Table of Contents

[Задания 1.1. – 1.4 2](#_Toc33563330)

[Задание 1.5. 5](#_Toc33563331)

[Задание 2.1. 8](#_Toc33563332)

[Задание 2.2. 11](#_Toc33563333)

[Задание 2.3. 13](#_Toc33563334)

[Задание 2.4. 16](#_Toc33563335)

[Задание 2.5. 19](#_Toc33563336)

[Задание 2.6. 22](#_Toc33563337)

[Задание 2.7. 25](#_Toc33563338)

[Задание 2.8. 28](#_Toc33563339)

[Задание 2.9. 31](#_Toc33563340)

[Задание 2.10. 34](#_Toc33563341)

[Задание 2.11. 37](#_Toc33563342)

[Задание 2.12. 39](#_Toc33563343)

[Задание 2.13. 41](#_Toc33563344)

[Задание 2.14. 44](#_Toc33563345)

[Задание 2.15. 46](#_Toc33563346)

Задания 1.1. – 1.4**import** math  
**from** random **import** randrange  
  
  
**class** Task1Class:  
 *"""  
 Задача 1. Создайте класс с методом класса, в котором определялась бы сумма двух целых чисел.  
 """* @staticmethod  
 **def** sum(a, b):  
 **return** a + b  
  
  
**class** Task2Class:  
 *"""  
 Задача 2. 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.  
 """* **def** \_\_init\_\_(self, a, b):  
 self.a = a  
 self.b = b  
  
 **def** sum(self):  
 **return** self.a + self.b  
  
  
**class** Task3Class:  
 *"""  
 Задача 3. 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.  
 """* **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):  
 *"""  
 Фильтрация на числа  
 """* **try**:  
 **return** int(e)  
 **except** ValueError:  
 **return** e  
  
 **def** user\_input(self):  
 *"""  
 Ручное заполнение последовательности  
 """* **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):  
 *"""  
 Автоматическое заполнение последовательности  
 """* self.main\_list = [randrange(-10, 10) **for** \_ **in** range(self.list\_len)]  
  
 **def** out\_list(self):  
 *"""  
 Вывод полученной последовательности  
 """* print(**"Полученная последовательность:"**)  
 print(self.main\_list)  
  
  
**class** Task4Class:  
 *"""  
 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.  
 """* **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  
  
  
**def** main():  
 print(**"Задание 1:"**, Task1Class.sum(3, 4))  
 obj = Task2Class(2, 4)  
 print(**"Задание 2:"**, obj.sum())  
  
 print(**"Задание 3:"**)  
 Task3Class()  
  
 print(**"\nЗадание 4:"**)  
 **try**:  
 x = float(input(**"Введите x -> "**))  
 **except** ValueError:  
 print(**"Проблема ввода данных!"**)  
 **return** obj = Task4Class(x)  
 print(**"Результат: "** + str(obj.result))  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()

Задание 1.5.

*"""  
Задача 5. 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.  
"""***import** random  
**from** datetime **import** datetime  
  
**from** faker **import** Faker  
  
  
**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  
  
  
**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)  
  
  
**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)  
  
  
**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)  
  
  
**def** main():  
 **try**:  
 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)  
  
 **except** ValueError:  
 print(**"Некорректный ввод данных !"**)  
 **return** 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"**,  
 ]  
  
 jobs\_list = [  
 **"Старший научный сотрудник"**,  
 **"Старший преподаватель"**,  
 **"Профессор"**,  
 **"Преподаватель"**,  
 **"Научный сотрудник"**,  
 **"Младший научный сотрудник"**,  
 **"Доцент"**,  
 **"Докторант"**,  
 **"Главный научный сотрудник"**,  
 **"Ведущий научный сотрудник"**,  
 **"Ассистент"** ]  
  
 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(), random.choice(ways\_list)],  
 3: [fake.name(), fake.date(), random.choice(ways\_list), random.randint(1, 4)],  
 4: [fake.name(), fake.date(), random.choice(ways\_list), random.choice(jobs\_list), 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]"**)  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()

Задание 2.1.

