Introducció a Python: Sessió4:Programació Orientada a Objectes

1. Introducció

Python té característiques que donen suport a diferents paradigmes de programació: estructurada, orientada a objectes i funcional.
En aquest document es veurà com utilitzar en Python les estructures pròpies de la Programació Orientada a Objectes.

La sintaxi per definir una classe és:

```
class nom_de_la_classe:
        definicions d'atributs i mètodes
```

La manera d'instanciar objectes, cridar mètodes etc es descriu en les següents activitats de manera progressiva.

2. Activitats de la Sessió 4.

1. Definició i utilització d'una classe (classe Producto):

Crear un fitxer d'script amb el següent contingut d'exemple:

```
1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3
4  class Producto:
5      """ Ejemplo de clase con la cantidad y el precio de un producto"""
6
7      def __init__(self,producto,precio,unidades):
8          self.producto = producto
9          self.precio = precio
10         self.unidades = unidades
11
12     def costo_total(self):
13         costo = self.precio * self.unidades
14         return costo
15
16
17 mi_objeto_producto = Producto("corbata",35,67)
18
19 print mi_objeto_producto.producto
20 print mi_objeto_producto.precio
21 print mi_objeto_producto.unidades
22 print mi_objeto_producto.costo_total()
```

Comentaris:

Aquest codi defineix la classe **Producto**. A dins hi ha dos mètodes: **__init__** i **costo_total**.
La sintaxi dels mètodes és idèntica a la de les funcions, excepte que sempre tenen com a mínim un primer paràmetre, anomenat **self**. Aquest paràmetre **NO s'ha d'indicar en les crides** al mètode. Representa el propi objecte al qual partany el mètode. L'quivalent en Java de **self** és **this**.

El mètode **__init__** és el mètode constructor de la classe. S'executa sempre en instanciar un objecte. Per convenció, el nom comença i acaba amb dos caràcters underscore.

En el mètode **__init__** s'inicialitzen tres atributs, que van precedits de **self**.
Els atributs no s'han definit abans, però es podria fer. En el moment en que es fa referència a un atribut (self.nom_atribut), s'afegeix a la definició de la classe.

A la línia 17 es fa una **instanciació** d'un objecte, passant els paràmetres que requereix el constructor (excepte self).

A les línies 19 a 22 s'accedeix als atributs i mètodes de l'objecte, amb la sintaxi habitual objecte.atribut i objecte.mètode.

Igual que passa amb les funcions, les classes admeten documentació proporcionada per l'usuari per mitjà de **docstrings**. El text de la docstring pasa a ser el valor de l'atribut __doc__ de la classe. Veure recomanacions d'estil a **https://www.python.org/dev/peps/pep-0257/**

Activitats:

Executar el programa. Explicar els resultats obtinguts.
Modificar el programa definint al començament de la classe els atributs amb el valor inicial adient. Comprovar si funciona igual que la versió original.

Explicar per a què serveixen els docstrings. (Buscar a la web)
Explicar que és una PEP: Python Enhancement Proposal. (Buscar a la web)

2. Definició i utilització d'una classe. Mètodes privats:

Crear un fitxer d'script amb el següent contingut d'exemple:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

class Producto:
    """ Ejemplo de clase con la cantidad y el precio de un producto"""

    def __init__(self,producto,precio,unidades):
        self.producto = producto
        self.precio = precio
        self.unidades = unidades

    def __costo_total(self):
        costo = self.precio * self.unidades
        return costo

    def nuevo_precio(self,precio):
        self.precio = precio

    def agrega(self,cantidad):
        self.unidades = self.unidades + cantidad

    def saca(self,cantidad):
        if cantidad <= self.unidades:
            self.unidades = self.unidades - cantidad
        else:
            print "No hay suficientes"

    def informe(self):
        print "Producto: " + self.producto
        print "Precio: " + str(self.precio)
        print "Unidades: " + str(self.unidades)
        print "Precio Total: " + str(self.__costo_total())

