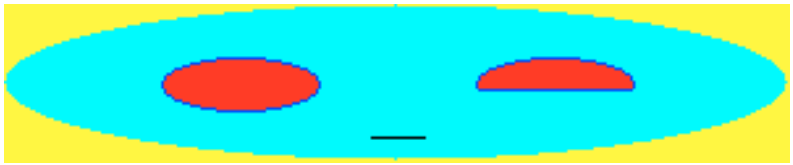
Design Pattern Proxy: esempio

- Abbiamo bisogno di creare immagini delle quali però una sola verrà usata realmente alla fine.
- Abbiamo un modulo Image e un modulo quasi equivalente più veloce cyImage. Entrambi i moduli creano le loro immagini in memoria.
- Siccome avremo bisogno solo di un'immagine tra quelle create, sarebbe meglio utilizzare dei proxy "leggeri" che permettano di creare una vera immagine solo quando sapremo di quale immagine avremo bisogno.
- L'interfaccia Image.Image consiste di 10 metodi in aggiunta al costruttore: load(), save(), pixel(), set_pixel(), line(), rectangle(), ellipse(), size(), subsample(), scale().
  - Non sono elencati alcuni metodi statici aggiuntivi, quali Image.Image.color_for_name() e Image.color_for_name().

Design Pattern Proxy: esempio



```
YELLOW, CYAN, BLUE, RED, BLACK = (Image.color_for_name(color)
    for color in ("yellow", "cyan", "blue", "red", "black"))
image = ImageProxy(Image.Image, 300, 60)
image.rectangle(0, 0, 299, 59, fill=YELLOW)
image.ellipse(0, 0, 299, 59, fill=CYAN)
image.ellipse(60, 20, 120, 40, BLUE, RED)
image.ellipse(180, 20, 240, 40, BLUE, RED)
image.rectangle(180, 32, 240, 41, fill=CYAN)
image.line(181, 32, 239, 32, BLUE)
image.line(140, 50, 160, 50, BLACK)
image.save(filename)
```

Design Pattern Proxy: esempio

- La classe ImageProxy può essere usata al posto di Image.Image (o di qualsiasi altra classe immagine che supporta l'interfaccia di Image) a patto che l'interfaccia incompleta fornita da ImageProxy sia sufficiente.
- Un oggetto ImageProxy non salva un'immagine ma mantiene una lista di tuple di comandi dove il primo elemento in ciascuna tupla è una funzione o un metodo unbound (non legato ad una particolare istanza) e i rimanenti elementi sono gli argomenti da passare quando la funzione o il metodo è invocato.

```
class ImageProxy:

    def __init__(self, ImageClass, width=None, height=None, filename=None):
        assert (width is not None and height is not None) or \
                filename is not None
        self.Image = ImageClass
        self.commands = []
        if filename is not None:
            self.load(filename)
        else:
            self.commands = [(self.Image, width, height)]

    def load(self, filename):
        self.commands = [(self.Image, None, None, filename)]
```


Design Pattern Proxy: esempio

- Quando viene creato un ImageProxy, gli deve essere fornita l'altezza e la larghezza dell'immagine o il nome di un file. In caso contrario viene lanciata AssertionError.
- Se viene fornito il nome di un file, l'ImageProxy immagazzina una tupla con il costruttore Image.Image(), None e None (per la larghezza e l'altezza) e il nome del file da cui il metodo load di ImageClass carichera` le informazioni per costruire l'immagine.
- Se non viene fornito il nome di un file allora viene immagazzinato il costruttore Image.Image() insieme alla larghezza e l'altezza.

```
class ImageProxy:

    def __init__(self, ImageClass, width=None, height=None, filename=None):
        assert (width is not None and height is not None) or \
                filename is not None
        self.Image = ImageClass
        self.commands = []
        if filename is not None:
            self.load(filename)
        else:
            self.commands = [(self.Image, width, height)]

    def load(self, filename):
        self.commands = [(self.Image, None, None, filename)]
```

Design Pattern Proxy: esempio

- La classe Image.Image ha 4 metodi: line(), rectangle(), ellipse(), set_pixel().
- La classe ImageProxy supporta pienamente questa interfaccia solo che invece di eseguire questi comandi, semplicemente li aggiunge insieme ai loro argomenti alla lista dei comandi.
- Il metodo inserito all'inizio della tupla è unbound in quanto non è legato ad un'istanza di self.Image (self.Image è la classe che fornisce il metodo)

```
def set_pixel(self, x, y, color):
    self.commands.append((self.Image.set_pixel, x, y, color))

def line(self, x0, y0, x1, y1, color):
    self.commands.append((self.Image.line, x0, y0, x1, y1, color))

def rectangle(self, x0, y0, x1, y1, outline=None, fill=None):
    self.commands.append((self.Image.rectangle, x0, y0, x1, y1,
            outline, fill))

def ellipse(self, x0, y0, x1, y1, outline=None, fill=None):
    self.commands.append((self.Image.ellipse, x0, y0, x1, y1,
            outline, fill))
```


