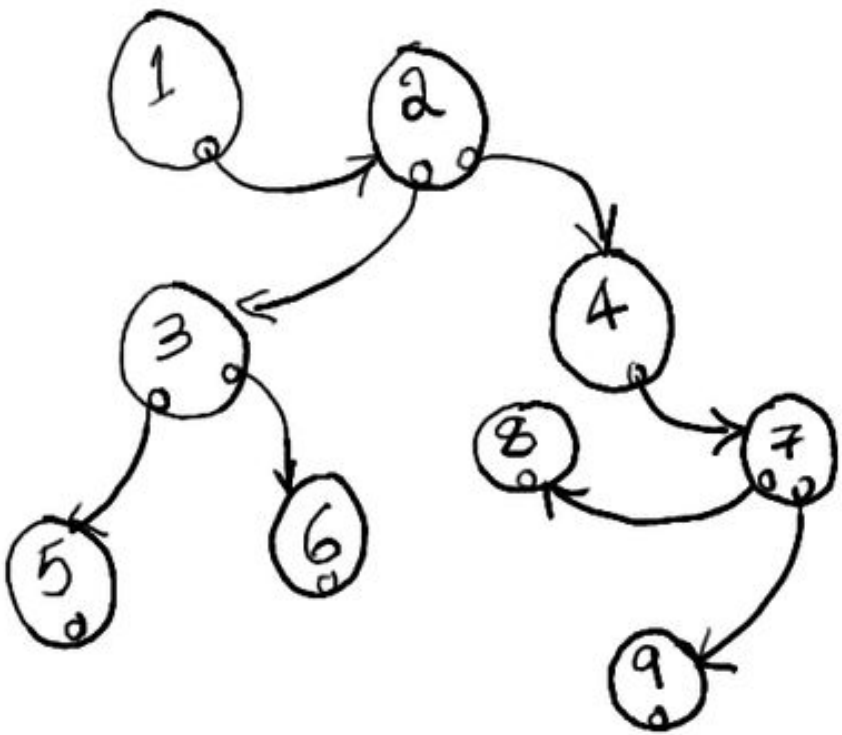
PROGRAMACION ORIENTADA A OBJETOS CON PYTHON



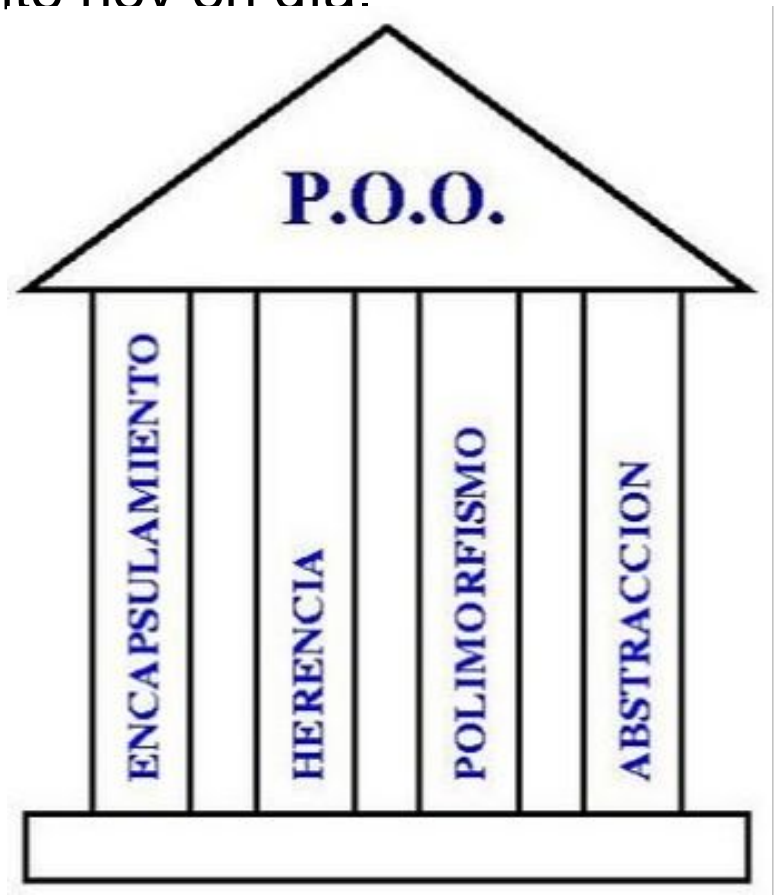
APRENDER PYTHON

CONTENIDO

- Introducción
- Formas de Pensar
- Algunos Paradigmas Habituales
- Programación Multiparadigma
- Tipos de Datos
- Clases y Objetos
- Fisonomía de una Clase
- Un Paseo entre Objetos
- Acceso individualizado a Objetos: self
- Método de inicialización __init__
- Propiedades y Atributos
- Herencia y derivación de clases
- Jerarquías de Clases
- Encapsulación y Grados de Privacidad
- Resumen Final