mi_producto1 = Producto("Pantalón",100,6)

mi_producto2 = Producto("Camiseta",50,5)

print mi_producto1.precio
print mi_producto2.unidades

mi_producto2.agrega(5)
print mi_producto2.unidades

mi_producto2.informe()

```

Comentaris:

S'han afegit més mètodes a la classe. Un dels mètodes és **__costo_total** que, pel fet de començar amb dos caràcters underscore, té la particularitat de ser **privat**: només es pot cridat des de dintre de l'objecte.

Activitats:
Executar el programa. Explicar els resultats obtinguts.
Des del programa principal, cridar directament al mètode __costo_total i comprovar que es produeix un error.

3. Herència:

Crear un fitxer d'script amb el següent contingut d'exemple:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

class Animal:
    """Clase base para mostrar la herencia"""

    def __init__(self, nombre, patas):
        self.nombre = nombre
        self.patas = patas

    def saluda(self):
        print "El animal llamado " + str(self.nombre) + " saluda"

class Perro(Animal):
    """Clase hija para mostrar la herencia"""
    # Simplemente, no hacemos nada
    pass

mi_mascota = Perro("Rufo",4)

mi_mascota.saluda()
```

Comentaris:

Es crea una classe Animal i una subclasse Perro, que hereta d'Animal. La relació d'herència s'indica posant al costat del nom de la classe filla el nom de la classe pare entre parèntesi.

La sentència **pass** s'utilitza per indicar un bloc de codi buit.

Activitats:

Executar el programa. Explicar els resultats obtinguts.

4. Herència:

Crear un fitxer d'script amb el següent contingut d'exemple:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

class Animal:
    """Clase base para mostrar la herencia"""

    def __init__(self, nombre, patas):
        self.nombre = nombre
        self.patas = patas

    def saluda(self):
        print "El animal llamado " + str(self.nombre) + " saluda"


class Perro(Animal):
    """Clase hija para mostrar la herencia"""

    def ladra(self):
        print "Guau"


class Gato(Animal):
        """Clase hija para mostrar la herencia"""

    def maulla(self):
        print "Miau miau"

mi_mascota = Perro("Rufo",4)
mi_mascota.saluda()
mi_mascota.ladra()

mi_otra_mascota = Gato("Azrael",4)
mi_otra_mascota.saluda()
mi_otra_mascota.maulla()

```

Comentaris:

Es creen dues subclasses amb mètodes heretats i propis de cadascuna.

Activitats:

Executar el programa. Explicar els resultats obtinguts.

5. Herència amb sobreesciptura de mètodes:

Crear un fitxer d'script amb el següent contingut d'exemple:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

class Animal:
    """Clase base para mostrar la herencia"""

    def __init__(self, nombre, patas):
        self.nombre = nombre
        self.patas = patas

    def saluda(self):
        print "El animal llamado " + str(self.nombre) + " saluda"


class Perro(Animal):
    """Clase hija para mostrar la herencia"""

    def ladra(self):
        print "Guau"


class Gato(Animal):
    """Clase hija para mostrar la herencia"""

    def maulla(self):
        print "Miau miau"

    def saluda(self):
        print "El gato " + str(self.nombre) + " te mira fijamente"


mi_mascota = Perro("Rufo",4)
mi_mascota.saluda()
mi_mascota.ladra()

mi_otra_mascota = Gato("Azrael",4)
mi_otra_mascota.saluda()
mi_otra_mascota.maulla()
```

Comentaris:
S'utilitza sobreescriptura de mètodes.

Activitats:
Executar el programa. Explicar els resultats obtinguts.

