

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение  
высшего образования  
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии  
Департамент цифровых, робототехнических систем и электроники

**ОТЧЕТ**  
**ПО ЛАБОРАТОРНОЙ РАБОТЕ №1**  
**дисциплины «Объектно-ориентированное программирование»**  
**Вариант \_\_\_\_**

Выполнил:  
Епифанов Алексей Александрович  
3 курс, группа ИВТ-б-о-22-1,  
09.03.01 «Информатика и  
вычислительная техника»,  
направленность (профиль)  
«Программное обеспечение средств  
вычислительной  
техники и автоматизированных систем  
», очная форма обучения

---

(подпись)

Проверил:  
Воронкин Роман Александрович,  
доцент департамента цифровых,  
робототехнических систем и  
электроники

---

(подпись)

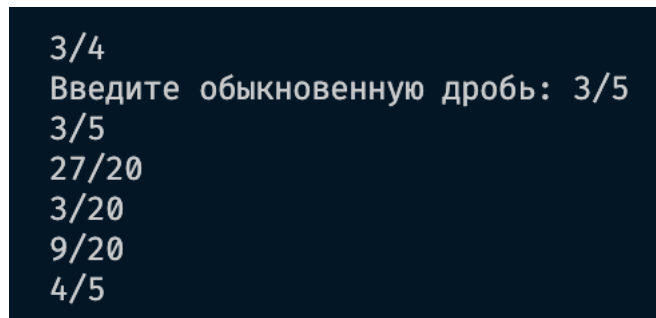
Отчет защищен с оценкой \_\_\_\_\_ Дата защиты \_\_\_\_\_  
Ставрополь, 2024 г.

Тема: элементы объектно-ориентированного программирования в языке Python.

Цель: приобретение навыков по работе с классами и объектами при написании программ с помощью языка программирования Python версии 3.x.

Порядок выполнения работы:

1. Создал новый репозиторий, клонировал его, в нем создал ветку developer и перешел на нее.
2. Проработал пример:

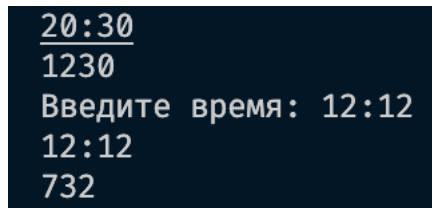


```
3/4
Введите обыкновенную дробь: 3/5
3/5
27/20
3/20
9/20
4/5
```

Рисунок 1. Результат работы примера

3. Выполнил индивидуальное задание 1 вариант 9: Парой называется класс с двумя полями, которые обычно имеют имена first и second. Требуется реализовать тип данных с помощью такого класса. Во всех заданиях обязательно должны присутствовать: метод инициализации `__init__`; метод должен контролировать значения аргументов на корректность; ввод с клавиатуры `read`; вывод на экран `display`. Реализовать внешнюю функцию с именем `make_тип()`, где `тип` — тип реализуемой структуры. Функция должна получать в качестве аргументов значения для полей структуры и возвращать структуру требуемого типа. При передаче ошибочных параметров следует выводить сообщение и заканчивать работу.

Поле `first` — целое положительное число, часы; поле `second` — целое положительное число, минуты. Реализовать метод `minutes()` — приведение времени в минуты.



```
20:30
1230
Введите время: 12:12
12:12
732
```