*"""  
Задача 1. 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.  
"""***from** abc **import** ABCMeta, abstractmethod  
**from** math **import** pi, sqrt  
**from** random **import** randint  
  
  
**class** FigureClass:  
 *"""  
 Абстрактный класс фигуры  
 """* \_\_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))  
  
  
**class** RectangleClass(FigureClass):  
 *"""  
 Класс прямоугольника  
 """* **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  
  
  
**class** CircleClass(FigureClass):  
 *"""  
 Класс круга  
 """* **def** \_\_init\_\_(self, r):  
 self.r = r  
 self.perimeter\_calculation()  
 self.area\_calculation()  
  
 **def** area\_calculation(self):  
 self.area = pi \* self.r \*\* 2  
  
 **def** perimeter\_calculation(self):  
 self.perimeter = 2 \* pi \* self.r  
  
  
**class** TriangleClass(FigureClass):  
 *"""  
 Класс треугольника  
 """* **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):  
 *# Полупериметр* p = (self.a + self.b + self.c) / 2  
 s = sqrt(p \* (p - self.a) \* (p - self.b) \* (p - self.c))  
 self.area = s  
  
 **def** perimeter\_calculation(self):  
 self.perimeter = self.a + self.b + self.c  
  
  
**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\_\_)  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()

Задание 2.2.

*"""  
Задача 2. 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.   
"""***from** faker **import** Faker  
  
  
**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)  
  
  
**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)  
  
  
**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)  
  
  
**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)  
  
  
**def** main():  
 **try**:  
 n = int(input(**"Введите количество изданий -> "**))  
 **except** ValueError:  
 print(**"Некорректный ввод данных"**)  
 **return** fake = Faker([**'ru\_RU'**])  
 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 -> "**)  
 search\_flag = **False  
  
 for** pub **in** all\_publications\_list:  
 **if** search\_name **in** pub.author\_name:  
 print(**"[Издание найдено]"**)  
 pub.info()  
 search\_flag = **True  
  
 if** search\_flag == **False**:  
 print(**"Изданий не найдено"**)  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()

Задание 2.3.

*"""  
Задача 3. 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.  
"""***import** math  
**from** abc **import** ABCMeta, abstractmethod  
**from** random **import** randint  
  
  
**class** TriangleClass:  
 *"""  
 Базовый класс треугольника  
 """* **def** \_\_init\_\_(self, a\_side, b\_side, angle):  
 self.a\_side = a\_side  
 self.b\_side = b\_side  
 self.angle = angle  
  
 self.perimeter\_calculation()  
 self.area\_calculation()  
  
 **def** perimeter\_calculation(self):  
 a = self.a\_side  
 b = self.b\_side  
 c = math.sqrt(b \*\* 2 + a \*\* 2 - 2 \* b \* a \* math.cos(math.radians(self.angle)))  
  
 self.c\_side = c  
 self.perimeter = a + b + c  
  
 **def** area\_calculation(self):  
 *# Вычисление полупериметра* a = self.a\_side  
 b = self.b\_side  
 c = self.c\_side  
  
 p = (a + b + c) / 2  
 s = math.sqrt(p \* (p - a) \* (p - b) \* (p - c))  
 self.area = s  
  
 **def** info(self, cl\_name):  
 a = self.area  
 p = self.perimeter  
 p = str(round(self.perimeter, 2))  
 *# Можно сделать вот так вообще, потому что периметр не влияет вовсе* **if** type(self.area) == float:  
 a = str(round(self.area, 2))  
 print(**"\nВызов от {}\nПлощадь фигуры: {}\nПериметр фигуры: {}"**.format(cl\_name, a, p))  
  
  
**class** TectangularTriangleClass(TriangleClass):  
 *"""  
 Класс прямоугольного треугольника  
 """* **def** \_\_init\_\_(self, a\_side, b\_side, angle=90):  
 *# Т.к. прямой угол* 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  
  
  
**class** IsoscelesTriangleClass(TriangleClass):  
 *"""  
 Класс равнобедренного треугольника  
 """* **def** \_\_init\_\_(self, a\_side, b\_side, angle):  
 *# angle необходим для того, чтоб в main снова не вводить разные аргументы для вызова разных классов* super().\_\_init\_\_(a\_side, b\_side, angle)  
 *# Основание* self.a\_side = a\_side  
 *# Боковая сторона* self.b\_side = b\_side  
  
