Лабораторная работа №11

Тема: Объектно-ориентированное программирование

Задача 1. 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 n фигур и выведите полную информацию о фигурах на экран.

Таблица данных:

Класс Имя Смысл Тип, структура Ограничения на значения

Входные данные width Ширина прямоугольника вещественное ширина > 0

height Высота прямоугольника вещественное высота > 0

radius Радиус круга вещественное радиус > 0

a, b, c Стороны треугольника вещественное стороны > 0

Выходные данные площадь Площадь фигуры вещественное площадь > 0

периметр Периметр фигуры вещественное периметр > 0

Тесты:

Входные данные Ожидаемый результат

width = 3, height = 4 Площадь: 12, Периметр: 14

radius = 5 Площадь: 78.5, Периметр: 31.4

a = 3, b = 4, c = 5 Площадь: 6, Периметр: 12

Листинг программы (текстом):

class Figura:

def area(self):

pass

def perimeter(self):

pass

def info(self):

pass

class Pryamougolnik(Figura):

def init(self, width, height):

self.width = width

self.height = height

def area(self):

return self.width \* self.height

def perimeter(self):

return 2 \* (self.width + self.height)

def info(self):

print(f"Прямоугольник: ширина = {self.width}, высота = {self.height}, площадь = {self.area()}, периметр = {self.perimeter()}")

class Krug(Figura):

def init(self, radius):

self.radius = radius

def area(self):

return 3.14 \* self.radius \*\* 2

def perimeter(self):

return 2 \* 3.14 \* self.radius

def info(self):

print(f"Круг: радиус = {self.radius}, площадь = {self.area()}, периметр = {self.perimeter()}")

class Treugolnik(Figura):

def init(self, a, b, c):

self.a = a

self.b = b

self.c = c

def area(self):

p = (self.a + self.b + self.c) / 2

return (p \* (p - self.a) \* (p - self.b) \* (p - self.c)) \*\* 0.5

def perimeter(self):

return self.a + self.b + self.c

def info(self):

print(f"Треугольник: стороны = {self.a}, {self.b}, {self.c}, площадь = {self.area()}, периметр = {self.perimeter()}")

figury = [Pryamougolnik(2, 3), Krug(1), Treugolnik(3, 4, 5)]

for figura in figury:

figura.info()

Задача 2. 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 n изданий, выведите полную информацию из списка, а также организуйте поиск изданий по фамилии автора.

Таблица данных:

Класс Имя Смысл Тип, структура Ограничения на значения

Входные данные name Название издания строка Не пустая строка

author Автор издания строка Не пустая строка

year Год издания целое year > 0

journal Название журнала строка Не пустая строка

link Ссылка на ресурс строка Действительная ссылка

Выходные данные information Информация об издании строка Не пустая строка

Тесты:

Входные данные Ожидаемый результат

Название: Книга1, Автор: Иванов Информация: Книга1, Иванов, 2019, Издательство1

Название: Статья1, Автор: Петров Информация: Статья1, Петров, Журнал1, 5, 2021

Название: Ресурс1, Автор: Сидоров Информация: Ресурс1, Сидоров, http://link.com

Листинг программы (текстом):

class Izdanie:

def init(self, name, author):

self.name = name

self.author = author

def info(self):

pass

def search\_by\_author(self, search\_author):

return self.author == search\_author

class Kniga(Izdanie):

def init(self, name, author, year, publisher):

super().init(name, author)

self.year = year

self.publisher = publisher

def info(self):

print(f"Книга: {self.name}, Автор: {self.author}, Год издания: {self.year}, Издательство: {self.publisher}")

class Statya(Izdanie):

def init(self, name, author, journal, issue, year):

super().init(name, author)

self.journal = journal

self.issue = issue

self.year = year

def info(self):

print(f"Статья: {self.name}, Автор: {self.author}, Журнал: {self.journal}, Номер: {self.issue}, Год: {self.year}")

class ElektronnyResurs(Izdanie):

def init(self, name, author, link, annotation):

super().init(name, author)

self.link = link

self.annotation = annotation

def info(self):

print(f"Электронный ресурс: {self.name}, Автор: {self.author}, Ссылка: {self.link}, Аннотация: {self.annotation}")

izdania = [Kniga("Книга1", "Автор1", 2020, "Издательство1"), Statya("Статья1", "Автор2", "Журнал1", 5, 2019), ElektronnyResurs("Ресурс1", "Автор3", "http://link.com", "Аннотация ресурса")]

for izdanie in izdania:

izdanie.info()

search\_author = input("Введите фамилию автора для поиска: ")

for izdanie in izdania:

if izdanie.search\_by\_author(search\_author):

print(f"Найдено издание: {izdanie.name}")

#### **Задача 3. 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 n треугольников и выведите полную информацию о треугольниках на экран.

