

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

Институт цифрового развития
Кафедра инфокоммуникаций

«Перегрузка операторов в языке Python»

ОТЧЕТ
по лабораторной работе №4.2
дисциплины
«Основы программной инженерии»

Выполнила:

Кувшин Ирина Анатольевна
2 курс, группа ПИЖ-б-о-21-1,
011.03.04 «Программная инженерия»,
направленность (профиль) «Разработка
и сопровождение программного
обеспечения», очная форма обучения

(подпись)

Проверил:

(подпись)

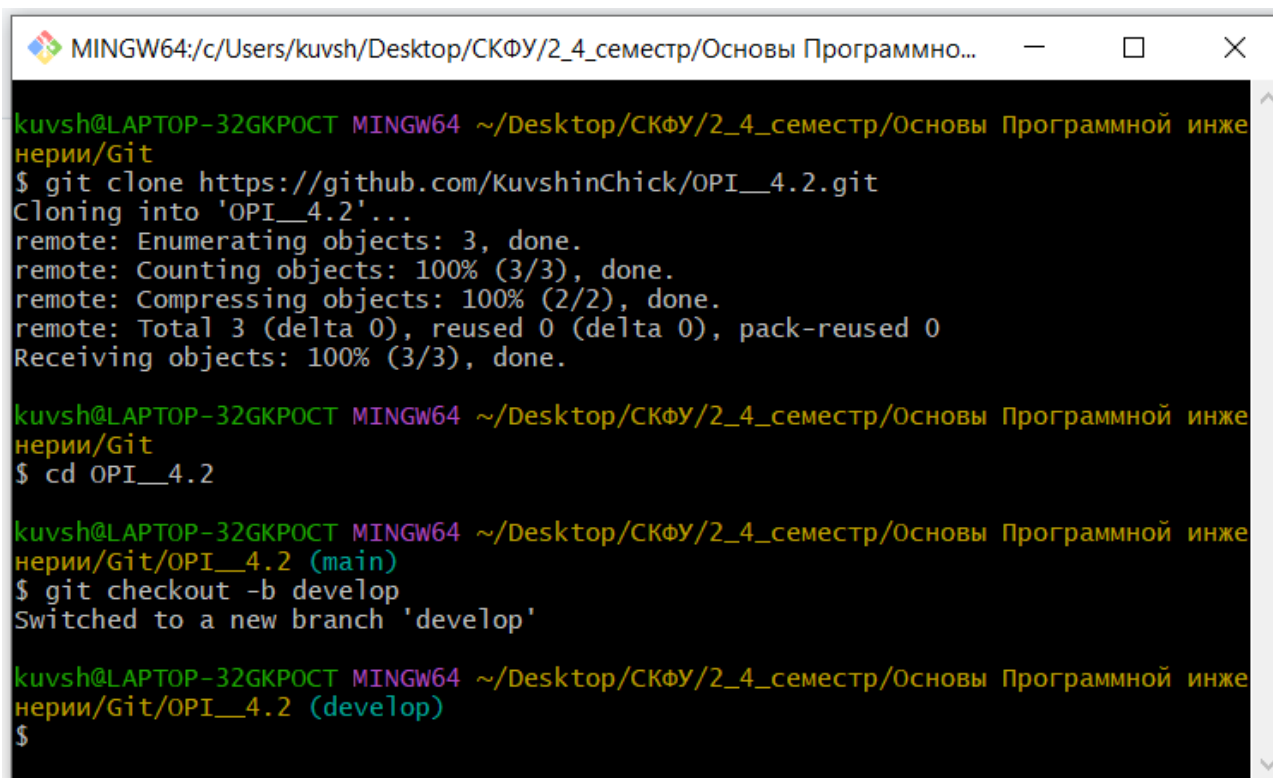
Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2023 г.

Цель работы: приобретение навыков по перегрузке операторов при написании программ с помощью языка программирования Python версии 3.x.

