom):
    #Carga los datos del alumno
    alu[0]=nom
    alu[1]=leg
    alu[2]=prom
```

Implementación del TAD alumno

```
def verNom(alu):
    #Retorna el nombre del alumno
    return alu[0]

def verLeg(alu):
    #Retorna el legajo del alumno
    return alu[1]

def verProm(alu):
    #Retorna el promedio del alumno
    return alu[2]
```

Implementación del TAD alumno

```
def modiNom(alu, otroN):
    #Modifica el nombre del alumno
    alu[0]=otroN

def modiLeg(alu,otroL):
    #Modifica el legajo del alumno
    alu[1]=otroL
```

Implementación del TAD alumno

```
def modiProm(alu, otroP):
    #Modifica el promedio del alumno
    alu[2]=otroP

def asignarAlu(alu1, alu2):
    #Copia los datos del alumno2 al alumno 1
    alu1[0]=alu2[0]
    alu1[1]=alu2[1]
    alu1[3]=alu2[3]
```