

ESTRUCTURAS y TDA EN PYTHON

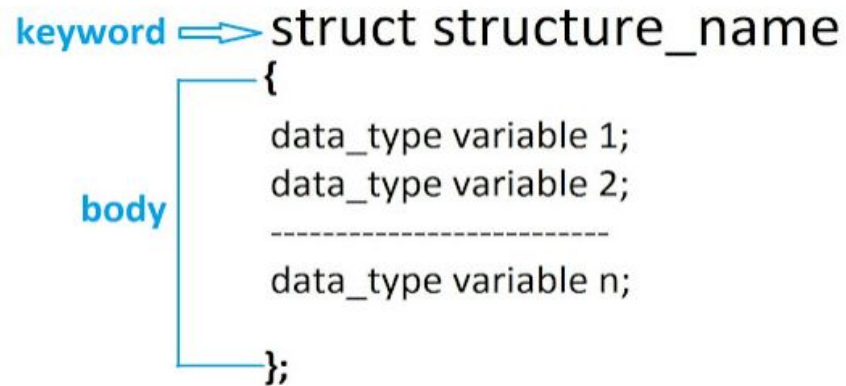
UNPAZ

Algoritmos y Programación

Estructuras

Definición

La estructura es una colección de variables, la cual puede poseer distintos tipos de datos (a diferencia de los arreglos que solamente pueden tener un solo tipo de dato). Es un tipo de dato definido por el usuario. Son también conocidas como Registros. Ayudan a organizar y manejar datos complicados en programas debido a que agrupan diferentes tipos de datos a los que se los trata como una sola unidad en lugar de ser vistas como unidades separadas.



The diagram illustrates the syntax of a C struct definition. It shows the keyword 'struct' followed by a space and a structure name 'structure_name'. A blue arrow points from the word 'keyword' to 'struct'. The opening curly brace '{' is followed by the body of the struct, which contains three lines of variable declarations: 'data_type variable 1;', 'data_type variable 2;', and 'data_type variable n;'. A blue arrow points from the word 'body' to the opening curly brace. The body is enclosed by a closing curly brace '}' followed by a semicolon ';'. A dashed line is used to separate the second and third variable declarations.

```
keyword ⇒ struct structure_name  
{  
    data_type variable 1;  
    data_type variable 2;  
    -----  
    data_type variable n;  
};
```

Estructuras

Clasificación de estructuras de datos por su naturaleza.

- Homogénea: Todos datos básicos del mismo tipo(vectores,tablas,matrices,etc.)
- Heterogénea: Datos de diferentes tipos (registros, etc.)

Clasificación de estructuras de datos según el manejo en memoria.

- Contiguas: Componentes ubicados en áreas adyacentes de memoria (Ej. vectores)
- Enlazadas: No tienen porque estar en zonas contiguas de memoria, uso de direcciones (Ej. listas enlazadas)

Clasificación por su forma de almacenamiento.

- Estáticas: Tamaño conocido a priori
- Dinámicas: El tamaño varía durante la ejecución del programa (listas, árboles, grafos)
- Memoria externa: Archivos y bases de datos

Estructuras

Emulando Estructuras en Python

Otros lenguajes (ejemplo TDA en C) , utilizamos una combinación de typedef a modo de definir un nuevo tipo, con structs. El struct era una construcción que nos permitía componer, es decir, definir un tipo de dato que estaba compuesto por miembros. Por ejemplo, la Persona tenía un nombre, apellido, documento, que a su vez podía ser otro tipo compuesto con tipo y número, etc. Para nosotros estos tipos compuestos eran estructuras de datos, es decir que solo contenían datos y cualquier operación que yo quería hacer sobre ellos, debería ser una función que los reciba por parámetro.

Ahora, **Python es un lenguaje orientado a objetos**. En estos lenguajes aparece una idea muy poderosa de unificar estas dos cosas: el estado/datos, con el comportamiento. Eso es un objeto, y (en general) se define a través de una clase.

Nosotros no vamos a ver objetos en esta materia, ya que tendrán luego otra materia exclusivamente para este paradigma. Pero sucede que Python no tiene una forma específica para definir "*structs*", ya que los objetos son más poderosos, no es necesario. Entonces, toda esa introducción es para decir que nosotros vamos a buscar una forma de "emular" la idea de hacer structs, pero en el medio van a aparecer cositas del paradigma de objetos, como las clases.

No se asusten y por ahora traten de verlo como si definiéramos un struct.

TDA-Tipo de dato abstracto

TDA Definición

Un tipo de dato abstracto es un tipo cuya representación como dato concreto ha sido abstraída y a cuyos datos puede accederse por medio de un conjunto de operaciones. Una abstracción es la simplificación de un objeto o de un proceso de la realidad en la que sólo se consideran los aspectos más relevantes.

Un tipo abstracto de datos (TAD) es un tipo definido por el usuario que:

- Tiene un conjunto de valores y un conjunto de operaciones.
- Cumple con los principios de abstracción, ocultación de la información y se puede manejar sin conocer la representación interna.

Operaciones Típicas: inicialización, modificación, búsqueda, mostrar, etc.

TDA = Estructura de datos + Operaciones

Características de abstracción:

Ocultamiento: Usuario sabe que puede hacer, pero no como lo hace

Encapsulamiento: Usuario no puede modificarlo, agrupa implementación del TDA

- Abstracción -> Flexibilidad

El uso de un TDA solo depende de su especificación, no de la implementación.
Dada una especificación hay muchas implementaciones válidas.
Cambios en la implementación son transparentes al usuario.

