El futuro digital es de todos

MinTIC

'Mision TIC 2022'

Fund. de Programación

Grupos 79,80,81

UNIVERSIDAD DE ANTIOQUIA

1803

Facultad de Ingeniería

Resumen sesión anterior

ELEMENTOS DE POO

- **Clases**: Definen la estructura que van a tener los objetos que se creen a partir de ella, indicando que propiedades y métodos tendrán los objetos
- **Atributos:** Definen las características del objeto
- **Métodos:** Definen el comportamiento del objeto

CLASES EN PYTHON

```python
class Aves(object):
    def __init__(self, especie, pais):#Constructor
        self.especie=especie
        self.pais=pais

    def volar(self):
        return True

    def poner_huevos(self, cantidad):
        print(f"l@s {self.especie} ponen {cantidad} huevos a la vez")
```

Clases derivadas y Polimorfismo

JUAN FERNANDO GONZÁLEZ
GRUPOS 79,80,81
Semana 4 Sesión 1

CLASES DERIVADAS HERENCIA

La programación orientada a objetos permite definir nuevas clases con base en clases previamente definidas.

La clase original se denomina “clase base” o “superclase”, y la nueva clase se denomina “clase derivada” o “subclase”.

Los objetos pertenecientes a la clase derivada pueden usar los métodos de la clase base

Actividad 1

Crear una clase derivada de **lista** que permita calcular el promedio aritmético.

POLIMORFISMO

El polimorfismo es la capacidad de tomar más de una forma.
En POO, indica que una operación puede presentar diferentes comportamientos en diferentes instancias. El comportamiento depende de los tipos de datos utilizados en la operación. El polimorfismo es ampliamente utilizado en la aplicación de la herencia.

Actividad 2

Crear una superclase empleado que permita calcular el salario, posteriormente crear una subclase para las distintas áreas (ventas administrativos) que permita agregar características diferentes al pago.

Función super()

Esta función permite invocar y conservar un método de una clase padre (primaria) desde una clase hija (secundaria) sin tener que nombrarla explícitamente.

```
class Child(Parent):

    def __init__(self):
        Parent.__init__(self)
```

```
class Child(Parent):

    def __init__(self):
        super().__init__()
```

Actividad 3

Modificar el método append de la clase lista para que nos indique cual es el nuevo tamaño de lista cuando se agregue algo.