Mètodes especials

Tecnologia de la Programació

Sebastià Vila-Marta

Enginyeria de Sistemes TIC
Universitat Politècnica de Catalunya
http://epsem.upc.edu

23 de febrer de 2020

En el tema anterior...

- Una instància pot tenir com atribut una altra instància.
- Associació de composició.
- Principi d'«information hiding».
- Arguments per omissió en paràmetres.

Escriure instàncies amb dignitat I

Quan escrivim una instància, obtenim quelcom misteriós:

```
>>> w = Wallet(q2=3)
>>> print w
<__main__.Wallet instance at 0xb74627cc>
```

Això s'ha de llegir com «w referencia una instància de Wallet que viu a l'adreça de memòria 0xb74627cc».
Seria preferible que escrivís quelcom semblant a:

```
>>> w = Wallet(q2=3)
>>> print w
wallet: [1 => 0, 2 => 3, 5 => 0]
```

o encara millor:

```
>>> w = Wallet(q2=3)
>>> print w
wallet: [2 => 3]
```

Escriure instàncies amb dignitat II

A tal efecte podem definir el següent mètode a la classe Wallet:

```
def __str__(self):
    tmp = ['{0}=>{1}'.format(k,v) for k,v in self.mon.items() if v != 0]
    return '[' + ', '.join(tmp) + ']'
```

amb aquesta nova classe Wallet l'experiment anterior resulta ser:

```
>>> w = Wallet(q2=3)
>>> print w
wallet: [2 => 3]
```

Què ha succeït?

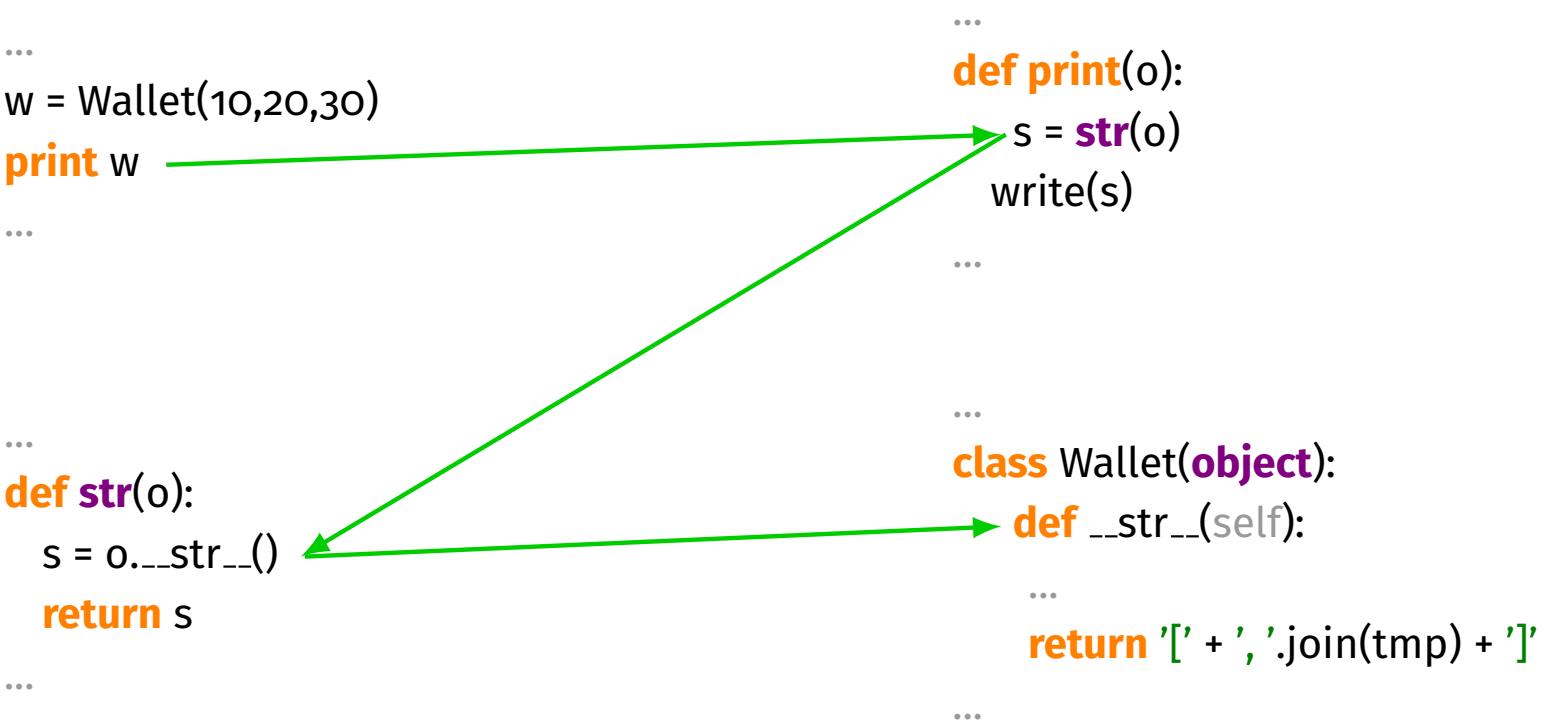
Escriure instàncies amb dignitat III

El mètode especial __str__ és un mètode que s'invoca automàticament quan apliquem la funció **str** a una instància. Fixeu-vos:

```
>>> w = Wallet(q2=3)
>>> str(w)
wallet: [2 => 3]
```

D'aquest patró o esquema en diem delegació. La funció **str** delega la feina de convertir a cadena de caràcters una instància al mètode __str__ de la pròpia instància en cas que existeixi. Aquest és l'únic secret.
La sentència **print** usa la funció **str** sobre la instància que es vol imprimir per tal de convertir-la en una cadena amb la finalitat de poder-la escriure.