**Таблица данных**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Класс** | **Имя** | **Смысл** | **Тип, структура** | **Ограничения на значения** |
| Входные данные | a, b | Длины двух сторон | вещественное | стороны > 0 |
|  | угол | Угол между сторонами | вещественное | 0 < угол < 180 |
| Выходные данные | площадь | Площадь треугольника | вещественное | площадь > 0 |
|  | периметр | Периметр треугольника | вещественное | периметр > 0 |

**Тесты**:

|  |  |
| --- | --- |
| **Входные данные** | **Ожидаемый результат** |
| a = 3, b = 4, угол = 90 | Площадь: 6, Периметр: 12 |
| a = 5, b = 5, угол = 60 | Площадь: 10.83, Периметр: 15 |
| a = 6, b = 6, угол = 60 | Площадь: 15.59, Периметр: 18 |

**Листинг программы (текстом)**:

import math

class Treugolnik:  
def **init**(self, a, b, angle):  
self.a = a  
self.b = b  
self.angle = math.radians(angle)

def area(self):  
return 0.5 \* self.a \* self.b \* math.sin(self.angle)

def perimeter(self):  
c = math.sqrt(self.a**2 + self.b**2 - 2*self.a*self.b\*math.cos(self.angle))  
return self.a + self.b + c

def info(self):  
print(f"Треугольник: стороны = {self.a}, {self.b}, угол = {math.degrees(self.angle)}, площадь = {self.area()}, периметр = {self.perimeter()}")

class Pryamougolny(Treugolnik):  
def info(self):  
print(f"Прямоугольный треугольник: стороны = {self.a}, {self.b}, площадь = {self.area()}, периметр = {self.perimeter()}")

class Ravnostoronny(Treugolnik):  
def **init**(self, a):  
super().**init**(a, a, 60)

def info(self):  
print(f"Равносторонний треугольник: сторона = {self.a}, площадь = {self.area()}, периметр = {self.perimeter()}")

class Ravnobedrenny(Treugolnik):  
def **init**(self, a, b):  
super().**init**(a, b, 45)

def info(self):  
print(f"Равнобедренный треугольник: стороны = {self.a}, {self.b}, площадь = {self.area()}, периметр = {self.perimeter()}")

treugolniki = [Pryamougolny(3, 4, 90), Ravnostoronny(5), Ravnobedrenny(6, 8)]

for treugolnik in treugolniki:  
treugolnik.info()

#### **Задача 4. 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 n машин, выведите полную информацию на экран, а также организуйте поиск машин, удовлетворяющих требованиям грузоподъемности.

**Таблица данных**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Класс** | **Имя** | **Смысл** | **Тип, структура** | **Ограничения на значения** |
| Входные данные | марка | Марка транспортного средства | строка | Не пустая строка |
|  | номер | Номер транспортного средства | строка | Не пустая строка |
|  | скорость | Скорость транспортного средства | вещественное | скорость > 0 |
|  | грузоподъемность | Грузоподъемность | вещественное | грузоподъемность > 0 |
| Выходные данные | информация | Информация о транспортном средстве | строка | Не пустая строка |

**Тесты**:

|  |  |
| --- | --- |
| **Входные данные** | **Ожидаемый результат** |
| Марка: Авто1, Номер: 123, Грузоподъемность: 1000 | Грузоподъемность = 1000 |
| Марка: Мотоцикл1, Номер: 456, Наличие коляски: да | Грузоподъемность = 200 |
| Марка: Грузовик1, Номер: 789, Наличие прицепа: да | Грузоподъемность = 3000 |

**Листинг программы (текстом)**:

class Transport:  
def **init**(self, marka, nomer, skorost):  
self.marka = marka  
self.nomer = nomer  
self.skorost = skorost

def info(self):  
pass

def gruzopodemnost(self):  
pass

class Avtomobil(Transport):  
def **init**(self, marka, nomer, skorost, gruzopodemnost):  
super().**init**(marka, nomer, skorost)  
self.gruzopodemnost = gruzopodemnost

def info(self):  
print(f"Автомобиль: марка = {self.marka}, номер = {self.nomer}, скорость = {self.skorost}, грузоподъемность = {self.gruzopodemnost}")

class Mototsikl(Transport):  
def **init**(self, marka, nomer, skorost, kolyaska):  
super().**init**(marka, nomer, skorost)  
self.kolyaska = kolyaska

def gruzopodemnost(self):  
if self.kolyaska:  
return 200  
else:  
return 0

def info(self):  
print(f"Мотоцикл: марка = {self.marka}, номер = {self.nomer}, скорость = {self.skorost}, грузоподъемность = {self.gruzopodemnost()}")

class Gruzovik(Transport):  
def **init**(self, marka, nomer, skorost, gruzopodemnost, pricep):  
super().**init**(marka, nomer, skorost)  
self.gruzopodemnost = gruzopodemnost  
self.pricep = pricep

def gruzopodemnost(self):  
if self.pricep:  
return self.gruzopodemnost \* 2  
else:  
return self.gruzopodemnost

def info(self):  
print(f"Грузовик: марка = {self.marka}, номер = {self.nomer}, скорость = {self.skorost}, грузоподъемность = {self.gruzopodemnost()}")

transport = [Avtomobil("Авто1", "123", 150, 1000), Mototsikl("Мотоцикл1", "456", 100, True), Gruzovik("Грузовик1", "789", 120, 1500, True)]

for t in transport:  
t.info()

#### **Задача 5. 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 n товаров, выведите полную информацию из базы на экран, а также организуйте поиск товара, который может приобрести покупатель, имеющий заданную сумму денег.

