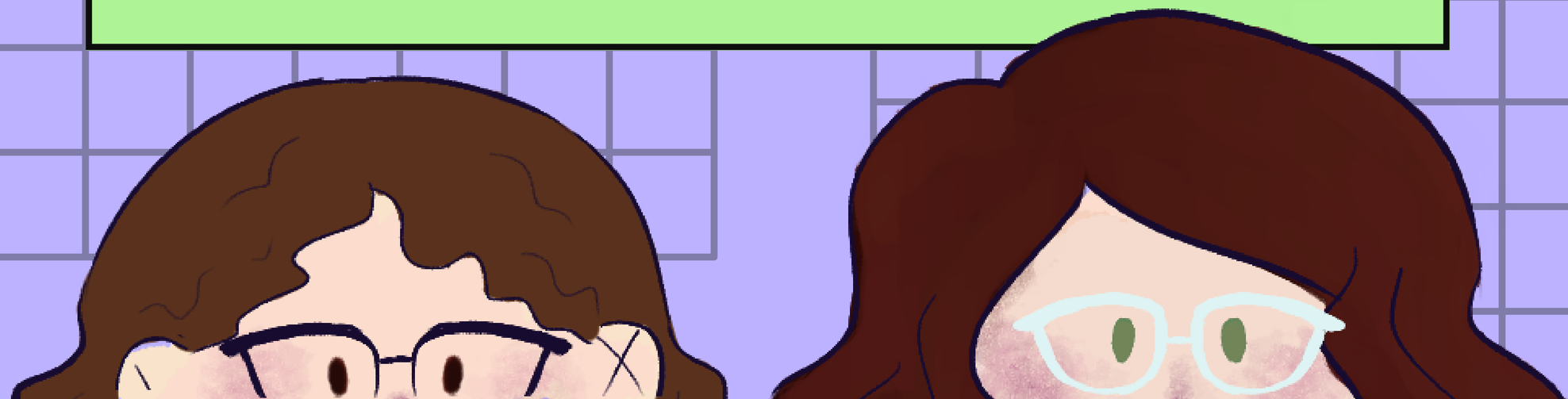
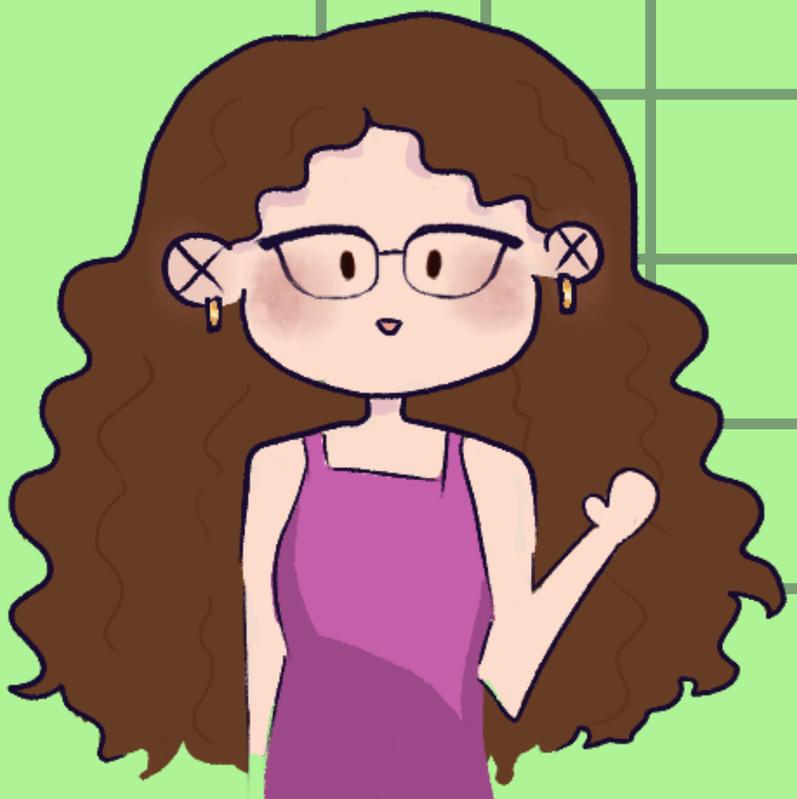
LOS 4 PILARES DE LA OOP EN PHYTON



POO TE PERMITE CODIFICAR MÁS RÁPIDO



Codificar más rápido no significa escribir menos líneas de código. Significa que puedes implementar más funciones en menos tiempo sin comprometer la estabilidad de un proyecto.