TDA – Implementación

Implementación de Estructuras en Python

Aquí está la implementación clásica en el lenguaje C. Básicamente la palabra reservada struct y entre llaves los atributos que la describen.

```
struct Producto {  
    char[20] nombre;  
    int precio;  
}
```

Aquí está la implementación emulada utilizando las herramientas que Python provee.

Opción 1

```
class Producto:  
    def __init__(self, nombre, precio):  
        self.nombre = nombre  
        self.precio = precio
```

En realidad estamos definiendo una Clase, y fíjense que, además, estamos creando una función dentro de la clase, que se tiene que llamar `__init__`, y cuyo primer parámetro tiene que ser `self`, y que luego, puede tener cuantos parámetros nosotros queramos. Técnicamente lo que sucede es que esta función le está agregando miembros al objeto nuevo que referenciamos como `self`. Porqué Python es un lenguaje dinámico, y me permite modificar un objeto agregándole miembros.

TDA – Implementación

Implementación de Estructuras en Python

Aquí esta otra forma para la implementación emulada utilizando las herramientas. La forma de emular structs explicada recién, no escondía demasiado los detalles de la programación con objetos. Ya que teníamos que definir la función constructora dentro de la clase.

Otra variante más parecido a lo que hacíamos en C

- Definir el tipo
- Definir una función que construye una nueva instancia.

Opción 2

```
class Producto:
    pass

def crearProducto(nombre, precio):
    p = Producto()
    p.nombre = nombre
    p.precio = precio
    return p

print crearProducto("Aceite", 9.5)
```

TDA – Ejemplo Empleado

Información a almacenar

Ejemplo: TDA Empleado •

Información a almacenar:

- Nombre
- Apellido
- Edad
- DNI

Operaciones Necesarias

- Crear un empleado nuevo
- Modificar datos de un empleado existente
- Borrar un empleado
- Obtener valores de un empleado (Ej. Nombre)
- ...

TDA - Ejemplo Empleado

Ejemplo de Estructura EMPLEADO en Python

Aquí se describe la interpretación gráfica de dos registros empleados.

| EMPLEADO 1 |
|------------------|
| Nombre: Jose |
| Apellido: Castro |
| DNI: 243434 |
| Edad: 15 |

| EMPLEADO 2 |
|--------------------|
| Nombre: Lucia |
| Apellido: Martinez |
| DNI: 111434 |
| Edad: 22 |

Otra forma que podríamos imaginar un registro es mediante esta interpretación gráfica.

| Empleado | Nombre | Apellido | DNI | Edad |
|----------|--------|----------|--------|------|
| 1 | Jose | Castro | 243434 | 15 |
| 2 | Lucia | Martinez | 111434 | 22 |

TDA - Ejemplo Empleado

Ejemplo de Estructura EMPLEADO en Python

```
# TDA EMPLEADO
class tdaEMPLEADO:
    def __init__(self, pNombre, pApellido, pDNI, pEdad):
        self.nombre = pNombre
        self.apellido = pApellido
        self.dni = pDNI
        if pEdad>0:
            self.edad = pEdad
        else:
            raise Exception ("la edad no puede ser negativa")
    ##Retorno concatenado
    def __str__(self):
        return self.nombre + ' ' + self.apellido
    ##Retorno En Funciones
    def ObtenerNombre(self):
        return self.nombre
    def ObtenerApellido(self):
        return self.apellido

##Programa Principal
miEmpleado1 = tdaEMPLEADO("Jose", "Castro", "243434", 15)
miEmpleado2 = tdaEMPLEADO("Lucia", "Martinez", "111434", 22)
print(miEmpleado1)
print(miEmpleado2)
##Obtener valores de la estructura
print("Empleado 1 Nombre:", tdaEMPLEADO.ObtenerNombre(miEmpleado1))
print("Empleado 2 Nombre:", tdaEMPLEADO.ObtenerNombre(miEmpleado2))
```



| EMPLEADO |
|----------|
| Nombre |
| Apellido |
| DNI |
| Edad |



| EMPLEADO 1 |
|------------------|
| Nombre: Jose |
| Apellido: Castro |
| DNI: 243434 |
| Edad: 15 |

| EMPLEADO 2 |
|--------------------|
| Nombre: Lucia |
| Apellido: Martinez |
| DNI: 111434 |
| Edad: 22 |

Arreglos(listas) de Estructuras

Arrays de Registros

Es posible agrupar un conjunto de elementos de tipo estructura en un array. Esto se conoce como array de estructuras.

Supongamos que queremos guardar información de todas las personas que tenemos en una empresa. Con una variable de tipo EMPLEADO, solo podemos guardar los datos de uno.

Necesitaremos un array de EMPLEADOS o mejor dicho una Lista ya que Python necesita de ellas para poder interpretar este concepto de programación..

| [0] | [1] | [2] | [3] | [4] |
|--------------------------------|-----------------------------------|-----|-----|-----|
| Jose Castro 243434 15 | Lucia Martinez 111434 22 | ... | ... | ... |

Interpretación Python

Aquí la interpretación mediante código Python de la representación gráfica descrita anteriormente

```
##Programa Principal
Empleados=[0]*5 ##Una de las tantas formas que tiene la lista de inicializar valores predefinidos
viCantElem=len(Empleados)
print(Empleados)
for i in range(0,viCantElem,1):
    vsNombre=input("Ingrese Nombre:")
    vsApellido =input("Ingrese Apellido:")
    vsDNI =input("Ingrese DNI:")
    viEdad = int(input("Ingrese Edad:"))
    Empleados[i]=tdaEMPLEADO(vsNombre, vsApellido,vsDNI, viEdad)
```

Gracias