Delegació en el print



Mètodes especials

Els mètodes d'un objecte el nom dels quals està envoltat per dues ratlles baixes, com el cas de __init__ o __str__, són mètodes especials.
Hi ha mètodes especials per a:

- Construir l'objecte. Com __init__.
- Convertir-lo a altres tipus. Com __str__.
- Permetre operacions aritmètiques sobre l'objecte. Com __add__.
- Permetre operacions d'accés específiques sobre l'objecte. Com __getitem__.
- etc.

Exercici

- Hi ha molts mètodes especials interessants.
- És important tenir al cap quins són per poder-hi recórrer quan calgui.
- La referència principal de tots els mètodes especials és: `http://docs.python.org/reference/datamodel.html#special-method-names`
- Feu una llegida d'aquesta documentació i preneu nota dels mètodes més interessants.

Funcions booleanes. Termes

- Una funció booleana f de k variables és una funció definida:

$$f : \mathbb{B}^k \longrightarrow \mathbb{B}$$

essent $\mathbb{B} = \{T, F\}$ el conjunt dels booleans.

- Un terme és un producte booleà en que cada variable participa com a molt una única vegada (complementada o no). Un terme sobre 5 variables pot ser:

$$f(\mathbf{x}) = x_0 \cdot \bar{x}_1 \cdot \bar{x}_3 \cdot x_4$$

Noteu que x_2 és un *don't care*.

- Un terme amb *don't care*'s equival a una suma de termes:

$$\begin{aligned} f(\mathbf{x}) &= x_0 \cdot \bar{x}_1 \cdot \bar{x}_3 \cdot x_4 \\ &= x_0 \cdot \bar{x}_1 \cdot \bar{x}_2 \cdot \bar{x}_3 \cdot x_4 + x_0 \cdot \bar{x}_1 \cdot x_2 \cdot \bar{x}_3 \cdot x_4 \end{aligned}$$

La classe Term I

- Usarem com exemple la classe Term, que representa un producte de variables booleanes:
- Anem a definir una classe per representar aquests termes. Usarem com a representació un tuple amb tants elements com variables i la següent correspondència:

$$\begin{aligned} \bar{x}_i &\longrightarrow 0 \\ x_i &\longrightarrow 1 \\ \text{DC} &\longrightarrow 2 \end{aligned}$$

El terme $f(\mathbf{x}) = x_0 \cdot \bar{x}_1 \cdot \bar{x}_3 \cdot x_4$ es representaria amb el tuple (1,0,2,0,1).

La classe Term II

- Una primera aproximació a la classe ens condueix a:

```
class Term(object):

    def __init__(self, t):
        """Construim a partir d'un tuple t"""
        self.t = t

    def __str__(self):
        """Retorna el minterm ben escrit"""
        c="ABCDEFGHIJKLMNOPQRST"
        l=[]
        for i,v in enumerate(self.t):
            if v == 0:
                l.append(c[i]+"'")
            elif v == 1:
                l.append(c[i])
        return ''.join(l)
```

La classe Term III

- Una operació interessant és la potència, que ens indica el nombre de *don't care*'s. Afegim-la:

```
def potencia(self):
    return self.t.count(2)
```

- També seria interessant poder interrogar el nombre de variables d'una instància de Term. La manera natural de fer-ho seria usant la funció **len**, però si fem:

```
>>> len(t)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: object of type 'Term' has no len()
```

La classe Term IV

- Efectivament, **len** no és una funció definida per les instàncies de Term. Els mètodes especials ens poden rescatar. Afegim el següent mètode a Term:

```
def __len__(self):
    return len(self.t)
```

  i...Voilà!

- Com sabeu, un Term pot representar a un conjunt de minterms. Per exemple, el terme de tres variables $x_0\bar{x}_2$ representa al conjunt de termes $\{x_0x_1\bar{x}_2, x_0\bar{x}_1\bar{x}_2\}$. Es natural, dotar a la classe Term de l'operació de pertinença. Podem fer-ho així:

```
def __contains__(self, t):
    return all([a==b for a,b in zip(self.t, t) if a!=2])
```

La classe Term V

- Amb aquesta operació la classe ha guanyat notablement. Vegem-ne una sessió de treball:

```
>>> t1 = Term((1,2,0))
>>> t2 = Term((1,0,0))
>>> len(t1)
3
>>> t2 in t1
True
>>> Terme((1,0,1)) in t1
False
>>> print t1
AC'
```

Com veieu, els mètodes especials permeten que les instàncies de classes definides per nosaltres puguin ser usades amb la mateixa amabilitat que les instàncies de classes predefinides.

Feina per aquesta setmana

1. Estudi de la teoria. A partir de la referència principal i les transparències. Inclou provar els conceptes en el computador.
2. Confecció d'un xuletari i un glossari del tema.
3. Solució dels problemes del tema.
4. Solució del problema especial, que té com objectiu aconseguir la solució més senzilla i entenedora possible.