La programación orientada a objetos te permite reutilizar el código mediante la implementación de la abstracción. Este principio hace que tu código sea más conciso y legible.

ABSTRACCIÓN

La abstracción oculta al usuario la funcionalidad interna de una aplicación. El usuario puede ser el cliente final u otros desarrolladores.

Podemos encontrar **abstracción** en nuestra vida cotidiana. Por ejemplo, sabes cómo usar tu teléfono, pero probablemente no sepas exactamente lo que ocurre dentro de él cada vez que abres una aplicación.

Otro ejemplo es el propio Python. Sabes cómo usarlo para construir software funcional, y puedes hacerlo, aunque no entiendas el funcionamiento interno de Python.

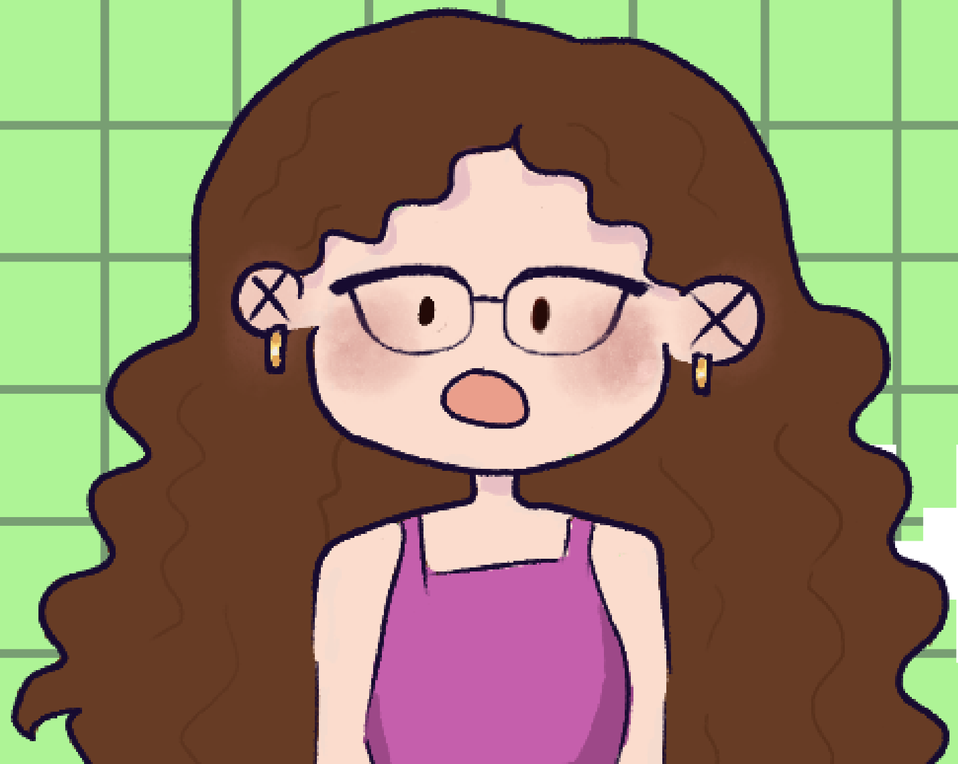
Aplicar lo mismo al código permite reunir todos los objetos de un problema y **abstraer la** funcionalidad estándar en clases.

HERENCIA

La herencia nos permite definir múltiples **subclases** a partir de una clase ya definida.