6. Herència múltiple:

Crear un fitxer d'script amb el següent contingut d'exemple:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

class Animal:
    """Clase base para mostrar la herencia"""

    def __init__(self, nombre, patas):
        self.nombre = nombre
        self.patas = patas

    def saluda(self):
        print "El animal llamado " + str(self.nombre) + " saluda"


class Amigo:
    """Clase base para mostrar la herencia"""

    def __init__(self, nombre):
        self.nombre = nombre

    def salir(self, num):
        if num == 0:
            print "Vamos a pasear"
        elif num == 1:
            print "Vamos a jugar"
        else:
            print "Vamos al parque"

class Perro(Animal,Amigo):
    """Clase hija para mostrar la herencia"""

    def ladra(self):
        print "Guau"

mi_mascota = Perro("Rufo",4)
mi_mascota.saluda()
mi_mascota.salir(1)
```

Comentaris:

En Python s'admet l'herència múltiple (heredar de més d'una classe). Per indicar-ho s'escriuen dins del paréntesi totes les superclasses, separades per coma. En cas d'heredar atributs o mètodes iguals de diferents classes, Python agafa els de la classe que s'ha indicat primer.

Activitats:

Executar el programa. Explicar els resultats obtinguts.

7. Cridar mètodes (sobreescrits) de la superclasse:

Crear un fitxer d'script amb el següent contingut d'exemple:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

class Animal:
    """Clase base para mostrar la herencia"""

    def __init__(self, nombre, patas):
        self.nombre = nombre
        self.patas = patas

    def saluda(self):
        print "El animal llamado " + str(self.nombre) + " saluda"


class Perro(Animal):
    """Clase hija para mostrar la herencia"""

    def __init__(self, nombre):
        Animal.__init__(self, nombre, 4)
        self.sonido = "Guau"
    def ladra(self):
        print self.sonido


mi_mascota = Perro("Chucho")
mi_mascota.saluda()
mi_mascota.ladra()

```

Comentaris:

La classe filla defineix el seu propi mètode __init__. Si, com és el cas, interessa cridar el mètode del pare, es pot fer prefixant-lo amb el nom de la classe pare.

Activitats:

Executar el programa. Explicar els resultats obtinguts.

8. Altres mètodes predefinits de les classes.

Crear un fitxer d'script amb el següent contingut

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

class Palabra:
    """Clase para mostrar el método __cmp__"""

    def __init__(self, contenido):
        self.contenido = contenido

    def __cmp__(self, otro):
        if self.contenido > otro.contenido:
            return 1
        elif self.contenido < otro.contenido:
            return -1
        else:
            return 0


larga = Palabra("supercalifragilisticoespialidoso")
corta = Palabra ("bah")

if (larga > corta):
    print larga.contenido + " es mayor que " + corta.contenido
else:
    print larga.contenido + " NO es mayor que " + corta.contenido
```

Comentaris:
Es reescriu el mètode **__cmp__**, que serveix per comparar objectes d'aquesta classe.

A part de **__init__** hi ha molts altres mètodes predefinits com:

- **__del__(self)** s'executa en el moment d'esborrar l'objecte de memòria. Com que l'esborrat el fa el garbage collector, no es pot assegurar si s'executarà ni quan.
- **__str__(self)** s'executa quan es transforma un objecte en string, per mitjà de la funció **str()** o quan es fa **print**.
- **__len__(self)** s'executa quan es demana la longitud de l'objecte per mitjà de la funció **len()**.
- **__cmp__(self, altre_objecte)** s'executa quan es compara l'objecte amb un altre object, per mitjà dels operadors de comparació. Si es programa aquest mètode, ha de retornar **0** si els objectes són **iguals**, un **nombre negatiu** si l'objecte **self** és **més petit** que **altre_objecte** i un nombre **positiu** si és **més gran**.

El comportament, **per a una classe determinada,** de tots el operadors relacionals i aritmètics es pot personalitzar reescribint els mètodes predefinits **__eq__(self, altre)** per l'operador ==, **__ne__(self, altre)** per !=, etc i

__add__(self, altre) per l'operador +, **__mul__(self, altre)** per l'operador *, etc.

Activitats:
Executar el programa. Explicar els resultats obtinguts.
Cambiar la implementació de __cmp__ i comprovar l'efecte del canvi.