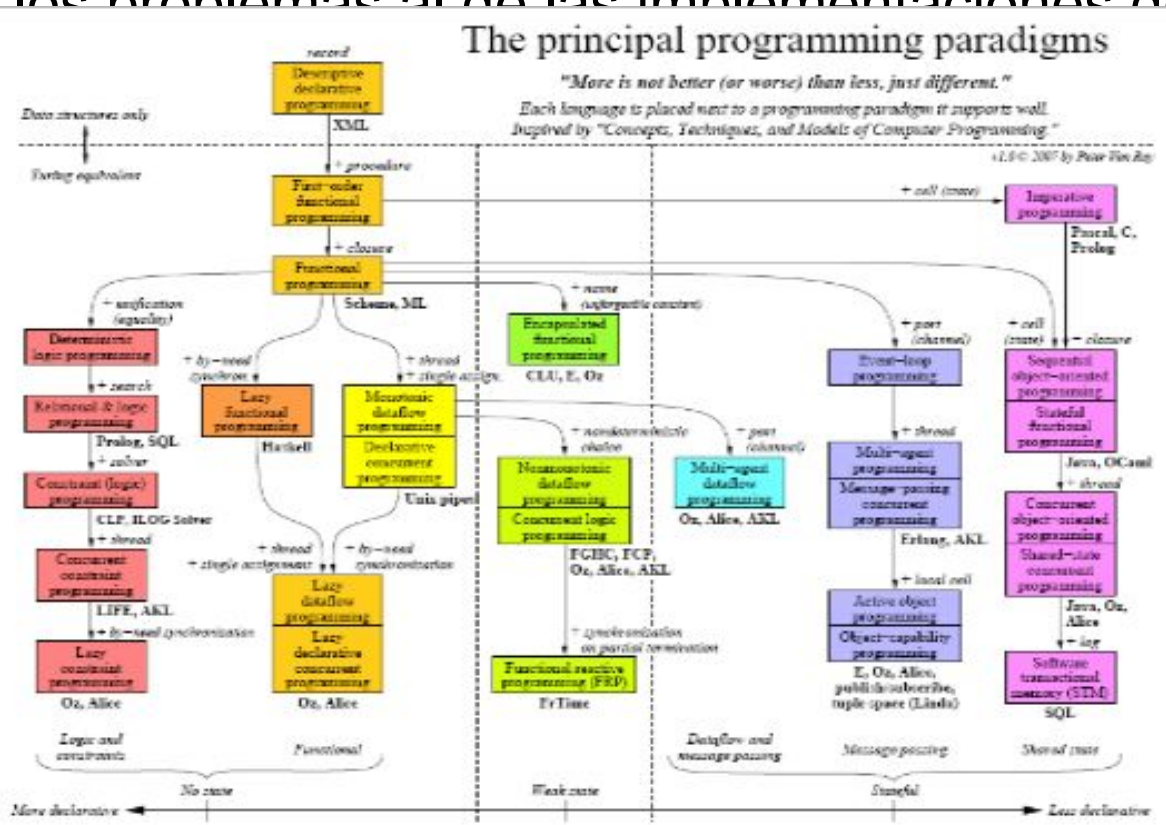
INTRODUCCION

La programación orientada a objetos (POO, o OOP por sus siglas en inglés), uno de los paradigmas de programación estructurada más importante hoy en día.



FORMAS DE PENSAR

Los paradigmas de programación son herramientas conceptuales para analizar, representar y abordar los problemas, presentando sistematizaciones alternativas o complementarias para pasar del espacio de los problemas al de las implementaciones de una solución.



ALGUNOS PARADIGMAS HABITUALES

- La programación Modular
- La programación Procedural
- La programación Estructurada
- La programación Imperativa
- La programación Declarativa
- La programación Funcional

PROGRAMACION MULTIPARADIGMA

Los paradigmas de programación son idealizaciones, y, como tales, no siempre se presentan de forma totalmente ‘pura’, ni siempre resultan incompatibles entre sí. Cuando se entremezclan diversos paradigmas se produce lo que se conoce como programación multiparadigma.