Ход работы:

1. Изучить теоретический материал работы.
2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python.
3. Выполните клонирование созданного репозитория.
4. Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm.
5. Организуйте свой репозиторий в соответствии с моделью ветвления git-flow.
6. Создайте проект PyCharm в папке репозитория.
7. Проработайте примеры лабораторной работы.



```
MINGW64/c:/Users/kuvsh/Desktop/СКФУ/2_4_семестр/Основы Программно...
kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git
$ git clone https://github.com/KuvshinChick/OPI__4.2.git
Cloning into 'OPI__4.2'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.

kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git
$ cd OPI__4.2

kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git/OPI__4.2 (main)
$ git checkout -b develop
Switched to a new branch 'develop'

kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git/OPI__4.2 (develop)
$
```

Рисунок 4.2.1 – Клонирование репозитория и создание ветки develop

```
MINGW64:/c:/Users/kuvsh/Desktop/СКФУ/2_4_семестр/Основы Программно...
kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инже
нерии/Git/OPI__4.2 (develop)
$ git add .

kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инже
нерии/Git/OPI__4.2 (develop)
$ git commit -m "modified .gitignore & readme"
[develop 7e894d6] modified .gitignore & readme
2 files changed, 134 insertions(+)
create mode 100644 .gitignore
create mode 100644 README.md

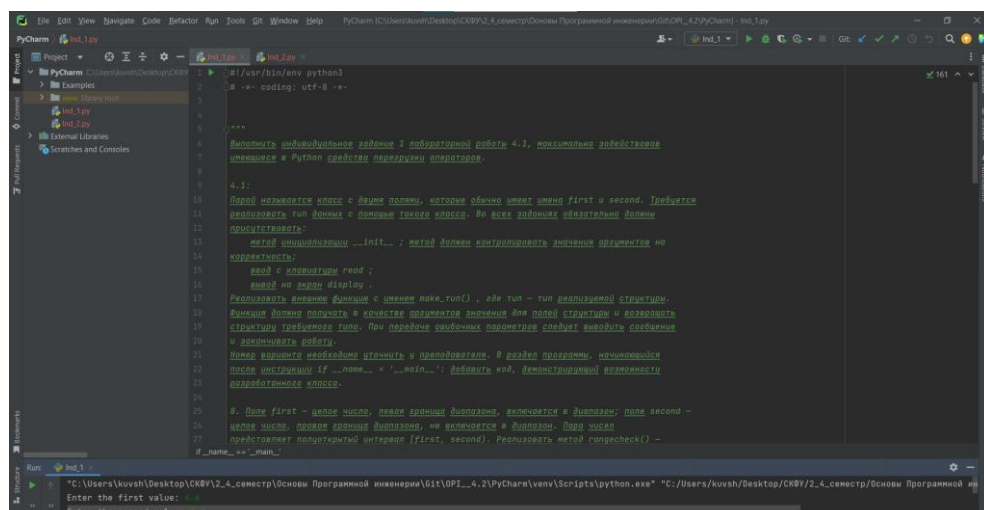
kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инже
нерии/Git/OPI__4.2 (develop)
$ git push origin develop
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 1.42 KiB | 1.42 MiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'develop' on GitHub by visiting:
```

Рисунок 4.2.2 – Обновление .gitignore и readme

8. Выполните индивидуальные задания. Приведите в отчете скриншоты работы программ решения индивидуального задания.

Задание 1

Выполнить индивидуальное задание 1 лабораторной работы 4.1, максимально задействовав имеющиеся в Python средства перегрузки операторов.



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

"""
Выполнить индивидуальное задание 1 лабораторной работы 4.1, максимально задействовав
имеющиеся в Python средства перегрузки операторов.
"""

class Calculator:
    """
    Класс Calculator с двумя параметрами, которые обычно имеют имена first и second. Требуется
    реализовать тип данных с помощью этого класса. Во всех заданиях обязательным условием
    является:
    - метод инициализации __init__ ; метод должен контролировать значения аргументов на
    корректность;
    - метод с клавиатуры read ;
    - метод с экраном display .
    Реализовать также функцию с именем main_run() , где run - тип реализованной структуры.
    Функция должна принимать в качестве аргументов значения для типа структуры и возвращать
    структуру требуемого типа. При передаче ошибочных параметров следует вывести сообщение
    и закончить работу.
    Намер верните необходимо уточнить в преподавателе. В раздле программы, начинающейся
    после инструкции if __name__ == '__main__': , добавить код, демонстрирующий возможности
    разработанного класса.
    """
    def __init__(self, first, second):
        self.first = first
        self.second = second