**Таблица данных**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Класс** | **Имя** | **Смысл** | **Тип, структура** | **Ограничения на значения** |
| Входные данные | название | Название товара | строка | Не пустая строка |
|  | цена | Цена товара | вещественное | цена > 0 |
|  | дата\_производства | Дата производства | строка | Формат даты |
|  | срок\_годности | Срок годности | строка | Формат даты |
| Выходные данные | информация | Информация о товаре | строка | Не пустая строка |

**Тесты**:

|  |  |
| --- | --- |
| **Входные данные** | **Ожидаемый результат** |
| Название: Хлеб, Цена: 30, Срок годности: 3 дня | Может купить при наличии 50 руб. |
| Название: Телефон, Цена: 5000 | Может купить при наличии 6000 руб. |
| Название: Партия, Цена за штуку: 20, Количество: 50 | Может купить 50 штук за 1000 руб. |

**Листинг программы (текстом)**:

class Tovar:  
def **init**(self, nazvanie, cena):  
self.nazvanie = nazvanie  
self.cena = cena

def info(self):  
pass

def mozhno\_kupit(self, suma):  
return suma >= self.cena

class Produkt(Tovar):  
def **init**(self, nazvanie, cena, data\_proizvodstva, srok\_godnosti):  
super().**init**(nazvanie, cena)  
self.data\_proizvodstva = data\_proizvodstva  
self.srok\_godnosti = srok\_godnosti

def info(self):  
print(f"Продукт: {self.nazvanie}, цена = {self.cena}, дата производства = {self.data\_proizvodstva}, срок годности = {self.srok\_godnosti}")

class Partiya(Tovar):  
def **init**(self, nazvanie, cena\_za\_shtuku, kolichestvo, data\_proizvodstva, srok\_godnosti):  
super().**init**(nazvanie, cena\_za\_shtuku \* kolichestvo)  
self.cena\_za\_shtuku = cena\_za\_shtuku  
self.kolichestvo = kolichestvo  
self.data\_proizvodstva = data\_proizvodstva  
self.srok\_godnosti = srok\_godnosti

def info(self):  
print(f"Партия: {self.nazvanie}, цена за штуку = {self.cena\_za\_shtuku}, количество = {self.kolichestvo}, общая цена = {self.cena}, дата производства = {self.data\_proizvodstva}, срок годности = {self.srok\_godnosti}")

class Telefon(Tovar):  
def info(self):  
print(f"Телефон: {self.nazvanie}, цена = {self.cena}")

tovary = [Produkt("Хлеб", 30, "01.01.2024", "3 дня"), Partiya("Молоко", 20, 50, "01.01.2024", "5 дней"), Telefon("iPhone", 5000)]

for tovar in tovary:  
tovar.info()

# **Проверка, может ли покупатель купить товар на заданную сумму**

suma = int(input("Введите сумму для проверки: "))  
for tovar in tovary:  
if tovar.mozhno\_kupit(suma):  
print(f"Можно купить: {tovar.nazvanie}")  
else:  
print(f"Нельзя купить: {tovar.nazvanie}")

#### **Задача 6. 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 n товаров, выведите полную информацию из базы на экран, а также организуйте поиск товаров для потребителя в заданном возрастном диапазоне.

**Таблица данных**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Класс** | **Имя** | **Смысл** | **Тип, структура** | **Ограничения на значения** |
| Входные данные | возраст | Возраст потребителя | целое | возраст > 0 |
| Выходные данные | информация | Информация о товаре | строка | Не пустая строка |

**Тесты**:

|  |  |
| --- | --- |
| **Входные данные** | **Ожидаемый результат** |
| Название: Игрушка, Возраст: 3-5 лет | Подходит для детей 4 лет |
| Название: Книга, Возраст: 10-15 лет | Подходит для подростков 12 лет |

**Листинг программы (текстом)**:

class Tovar:  
def **init**(self, nazvanie, cena):  
self.nazvanie = nazvanie  
self.cena = cena

def info(self):  
pass

def podhodit\_dlya\_vozrasta(self, vozrast):  
pass

class Igrushka(Tovar):  
def **init**(self, nazvanie, cena, proizvoditel, material, vozrast):  
super().**init**(nazvanie, cena)  
self.proizvoditel = proizvoditel  
self.material = material  
self.vozrast = vozrast

def info(self):  
print(f"Игрушка: {self.nazvanie}, цена = {self.cena}, производитель = {self.proizvoditel}, материал = {self.material}, возраст = {self.vozrast}")

def podhodit\_dlya\_vozrasta(self, vozrast):  
return vozrast in range(self.vozrast[0], self.vozrast[1]+1)

class Kniga(Tovar):  
def **init**(self, nazvanie, cena, avtor, izdatelstvo, vozrast):  
super().**init**(nazvanie, cena)  
self.avtor = avtor  
self.izdatelstvo = izdatelstvo  
self.vozrast = vozrast

def info(self):  
print(f"Книга: {self.nazvanie}, автор = {self.avtor}, цена = {self.cena}, издательство = {self.izdatelstvo}, возраст = {self.vozrast}")

def podhodit\_dlya\_vozrasta(self, vozrast):  
return vozrast in range(self.vozrast[0], self.vozrast[1]+1)

class Sportinventar(Tovar):  
def **init**(self, nazvanie, cena, proizvoditel, vozrast):  
super().**init**(nazvanie, cena)  
self.proizvoditel = proizvoditel  
self.vozrast = vozrast

def info(self):  
print(f"Спортинвентарь: {self.nazvanie}, цена = {self.cena}, производитель = {self.proizvoditel}, возраст = {self.vozrast}")

def podhodit\_dlya\_vozrasta(self, vozrast):  
return vozrast in range(self.vozrast[0], self.vozrast[1]+1)

tovary = [Igrushka("Кукла", 500, "Производитель1", "Пластик", (3, 6)), Kniga("Гарри Поттер", 800, "Дж. Роулинг", "Издательство1", (10, 15)), Sportinventar("Мяч", 300, "Производитель2", (5, 12))]