 **def** perimeter\_calculation(self):  
 *# Чтоб не учитывать c\_side* self.perimeter = self.b\_side \* 2 + self.a\_side  
  
 **def** area\_calculation(self):  
 a = self.a\_side  
 b = self.b\_side  
  
 **try**:  
 *# Половинка основания* part\_a = a / 2  
 *# Вычисляем высоту* h = math.sqrt(pow(b, 2) - pow(part\_a, 2))  
 *# По формуле площади равнобедренного треугольника* self.area = (h \* a) / 2  
 **except** ValueError:  
 self.area = **"Ошибка вычисления"  
  
  
class** EquilateralTriangleClass(TriangleClass):  
 *"""  
 Класс равностороннего треугольника  
 """* **def** \_\_init\_\_(self, a\_side, b\_side, angle):  
 *# angle и b\_side необходимы для того, чтоб в main снова не вводить разные аргументы для вызова разных классов* super().\_\_init\_\_(a\_side, b\_side, angle)  
 *# Единсттвенная труъ сторона* self.a\_side = a\_side  
  
 **def** perimeter\_calculation(self):  
 self.perimeter = self.a\_side \* 3  
  
 **def** area\_calculation(self):  
  
 **try**:  
 *# Высота* 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"  
  
  
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\_\_)  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()

Задание 2.4.

*"""  
Задача 4. 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.  
"""***from** random **import** randint  
  
**from** random\_auto **import** random\_auto, random\_number  
  
  
**class** TransportClass:  
 *"""  
 Базовый класс транспорт  
 """* **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  
  
  
**class** CarClass(TransportClass):  
 *"""  
 Дочерний класс автомобиль  
 """* **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)  
  
  
**class** MotorcycleClass(TransportClass):  
 *"""  
 Дочерний класс мотоцикл  
 """* **def** \_\_init\_\_(self, model, number, speed, carrying, addition):  
 super().\_\_init\_\_(model, number, speed, carrying)  
  
 self.addition = addition  
 *# Для красивого вывода в get\_info* self.formater\_dict = {  
 **True**: **"Да"**,  
 **False**: **"Нет"**,  
 }  
 self.carrying\_calculating()  
  
 **def** carrying\_calculating(self):  
 *# Это все адекватные люди сделали бы это в конструкторе, но задание требует* **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]  
  
  
*# Я бы занаследовался бы от мотоцикла***class** TruckClass(TransportClass):  
 *"""  
 Дочерний класс грузовик  
 """* **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]  
  
  
**def** main():  
 **try**:  
 n = int(input(**"Введите количество транспорта -> "**))  
 **except** ValueError:  
 print(**"Некорректный ввод данных"**)  
 **return** d = {  
 1: TransportClass,  
 2: CarClass,  
 3: MotorcycleClass,  
 4: TruckClass,  
 }  
  
 transport\_list = []  
 **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"**)  
  
 *# Организуйте поиск машин, удовлетворяющих требованиям грузоподъемности.* **try**:  
 carrying\_input = float(input(**"Введите грузоподъёмность -> "**))  
 **except** ValueError:  
 print(**"Некорректный ввод данных"**)  
 **return** print(**"\n\*\*Транспорт, грузоподъёмность которого меньше или равна заданной\*\*\n"**)  
 search\_flag = **False  
 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\*\*"**)  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()

Задание 2.5.

*"""  
Задача 5. 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, дата производства, срок годности, количество штук),  
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.  
"""***from** random **import** randint  
  
**from** faker **import** Faker  
  
  
**class** GoodsClass:  
 *"""  
 Родительский класс товар  
 """* **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  
  
  
class** ProductClass(GoodsClass):  
 *"""  
 Дочерний класс продукт  
 """* **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  
  
  
**class** GroupClass(GoodsClass):  
 *"""  
 Дочерний класс партия  
 """* **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)  
  
  
**class** PhoneClass(GoodsClass):  
 *# Не обращаемся через super().\_\_init\_\_ т.к. нет некоторых полей* **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."  
  
  
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]))  
  
 **for** goods **in** goods\_list:  
 print(goods.get\_info() + **"\n"**)  
  
 *# Организуйте поиск товара, который может приобрести покупатель, имеющий заданную сумму денег.* **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"**)  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()

Задание 2.6.

