



Объектно-ориентированное программирование

Лавприт Сингх-Пальчевская
младший научный сотрудник МГУ им. Ломоносова,
кафедра биоинженерии

Проверка связи



Отправьте «**+**», если меня видно и слышно

Если у вас нет звука или изображения:

- перезагрузите страницу
- попробуйте зайти заново
- откройте трансляцию в другом браузере

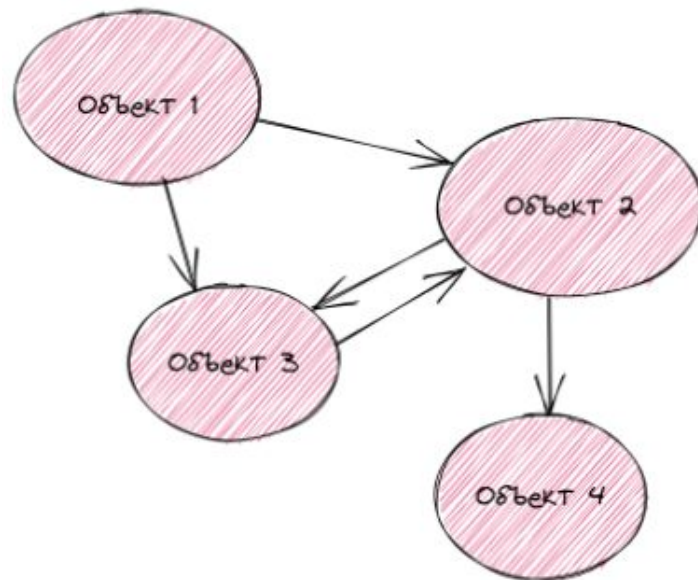
О чем поговорим сегодня

1. Рассмотрим принципы объектно-ориентированного подхода в программировании на Python
2. Рассмотрим организацию механизма наследования в Python
3. Рассмотрим организацию и принципы работы со специальными методами класса в Python
4. Потренируемся в написании программ в объектно-ориентированной парадигме

Объектно-ориентированное программирование в Python

Главный ключ построению сложной системы — введение абстракции.

Программа с точки зрения **ОПП** рассматривается как набор объектов — *отдельных сущностей с внутренней структурой и определенным поведением.* Объекты в программе — это модели объектов, с которыми мы работаем в реальной жизни.



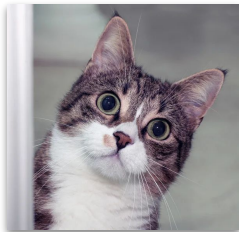
Немного теории

Парадигма ООП предполагает создание интерфейс, который будет обладать определенной структурой: атрибуты и свойства объекта. Например,

Интерфейс, или модель объектов

```
class Cat():
    def __init__(self, name, sex, age, weight):
        self.name = name
        self.sex = sex
        self.age = age
        self.weight = weight

    def cat_info(self):
        return f"My name is {self.name}. I am {'female' if
self.sex=='f' else 'male'}. I am {self.age}. My weight
is {self.weight}."
```

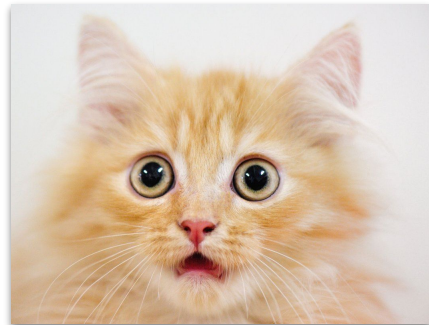


Экземпляр, или реализация модели

```
cats = [Cat('Mila', 'f', 2, 4),
        Cat('Tisha', 'f', 3, 7),
        Cat('Dasha', 'f', 2, 5),
        Cat('Murka', 'f', 1, 3),
        Cat('Milka', 'f', 3, 6),
        Cat('Sima', 'f', 2, 6),
        Cat('Lilia', 'f', 3, 8),
        Cat('Snejana', 'f', 1, 3)]

cats[2].cat_info()
```