    def read(self):
        """
        Метод с клавиатуры read
        """
        pass

    def display(self):
        """
        Метод с экраном display
        """
        pass

    def main_run(self):
        """
        Метод с именем main_run
        """
        pass

if __name__ == '__main__':
    calc = Calculator(1, 2)
    calc.read()
    calc.display()
    calc.main_run()
```

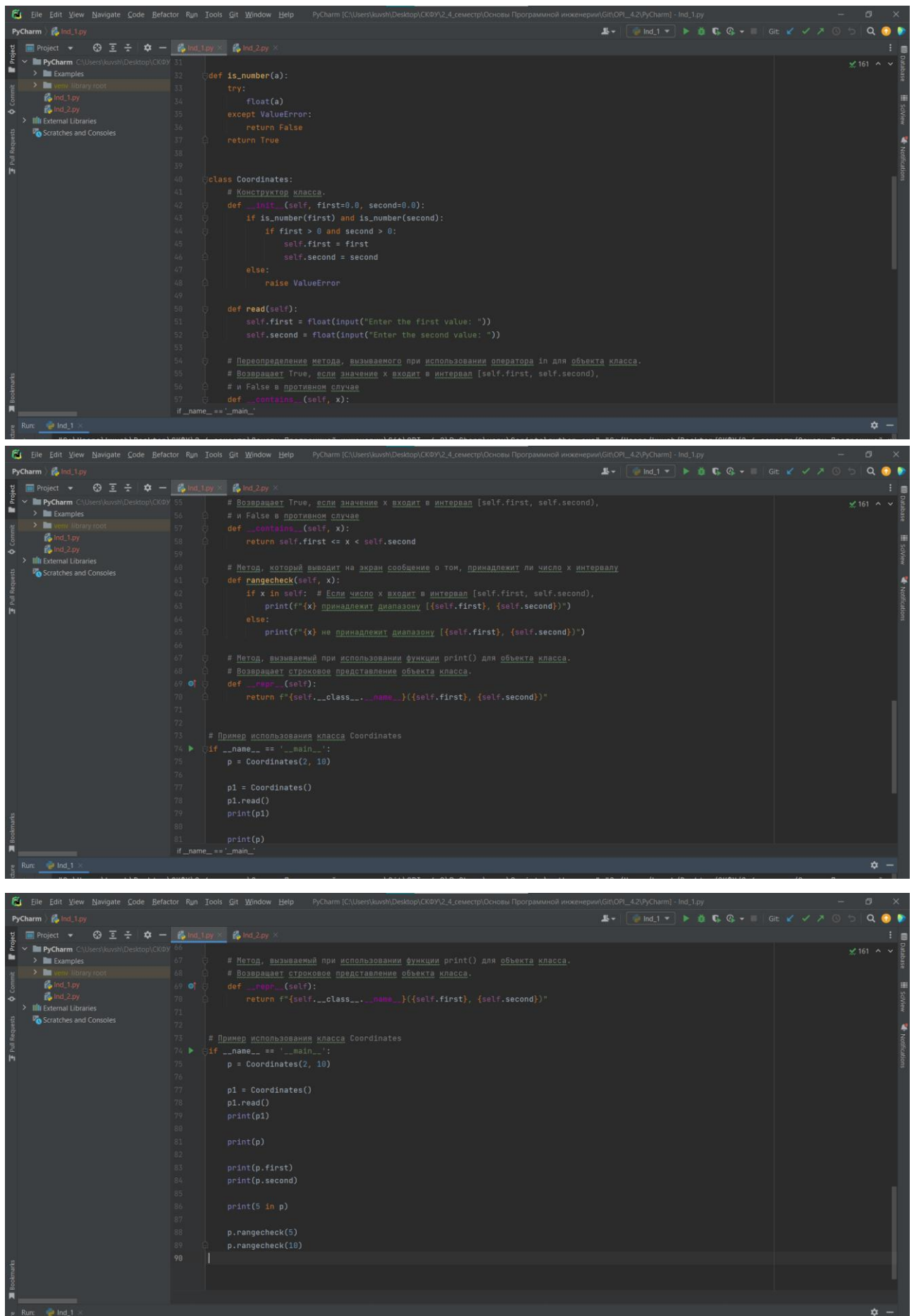
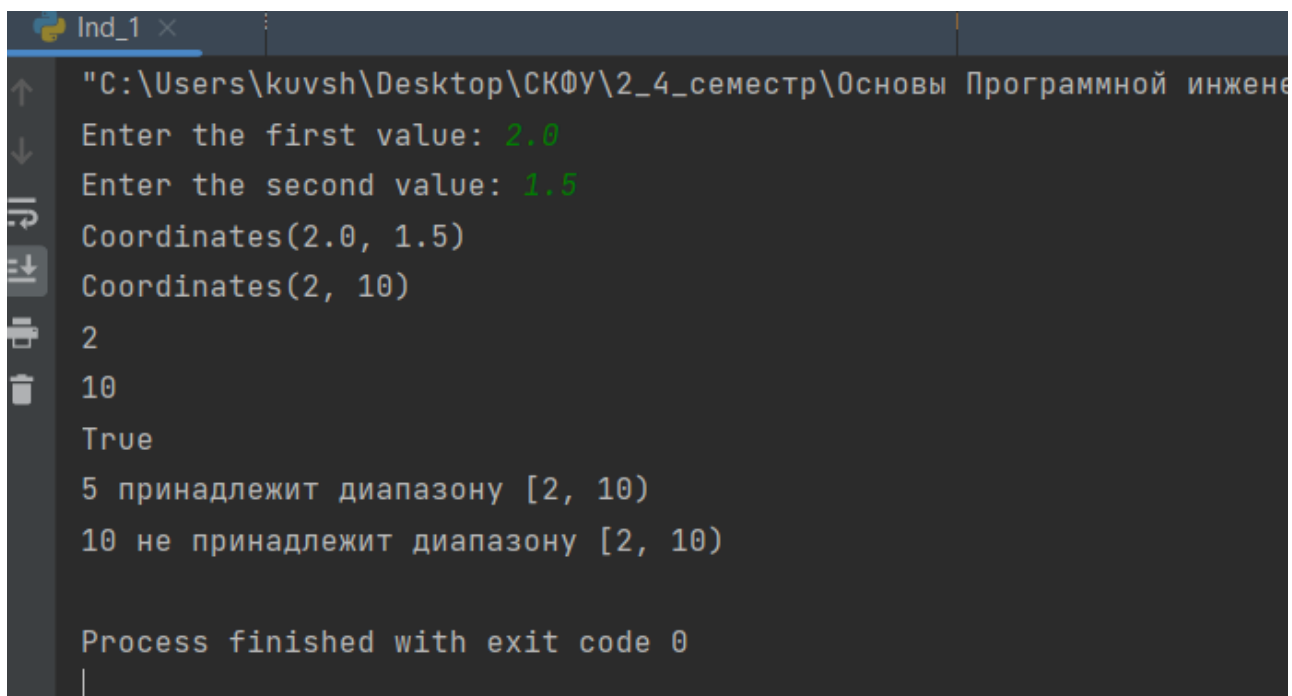


Рисунок 4.2.3 – Проработка программы

A screenshot of a Python IDE window titled 'Ind_1'. The console output shows the execution of a program. It starts with the file path 'C:\Users\kuvsh\Desktop\СКОУ\2_4_семестр\Основы Программной инженерии'. Then, it prompts for two values: 'Enter the first value: 2.0' and 'Enter the second value: 1.5'. These values are used in two function calls: 'Coordinates(2.0, 1.5)' and 'Coordinates(2, 10)'. The output then shows the values '2' and '10', followed by 'True'. Below this, two lines of Russian text are displayed: '5 принадлежит диапазону [2, 10)' and '10 не принадлежит диапазону [2, 10)'. Finally, the console shows 'Process finished with exit code 0'.