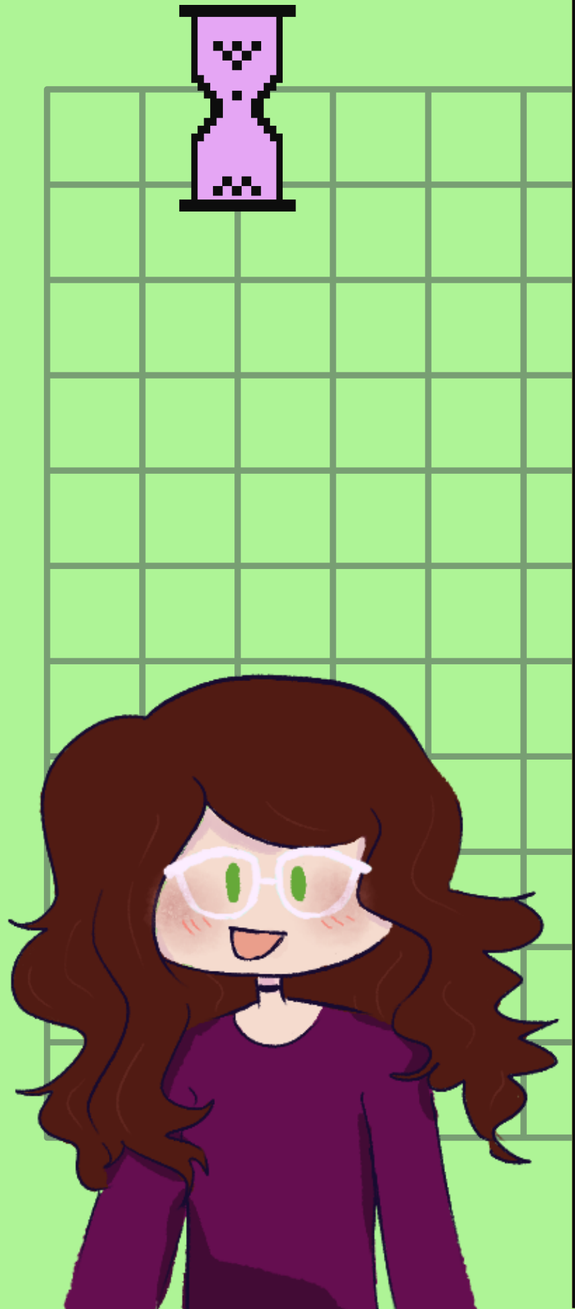
El propósito principal es seguir el principio DRY. Podrás reutilizar mucho código implementando todos los componentes compartidos en **superclases**.
Puedes pensar en ello como el concepto de **herencia genética** en la vida real.

Los hijos (subclases) son el resultado de la herencia entre dos padres (superclases). Heredan todas las características físicas (atributos) y algunos comportamientos comunes (métodos).



POLIMORFISMO

El polimorfismo nos permite modificar ligeramente los métodos y atributos de las **subclases** previamente definidas en la **superclase**.
El significado literal es «**muchas formas**«. Esto se debe a que construimos métodos con el mismo nombre pero con diferente funcionalidad.
Volviendo a la idea anterior, los niños también son un ejemplo perfecto de polimorfismo. Pueden heredar un comportamiento definido **get_hungry()** pero de una manera ligeramente diferente, por ejemplo, tener hambre cada 4 horas en lugar de cada 6.



ENCAPSULACIÓN

La encapsulación es el proceso en el que protegemos la integridad interna de los datos en una clase.

Aunque no hay una declaración **privada** en Python, se puede aplicar la encapsulación mediante el uso de mangling en Python. Existen métodos especiales llamados **getters** y **setters** que nos permiten acceder a atributos y métodos únicos.

Imaginemos una clase **Humana** que tiene un único atributo llamado **_altura**. Este atributo solo se puede modificar dentro de ciertas restricciones (es casi imposible ser más alto que 3 metros).