Table of Contents

[Main.py 5](#_Toc35435431)

[\_\_init\_\_.py (на уровне modules) 5](#_Toc35435432)

[Задание 1 5](#_Toc35435433)

[Условие 5](#_Toc35435434)

[\_\_init\_\_.py 5](#_Toc35435435)

[task1.py 5](#_Toc35435436)

[Задание 2 6](#_Toc35435437)

[Условие 6](#_Toc35435438)

[\_\_init\_\_.py 6](#_Toc35435439)

[task2.py 6](#_Toc35435440)

[Задание 3 6](#_Toc35435441)

[Условие 6](#_Toc35435442)

[\_\_init\_\_.py 6](#_Toc35435443)

[task3.py 6](#_Toc35435444)

[Задание 4 7](#_Toc35435445)

[Условие 7](#_Toc35435446)

[\_\_init\_\_.py 7](#_Toc35435447)

[task4.py 7](#_Toc35435448)

[Задание 5 7](#_Toc35435449)

[Условие 7](#_Toc35435450)

[\_\_init\_\_.py 8](#_Toc35435451)

[abiturientclass.py 8](#_Toc35435452)

[baseclass.py 8](#_Toc35435453)

[get\_jobs\_list.py 9](#_Toc35435454)

[get\_userinfo.py 9](#_Toc35435455)

[get\_ways\_list.py 9](#_Toc35435456)

[studentclass.py 9](#_Toc35435457)

[teacherclass.py 9](#_Toc35435458)

[Задание 6 10](#_Toc35435459)

[Условие 10](#_Toc35435460)

[\_\_init\_\_.py 10](#_Toc35435461)

[circle\_class.py 10](#_Toc35435462)

[figure\_class.py 10](#_Toc35435463)

[rectangle\_class.py 11](#_Toc35435464)

[triangle\_class.py 11](#_Toc35435465)

[Задание 7 11](#_Toc35435466)

[Условие 11](#_Toc35435467)

[\_\_init\_\_.py 11](#_Toc35435468)

[article\_class.py 12](#_Toc35435469)

[book\_class.py 12](#_Toc35435470)

[publication\_class.py 12](#_Toc35435471)

[web\_class.py 12](#_Toc35435472)

[Задание 8 13](#_Toc35435473)

[Условие 13](#_Toc35435474)

[\_\_init\_\_.py 13](#_Toc35435475)

[equilateral\_triangle\_class.py 13](#_Toc35435476)

[isosceles\_triangle\_class.py 13](#_Toc35435477)

[tectangular\_triangle\_class.py 14](#_Toc35435478)

[triangle\_class.py 14](#_Toc35435479)

[triangleclass\_calculation.py 14](#_Toc35435480)

[Задание 9 14](#_Toc35435481)

[Условие 14](#_Toc35435482)

[\_\_init\_\_.py 15](#_Toc35435483)

[car\_class.py 15](#_Toc35435484)

[motorcycle\_class.py 15](#_Toc35435485)

[transport\_class.py 15](#_Toc35435486)

[truck\_class.py 16](#_Toc35435487)

[Задание 10 16](#_Toc35435488)

[Условие 16](#_Toc35435489)

[\_\_init\_\_.py 16](#_Toc35435490)

[goods\_class.py 16](#_Toc35435491)

[group\_class.py 17](#_Toc35435492)

[phone\_class.py 17](#_Toc35435493)

[product\_class.py 17](#_Toc35435494)

[search\_module.py 17](#_Toc35435495)

[Задание 11 18](#_Toc35435496)

[Условие 18](#_Toc35435497)

[\_\_init\_\_.py 18](#_Toc35435498)

[book\_class.py 18](#_Toc35435499)

[goods\_class.py 18](#_Toc35435500)

[search\_module.py 19](#_Toc35435501)

[sportgoods\_class.py 19](#_Toc35435502)

[toy\_class.py 19](#_Toc35435503)

[Задание 12 19](#_Toc35435504)

[Условие 19](#_Toc35435505)

[\_\_init\_\_.py 19](#_Toc35435506)

[friend\_class.py 20](#_Toc35435507)

[organization\_class.py 20](#_Toc35435508)

[person\_class.py 20](#_Toc35435509)

[phonedictionary\_class.py 20](#_Toc35435510)

[search\_module.py 21](#_Toc35435511)

[Задание 13 21](#_Toc35435512)

[Условие 21](#_Toc35435513)

[\_\_init\_\_.py 21](#_Toc35435514)

[clent\_class.py 21](#_Toc35435515)

[contributor\_class.py 22](#_Toc35435516)

[creditor\_class.py 22](#_Toc35435517)

[organization\_class.py 22](#_Toc35435518)

[search\_module.py 22](#_Toc35435519)

[Задание 14 23](#_Toc35435520)

[Условие 23](#_Toc35435521)

[\_\_init\_\_.py 23](#_Toc35435522)

[commercial\_class.py 23](#_Toc35435523)

[freeware\_class.py 24](#_Toc35435524)

[software\_class.py 24](#_Toc35435525)

[trial\_class.py 24](#_Toc35435526)

[util\_module.py 24](#_Toc35435527)

[Задание 15 25](#_Toc35435528)

[Условие 25](#_Toc35435529)

[\_\_init\_\_.py 25](#_Toc35435530)

[airplane\_class.py 25](#_Toc35435531)

[car\_class.py 25](#_Toc35435532)

[ship\_class.py 26](#_Toc35435533)

[transport\_class.py 26](#_Toc35435534)

[util\_module.py 26](#_Toc35435535)

[Задание 16 27](#_Toc35435536)

[Условие 27](#_Toc35435537)

[\_\_init\_\_.py 27](#_Toc35435538)

[ball\_class.py 27](#_Toc35435539)

[car\_class.py 27](#_Toc35435540)

[cube\_class.py 27](#_Toc35435541)

[toy\_class.py 28](#_Toc35435542)

[util\_module.py 28](#_Toc35435543)

[Задание 17 28](#_Toc35435544)

[Условие 28](#_Toc35435545)

[\_\_init\_\_.py 28](#_Toc35435546)

[ball\_class.py 29](#_Toc35435547)

[body\_class.py 29](#_Toc35435548)

[parallelepiped\_class.py 29](#_Toc35435549)

[pyramid\_class.py 29](#_Toc35435550)

[Задание 18 30](#_Toc35435551)

[Условие 30](#_Toc35435552)

[\_\_init\_\_.py 30](#_Toc35435553)

[equationclass.py 30](#_Toc35435554)

[linearequation.py 30](#_Toc35435555)

[parse\_exp.py 31](#_Toc35435556)

[parseexp\_koff\_a.py 31](#_Toc35435557)

[parseexp\_koff\_b.py 31](#_Toc35435558)

[parseexp\_koff\_c.py 31](#_Toc35435559)

[quadraticequation.py 31](#_Toc35435560)

[quadraticequation\_calculation.py 32](#_Toc35435561)

[utils.py 32](#_Toc35435562)

[Задание 19 32](#_Toc35435563)

[Условие 32](#_Toc35435564)

[\_\_init\_\_.py 32](#_Toc35435565)

[currency\_class.py 33](#_Toc35435566)

[eur\_class.py 33](#_Toc35435567)

[usd\_class.py 33](#_Toc35435568)

[Задание 20 33](#_Toc35435569)

[Условие 33](#_Toc35435570)

[\_\_init\_\_.py 33](#_Toc35435571)

[arithmetic\_class.py 34](#_Toc35435572)

[geometric\_class.py 34](#_Toc35435573)

[progression\_class.py 34](#_Toc35435574)

Main.py**import** modules  
**def** main():  
 d\_selector = {}  
 **for** i **in** range(1,21):  
 d\_selector[str(i)]=eval(**"modules.task"**+str(i))  
 input\_str = **""  
 while** input\_str != **"0"**:  
 input\_str = input(**"\033[93mВведите номер задания (1-20) или 0 для завершения работы -> \033[0m"**)  
 **if** input\_str **in** d\_selector:  
 d\_selector[input\_str]()  
 **else**:  
 print(**"Такого номера не существует!"**)  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()

# \_\_init\_\_.py (на уровне modules)

**from** .task1 **import** main **as** task1  
**from** .task2 **import** main **as** task2  
**from** .task3 **import** main **as** task3  
**from** .task4 **import** main **as** task4  
**from** .task5 **import** main **as** task5  
**from** .task6 **import** main **as** task6  
**from** .task7 **import** main **as** task7  
**from** .task8 **import** main **as** task8  
**from** .task9 **import** main **as** task9  
**from** .task10 **import** main **as** task10  
**from** .task11 **import** main **as** task11  
**from** .task12 **import** main **as** task12  
**from** .task13 **import** main **as** task13  
**from** .task14 **import** main **as** task14  
**from** .task15 **import** main **as** task15  
**from** .task16 **import** main **as** task16  
**from** .task17 **import** main **as** task17  
**from** .task18 **import** main **as** task18  
**from** .task19 **import** main **as** task19  
**from** .task20 **import** main **as** task20

# Задание 1

## Условие

Создайте класс с методом класса, в котором определялась бы сумма двух целых чисел.

## \_\_init\_\_.py

**import** modules  
**def** main():  
 d\_selector = {}  
 **for** i **in** range(1,21):  
 d\_selector[str(i)]=eval(**"modules.task"**+str(i))  
 input\_str = **""  
 while** input\_str != **"0"**:  
 input\_str = input(**"\033[93mВведите номер задания (1-20) или 0 для завершения работы -> \033[0m"**)  
 **if** input\_str **in** d\_selector:  
 d\_selector[input\_str]()  
 **else**:  
 print(**"Такого номера не существует!"**)  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()

## task1.py

**class** Task1Class:  
 @staticmethod  
 **def** sum(a, b):  
 **return** a + b

# Задание 2

## Условие

Создайте класс с методом-конструктором, в котором следует определить атрибуты экземпляра класса, необходимые для сложения двух целых чисел. Напишите метод, в котором бы определялась сумма двух целых чисел.

## \_\_init\_\_.py

**from** .task2 **import** Task2Class  
**def** main():  
 a = float(input(**"Введите число №1 ->"**))  
 b = float(input(**"Введите число №2 ->"**))  
 obj = Task2Class(a, b)  
 print(**"Задание 2:"**, obj.sum())

## task2.py

**class** Task2Class:  
 **def** \_\_init\_\_(self, a, b):  
 self.a = a  
 self.b = b  
 **def** sum(self):  
 **return** self.a + self.b

# Задание 3

## Условие

Создайте класс с методами, формирующими вложенную последовательность. Пользователю должна быть предоставлена возможность заполнить ее либо случайными числами в интервале [-10; 10], либо осуществить ввод данных с клавиатуры.

## \_\_init\_\_.py

**from** .task3 **import** Task3Class  
**def** main():  
 print(**"Задание 3:"**)  
 Task3Class()

## task3.py

**from** random **import** randrange  
**class** Task3Class:  
 **def** \_\_init\_\_(self):  
 **try**:  
 d = {**"1"**: self.user\_input,**"2"**: self.random\_input,}  
 self.main\_list = []  
 self.list\_len = int(input(**"Введите длинну последовательности -> "**))  
 method\_number = input(**"Как вы хотите заполнить последовательность?\n1. Вручную\n2. Автоматически\n-> "**)  
 **if** method\_number **in** d:  
 d[method\_number]()  
 self.out\_list()  
 **else**: print(**"Некорректный ввод"**)  
 **except** Exception **as** e: print(**"Возникла ошибка: "**, e)  
 **def** number\_checker(self, e):  
 **try**: **return** int(e)  
 **except** ValueError: **return** e  
 **def** user\_input(self):  
 **for** i **in** range(self.list\_len):  
 locale\_element = self.number\_checker(input(**"Введите элемент №"** + str(i + 1) + **" -> "**))  
 self.main\_list.append(locale\_element)  
 **def** random\_input(self):  
 self.main\_list = [randrange(-10, 10) **for** \_ **in** range(self.list\_len)]  
 **def** out\_list(self):  
 print(**"Полученная последовательность:"**)  
 print(self.main\_list)

# Задание 4

## Условие

Разработайте класс с соответствующими методами, обеспечивающий нахождение значения функции г и вывод на экран результатов вычислений, Исходные данные в соответстви с вариантом функции первого семестра.

