

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 №2
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

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2024 г.

Тема: перегрузка операторов в языке Python

Цель: приобретение навыков по перегрузке операторов 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:

```
r1 = 3 / 4
r2 = 5 / 6
r1 + r2 = 19 / 12
r1 - r2 = -1 / 12
r1 * r2 = 5 / 8
r1 / r2 = 9 / 10
r1 == r2: False
r1 != r2: True
r1 > r2: False
r1 < r2: True
r1 >= r2: False
r1 <= r2: True
```

Рисунок 1. Результат работы примера

3. Выполнил индивидуальное задание 1: выполнить индивидуальное задание 2 лабораторной работы 4.1, максимально задействовав имеющиеся в Python средства перегрузки операторов.

```
> /Users/aleksejepifanov/Library/Caches/pypoetry/virtual
П/практика/OOP_2/program/ind_1.py
Владелец счета: Иванов
Номер счета: 10032
Текущая сумма на счете: 9045

Изменённый счет:

Владелец счета: Петров
Номер счета: 10032
Текущая сумма на счете: 9707.25
Баланс в USD: $776580.0
Баланс в EUR: €873652.5
Округленная сумма в рублях: девять тысяч семьсот семь
Введите владельца счета: Алексей

Владелец счета: Алексей
Номер счета: 10032
Текущая сумма на счете: 9707.25
```

Рисунок 2. Результат работы индивидуального задания 1

```
#!/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

    def __iadd__(self, rhs: float): # +=
        self.__balance += rhs
        return self

    def __isub__(self, rhs: float): # -=
        self.__balance -= rhs
        return self

    def __lt__(self, rhs: float): # <
        return self.__balance < rhs

    def __mul__(self, rhs: float): # *
        return self.__balance * rhs

    def __int__(self): # int
        return int(self.__balance)

    def __str__(self): # str
        return str(self.__balance)

```

```

class BalanceOperations:
    def __init__(self, acc_balance: AccountBalance):
        self.__acc_balance = acc_balance

    def withdraw(self, amount: float):
        if self.__acc_balance < amount:
            print("Недостаточно средств на счете.")
        else:
            self.__acc_balance -= amount

    def deposit(self, amount: float):
        self.__acc_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 += (
            self.__acc_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 * self.__rub_dollar_rate

    def convert_to_eur(self):
        return self.__acc_balance * self.__rub_eur_rate

```

```

class AmountInWords:
    def __init__(self, acc_balance: AccountBalance):
        self.__acc_balance = acc_balance

    def __str__(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))))
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 __str__(self):
        return f"Владелец счета: {self.__acc_owner.owner}\n" \
            f"Номер счета: {self.__account_number}\n" \
            f"Текущая сумма на счете: {self.__acc_balance}"

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):
        print(self.account_storage)

    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

    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()

```

4. Выполнил индивидуальное задание 2 вариант 9: дополнительно к требуемым в заданиях операциям перегрузить операцию индексирования []. Максимально возможный размер списка задать константой. В отдельном поле size должно храниться максимальное для данного объекта количество элементов списка; реализовать метод size(), возвращающий установленную длину. Если количество элементов списка изменяется во время работы, определить в классе поле count. Первоначальные значения size и count устанавливаются конструктором.

В тех задачах, где возможно, реализовать конструктор инициализации строкой.

Карточка иностранного слова представляет собой словарь, содержащую иностранное слово и его перевод. Для моделирования электронного словаря

иностранных слов реализовать класс Dictionary. Данный класс имеет поле-название словаря и содержит список словарей WordCard, представляющих собой карточки иностранного слова. Название словаря задается при создании нового словаря, но должна быть предоставлена возможность его изменения во время работы. Карточки добавляются в словарь и удаляются из него. Реализовать поиск определенного слова как отдельный метод. Аргументом операции индексирования должно быть иностранное слово. В словаре не должно быть карточек-дублей. Реализовать операции объединения, пересечения и вычитания словарей. При реализации должен создаваться новый словарь, а исходные словари не должны изменяться. При объединении новый словарь должен содержать без повторений все слова, содержащиеся в обоих словарях-операндах. При пересечении новый словарь должен состоять только из тех слов, которые имеются в обоих словарях-операндах. При вычитании новый словарь должен содержать слова первого словаря-операнда, отсутствующие во втором.

```
сем/ООП/практика/ООП_2/program/ind_2.py
Словарь 'Словарь 1': [cat: кот, bird: птица, dog: собака]
Словарь 'Словарь 1': [cat: кот, bird: птица]
None
Словарь 'Словарь 1 |(или) Словарь 2': [cat: кот, bird: птица, mouse: мышь]
Словарь 'Словарь 1 &(и) Словарь 2': [cat: кот]
Словарь 'Словарь 1 -(минус) Словарь 2': [bird: птица]
cat: кот
bird: птица
Словарь 'Словарь 3': [car: машина, airplane: самолет, ship: корабль]
Словарь 'Новый словарь 3': [car: машина, airplane: самолет, ship: корабль]
Количество карточек в словаре Новый словарь 3: 3
```