```
"C:\Users\kuvsh\Desktop\СКОУ\2_4_семестр\Основы Программной инженерии"
Enter the first value: 2.0
Enter the second value: 1.5
Coordinates(2.0, 1.5)
Coordinates(2, 10)
2
10
True
5 принадлежит диапазону [2, 10)
10 не принадлежит диапазону [2, 10)

Process finished with exit code 0
```

Рисунок 4.2.4 – Результат работы программы

Задание 2

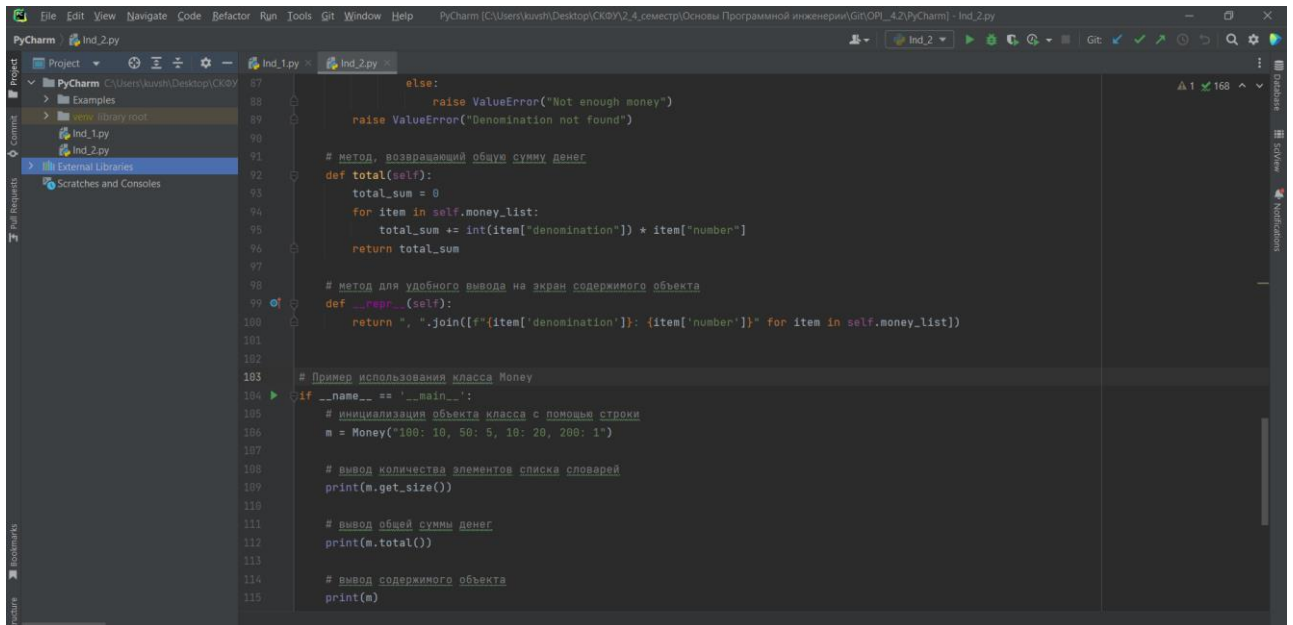
Дополнительно к требуемым в заданиях операциям перегрузить операцию индексирования []. Максимально возможный размер списка задать константой. В отдельном поле `size` должно храниться максимальное для данного объекта количество элементов списка; реализовать метод `size()`, возвращающий установленную длину.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

8. Реализовать класс `Money`, используя для представления суммы денег список словарей. Словарь имеет два ключа: номинал купюры и количество купюр данного достоинства. Номиналы представить как строку. Элемент списка словарей с меньшим индексом содержит меньший номинал.

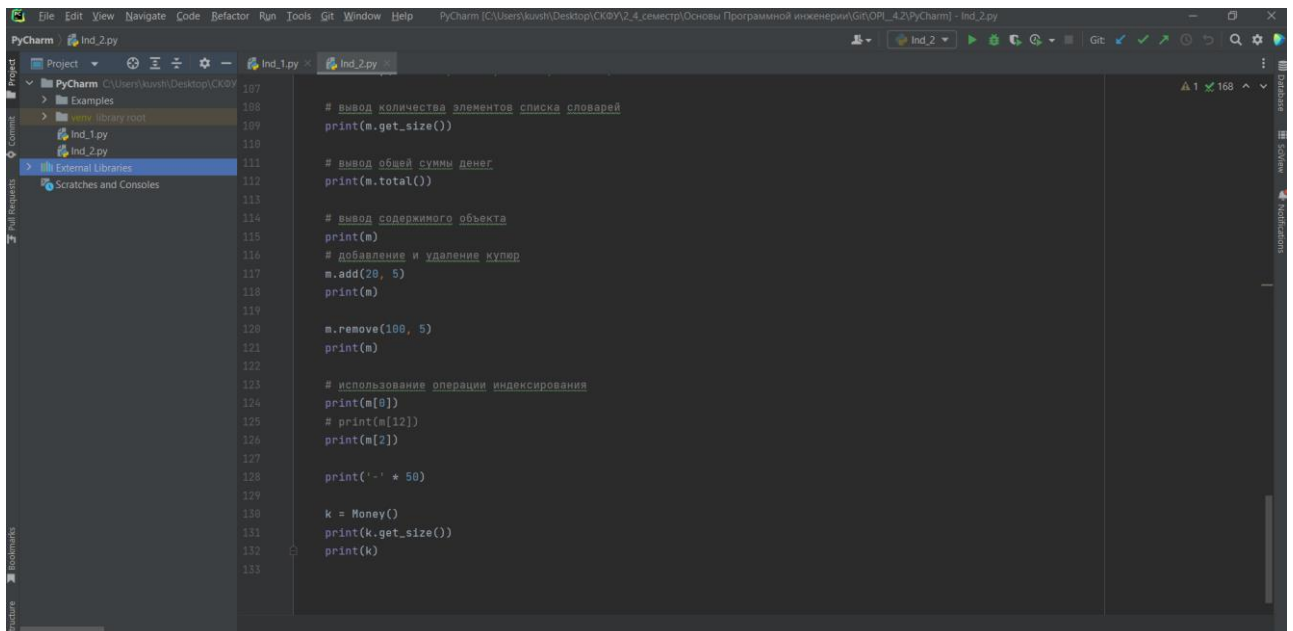
```
PyCharm [C:\Users\kuvsh\Desktop\CKF\V2_4_семестр\Основы Программной инженерии\Git\OPL_4.2\PyCharm] - Ind_2.py
Project: Ind_2.py
File Edit View Navigate Code Refactor Run Tools Git Window Help
Ind_1.py Ind_2.py
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4
5 """
6 Дополнить к требуем в заданиях операций переопределить операции индексирования [].
7 Максимально возможный размер списка задать константой. В отдельном поле size должно
8 храниться максимальное для данного объекта количество элементов списка; реализовать метод
9 size(), возвращающий установленную длину. Если количество элементов списка изменяется во
10 время работы, определить в классе поле count. Первоначальные значения size и count
11 устанавливаются конструктором.
12 В тех задачах, где возможно, реализовать конструктор инициализации строк.
13
14
15 8. Реализовать класс Money, используя для представления суммы денег список словарей.
16 Словарь имеет два ключа: номинал купюры и количество купюр данного достоинства.
17 Номиналы представить как строку. Элемент списка словарей с меньшим индексом
18 содержит меньший номинал.
19
20
21
22 class Money:
23     # константа для максимального размера списка словарей
24     MAX_SIZE = 10
25
26     # конструктор класса
27     def __init__(self, n=None):
28         self.count = 0
29         self.size = Money.MAX_SIZE
```

