1- Plantear una clase Persona que contenga dos atributos: nombre y edad. Definir como responsabilidades la carga por teclado y su impresión.
En el bloque principal del programa definir un objeto de la clase persona y llamar a sus métodos.
Declarar una segunda clase llamada Empleado que herede de la clase Persona y agregue un atributo sueldo y muestre si debe pagar impuestos (sueldo superior a 3000)
También en el bloque principal del programa crear un objeto de la clase Empleado.

2- Supongamos que necesitamos implementar dos clases que llamaremos Suma y Resta. Cada clase tiene como atributo valor1, valor2 y resultado. Los métodos a definir son cargar1 (que inicializa el atributo valor1), carga2 (que inicializa el atributo valor2), operar (que en el caso de la clase "Suma" suma los dos atributos y en el caso de la clase "Resta" hace la diferencia entre valor1 y valor2), y otro método mostrar_resultado.
Si analizamos ambas clases encontramos que muchos atributos y métodos son idénticos. En estos casos es bueno definir una clase padre que agrupe dichos atributos y responsabilidades comunes.
La relación de herencia que podemos disponer para este problema es:

Operacion

Suma Resta

Solamente el método operar es distinto para las clases Suma y Resta (esto hace que no lo podamos disponer en la clase Operacion en principio), luego los métodos cargar1, cargar2 y mostrar_resultado son idénticos a las dos clases, esto hace que podamos disponerlos en la clase Operacion. Lo mismo los atributos valor1, valor2 y resultado se definirán en la clase padre Operacion.

3- Declarar una clase Cuenta y dos subclases CajaAhorro y PlazoFijo. Definir los atributos y métodos comunes entre una caja de ahorro y un plazo fijo y agruparlos en la clase Cuenta.
Una caja de ahorro y un plazo fijo tienen un nombre de titular y un monto. Un plazo fijo añade un plazo de imposición en días y una tasa de interés. Hacer que la caja de ahorro no genera intereses.
En el bloque principal del programa definir un objeto de la clase CajaAhorro y otro de la clase PlazoFijo.

4- Tenemos una clase "Largometraje" y ahora pensemos en la clase "Cine". En un cine se reproducen largometrajes. Puedes, no obstante, tener varios tipos de largometrajes, como películas o documentales, etc. Quizás las películas y documentales tienen diferentes características, distintos horarios de audiencia, distintos precios para los espectadores y por ello has decidido que tu clase "Largometraje" tenga clases hijas o derivadas como "Película" y "Documental".
Imagina que en tu clase "Cine" creas un método que se llama "reproducir()". Este método podrá recibir como parámetro aquello que quieres emitir en una sala de cine y podrán llegarte a veces objetos de la clase "Película" y otras veces objetos de la clase "Documental". Si has entendido el sistema de tipos, y sin entrar todavía en polimorfismo, debido a que los métodos declaran los tipos de los parámetros que recibes, tendrás que hacer algo como esto:
reproducir(Pelicula peliculaParaReproducir)

Pero si luego tienes que reproducir documentales, tendrás que declarar:

reproducir(Documental documentaParaReproducir)

Probablemente el código de ambos métodos sea exactamente el mismo. Poner la película en el proyector, darle al *play*, crear un registro con el número de entradas vendidas, parar la cinta cuando llega al final, etc. ¿Realmente es necesario hacer dos métodos? De acuerdo, igual no te supone tanto problema, ¿pero si mañana te mandan otro tipo de cinta a reproducir, como la grabación de la final del mundial de fútbol en 3D? ¿Tendrás que crear un nuevo método reproducir() sobre la clase "Cine" que te acepte ese tipo de emisión? ¿es posible ahorrarnos todo ese mantenimiento?
Aquí es donde el polimorfismo nos ayuda. Podrías crear perfectamente un método "reproducir()" que recibe un largometraje y donde podrás recibir todo tipo de elementos, películas, documentales y cualquier otra cosa similar que sea creada en el futuro.
Entonces lo que te permiten hacer los lenguajes es declarar el método "reproducir()" indicando que el parámetro que vas a recibir es un objeto de la clase padre "Largometraje", pero donde realmente el lenguaje y compilador te aceptan cualquier objeto de la clase hija o derivada, "Película", "Documental", etc.
reproducir(Largometraje elementoParaReproducir)
Podremos crear películas y reproducirlas, también crear documentales para luego reproducir y lo bonito de la historia es que todos estos objetos son aceptados por el método "reproducir()", gracias a la relajación del sistema de tipos. Incluso, si mañana quieres reproducir otro tipo de cinta, no tendrás que tocar la clase "Cine" y el método "reproducir()". Siempre que aquello que quieras reproducir sea de la clase "Largometraje" o una clase hija, el método te lo aceptará.

Te animas a traducir este texto a Código Pyhton?