Introducció a la Programació
Grau en Estadística
Universitat Autònoma de Barcelona

Sessió 7: Classes i objectes

Vicenç Soler

Índex

1. Introducció

Alguna vegada haureu sentit a parlar de la Programació Orientada a Objectes. En aquesta sessió es començarà a tractar aquest tema, iniciant-nos en les classes i objectes.

2. Classes

Fins ara hem vist que les dades les podem guardar a variables, i que si necessitem guardar un nombre important la millor opció és fer servir arrays (llistes, diccionaris, etc.).

Ara veurem una altra manera de guardar dades: les classes. Una classe és un tipus de dada (com els enters, strings, etc.) que permet guardar més d'una dada i que també permet tenir funcions associades. Per a definir una classe, la seva sintaxi és:

class <nom de la classe> :

variables (també anomenades "propietats de la classe")

funció inicialitzadora __init__ (anomenada "constructor")

altres funcions (anomenades "funcions o mètodes de la classe")

}

Per a il·lustrar-ho, veiem un exemple. Imaginem que volem treballar amb punts definits per coordenades (x,y). Si volguéssim llegir aquests punts d'un fitxer, el que faríem seria emmagatzemar-los en una llista, però com hem de guardar 2 valors de cada punt, el que hauríem de fer és crear 2 llistes: una per les 'x' i l'altra per les 'y', o bé una matriu de n files i 2 columnes.

En aquest punt és on es poden aplicar les classes i els objectes. Definiríem una classe anomenada Punt que tingués 2 propietats (variables). Ho faríem escrivint:

```
class Punt:
    x=0
    y=0
```

No cal que inicialitzem aquestes 2 variables com acabem de fer, però s'ha fet per a aclarir quines variables conté.

Amb lo escrit a dalt, el que hem fet és, resumint, crear un nou tipus de dada. Tenim: enters, floats, strings, etc. I també "Punt". Per tant, puc crear variables de tipus "Punt" i treballar amb elles. Per a accedir a cadascuna de les 2 propietats, ho farem via el símbol "." .

Les variables de tipus Punt s'anomenaran objectes, com veurem a la següent secció.

3. Objectes

Declaració de variable (objecte).

Per a declarar una variable com a tipus Punt, hem de fer-ho de la següent manera:

```
1.  p1 = Punt()
```

inclosos els parèntesi. Si no inicialitzem la variable d'aquesta manera, no podem treballar amb ella.

La variable 'p1' és de tipus punt i l'anomenarem que **és un objecte de la classe Punt**. Cada variable de tipus classe serà un **objecte d'aquella classe**.

Cada objecte té el seu espai de variables propi. És a dir, si creem 2 objectes de la classe Punt, haurem creat espai per a guardar 4 variables: 2 propietats[1] de cada objecte.

Finalment, un objecte, com passa amb els arrays, **sempre es passa per referència a una funció**.

Treballar amb l'objecte.

Si volem guardar valors a cadascuna de les 2 propietats de Punts, ho hem de fer assignant el valor a cadascun dels camps. Exemple:

```
1.  p1=Punt()
2.  p1.x = 3.2
3.  p1.y = 8
```

Veiem que, per a especificar els camps de la variable "p1", fem servir el símbol de punt "." .

Finalment, si volem mostrar el valor de la variable "p1" per pantalla, no ho podem fer directament, hem de mostrar camp per camp:

```
print(p1.x,",",p1.y)
```

que mostraria, amb les dades anteriors:

```
3.2 , 8
```

Resum

Termes:

- Definir una **classe** és com definir un nou tipus de dada.
- **Propietat** és cadascuna de les variables internes d'una classe.
- **Objecte** és cadascuna de les variables de tipus de la classe.
- Un **objecte** sempre **es passa per referència** a una funció.

[1] Recordem que les variables d'una clase també s'anomenen propietats de la classe.

Exemple complet:

```
1.  class Punt:   #declarem la classe Punt
2.    x=0
3.    y=0
4.
5.  p1=Punt()   #Per a poder treballar amb Punt's, hem d'inicialitzar
6.              #la variable d'aquesta manera.
7.  p1.x=3
8.  p1.y=8
9.  print(p1.x,",",p1.y)  #no podem mostrar p1 directament, no dóna resultat
10.
```

Com veiem, hem creat una variable de tipus Punt (variable 'p1'). Com la variable 'p1' és de tipus Punt, hem creat un **objecte** de la classe Punt.