*"""  
Задача 6. 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, возраст, на который рассчитана, материал),  
КНИГА (название, цена, издательство, возраст, на который рассчитана, автор),  
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.  
"""***from** random **import** randint  
  
**from** faker **import** Faker  
  
  
**class** GoodsClass:  
 *"""  
 Класс товар  
 """* **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  
  
  
class** ToyClass(GoodsClass):  
 *"""  
 Класс игрушка  
 """* **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  
  
  
**class** BookClass(GoodsClass):  
 *"""  
 Класс книга  
 """* **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  
  
  
**class** SportGoodsClass(GoodsClass):  
 *"""  
 Класс спортинвентарь  
 """* **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)  
  
  
**def** main():  
 *"""  
 Создайте список из п товаров, выведите полную информацию из базы на экран,  
 а также организуйте поиск товаров для потребителя в заданном возрастном диапазоне.  
 """* **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"**)  
  
 *# Организуйте поиск товара, который может приобрести покупатель, имеющий заданную сумму денег.* **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"**)  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()

Задание 2.7.

*"""  
Задача 7. 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.  
"""***from** random **import** randint  
  
**from** faker **import** Faker  
  
  
**class** PhoneDictionaryClass:  
 *"""  
 Класс телефонный справочник  
 """* **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):  
 ...  
  
  
**class** PersonClass(PhoneDictionaryClass):  
 *"""  
 Класс персона  
 """* **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  
  
  
**class** OrganizationClass(PhoneDictionaryClass):  
 *"""  
 Класс организация  
 """* **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  
  
  
class** FriendClass(PhoneDictionaryClass):  
 *"""  
 Класс друг  
 """* **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  
  
  
**def** main():  
 *"""  
 Создайте список из п записей, выведите полную информацию из базы на экран,  
 а также организуйте поиск в базе по фамилии.  
 """* **try**:  
 n = int(input(**"Введите количество записей -> "**))  
 **except** ValueError:  
 print(**"Некорректный ввод данных"**)  
 **return** d = {  
 1: PersonClass,  
 2: OrganizationClass,  
 3: FriendClass,  
 }  
  
 fake = Faker([**'ru\_RU'**])  
 obj\_list = []  
  
 **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"**)  
  
 **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"**)  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()

Задание 2.8.

*"""  
Задача 8. 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.  
"""***from** random **import** randint  
  
**from** faker **import** Faker  
  
  
**class** ClentClass:  
 *"""  
 Класс клиент  
 """* **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  
  
  
class** ContributorClass(ClentClass):  
 *"""  
 Класс вкладчик  
 """* **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) + **"%"  
  
  
class** CreditorClass(ClentClass):  
 *"""  
 Класс кредитор  
 """* **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."  
  
  
class** OrganizationClass(ClentClass):  
 *"""  
 Класс организация  
 """* **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."  
  
  
def** main():  
 *"""  
 Создайте список из п клиентов, выведите полную информацию из базы на экран,  
 а также организуйте поиск клиентов, начавших сотрудничать с банком в заданную дату.  
 """* **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"**)  
  
 *# Поиск клиентов, начавших сотрудничать с банком в заданную дату.* **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"**)  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()

Задание 2.9.

*"""  
Задача 9. 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, дата установки, срок использования, цена)   
  
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.  
"""***import** datetime  
**from** random **import** randint  
  
**from** faker **import** Faker  
  
  
**class** SoftwareClass:  
 *"""  
 Класс ПО  
 """* **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* ...  
  
  
**class** FreewareClass(SoftwareClass):  
 *"""  
 Свободное ПО  
 """* **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  
  
  
class** TrialClass(SoftwareClass):  
 *"""  
 Условно бесплатное ПО  
 """* **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):  
 *# Если дата установки + кол-во дней тестирования >= текущей даты, то ок* **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  
  
  
**class** CommercialClass(SoftwareClass):  
 *"""  
 Коммерческое ПО  
 """* **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  
  
  
**def** main():  
 *"""  
 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.  
 """* **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]))  
  
 **for** software **in** software\_list:  
 print(software.software\_info() + **"\n"**)  
  
 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"**)  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()

Задание 2.10.