## \_\_init\_\_.py

**from** .task4 **import** Task4Class  
**def** main():  
 print(**"\nЗадание 4:"**)  
 **try**:  
 x = float(input(**"Введите x -> "**))  
 **except** ValueError:  
 print(**"Проблема ввода данных!"**)  
 **return** obj = Task4Class(x)  
 print(**"Результат: "** + str(obj.result))

## task4.py

**import** math  
**class** Task4Class:  
 **def** \_\_init\_\_(self, x):  
 self.x = x  
 self.getter()  
 **def** getter(self):  
 x = self.x  
 upper = x \*\* 3 \* math.e \*\* (x - 1)  
 lower = x \*\* 3 - math.fabs(x)  
 **if** lower == 0:  
 print(**"Знаменатель равен нулю, деление на 0!"**)  
 self.result = 0  
 **return** first = upper / lower  
 log\_sqrt = math.sqrt(x) - x  
 **if** log\_sqrt >= 0:  
 buf\_log = math.log(log\_sqrt, 2)  
 **else**:  
 print(**"Выражение в log[sqrt(x)-x,2] меньше 0!"**)  
 self.result = 0  
 **return** self.result = first - buf\_log

# Задание 5

## Условие

Создайте класс ПЕРСОНА с методами, позволяющими вывести на экран информацию о персоне, а также определить ее возраст (в текущем году). Создайте дочерние классы: АБИТУРИЕНТ (фамилия, дата рождения, факультет), СТУДЕНТ (фамилия, дата рождения, факультет, курс), ПРЕПОДАВАТЕЛЬ (фамилия, дата рождения, факультет, должность, стаж), со своими методами вывода информации на экран и определения возраста. Создайте список из n персон, выведите полную информацию из базы на экран, а также организуйте поиск персон, чей возраст попадает в заданный диапазон. Комментарий. В родительском классе Persona() определим, в соответствии с условием задачи, метод vozrast(), служащий для определения возраста и метод info(), позволяющий вывести информацию о персоне. Далее создаем три дочерних класса: Abiturient(Persona), Student(Persona), Prepodavatel(Persona), основанные на классе Persona(). Соответственно, все дочерние классы будут наследовать методы родительского класса. Чтобы вызвать конструктор базового класса, можно использовать функцию Python - super(). Заметим, что при использовании функции super() можно не передавать в явном виде параметр self.

## \_\_init\_\_.py

**from** .studentclass **import** Student  
**from** .baseclass **import** Person  
**from** .abiturientclass **import** Abiturient  
**from** .teacherclass **import** Teacher  
**from** .get\_jobs\_list **import** get **as** get\_job  
**from** .get\_ways\_list **import** get **as** get\_way  
**from** .get\_userinfo **import** processing **as** input\_proc  
**import** random  
**from** datetime **import** datetime  
**from** faker **import** Faker  
**def** main():  
 n, min\_age, max\_age = input\_proc()  
 fake = Faker([**'ru\_RU'**])  
 all\_obj\_list = []  
 d = {1: Person, 2: Abiturient, 3: Student, 4: Teacher,}  
 **for** \_ **in** range(n):  
 d\_args = {1: [fake.name(), fake.date()], 2: [fake.name(), fake.date(), get\_way()],3: [fake.name(), fake.date(), get\_way(), random.randint(1, 4)],4: [fake.name(), fake.date(), get\_way(), get\_job(), random.randint(5, 35)],}  
 r = random.randint(1, 4)  
 all\_obj\_list.append(d[r](\*d\_args[r]))  
 all\_obj\_list.sort(key=**lambda** e: e.years\_old\_int())  
 **for** obj **in** all\_obj\_list:  
 obj.info()  
 obj.years\_old()  
 **if** obj.years\_old\_int() **in** range(min\_age, max\_age + 1): print(**"[Попадает в диапазон]"**)

## abiturientclass.py

**from** .baseclass **import** Person  
**class** Abiturient(Person):  
 **def** \_\_init\_\_(self, name, birthday, way):  
 super().\_\_init\_\_(name, birthday)  
 self.way = way  
  
 **def** info(self):  
 print(**"\n\*Класс абитуриента\*\nФИО: "** + self.name + \  
 **"\n"** + **"Дата рождения: "** + self.birthday + \  
 **"\nФакультет: "** + self.way)

## baseclass.py

**from** datetime **import** datetime  
**class** Person:  
 **def** \_\_init\_\_(self, name, birthday):  
 self.name = name  
 self.birthday = birthday  
 self.year\_detector()  
 **def** year\_detector(self):  
 *"""  
 Метод определения возраста на основе полной даты рождения + datetime  
 """* now = datetime.now()  
 year\_now = int(now.strftime(**"%Y"**))  
  
 year, \_, \_ = map(int, self.birthday.split(**"-"**))  
 self.year = year\_now - year  
 **def** info(self):  
 print(**"\n\*Класс персона\*\nФИО: "** + self.name + \  
 **"\n"** + **"Дата рождения: "** + self.birthday)  
 **def** years\_old(self):  
 print(**"Возраст: "** + str(self.year))  
 **def** years\_old\_int(self):  
 **return** self.year

## get\_jobs\_list.py

**from** random **import** choice   
**def** get():  
 jobs\_list = [  
 **"Старший научный сотрудник"**,  
 **"Старший преподаватель"**,  
 **"Профессор"**,  
 **"Преподаватель"**,  
 **"Научный сотрудник"**,  
 **"Младший научный сотрудник"**,  
 **"Доцент"**,  
 **"Докторант"**,  
 **"Главный научный сотрудник"**,  
 **"Ведущий научный сотрудник"**,  
 **"Ассистент"** ]  
 **return** choice(jobs\_list)

## get\_userinfo.py

**def** processing():  
 n = int(input(**"Ведите количество персон -> "**))  
 person\_range = list(map(int, input(**"Введите диапазон для поиска персон по возрасту\nПример: 36-40\n-> "**).split(**"-"**)))  
 min\_age, max\_age = sorted(person\_range)  
 **return** n, min\_age, max\_age

## get\_ways\_list.py

**from** random **import** choice   
**def** get():  
 ways\_list = [  
 **"Факультет анализа рисков и экономической безопасности им. профессора В.К. Сенчагова"**,  
 **"Государственное управление и финансовый контроль"**,  
 **"Менеджмент"**,  
 **"Международные экономические отношения​"**,  
 **"Международный туризм, спорт и гостиничный бизнес"**,  
 **"Международный финансовый"**,  
 **"Налоги и налогообложение"**,  
 **"Прикладная математика и информационные технологии"**,  
 **"Социология и политология"**,  
 **"Учет и аудит"**,  
 **"Финансово-экономический"**,  
 **"Финансовых рынков"**,  
 **"Юридический"**,  
 ]  
 **return** choice(ways\_list)

## studentclass.py

**from** .baseclass **import** Person  
**class** Student(Person):  
 **def** \_\_init\_\_(self, name, birthday, way, course):  
 super().\_\_init\_\_(name, birthday)  
 self.way = way  
 self.course = str(course)  
 **def** info(self):  
 print(**"\n\*Класс студента\*\nФИО: "** + self.name + \  
 **"\n"** + **"Дата рождения: "** + self.birthday + \  
 **"\nФакультет: "** + self.way + \  
 **"\nКурс: "** + self.course)

## teacherclass.py

**from** .baseclass **import** Person  
**class** Teacher(Person):  
 **def** \_\_init\_\_(self, name, birthday, way, position, years):  
 super().\_\_init\_\_(name, birthday)  
 self.way = way  
 self.job\_position = position  
 self.job\_years = str(years)  
  
 **def** info(self):  
 print(**"\n\*Класс учителя\*\nФИО: "** + self.name + \  
 **"\n"** + **"Дата рождения: "** + self.birthday + \  
 **"\nФакультет: "** + self.way + \  
 **"\nДолжность: "** + self.job\_position + \  
 **"\nСтаж: "** + self.job\_years)

# Задание 6

## Условие

Создайте класс ФИГУРА с методами вычисления площади и периметра, а также методом, выводящим информацию о фигуре на экран. Создайте дочерние классы ПРЯМОУГОЛЬНИК, КРУГ, ТРЕУГОЛЬНИК со своими методами вычисления площади и периметра. Создайте список п фигур и выведите полную информацию о фигурах на экран.

## \_\_init\_\_.py

**from** math **import** pi, sqrt  
**from** random **import** randint  
**from** .triangle\_class **import** TriangleClass  
**from** .rectangle\_class **import** RectangleClass  
**from** .circle\_class **import** CircleClass  
**def** main():  
 **try**:  
 n = int(input(**"Введите количество фигур -> "**))  
 **except** ValueError:  
 print(**"Некорректный ввод данных"**)  
 **return** d = {0: RectangleClass,1: CircleClass,2: TriangleClass,}  
 figures\_list = []  
 **for** \_ **in** range(n):  
 d\_args = {0: [randint(1, 100), randint(1, 100)], 1: [randint(1, 100)], 2: [randint(1, 100), randint(1, 100), randint(1, 100)]}  
 r\_number = randint(0, 2)  
 figures\_list.append(d[r\_number](\*d\_args[r\_number]))  
 **for** figure **in** figures\_list: figure.info(type(figure).\_\_name\_\_)

## circle\_class.py

**import** math  
**from** .figure\_class **import** FigureClass  
**class** CircleClass(FigureClass):  
 **def** \_\_init\_\_(self, r):  
 self.r = r  
 self.perimeter\_calculation()  
 self.area\_calculation()  
 **def** area\_calculation(self):  
 self.area = math.pi \* self.r \*\* 2  
 **def** perimeter\_calculation(self):  
 self.perimeter = 2 \* math.pi \* self.r

## figure\_class.py

**from** abc **import** ABCMeta, abstractmethod  
**class** FigureClass:  
 \_\_metaclass\_\_ = ABCMeta  
 @abstractmethod  
 **def** perimeter\_calculation(self):  
 self.perimeter = 0  
 @abstractmethod  
 **def** area\_calculation(self):  
 self.area = 0  
 **def** info(self, lclass):  
 a = self.area  
 p = self.perimeter  
 **if** type(self.area) == float **or** type(self.perimeter) == float:  
 a = str(round(self.area, 2))  
 p = str(round(self.perimeter, 2))  
 print(**"\nВызов от {}\nПлощадь фигуры: {}\nПериметр фигуры: {}"**.format(lclass, a, p))

## rectangle\_class.py

**from** .figure\_class **import** FigureClass  
**class** RectangleClass(FigureClass):  
 **def** \_\_init\_\_(self, a, b):  
 self.a = a  
 self.b = b  
 self.perimeter\_calculation()  
 self.area\_calculation()  
 **def** perimeter\_calculation(self):  
 self.perimeter = (self.a + self.b) \* 2  
 **def** area\_calculation(self):  
 self.area = self.a \* self.b

## triangle\_class.py

**from** .figure\_class **import** FigureClass  
**class** TriangleClass(FigureClass):  
 **def** \_\_init\_\_(self, a, b, c):  
 self.a = a  
 self.b = b  
 self.c = c  
 **if** self.validator():  
 self.area\_calculation()  
 self.perimeter\_calculation()  
 **def** validator(self):  
 a = self.a  
 b = self.b  
 c = self.c  
 **if** a + b <= c **or** a + c <= b **or** b + c <= a:  
 self.area = **"Не существует"** self.perimeter = **"Не существует"  
 return False  
 return True  
 def** area\_calculation(self):  
 p = (self.a + self.b + self.c) / 2  
 s = pow(p \* (p - self.a) \* (p - self.b) \* (p - self.c),1/2)  
 self.area = s  
 **def** perimeter\_calculation(self):  
 self.perimeter = self.a + self.b + self.c

# Задание 7

## Условие

Создайте класс ИЗДАНИЕ с методом, позволяющим вывести на экран информацию об издании, а также определить, является ли данное издание искомым. Создайте дочерние классы КНИГА (название, фамилия автора, год издания, издательство), СТАТЬЯ (название, фамилия автора, название журнала, его номер и год издания), ЭЛЕКТРОННЫЙ РЕСУРС (название, фамилия автора, ссылка, аннотация) со своими методами вывода информации на экран. Создайте список из п изданий, выведите полную информацию из списка, а также организуйте поиск изданий по фамилии автора.