# **Вывод информации**

vozrast = int(input("Введите возраст для проверки: "))  
for tovar in tovary:  
if tovar.podhodit\_dlya\_vozrasta(vozrast):  
print(f"Подходит для возраста: {tovar.nazvanie}")  
else:  
print(f"Не подходит для возраста: {tovar.nazvanie}")

#### **Задача 7. 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 n записей, выведите полную информацию из базы на экран, а также организуйте поиск в базе по фамилии.

**Таблица данных**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Класс** | **Имя** | **Смысл** | **Тип, структура** | **Ограничения на значения** |
| Входные данные | фамилия | Фамилия контакта | строка | Не пустая строка |
|  | адрес | Адрес контакта | строка | Не пустая строка |
|  | номер\_телефона | Телефонный номер | строка | Должен быть в формате телефона |
|  | дата\_рождения | Дата рождения | строка | Должна быть валидной датой |
| Выходные данные | информация | Информация о контакте | строка | Не пустая строка |

**Тесты**:

|  |  |
| --- | --- |
| **Входные данные** | **Ожидаемый результат** |
| Фамилия: Иванов, Телефон: 555-1234 | Вывод записи с телефоном |
| Название организации: ООО "Ромашка", Телефон: 555-9876 | Вывод записи с организацией |

**Листинг программы (текстом)**:

class TelefonnyiSpravochnik:  
def info(self):  
pass

class Persona(TelefonnyiSpravochnik):  
def **init**(self, familiya, adres, telefon):  
self.familiya = familiya  
self.adres = adres  
self.telefon = telefon

def info(self):  
print(f"Персона: {self.familiya}, адрес = {self.adres}, телефон = {self.telefon}")

def poisk\_po\_familii(self, familiya):  
return self.familiya == familiya

class Organizatsiya(TelefonnyiSpravochnik):  
def **init**(self, nazvanie, adres, telefon, faks, kontaktnoe\_litso):  
self.nazvanie = nazvanie  
self.adres = adres  
self.telefon = telefon  
self.faks = faks  
self.kontaktnoe\_litso = kontaktnoe\_litso

def info(self):  
print(f"Организация: {self.nazvanie}, адрес = {self.adres}, телефон = {self.telefon}, факс = {self.faks}, контактное лицо = {self.kontaktnoe\_litso}")

class Drug(TelefonnyiSpravochnik):  
def **init**(self, familiya, adres, telefon, data\_rozhdeniya):  
self.familiya = familiya  
self.adres = adres  
self.telefon = telefon  
self.data\_rozhdeniya = data\_rozhdeniya

def info(self):  
print(f"Друг: {self.familiya}, адрес = {self.adres}, телефон = {self.telefon}, дата рождения = {self.data\_rozhdeniya}")

def poisk\_po\_familii(self, familiya):  
return self.familiya == familiya

spravochnik = [Persona("Иванов", "Москва", "555-1234"), Organizatsiya("ООО 'Ромашка'", "СПб", "555-9876", "555-5678", "Петров"), Drug("Сидоров", "Казань", "555-3456", "01.01.1990")]

# **Поиск записи по фамилии**

familiya = input("Введите фамилию для поиска: ")  
for zapis in spravochnik:  
if isinstance(zapis, Persona) or isinstance(zapis, Drug):  
if zapis.poisk\_po\_familii(familiya):  
zapis.info()

#### **Задача 8. 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 n клиентов, выведите полную информацию из базы на экран, а также организуйте поиск клиентов, начавших сотрудничать с банком в заданную дату.

**Таблица данных**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Класс** | **Имя** | **Смысл** | **Тип, структура** | **Ограничения на значения** |
| Входные данные | фамилия | Фамилия клиента | строка | Не пустая строка |
|  | дата\_вклада | Дата открытия вклада | строка | Формат даты |
|  | размер\_вклада | Сумма вклада | вещественное | Неотрицательное число |
|  | процент\_по\_вкладу | Процентная ставка | вещественное | Процент > 0 |
| Выходные данные | информация | Информация о клиенте | строка | Не пустая строка |

**Тесты**:

|  |  |
| --- | --- |
| **Входные данные** | **Ожидаемый результат** |
| Вкладчик: Иванов, Вклад: 10000, Процент: 5 | Вывод информации о вкладе |
| Кредитор: Петров, Кредит: 50000, Остаток долга: 30000 | Вывод информации о кредите |