Немного теории



Атрибут - собственная
переменная экземпляра класса

Конструктор

```
class Cat():  
    def __init__(self, name, sex, age, weight):  
        self.name = name  
        self.sex = sex  
        self.age = age  
        self.weight = weight
```

self - экземпляр
класса

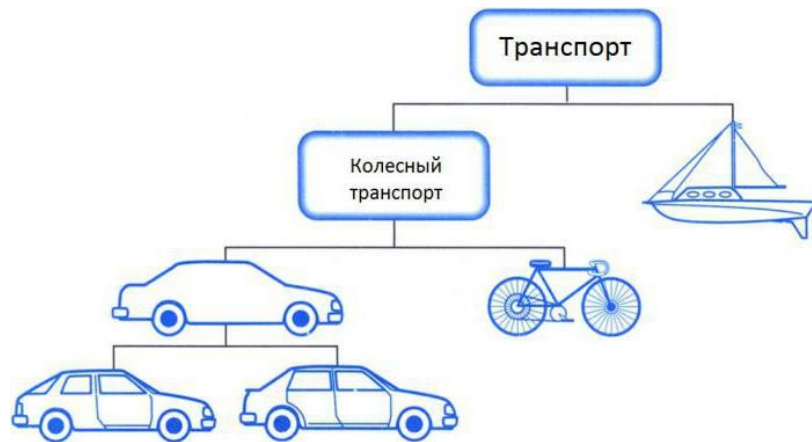
```
def cat_info(self):  
    return f"My name is {self.name}."
```

Метод - СВОЙСТВО КЛАССА

Наследование и композиция

Наследование и композиция

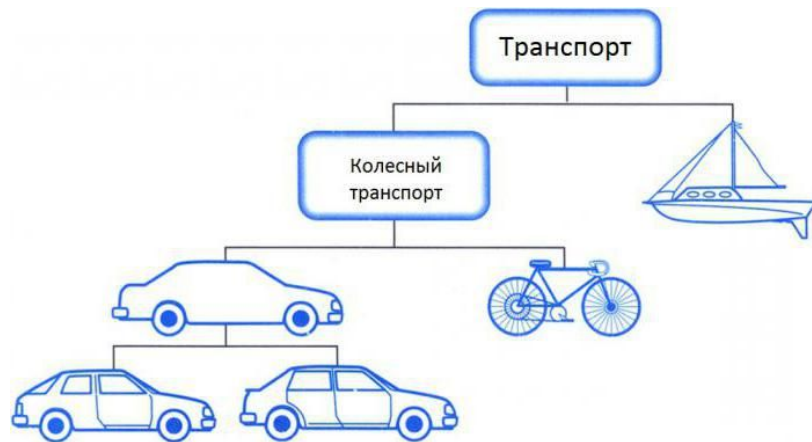
Концепция **наследования (inheritance)** в ООП занимает ключевое место. Она позволяет использовать данные и функциональность некоторого существующего класса как свои собственные.



Наследование и композиция

Концепция **наследования (inheritance)** в ООП занимает ключевое место. Она позволяет использовать данные и функциональность некоторого существующего класса как свои собственные.

Класс-наследник не только получает всю функциональность базового класса, а также может расширять его или добавлять новые детали, которых не было в базовом классе.



Наследование и композиция



```
class Person:
    def __init__(self, id, name, surname, age, tel):
        self.id = id
        self.name = name
        self.surname = surname
        self.age = age
        self.tel = tel
    def full_name(self):
        return f'{self.name} {self.surname}'
```

суперкласс для Student

```
class Student(Person):
    def add_to_group(self, group):
        group.add(self)
    def student_info(self):
        return f'id: {self.id}, university: {self.university}, department: {self.department}'
```

подкласс для Person

Наследование и композиция

```
class Person:
    def __init__(self, id, name, surname, age, tel):
        self.id = id
        self.name = name
        self.surname = surname
        self.age = age
        self.tel = tel
    def full_name(self):
        return f'{self.name} {self.surname}'
```

```
class Student(Person):
    def full_name(self):
        return f'id: {self.id}, full name: {self.name} {self.surname}'
    def add_to_group(self, group):
        group.add(self)
    def student_info(self):
        return f'id: {self.id}, university: {self.university}, department: {self.department}'
```