Рисунок 2. Результат работы индивидуального задания 1

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
class Time:
    def __init__(self, first=0, second=0):
        self.__first = first
        self.__second = second
    def read(self, prompt=None):
        line = input() if prompt is None else input(prompt)
        parts = list(map(int, line.split(":")))
        if len(parts) != 2:
            raise ValueError("Неправильный формат времени")
        self.__first, self.__second = parts
    def display(self):
        print(f"{self.__first}:{self.__second:02}")
    def minutes(self):
        return self.__first * 60 + self.__second
def make_time(hours=0, minutes=0):
    if not (0 <= hours < 24 and isinstance(hours, int)):
        raise ValueError("Неправильное количество часов")
    if not (0 <= minutes < 60 and isinstance(minutes, int)):
        raise ValueError("Неправильное количество минут")
    return Time(hours, minutes)
if __name__ == "__main__":
    t1 = make_time(20, 30)
    t1.display()
    print(t1.minutes())
    t2 = make_time()
    t2.read("Введите время: ")
    t2.display()
    print(t2.minutes())
```

4. Выполнил индивидуальное задание 2: составить программу с использованием классов и объектов для решения задачи. Во всех заданиях, помимо указанных в задании операций, обязательно должны быть реализованы следующие методы: метод инициализации `__init__`; ввод с клавиатуры `read`; вывод на экран `display`.

Реализовать класс `Account`, представляющий собой банковский счет. В классе должны быть четыре поля: фамилия владельца, номер счета, процент начисления и сумма в рублях. Открытие нового счета выполняется операцией инициализации. Необходимо выполнять следующие операции: сменить владельца счета, снять некоторую сумму денег со счета, положить деньги на

счет, начислить проценты, перевести сумму в доллары, перевести сумму в евро, получить сумму прописью (преобразовать в числительное).

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
class AccountOwner:
    def __init__(self, owner: str):
        self.__owner = owner
    def read(self):
        self.__owner = input("Введите владельца счета: ")
    @property
    def owner(self) -> str:
        return self.__owner
    @owner.setter
    def owner(self, new_owner: str):
        self.__owner = new_owner
class AccountBalance:
    def __init__(self, balance: float):
        self.__balance = balance
    @property
    def balance(self):
        return self.__balance
    @balance.setter
    def balance(self, new_balance: float):
        self.__balance = new_balance
class BalanceOperations:
    def __init__(self, acc_balance: AccountBalance):
        self.__acc_balance = acc_balance
    def withdraw(self, amount: float):
        if amount > self.__acc_balance.balance:
            print("Недостаточно средств на счете.")
        else:
            self.__acc_balance.balance -= amount
    def deposit(self, amount: float):
        self.__acc_balance.balance += amount
class InterestConverter:
    def __init__(self, acc_balance: AccountBalance, interest_rate: float):
        self.__acc_balance = acc_balance
        self.__interest_rate = interest_rate
    def add_interest(self):
        self.__acc_balance.balance += (
            self.__acc_balance.balance * self.__interest_rate
        )
class CurrencyConverter:
    def __init__(
        self,
        acc_balance: AccountBalance,
        rub_dollar_rate: float,
        rub_eur_rate: float,
    ):
        self.__acc_balance = acc_balance
        self.__rub_dollar_rate = rub_dollar_rate
        self.__rub_eur_rate = rub_eur_rate
    def convert_to_usd(self):
        return self.__acc_balance.balance * self.__rub_dollar_rate
    def convert_to_eur(self):
        return self.__acc_balance.balance * self.__rub_eur_rate
class AmountInWords:
    def __init__(self, acc_balance: AccountBalance):
        self.__acc_balance = acc_balance
    def amount_in_words(self) -> str | None:
        # Реализация преобразования суммы в числительное
```

```

sl_n = [
    {
        0: "ноль",
        1: "один",
        2: "два",
        3: "три",
        4: "четыре",
        5: "пять",
        6: "шесть",
        7: "семь",
        8: "восемь",
        9: "девять",
    },
    {
        1: "десять",
        2: "двадцать",
        3: "тридцать",
        4: "сорок",
        5: "пятьдесят",
        6: "шестьдесят",
        7: "семьдесят",
        8: "восемьдесят",
        9: "девяносто",
    },
    {
        1: "сто",
        2: "двести",
        3: "триста",
        4: "четыреста",
        5: "пятьсот",
        6: "шестьсот",
        7: "семьсот",
        8: "восемьсот",
        9: "девятьсот",
    },
    {
        1: "тысяча",
        2: "две тысячи",
        3: "три тысячи",
        4: "четыре тысячи",
        5: "пять тысяч",
        6: "шесть тысяч",
        7: "семь тысяч",
        8: "восемь тысяч",
        9: "девять тысяч",
    },
    {
        1: "одиннадцать",
        2: "двенадцать",
        3: "тринадцать",
        4: "четырнадцать",
        5: "пятнадцать",
        6: "шестнадцать",
        7: "семнадцать",
        8: "восемнадцать",
        9: "девятнадцать",
    },
]
bal = list(map(int, str(int(self.__acc_balance.balance))))
bal.reverse()
list_bal = []
if len(bal) == 1:
    str_bal = sl_n[bal[0]]