*"""  
Задача 10. 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, координаты, количество пассажиров, максимальная скорость, максимаьнльная высота),  
АВТОМОБИЛЬ (марка, координаты, номер, год выпуска),  
КОРАБЛЬ (название, координаты, количество пассажиров, максимальная скорость, 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.  
"""***from** random **import** randint  
  
**from** faker **import** Faker  
  
  
**class** TransportClass():  
 *"""  
 Родительский класс транспорт  
 """* **def** \_\_init\_\_(self, name, coords):  
 self.name = name  
 self.coords = coords  
  
 **def** coords\_detector(self, coord\_list):  
 *# Пример coord\_list: ([x\_begin,y\_begin],[x\_end,y\_end])* 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()  
  
  
**class** AirplaneClass(TransportClass):  
 *"""  
 Класс Самолет  
 """* **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)  
  
  
**class** CarClass(TransportClass):  
 *"""  
 Класс автомобиль  
 """* **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)  
  
  
**class** ShipClass(TransportClass):  
 *"""  
 Класс Корабль  
 """* **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  
  
  
**def** main():  
 *"""  
 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.  
 """* **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]))  
  
 **for** transport **in** transport\_list:  
 print(transport.info() + **"\n"**)  
  
 *# поиск транспортных средств, которые сейчас находятся в пределах заданных координат.* search\_flag = **False** *#* ***TODO Задание координат* 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"**)  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()

Задание 2.11.

*"""  
Задача 11. 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 (цвет, цена, материал, диаметр),  
МАШИНКА (цвет, цена, название, производитель) 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.  
"""***from** random **import** randint  
  
**from** faker **import** Faker  
**from** random\_color **import** random\_color  
**from** random\_color **import** random\_color  
  
  
**class** ToyClass:  
 *"""  
 Класс игрушка  
 """* **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  
class** CubeClass(ToyClass):  
 *"""  
 Класс Куб  
 """* **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."  
class** BallClass(ToyClass):  
 *"""  
 Класс мяч  
 """* **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."  
class** CarClass(ToyClass):  
 *"""  
 Класс машинка  
 """* **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  
**def** main():  
 *"""  
 Создайте список из п игрушек, выведите полную информацию из базы на экран,  
 а также организуйте поиск игрушек заданного цвета.  
 """* **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]))  
 **for** toy **in** toy\_list:  
 print(toy.toy\_info() + **"\n"**)  
 *# 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"**)  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()

Задание 2.12.

*"""  
Задача 12. 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.  
"""***from** abc **import** ABCMeta, abstractmethod  
**from** math **import** pi  
**from** random **import** randint  
  
  
**class** BodyClass:  
 *"""  
 Абстрактный класс тела  
 """* \_\_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))  
  
  
**class** ParallelepipedClass(BodyClass):  
 *"""  
 Класс параллелепипед  
 """* **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  
  
  
**class** BallClass(BodyClass):  
 *"""  
 Класс шар  
 """* **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)  
  
  
**class** PyramidClass(BodyClass):  
 *"""  
 Класс пирамида  
 """* **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  
  
  
**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\_\_)  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()

Задание 2.13.

*"""  
Задача 13. 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.  
"""***import** math  
**from** random **import** randint  
  
  
**class** EquationClass:  
 *"""  
 Базовый класс уравнение  
 """* **def** \_\_init\_\_(self):  
 **pass  
  
 def** calculation(self):  
 ...  
  
 **def** info(self):  
 ...  
  
  
**class** LinearEquation(EquationClass):  
 **def** \_\_init\_\_(self, a, b, c):  
 self.a = a  
 self.b = b  
 self.c = c  
 *# Поскольку я устал и мне надо сделать просто КАКОЕ-ТО ЛИНЕЙНОЕ УРАВНЕНИЕ* self.calculation()  
  
 **def** calculation(self):  
 *# Просто представляем, что вся задача сводится к решению ax+b=c* self.result = (self.c - self.b) / self.a  
  
 **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)  
  
  
**class** QuadraticEquation(EquationClass):  
 *"""  
 Квадратное уравнение  
 """* **def** \_\_init\_\_(self, input\_str):  
 self.equation\_results\_list = []  
  
 self.input\_str = input\_str  
 self.parse\_exp()  
 self.calculation()  
  
 **def** is\_digital(self, number):  
 **try**:  
 float(number)  
 **return True  
 except**:  
 **return False  
  
 def** parse\_exp(self):  
 input\_str = self.input\_str  
  
 *# Определяем коэффициент a* symbols = 1  
 return\_flag = **False  
 while** return\_flag == **False**:  
  
 index\_a = input\_str[:symbols]  
 **if** self.is\_digital(index\_a) == **False and** symbols != 1:  
 return\_flag = **True** symbols += 1  
  
 index\_a = input\_str[:symbols - 2]  
  
 *# Определяем коэффициент b* 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** self.is\_digital(index\_b) == **True**:  
 return\_flag = **True** symbols += 1  
  
