

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 №4.1
дисциплины «Объектно-ориентированное программирование»
Вариант 2

Выполнил:
Иващенко Олег Андреевич
3 курс, группа ИВТ-б-о-22-1,
09.03.02 «Информационные и
вычислительные машины»,
направленность (профиль)
«Программное обеспечение средств
вычислительной техники и
автоматизированных систем»

(подпись)

Руководитель практики:
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:

Пример 1. Рациональная (несократимая) дробь представляется парой целых чисел (a, b), где a – числитель, b – знаменатель. Создать класс Rational для работы с рациональными дробями. Обязательно должны быть реализованы операции:

- сложения add, $(a, b) + (c, d) = (ad + bc, bd)$;
- вычитания sub, $(a, b) - (c, d) = (ad - bc, bd)$;
- умножения mul, $(a, b) \times (c, d) = (ac, bd)$;
- деления div, $(a, b) / (c, d) = (ad, bc)$;
- сравнения equal, greater, less.

Должна быть реализована приватная функция сокращения дроби reduce, которая обязательно вызывается при выполнении арифметических операций.

Листинг 1 – Код программы примера

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

class Rational:

    def __init__(self, a=0, b=1):
        a = int(a)
        b = int(b)

        if b == 0:
            raise ValueError()

        self.__numerator = abs(a)
        self.__denominator = abs(b)

        self.__reduce()

    # Сокращение дроби
```

```

def __reduce(self):
    # Функция для нахождения наибольшего общего делителя
    def gcd(a, b):
        if a == 0:
            return b
        elif b == 0:
            return a
        elif a >= b:
            return gcd(a % b, b)
        else:
            return gcd(a, b % a)

    c = gcd(self.__numerator, self.__denominator)

    self.__numerator //= c
    self.__denominator //= c

@property
def numerator(self):
    return self.__numerator

@property
def denominator(self):
    return self.__denominator

# Прочитать значение дроби с клавиатуры. Дробь вводится
# как a/b.
def read(self, prompt=None):
    line = input() if prompt is None else input(prompt)
    parts = list(map(int, line.split('/', maxsplit=1)))

    if parts[1] == 0:
        raise ValueError()

    self.__numerator = abs(parts[0])
    self.__denominator = abs(parts[1])

    self.__reduce()

# Вывести дробь на экран
def display(self):
    print(f"{self.__numerator}/{self.__denominator}")

# Сложение обыкновенных дробей.
def add(self, rhs):
    if isinstance(rhs, Rational):
        a = self.numerator * rhs.denominator + \
            self.denominator * rhs.numerator
        b = self.denominator * rhs.denominator
        return Rational(a, b)
    else:
        raise ValueError()

```

```

# Вычитание обыкновенных дробей.
def sub(self, rhs):
    if isinstance(rhs, Rational):
        a = self.numerator * rhs.denominator - \
            self.denominator * rhs.numerator
        b = self.denominator * rhs.denominator

        return Rational(a, b)
    else:
        raise ValueError()

# Умножение обыкновенных дробей.
def mul(self, rhs):
    if isinstance(rhs, Rational):
        a = self.numerator * rhs.numerator
        b = self.denominator * rhs.denominator

        return Rational(a, b)
    else:
        raise ValueError()

# Деление обыкновенных дробей.
def div(self, rhs):
    if isinstance(rhs, Rational):
        a = self.numerator * rhs.denominator
        b = self.denominator * rhs.numerator

        return Rational(a, b)
    else:
        raise ValueError()

# Отношение обыкновенных дробей.
def equals(self, rhs):
    if isinstance(rhs, Rational):
        return (self.numerator == rhs.numerator) and \
            (self.denominator == rhs.denominator)
    else:
        return False

def greater(self, rhs):
    if isinstance(rhs, Rational):
        v1 = self.numerator / self.denominator
        v2 = rhs.numerator / rhs.denominator

        return v1 > v2
    else:
        return False

def less(self, rhs):
    if isinstance(rhs, Rational):

```

```

        v1 = self.numerator / self.denominator
        v2 = rhs.numerator / rhs.denominator

        return v1 < v2
    else:
        return False

if __name__ == '__main__':
    r1 = Rational(3, 4)
    r1.display()

    r2 = Rational()
    r2.read("Введите обыкновенную дробь: ")
    r2.display()