```
PyCharm [C:\Users\kuvsh\Desktop\CKF\V2_4_семестр\Основы Программной инженерии\Git\OPL_4.2\PyCharm] - Ind_2.py
Project: Ind_2.py
File Edit View Navigate Code Refactor Run Tools Git Window Help
Ind_1.py Ind_2.py
28 self.count = 0
29 self.size = Money.MAX_SIZE
30 self.money_list = [] # список словарей
31
32 # если передана строка, инициализируем список словарей соответствующим образом
33 # isinstance - позволяет проверить принадлежность экземпляра к классу.
34 if isinstance(n, str):
35     denominations = n.split(" ")
36     for d in denominations:
37         money = d.split(": ")
38         if money[0].isnumeric() and money[1].isnumeric():
39             denomination = int(money[0])
40             number = int(money[1])
41             if self.count < self.size:
42                 self.money_list.append({"denomination": str(denomination), "number": number})
43                 self.count += 1
44
45 # иначе инициализируем список словарей нулями
46 else:
47     for i in range(self.count):
48         self.money_list.append({"denomination": str(i), "number": 0})
49
50 # сортировка списка по номиналу
51 self.money_list = sorted(self.money_list, key=lambda d: int(d['denomination']))
52
53 # перегрузка операции индексирования
54 def __getitem__(self, index):
55     if 0 <= index <= self.count:
56         return self.money_list[index]
```