El uso de distintos modelos, según se adapten mejor a las diversas partes de un problema o a nuestra forma de pensamiento, resulta también más natural permite expresar de forma más clara y concisa

TIPOS DE DATOS

Una forma de almacenar y representar una fecha en un programa podría ser la siguiente:

```
d = 14
m = "Noviembre"
a = 2006
def dime_fecha(dia, mes, anho):
return "%i de %s de %i del calendario gregoriano" % (dia, mes, anho)
print dime_fecha(d, m, a)
```

para obtener la siguiente salida:

"14 de Noviembre de 2006"

En este ejemplo, para representar y manipular lo que conceptualmente entendemos como una fecha, se utilizan las variables d, m y a como almacenes de datos, y un procedimiento o función, de nombre ***dime_fecha,***

CLASES Y OBJETOS

- Al enunciar algunos tipos de paradigmas hemos visto que la programación orientada a objetos define los programas en términos de “clases de objetos” que se comunican entre sí mediante el envío de mensajes.

- Las clases surgen de la generalización de los tipos de datos y permiten una representación más directa de los conceptos necesarios para la modelización de un problema permitiendo definir nuevos tipos al usuario.

- Las clases permiten agrupar en un nuevo tipo los datos y las funcionalidades asociadas a dichos datos, favoreciendo la separación entre los detalles de la implementación de las propiedades esenciales para su uso.

FISONOMIA DE UNA CLASE

En python, el esquema para la definición de una clase es el siguiente:

class NombreClase:
<instrucción_1>
...
<instrucción_n>

en donde ***class*** es una palabra reservada que indica la declaración de una clase, ***NombreClase*** una etiqueta que da nombre a la clase y, los dos puntos, que señalan el inicio del bloque de instrucciones de la clase.

UN PASEO ENTRE OBJETOS

Veremos algunas características de los objetos:

- Identidad. Los objetos se diferencian entre sí, de forma que dos objetos, creados a partir de la misma clase y con los mismos parámetros de inicialización, son entes distintos.

- Definen su comportamiento (y operar sobre sus datos) a través de métodos, equivalentes a funciones.

- Definen o reflejan su estado (datos) a través de propiedades o atributos, que pueden ser tipos concreto

ACCESO INDIVIDUALIZADO A OBJETOS: SELF

Hemos visto cómo usar atributos de clase para compartir datos entre todos los objetos de una misma clase, también nos interesa conocer cómo utilizar atributos comunes a una clase pero que puedan tomar valores distintos para cada uno de los objetos.

Retomamos ahora el misterioso parámetro ***self*** que vimos estaba presente como primer parámetro de todos los métodos de una clase. n su momento ya comentamos que ***self*** contiene una referencia al objeto que recibe la señal (el objeto cuyo método es llamado), por lo que podemos usar esa referencia para acceder al espacio de nombres del

EL METODO DE INICIALIZACION __init__

Hemos visto ya cómo definir y usar atributos de clase y de instancia. Pero en python no es necesaria la declaración de variables, por lo que cualquier atributo al que se realice una asignación en el código se convierte en un atributo de instancia:

```
>>> a = F()
>>> a.i = 6
>>> a.atr = 3
>>> print a.atr
3
>>> a.__dict__
```

EL METODO DE INICIALIZACION __init__

Para poder inicializar los atributos de una instancia existe un método especial que permite actuar sobre la inicialización del objeto. Dicho método se denomina ***__init__*** (con dos guiones bajos al principio y al final) y admite cualquier número de parámetros, siendo el primero la referencia al objeto que es inIcializado (s***elf***, por convención), que nos permite realizar las asignaciones a atributos de la instancia.

```
class Clase:
def __init__(self, x=2):
self.x = x
self.a = x**2
self.b = x**3
self.c = 999
def dime_datos(self):
return "Con x=%i obtenemos: a=%i, b=%i, c=%i" % (self.x, self.a, self.b
a1 = Clase(2)
a1.dime_datos() # Salida Con x=2 obtenemos: a=4, b=8, c=999
a2 = Clase(3)
```