перегрузка метода - замена
атрибутов суперкласса `Student`
за счет их переопределения в
подклассах `Person`

Наследование и композиция

```
class Person:
    def __init__(self, id, name, surname, age, tel):
        self.id = id
        self.name = name
        self.surname = surname
        self.age = age
        self.tel = tel
    def full_name(self):
        return f'{self.name} {self.surname}'
```

`super()` возвращает временный объект суперкласса, что позволяет получить доступ к методам базового класса.

```
class Student(Person):
    def __init__(self, id, name, surname, age, tel, university, department):
        super().__init__(id, name, surname, age, tel)
        self.university = university
        self.department = department
    def full_name(self):
        return f'id: {self.id}, full name: {self.name} {self.surname}'
    def student_info(self):
        return f'id: {self.id}, university: {self.university}, department: {self.department}'
```

Композиционная модель отношений в ООП

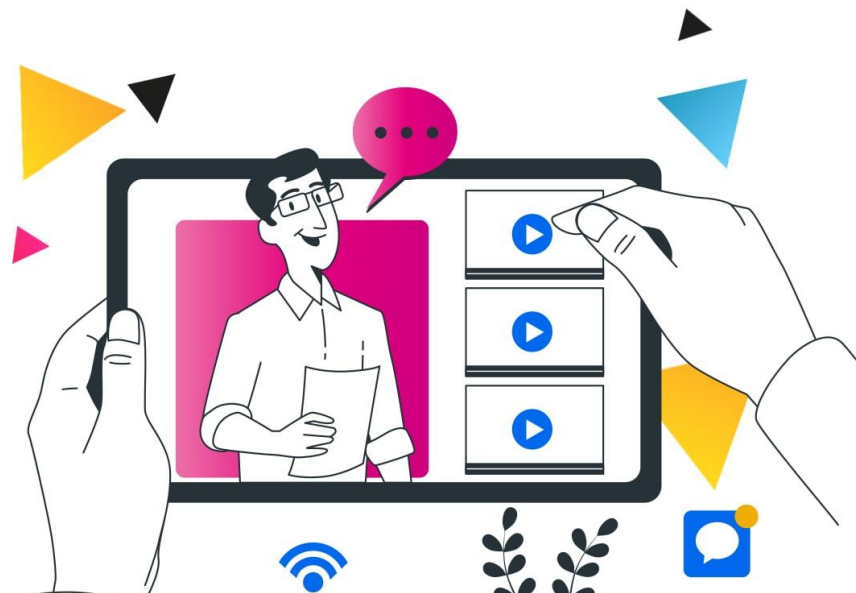
```
class Group(object):
    def __init__(self, id, name, students=None):
        self.id = id
        self.name = name
        self.students = students
    def add(self, student):
        if not self.students: self.students = []
        self.students.append(student)
```

Объекты класса `Student`

Компонент класса `Group`

```
class Student(Person):
    def __init__(self, id, name, surname, age, tel, university, department):
        super().__init__(id, name, surname, age, tel)
        self.university = university
        self.department = department
    def full_name(self):
        return f'id: {self.id}, full name: {self.name} {self.surname}'
    def student_info(self):
        return f'id: {self.id}, university: {self.university}, department: {self.department}'
```

Ваши вопросы?



Специальные методы

Ввиду того, что все пользовательские классы наследуются от `object` (явно или неявно), существуют *базовые методы*, которые также можно переопределять. Такие методы всегда начинаются и заканчиваются *двумя нижними подчеркиваниями*, например `__init__(self [,args...])`.

```
class Person(object):
    def __init__(self, id, name, surname, age, tel):
        self.id = id
        self.name = name
        self.surname = surname
        self.age = age
        self.tel = tel
    def full_name(self):
        return f'{self.name} {self.surname}'
```

Специальные методы



Для более понятного представления бывает полезным представить свой экземпляр класса в виде строки:

```
class Person(object):
    def __init__(self, id, name, surname, age, tel):
        self.id = id
        self.name = name
        self.surname = surname
        self.age = age
        self.tel = tel
    def __str__(self):
        return f'Person ID: {self.id}\nPerson name: {self.name} {self.surname}'
    def full_name(self):
        return f'{self.name} {self.surname}'
```

```
p = Person(1245, 'Nikita', 'Suhov', 25, '+79558682594')
print(p)
```