**Листинг программы (текстом)**:

class Klient:  
def info(self):  
pass

class Vkladchik(Klient):  
def **init**(self, familiya, data\_vklada, razmer\_vklada, procent):  
self.familiya = familiya  
self.data\_vklada = data\_vklada  
self.razmer\_vklada = razmer\_vklada  
self.procent = procent

def info(self):  
print(f"Вкладчик: {self.familiya}, дата вклада = {self.data\_vklada}, сумма вклада = {self.razmer\_vklada}, процент = {self.procent}")

def poisk\_po\_date(self, data):  
return self.data\_vklada == data

class Kreditor(Klient):  
def **init**(self, familiya, data\_kredita, razmer\_kredita, procent, dolg):  
self.familiya = familiya  
self.data\_kredita = data\_kredita  
self.razmer\_kredita = razmer\_kredita  
self.procent = procent  
self.dolg = dolg

def info(self):  
print(f"Кредитор: {self.familiya}, дата кредита = {self.data\_kredita}, сумма кредита = {self.razmer\_kredita}, остаток долга = {self.dolg}, процент = {self.procent}")

def poisk\_po\_date(self, data):  
return self.data\_kredita == data

class Organizatsiya(Klient):  
def **init**(self, nazvanie, data\_scheta, nomer\_scheta, summa\_na\_schetu):  
self.nazvanie = nazvanie  
self.data\_scheta = data\_scheta  
self.nomer\_scheta = nomer\_scheta  
self.summa\_na\_schetu = summa\_na\_schetu

def info(self):  
print(f"Организация: {self.nazvanie}, дата счета = {self.data\_scheta}, номер счета = {self.nomer\_scheta}, сумма на счету = {self.summa\_na\_schetu}")

def poisk\_po\_date(self, data):  
return self.data\_scheta == data

klienty = [Vkladchik("Иванов", "01.01.2020", 10000, 5), Kreditor("Петров", "01.06.2020", 50000, 10, 30000), Organizatsiya("ООО 'Вектор'", "15.03.2020", "40702810900000012345", 500000)]

# **Поиск клиента по дате**

data = input("Введите дату для поиска: ")  
for klient in klienty:  
if klient.poisk\_po\_date(data):  
klient.info()

#### **Задача 9. 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 n видов программного обеспечения, выведите полную информацию из базы на экран, а также организуйте поиск программного обеспечения, которое допустимо использовать на текущую дату.

**Таблица данных**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Класс** | **Имя** | **Смысл** | **Тип, структура** | **Ограничения на значения** |
| Входные данные | название | Название ПО | строка | Не пустая строка |
|  | дата\_установки | Дата установки | строка | Формат даты |
|  | срок\_использования | Срок использования | строка | Должен быть валидной датой |
|  | цена | Цена ПО | вещественное | Цена > 0 |
| Выходные данные | информация | Информация о ПО | строка | Не пустая строка |

**Тесты**:

|  |  |
| --- | --- |
| **Входные данные** | **Ожидаемый результат** |
| Название: Photoshop, Срок использования: 1 год | Программа доступна |
| Название: LibreOffice, Свободное ПО | Программа доступна |

**Листинг программы (текстом)**:

class ProgramnoeObespechenie:  
def info(self):  
pass

def dostupno(self):  
pass

class Svobodnoe(ProgramnoeObespechenie):  
def **init**(self, nazvanie, proizvoditel):  
self.nazvanie = nazvanie  
self.proizvoditel = proizvoditel

def info(self):  
print(f"Свободное ПО: {self.nazvanie}, производитель = {self.proizvoditel}")

def dostupno(self):  
return True

class UsovnoBesplatnoe(ProgramnoeObespechenie):  
def **init**(self, nazvanie, proizvoditel, data\_ustanovki, srok\_besplatnogo\_ispolzovaniya):  
self.nazvanie = nazvanie  
self.proizvoditel = proizvoditel  
self.data\_ustanovki = data\_ustanovki  
self.srok\_besplatnogo\_ispolzovaniya = srok\_besplatnogo\_ispolzovaniya

def info(self):  
print(f"Условно-бесплатное ПО: {self.nazvanie}, производитель = {self.proizvoditel}, дата установки = {self.data\_ustanovki}, срок бесплатного использования = {self.srok\_besplatnogo\_ispolzovaniya}")

def dostupno(self):

# **Проверка, доступно ли ПО на текущую дату**

# **Логика расчета доступности не указана в задаче**

return True

class Kommercheskoe(ProgramnoeObespechenie):  
def **init**(self, nazvanie, proizvoditel, data\_ustanovki, srok\_ispolzovaniya, cena):  
self.nazvanie = nazvanie  
self.proizvoditel = proizvoditel  
self.data\_ustanovki = data\_ustanovki  
self.srok\_ispolzovaniya = srok\_ispolzovaniya  
self.cena = cena

def info(self):  
print(f"Коммерческое ПО: {self.nazvanie}, производитель = {self.proizvoditel}, дата установки = {self.data\_ustanovki}, срок использования = {self.srok\_ispolzovaniya}, цена = {self.cena}")

def dostupno(self):

# **Проверка доступности ПО по сроку**

return True

progs = [Svobodnoe("LibreOffice", "The Document Foundation"), UsovnoBesplatnoe("WinRAR", "RARLAB", "01.01.2023", "30 дней"), Kommercheskoe("Photoshop", "Adobe", "01.01.2022", "1 год", 5000)]

# **Проверка доступности ПО на текущую дату**

for prog in progs:  
if prog.dostupno():  
prog.info()

#### **Задача 10. 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 n транспортных средств, выведите полную информацию из базы на экран, а также организуйте поиск транспортных средств, которые сейчас находятся в пределах заданных координат.

