Ayudantía 01

Polimorfismo, Decorators, Properties

Javiera Astudillo Vicente Domínguez

Departmento de Ciencias de la Computación
Pontificia Universidad Católica de Chile

IIC2233, 2015-1

Tabla de contenidos

Polimorfismo

¿Qué es?

- Se llama Polimorfismo a la posibilidad de obtener distintos comportamientos mediante la invocación a métodos de un mismo nombre, pero de clases distintas.
- En Python basta con redefinir la función de una clase padre en una clase hija utilizando el mismo nombre de la función
- Al hacer lo anterior se sobrescribe completamente el código de la función heredada, si queremos extender el código de la función de la clase padre debemos usar *super().NombreMetodo()*

Sintaxis

Sobrescritura

La clase padre:

```
1 class Objeto:
2
3     def __init__(self, peso, volumen, **kwargs):
4         self.peso = peso
5         self.volumen = volumen
6
7     def descripcion(self):
8         print("{} {}".format(self.peso, self.volumen))
```

La clase hijo:

```
1 class Calculadora(Objeto):
2
3     def __init__(self, marca, modelo, **kwargs):
4         super().__init__(**kwargs)
5         self.marca = marca
6         self.modelo = modelo
7
8     def descripcion(self):
9         print("{} {}".format(self.marca, self.modelo))
```