Person ID: 1245
Person name: Nikita Suhov

Специальные методы



Сравнение объектов:

```
class Person(object):
    def __init__(self, id, name, surname, age, tel):
        self.id = id
        self.name = name
        self.surname = surname
        self.age = age
        self.tel = tel
    def __gt__(self, other):
        return self.age > other.age
    def __lt__(self, other):
        return self.age < other.age
    def __ge__(self, other):
        return self.age >= other.age
    def __le__(self, other):
        return self.age <= other.age
```

```
p1 = Person(1245, 'Nikita', 'Suhov',
25, '+79558682594')
p2 = Person(1345, 'Mihail',
'Neugodov', 28, '+79558682594')
print(p1 > p2)
print(p1 < p2)
print(p1 >= p2)
print(p1 <= p2)
```

False
True
False
True

Специальные методы



Определение арифметических и математических операций:

```
class Vector:
    def __init__(self, *vec):
        self.vec = vec
    def __str__(self):
        return ', '.join([str(i) for i in self.vec])
    def __len__(self):
        return len(self.vec)
    def __add__(self, other):
        return Vector(*[i+j for i, j in zip(self.vec, other.vec)])
    def __sub__(self, vec2):
        return Vector(*[i-j for i, j in zip(self.vec, vec2.vec)])
    def __mul__(self, vec2):
        return sum([i*j for i, j in zip(self.vec, vec2.vec)])
    def __neg__(self):
        return Vector(*[-i for i in self.vec])
```

```
v1 = Vector(2,10)
v2 = Vector(5,-2)
print(v1 + v2)
print(v1 - v2)
print(v1 * v2)
print(-v1)
```

7, 8
-3, 12
-10
-2, -10

Специальные методы

Определение арифметических и математических операций:

```
class Vector:
    def __init__(self, *vec):
        self.vec = vec
    def __str__(self):
        return ', '.join([str(i) for i in self.vec])
    def __len__(self):
        return len(self.vec)
    def __add__(self, other):
        return Vector(*[i+j for i, j in zip(self.vec, other.vec)])
    def __sub__(self, vec2):
        return Vector(*[i-j for i, j in zip(self.vec, vec2.vec)])
    def __mul__(self, vec2):
        return sum([i*j for i, j in zip(self.vec, vec2.vec)])
    def __neg__(self):
        return Vector(*[-i for i in self.vec])
```

```
v1 = Vector(2,10)
v2 = Vector(5,-2)
print(v1 + v2)
print(v1 - v2)
print(v1 * v2)
print(-v1)
```

Благодаря
описанию метода
для функции str()

7, 8
-3, 12
-10
-2, -10

Практика в решении задач

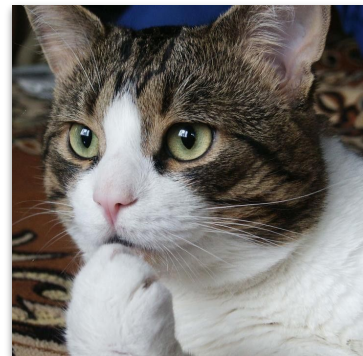
Задача

Создайте структуру классов для работы с векторами и матрицами. Для этого опишите 2 класса: `Vector` и `Matrix`.

Класс `Vector` должен хранить кортеж координат вектора. Чтобы конструктор мог принимать неограниченное количество аргументов, т.е. координат вектора, используйте оператор `*` (подробнее [тут](#)). Для класса `Vector` опишите также методы сложения, вычитания, расчета нормы (длины) вектора и произведения на вектор. (возвращаемый объект должен принадлежать классу `Vector`).