**Таблица данных**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Класс** | **Имя** | **Смысл** | **Тип, структура** | **Ограничения на значения** |
| Входные данные | марка | Марка транспортного средства | строка | Не пустая строка |
|  | скорость | Максимальная скорость | вещественное | Скорость > 0 |
|  | координаты | Текущие координаты | кортеж | Формат (x, y) |
| Выходные данные | информация | Информация о транспортном средстве | строка | Не пустая строка |

**Тесты**:

|  |  |
| --- | --- |
| **Входные данные** | **Ожидаемый результат** |
| Самолет: Boeing 747, Координаты: (100, 200) | Вывод информации о самолете |
| Автомобиль: Tesla Model S, Координаты: (50, 75) | Вывод информации об автомобиле |

**Листинг программы (текстом)**:

class Transport:  
def info(self):  
pass

def v\_predelah\_koordinat(self, min\_x, max\_x, min\_y, max\_y):  
pass

class Samolet(Transport):  
def **init**(self, marka, max\_skorost, max\_vysota, kolichestvo\_passazhirov, koordinaty):  
self.marka = marka  
self.max\_skorost = max\_skorost  
self.max\_vysota = max\_vysota  
self.kolichestvo\_passazhirov = kolichestvo\_passazhirov  
self.koordinaty = koordinaty

def info(self):  
print(f"Самолет: {self.marka}, максимальная скорость = {self.max\_skorost} км/ч, максимальная высота = {self.max\_vysota} м, количество пассажиров = {self.kolichestvo\_passazhirov}, координаты = {self.koordinaty}")

def v\_predelah\_koordinat(self, min\_x, max\_x, min\_y, max\_y):  
x, y = self.koordinaty  
return min\_x <= x <= max\_x and min\_y <= y <= max\_y

class Avtomobil(Transport):  
def **init**(self, marka, nomer, god\_vypuska, koordinaty):  
self.marka = marka  
self.nomer = nomer  
self.god\_vypuska = god\_vypuska  
self.koordinaty = koordinaty

def info(self):  
print(f"Автомобиль: {self.marka}, номер = {self.nomer}, год выпуска = {self.god\_vypuska}, координаты = {self.koordinaty}")

def v\_predelah\_koordinat(self, min\_x, max\_x, min\_y, max\_y):  
x, y = self.koordinaty  
return min\_x <= x <= max\_x and min\_y <= y <= max\_y

class Korabl(Transport):  
def **init**(self, nazvanie, koordinaty, skorost, kolichestvo\_passazhirov, port\_pripiski):  
self.nazvanie = nazvanie  
self.koordinaty = koordinaty  
self.skorost = skorost  
self.kolichestvo\_passazhirov = kolichestvo\_passazhirov  
self.port\_pripiski = port\_pripiski

def info(self):  
print(f"Корабль: {self.nazvanie}, скорость = {self.skorost} км/ч, количество пассажиров = {self.kolichestvo\_passazhirov}, координаты = {self.koordinaty}, порт приписки = {self.port\_pripiski}")

def v\_predelah\_koordinat(self, min\_x, max\_x, min\_y, max\_y):  
x, y = self.koordinaty  
return min\_x <= x <= max\_x and min\_y <= y <= max\_y

# **Создаем список транспортных средств**

transporty = [Samolet("Boeing 747", 900, 10000, 500, (100, 200)), Avtomobil("Tesla Model S", "A123BC", 2020, (50, 75)), Korabl("Титаник", (30, 60), 40, 3000, "Лондон")]

# **Поиск транспортных средств в пределах заданных координат**

min\_x, max\_x = 0, 100  
min\_y, max\_y = 0, 200  
for transport in transporty:  
if transport.v\_predelah\_koordinat(min\_x, max\_x, min\_y, max\_y):  
transport.info()

#### **Задача 11. 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 n игрушек, выведите полную информацию из базы на экран, а также организуйте поиск игрушек заданного цвета.

**Таблица данных**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Класс** | **Имя** | **Смысл** | **Тип, структура** | **Ограничения на значения** |
| Входные данные | цвет | Цвет игрушки | строка | Не пустая строка |
|  | цена | Цена игрушки | вещественное | Цена > 0 |
|  | материал | Материал игрушки | строка | Не пустая строка |
| Выходные данные | информация | Информация о игрушке | строка | Не пустая строка |

**Тесты**:

|  |  |
| --- | --- |
| **Входные данные** | **Ожидаемый результат** |
| Кубик: Красный, Цена: 100 | Вывод информации о кубике |
| Мяч: Синий, Диаметр: 20 см | Вывод информации о мяче |