## \_\_init\_\_.py

**from** faker **import** Faker  
**from** .publication\_class **import** PublicationClass  
**from** .web\_class **import** WebClass  
**from** .article\_class **import** ArticleClass  
**from** .book\_class **import** BookClass  
**def** main():  
 **try**:  
 n = int(input(**"Введите количество изданий -> "**))  
 **except** ValueError:  
 print(**"Некорректный ввод данных"**)  
 **return** fake, search\_flag = Faker([**'ru\_RU'**]), **False** all\_publications\_list = []  
 **for** \_ **in** range(n): all\_publications\_list.append(PublicationClass(fake.word(), fake.name()))  
 **for** pub **in** all\_publications\_list:  
 pub.info()  
 print()  
 search\_name = input(**"Введите фамилию автора для поиска издания -> "**)  
 **for** pub **in** all\_publications\_list:  
 **if** search\_name **in** pub.author\_name:  
 print(**"[Издание найдено]"**)  
 pub.info()  
 search\_flag = **True  
 if not** search\_flag:  
 print(**"Изданий не найдено"**)

## article\_class.py

**from** .publication\_class **import** PublicationClass  
**class** ArticleClass(PublicationClass):  
 *"""Класс СТАТЬЯ (название, фамилия автора, название журнала, его номер и год издания),"""* **def** \_\_init\_\_(self, name, author\_name, publish\_year, journal\_name, journal\_number):  
 super().\_\_init\_\_(name, author\_name)  
 self.publish\_year = str(publish\_year)  
 self.journal\_name = str(journal\_name)  
 self.journal\_number = str(journal\_number)  
 **def** info(self):  
 print(**"Информация о статье\nНазвание:"** + self.name + **"\nАвтор:"** + self.author\_name + **"\n"** + **"Год издания:"** + self.publish\_year + **"\nНазвание журнала:"** + self.journal\_name + **"\nНомер журнала"** + self.journal\_number)

## book\_class.py

**from** .publication\_class **import** PublicationClass  
**class** BookClass(PublicationClass):  
 *"""Класс КНИГА (название, фамилия автора, год издания, издательство)"""* **def** \_\_init\_\_(self, name, author\_name, publish\_year, publishing\_house):  
 super().\_\_init\_\_(name, author\_name)  
 self.publish\_year = str(publish\_year)  
 self.publishing\_house = str(publishing\_house)  
 **def** info(self):  
 print(  
 **"Информация о книге\nНазвание:"** + self.name + **"\nАвтор:"** + self.author\_name + **"\n"** + **"Год издания:"** + self.publish\_year + **"\nИздательство:"** + self.publishing\_house)

## publication\_class.py

**class** PublicationClass:  
 *"""  
 Класс ИЗДАНИЕ с методом, позволяющим вывести на экран информацию об издании,  
 а также определить, является ли данное издание искомым.  
 """* **def** \_\_init\_\_(self, name, author\_name):  
 self.name = str(name)  
 self.author\_name = str(author\_name)  
 **def** info(self):  
 print(**"Информация об издании\nНазвание:"** + self.name + **"\nАвтор:"** + self.author\_name)

## web\_class.py

**from** .publication\_class **import** PublicationClass  
**class** WebClass(PublicationClass):  
 *"""Класс ЭЛЕКТРОННЫЙ РЕСУРС (название, фамилия автора, ссылка, аннотация)"""* **def** \_\_init\_\_(self, name, author\_name, link, annotation):  
 super().\_\_init\_\_(name, author\_name)  
 self.link = str(link)  
 self.annotation = str(annotation)  
 **def** info(self):  
 print(**"Информация об электронном ресурсе\nНазвание: "** + self.name + **"\nАвтор: "** + self.author\_name + **"\n"** + **"Ссылка: "** + self.link + **"\nАннотация"** + self.annotation)

# Задание 8

## Условие

Создайте класс ТРЕУГОЛЬНИК, заданный длинами двух сторон и угла между ними, с методами вычисления площади и периметра треугольника, а также методом, выводящим информацию о фигуре на экран. Создайте дочерние классы ПРЯМОУГОЛЬНЫЙ, РАВНОБЕДРЕННЫЙ, РАВНОСТОРОННИЙ со своими методами вычисления площади и периметра. Создайте список п треугольников и выведите полную информацию о треугольниках на экран.

## \_\_init\_\_.py

**import** math  
**from** abc **import** ABCMeta, abstractmethod  
**from** random **import** randint  
**from** .equilateral\_triangle\_class **import** EquilateralTriangleClass  
**from** .isosceles\_triangle\_class **import** IsoscelesTriangleClass  
**from** .tectangular\_triangle\_class **import** TectangularTriangleClass  
**from** .triangle\_class **import** TriangleClass  
**def** main():  
 **try**:  
 n = int(input(**"Введите количество треугольников -> "**))  
 **except** ValueError:  
 print(**"Некорректный ввод данных"**)  
 **return** triangle\_list = []  
 d = {0: EquilateralTriangleClass,1: IsoscelesTriangleClass, 2: TectangularTriangleClass,3: TriangleClass,}  
 **for** \_ **in** range(n):  
 r\_number = randint(0, 3)  
 r\_args = [randint(1, 100), randint(1, 100), randint(1, 100)]  
 triangle\_list.append(d[r\_number](\*r\_args))  
 **for** triangle **in** triangle\_list:  
 triangle.info(type(triangle).\_\_name\_\_)

## equilateral\_triangle\_class.py

**import** math  
**from** .triangle\_class **import** TriangleClass  
**class** EquilateralTriangleClass(TriangleClass):  
 **def** \_\_init\_\_(self, a\_side, b\_side, angle):  
 super().\_\_init\_\_(a\_side, b\_side, angle)  
 self.a\_side = a\_side  
 **def** perimeter\_calculation(self):  
 self.perimeter = self.a\_side \* 3  
 **def** area\_calculation(self):  
 **try**:  
 a = self.a\_side  
 h = math.sqrt(pow(a, 2) - (pow(a, 2) / 4))  
 self.area = (1 / 2) \* a \* h  
 **except** ValueError:  
 self.area = **"Ошибка вычисления"**

## isosceles\_triangle\_class.py

**import** math  
**from** .triangle\_class **import** TriangleClass  
**class** IsoscelesTriangleClass(TriangleClass):  
 **def** \_\_init\_\_(self, a\_side, b\_side, angle):  
 super().\_\_init\_\_(a\_side, b\_side, angle)  
 self.a\_side = a\_side  
 self.b\_side = b\_side  
 **def** perimeter\_calculation(self):  
 self.perimeter = self.b\_side \* 2 + self.a\_side  
 **def** area\_calculation(self):  
 a = self.a\_side  
 b = self.b\_side  
 **try**:  
 part\_a = a / 2  
 h = math.sqrt(pow(b, 2) - pow(part\_a, 2))  
 self.area = (h \* a) / 2  
 **except** ValueError:  
 self.area = **"Ошибка вычисления"**

## tectangular\_triangle\_class.py

**import** math  
**from** .triangle\_class **import** TriangleClass  
**class** TectangularTriangleClass(TriangleClass):  
 **def** \_\_init\_\_(self, a\_side, b\_side, angle=90):  
 self.angle = 90  
 super().\_\_init\_\_(a\_side, b\_side, angle)  
 self.a\_side = a\_side  
 self.b\_side = b\_side  
 self.perimeter\_calculation()  
 self.area\_calculation()  
 self.area\_calculation()  
 **def** area\_calculation(self):  
 self.area = (1 / 2) \* self.a\_side \* self.b\_side  
 **def** perimeter\_calculation(self):  
 a = self.a\_side  
 b = self.b\_side  
 c = math.sqrt(pow(a, 2) + pow(b, 2))  
 self.perimeter = a + b + c

## triangle\_class.py

**import** math  
**from** .triangleclass\_calculation **import** \*  
**class** TriangleClass:  
 **def** \_\_init\_\_(self, a\_side, b\_side, angle):  
 self.a\_side = a\_side  
 self.b\_side = b\_side  
 self.angle = angle  
 self.c\_side, self.perimeter = perimeter\_calculation(a\_side, b\_side, angle)  
 self.area = area\_calculation(a\_side, b\_side, self.c\_side)  
   
 **def** info(self, cl\_name):  
 a = self.area  
 p = self.perimeter  
 p = str(round(self.perimeter, 2))  
 **if** type(self.area) == float:  
 a = str(round(self.area, 2))  
 print(**"\nВызов от {}\nПлощадь фигуры: {}\nПериметр фигуры: {}"**.format(cl\_name, a, p))

## triangleclass\_calculation.py

**import** math  
**def** area\_calculation(a,b,c):  
 *# Вычисление полупериметра* p = (a + b + c) / 2  
 s = math.sqrt(p \* (p - a) \* (p - b) \* (p - c))  
 **return** s  
  
**def** perimeter\_calculation(a,b, angle):  
 c = math.sqrt(b \*\* 2 + a \*\* 2 - 2 \* b \* a \* math.cos(math.radians(angle)))  
 **return** c, a + b + c

# Задание 9

## Условие

Создайте класс ТРАНСПОРТ с методами, позволяющими вывести на экран информацию о транспортном средстве, а также определить грузоподъемность транспортного средства. Создайте дочерние классы АВТОМОБИЛЬ (марка, номер, скорость, грузоподъемность), МОТОЦИКЛ (марка, номер, скорость, грузоподъемность, наличие коляски, при этом если коляска отсутствует, то грузоподъемность равна нулю), ГРУЗОВИК (марка, номер, скорость, грузоподъемность, наличие прицепа, при этом если есть прицеп, то грузоподъемность увеличивается в два раза) со своими методами вывода информации на экран и определения грузоподъемности. Создайте список из п машин, выведите полную информацию на экран, а также организуйте поиск машин, удовлетворяющих требованиям грузоподъемности.

