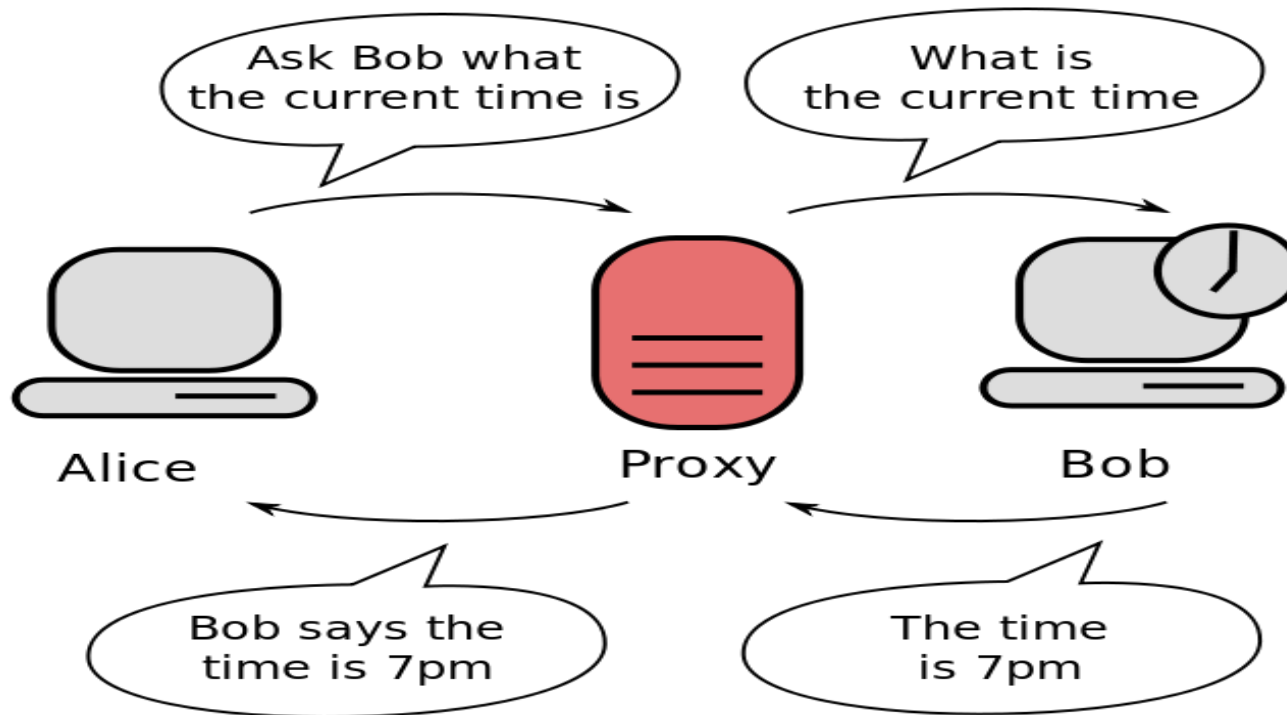


# Proxy Design Pattern

# Que es el patron proxy

.Consiste en interponer un intermediario (proxy) entre un objeto y los demas que lo utilizan



- .Da soporte a objetos que controlan la creacion y el acceso a otros objetos**
- .Puede representar a un objeto que toma tiempo o recursos en crearse, de esta forma el cliente piensa que esta comunicado con el objeto, mientras el proxy decide la creacion cuando sea realmente necesario.**
- .El sujeto mantiene una cuenta de referencias, solo cuando se hace el objeto.**

# Problema

- .El patrón proxy se usa cuando se necesita una referencia a un objeto más flexible o sofisticada que un puntero. Dependiendo de la función que se desea realizar con dicha referencia podemos distinguir diferentes tipos de proxies:**
- .Proxy remoto: representante local de un objeto remoto.**
- .Proxy virtual: crea objetos costosos bajo demanda.**

**.Proxy de protección: controla el acceso al objeto original.**

**.Proxy de referencia inteligente: sustituto de un puntero que lleva a cabo operaciones adicionales cuando se accede a un objeto.**

# Elementos

- **Proxy remoto:** responsable de codificar una petición y sus argumentos, y de enviarla al objeto remoto.
- **Proxy virtual:** puede hacer caché de información del objeto real para diferir en lo posible el acceso a este.
- **Proxy de protección:** comprueba que el cliente tiene los permisos necesarios para realizar la petición.