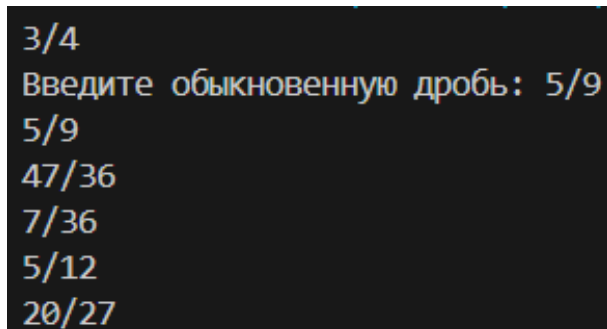
    r3 = r2.add(r1)
    r3.display()

    r4 = r2.sub(r1)
    r4.display()

    r5 = r2.mul(r1)
    r5.display()

    r6 = r2.div(r1)
    r6.display()

```



```

3/4
Введите обыкновенную дробь: 5/9
5/9
47/36
7/36
5/12
20/27

```

Рисунок 1.1 – Результат выполнения программы

Индивидуальное задание 1. Поле `first` — целое положительное число, часы; поле `second` — целое положительное число, минуты. Реализовать метод `minutes()` — приведение времени в минуты.

Листинг 2 – Код программы индивидуального задания 1

```

#!/usr/bin/env python3
# -*- coding: utf-8 -*-

'''

```

...

```
def __init__(self, first=0, second=0):
```

...

...

```
print("Итоговое количество минут:", self.minutes(first, second))
```

...

111

...

...

```

return False

```

...

```
exit(1)
```

111

111

```
input("Введите количество минут: "))
```

```
time.__init__
```

```
Введите количество часов: 8
Введите количество минут: 12
Итоговое количество минут: 492
```

Рисунок 2.1 – Результат выполнения программы

Индивидуальное задание 2. 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).

Листинг 3 – Код программы индивидуального задания 2

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

"""
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).
"""

class Account:

    def __init__(self, surname, number, percent, cash):
        'Метод инициализации необходимых атрибутов класса'

        self.surname = str(surname)
        self.number = str(number)
        self.percent = float(percent)
        self.cash = float(cash)

    def change_owner(self, newOwner):
```

'Метод смены владельца счёта'

self.surname = newOwner

def display(self):

'Метод вывода информации о созданном счёте'

print(f"{self.surname} | {self.number} | {self.percent} |",
f"{self.cash}")

class AccountManagement:

def withdraw_money(self, money, account: Account):

'Метод для снятия денежных средств со счёта аккаунта'

if account.cash >= int(money):

account.cash -= int(money)

else:

print("На счету недостаточная сумма для выполнения операции")

def put_money(self, money, account: Account):

'Метод пополнения денежных средств на аккаунте'

account.cash += int(money)

class InterestHandler:

def __init__(self, account: Account):

'Метод инициализации'

self.percent = account.percent

def accrue_percent(self, account: Account):

account.cash += account.cash * self.percent / 100

class Converter:

def transfer_dollar(self, account: Account):

'Метод перевода валюты на счёте аккаунта в доллары'

print(f'Перевод в доллары: \${round(account.cash / 90, 2)}')

def transfer_euro(self, account: Account):

'Метод перевода валюты на счёте аккаунта в евро'

print(f'Перевод в евро: €{round(account.cash / 100, 2)}')

def transfer_text(self, account: Account):

'Метод перевода валюты в текстовый формат'

text_number = round(account.cash)

result = ""

if text_number > 10000:

print(f'Сумма больше 10.000 ({account.cash})')


```

return
else:
    units = ["", "один", "два", "три", "четыре", "пять", "шесть",
             "семь", "восемь", "девять"]
    teens = ["", "десять", "одиннадцать", "двенадцать", "тринадцать",
             "четырнадцать", "пятнадцать", "шестнадцать",
             "семнадцать", "восемнадцать", "девятнадцать"]
    tens = ["", "десять", "двадцать", "тридцать", "сорок",
            "пятьдесят", "шестьдесят", "семьдесят", "восемьдесят",
            "девяносто"]
    hunders = ["", "сто", "двести", "триста", "четыреста", "пятьсот",
              "шестьсот", "семьсот", "восемьсот", "десятьсот"]

    if text_number >= 1000:
        thousand = text_number // 1000
        if thousand == 1:
            result += "одна тысяча "
        elif thousand == 2:
            result += "две тысячи "
        elif thousand == 3 or thousand == 4:
            result += f"{units[thousand]} тысячи"
        else:
            result += f"{units[thousand]} тысяч"

        text_number -= thousand * 1000

    if text_number >= 100 and text_number <= 999:
        hunder = text_number // 100
        result += f"{hunders[hunder]}"

        text_number -= hunder * 100

    if text_number >= 10 and text_number <= 19:
        result += f"{teens[text_number - 11]}"
    elif text_number >= 20 and text_number <= 99:
        ten = text_number // 10
        result += f"{tens[ten]}"
        text_number -= ten * 10

    if text_number > 0 and text_number <= 9:
        result += f"{units[text_number]}"

    if text_number == 0 or (text_number >= 5 and text_number <= 9):
        result += " рублей"
    elif text_number == 1:
        result += " рубль"
    elif text_number >= 2 and text_number <= 4:
        result += " рубля"

    print(f'Сумма прописью: {result} (₽{account.cash})')

if __name__ == "__main__":
    'Метод инициализации'

    account = Account("Иванов", "333-123", 20, 10000)
    account.display()