## \_\_init\_\_.py

**from** random **import** randint  
**from** .random\_auto **import** random\_auto, random\_number  
**from** .transport\_class **import** TransportClass  
**from** .car\_class **import** CarClass  
**from** .motorcycle\_class **import** MotorcycleClass  
**from** .truck\_class **import** TruckClass  
**def** main():  
 n = int(input(**"Введите количество транспорта -> "**))  
 d = {1: TransportClass, 2: CarClass, 3: MotorcycleClass, 4: TruckClass,}  
 transport\_list, search\_flag = [], **False  
 for** \_ **in** range(n):  
 r\_int = randint(1, 4)  
 d\_args = {1: (random\_auto(), random\_number(), randint(60, 300), randint(500, 10000)), 2: (random\_auto(), random\_number(), randint(60, 300), randint(500, 10000)), 3: (random\_auto(), random\_number(), randint(60, 300), randint(500, 10000), bool(randint(0, 1))), 4: (random\_auto(), random\_number(), randint(60, 300), randint(500, 10000), bool(randint(0, 1))),}  
 transport\_list.append(d[r\_int](\*d\_args[r\_int]))  
 **for** transport **in** transport\_list: print(transport.get\_info() + **"\n"**)  
 carrying\_input = float(input(**"Введите грузоподъёмность -> "**))  
 print(**"\n\*\*Транспорт, грузоподъёмность которого меньше или равна заданной\*\*\n"**)  
 **for** transport **in** transport\_list:  
 **if** transport.get\_carrying() <= carrying\_input:  
 print(transport.get\_info() + **"\n"**)  
 search\_flag = **True  
 if not** search\_flag: print(**"\*\*Транспорт не найден\*\*"**)

## car\_class.py

**from** .transport\_class **import** TransportClass  
**class** CarClass(TransportClass):  
 **def** \_\_init\_\_(self, model, number, speed, carrying):  
 super().\_\_init\_\_(model, number, speed, carrying)  
 **def** get\_info(self):  
 **return "[Автомобиль]\nМарка: "** + self.model + **"\nНомер: "** + self.number + **"\nСкорость: "** + str(  
 self.speed) + **"\nГрузоподъёмность: "** + str(self.carrying)

## motorcycle\_class.py

**from** .transport\_class **import** TransportClass  
**class** MotorcycleClass(TransportClass):  
 **def** \_\_init\_\_(self, model, number, speed, carrying, addition):  
 super().\_\_init\_\_(model, number, speed, carrying)  
 self.addition = addition  
 self.formater\_dict = {**True**: **"Да"**, **False**: **"Нет"**,}  
 self.carrying\_calculating()  
 **def** carrying\_calculating(self):  
 **if** self.addition == **False**: self.carrying = 0  
 **def** get\_info(self):  
 **return "[Мотоцикл]\nМарка: "** + self.model + **"\nНомер: "** + self.number + **"\nСкорость: "** + str(  
 self.speed) + **"\nГрузоподъёмность: "** + str(self.carrying) + **"\nНаличие коляски: "** + self.formater\_dict[  
 self.addition]

## transport\_class.py

**class** TransportClass:  
 **def** \_\_init\_\_(self, model, number, speed, carrying):  
 self.model = str(model)  
 self.number = str(number)  
 self.speed = speed  
 self.carrying = carrying  
 **def** get\_info(self):  
 **return "[Транспорт]\nМарка: "** + self.model + **"\nНомер: "** + self.number + **"\nСкорость: "** + str(  
 self.speed) + **"\nГрузоподъёмность: "** + str(self.carrying)  
 **def** get\_carrying(self):  
 **return** self.carrying

## truck\_class.py

**from** .transport\_class **import** TransportClass  
**class** TruckClass(TransportClass):  
 **def** \_\_init\_\_(self, model, number, speed, carrying, addition):  
 super().\_\_init\_\_(model, number, speed, carrying)  
 self.addition = addition  
 self.formater\_dict = {**True**: **"Да"**,**False**: **"Нет"**,}  
 self.carrying\_calculating()  
 **def** carrying\_calculating(self):  
 **if** self.addition == **True**: self.carrying \*= 2  
 **def** get\_info(self):  
 **return "[Грузовик]\nМарка: "** + self.model + **"\nНомер: "** + self.number + **"\nСкорость: "** + str(self.speed) + **"\nГрузоподъёмность: "** + str(self.carrying) + **"\nНаличие прицепа: "** + self.formater\_dict[self.addition]

# Задание 10

## Условие

Создайте класс ТОВАР с методами, позволяющими вывести на экран информацию о товаре, а также определить, может ли приобрести товар покупатель, имеющий заданную сумму денег. Создайте дочерние классы ПРОДУКТ (название, цена, дата производства, срок годности), ПАРТИЯ (название, цена за штуку, количество штук, дата производства, срок годности), ТЕЛЕФОН (название, цена) со своими методами вывода информации на экран и определения соответствия заданной цене. Создайте список из п товаров, выведите полную информацию из базы на экран, а также организуйте поиск товара, который может приобрести покупатель, имеющий заданную сумму денег.

## \_\_init\_\_.py

**from** random **import** randint  
**from** faker **import** Faker  
**from** .goods\_class **import** GoodsClass  
**from** .product\_class **import** ProductClass  
**from** .group\_class **import** GroupClass  
**from** .phone\_class **import** PhoneClass  
**from** .search\_module **import** \*  
**def** main():  
 **try**:  
 n = int(input(**"Введите количество товаров -> "**))  
 **except** ValueError:  
 print(**"Некорректный ввод данных"**)  
 **return** d = {1: GoodsClass, 2: ProductClass, 3: GroupClass, 4: PhoneClass,}  
 fake = Faker([**'ru\_RU'**])  
 goods\_list = []  
 **for** \_ **in** range(n):  
 r\_int = randint(1, 4)  
 d\_args = {1: (fake.word(), randint(1, 1000000), fake.date(pattern=**'%d.%m.%Y'**), fake.date(pattern=**'%d.%m.%Y'**)), 2: (fake.word(), randint(1, 1000000), fake.date(pattern=**'%d.%m.%Y'**), fake.date(pattern=**'%d.%m.%Y'**)),3: (fake.word(), randint(1, 1000000), fake.date(pattern=**'%d.%m.%Y'**), fake.date(pattern=**'%d.%m.%Y'**),randint(1, 100000)),4: (fake.word(), randint(1, 1000000)),}  
 goods\_list.append(d[r\_int](\*d\_args[r\_int]))  
 displaying(goods\_list)  
 search(goods\_list)

## goods\_class.py

**class** GoodsClass:  
 **def** \_\_init\_\_(self, name, price, manufacture\_date, expiration\_date):  
 self.name = name  
 self.price = price  
 self.manufacture\_date = manufacture\_date  
 self.expiration\_date = expiration\_date  
 **def** get\_info(self):  
 **return "[Товар]\nНазвание: "** + self.name + **"\nЦена: "** + str(  
 self.price) + **" руб.\nДата производства: "** + self.manufacture\_date + **"\nСрок годности: "** + self.expiration\_date  
 **def** opportunity\_detector(self, user\_price):  
 **if** user\_price >= self.price:  
 **return True  
 return False**

## group\_class.py

**from** .goods\_class **import** GoodsClass  
**class** GroupClass(GoodsClass):  
 **def** \_\_init\_\_(self, name, price, manufacture\_date, expiration\_date, amount):  
 super().\_\_init\_\_(name, price, manufacture\_date, expiration\_date)  
 self.amount = amount  
 **def** get\_info(self):  
 **return "[Партия]\nНазвание: "** + self.name + **"\nЦена: "** + str(  
 self.price) + **" руб.\nДата производства: "** + self.manufacture\_date + **"\nСрок годности: "** + self.expiration\_date + **"\nКоличество штук: "** + str(  
 self.amount)

## phone\_class.py

**from** .goods\_class **import** GoodsClass  
**class** PhoneClass(GoodsClass):  
 **def** \_\_init\_\_(self, name, price):  
 self.name = name  
 self.price = price  
 **def** get\_info(self):  
 **return "[Телефон]\nНазвание: "** + self.name + **"\nЦена: "** + str(self.price) + **" руб."**

## product\_class.py

**from** .goods\_class **import** GoodsClass  
**class** ProductClass(GoodsClass):  
 **def** \_\_init\_\_(self, name, price, manufacture\_date, expiration\_date):  
 super().\_\_init\_\_(name, price, manufacture\_date, expiration\_date)  
 **def** get\_info(self):  
 **return "[Продукт]\nНазвание: "** + self.name + **"\nЦена: "** + str(  
 self.price) + **" руб.\nДата производства: "** + self.manufacture\_date + **"\nСрок годности: "** + self.expiration\_date

## search\_module.py

**def** displaying(goods\_list):  
 **for** goods **in** goods\_list:  
 print(goods.get\_info() + **"\n"**)  
**def** search(goods\_list):  
 **try**:  
 price\_input = float(input(**"Введите цену -> "**))  
 **except** ValueError:  
 print(**"Некорректный ввод данных"**)  
 **return** search\_flag = **False** print(**"Товары, которые вы можете себе позволить:"**)  
 **for** goods **in** goods\_list:  
 **if** goods.opportunity\_detector(price\_input) == **True**:  
 print(goods.get\_info() + **"\n"**)  
 search\_flag = **True  
 if not** search\_flag:  
 print(**"Товары не найдены"**)

# Задание 11

## Условие

Создайте класс ТОВАР с методами, позволяющими вывести на экран информацию о товаре, а также определить, предназначен ли он для заданного возраста потребителя. Создайте дочерние классы ИГРУШКА (название, цена, производитель, материал, возраст, на который рассчитана), КНИГА (название, автор, цена, издательство, возраст, на который рассчитана), СПОРТИНВЕНТАРЬ (название, цена, производитель, возраст, на который рассчитан) со своими методами вывода информации на экран и определения соответствия возрасту потребителя. Создайте список из п товаров, выведите полную информацию из базы на экран, а также организуйте поиск товаров для потребителя в заданном возрастном диапазоне.

## \_\_init\_\_.py

**from** random **import** randint  
**from** faker **import** Faker  
**from** .goods\_class **import** GoodsClass  
**from** .toy\_class **import** ToyClass  
**from** .book\_class **import** BookClass  
**from** .sportgoods\_class **import** SportGoodsClass  
**from** .search\_module **import** search   
**def** main():  
 **try**:  
 n = int(input(**"Введите количество товаров -> "**))  
 **except** ValueError:  
 print(**"Некорректный ввод данных"**)  
 **return** d = {1: GoodsClass, 2: ToyClass, 3: BookClass, 4: SportGoodsClass,}  
 fake = Faker([**'ru\_RU'**])  
 goods\_list = []  
 **for** \_ **in** range(n):  
 r\_int = randint(1, 4)  
 d\_args = {1: (fake.word(), randint(1, 1000000), fake.word(), randint(1, 18)),2: (fake.word(), randint(1, 1000000), fake.word(), randint(1, 18), **"пластик"**),3: (fake.word(), randint(1, 1000000), fake.word(), randint(1, 18), fake.name()),4: (fake.word(), randint(1, 1000000), fake.word(), randint(1, 18)),}  
 goods\_list.append(d[r\_int](\*d\_args[r\_int]))  
 **for** goods **in** goods\_list:  
 print(goods.get\_info() + **"\n"**)  
 search(goods\_list)

## book\_class.py

**from** .goods\_class **import** GoodsClass  
**class** BookClass(GoodsClass):  
 **def** \_\_init\_\_(self, name, price, manufacturer, age, author):  
 super().\_\_init\_\_(name, price, manufacturer, age)  
 self.author = author  
 **def** get\_info(self):  
 **return "[Информация о книге]\nНазвание: "** + self.name + **"\nЦена: "** + str(  
 self.price) + **"\nИздательство: "** + self.manufacturer + **"\nВозраст, на который рассчитана: "** + str(  
 self.age) + **"\nАвтор: "** + self.author

