- Polimorfismo
    - O polimorfismo permite que diferentes classes tenham métodos com o mesmo nome, mas comportamentos diferentes.
    - Em Python, isso pode ser facilmente implementado usando herança e sobrescrita de métodos.
    - Existe 4 tipos:
        - Inclusão: um ponteiro da classe mãe pode apontar para uma instância de uma classe filha
        - Paramétrico: se restringe ao uso de templates (C++, por exemplo) e generics (C#/Java)
        - Sobrecarga: duas funções/métodos com o mesmo nome mas assinaturas diferentes
        - Coerção: conversão implícita de tipos sobre os parâmetros de uma função
    - isinstance(): verifica se o objeto é uma instância de Type ou de qualquer subclasse de Tipo.
        - if isinstance(obj, str): print("obj é uma string ou subclasse de string")
    - type(): verificar se o tipo de um objeto é exatamente tipo, excluindo quaisquer subclasses.
        - if type(objeto) is str: print("obj é exatamente uma string")
    - x=100 y=90 -> print(x==y) #false -> print(x is y) #false
    - a=[2],b=[2] -> print(a==b) #true -> print(x is y) #false

```
from abc import ABC, abstractmethod
class Animal(ABC):
    @abstractmethod
    def fazer_som(self):
        pass
class Cachorro(Animal):
    def fazer_som(self):
        return 'au au'
class Gato(Animal):
    def fazer_som(self):
        return 'miau'
animais = [Cachorro(), Gato()]
for animal in animais:
    print(animal.fazer_som())
```

Singleton

- Singleton é um dos padrões de criação que garante que:
    - uma classe tenha apenas uma instância;
    - e fornece um ponto de acesso global a essa instância.
- Como tornar uma classe Singleton?
    1. A classe vai conter o único objeto existente.
    2. Deve-se restringir o acesso ao construtor, tornando-o um construtor privado, de forma que novos objetos não possam ser criados.
    3. Então utilizar um método público que faça o controle da instanciação, de modo que ela só possa ser feita uma vez.

```
class Singleton:
    _instance = None
    def __new__(cls, *args, **kwargs):
        if not cls._instance:
            cls._instance = super(Singleton, cls).__new__(cls, *args, **kwargs)
        return cls._instance
# main -> s1 = Singleton(); s2 = Singleton()
print(s1 is s2) # True
```