 *# Определяем коэффициент c* 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** self.is\_digital(index\_c) == **True**:  
 return\_flag = **True** symbols += 1  
  
 self.index\_a = float(index\_a)  
 self.index\_b = float(index\_b)  
 self.index\_c = float(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  
  
 **def** calculation(self):  
 a = self.index\_a  
 b = self.index\_b  
 c = self.index\_c  
  
 D = b \*\* 2 - 4 \* a \* c  
  
 **if** D == 0:  
 A1 = -b / (2 \* a)  
 A2 = A1  
 self.equation\_results\_list.append(A1)  
 self.equation\_results\_list.append(A2)  
  
 **elif** D > 0:  
 A1 = (-b + math.sqrt(D)) / (2 \* a)  
 A2 = (-b - math.sqrt(D)) / (2 \* a)  
 self.equation\_results\_list.append(A1)  
 self.equation\_results\_list.append(A2)  
  
 **else**:  
 self.equation\_results\_list.append(**"Нет решения, D < 0"**)  
  
  
**def** main():  
 *# Создайте список n уравнений и выведите полную информацию об уравнениях на экран.* **try**:  
 n = int(input(**"Введите количество уравнений -> "**))  
 **except** ValueError:  
 print(**"Некорректный ввод данных"**)  
 **return** d = {  
 1: LinearEquation,  
 2: QuadraticEquation,  
 }  
  
 main\_list = []  
  
 **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"**)  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()

Задание 2.14.

*"""  
Задача 14. 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.  
"""***from** random **import** randint  
  
  
**class** CurrencyClass:  
 *"""  
 Родительский класс с абстрактной валютой  
 """  
 # Стат поля* eur\_currency = 70.94  
 usd\_currency = 65.27  
  
 **def** \_\_init\_\_(self, balance, exchange):  
 self.balance = balance  
 self.exchange = exchange  
  
 **def** converter(self):  
 **return** self.balance \* self.exchange  
  
 **def** info(self):  
 **return "[Родительский класс валюты]\nКол-во: "** + str(self.balance) + **"\nКурс обмена: "** + str(  
 self.exchange) + **"\nСконвертированная валюта: "** + str(self.converter())  
  
  
**class** EURClass(CurrencyClass):  
 *"""  
 Класс для работы с EUR  
 """* **def** \_\_init\_\_(self, balance):  
 self.currency = super().eur\_currency  
 self.balance = balance  
  
 **def** converter(self):  
 **return** self.balance \* self.currency  
  
 **def** info(self):  
 **return "[EUR]\nКол-во EUR: "** + str(self.balance) + **"\nКурс обмена: "** + str(  
 self.currency) + **"\nСконвертированная валюта в RUB: "** + str(self.converter())  
  
  
**class** USDClass(CurrencyClass):  
 *"""  
 Класс для работы с USD  
 """* **def** \_\_init\_\_(self, balance):  
 self.currency = super().usd\_currency  
 self.balance = balance  
  
 **def** converter(self):  
 **return** self.balance \* self.currency  
  
 **def** info(self):  
 **return "[USD]\nКол-во USD: "** + str(self.balance) + **"\nКурс обмена: "** + str(  
 self.currency) + **"\nСконвертированная валюта в RUB: "** + str(self.converter())  
  
  
**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"**)  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()

Задание 2.15.

*"""  
Задача 15. Создайте класс ПРОГРЕССИЯ с методами вычисления N-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.  
"""***from** random **import** randint  
  
  
**class** ProgressionClass:  
  
 **def** get\_sum(self):  
 **return** self.s  
  
 **def** calculate(self):  
 ...  
  
 **def** info(self):  
 ...  
  
  
**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)  
  
  
**class** GeometricClass(ProgressionClass):  
 **def** \_\_init\_\_(self, b1, q):  
 self.b1 = b1  
 self.q = q  
 self.calculate()  
  
 **def** calculate(self):  
 self.s = self.b1 / (1 - self.q)  
  
 **def** info(self):  
 **return "[Геометрическая прогрессия]\nЗнаменатель прогрессии: "** + str(self.q) + **"\n1-й элемент: "** + str(  
 self.b1) + **"\nСумма: "** + str(self.s)  
  
  
**def** main():  
 *# Создайте список п прогрессий и выведите сумму каждой из них экран.* **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"**)  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()