```

```

elif len(bal) < 5:
    prew = 0
    for count, i in enumerate(bal):
        if (count == 1) and (i == 1) and (prew != 0):
            list_bal[0] = sl_n[-1][prew]
        else:
            val = sl_n[count].get(i, None)
            if val:
                list_bal.append(val)
            prew = i
    list_bal.reverse()
    str_bal = " ".join(list_bal)
else:
    print("Сумма больше 99999")
    return None
return str_bal

class AccountStorage:
    def __init__(
        self,
        acc_owner: AccountOwner,
        account_number: int,
        acc_balance: AccountBalance,
    ):
        self.__acc_owner = acc_owner
        self.__account_number = account_number
        self.__acc_balance = acc_balance
    def display(self):
        print(f"Владелец счета: {self.__acc_owner.owner}")
        print(f"Номер счета: {self.__account_number}")
        print(f"Баланс: {self.__acc_balance.balance} руб")

class Account:
    def __init__(
        self,
        owner: str,
        account_number: int,
        interest_rate: float,
        balance: float,
        rub_dollar_rate: float,
        rub_eur_rate: float,
    ):
        # Создаем объекты, которые нужны для работы с аккаунтом
        self.account_owner = AccountOwner(owner)
        self.account_balance = AccountBalance(balance)
        self.balance_operations = BalanceOperations(self.account_balance)
        self.interest_converter = InterestConverter(
            self.account_balance, interest_rate
        )
        self.currency_converter = CurrencyConverter(
            self.account_balance, rub_dollar_rate, rub_eur_rate
        )
        self.am_in_words = AmountInWords(self.account_balance)
        self.account_storage = AccountStorage(
            self.account_owner, account_number, self.account_balance
        )
    # Методы для управления аккаунтом
    def display(self):
        self.account_storage.display()
    def change_owner(self, new_owner: str):
        self.account_owner.owner = new_owner
    def withdraw(self, amount: float):
        self.balance_operations.withdraw(amount)
    def deposit(self, amount: float):
        self.balance_operations.deposit(amount)

```

```

def add_interest(self):
    self.interest_converter.add_interest()
def convert_to_usd(self):
    return self.currency_converter.convert_to_usd()
def convert_to_eur(self):
    return self.currency_converter.convert_to_eur()
def amount_in_words(self):
    return self.am_in_words.amount_in_words()
def change_currency_converter(
    self, rub_dollar_rate: float, rub_eur_rate: float
):
    self.currency_converter = CurrencyConverter(
        self.account_balance, rub_dollar_rate, rub_eur_rate
    )
def change_interest_converter(self, interest_rate: float):
    self.interest_converter = InterestConverter(
        self.account_balance, interest_rate
    )
def read_owner(self):
    self.account_owner.read()
# Демонстрация возможностей класса
if __name__ == "__main__":
    rub_dollar_rate = 80
    rub_eur_rate = 90
    my_account = Account(
        "Иванов", "10032", 0.05, 9045, rub_dollar_rate, rub_eur_rate
    )
    my_account.display()
    my_account.change_owner("Петров")
    my_account.deposit(500)
    my_account.withdraw(300)
    my_account.add_interest()
    print("\nИзменённый счет:\n")
    my_account.display()
    usd_amount = my_account.convert_to_usd()
    print(f"Баланс в USD: ${usd_amount}")
    eur_amount = my_account.convert_to_eur()
    print(f"Баланс в EUR: €{eur_amount}")
    word = my_account.amount_in_words()
    print(f"Округленная сумма в рублях: {word}")
    my_account.read_owner()
    print()
    my_account.display()

```

### Ответы на контрольные вопросы:

1. Как осуществляется объявление класса в языке Python?

Класс объявляется с помощью ключевого слова `class`, за которым следует имя класса.

2. Чем атрибуты класса отличаются от атрибутов экземпляра?

Атрибуты класса общие для всех экземпляров класса, а атрибуты экземпляра принадлежат конкретному объекту и могут быть уникальными для каждого экземпляра.

3. Каково назначение методов класса?

Методы класса выполняют действия или операции, связанные с объектами класса, и могут изменять атрибуты экземпляра или взаимодействовать с данными.

4. Для чего предназначен метод `__init__()` класса?

Метод `__init__()` используется для инициализации объекта при его создании, задавая начальные значения атрибутов экземпляра.

5. Каково назначение `self`?

`self` указывает на текущий экземпляр класса, позволяя обращаться к атрибутам и методам этого экземпляра внутри класса.

6. Как добавить атрибуты в класс?

Атрибуты экземпляров добавляются в любом методе через `self`: `self.attr = value` или вне класса для определенного объекта: `obj.new_attr = value`

Атрибуты класса создаются напрямую в теле класса:

```
class MyClass:
```

```
    class_attr = value
```

7. Как осуществляется управление доступом к методам и атрибутам в языке Python?

Атрибуты, начинающиеся с одного подчеркивания (`_attr`), считаются защищенными, но к ним можно получить доступ извне, если же атрибут начинается с двух подчеркиваний (`__attr`), то прямого доступа снаружи к нему не будет, но через `_ClassName__attr` к нему все еще можно будет получить доступ.

8. Каково назначение функции `isinstance()`?

Функция `isinstance()` проверяет, является ли объект экземпляром определенного класса или его подклассов.

Вывод: в результате выполнения работы были приобретены навыки по работе с классами и объектами при написании программ с помощью языка программирования Python версии 3.x.