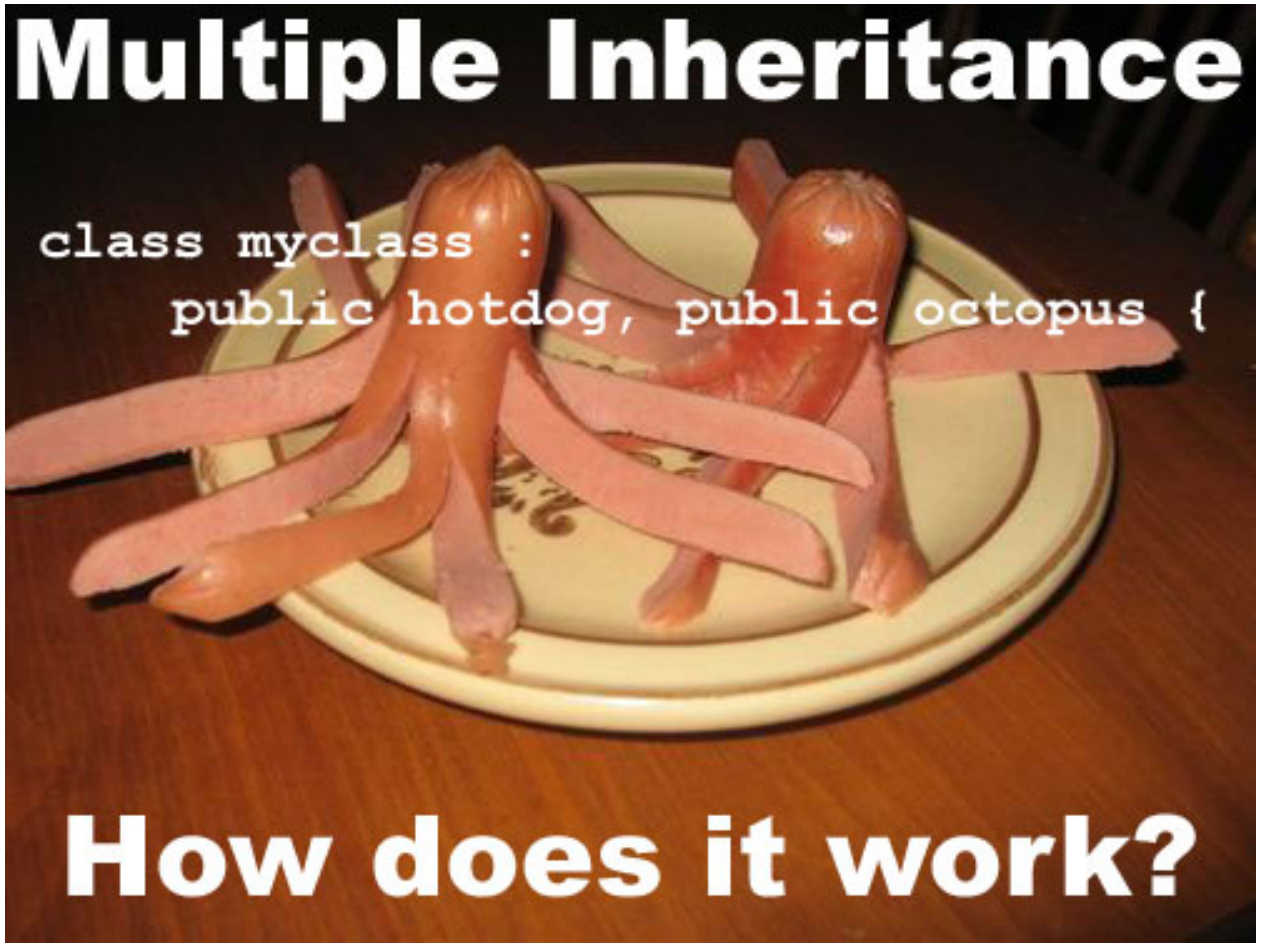
Sintaxis

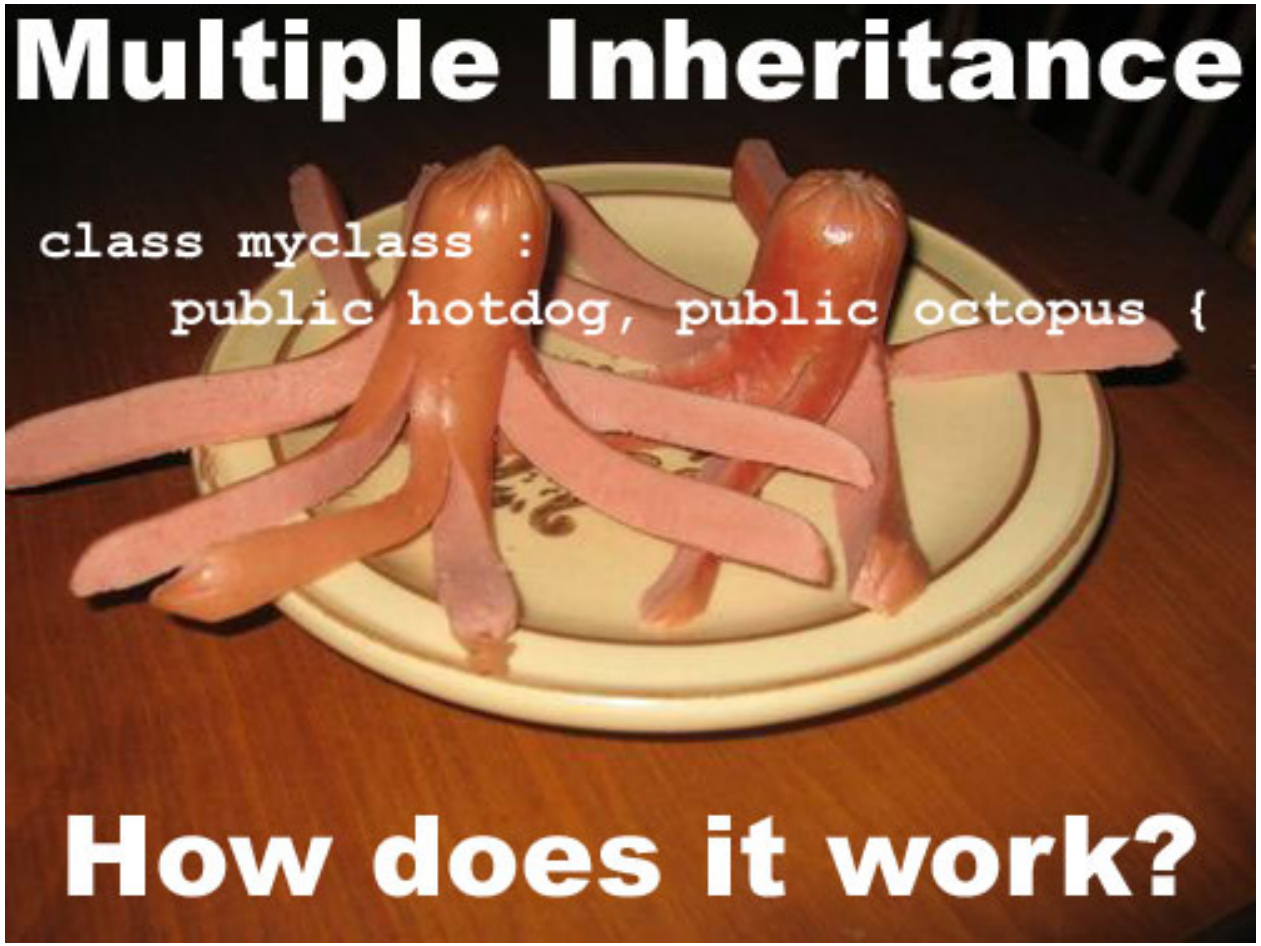
Extensión de código

```
1  class Objeto:
2
3      def __init__(self, peso, volumen, **kwargs):
4          self.peso = peso
5          self.volumen = volumen
6
7      def descripcion(self):
8          print("{} {}".format(self.peso, self.volumen))
```

La clase hijo:

```
1  class Calculadora(Objeto):
2
3      def __init__(self, marca, modelo, **kwargs):
4          super().__init__(**kwargs)
5          self.marca = marca
6          self.modelo = modelo
7
8      def descripcion(self):
9          super().descripcion()
10         print("{} {}".format(self.marca, self.modelo))
```


Multiple Inheritance
class myclass :
public hotdog, public octopus {
How does it work?



Que ocurre con el Polimorfismo aplicado a la Multiherencia

Multiherencia

- Al haber Multiherencia en Python ocurre el problema de no saber de que "padre"proviene la función a la que le estamos haciendo polimorfismo
- Lo que hace Python es revisar si el método tiene en su código *super().NombreMetodo()*, si no lo tiene simplemente sobrescribe el método de los padres por ese y lo ejecuta al ser llamado.
- Si tiene *super().NombreMetodo()*, Python empieza a llamar a las clases padres en un orden dado por *Clase__mro__*

Properties

¿Qué son?

- Son atributos a los que se les asigna los métodos *getter, setter y deleter* de manera manual (en código).
- Entre sus utilidades, permite determinar si un atributo será *read-only*, las restricciones para modificarlo, si puede ser eliminado, etc.
- Para crear una property, se hace por medio de la función (built-in) *property(fget, fset, fdel,doc)*, donde fget, fset y fdel son las funciones getter, setter y deleter correspondientemente.

Sintaxis

Sin decorators

```
1  class Objeto():
2      def __init__(self):
3          self._x = 100
4
5      def getx(self):
6          return self._x
7
8      def setx(self, value):
9          self._x = value
10
11     def delx(self):
12         del self._x
13
14     x = property(getx, setx, delx, "Soy la property de x.")
```

Sintaxis

Con decorators

```
1  class Objeto():
2      def __init__(self):
3          self._x = 100
4
5      @property
6      def x(self):
7          """Soy la property de x."""
8          return self._x
9
10     @x.setter
11     def x(self, value):
12         self._x = value
13
14     @x.deleter
15     def x(self):
16         del self._x
```