**Subject:** Define la interfaz común para el

# Pros

- .Mantiene una referencia que permite al proxy acceder al objeto real.**
- .Proporciona una interfaz igual que la del sujeto real.**
- .Controla el acceso al sujeto real y puede ser responsable de crearlo y borrarlo.**
- .Copiar un objeto grande puede ser costoso, si la copia no se modifica, no hay necesidad de incurrir en dicho gasto.**

# Contras

- .Un proxy remoto puede ocultar el hecho de que un objeto reside en otro espacio de direcciones.**
- .Un proxy virtual puede optimizarse, como crear objetos de bajo demanda.**
- .Tanto los proxy de proteccion, como las referencias inteligentes permiten realizar tareas de un mantenimiento adicionales cuando se accede a un objeto,**

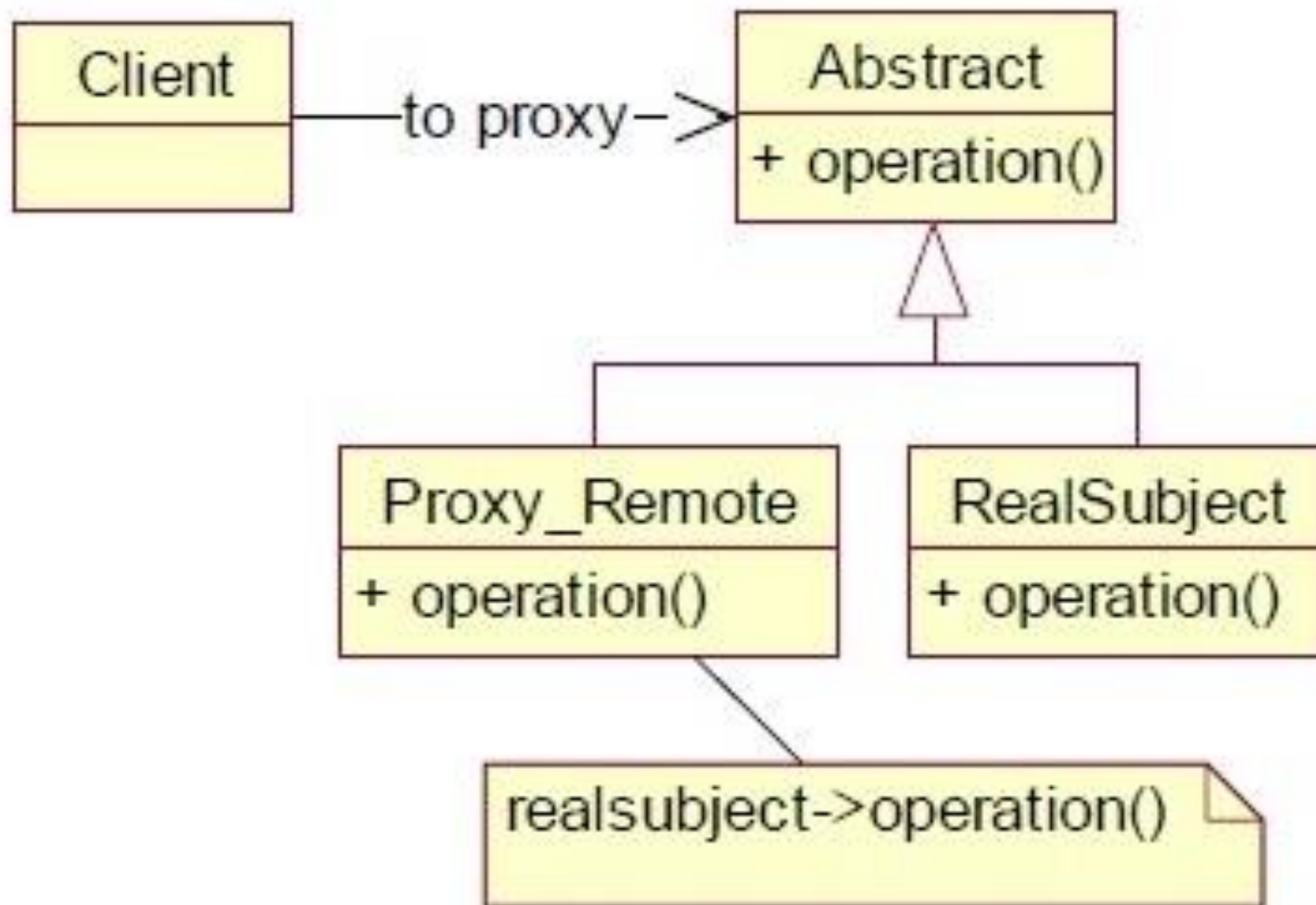


# Patrones con relacion

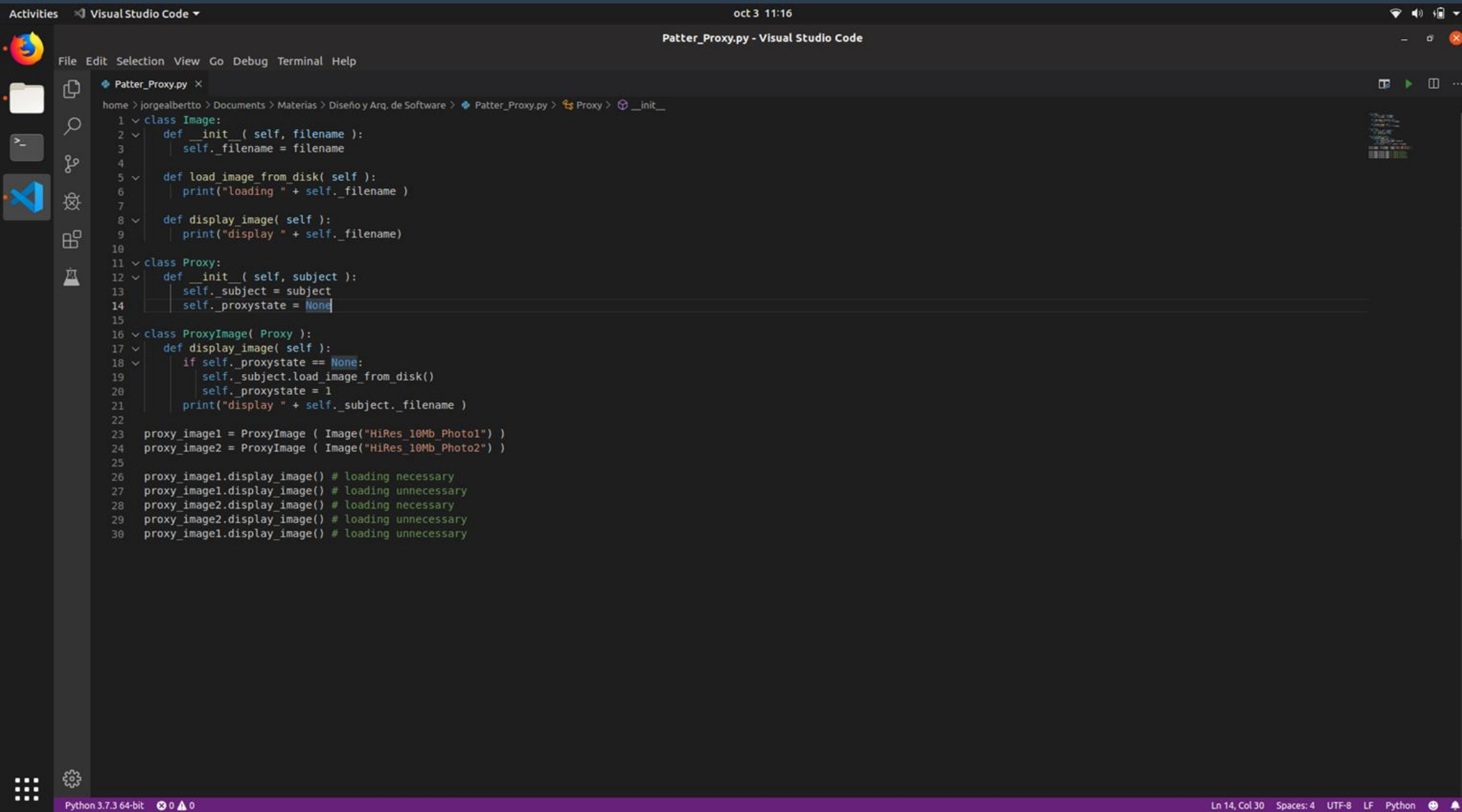
- **Adapter proporciona al objeto que está adaptando una interfaz diferente, mientras que Apoderado presenta la misma interfaz que su representado. No obstante, en el caso del apoderado de protección se pueden rechazar peticiones que el representado sí puede atender, con lo que, de hecho, se está modificando su interfaz.**
- **Decorator puede tener una implementación similar a la de Apoderado, a pesar de tener una**

# Ejemplos de aplicacion

- **Añadir acceso de seguridad a un objeto existente. El proxy determinará si el cliente puede acceder al objeto de interés (proxy de protección).**