Рисунок 3. Результат работы индивидуального задания 2

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

class WordCard:
    """
    Карточка словаря.
    """

    def __init__(self, foreign_word: str, translation: str):
        self.foreign_word = foreign_word
        self.translation = translation

    def __eq__(self, other: str):
        return self.foreign_word == other
```



```
def __hash__(self):
    return hash(self.foreign_word)
```

```
def __repr__(self):
    return f"{self.foreign_word}: {self.translation}"
```

```
class Dictionary:
```

```
    """
```

```
    Класс словаря.
```

```
    """
```

```
    MAX_SIZE = 100
```

```
def __init__(self, name: str, init_string: str | None = None):
    self.name = name
    self.cards = set()
    self.count = 0
    self.size = self.MAX_SIZE
    if init_string:
        self.init_from_string(init_string)
```

```
def init_from_string(self, init_string: str):
    pairs = init_string.split(",")
    for pair in pairs:
        foreign_word, translation = pair.split(":")
        self.__setitem__(foreign_word.strip(), translation.strip())
```

```
def rename(self, new_name: str):
    self.name = new_name
```

```
def __add_card(self, word_card: WordCard):
    if self.count < self.size:
        self.cards.add(word_card)
        self.count += 1
    else:
        print(f"Словарь {self.name} переполнен")
```

```
def __remove_card(self, foreign_word: str):
    self.cards.remove(foreign_word)
    self.count -= 1
```

```
def __find_word(self, foreign_word: str) -> WordCard | None:
    for card in self.cards:
        if card.foreign_word == foreign_word:
            return card
    return None
```

```
def __getitem__(self, foreign_word: str):
    return self.__find_word(foreign_word)
```

```
def __setitem__(self, foreign_word: str, translation: str):
    card = self.__find_word(foreign_word)
    if card:
        card.translation = translation
    else:
        self.__add_card(WordCard(foreign_word, translation))
```

```
def __delitem__(self, foreign_word: str):
    card = self.__find_word(foreign_word)
    if card:
        self.__remove_card(foreign_word)
```

```

        else:
            print(f"Перевод слова {foreign_word} не найден")

    def __iter__(self):
        return iter(self.cards)

    def __str__(self):
        return f"Словарь '{self.name}': {list(self.cards)}"

    def __len__(self):
        return self.count

    def size(self):
        return self.size

class DictUtils:
    """
    Класс для работы с словарями.
    """

    @staticmethod
    def union(first: Dictionary, second: Dictionary) -> Dictionary:
        new_dict = Dictionary(f"{first.name} |(или) {second.name}")
        new_dict.cards = first.cards | second.cards
        return new_dict

    @staticmethod
    def intersection(first: Dictionary, second: Dictionary) -> Dictionary:
        new_dict = Dictionary(f"{first.name} &(и) {second.name}")
        new_dict.cards = first.cards & second.cards
        return new_dict

    @staticmethod
    def difference(first: Dictionary, second: Dictionary) -> Dictionary:
        new_dict = Dictionary(f"{first.name} -(минус) {second.name}")
        new_dict.cards = first.cards - second.cards
        return new_dict

if __name__ == "__main__":
    dict1 = Dictionary("Словарь 1")
    dict1["cat"] = "кот"
    dict1["dog"] = "собака"
    dict1["bird"] = "птица"

    dict2 = Dictionary("Словарь 2")
    dict2["cat"] = "кот"
    dict2["mouse"] = "мышь"
    print(dict1)
    del dict1["dog"]
    print(dict1)
    print(dict1["dog"])
    # Объединение
    ut = DictUtils
    union_dict = ut.union(dict1, dict2)
    print(union_dict)

    # Пересечение
    intersection_dict = ut.intersection(dict1, dict2)
    print(intersection_dict)

```

```
# Вычитание
difference_dict = ut.difference(dict1, dict2)
print(difference_dict)

for card in dict1:
    print(card)

