// Class and object

```
class Animal:                  # blueprint
    def __init__(self, name, species, age):
        self.name    = name   # instance attr
        self.species = species
        self.age     = age
        self.status  = 'available'  # default

lassie = Animal('Lassie', 'dog', 4)
print(lassie.name)             # Lassie
```

// self

```
def describe(self):
    print(f'{self.name} is {self.age}.')
# self = the object the method is called on
# passed automatically — never manually
lassie.describe()  # Python passes lassie as self
```

// Class vs instance attribute

```
class Animal:
    shelter = 'Safe Paws'   # class — shared
    def __init__(self, name):
        self.name = name    # instance — unique
```

// Encapsulation

```
self._health  = health   # private by convention
self.__status = status   # name-mangled

def get_health(self):
    return self._health
def set_health(self, h):
    if h in ['healthy', 'ill']:
        self._health = h
```

// Common mistakes

```
self.name = name     # correct
name = name          # wrong — local only

# always call super().__init__()
# never mutable defaults — use None
def __init__(self, items=None):
    self.items = items or []
# always define __str__
```

// Inheritance

```
class Dog(Animal):             # inherits Animal
    def __init__(self, name, age, breed):
        super().__init__(name, age)  # parent first
        self.breed = breed   # Dog's own

    def fetch(self):         # Dog's own method
        print(f'{self.name} fetches!')

isinstance(lassie, Dog)      # True
isinstance(lassie, Animal)   # True
```

// Polymorphism

```
class Dog(Animal):
    def describe(self):      # overrides Animal
        print(f'{self.name} — {self.breed}')

class Cat(Animal):
    def describe(self):      # overrides Animal
        print(f'{self.name} — indoor cat')

for animal in shelter:
    animal.describe()        # correct class auto
```

// Special methods

```
def __str__(self):           # print(obj)
    return f'{self.name}'
def __repr__(self):          # repr(obj)
    return f"Animal('{self.name}')"
def __len__(self):           # len(obj)
    return len(self.animals)
def __eq__(self, other):     # obj == other
    return self.name == other.name
def __contains__(self, item): # item in obj
    return item in self.animals
```

// Two classes — shelter example

```
class Animal:
    def __init__(self, id, name, species):
        self.id      = id
        self.name    = name
        self.species = species
        self.status  = 'available'

class Shelter:
    def __init__(self, name):
        self.name    = name
        self.animals = {}
        self.filename = f'{name}.txt'

s = Shelter('safe_paws')     # independent instance
```