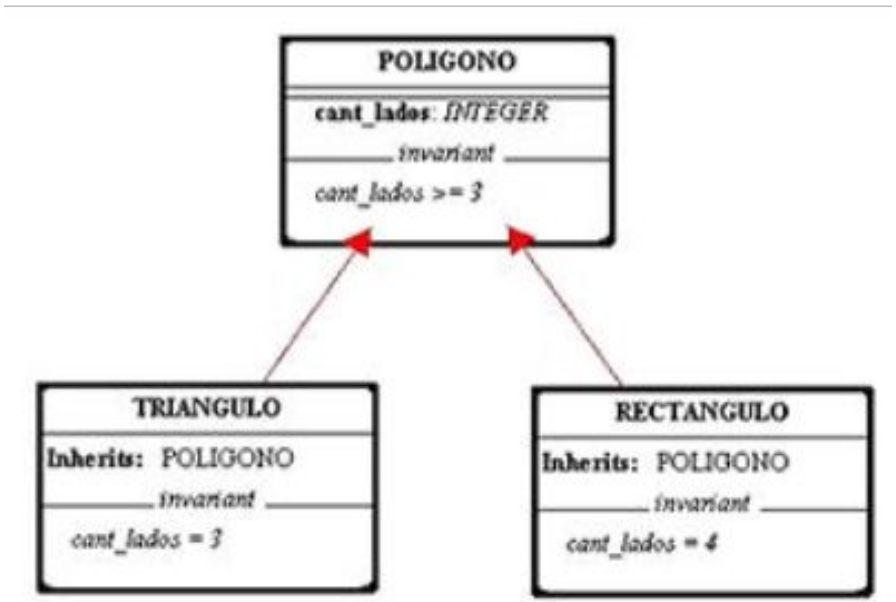
PROPIEDADES Y ATRIBUTOS

Aunque en la terminología general de la programación orientada a objetos atributo y propiedad se pueden utilizar como sinómimos, en python se particulariza el uso de propiedades para un tipo especial de atributos cuyo acceso se produce a través de llamadas a funciones.

```
class ClaseC(object):
def __init__(self):
self.__b = 0
def __get_b(self):
return self.__b
def __set_b(self, valor):
if valor > 10:
self.__b = 0
else:
self.__b = valor
```

PROPIEDADES Y ATRIBUTOS

```
c1 = ClaseC()
print c1.b
# b es 0
c1.b = 5
print c1.b
# b es 5
c2 = ClaseC()
print c2.b
# b es 0
c2.b = 12
print c2.b
#b es 0
```



En Python, para poder usar propiedades en una clase es necesario hacerla derivar de la clase **object** de ahí que aparezca al lado del nombre de la clase ClaseC.
La signatura de la función **property**, que define una propiedad de la clase es la siguiente: **nombre_propiedad = property(get_f, set_f, del_f, doc)** donde **get_f** es la función llamada cuando se produce la lectura del atributo; **set_f** la función llamada cuando se

HERENCIA Y DERIVACION DE CLASES

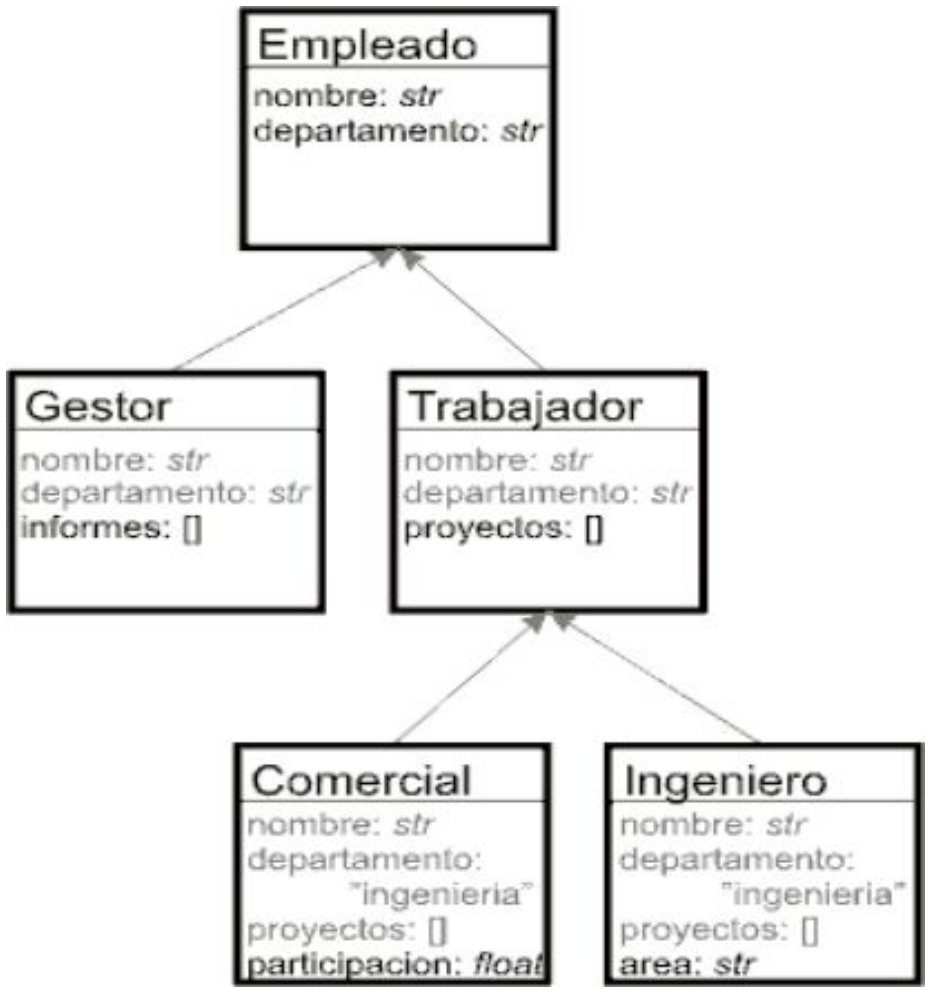
El uso de clases hace más adecuada la representación de conceptos en nuestros programas. Además, es posible generar una clase nueva a partir de otra, de la que recibe su comportamiento y estado (métodos y atributos), adaptándolos o ampliándolos según sea necesario.