Cada objecte té el seu espai de variables propi. En següent exemple crearem 2 objectes de la classe Punt, amb valors diferents, i veurem que es guarden de manera independent:

```
1.  class Punt:   #declarem la classe Punt
2.    x=0
3.    y=0
4.
5.  p1=Punt()   #Creem 2 objectes de la classe Punt
6.  p2=Punt()
7.
8.  p1.x=3
9.  p1.y=8
10. p2.x=5
11. p2.y=-2
12.
13. print(p1.x,",",p1.y)  #mostra 3 , 8
14. print(p2.x,",",p2.y)  #mostra 5 , -2
```

Arrays d'objectes

Per a treballar amb arrays (llistes, tuples, diccionaris, etc.) d'objectes, ho hem de fer com fins ara, cap diferència: **variable.propietat** ('p1.x' abans i 'v[0].x ,v[1].x' ara, ja que les variables Punt són 'p1' i 'v[0],v[1]'). Aquí veiem l'exemple anterior en una llista de 2 elements, en lloc de 2 variables:

```
1.  class Punt:   #declarem la classe Punt
2.    x=0
3.    y=0
4.
5.  v=[None]*2
6.  v[0]=Punt()   #Creem 2 objectes de la classe Punt
7.  v[1]=Punt()
8.
9.  v[0].x=3
10. v[0].y=8
11. v[1].x=5
12. v[1].y=-2
13.
14. print(v[0].x,",",v[0].y)  #mostra 3 , 8
15. print(v[1].x,",",v[1].y)  #mostra 5 , -2
16.
17. #Exemples de 'for' amb llistes d'objectes:
18. for i in range(len(v)):  #Exemple de 'for' amb 'range'
19.   print(v[i].x,",",v[i].y)
20.
21. for n in v:             #Exemple de 'for' de cada element
22.   print(n.x,",",n.y)
```

4. Funcions de la classe

Dins d'una classe hi podem definir funcions. Aquestes funcions només es poden cridar a través dels objectes. Com amb les propietats, hi accedim a partir del símbol punt "." A partir d'un objecte creat.

Totes les funcions d'una classe tenen la mateixa sintaxi que les funcions que hem vist fins ara, amb la diferència que, obligatòriament, han de tenir un primer paràmetre "fantasma" que anomenem **self**. Aquest paràmetre és obligatori de posar, però quan es crida a la funció, no s'ha de posar (per això l'anomenem "fantasma").

Aquest paràmetre **self** es refereix a l'objecte on estem ara i serveix, principalment, per a diferenciar entre si quan fem servir una variable ens referim una propietat de de la classe o una variable local de la pròpia funció: si porta **self** és una **propietat** i si no en porta és una **variable local**.

Veiem un exemple de funció que mostra per pantalla el contingut dels valors x,y , i com la podríem fer servir:

```
1.  class Punt:   #declarem la classe Punt
2.    x=0
3.    y=0
4.    def mostra(self):
5.      print(self.x,",",self.y)
6.
7.  p1=Punt(3,8)   #Creem 2 objectes de la classe Punt
8.
9.  p1.mostra()  #mostra 3 , 8
10.
```

Aquesta funció "mostra" es defineix, només, amb el paràmetre obligatori "self" (per això des de la línia 9 la cridem sense passar cap paràmetre) i el que fa és mostrar per pantalla el contingut de les propietats x,y de l'objecte 'p1'. Si no haguéssim posat el "self." A la línia 5:

```
4.  def mostra(self):
5.      print(x,",",y)    #ERROR: x i y no tenen valor
```

ens hagués donat **error**, ja que voldrà mostrar per pantalla el contingut de les variables x,y **locals** de la funció mostra, i encara no les hem donat valor. Per tant, és important que ens fixem si ens volem referir a les variables de l'objecte (propietats) on es necessita posar "self." o a les variables locals de la funció on estem (on no es necessita).