Примеры работы с объектами класса `Vector`:

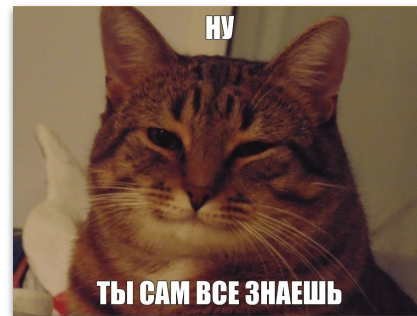
```
a = Vector(1, 2, 3)
b = Vector(3, 4, 5)
print(a.add(b))          <__main__.Vector object at 0x7f32438de2d0>
print(a.subtract(b))     <__main__.Vector object at 0x7f324223bd90>
print(a.add(b).vec)      (4, 6, 8)
print(a.subtract(b).vec) (-2, -2, -2)
```



Задача

Создайте структуру классов для работы с векторами и матрицами. Для этого опишите 2 класса: `Vector` и `Matrix`.

```
class Vector:
    def __init__(self, *vec):
        self.vec = vec
    def norm(self):
        return sum([i**2 for i in self.vec])**.5
    def add(self, vec2):
        return Vector(*[i+j for i, j in zip(self.vec, vec2.vec)])
    def subtract(self, vec2):
        return Vector(*[i-j for i, j in zip(self.vec, vec2.vec)])
    def dot(self, vec2):
        return sum([i*j for i, j in zip(self.vec, vec2.vec)])
```



Задача

Создайте структуру классов для работы с векторами и матрицами. Для этого опишите 2 класса: `Vector` и `Matrix`.

Класс `Matrix` должен хранить кортеж объектов класса `Vector` (т.е. как набор вектор-строк). Чтобы конструктор мог принимать неограниченное количество аргументов, т.е. координат вектора, используйте оператор `*` (подробнее [тут](#)). Для класса `Matrix` опишите также методы сложения, вычитания (возвращаемый объект должен принадлежать классу `Matrix`).

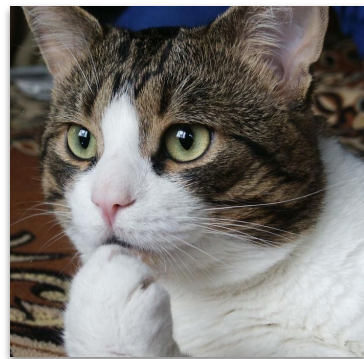
Примеры работы с объектами класса `Matrix`:

```
a = Vector(1, 2, 3)
b = Vector(3, 4, 5)
c = Vector(5, 6, 7)
m = Matrix(a, b, c)
```

сумма двух одинаковых
матриц m

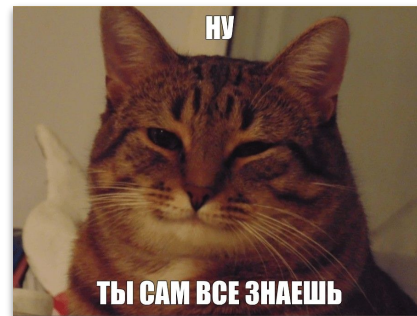
2	4	6
6	8	10
10	12	14

```
print(m.add(Matrix(a, b, c))) <__main__.Matrix object at 0x7f3242215610>
```



Задача

```
class Matrix:
    def __init__(self, *vectors):
        self.matrix = vectors
    def add(self, m2):
        return Matrix(*[Vector(*[i+j for i, j in zip(v.vec, v2.vec)])
                        for v, v2 in zip(self.matrix, m2.matrix)])
    def subtract(self, m2):
        return Matrix(*[Vector(*[i-j for i, j in zip(v.vec, v2.vec)])
                        for v, v2 in zip(self.matrix, m2.matrix)])
```

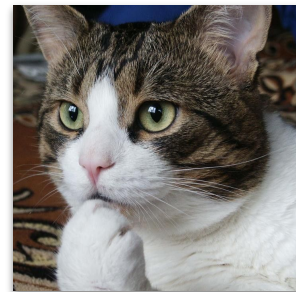


Задача 2

Предположим, в группе порядка 40 человек студентов и вам необходимо их оценить и выбрать 5 лучших. Как это можно было бы автоматизировать?