Esto facilita la reutilización del código, puesto que se pueden implementar los comportamientos y datos básicos en una clase base y especializarlos en las clases derivadas.

En el lenguaje python, para expresar que una clase deriva, desciende o es heredera de otra u otras clases se añade tras el nombre, en la declaración de la clase, una tupla con los nombres de las clases base.

JERARQUIAS DE CLASES

La herencia permite establecer relaciones entre clases, y, estas relaciones de pertenencia pueden ser a varios niveles, y ramificarse en lo que se denominan jerarquías de clases.



APRENDER PYTHON

ENCAPSULACION Y GRADOS DE PRIVACIDAD

Uno de los principios que guían la POO, heredados de la programación modular, es el de encapsulación. Hemos visto cómo se puede agrupar comportamiento y datos gracias al uso de objetos, pero hasta el momento, tanto los atributos como los métodos que se definen en las clases correspondientes se convierten en métodos y atributos visibles para los usuarios de la clase (la API de la clase).

Python utiliza para ello convenciones a la hora de nombrar métodos y atributos, de forma que se señale su carácter privado, para su exclusión del espacio de nombres, o para indicar una función especial, normalmente asociada a funcionalidades estándar del lenguaje.

_nombre

Los nombres que comienzan con un único guión bajo indican de forma débil un uso interno. Además, estos nombres no se incorporan en el espacio de nombres de un módulo al importarlo con "***from ... import ****".

__nombre

APRENDER PYTHON

RESUMEN FINAL

La programación orientada a objetos enuncia la posibilidad de escribir un programa como un conjunto de clases de objetos capaces de almacenar su estado, y que interactúan entre sí a través del envío de mensajes.

Las técnicas más importantes utilizadas en la programación orientada a objetos son:

Abstracción: Los objetos pueden realizar tareas, interactuar con otros objetos, o modificar e informar sobre su estado sin necesidad de comunicar cómo se realizan dichas acciones.

Encapsulación (u ocultación de la información): los objetos impiden la modificación de su estado interno o la llamada a métodos internos por parte de otros objetos, y solamente se relacionan a través de una interfaz clara que define cómo se relacionan con otros objetos.

Polimorfismo: comportamientos distintos pueden estar asociados al mismo nombre

Herencia: los objetos se relacionan con otros estableciendo jerarquías, y es posible que unos objetos hereden las propiedades y métodos de otros objetos, extendiendo su comportamiento y/o especializándolo. Los objetos se agrupan así en clases que forman jerarquías.

Las clases definen el comportamiento y estado disponible que se concreta en los objetos.

Los objetos se caracterizan por:

- Tener identidad. Se diferencian entre sí.
- Definir su comportamiento a través de métodos.
- Definir o reflejar su estado a través de propiedades y atributos.

BIBLIOGRAFIA

Esta información ha sido extraído de la pagina web http://blog.rvburke.com cumpliendo con la norma de copyright establecida.