Exercici proposat 1: Afegir una altra funció a la classe Punt que permeti, passant-li un punt, retornar la pendent de la recta entre els 2 punts (Δy / Δx).

Exercici proposat 2: Afegir una funció a la classe Punt que permeti, passant-li un altre punt, retornar el punt mig (valor mig de la x' i valor mig de la 'y') entre el x,y que conté i el punt que se li passa. Per exemple, el punt mig entre (0,10) i (4,8) és (2,9).

5. Cas especial: la funció constructora

La funció constructora és la funció inicialitzadora, la que es crida quan es crea un objecte. Quan fem:

p1=Punt()

estem cridant a la funció constructora de la classe Punt (també anomenada **constructor**). Si no tenim definit cap constructor (com en els exemples anteriors), Python ens proporciona un que no fa res (el que cridem sense res entre parèntesi). Però podem tenir un que inicialitzi les propietats de l'objecte com nosaltres necessitem, ja que tenim la possibilitat d'inicialitzar els valors x,y en aquest mateix moment.

El constructor és una funció dins la classe que porta el nom especial "__init__" (doble guió baix abans i després):

def __init__(self, altres paràmetres) :

instruccions de la funció

Aquesta funció s'executa quan es crea un objecte. El primer paràmetre, com en la secció anterior quan es parlava de funcions, és un paràmetre "fantasma" que s'anomena **self** i és obligatori posar-lo. "**self**" es refereix a l'objecte propi de la classe (és a dir, a ell mateix) i serveix per a referir-se a les variables de l'objecte des de dins de les funcions.

Quan cridem a la funció constructora no ho fem mai fent servir el nom **__init__,** sinó el nom de la classe -a dalt hem fet "p1=Punt()"- i el primer paràmetre "self" no es passa mai en la crida, ja que és un tema intern de Python.

Un exemple de la classe 'Punt' amb constructor definit:

```
6.  class Punt:   #declarem la classe Punt
7.    x=0
8.    y=0
9.    def __init__(self,x,y):
10.     self.x=x   #Si no fem self.x, no guarda el valor a la 'x' de la línia 2
11.     self.y=y  #'x' i 'y' són variables locals de la funció __init__ . També self
12.
13. p1=Punt(3,8)   #Creem 2 objectes de la classe Punt
14. p2=Punt(5,-2)
15.
16. print(p1.x,",",p1.y)  #mostra 3 , 8
17. print(p2.x,",",p2.y)  #mostra 5 , -2
18.
```

Gràcies a definir un constructor, podem passar els valors inicials de 'x' i de 'y' quan creem l'objecte, i així ens estalviem anar posant els valors després. Ara ja no tenim accés al constructor sense paràmetres Punt(), ja que hem definit un. Per tant, ja no necessitem les inicialitzacions a 0 de les línies 2 i 3.

Si volguéssim poder també crear un objecte a partir d'un constructor sense paràmetres, també podríem tenir un altre constructor sense paràmetres x,y o bé definir la funció constructora amb valors per defecte als paràmetres, per a poder inicialitzar a 0 les dues propietats x,y cridant al constructor Punt() sense paràmetres, com abans:

```
1.  class Punt:   #declarem la classe Punt
2.    def __init__(self,x=0,y=0):
3.      self.x=x   #Si no fem self.x, no guarda el valor a la 'x' de la línia 2
4.      self.y=y  #'x' i 'y' són variables locals de la funció __init__ . També self
5.
6.  p1=Punt(3,8)   #Creem 2 objectes de la classe Punt
7.  p2=Punt(5,-2)
8.  p3=Punt()      #p3 té x=0 i y=0, ja que ho tenim així definit al constructor
9.
10. print(p1.x,",",p1.y)  #mostra 3 , 8
11. print(p2.x,",",p2.y)  #mostra 5 , -2
12. print(p3.x,",",p3.y)  #mostra 0 , 0
13.
```

6. He vist abans tot això?

La resposta és que sí. Fins ara, tant en Python (en aquesta assignatura i en altres) com en R, has treballat amb objectes de manera natural sense saber que eren objectes.