## goods\_class.py

**class** GoodsClass:  
 **def** \_\_init\_\_(self, name, price, manufacturer, age):  
 self.name = name  
 self.price = price  
 self.manufacturer = manufacturer  
 self.age = age  
 **def** get\_info(self):  
 **return "[Информация о товаре]\nНазвание: "** + self.name + **"\nЦена: "** + str(  
 self.price) + **"\nПроизводитель: "** + self.manufacturer + **"\nВозраст, на который рассчитан: "** + str(self.age)  
 **def** age\_calculation(self, age\_input):  
 **if** age\_input >= self.age:  
 **return True  
 return False**

## search\_module.py

**def** search(goods\_list):  
 **try**:  
 age\_input = int(input(**"Введите возраст для фильтрации -> "**))  
 **except** ValueError:  
 print(**"Некорректный ввод данных"**)  
 **return** search\_flag = **False** print(**"Товары c фильтрацией по возрасту"**)  
 **for** goods **in** goods\_list:  
 **if** goods.age\_calculation(age\_input) == **True**:  
 print(goods.get\_info() + **"\n"**)  
 search\_flag = **True  
 if not** search\_flag:  
 print(**"Товары не найдены"**)

## sportgoods\_class.py

**from** .goods\_class **import** GoodsClass  
**class** SportGoodsClass(GoodsClass):  
 **def** \_\_init\_\_(self, name, price, manufacturer, age):  
 super().\_\_init\_\_(name, price, manufacturer, age)  
 **def** get\_info(self):  
 **return "[Информация о спортинвентаре]\nНазвание: "** + self.name + **"\nЦена: "** + str(  
 self.price) + **"\nПроизводитель: "** + self.manufacturer + **"\nВозраст, на который рассчитан: "** + str(self.age)

## toy\_class.py

**from** .goods\_class **import** GoodsClass  
**class** ToyClass(GoodsClass):  
 **def** \_\_init\_\_(self, name, price, manufacturer, age, material):  
 super().\_\_init\_\_(name, price, manufacturer, age)  
 self.material = material  
 **def** get\_info(self):  
 **return "[Информация об игрушке]\nНазвание: "** + self.name + **"\nЦена: "** + str(  
 self.price) + **"\nПроизводитель: "** + self.manufacturer + **"\nВозраст, на который рассчитана: "** + str(  
 self.age) + **"\nМатериал: "** + self.material

# Задание 12

## Условие

Создайте класс ТЕЛЕФОННЫЙ СПРАВОЧНИК с методами, позволяющими вывести на экран информацию о записях в телефонном справочнике, а также определить соответствие записи критерию поиска. Создайте дочерние классы ПЕРСОНА (фамилия, адрес, номер телефона), ОРГАНИЗАЦИЯ (название, адрес, телефон, факс, контактное лицо), ДРУГ (фамилия, адрес, номер телефона, дата рождения) со своими методами вывода информации на экран и определения соответствия заданной фамилии. Создайте список из п записей, выведите полную информацию из базы на экран, а также организуйте поиск в базе по фамилии.

## \_\_init\_\_.py

**from** random **import** randint  
**from** faker **import** Faker  
**from** .phonedictionary\_class **import** PhoneDictionaryClass  
**from** .person\_class **import** PersonClass  
**from** .organization\_class **import** OrganizationClass  
**from** .friend\_class **import** FriendClass  
**from** .search\_module **import** search  
**def** main():  
 **try**:  
 n = int(input(**"Введите количество записей -> "**))  
 **except** ValueError:  
 print(**"Некорректный ввод данных"**)  
 **return** d = {1: PersonClass,2: OrganizationClass,3: FriendClass,}  
 fake, obj\_list = Faker([**'ru\_RU'**]), []  
 **for** \_ **in** range(n):  
 r\_int = randint(1, 3)  
 d\_args = {1: (fake.name(), fake.address(), fake.phone\_number()),2: (fake.word(), fake.address(), fake.phone\_number(), fake.phone\_number(), fake.name()), 3: (fake.name(), fake.address(), fake.phone\_number(), fake.date(pattern=**'%d.%m.%Y'**)),}  
 obj\_list.append(d[r\_int](\*d\_args[r\_int]))  
 **for** obj **in** obj\_list:  
 print(obj.out\_info() + **"\n"**)  
 search(obj\_list)

## friend\_class.py

**from** .phonedictionary\_class **import** PhoneDictionaryClass  
**class** FriendClass(PhoneDictionaryClass):  
 **def** \_\_init\_\_(self, name, address, phone\_number, birth\_date):  
 super().\_\_init\_\_(name, address, phone\_number)  
 self.birth\_date = birth\_date  
 **def** out\_info(self):  
 **return "[Класс друг]\nФИО: "** + self.name + **"\nAдрес: "** + self.address + **"\nНомер телефона: "** + self.phone\_number + **"\nДата рождения: "** + self.birth\_date

## organization\_class.py

**from** .phonedictionary\_class **import** PhoneDictionaryClass  
**class** OrganizationClass(PhoneDictionaryClass):  
 **def** \_\_init\_\_(self, name, address, phone\_number, fax, contact\_person):  
 super().\_\_init\_\_(name, address, phone\_number)  
 self.fax = fax  
 self.contact\_person = contact\_person  
 **def** out\_info(self):  
 **return "[Класс организация]\nНазвание: "** + self.name + **"\nAдрес: "** + self.address + **"\nНомер телефона: "** + self.phone\_number + **"\nФакс: "** + self.fax + **"\nКонтактое лицо: "** + self.contact\_person  
 **def** search(self, input\_name):  
 **if** input\_name **in** self.contact\_person:  
 **return True  
 return False**

## person\_class.py

**from** .phonedictionary\_class **import** PhoneDictionaryClass  
**class** PersonClass(PhoneDictionaryClass):  
 **def** \_\_init\_\_(self, name, address, phone\_number):  
 super().\_\_init\_\_(name, address, phone\_number)  
 **def** out\_info(self):  
 **return "[Класс персона]\nФИО: "** + self.name + **"\nAдрес: "** + self.address + **"\nНомер телефона: "** + self.phone\_number

## phonedictionary\_class.py

**class** PhoneDictionaryClass:  
 **def** \_\_init\_\_(self, name, address, phone\_number):  
 self.name = name  
 self.address = address  
 self.phone\_number = phone\_number  
 **def** search(self, input\_name):  
 **if** input\_name **in** self.name:  
 **return True  
 return False  
 def** out\_info(self):  
 ...

## search\_module.py

**def** search(obj\_list):  
 **try**:  
 input\_name = input(**"Введите фамилию для поиска -> "**)  
 **except** ValueError:  
 print(**"Некорректный ввод данных"**)  
 **return** print(**"\n\*Абоненты системы с фамилией \""** + input\_name + **"\"\*"**)  
  
 search\_flag = **False  
 for** obj **in** obj\_list:  
 **if** obj.search(input\_name):  
 search\_flag = **True** print(obj.out\_info() + **"\n"**)  
  
 **if not** search\_flag:  
 print(**"Абоненты не найдены"**)

# Задание 13

## Условие

Создайте класс КЛИЕНТ с методами, позволяющими вывести на экран информацию о клиентах банка, а также определить соответствие клиента критерию поиска. Создайте дочерние классы ВКЛАДЧИК (фамилия, дата открытия вклада, размер вклада, процент по вкладу), КРЕДИТОР (фамилия, дата выдачи кредита, размер кредита, процент по кредиту, остаток долга), ОРГАНИЗАЦИЯ (название, дата открытия счета, номер счета, сумма на счету) со своими методами вывода информации на экран и определения соответствия дате (открытия вклада, выдаче кредита, открытия счета). Создайте список из п клиентов, выведите полную информацию из базы на экран, а также организуйте поиск клиентов, начавших сотрудничать с банком в заданную дату.

## \_\_init\_\_.py

**from** random **import** randint  
**from** faker **import** Faker  
**from** .clent\_class **import** ClentClass  
**from** .contributor\_class **import** ContributorClass  
**from** .creditor\_class **import** CreditorClass  
**from** .organization\_class **import** OrganizationClass  
**from** .search\_module **import** search  
**def** main():  
 **try**:  
 n = int(input(**"Введите количество клиентов -> "**))  
 **except** ValueError:  
 print(**"Некорректный ввод данных"**)  
 **return** d = {1: ClentClass,2: ContributorClass,3: CreditorClass,4: OrganizationClass,}  
 fake = Faker([**'ru\_RU'**])  
 bankclients\_list = []  
 **for** \_ **in** range(n):  
 r\_int = randint(1, 4)  
 d\_args = {1: (fake.name(), fake.date(pattern=**'%d.%m.%Y'**), randint(1, 1000000), randint(0, 100)),2: (fake.name(), fake.date(pattern=**'%d.%m.%Y'**), randint(1, 1000000), randint(0, 100)),3: (fake.name(), fake.date(pattern=**'%d.%m.%Y'**), randint(1, 1000000), randint(0, 100), randint(1, 1000000)),4: (fake.word(), fake.date(pattern=**'%d.%m.%Y'**), randint(1000000, 9999999), randint(1, 1000000)),}  
 bankclients\_list.append(d[r\_int](\*d\_args[r\_int]))  
 **for** cli **in** bankclients\_list:  
 print(cli.get\_info() + **"\n"**)  
 search(bankclients\_list)

## clent\_class.py

**class** ClentClass:  
 **def** \_\_init\_\_(self, name, open\_date, money\_count, percent):  
 self.name = name  
 self.open\_date = open\_date  
 self.money\_count = money\_count  
 self.percent = percent  
 **def** get\_info(self):  
 **return "[Информация о клиенте]\nФИО: "** + self.name + **"\nДата оформления операции: "** + self.open\_date + **"\nРазмер денежных единиц в операции: "** + str(  
 self.money\_count) + **" руб.\nПроцент: "** + str(self.percent) + **"%"  
 def** date\_calculation(self, input\_date):  
 **if** input\_date == self.open\_date:  
 **return True  
 return False**

## contributor\_class.py

**from** .clent\_class **import** ClentClass  
**class** ContributorClass(ClentClass):  
 **def** \_\_init\_\_(self, name, open\_date, money\_count, percent):  
 super().\_\_init\_\_(name, open\_date, money\_count, percent)  
 **def** get\_info(self):  
 **return "[Информация о вкладчике]\nФИО: "** + self.name + **"\nДата открытия вклада: "** + self.open\_date + **"\nРазмер вклада: "** + str(  
 self.money\_count) + **" руб.\nПроцент по вкладу: "** + str(self.percent) + **"%"**

## creditor\_class.py

**from** .clent\_class **import** ClentClass  
**class** CreditorClass(ClentClass):  
 **def** \_\_init\_\_(self, name, open\_date, money\_count, percent, balance\_owed):  
 super().\_\_init\_\_(name, open\_date, money\_count, percent)  
 self.balance\_owed = balance\_owed  
 **def** get\_info(self):  
 **return "[Информация о кредиторе]\nФИО: "** + self.name + **"\nДата выдачи кредита: "** + self.open\_date + **"\nРазмер кредита: "** + str(  
 self.money\_count) + **" руб.\nПроцент по кредиту: "** + str(self.percent) + **"%\nОстаток долга: "** + str(  
 self.balance\_owed) + **" руб."**

## organization\_class.py

**from** .clent\_class **import** ClentClass  
**class** OrganizationClass(ClentClass):  
 **def** \_\_init\_\_(self, name, open\_date, account\_number, balance):  
 self.name = name  
 self.open\_date = open\_date  
 self.account\_number = account\_number  
 self.balance = balance  
 **def** get\_info(self):  
 **return "[Информация об организации]\nНазвание: "** + self.name + **"\nДата открытия счёта: "** + self.open\_date + **"\nНомер счёта: "** + str(  
 self.account\_number) + **"\nСумма на счету: "** + str(self.balance) + **" руб."**

## search\_module.py

*# Поиск клиентов, начавших сотрудничать с банком в заданную дату.***def** search(bankclients\_list):  
 **try**:  
 input\_date = input(**"Введите дату для поиска клиентов в формате 01.01.2020 -> "**)  
 \_, \_, \_ = input\_date.split(**"."**)  
 **except** ValueError:  
 print(**"Некорректный ввод данных"**)  
 **return** search\_flag = **False** print(**"Найденные клиенты:"**)  
 **for** cli **in** bankclients\_list:  
 **if** cli.date\_calculation(input\_date) == **True**:  
 print(cli.get\_info() + **"\n"**)  
 search\_flag = **True  
 if not** search\_flag:  
 print(**"Клиенты не найдены"**)

# Задание 14

## Условие

Создайте класс ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ с методами, позволяющими вывести на экран информацию о программном обеспечении, а также определить соответствие возможности использования (на текущую дату). Создайте дочерние классы СВОБОДНОЕ (название, производитель), УСЛОВНО БЕСПЛАТНОЕ (название, производитель, дата установки, срок бесплатного использования), КОММЕРЧЕСКОЕ (название, производитель, цена, дата установки, срок использования) со своими методами вывода информации на экран и определения возможности использования на текущую дату. Создайте список из п видов программного обеспечения, выведите полную информацию из базы на экран, а также организуйте поиск программного обеспечения, которое допустимо использовать на текущую дату.