```

```

account.change_owner("Петров")
account.display()

account_manager = AccountManagement()
account_manager.withdraw_money(5000, account)
account.display()
account_manager.put_money(557, account)
account.display()

interest = InterestHandler(account)
interest accrue_percent(account)
account.display()

converter = Converter()
converter.transfer_dollar(account)
converter.transfer_euro(account)
converter.transfer_text(account)

```

```

Иванов | 333-123 | 20.0 | 10000.0
Петров | 333-123 | 20.0 | 10000.0
Петров | 333-123 | 20.0 | 5000.0
Петров | 333-123 | 20.0 | 5557.0
Петров | 333-123 | 20.0 | 6668.4
Перевод в доллары: $74.09
Перевод в евро: €66.68
Сумма прописью: шесть тысяч шестьсот шестьдесят восемь рублей (₽6668.4)

```

Рисунок 3.1 – Результаты выполнения программы

Ответы на контрольные вопросы:

1. Как осуществляется объявление класса в языке Python?

Классы объявляются с помощью ключевого слова `class` и имени класса:

```
class MyClass:
```

```
    var = ... # некоторая переменная
```

```
    def do_smt(self):
```

```
        # какой-то метод
```

2. Чем атрибуты класса отличаются от атрибутов экземпляра?

Атрибут класса – это атрибут, общий для всех экземпляров класса. Они определены внутри класса, но вне каких-либо методов. Их значения

одинаковы для всех экземпляров класса, так что их можно рассматривать как тип значений по умолчанию для всех объектов.

Атрибуты экземпляра определяются в методах и хранят информацию, специфичную для экземпляра. Атрибуты экземпляра доступны только из области видимости объекта. Атрибуты экземпляра, естественно, используются для различения объектов: их значения различны для разных экземпляров.

3. Каково назначение методов класса?

Методы определяют функциональность объектов, принадлежащих конкретному классу.

Хотя методы и функции похожи друг на друга, всё же есть некоторые различия между ними. Основное отличие заключается в том, что методы являются частью структуры ООП, а функции – нет.

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

Метод `__init__` является конструктором. Конструкторы – это концепция объектно-ориентированного программирования. Класс может иметь один и только один конструктор. Если `__init__` определён внутри класса, он автоматически вызывается при создании нового экземпляра.

5. Каково назначение `self`?

Аргумент `self` представляет конкретный экземпляр класса и позволяет получить доступ к его атрибутам и методам. Важно использовать параметр `self` внутри метода, если мы хотим сохранить значения экземпляра для последующего использования.

6. Как добавить атрибуты в класс?

Рассмотрим пример:

класс Book

class Book:

material= "paper"

cover = "paperback"

all_books = []

Этот класс имеет строковую переменную `material` со значением «paper», строковую переменную `cover` со значением «paperback» и пустой список в качестве атрибута `all_books`. Все эти переменных являются атрибутами класса и к ним можно получить доступ, используя точечную запись с именем класса:

Book.material # "paper"

Book.cover # "paperback"

Book.all_books # []

Переменные экземпляра хранят данные, уникальные для каждого объекта класса.

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

В Python нет возможности указания доступа к переменной, и любой может обратиться к атрибутам и методам класса, если возникнет такая необходимость. Это существенный недостаток языка Python, так как нарушается один из ключевых принципов ООП – инкапсуляция. Хорошим тоном считается, что для чтения/изменения какого-либо атрибута должны использоваться специфичные методы, которые называются `getter/setter`, из можно реализовать, но ничего не мешает изменить атрибут напрямую. При этом есть соглашение, что метод или атрибут, который начинается с нижнего подчёркивания, является скрытым, и снаружи класса его трогать не нужно.

8. Каково назначение функции `isinstance`?

Встроенная функция `isinstance(obj, cls)` позволяет узнать, что некоторый объект `obj` является либо экземпляром класса `cls`, либо экземпляром одного из потомков класса `cls`.

Выводы: В процессе выполнения лабораторной работы были приобретены навыки по работе с классами и объектами при написании программ с помощью языка программирования Python версии 3.x, проработан пример лабораторной работы и выполнены два индивидуальных задания.