Design Pattern Proxy: esempio

- Solo quando si sceglie di salvare l'immagine, essa viene effettivamente creata e viene quindi pagato il prezzo relativo alla sua creazione, in termini di computazione e uso di memoria.
- Il primo comando della lista self.commands è sempre quello che crea una nuova immagine. Quindi il primo comando viene trattato in modo speciale salvando il suo valore di ritorno (che è un Image.Image o un cyImage.Image) in image.
- Poi vengono invocati nel for i restanti comandi passando image come argomento insieme agli altri argomenti.
- Alla fine, si salva l'immagine con il metodo Image.Image.save().

```
def save(self, filename=None):
    command = self.commands.pop(0)
    function, *args = command
    image = function(*args)
    for command in self.commands:
        function, *args = command
        function(image, *args)
    image.save(filename)
    return image
```

Design Pattern Proxy: esempio

- Il metodo Image.Image.save() non ha un valore di ritorno (sebbene possa lanciare un'eccezione se accade un errore).
- L'interfaccia è stata modificata leggermente per ImageProxy per consentire a save() di restituire l'immagine Image.Image creata per eventuali ulteriori usi dell'immagine.
- Si tratta di una modifica innocua in quanto se il valore di ritorno è ignorato, esso viene scartato.

```
    def save(self, filename=None):
        command = self.commands.pop(0)
        function, *args = command
        image = function(*args)
        for command in self.commands:
            function, *args = command
            function(image, *args)
        image.save(filename)
        return image
```


Design Pattern Proxy: esempio

- Se un metodo non supportato viene invocato (ad esempio, pixel()), Python lancia un AttributeError.
- Un approccio alternativo per gestire i metodi che non sono supportati dal proxy è di creare una vera immagine non appena uno di questi metodi è invocato e da quel momento usare la vera immagine.

Design Pattern Proxy: esempio

Questo codice prima crea alcune costanti colore con la funzione color_for_name del modulo Image e poi crea un oggetto ImageProxy passando come argomento a __init__ la classe che si vuole usare. L'ImageProxy creato è usato quindi per disegnare e infine salvare l'immagine risultante.

```
YELLOW, CYAN, BLUE, RED, BLACK = (Image.color_for_name(color)
    for color in ("yellow", "cyan", "blue", "red", "black"))
image = ImageProxy(Image.Image, 300, 60)
image.rectangle(0, 0, 299, 59, fill=YELLOW)
image.ellipse(0, 0, 299, 59, fill=CYAN)
image.ellipse(60, 20, 120, 40, BLUE, RED)
image.ellipse(180, 20, 240, 40, BLUE, RED)
image.rectangle(180, 32, 240, 41, fill=CYAN)
image.line(181, 32, 239, 32, BLUE)
image.line(140, 50, 160, 50, BLACK)
image.save(filename)
```


93

Design Pattern Proxy: esempio

- Il codice alla pagina precedente avrebbe funzionato allo stesso modo se avessimo usato Image.Image() al posto di ImageProxy().
- Usando un image proxy, la vera immagine non viene creata fino a che il metodo save non viene invocato. In questo modo il costo per creare un'immagine prima di salvarlo è estremamente basso (sia in termini di memoria che di computazione) e se alla fine scartiamo l'immagine senza salvarla perdiamo veramente poco.
- Se usassimo Image.Image, verrebbe effettivamente creato un array di dimensioni width × height di colori e si farebbe un costoso lavoro di elaborazione per disegnare (ad esempio, per settare ogni pixel del rettangolo) che verrebbe sprecato se alla fine scartassimo l'immagine.

94

Esercizio 24 ottobre -1

1. Scrivere una classe MyProxy che è il proxy della classe MyClass. Ogni volta che viene invocato un metodo di istanza della classe MyProxy, di fatto viene invocato l'omonimo metodo di istanza di MyClass. **NON deve essere usata l'ereditarietà.**
   - **Si assuma che __init__ di MyClass prenda in input unn argomento x e che il comportamento dei suoi metodi di istanza dipenda dal valorie di x passati a __init__.**

Esercizio 24 ottobre

- Scrivere un decorator factory che prende in input una classe ClasseConFF e due stringhe funz e ff e restituisce un decoratore di classe che decora una classe in modo tale che se viene invocata funz di fatto al posto di funz viene invocata la funzione ff della classe ClasseConFF.