Per exemple, en Python quan has treballat amb fitxers o strings, ja has cridat a funcions d'objectes.

Per exemple, en l'exercici 10 de pràctiques de la Sessió 6:

```
1.  f=open("ex10.txt","r")
2.  s=f.read()  #crida a la funció read() de l'objecte "f"
3.  f.close()   #crida a la funció close() de l'objecte "f"
4.  v=s.split(",")    #crida a la funció split() de l'objecte "s" String
5.  suma=0
6.  for x in v:
7.    suma+=float(x)
8.  print("La suma és:",suma)
9.
10.
```

Veiem que:

- A la línia 2 cridem a la funció "read()" de l'objecte "f", ja que la funció "open(...)" crea un objecte d'una classe que tracta els fitxers (no sabem el seu nom) i el retorna.
- A la línia 3 cridem la funció "close()" de l'objecte "f"-
- A la línia 4 cridem la funció "split(...)" de l'objecte "s", que és un String.

Com veiem, només als objectes se'ls pot cridar a una funció fent servir la sintaxi **variable.funció(...)**. Per exemple, a la línia 5 suma és un enter, i no existeix cap funció que puguem fer "suma.funció(...)" ja que un enter no és un objecte (no ve d'una classe). Totes les funcions que treballen amb enters es criden de manera habitual (sense fer servir el punt ".").

Cas partircular dels Strings

Els Strings són, en tots els llenguatges de programació, un cas especial de tipus de dada. En Python, són objectes, però amb algunes diferències respecte lo que s'ha vist fins ara:

- Els Strings no es creen cridant al constructor de manera explícita. Només cal fer s="Hola" i Python ja crida al constructor passant el paràmetre "Hola".
- Els Strings no es passen per referència a una funció. Els objectes, sí.
- Els Strings tenen alguna altra diferencia més respecte als objectes, com algun operador, que ja veurem a mesura que avanci el curs i surtin exemples. Fins ara hem vist l'operador "+", que serveix per a concatenar Strings, però que no té cap efecte en objectes.

En R has vist que el símbol del dolar "$" es fa servir per a accedir a les propietats d'un objecte (anomenat dataframe), per exemple. Normalment, després de llegir un fitxer csv, cadascuna de les propietats del dataframe conté una llista amb la columna llegida del fitxer i hi accedeixes via el símbol $ (dades$temperatura, dades$hora , etc.).

Exercici proposat 3: Proposa una definició completa de la classe Punt i fes un programa exemple o fer-la servir.

7. Solucions als exercicis proposats

Exercici proposat 1: Afegir una altra funció a la classe Punt que permeti, passant-li un punt, retornar la pendent de la recta entre els 2 punts (Δy / Δx).

```
1.  class Punt:
2.    x=0
3.    y=0
4.    def mostra(self):
5.      print(self.x,",",self.y)
6.
7.    def pendent(self,punt):
8.      m=(punt.y-self.y)/(punt-x-self.x)
9.      return m
10.
11.   def pendent2(self,x,y):
12.     m=(punt.y-self.y)/(punt-x-self.x)
13.     return m
14.
15.
```

En aquest exercici s'ha plantejat 2 solucions per a calcular la pendent: que es rebi un objecte Punt o que es rebin 2 valors x,y (com no podem tenir 2 funcions amb el mateix nom, hem de posar noms diferents: pendent i pendent2). En aquest cas, hem implementat 2 funcions que fan aquest mateix càlcul. És molt habitual que el dissenyador d'una classe ofereixi la possibilitat de fer un mateix càlcul amb diversos paràmetres d'entrada, oferint així a l'usuari de la classe una major facilitat en els càlculs a fer.

Una altra millora a aplicar és l'evitar fer duplicació de codi, ja que si es troba un error en una, ens hauríem d'enrecordar on està aquest mateix càlcul a la resta del. Les línies 7 i 10 tenen exactament el mateix càlcul. Una millora habitual és que una cridi a l'altra adaptant ela paràmetres:

```
1.  class Punt:   #declarem la classe Punt
2.    x=0
3.    y=0
4.    def mostra(self):
5.      print(self.x,",",self.y)
6.
7.    def pendent(self,punt):
8.      return self.pendent2(punt.x,punt.y) #self és necessari
9.
10.   def pendent2(self,x,y):
11.     m=(y-self.y)/(x-self.x)
12.     return m
13.
```

Important: A la línia 8 s'observa que, per a **cridar a una funció** que està a la **mateixa classe**, necessitem posar el "**self**." al davant de la crida.