Система оценивания следующая:

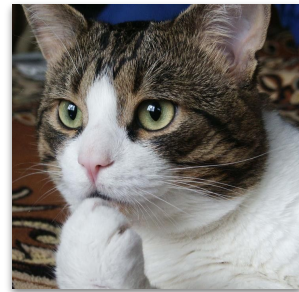
- за каждую работу можно получить от 0 до 10 баллов
- типы работ: контрольная, дз, работа на занятиях, экзамен
- суммарная оценка за семестр рассчитывается так: $0.3 \cdot \text{кр} + 0.2 \cdot \text{дз} + 0.1 \cdot \text{работа на занятии} + 0.4 \cdot \text{экзамен}$



Задача 2

Создайте 2 класса: **Grade** и **StudentCard**.

Grade должна хранить в себе оценки за каждую из работ и метод расчета суммарной оценки за семестр `calc_semester_assessment()`, причем таким образом, чтобы коэффициенты по типам работ можно было передавать в качестве аргумента (но не обязательно!).



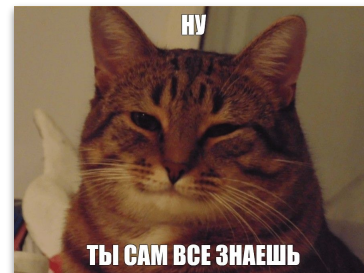
суммарная оценка за семестр: $0.3 \cdot \text{кр} + 0.2 \cdot \text{дз} + 0.1 \cdot \text{работа на занятии} + 0.4 \cdot \text{экзамен}$

Задача 2

Создайте 2 класса: **Grade** и **StudentCard**.

Grade должна хранить в себе оценки за каждую из работ и метод расчета суммарной оценки за семестр `calc_semester_assessment()`, причем таким образом, чтобы коэффициенты по типам работ можно было передавать в качестве аргумента (но не обязательно!).

```
class Grade:
    def __init__(self, kr, hw, attendance, exam):
        self.kr = kr
        self.hw = hw
        self.attendance = attendance
        self.exam = exam
    def calc_semester_assessment(self, coefs=(.3, .2, .1, .4)):
        return
self.kr*coefs[0]+self.hw*coefs[1]+self.attendance*coefs[2]+self.exam*coefs[3]
```

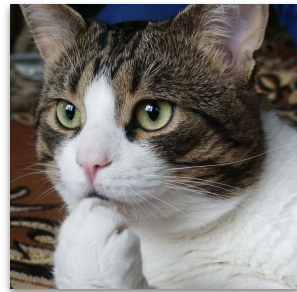


суммарная оценка за семестр: $0.3 \cdot \text{кр} + 0.2 \cdot \text{дз} + 0.1 \cdot \text{работа на занятии} + 0.4 \cdot \text{экзамен}$

Задача 2

Создайте 2 класса: **Grade** и **StudentCard**.

StudentCard должна хранить номер студента, имя и фамилию, причем номер студента должен генериться автоматически. При вызове метода `str(student)` (где `student` - объект класса `Student`) мы должны получать строку вида 'имя фамилия'.

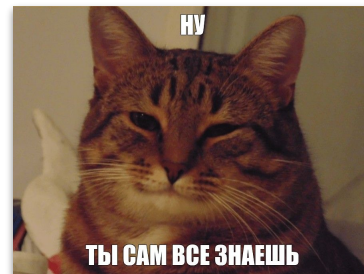


Задача 2

Создайте 2 класса: **Grade** и **StudentCard**.

StudentCard должна хранить номер студента, имя и фамилию, причем номер студента должен генериться автоматически. При вызове метода `str(student)` (где `student` - объект класса `Student`) мы должны получать строку вида 'имя фамилия'.

