# Что такое ООП

В ООП на первом месте стоят объекты. Объект — это набор данных и функций.

Данные внутри объекта называются **свойствами** (или полями), а функции — **методами**. Свойства — это характеристики объекта, а методы — действия, которые умеет выполнять объект.

Каждый объект создаётся по шаблону. Такие шаблоны в ООП называют **классами**.

# Классы в Python

Порядок объявления класса в Python:

1. Создать класс, используя оператор class.
2. После оператора указать имя класса — с большой буквы, в единственном числе; если имя состоит из двух и более слов — используется CamelStyle.
3. Объявить метод __init__, который отвечает за инициализацию объектов класса. Это специальная функция установки начального состояния объекта; её ещё называют **конструктор класса**.
4. Описать остальные методы класса и указать необходимые параметры.

> Параметр self обязательно передаётся первым для всех методов.

Пример объявления класса:

```
class Sword:

    def __init__(self, name, material, blade_length, grip):
        self.name = name
        self.blade_length = blade_length
        self.material = material
        self.grip = grip
        print(f'Новый меч {name} выкован!')

    def slashing_blow(self):
        return (f'Нанесён рубящий удар мечом {self.name}. '
                f'Радиус поражения: {self.blade_length}.')

    def piercing_strike(self):
        return (f'Нанесён пронзающий удар мечом {self.name}. '
                f'Рукоять {self.grip} мягко легла в руку.')

    def sharpen(self):
        return (f'Меч "{self.name}" заточен,'
                f' {self.material} отлично поддалась обработке.')
```

Чтобы создать объект, нужно вызвать конструктор класса. Это значит, что нужно обратиться к классу по имени и передать ему в параметры значения, которые нужны новому объекту:

```
class Sword:

    def __init__(self, name, material, blade_length, grip):
        self.name = name
        self.blade_length = blade_length
        self.material = material
        self.grip = grip
        print(f'Новый меч {name} выкован!')

    def slashing_blow(self):
        return (f'Нанесён рубящий удар мечом {self.name}. '
                f'Радиус поражения: {self.blade_length}.')

    def piercing_strike(self):
        return (f'Нанесён пронзающий удар мечом {self.name}. '
                f'Рукоять {self.grip} мягко легла в руку.')

    def sharpen(self):
        return (f'Меч "{self.name}" заточен,'
                f' {self.material} отлично поддалась обработке.')

# Создаём экземпляр класса Sword.
first_sword = Sword('Тренировочный',
                    'Кора железного дуба',
                    1.2, 'хват одной рукой')
```

## Значения по умолчанию в методах и свойствах класса

Для методов класса можно указывать значения параметров по умолчанию:

```
class Sword:
    # У параметра material указано значение по умолчанию.
    # Параметры со значениями по умолчанию всегда указываются в конце.
    def __init__(self, name, blade_length, grip, material = 'сталь'):
        self.name = name
        self.blade_length = blade_length
        self.material = material
        self.grip = grip
        print(f'Новый меч {name} выкован!')

    ...
```

Параметры со значение по умолчанию нужно записывать **после** параметров без значений по умолчанию.

В свойствах класса также можно задать значение по умолчанию, необязательно передавать его из параметров:

```
class Sword:

    def __init__(self, name, blade_length, grip, material = 'сталь'):
        self.name = name
        self.blade_length = blade_length
        self.material = material
        self.grip = grip
        # Задаём значение по умолчанию в свойствах класса.
        self.strength = 100
        print(f'Новый меч {name} выкован!')

    ...
```

### Печать содержимого объектов

Чтобы напечатать содержимое пользовательских объектов, используйте метод __str__:

```
...

    def __str__(self):
        return (
            f'Меч — «{self.name}». '
            f'Выкован из материала {self.material}, '
            f'длина клинка — {self.blade_length}, '
            f'прочность — {self.strength}.')
```

## Принципы ООП

У парадигмы объектно-ориентированного программирования есть четыре принципа:

- абстракция,
- наследование,
- полиморфизм,
- инкапсуляция.

### Абстракция

Абстракция — это когда мы выделяем только значимые для решения задачи характеристики объекта и игнорируем всё остальное. Чем меньше характеристик, тем лучше абстракция, но ключевые характеристики убирать нельзя.

Класс в ООП — это абстракция, благодаря которой вы можете объявлять объекты внутри программы, не описывая их подробно.

Интерфейс класса — это функциональная часть класса. В ООП интерфейсами называют свойства и методы класса, используя которые можно взаимодействовать с объектом этого класса.

## Наследование

Наследование — возможность описать новый класс на базе существующего. При этом дочерние классы могут заимствовать свойства и методы родительского класса.

> В Python все классы напрямую или через классы-родители — наследники встроенного базового класса object. Все классы в Python могут использовать методы класса object, например, знакомый вам метод __str__() определён именно в классе object.

### Переопределяем методы родительского класса

Функционал класса-родителя можно не только расширить в классах-наследниках, но и полностью переопределить, то есть полностью изменить:

```
class MeleeWeapon:

    def __init__(self, name):
        self.name = name
        self.strength = 100

    # Метод родительского класса.
    def slashing_blow(self):
        # При рубящем ударе уменьшаем прочность меча на 10.
        self.strength -= 10
        return f'Нанесён рубящий удар оружием {self.name}.'

    def sharpen(self):
        self.strength = 100
        return (f'Оружие "{self.name}" заточено.')


class Sword(MeleeWeapon):

    # Переопределяем метод родительского класса...
    def slashing_blow(self):
        # ...меняем значение снижения прочности...
        self.strength -= 5
        # ...и меняем сообщение.
        return f'Мечом {self.name} был нанесён рубящий удар.'

# Этот класс-наследник полностью наследует все методы и свойства
# родительского класса.
class Axe(MeleeWeapon):
    ...
```

**Собственные методы дочерних классов**

```
class Sword(MeleeWeapon):

    def slashing_blow(self):
        self.strength -= 5
        return f'Мечом {self.name} был нанесён рубящий удар.'

    # Собственный метод для класса Sword.
    def piercing_strike(self):
        self.strength -= 5
        return f'Нанесён пронзающий удар мечом {self.name}.'
```

**Расширяем методы родительского класса**

Чтобы объявить свойство класса-родителя в классе-наследнике, используется функция super():

```
class Axe(MeleeWeapon):

    # В классе-наследнике определяется конструктор
    # с собственным параметром material.
    def __init__(self, name, material):
        # Вызываем конструктор класса-родителя.
        super().__init__(name)
        # Передаём значение параметра в новое свойство.
        self.material = material

# Теперь при создании объекта класса Axe
# нужно обязательно указывать два параметра:
# название и материал.
brodex = Axe('Верный', 'железо')
```

## Инкапсуляция

Инкапсуляция — объединение и скрытие методов и свойств и предоставление доступа к ним через простой внешний интерфейс.

## Полиморфизм

Полиморфизм — возможность взаимодействовать с объектами разных классов через одинаковые интерфейсы, обращаться к свойствам и методам, общим для всех объектов.

# Полезные ссылки

Раздел документации Python о классах

Конспект курса информационных технологий в Университете Кейптауна (на английском)