**Листинг программы (текстом)**:

class Igrushka:  
def info(self):  
pass

def sootvetstvuet\_tsvetu(self, tsvet):  
pass

class Kubik(Igrushka):  
def **init**(self, tsvet, tsena, material, razmer\_rebra):  
self.tsvet = tsvet  
self.tsena = tsena  
self.material = material  
self.razmer\_rebra = razmer\_rebra

def info(self):  
print(f"Кубик: цвет = {self.tsvet}, цена = {self.tsena}, материал = {self.material}, размер ребра = {self.razmer\_rebra} см")

def sootvetstvuet\_tsvetu(self, tsvet):  
return self.tsvet == tsvet

class Myach(Igrushka):  
def **init**(self, tsena, tsvet, diametr, material):  
self.tsvet = tsvet  
self.tsena = tsena  
self.diametr = diametr  
self.material = material

def info(self):  
print(f"Мяч: цвет = {self.tsvet}, цена = {self.tsena}, диаметр = {self.diametr} см, материал = {self.material}")

def sootvetstvuet\_tsvetu(self, tsvet):  
return self.tsvet == tsvet

class Mashinka(Igrushka):  
def **init**(self, nazvanie, tsena, proizvoditel, tsvet):  
self.nazvanie = nazvanie  
self.tsena = tsena  
self.proizvoditel = proizvoditel  
self.tsvet = tsvet

def info(self):  
print(f"Машинка: название = {self.nazvanie}, цена = {self.tsena}, производитель = {self.proizvoditel}, цвет = {self.tsvet}")

def sootvetstvuet\_tsvetu(self, tsvet):  
return self.tsvet == tsvet

# **Список игрушек**

igrushki = [Kubik("красный", 100, "пластик", 5), Myach(200, "синий", 20, "резина"), Mashinka("Hot Wheels", 300, "Mattel", "зеленый")]

# **Поиск игрушек по цвету**

iskomyi\_tsvet = input("Введите цвет для поиска: ")  
for igrushka in igrushki:  
if igrushka.sootvetstvuet\_tsvetu(iskomyi\_tsvet):  
igrushka.info()

#### **Задача 12. 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 n фигур и выведите полную информацию о фигурах на экран.

**Таблица данных**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Класс** | **Имя** | **Смысл** | **Тип, структура** | **Ограничения на значения** |
| Входные данные | длина | Длина тела (если параллелепипед) | вещественное | > 0 |
|  | радиус | Радиус шара | вещественное | > 0 |
|  | площадь | Площадь поверхности | вещественное | > 0 |
|  | объем | Объем тела | вещественное | > 0 |
| Выходные данные | информация | Информация о теле | строка | Не пустая строка |

**Тесты**:

|  |  |
| --- | --- |
| **Входные данные** | **Ожидаемый результат** |
| Шар: радиус 5 | Площадь поверхности и объем шара |
| Параллелепипед: длина 10, ширина 5, высота 3 | Площадь и объем параллелепипеда |

**Листинг программы (текстом)**:

class Telo:  
def ploshad(self):  
pass

def obyem(self):  
pass

def info(self):  
pass

class Parallelepiped(Telo):  
def **init**(self, dlina, shirina, vysota):  
self.dlina = dlina  
self.shirina = shirina  
self.vysota = vysota

def ploshad(self):  
return 2 \* (self.dlina \* self.shirina + self.dlina \* self.vysota + self.shirina \* self.vysota)

def obyem(self):  
return self.dlina \* self.shirina \* self.vysota

def info(self):  
print(f"Параллелепипед: длина = {self.dlina}, ширина = {self.shirina}, высота = {self.vysota}, площадь поверхности = {self.ploshad()}, объем = {self.obyem()}")

class Shar(Telo):  
def **init**(self, radius):  
self.radius = radius

def ploshad(self):  
import math  
return 4 \* math.pi \* self.radius \*\* 2

def obyem(self):  
import math  
return (4 / 3) \* math.pi \* self.radius \*\* 3

def info(self):  
print(f"Шар: радиус = {self.radius}, площадь поверхности = {self.ploshad()}, объем = {self.obyem()}")

class Piramida(Telo):  
def **init**(self, osnovanie, vysota):  
self.osnovanie = osnovanie  
self.vysota = vysota

def ploshad(self):  
import math  
return self.osnovanie \*\* 2 + 2 \* self.osnovanie \* math.sqrt((self.osnovanie / 2) \*\* 2 + self.vysota \*\* 2)

def obyem(self):  
return (1 / 3) \* self.osnovanie \*\* 2 \* self.vysota

def info(self):  
print(f"Пирамида: основание = {self.osnovanie}, высота = {self.vysota}, площадь поверхности = {self.ploshad()}, объем = {self.obyem()}")

# **Список тел**

tela = [Parallelepiped(10, 5, 3), Shar(5), Piramida(4, 6)]

# **Вывод информации о телах**

for telo in tela:  
telo.info()

#### **Задача 13. 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.

**Таблица данных**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Класс** | **Имя** | **Смысл** | **Тип, структура** | **Ограничения на значения** |
| Входные данные | a, b | Коэффициенты уравнения | вещественное | a ≠ 0 (для линейного уравнения) |
|  | a, b, c | Коэффициенты квадратного уравнения | вещественное | a ≠ 0 |
| Выходные данные | корни | Решение уравнения | список чисел | Не пустой список |

**Тесты**:

|  |  |
| --- | --- |
| **Входные данные** | **Ожидаемый результат** |
| Линейное уравнение: 2x + 3 = 0 | Корень: -1.5 |
| Квадратное уравнение: x^2 - 5x + 6 = 0 | Корни: 2 и 3 |

