```python
# Примитивная реализация класса
# def — определить поведение
# dict — чтобы составить структуру
# int, str, ... — чтобы хранить данные

class_dog: dict = {
  "fields": {
    "name": ""
    "colour": "",
    "age": 0
  },
  "methods": {
    "__init__": dog__init__
    "bork": dog_static_bork,
    "walk": dog_walk
  }
}

def dog__init__(name, colour, age) -> dict:
  new_dog = class_dog.copy()
  new_dog["fields"]["name"] = name
  new_dog["fields"]["colour"] = colour
  new_dog["fields"]["age"] = age
  return new_dog

def dog_static_bork():
  print("bork-bork")


def dog_walk(self):
  print(f'{self["fields"]["name"]} goes for a
walk!')


new_dog = class_dog.copy()
dog_instance = class_dog.dog__init__("Boba",
"black", 3)

#                .
dog_instance["methods"]["dog_static_bork"]()
dog_instance["methods"]["dog_walk"](dog_instance)

```

```python
class Dog:
  # конструктор — метод, который создает новый
экземпляр класса
  def __init__(self, name: str, colour: str, age:
int):
    self.name: str = name
    self.colour: str = colour
    self.age: int = age

  # Метод — функция ассоциированная с классом
которая как-то использует состояние экземпляра.
Вызывается только от экземпляра
  def dog_walk(self):
    print("bork-bork")

  # Статический метод — функция ассоциированная с
классом которая НЕ использует состояние экземпляра.
Вызывается как от экземпляра, так и от класса
  @staticmethod
  def dog_static_bork():
    print("bork-bork")

  @staticmethod
  def dog_fabric(type: str):
    if type == "malinois":
      return Dog("new_malinois", "deer-black-mask",
0)
    if type == "beegle":
      return Dog("new_beegle", "red", 0)


dog_instance = Dog("Boba", "black", 3)
dog_fabric_instance = Dog.dog_fabric("malinois")

dog_instance.dog_static_bork()
dog_instance.dog_walk()

# Экземпляр/Инстанс — это объект принадлежащий
классу.
# Класс — это инструкция по сборке, а экземпляр —
шкаф который по ней собирают
```

```python
# Dunder Methods or Magical Mathods
# Dunder - double underscore (__)

# Открывает возможность создавать экземпляр вызовом
класса
def __init__(self)  # метод-конструктор

__dict__  # метод который вызывает примитив

# открывает функцию print
def __repr__(self)  # method for representation

# открывает операторы == и in
def __eq__(self)  # метод "equal"/"равно" —
вызывается при обращении к оператору == или при
другом сравнении (in, etc)

# открывает функцию len()
def __len__(self)  # возращает int который как-то
отражает "Длину" объекта при обращении к функции
len()

# открывает возможность использовать экземпляр в
качестве ключа в коллекции dict или в качестве
объекта в set и frozenset
def __hash__(self)

# etc
```

```python
from dataclasses import dataclass

@dataclass
class Dog:
  name: str
  colour: str
  age: int

```