## \_\_init\_\_.py

**import** datetime  
**from** random **import** randint  
**from** faker **import** Faker  
**from** .software\_class **import** SoftwareClass  
**from** .freeware\_class **import** FreewareClass  
**from** .trial\_class **import** TrialClass  
**from** .commercial\_class **import** CommercialClass  
**from** .util\_module **import** search, display  
**def** main():  
 **try**:  
 n = int(input(**"Введите количество ПО -> "**))  
 **except** ValueError:  
 print(**"Некорректный ввод данных"**)  
 **return** d = {1: FreewareClass,2: TrialClass,3: CommercialClass,}  
 fake = Faker([**'ru\_RU'**])  
 software\_list = []  
 **for** \_ **in** range(n):  
 r\_int = randint(1, 3)  
 d\_args = {1: (fake.word(), fake.word() + **" "** + fake.word()),2: (fake.word(), fake.word() + **" "** + fake.word(), fake.date(pattern=**'%d.%m.%Y'**), randint(0, 1800)),3: (fake.word(), fake.word() + **" "** + fake.word(), fake.date(pattern=**'%d.%m.%Y'**), randint(0, 1800)),}  
 software\_list.append(d[r\_int](\*d\_args[r\_int]))  
 display(software\_list)  
 search(software\_list)

## commercial\_class.py

**import** datetime  
**from** .software\_class **import** SoftwareClass  
**class** CommercialClass(SoftwareClass):  
 **def** \_\_init\_\_(self, name, manufacturer, install\_date, days\_pay):  
 super().\_\_init\_\_(name, manufacturer)  
 self.install\_date = datetime.datetime.strptime(install\_date, **'%d.%m.%Y'**)  
 self.days\_pay = int(days\_pay)  
 **def** opportunity\_detector(self):  
 *# Если дата установки + кол-во оплаченных дней >= текущей даты, то ок* **if** self.install\_date + datetime.timedelta(days=self.days\_pay) >= self.date:  
 **return True  
 return False  
 def** \_\_days\_calculation(self):  
 days\_left = self.install\_date + datetime.timedelta(days=self.days\_pay) - self.date  
 **return** str(abs(days\_left.days))  
 **def** software\_info(self):  
 days\_left\_str = self.\_\_days\_calculation()  
 install\_date = self.install\_date.strftime(**"%d.%m.%Y"**)  
 **if** self.opportunity\_detector() == **True**:  
 status\_msg = **"Активировано, осталось "** + days\_left\_str + **" дня/дней"  
 else**:  
 status\_msg = **"Проблемы с активацией, просрочено на "** + days\_left\_str + **" дня/дней"  
 return "[Коммерческое ПО]\nНазвание: "** + self.name + **"\nПроизводитель: "** + self.manufacturer + **"\nДата установки: "** + install\_date + **"\nКоличество дней: "** + str(self.days\_pay) + **"\nСтатус: "** + status\_msg

## freeware\_class.py

**from** .software\_class **import** SoftwareClass  
**class** FreewareClass(SoftwareClass):  
 **def** \_\_init\_\_(self, name, manufacturer):  
 *# Можно и через super, но в нашей жизни очень мало разнообразия* self.name = name  
 self.manufacturer = manufacturer  
  
 **def** software\_info(self):  
 **return "[Свободное ПО]\nНазвание: "** + self.name + **"\nПроизводитель: "** + self.manufacturer  
  
 **def** opportunity\_detector(self):  
 *# Т.к. свободное ПО всегда можно использовать* **return True**

## software\_class.py

**import** datetime  
**class** SoftwareClass:  
 **def** \_\_init\_\_(self, name, manufacturer):  
 self.name = name  
 self.manufacturer = manufacturer  
 self.date = datetime.datetime.now()  
 **def** software\_info(self):  
 **return "[Родительский класс ПО]\nНазвание: "** + self.name + **"\nПроизводитель: "** + self.manufacturer  
 **def** opportunity\_detector(self):  
 *# Его все равно переопределят* ...

## trial\_class.py

**import** datetime  
**from** .software\_class **import** SoftwareClass  
**class** TrialClass(SoftwareClass):  
 **def** \_\_init\_\_(self, name, manufacturer, install\_date, days\_trial):  
 super().\_\_init\_\_(name, manufacturer)  
 self.install\_date = datetime.datetime.strptime(install\_date, **"%d.%m.%Y"**)  
 self.days\_trial = int(days\_trial)  
 **def** opportunity\_detector(self):  
 **if** self.install\_date + datetime.timedelta(days=self.days\_trial) >= self.date:  
 **return True  
 return False  
 def** \_\_days\_calculation(self):  
 days\_left = self.install\_date + datetime.timedelta(days=self.days\_trial) - self.date  
 **return** str(abs(days\_left.days))  
 **def** software\_info(self):  
 days\_left\_str = self.\_\_days\_calculation()  
 install\_date = self.install\_date.strftime(**"%d.%m.%Y"**)  
 **if** self.opportunity\_detector() == **True**:  
 status\_msg = **"Активировано, осталось "** + days\_left\_str + **" дня/дней"  
 else**:  
 status\_msg = **"Проблемы с активацией, просрочено на "** + days\_left\_str + **" дня/дней"  
 return "[Условно бесплатное ПО]\nНазвание: "** + self.name + **"\nПроизводитель: "** + self.manufacturer + **"\nДата установки: "** + install\_date + **"\nКоличество дней: "** + str(self.days\_trial) + **"\nСтатус: "** + status\_msg

## util\_module.py

**def** display(software\_list):  
 **for** software **in** software\_list:  
 print(software.software\_info() + **"\n"**)  
  
**def** search(software\_list):  
 search\_flag = **False** print(**"\n\*ПО, которое допустимо использовать на текущую дату\*"**)  
 **for** software **in** software\_list:  
 **if** software.opportunity\_detector():  
 print(software.software\_info() + **"\n"**)  
 search\_flag = **True  
 if not** search\_flag:  
 print(**"ПО не найдено"**)

# Задание 15

## Условие

Создайте класс ТРАНСПОРТ с методами, позволяющими вывести на экран информацию о транспортном средстве, а также определить, находится ли транспортное средство в пределах заданных координат. Создайте дочерние классы САМОЛЕТ (марка, максимальные скорость и высота, количество пассажиров, координаты), АВТОМОБИЛЬ (марка, номер, год выпуска, координаты), КОРАБЛЬ (название, координаты, скорость, количество пассажиров, порт приписки) со своими методами вывода информации на экран и определения присутствия транспортного средства в пределах заданных координат. Создайте список из п транспортных средств, выведите полную информацию из базы на экран, а также организуйте поиск транспортных средств, которые сейчас находятся в пределах заданных координат.

## \_\_init\_\_.py

**from** random **import** randint  
**from** faker **import** Faker  
**from** .ship\_class **import** ShipClass  
**from** .car\_class **import** CarClass  
**from** .airplane\_class **import** AirplaneClass  
**from** .transport\_class **import** TransportClass  
**from** .util\_module **import** display, search  
**def** main():  
 **try**:  
 n = int(input(**"Введите кол-во транспортных средств -> "**))  
 **except** ValueError:  
 print(**"Некорректный ввод данных"**)  
 **return** d = {1: TransportClass,2: AirplaneClass,3: CarClass,4: ShipClass,}  
 fake = Faker([**'ru\_RU'**])  
 transport\_list = []  
 **for** \_ **in** range(n):  
 r\_int = randint(1, 4)  
 d\_args = {1: (fake.word(), [randint(1, 100), randint(1, 100)]),2: (fake.word(), [randint(1, 100), randint(1, 100)], randint(1, 1000), randint(50, 300), randint(1000, 6000)),3: (fake.word(), [randint(1, 100), randint(1, 100)], randint(1, 1000), randint(1960, 2020)),4: (fake.word(), [randint(1, 100), randint(1, 100)], randint(1, 1000), randint(50, 300), fake.word()),}  
 transport\_list.append(d[r\_int](\*d\_args[r\_int]))  
 display(transport\_list)  
 search(transport\_list)

## airplane\_class.py

**from** .transport\_class **import** TransportClass  
**class** AirplaneClass(TransportClass):  
 **def** \_\_init\_\_(self, name, coords, passengers\_number, max\_speed, max\_height):  
 super().\_\_init\_\_(name, coords)  
 self.passengers\_number = passengers\_number  
 self.max\_speed = max\_speed  
 self.max\_height = max\_height  
 **def** info(self):  
 **return "[Класс самолёт]\nМодель: "** + self.name + **"\nКоординаты: "** + self.coords\_formater\_str() + **"\nКол-во пассажиров: "** + str(  
 self.passengers\_number) + **"\nМакс. скорость: "** + str(self.max\_speed) + **"\nМакс. высота: "** + str(  
 self.max\_height)

## car\_class.py

**from** .transport\_class **import** TransportClass  
**class** CarClass(TransportClass):  
 **def** \_\_init\_\_(self, name, coords, number, year):  
 super().\_\_init\_\_(name, coords)  
 self.number = number  
 self.year = year  
 **def** info(self):  
 **return "[Класс автомобиль]\nМодель: "** + self.name + **"\nКоординаты: "** + self.coords\_formater\_str() + **"\nНомер: "** + str(  
 self.number) + **"\nГод выпуска: "** + str(self.year)

## ship\_class.py

**from** .transport\_class **import** TransportClass  
**class** ShipClass(TransportClass):  
 **def** \_\_init\_\_(self, name, coords, passengers\_number, max\_speed, destination\_name):  
 super().\_\_init\_\_(name, coords)  
 self.passengers\_number = passengers\_number  
 self.max\_speed = max\_speed  
 self.destination\_name = destination\_name  
 **def** info(self):  
 **return "[Класс корабль]\nМодель: "** + self.name + **"\nКоординаты: "** + self.coords\_formater\_str() + **"\nКол-во пассажиров: "** + str(  
 self.passengers\_number) + **"\nМакс. скорость: "** + str(  
 self.max\_speed) + **"\nПорт приписки: "** + self.destination\_name