**Листинг программы (текстом)**:

class Uravnenie:  
def reshit(self):  
pass

def info(self):  
pass

class Lineynoe(Uravnenie):  
def **init**(self, a, b):  
self.a = a  
self.b = b

def reshit(self):  
return -self.b / self.a

def info(self):  
print(f"Линейное уравнение: {self.a}x + {self.b} = 0, корень: {self.reshit()}")

class Kvadratnoe(Uravnenie):  
def **init**(self, a, b, c):  
self.a = a  
self.b = b  
self.c = c

def reshit(self):  
import math  
d = self.b \*\* 2 - 4 \* self.a \* self.c  
if d > 0:  
x1 = (-self.b + math.sqrt(d)) / (2 \* self.a)  
x2 = (-self.b - math.sqrt(d)) / (2 \* self.a)  
return x1, x2  
elif d == 0:  
return -self.b / (2 \* self.a)  
else:  
return None

def info(self):  
korni = self.reshit()  
if korni:  
if isinstance(korni, tuple):  
print(f"Квадратное уравнение: {self.a}x^2 + {self.b}x + {self.c} = 0, корни: {korni[0]} и {korni[1]}")  
else:  
print(f"Квадратное уравнение: {self.a}x^2 + {self.b}x + {self.c} = 0, корень: {korni}")  
else:  
print(f"Квадратное уравнение: {self.a}x^2 + {self.b}x + {self.c} = 0, корней нет")

# **Список уравнений**

uravneniya = [Lineynoe(2, 3), Kvadratnoe(1, -5, 6)]

# **Вывод информации о каждом уравнении**

for uravnenie in uravneniya:  
uravnenie.info()

#### **Задача 14. 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 n валютных денежных сумм и выведите полную информацию о них на экран.

**Таблица данных**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Класс** | **Имя** | **Смысл** | **Тип, структура** | **Ограничения на значения** |
| Входные данные | сумма | Сумма денег в валюте | вещественное | Сумма > 0 |
| Выходные данные | сумма в рублях | Переведенная сумма | вещественное | Сумма > 0 |

**Тесты**:

|  |  |
| --- | --- |
| **Входные данные** | **Ожидаемый результат** |
| Доллары: 100 | Перевод в рубли: 7300 (по курсу 73 рубля за доллар) |
| Евро: 50 | Перевод в рубли: 4350 (по курсу 87 рублей за евро) |

**Листинг программы (текстом)**:

class Valyuta:  
def perevod\_v\_rubli(self):  
pass

def info(self):  
pass

class Dollar(Valyuta):  
def **init**(self, summa):  
self.summa = summa

def perevod\_v\_rubli(self):  
kurs\_dollara = 73  
return self.summa \* kurs\_dollara

def info(self):  
print(f"Доллары: {self.summa}, в рублях: {self.perevod\_v\_rubli()}")

class Evro(Valyuta):  
def **init**(self, summa):  
self.summa = summa

def perevod\_v\_rubli(self):  
kurs\_evro = 87  
return self.summa \* kurs\_evro

def info(self):  
print(f"Евро: {self.summa}, в рублях: {self.perevod\_v\_rubli()}")

# **Список валют**

valyuty = [Dollar(100), Evro(50)]

# **Вывод информации о валютах**

for valyuta in valyuty:  
valyuta.info()

#### **Задача 15. Условие.**

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

**Таблица данных**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Класс** | **Имя** | **Смысл** | **Тип, структура** | **Ограничения на значения** |
| Входные данные | a | Первый элемент прогрессии | вещественное | a ≠ 0 |
|  | d | Разность (для арифметической) | вещественное | Любое вещественное |
|  | r | Знаменатель (для геометрической) | вещественное | r ≠ 0 |
| Выходные данные | j-й элемент | Элемент прогрессии | вещественное | Не отрицательное |

**Тесты**:

|  |  |
| --- | --- |
| **Входные данные** | **Ожидаемый результат** |
| Арифметическая прогрессия: a = 2, d = 3, j = 4 | 11 |
| Геометрическая прогрессия: a = 2, r = 2, j = 3 | 8 |

**Листинг программы (текстом)**:

class Progressiya:  
def **init**(self, a):  
self.a = a

def j\_element(self, j):  
pass

def summa(self, n):  
pass

def info(self):  
pass

class Arifmeticheskaya(Progressiya):  
def **init**(self, a, d):  
super().**init**(a)  
self.d = d

def j\_element(self, j):  
return self.a + (j - 1) \* self.d

def summa(self, n):  
return (n / 2) \* (2 \* self.a + (n - 1) \* self.d)

def info(self):  
print(f"Арифметическая прогрессия: первый элемент = {self.a}, разность = {self.d}, сумма первых n элементов = {self.summa(5)}")

class Geometricheskaya(Progressiya):  
def **init**(self, a, r):  
super().**init**(a)  
self.r = r

def j\_element(self, j):  
return self.a \* self.r \*\* (j - 1)

def summa(self, n):  
if self.r != 1:  
return self.a \* (1 - self.r \*\* n) / (1 - self.r)  
else:  
return self.a \* n

def info(self):  
print(f"Геометрическая прогрессия: первый элемент = {self.a}, знаменатель = {self.r}, сумма первых n элементов = {self.summa(5)}")

# **Список прогрессий**

progressii = [Arifmeticheskaya(2, 3), Geometricheskaya(2, 2)]

# **Вывод информации о прогрессиях**

for progressiya in progressii:  
progressiya.info()