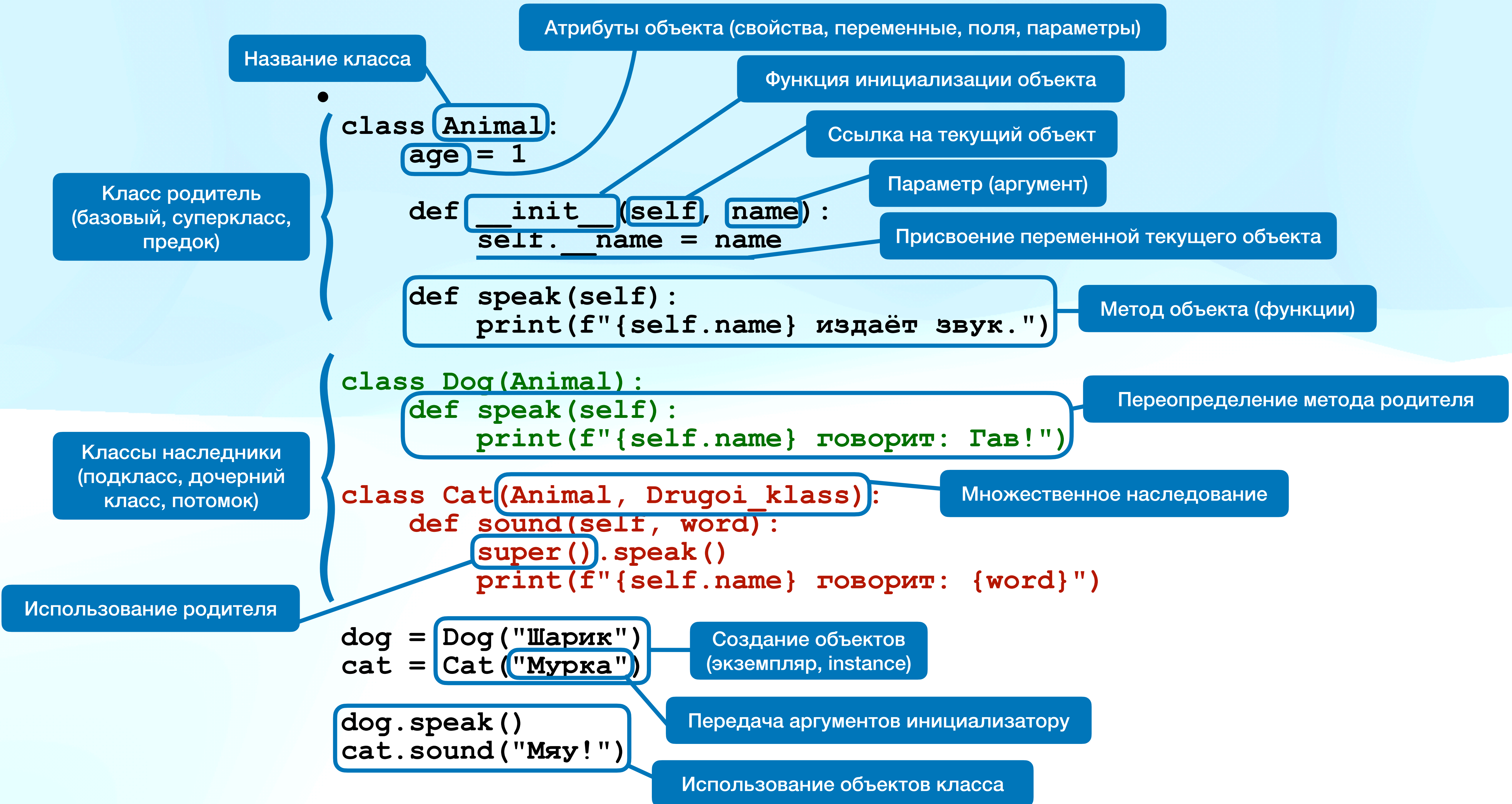


ООП

# Базовые элементы классов, наследование





# Интерфейс и абстрактный класс

Интерфейс

нет реализованных методов

Абстрактный класс

Есть абстрактные и реализованные методы

Реализация методов

```
from abc import ABC, abstractmethod
```

Импорт модуля (класса) абстракции

```
class MoiInterface(ABC):
```

```
@abstractmethod
```

Декоратор метода

оборачивает функцию другой функцией

```
def show_status(self):  
    pass
```

```
@abstractmethod
```

```
def assign_text(self, text):  
    pass
```

```
class MoiAbstract(ABC):
```

```
@abstractmethod
```

```
def work(self, type):  
    pass
```

```
def say(self, text):  
    print(text)
```

Наследуем интерфейс и абстрактный класс

```
class MoyaRealizacia(Moi_interface, Moi_abstract):
```

```
def show_status(self):  
    print('Статус суперский!!')
```






```
def assign_text(self, text):  
    self.text = text
```

```
def work(self, type):  
    self.work_type = type
```

```
moi_instance = MoyaRealizacia()  
moi_instance.show_status()
```

# Инкапсуляция

Это ограничение прямого доступа к внутренним данным объекта и предоставление доступа только через специальные методы.

-  Зачем нужна инкапсуляция:
  -  **Защита данных** — предотвращает случайное или неправильное изменение важных атрибутов.
  -  **Контроль** — можно проверять и валидировать значения при установке.
  -  **Изоляция внутренней логики** — пользователь объекта работает с “интерфейсом”, не вникая в детали.
-  Уровни доступа в Python  
Python не использует жёстких модификаторов доступа, как Java или C++. Вместо этого используются соглашения:
- | Вид        | Синтаксис                | Значение                          |
|------------|--------------------------|-----------------------------------|
| Публичный  | <code>self.name</code>   | Доступен отовсюду                 |
| Защищённый | <code>self._name</code>  | Неофициально: “не трогай извне”   |
| Приватный  | <code>self.__name</code> | Скрыт, используется name mangling |



# Инкапсуляция, пример

```
• class Employee:
```

```
    def __init__(self, name, salary):
```

Публичный атрибут

```
        self.name = name
```

защищённый атрибут (соглашение)

```
        self._department = "IT"
```

```
        self.__salary = salary
```

приватный атрибут

```
    def show_salary(self):  
        return self.__salary
```

Getter

```
    def set_salary(self, amount):  
        self.__salary = amount
```

Setter

```
    def info(self):
```

```
        print(f"Отдел: {self._department}")
```

```
        print(f"Сотрудник: {self.name}")
```

```
        print(f"Зарплата: {self.__salary}")
```

Внутри класса атрибут доступен

```
emp = Employee("Иван", 50000)
```

```
print(emp.name)
```

Работает, но противоречит соглашению

```
print(emp._department)
```

```
emp.set_salary(60000)
```

Установка защищенного атрибута

```
emp.info()
```

```
print(emp.get_salary())
```

Корректное чтение защищенного атрибута

```
print(emp.__salary)
```

Прямое чтение защищенного атрибута даст ошибку AttributeError