python™

Programando con Python

Clases y Objetos (Herencia y Polimorfismo)

POO introducción

- **Herencia:**
- Python soporta herencia y herencia múltiple
- Sintáxis: class SubclassName(SuperclassName):
  pass
- Funciones type() y isinstance(), la primera nos dice el tipo, la segunda si el objeto es o no derivado de una clase en particular

  (ver ejemplos)

Clases y Objetos: Herencia y Polimorfismo

- Herencia:
  - En Python se usa la función **super()**, para llamar algún método de la super clase
  - Ejemplo:

```
class Rectangle(Shape):

    def __init__(self, length, width):
        super().__init__()
        self.__length = length
        self.__width = width
```

Ver ejemplo inheritance.py / herencia_simple.py

Clases y Objetos: Herencia y Polimorfismo

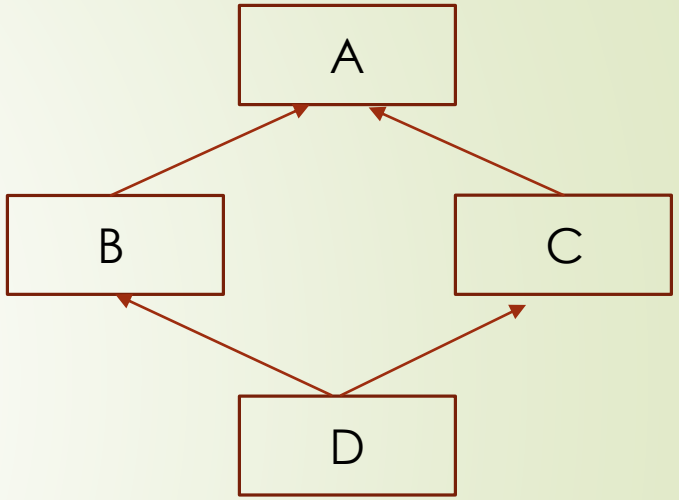
- Herencia múltiple:
  - En Python es posible derivar una clase desde otras clases, esto se llama herencia múltiple.
  - Sintaxis:

```
class ParentClass_1:
    # body de ParentClass_1

class ParentClass_2:
    # body de ParentClass_2

class ParentClass_3:
    # body de ParentClass_1

class ChildClass(ParentClass_1, ParentClass_2, ParentClass_3):
    # body de ChildClass
```

Ver ejemplo: herencia_multiple.py / herencia_multiple2.py

POO introducción

- **Herencia múltiple:**
- Sintaxis: class SubclassName(Superclass1, SuperClass2, …, SuperClassN):
  pass
- Para evitar el problema del "mortal diamante de la muerte", Python usa el llamado MRO (method resolution order), el cual le permite resolver el problema. Ver ejemplo.



POO introducción

- **Herencia – sobreescritura (overriding):**
- Cuando un método de la superclase se implementa de diferente forma en la subclase
- Es posible llamar a métodos de la superclase
- La subclase puede sus propios métodos (especialización)

  (ver ejemplos)

POO introducción

- **Diferencia entre overwriting, overloading y overriding:**
- Overwriting, cuando una función se declara nuevamente con diferentes parámetros
- Overloading, no existe en Python, pero se puede simular con parámetros predefinidos. O mejor aún, usando la forma def(*x):, lo cual indica que la función f puede recibir un número indeterminado de parámetros.
- Overriding, cuando un método de la subclase tiene el mismo nombre de otro en la superclase.