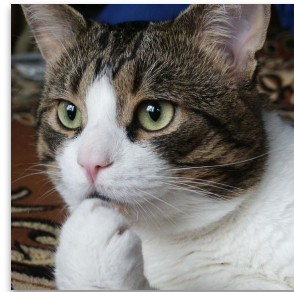
```
class StudentCard:
    ID = 0
    def __init__(self, first_name, last_name):
        self.id = StudentCard.ID
        self.first_name = first_name
        self.last_name = last_name
        StudentCard.ID += 1
    def __str__(self):
        return f'{self.first_name} {self.last_name}'
```



Задача 2

Потестируем созданные классы:

- создайте студента
- выведите полное имя студента (помните про магический метод `__str__`!)
- рассчитайте ее оценку с коэффициентами (.4, .2, .05, .35) и оценками [1, 10, 8, 10]
- создайте еще пару студентов и выведите для каждого id



Задача 2

Потестируем созданные классы:

- создайте студента
- выведите полное имя студента (помните про магический метод `__str__`!)
- рассчитайте ее оценку с коэффициентами (.4, .2, .05, .35) и оценками [1, 10, 8, 10]
- создайте еще пару студентов и выведите для каждого id

```
anna_kim = Student('Anna', 'Kim')  
str(anna_kim), anna_kim.id
```

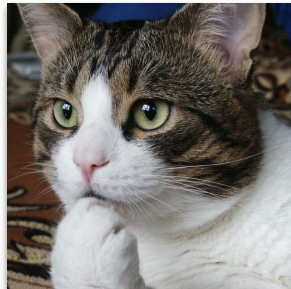
```
Grade(*[1, 10, 8, 10]).calc_semester_assessment(.4, .2, .05, .35)  
masha_perv = Student('Masha', 'Pervushina')  
nadya_muh = Student('Nadya', 'Muhina')  
masha_perv.id, nadya_muh.id
```



Задача 3

Создайте теперь класс **Student**, который будет иметь те же свойства, что и StudentCard, но с возможностью хранения оценки для данного студента. То есть класс Student должен иметь **атрибут**, который будет хранить оценку (по умолчанию *None*), а также **метод**, который рассчитывает ее (используйте класс Grade!).

Этот класс должен позволять **сравнивать** студентов между собой, в т.ч. сортировать их.

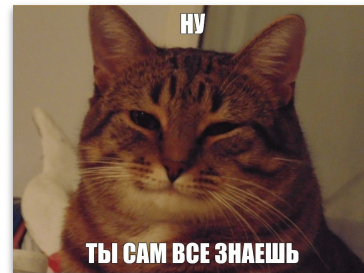


Задача 3

```
class Student(StudentCard):
    def __init__(self, first_name, last_name, grade=None):
        super().__init__(first_name, last_name)
        self.grade = grade

    def __eq__(self, other):
        return self.grade == other.grade
    def __ge__(self, other):
        return self.grade >= other.grade
    def __gt__(self, other):
        return self.grade > other.grade
    def __le__(self, other):
        return self.grade <= other.grade
    def __lt__(self, other):
        return self.grade < other.grade
    def calc_grade(self, grades):
        self.grade = Grade(*grades).calc_semester_assessment()
```

```
anna_kim = Student('Anna', 'Kim')
anna_kim.calc_grade([10, 10, 10, 10])
lana_rock = Student('Lana', 'Rock')
lana_rock.calc_grade([10, 8, 8, 10])
# anna_kim.grade = 10.0
# lana_rock.grade = 9.399999999999999
print(anna_kim == lana_rock) #False
print(anna_kim <= lana_rock) #False
print(anna_kim < lana_rock) #False
print(anna_kim >= lana_rock) #True
print(anna_kim > lana_rock) #True
```

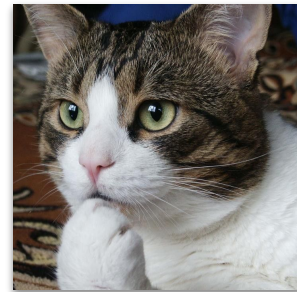


Задача 4

Напишите **функцию сортировки** списка *вставками* (см. описание [здесь](#)).
Реализуйте ее таким образом, чтобы менялся сам список (функция при этом не должна ничего возвращать).