dict3 = Dictionary(
    "Словарь 3",
    "car:машина, ship:корабль, airplane:самолет"
)
print(dict3)
dict3.rename("Новый словарь 3")
print(dict3)
print(f"Количество карточек в словаре {dict3.name}: {len(dict3)}")
```

Ответы на контрольные вопросы:

1. Какие средства существуют в Python для перегрузки операций?

Python можно перегружать операции с помощью специальных методов.

Эти методы начинаются и заканчиваются двумя символами подчеркивания (`__add__`, `__sub__`, и т.д.). Перегрузка позволяет изменять поведение стандартных операций для пользовательских объектов, таких как сложение, умножение, сравнение и другие.

2. Какие существуют методы для перегрузки арифметических операций и операций отношения в языке Python?

Перегрузка арифметических операторов

`__add__(self, other)` - сложение. $x + y$ вызывает `x.__add__(y)`.

`__sub__(self, other)` - вычитание ($x - y$).

`__mul__(self, other)` - умножение ($x * y$).

`__truediv__(self, other)` - деление (x / y).

`__floordiv__(self, other)` - целочисленное деление ($x // y$).

`__mod__(self, other)` - остаток от деления ($x \% y$).

`__divmod__(self, other)` - частное и остаток (`divmod(x, y)`).

`__pow__(self, other[, modulo])` - возведение в степень ($x ** y$, `pow(x, y[, modulo])`).

`__lshift__(self, other)` - битовый сдвиг влево ($x << y$).

`__rshift__(self, other)` - битовый сдвиг вправо ($x >> y$).

`__and__(self, other)` - битовое И ($x \& y$).

`__xor__(self, other)` - битовое исключающее или ($x \wedge y$).

`__or__(self, other)` - битовое ИЛИ ($x \mid y$).

`__radd__(self, other)`, `__rsub__(self, other)`, `__rmul__(self, other)`,
`__rtruediv__(self, other)`, `__rfloordiv__(self, other)`, `__rmod__(self, other)`,
`__rdivmod__(self, other)`, `__rpow__(self, other)`, `__rlshift__(self, other)`,
`__rrshift__(self, other)`, `__rand__(self, other)`, `__rxor__(self, other)`, `__ror__(self, other)` - делают то же самое, что и арифметические операторы, перечисленные выше, но для аргументов, находящихся справа, и только в случае, если для левого операнда не определён соответствующий метод.

`__iadd__(self, other)` - $+=$.

`__isub__(self, other)` - $-=$.

`__imul__(self, other)` - $*=$.

`__itruediv__(self, other)` - $/=$.

`__ifloordiv__(self, other)` - $//=$.

`__imod__(self, other)` - $\%=$.

`__ipow__(self, other[, modulo])` - $**=$.

`__ilshift__(self, other)` - $<<=$.

`__irshift__(self, other)` - $>>=$.

`__iand__(self, other)` - $\&=$.

`__ixor__(self, other)` - $\wedge=$.

`__ior__(self, other)` - $\mid=$.

`__neg__(self)` - унарный $-$.

`__pos__(self)` - унарный $+$.

`__abs__(self)` - модуль (`abs()`).

`__invert__(self)` - инверсия (\sim).

`__complex__(self)` - приведение к `complex`.

`__int__(self)` - приведение к `int`.

`__float__(self)` - приведение к `float`.

`__round__(self[, n])` - округление.

Перегрузка операторов отношения:

`__lt__(self, other)` - $x < y$ вызывает `x.__lt__(y)` .

`__le__(self, other)` - $x \leq y$ вызывает `x.__le__(y)` .

`__eq__(self, other)` - $x == y$ вызывает `x.__eq__(y)` .

`__ne__(self, other)` - $x != y$ вызывает `x.__ne__(y)` .

`__gt__(self, other)` - $x > y$ вызывает `x.__gt__(y)` .

`__ge__(self, other)` - $x \geq y$ вызывает `x.__ge__(y)` .

3. В каких случаях будут вызваны следующие методы: `__add__`, `__iadd__` и `__radd__`? Приведите примеры.

`__add__(self, other)` вызывается при обычной операции сложения объектов ($a+b$).

`__iadd__(self, other)` вызывается при операции сокращённого сложения с присваиванием ($a += b$).

`__radd__(self, other)` вызывается, если левый операнд не поддерживает операцию сложения, а правый операнд пытается выполнить операцию через $+$. Например, если операция $other + self$ не реализована для объекта `other`, будет вызван метод `self.__radd__(other)`:

4. Для каких целей предназначен метод `__new__`? Чем он отличается от метода `__init__`?

Метод `__new__` предназначен для создания нового экземпляра класса. Это метод, который отвечает за выделение памяти для объекта и является первой стадией создания объекта. Его основная задача — вернуть новый экземпляр класса.

`__new__` используется в основном для создания экземпляров неизменяемых типов (например, `int`, `str`, `tuple`).

В отличие от метода `__init__`, который инициализирует уже существующий объект, `__new__` создаёт сам объект.

5. Чем отличаются методы `__str__` и `__repr__`?

`__str__(self)` предназначен для возвращения "человеко-читаемого" строкового представления объекта. Этот метод используется функцией `str()` и выводится, когда объект передается в функцию `print()`. Цель этого метода — предоставление удобного для восприятия пользователем описания объекта.

`__repr__(self)` должен возвращать строковое представление объекта, которое предназначено для программистов и должно быть максимально информативным. Этот метод используется функцией `repr()` и должен возвращать строку, которая при необходимости может быть использована для создания идентичного объекта (если это возможно). Если `__str__` не определен, используется `__repr__`.

Вывод: в ходе выполнения лабораторной работы были приобретены навыки по перегрузке операторов при написании программ с помощью языка программирования Python версии 3.x.