INFORMATORIO

Taller de Programación Web

Clases y Objetos

Subsecretaría de Empleo
Chaco Gobierno de todos

Ministerio de Producción, Industria y Empleo
Chaco Gobierno de todos

CHACO
Gobierno de todos

Abstracción

La técnica de la abstracción consiste en aislar un determinado elemento sobre su contexto o el resto de elementos que lo acompañan y que posiblemente no sean válidos para nuestro propósito. Gracias a la abstracción separamos las características esenciales de un objeto de las no esenciales y tomaremos las características y propiedades a tener en cuenta de un objeto en el mundo real con el fin de utilizarlas en el sistema que tratamos de implementar.
En POO solo necesita saber cómo interaccionar con los objetos, no necesita conocer los detalles de cómo está implementada la clase a partir de la cual se instancia el objeto. Sólo necesita conocer su interfaz pública.
El nivel de abstracción puede ser bajo (en un objeto se manipulan datos y métodos individualmente), o alto (en un objeto solo se usan sus métodos de servicio).

Encapsulación

La encapsulación es una forma de abstracción, además es un mecanismo para llevar a la práctica la abstracción.
Cuando una clase existe (se define), se crean objetos a partir de ella, y se usan dichos objetos llamando los métodos necesarios. Es decir, crea objetos para usar los servicios que nos proporciona la clase a través de sus métodos.
No necesita saber cómo trabaja el objeto, ni saber las variables que usa, ni el código que contiene.
Este principio otorga beneficios importantes para el desarrollo de software:

- **Modularidad**: El código fuente para un objeto puede ser escrito y mantenido independientemente del código fuente para otros objetos. Además, un objeto puede ser fácilmente pasado de un lugar a otro del sistema. Por ejemplo, en el caso de la bicicleta de lecturas anteriores, es posible darle tu bicicleta a alguien más, y continuará trabajando.
- **Ocultamiento de información**: Un objeto tiene una interfaz pública que otros objetos pueden utilizar para comunicarse con él. El objeto puede mantener información y métodos privados que pueden ser modificados a cualquier tiempo sin afectar los otros objetos que dependen de él. No es necesario entender cómo usar el mecanismo de los engranajes de una bicicleta para poder utilizarlo.

La encapsulación describe el hecho de que los objetos se usan como cajas negras. Así, un objeto encapsula datos y métodos, que están dentro del objeto.

- **Interfaz pública de una clase**: Es el conjunto de métodos (métodos de servicio) que sirve para que los objetos de una clase proporcionen sus servicios. Estos servicios son los que pueden ser llamados por un cliente.
- **Métodos de soporte**: Son métodos adicionales en un objeto que no definen un servicio utilizable por un cliente, pero que ayudan a otros métodos en sus tareas.

La encapsulación es un mecanismo de control que restringe la capacidad de modificar el estado (el conjunto de propiedades, atributos o datos) de un objeto sólo por medio de los métodos del propio objeto, permitiendo que los atributos de un objeto pueden ocultarse (superficialmente) para que no sean accedidos desde fuera de la definición de una clase. Para ello, es necesario nombrar los atributos con un prefijo de doble subrayado: ***__atributo***.

Encapsulamiento y niveles de privacidad

En encapsulamiento nos permite definir niveles de privacidad para señalar qué métodos y atributos no deben utilizarse fuera de la clase y así evitar exponer excesivamente los detalles de la implementación de una clase o un objeto.
Esto se consigue en otros lenguajes de programación como Java utilizando modificadores de acceso que definen si cualquiera puede acceder a esa función o variable(public) o si está restringido el acceso a la propia clase (private).
En Python no existen los modificadores de acceso, y lo que se suele hacer es usar 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**: Los nombres que empiezan por dos guiones bajos (y no termina también con dos guiones bajos) indican su uso privado en la clase.
- **__nombre__**: Los nombres que empiezan y acaban con dos guiones bajos indican atributos "mágicos", de uso especial y que residen en espacios de nombres que puede

manipular el usuario. Solamente deben usarse en la manera que se describe en la documentación de Python y debe evitarse la creación de nuevos atributos de este tipo. Algunos ejemplos de nombres "singulares" de este tipo son:

- __init__, método de inicialización de objetos
- __del__, método de destrucción de objetos
- __doc__, cadena de documentación de módulos, clases...
- __class__, nombre de la clase
- __str__, método que devuelve una descripción de la clase como cadena de texto
- __repr__, método que devuelve una representación de la clase como cadena de texto
- __module__, módulo al que pertenece la clase

En el siguiente ejemplo sólo se imprimirá la cadena correspondiente al método ***publico()***, mientras que al intentar llamar al método ***__privado()*** Python lanzará una excepción debido a que no existe, evidentemente existe pero no lo podemos ver porque es privado.

Código

```
Definir una clase con métodos públicos y privados.


class Ejemplo:
      def publico(self):
          print("Publico")

      def __privado(self):
          print("Privado")

ej = Ejemplo()
ej.publico()
ej.__privado()
```

Resultado

```
Público
Traceback (most recent call last):
  File "<input>", line 11, in <module>
AttributeError: 'Ejemplo' object has no attribute '__privado'
```

Este mecanismo se basa en que los nombres que comienzan con un doble guión bajo se renombran para incluir el nombre de la clase (característica que se conoce con el nombre de name mangling). Esto implica que el método o atributo no es realmente privado, y podemos acceder a él mediante una pequeña trampa:

Código

```
Definir una clase con métodos públicos y privados.


class Ejemplo:
      def publico(self):
          print("Publico")

      def __privado(self):
          print("Privado")

ej = Ejemplo()

ej.publico()
ej._Ejemplo__privado()
```

Resultado

```
Público
Privado
```

En ocasiones también puede suceder que queramos permitir el acceso a algún atributo de nuestro objeto, pero que este se produzca de forma controlada. Para esto podemos escribir métodos cuyo único cometido sea este, métodos que normalmente, por convención, tienen nombres como getVariable y setVariable; de ahí que se conozcan también con el nombre de getters y setters.

Código

```
Definir una clase con sus atributos privados y sus métodos getters y setters.

class Fecha():
      def __init__(self):
          self.__dia = 1

      def getDia(self):
          return self.__dia

      def setDia(self, dia):
          if dia > 0 and dia < 31:
              self.__dia = dia
          else:
              print("Error")

mi_fecha = Fecha()

mi_fecha.setDia(33)

mi_fecha.__dia= 30
print(mi_fecha.getDia())
```

Resultado

```
Error
1
```

Esto se podría simplificar mediante propiedades, que abstraen al usuario del hecho de que se está utilizando métodos accesores para obtener y modificar los valores del atributo:

Código

Definir una clase con sus atributos privados y sus métodos getters y setters.

```
class Fecha(object):
      def __init__(self):
          self.__dia = 1

      def getDia(self):
          return self.__dia

      def setDia(self, dia):
          if dia > 0 and dia < 31:
              self.__dia = dia
          else:
               print("Error")

     dia = property(getDia, setDia)

mi_fecha = Fecha()

mi_fecha.dia = 30

print(mi_fecha.dia)
```

Resultado

```
30
```