## transport\_class.py

**class** TransportClass():  
 **def** \_\_init\_\_(self, name, coords):  
 self.name = name  
 self.coords = coords  
 **def** coords\_detector(self, coord\_list):  
 y\_list, x\_list = coord\_list  
 y1, y2 = y\_list  
 x1, x2 = x\_list  
 locale\_x, locale\_y = self.coords  
 **if** x1 < locale\_x < x2 **and** y1 < locale\_y < y2:  
 **return True  
 return False  
 def** coords\_formater\_str(self):  
 x, y = self.coords  
 **return "["** + str(x) + **", "** + str(y) + **"]"  
 def** info(self):  
 **return "[Родительский класс транспорт]\nМодель: "** + self.name + **"\nКоординаты: "** + self.coords\_formater\_str()

## util\_module.py

**def** display(transport\_list):  
 **for** transport **in** transport\_list:  
 print(transport.info() + **"\n"**)  
**def** search(transport\_list):  
 search\_flag = **False  
 try**:  
 x1\_input = int(input(**"Введите координату x1 -> "**))  
 y1\_input = int(input(**"Ввдетие координату y1 -> "**))  
 x2\_input = int(input(**"Введите координату x2 -> "**))  
 y2\_input = int(input(**"Ввдетие координату y2 -> "**))  
 **if** x1\_input < x2\_input: x2\_input, x1\_input = x1\_input, x2\_input  
 **if** y1\_input < y2\_input: y2\_input, y1\_input = y1\_input, y2\_input  
 **except** ValueError:  
 print(**"Некорректный ввод данных"**)  
 **return** l = ([x1\_input, y1\_input], [x2\_input, y2\_input])  
 print(**"\n\*ТС, которые сейчас находятся в пределах заданных координат\*"**)  
 **for** transport **in** transport\_list:  
 **if** transport.coords\_detector(l):  
 print(transport.info() + **"\n"**)  
 search\_flag = **True  
 if not** search\_flag: print(**"ТС не найдены"**)

# Задание 16

## Условие

Создайте класс ИГРУШКА с методами, позволяющими вывести на экран информацию о товаре, а также определить соответствие игрушки критерию поиска. Создайте дочерние классы КУБИК (цвет, цена, материал, размер ребра), МЯЧ (цена, цвет, диаметр, материал), МАШИНКА (название, цена, производитель, цвет) со своими методами вывода информации на экран и определения соответствия заданному цвету. Создайте список из п игрушек, выведите полную информацию из базы на экран, а также организуйте поиск игрушек заданного цвета.

## \_\_init\_\_.py

**from** random **import** randint  
**from** faker **import** Faker  
**from** .random\_color **import** random\_color  
**from** .car\_class **import** CarClass  
**from** .ball\_class **import** BallClass  
**from** .cube\_class **import** CubeClass  
**from** .toy\_class **import** ToyClass  
**from** .util\_module **import** search, display  
**def** main():  
 **try**:  
 n = int(input(**"Введите количество игрушек -> "**))  
 **except** ValueError:  
 print(**"Некорректный ввод данных"**)  
 **return** d = {1: ToyClass,2: CubeClass,3: BallClass,4: CarClass,}  
 fake = Faker([**'ru\_RU'**])  
 toy\_list = []  
 **for** \_ **in** range(n):  
 r\_int = randint(1, 4)  
 d\_args = {1: (random\_color(), randint(1, 1000), **"пластик"**, randint(1, 20)),2: (random\_color(), randint(1, 1000), **"пластик"**, randint(1, 20)),3: (random\_color(), randint(1, 1000), **"пластик"**, randint(1, 20)),4: (random\_color(), randint(1, 1000), fake.word(), fake.word() + **" "** + fake.word()),}  
 toy\_list.append(d[r\_int](\*d\_args[r\_int]))  
 display(toy\_list)  
 search(toy\_list)

## ball\_class.py

**from** .toy\_class **import** ToyClass  
**class** BallClass(ToyClass):  
 **def** \_\_init\_\_(self, color, price, material, size):  
 super().\_\_init\_\_(color, price, material, size)  
 **def** toy\_info(self):  
 **return "[Мяч]\nЦвет: "** + self.color + **"\nЦена: "** + str(self.price) + **" руб.\nМатериал: "** + self.material + **"\nДиаметр: "** + str(self.size) + **" см."**

## car\_class.py

**from** .toy\_class **import** ToyClass  
**class** CarClass(ToyClass):  
 **def** \_\_init\_\_(self, color, price, name, manufacturer):  
 self.color = color  
 self.price = price  
 self.name = name  
 self.manufacturer = manufacturer  
 **def** toy\_info(self):  
 **return "[Машинка]\nЦвет: "** + self.color + **"\nЦена: "** + str(self.price) + **" руб.\nНазвание: "** + self.name + **"\nПроизводитель: "** + self.manufacturer

## cube\_class.py

**from** .toy\_class **import** ToyClass  
**class** CubeClass(ToyClass):  
 **def** \_\_init\_\_(self, color, price, material, size):  
 super().\_\_init\_\_(color, price, material, size)  
 **def** toy\_info(self):  
 **return "[Кубик]\nЦвет: "** + self.color + **"\nЦена: "** + str(self.price) + **" руб.\nМатериал: "** + self.material + **"\nРазмер ребра: "** + str(self.size) + **" см."**

## toy\_class.py

**class** ToyClass:  
 **def** \_\_init\_\_(self, color, price, material, size):  
 self.color = color  
 self.price = price  
 self.material = material  
 self.size = size  
 **def** toy\_info(self):  
 **return "[Базовый класс игрушка]\nЦвет: "** + self.color + **"\nЦена: "** + str(self.price) + **" руб.\nМатериал: "** + self.material + **"\nРазмер: "** + str(self.size) + **" см."  
 def** color\_detector(self, color\_input):  
 **if** color\_input == self.color:  
 **return True  
 return False**

## util\_module.py

**def** display(toy\_list):  
 **for** toy **in** toy\_list:  
 print(toy.toy\_info() + **"\n"**)  
**def** search(toy\_list):  
 *# организуйте поиск игрушек заданного цвета.* **try**:  
 color\_input = input(**"Введите цвет для поиска игрушек -> "**)  
 **except** ValueError:  
 print(**"Некорректный ввод данных"**)  
 **return** search\_flag = **False** print(**"Игрушки c фильтрацией по цвету"**)  
 **for** toy **in** toy\_list:  
 **if** toy.color\_detector(color\_input):  
 print(toy.toy\_info() + **"\n"**)  
 search\_flag = **True  
 if not** search\_flag:  
 print(**"Игрушки не найдены"**)

# Задание 17

## Условие

Создайте класс ТЕЛО с методами вычисления площади поверхности и объема, а также методом, выводящим информацию о фигуре на экран. Создайте дочерние классы ПАРАЛЛЕЛЕПИПЕД, ШАР, ПИРАМИДА со своими методами вычисления площади и объема. Создайте список п фигур и выведите полную информацию о фигурах на экран.

## \_\_init\_\_.py

**from** math **import** pi  
**from** random **import** randint  
**from** .pyramid\_class **import** PyramidClass  
**from** .ball\_class **import** BallClass  
**from** .parallelepiped\_class **import** ParallelepipedClass  
**from** .body\_class **import** BodyClass  
**def** main():  
 **try**:  
 n = int(input(**"Введите количество фигур -> "**))  
 **except** ValueError:  
 print(**"Некорректный ввод данных"**)  
 **return** d = {1: ParallelepipedClass,2: BallClass,3: PyramidClass,}  
 figures\_list = []  
 **for** \_ **in** range(n):  
 d\_args = {1: [randint(1, 1000), randint(1, 1000), randint(1, 1000)],2: [randint(1, 1000)],3: [randint(1, 1000), randint(1, 1000), randint(1, 1000)],}  
 r\_number = randint(1, 3)  
 figures\_list.append(d[r\_number](\*d\_args[r\_number]))  
 **for** figure **in** figures\_list:  
 figure.info(type(figure).\_\_name\_\_)

## ball\_class.py

**from** .body\_class **import** BodyClass  
**from** math **import** pi  
**class** BallClass(BodyClass):  
 **def** \_\_init\_\_(self, r):  
 self.r = r  
 self.volume\_calculation()  
 self.surface\_area\_calculation()  
 **def** surface\_area\_calculation(self):  
 self.surface\_area = 4 \* pi \* pow(self.r, 2)  
 **def** volume\_calculation(self):  
 self.volume = (4 / 3) \* pi \* pow(self.r, 3)

## body\_class.py

**from** abc **import** ABCMeta, abstractmethod  
**class** BodyClass:  
 \_\_metaclass\_\_ = ABCMeta  
 @abstractmethod  
 **def** surface\_area\_calculation(self):  
 self.surface\_area = 0  
 @abstractmethod  
 **def** volume\_calculation(self):  
 self.volume = 0  
 **def** info(self, lclass=**None**):  
 a = self.surface\_area  
 v = self.volume  
 **if** type(self.surface\_area) == float **or** type(self.volume) == float:  
 a = str(round(self.surface\_area, 2))  
 v = str(round(self.volume, 2))  
 print(**"\nВызов от {}\nПлощадь поверхности фигуры: {}\nОбъем фигуры: {}"**.format(lclass, a, v))

## parallelepiped\_class.py

**from** .body\_class **import** BodyClass  
**class** ParallelepipedClass(BodyClass):  
 **def** \_\_init\_\_(self, a, b, c):  
 *# a,b,c - ребра параллелепипеда* self.a = a  
 self.b = b  
 self.c = c  
 self.volume\_calculation()  
 self.surface\_area\_calculation()  
 **def** surface\_area\_calculation(self):  
 *# S = 2(ab + bc + ac)* a = self.a  
 b = self.b  
 c = self.c  
 self.surface\_area = 2 \* (a \* b + b \* c + a \* c)  
 **def** volume\_calculation(self):  
 self.volume = self.a \* self.b \* self.c

## pyramid\_class.py

**from** .body\_class **import** BodyClass  
**class** PyramidClass(BodyClass):  
 **def** \_\_init\_\_(self, S\_main, S\_back, h):  
 *# S\_main - площадь основания пирамиды  
 # S\_back - площадь боковой поверхности  
 # h - высота пирамиды* self.S\_main = S\_main  
 self.S\_back = S\_back  
 self.h = h  
 self.volume\_calculation()  
 self.surface\_area\_calculation()  
 **def** surface\_area\_calculation(self):  
 self.surface\_area = self.S\_main + 4 \* self.S\_back  
 **def** volume\_calculation(self):  
 self.volume = (1 / 3) \* self.S\_main \* self.h

# Задание 18

## Условие

Создайте класс УРАВНЕНИЕ с методами вычисления корня уравнения и вывода результата на экран. Создайте дочерние классы ЛИНЕЙНОЕ, КВАДРАТНОЕ со своими методами вычисления корней и вывода на экран. Создайте список n уравнений и выведите полную информацию об уравнениях на экран.