А затем передайте на вход список студентов:

```
students = [Student('Anna', 'Kim', 10),  
            Student('Locky', 'Kranel', 8),  
            Student('Lana', 'Rock', 2),  
            Student('Anita', 'Roy', 8),  
            Student('Nick', 'Varma', 10),  
            Student('Mila', 'Kim', 5),  
            Student('Milana', 'Dante', 3),  
            Student('Luna', 'Kran', 5),  
            Student('Santa', 'Roon', 9),  
            Student('Evelina', 'Luck', 3),  
            Student('Maria', 'Roy', 6)]
```



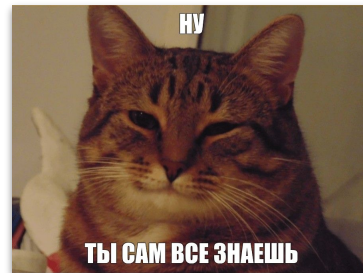
Задача 4

```
students = [Student('Anna', 'Kim', 10),
             Student('Locky', 'Kranel', 8),
             Student('Lana', 'Rock', 2),
             Student('Anita', 'Roy', 8),
             Student('Nick', 'Varma', 10),
             Student('Mila', 'Kim', 5),
             Student('Milana', 'Dante', 3),
             Student('Luna', 'Kran', 5),
             Student('Santa', 'Roon', 9),
             Student('Evelina', 'Luck', 3),
             Student('Maria', 'Roy', 6)]
```

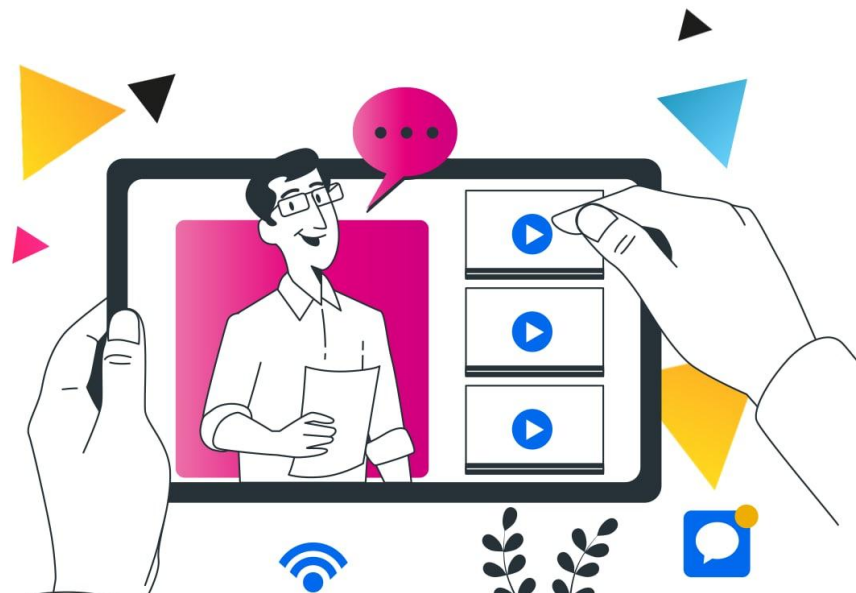
```
insertion_sort(students)
for s in students:
    print(s, s.grade)
```

```
def insertion_sort(mylist):
    for i, el in enumerate(mylist[1:]):
        for j, prev_el in enumerate(mylist[i::-1]):
            if el < prev_el:
                mylist[i-j] = el
                mylist[i-j+1] = prev_el
```

Lana Rock 2
Milana Dante 3
Evelina Luck 3
Mila Kim 5
Luna Kran 5
Maria Roy 6
Locky Kranel 8
Anita Roy 8
Santa Roon 9
Anna Kim 10
Nick Varma 10



Ваши вопросы?



Итоги занятия

1. Рассмотрели принципы объектно-ориентированного подхода в программировании на Python
2. Рассмотрели организацию и принципы работы со специальными методами класса в Python
3. Потренировались в написании программ в объектно-ориентированной парадигме

Дополнительные материалы по теме занятия



1. Документация по использованию аннотации:
<https://docs.python.org/3/library/typing.html>
2. Декораторы:
https://colab.research.google.com/drive/1vuuWNPFlOfFFeS5XW10JwkFfSUv5jP_F?usp=sharing
3. Магические методы
 - a. https://propyprogs.ru/python_oop
 - b. <https://habr.com/ru/post/186608/>

Ваши вопросы?