- **Proporcionando interfaz de recursos remotos como el servicio web o recursos REST.**
- **Coordinación de las operaciones costosas en recursos remotos pidiendo los recursos a distancia para iniciar la operación tan pronto como sea posible antes de acceder a los**



# Codigo



```
home > jorgealberto > Documents > Materias > Diseño y Arq. de Software > Patter_Proxy.py > Proxy > __init__
1 class Image:
2     def __init__( self, filename ):
3         self.filename = filename
4
5     def load_image_from_disk( self ):
6         print("loading " + self.filename )
7
8     def display_image( self ):
9         print("display " + self.filename)
10
11 class Proxy:
12     def __init__( self, subject ):
13         self.subject = subject
14         self.proxystate = None
15
16 class ProxyImage( Proxy ):
17     def display_image( self ):
18         if self.proxystate == None:
19             self.subject.load_image_from_disk()
20             self.proxystate = 1
21         print("display " + self.subject.filename )
22
23 proxy_image1 = ProxyImage ( Image("HiRes_10Mb_Photo1") )
24 proxy_image2 = ProxyImage ( Image("HiRes_10Mb_Photo2") )
25
26 proxy_image1.display_image() # loading necessary
27 proxy_image1.display_image() # loading unnecessary
28 proxy_image2.display_image() # loading necessary
29 proxy_image2.display_image() # loading unnecessary
30 proxy_image1.display_image() # loading unnecessary
```