## \_\_init\_\_.py

**import** math  
**from** random **import** randint  
**from** .quadraticequation **import** QuadraticEquation  
**from** .linearequation **import** LinearEquation  
**def** main():  
 **try**:  
 n = int(input(**"Введите количество уравнений -> "**))  
 **except** ValueError:  
 print(**"Некорректный ввод данных"**)  
 **return** d, main\_list = {1: LinearEquation, 2: QuadraticEquation}, []  
 **for** \_ **in** range(n):  
 d\_args = {1: [randint(-1000, 1000), randint(-1000, 1000), randint(-1000, 1000)],2: [str(randint(-1000, 1000)) + **"x^2+"** + str(randint(-1000, 1000)) + **"x+"** + str(randint(-1000, 1000)) + **"=0"**],}  
 r\_number = randint(1, 2)  
 main\_list.append(d[r\_number](\*d\_args[r\_number]))  
 **for** e **in** main\_list: print(e.info() + **"\n"**)

## equationclass.py

**class** EquationClass:  
 **def** \_\_init\_\_(self):  
 **pass  
 def** calculation(self):  
 ...  
 **def** info(self):  
 ...

## linearequation.py

**from** .equationclass **import** EquationClass  
**class** LinearEquation(EquationClass):  
 **def** \_\_init\_\_(self, a, b, c):  
 self.a = a  
 self.b = b  
 self.c = c  
 self.calculation()  
 **def** calculation(self):  
 **try**:  
 self.result = (self.c - self.b) / self.a  
 **except** ZeroDivisionError:  
 self.result = 0  
  
 **def** info(self):  
 a = self.a  
 b = self.b  
 c = self.c  
 equation\_str = str(a) + **"x+"** + str(b) + **"="** + str(c)  
 **return "\*Информация о простом линейном уравнении\*\nОбщий вид: "** + equation\_str + **"\nКоэффициент a = "** + str(  
 a) + **"\nКоэффициент b = "** + str(b) + **"\nКоэффициент c = "** + str(c) + **"\nОтвет:\n"** + str(self.result)

## parse\_exp.py

**from** .parseexp\_koff\_a **import** get **as** get\_a  
**from** .parseexp\_koff\_b **import** get **as** get\_b  
**from** .parseexp\_koff\_c **import** get **as** get\_c  
**def** parse\_exp(input\_str):  
 index\_a = get\_a(input\_str)  
 index\_b = get\_b(input\_str)  
 index\_c = get\_c(input\_str)  
 **return** list(map(float,[index\_a,index\_b,index\_c]))

## parseexp\_koff\_a.py

**from** .utils **import** is\_digital  
**def** get(input\_str):  
 symbols = 1  
 return\_flag = **False  
 while** return\_flag == **False**:  
 index\_a = input\_str[:symbols]  
 **if** is\_digital(index\_a) == **False and** symbols != 1:  
 return\_flag = **True** symbols += 1  
 index\_a = input\_str[:symbols - 2]  
 **return** index\_a

## parseexp\_koff\_b.py

**from** .utils **import** is\_digital  
**def** get(input\_str):  
 symbols = input\_str.rindex(**"x"**)  
 buf\_index\_b = input\_str[:symbols]  
 symbols = 1  
 return\_flag = **False  
 while** return\_flag == **False**:  
 index\_b = buf\_index\_b[symbols:]  
 **if** is\_digital(index\_b) == **True**:  
 return\_flag = **True** symbols += 1  
 **return** index\_b

## parseexp\_koff\_c.py

**from** .utils **import** is\_digital  
**def** get(input\_str):  
 symbols = input\_str.rindex(**"="**)  
 buf\_index\_c = input\_str[:symbols]  
 symbols = 1  
 return\_flag = **False  
 while** return\_flag == **False**:  
 index\_c = buf\_index\_c[symbols:]  
 **if** is\_digital(index\_c) == **True**:  
 return\_flag = **True** symbols += 1  
 **return** index\_c

## quadraticequation.py

**import** math  
**from** .equationclass **import** EquationClass  
**from** .parse\_exp **import** parse\_exp  
**from** .quadraticequation\_calculation **import** calculation  
**class** QuadraticEquation(EquationClass):  
 **def** \_\_init\_\_(self, input\_str):  
 self.equation\_results\_list = []  
 self.input\_str = input\_str  
 self.index\_a, self.index\_b, self.index\_c = parse\_exp(input\_str)  
 self.equation\_results\_list = calculation(self.index\_a, self.index\_b, self.index\_c)  
 **def** info(self):  
 locale\_list = self.equation\_results\_list  
 **if** len(locale\_list) == 1:  
 result\_msg = **""**.join(locale\_list)  
 **elif** len(locale\_list) == 2:  
 result\_msg = **"Корень А1 = "** + str(locale\_list[0]) + **"\nКорень А2 = "** + str(locale\_list[1])  
 **else**: result\_msg = **"Ошибка вычисления"  
 return "\*Информация о квадратном уравнении\*\nОбщий вид: "** + self.input\_str + **"\nКоэффициент a = "** + str(  
 self.index\_a) + **"\nКоэффициент b = "** + str(self.index\_b) + **"\nКоэффициент c = "** + str(  
 self.index\_c) + **"\nОтвет:\n"** + result\_msg

## quadraticequation\_calculation.py

**import** math  
**from** .equationclass **import** EquationClass  
**from** .parse\_exp **import** parse\_exp  
**from** .quadraticequation\_calculation **import** calculation  
**class** QuadraticEquation(EquationClass):  
 **def** \_\_init\_\_(self, input\_str):  
 self.equation\_results\_list = []  
 self.input\_str = input\_str  
 self.index\_a, self.index\_b, self.index\_c = parse\_exp(input\_str)  
 self.equation\_results\_list = calculation(self.index\_a, self.index\_b, self.index\_c)  
 **def** info(self):  
 locale\_list = self.equation\_results\_list  
 **if** len(locale\_list) == 1:  
 result\_msg = **""**.join(locale\_list)  
 **elif** len(locale\_list) == 2:  
 result\_msg = **"Корень А1 = "** + str(locale\_list[0]) + **"\nКорень А2 = "** + str(locale\_list[1])  
 **else**: result\_msg = **"Ошибка вычисления"  
 return "\*Информация о квадратном уравнении\*\nОбщий вид: "** + self.input\_str + **"\nКоэффициент a = "** + str(  
 self.index\_a) + **"\nКоэффициент b = "** + str(self.index\_b) + **"\nКоэффициент c = "** + str(  
 self.index\_c) + **"\nОтвет:\n"** + result\_msg

## utils.py

**def** is\_digital(number):  
 **try**:  
 float(number)  
 **return True  
 except**:  
 **return False**

# Задание 19

## Условие

Создайте класс ВАЛЮТА с методами перевода денежной суммы в рубли и вывода на экран. Создайте дочерние классы ДОЛЛАР, ЕВРО со своими методами перевода и вывода на экран. Создайте список п валютных денежных сумм и выведите полную информацию о них на экран.

## \_\_init\_\_.py

**from** random **import** randint  
**from** .currency\_class **import** CurrencyClass  
**from** .eur\_class **import** EURClass  
**from** .usd\_class **import** USDClass  
**def** main():  
 **try**:  
 n = int(input(**"Введите кол-во валютных денежных сумм -> "**))  
 **except** ValueError:  
 print(**"Некорректный ввод данных"**)  
 **return** d = {1: EURClass,2: USDClass,}  
 currency\_list = [d[randint(1, 2)](randint(1, 1000)) **for** \_ **in** range(n)]  
 **for** c **in** currency\_list: print(c.info() + **"\n"**)

## currency\_class.py

**class** CurrencyClass:  
 eur\_currency = 85.46 *#курс на 18.03.2020, хах* usd\_currency = 77.87   
 **def** \_\_init\_\_(self, balance, exchange):  
 self.balance = balance  
 self.exchange = exchange  
 **def** converter(self):  
 **return** round(self.balance \* self.exchange, 2)  
 **def** info(self):  
 **return "[Родительский класс валюты]\nКол-во: "** + str(self.balance) + **"\nКурс обмена: "** + str(self.exchange) + **"\nСконвертированная валюта: "** + str(self.converter())

## eur\_class.py

**from** .currency\_class **import** CurrencyClass  
**class** EURClass(CurrencyClass):  
 **def** \_\_init\_\_(self, balance):  
 self.currency = super().eur\_currency  
 self.balance = balance  
 **def** converter(self):  
 **return** round(self.balance \* self.currency, 2)  
 **def** info(self):  
 **return "[EUR]\nКол-во EUR: "** + str(self.balance) + **"\nКурс обмена: "** + str(self.currency) + **"\nСконвертированная валюта в RUB: "** + str(self.converter())

## usd\_class.py

**from** .currency\_class **import** CurrencyClass  
**class** USDClass(CurrencyClass):  
 **def** \_\_init\_\_(self, balance):  
 self.currency = super().usd\_currency  
 self.balance = balance  
 **def** converter(self):  
 **return** round(self.balance \* self.currency, 2)  
 **def** info(self):  
 **return "[USD]\nКол-во USD: "** + str(self.balance) + **"\nКурс обмена: "** + str(self.currency) + **"\nСконвертированная валюта в RUB: "** + str(self.converter())

# Задание 20

## Условие

Создайте класс ПРОГРЕССИЯ с методами вычисления ]-го элемента прогрессии, ее суммы и методом, выводящим сумму на экран. Создайте дочерние классы: АРИФМЕТИЧЕСКАЯ, ГЕОМЕТРИЧЕСКАЯ со своими методами вычисления. Создайте список п прогрессий и выведите сумму каждой из них экран.

## \_\_init\_\_.py

**from** random **import** randint  
**from** .progression\_class **import** ProgressionClass  
**from** .arithmetic\_class **import** ArithmeticClass  
**from** .geometric\_class **import** GeometricClass  
**def** main():  
 **try**:  
 n = int(input(**"Введите кол-во прогрессий для генерации -> "**))  
 **except** ValueError:  
 print(**"Некорректный ввод данных"**)  
 **return** d = {1: ArithmeticClass, 2: GeometricClass,}  
 progression\_list = []  
 **for** \_ **in** range(n):  
 d\_args = {1: [randint(-1000, 1000), randint(-1000, 1000), randint(1, 10)],2: [randint(-1000, 1000), randint(-1000, 1000)],}  
 r\_number = randint(1, 2)  
 progression\_list.append(d[r\_number](\*d\_args[r\_number]))  
 **for** e **in** progression\_list:  
 *# print(e.get\_sum())* print(e.info() + **"\n"**)

## arithmetic\_class.py

**from** .progression\_class **import** ProgressionClass  
**class** ArithmeticClass(ProgressionClass):  
 **def** \_\_init\_\_(self, a1, a2, n):  
 self.a1 = a1  
 self.a2 = a2  
 self.n = n  
 self.calculate()  
 **def** calculate(self):  
 self.s = (self.a1 + self.a2) \* self.n / 2  
 **def** info(self):  
 **return "[Арифметическая прогрессия]\n1-й элемент: "** + str(self.a1) + **"\n2-й элемент: "** + str(self.a2) + **"\nШаг: "** + str(self.n) + **"\nСумма: "** + str(self.s)

## geometric\_class.py

**from** .progression\_class **import** ProgressionClass  
**class** GeometricClass(ProgressionClass):  
 **def** \_\_init\_\_(self, b1, q):  
 self.b1 = b1  
 self.q = q  
 self.calculate()  
 **def** calculate(self):  
 **try**:  
 self.s = self.b1 / (1 - self.q)  
 **except** ZeroDivisionError:  
 self.s = -1  
 **def** info(self):  
 **return "[Геометрическая прогрессия]\nЗнаменатель прогрессии: "** + str(self.q) + **"\n1-й элемент: "** + str(self.b1) + **"\nСумма: "** + str(self.s)

## progression\_class.py

**class** ProgressionClass:  
 **def** \_\_init\_\_(self):  
 self.s = 0  
 **def** get\_sum(self):  
 **return** self.s  
 **def** calculate(self):  
 ...  
 **def** info(self):  
 ...