Exercici proposat 2: Afegir una funció a la classe Punt que permeti, passant-li un altre punt, retornar el punt mig (valor mig de la x' i valor mig de la 'y') entre el x,y que conté i el punt que se li passa. Per exemple, el punt mig entre (0,10) i (4,8) és (2,9).

```
1.  class Punt:
2.    x=0
3.    y=0
4.    def mostra(self):
5.      print(self.x,",",self.y)
6.    def puntMig(self,punt):
7.      p = Punt()      #Creem un punt a retornar
8.      p.x=(punt.x+self.x)/2
9.      p.y=(punt.y+self.y)/2
10.     return p  #retornem un objecte de la classe Punt
```

En aquesta solució s'ha decidit no fer una segona funció on rebi el x,y , però es podria a ver fer perfectament.

A la línia 7, creem un nou Punt que és el que farem servir per a retornar. En cap cas modificarem el Punt que rebem per paràmetre! Ja que es passa per referència.

Afegir que és IMPORTANT no modificar els paràmetres que se'ns passa a una funció a no ser que part de la feina de la funció consisteixi en modificar aquests paràmetres (com quan passem una llista buida i volem que una funció ens l'ompli de nombres).

Exercici proposat 3: Proposa una definició completa de la classe Punt i fes un programa exemple o fer-la servir.

Explicació:

- Definim un **constructor** que, si no rep algun paràmetre, aplica un 0.
- **Funció mostra**: Durant el programa preferirem mostrar el punt en una mateixa línia o en línies diferents, així que s'han fet 2 funcions: una que retorna el String que conté lo que es vol mostrar (**mostraString**)i l'altra que el mostra amb un print (**mostra**).
  Quan mostrem el punt per pantalla, hem preferit no col·locar espais entre els valors, i per això hem descartat les comes per a separar els elements a mostrar. En lloc d'això, s'ha preferit aprofitar que "mostraString" ja retorna el String ben formatat i mostrar-lo per pantalla, així no tenim codi duplicat.
- La línia 36 mostra un exemple de com podríem concatenar un string llarg per a mostrar un missatge ben formatat, sense dependre de si les comes ens afegeixen espais, etc.
- Les línies 32 i 35 podrien haver-se escrit intercanviant p1 per p2:

```
pmig=p2.puntMig(p1)
m=p2.pendent(p1)
```

```
1.  class Punt:
2.    def __init__(self,x=0,y=0):
3.      self.x=x
4.      self.y=y
5.
6.    def mostraString(self):
7.      return "("+str(self.x)+","+str(self.y)+")"  #Retornarà “(11,0)”, per exemple
8.
9.    def mostra(self):
10.     print(self.mostraString())  #Mostra els valors sense espais intercalats
11.
12.   def pendent(self,punt):
13.     return self.pendent2(punt.x,punt.y)
14.
15.   def pendent2(self,x,y):
16.     m=(y-self.y)/(x-self.x)
17.     return m
18.
19.   def puntMig(self,punt):
20.     p = Punt()
21.     p.x=(punt.x+self.x)/2
22.     p.y=(punt.y+self.y)/2
23.     return p
24.
25. #PROGRAMA PRINCIPAL
26. p1=Punt(5,3)
27. p2=Punt(11)   #p2 tindrà els valors 11,0
28. print("Tenim els punts:")
29. p1.mostra()
30. p2.mostra()
31.
32. pmig=p1.puntMig(p2)  #Calculem el punt mig entre p1 i p2 i el mostrem
33. print("El punt mig és:"+pmig.mostraString())
34.
35. m=p1.pendent(p2) #Calculem la pendent de la recta entre p1 i p2 i la mostrem
36. print("La pendent de la recta que passa per "+p1.mostraString()+" i
    "+p2.mostraString()+" és:"+str(m))
37. class Punt:
38.   x=0
```