Clases y Objetos: Herencia y Polimorfismo

- Polimorfismo:
  - En Python el polimorfismo es definido de manera que un método de la clase hija tenga el mismo nombre que el de la clase padre.
  - Es en esencia una sobre-escritura
  - Ver ejemplos:
    - method_overriding.py
    - method_overriding_2.py

Clases y Objetos: Herencia y Polimorfismo

- object – la clase base:
  - En Python todas las clases heredan de la clase **object** de forma implícita.
  - Eso quiere decir que esta dos expresiones son equivalentes:

```
class MyClass:
    pass

class MyClass(object):
    pass
```

Clases y Objetos: Herencia y Polimorfismo

- object – la clase base:
  - La clase object posee métodos que son heredados por todas las clases. Algunos importantes son:
    1. __new__() -> crea el objeto
    2. __init__() -> después __new__(), se llama para inicializar los atributos del objeto
    3. __str__() -> retorn a una representación en string del objeto (ver ejemplo __str__method.py), por lo general se sobreescribe según la necesidad

Clases y Objetos: Herencia y Polimorfismo

- Sobreescribiendo la funcionalidad de los operadores:
  - Python permite redefinir la forma como los operadores incluidos definen sus operaciones
  - De allí que el operador **+** sirva tanto para sumar números como para concatenar dos o más strings
  - Los métodos especiales que definen los operadores comienzan y terminan con doble guion bajo, así el de **+** es __add__()
  - Las clases int y la clase str, lo definen cada uno de acuerdo a lo que necesiten hacer y de allí que se pueda usar el mismo signo **+** de dos maneras diferentes.
  - Aunque empiezan con doble guion bajo no son privados, porque también terminan con doble guion bajo

Clases y Objetos: Herencia y Polimorfismo

- Operador y su método especial:

Operador	Método Especial	Descripción
+	__add__(self, object)	Suma
-	__sub__(self, object)	Resta
*	__mul__(self, object)	Multiplicación
**	__pow__(self, object)	Exponenciación
/	__truediv__(self, object)	División
//	__floordiv__(self, object)	División Entera
%	__mod__(self, object)	Modulo
==	__eq__(self, object)	Igual a

Clases y Objetos: Herencia y Polimorfismo

- Operador y su método especial:

Operador	Método Especial	Descripción
!=	__ne__(self, object)	Diferente de
>	__gt__(self, object)	Mayor que
>=	__ge__(self, object)	Mayor o igual que
<	__lt__(self, object)	Menor que
<=	__le__(self, object)	Menor o igual que
in	__contains__(self, value)	Operador de membresía
[index]	__getitem__(self, index)	Elemento en indice
len()	__len__(self)	Calcula número de elementos
str()	__str__(self)	Convierte objeto a string

Ver ejemplo: special_methods.py

Clases y Objetos: Herencia y Polimorfismo

- Sobreescribiendo la funcionalidad de los operadores:
  - Ejemplos:
    - special_methods.py
    - point.py

Clases y Objetos: Herencia y Polimorfismo

- Clases abstractas:
  - Las clases abstractas son clases que contienen uno o más métodos abstractos.
  - Un método abstracto es un método que se declara, pero que no contiene ninguna implementación.
  - Las clases abstractas no pueden ser instanciadas, y requieren subclases para proporcionar implementaciones para los métodos abstractos.
  - Aunque los métodos abstractos tengan implementación en la superclase, deben ser implementados en la subclase
  - Una clase que se deriva de una clase abstracta no puede ser instanciada a menos que todos sus métodos abstractos sean sobreescritos.
  - Un método abstracto puede tener una implementación en la clase abstracta! Incluso si se implementan, los diseñadores de subclases se verán obligados a sobreescribir la implementación.

Clases y Objetos: Herencia y Polimorfismo

- Clases abstractas:
  - Python viene con un módulo que proporciona la infraestructura para definir las Clases Base Abstractas (ABCs). Este módulo se llama - por razones obvias - abc.
  - El siguiente código Python utiliza el módulo abc y define una clase base abstracta:

```
from abc import ABC, abstractmethod
class AbstractClassExample(ABC):
    def __init__(self, value):
        self.value = value
        super().__init__()
    @abstractmethod
    def do_something(self):
        pass
```

Clases y Objetos: Herencia y Polimorfismo

- Clases abstractas:
  - Ejemplos:
    - Ver abstract_class_example_1.py
    - abstract_class_example_2.py
    - abstract_class_example_3.py

Clases y Objetos: Herencia y Polimorfismo

FIN