```
PyCharm [C:\Users\kuvsh\Desktop\CKF\V2_4_семестр\Основы Программной инженерии\Git\OPL_4.2\PyCharm] - Ind_2.py
Project: Ind_2.py
File Edit View Navigate Code Refactor Run Tools Git Window Help
Ind_1.py Ind_2.py
57 else:
58     raise IndexError("Index out of range")
59
60 # метод, возвращающий установленную длину
61 def get_size(self):
62     return self.count
63
64 # метод, добавляющий количество купюр заданного номинала
65 def add(self, denomination, number):
66     for item in self.money_list:
67         if item["denomination"] == str(denomination):
68             item["number"] += number
69             return
70 # если номинал не найден, добавляем словарь в список
71 if self.count < self.size:
72     self.money_list.append({"denomination": str(denomination), "number": number})
73     self.count += 1
74 else:
75     raise ValueError("List is full")
76
77 # метод, удаляющий количество купюр заданного номинала
78 def remove(self, denomination, number):
79     for item in self.money_list:
80         if item["denomination"] == str(denomination):
81             if item["number"] >= number:
82                 item["number"] -= number
83                 if item["number"] == 0:
84                     self.money_list.remove(item)
85                     self.count -= 1
```

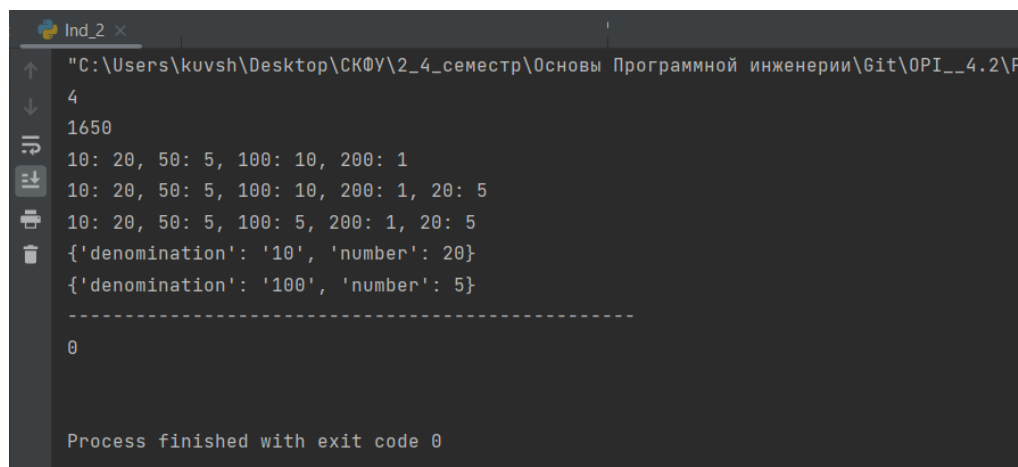


```
87         else:
88             raise ValueError("Not enough money")
89             raise ValueError("Denomination not found")
90
91     # метод, возвращающий общую сумму денег
92     def total(self):
93         total_sum = 0
94         for item in self.money_list:
95             total_sum += int(item["denomination"]) * item["number"]
96         return total_sum
97
98     # метод для удобного вывода на экран содержимого объекта
99     def __repr__(self):
100         return ", ".join([f'{item["denomination"]}: {item["number"]}' for item in self.money_list])
101
102
103     # Пример использования класса Money
104     if __name__ == '__main__':
105         # инициализация объекта класса с помощью строки
106         m = Money("100: 10, 50: 5, 10: 20, 200: 1")
107
108         # вывод количества элементов списка словарей
109         print(m.get_size())
110
111         # вывод общей суммы денег
112         print(m.total())
113
114         # вывод содержимого объекта
115         print(m)
```



```
107
108     # вывод количества элементов списка словарей
109     print(m.get_size())
110
111     # вывод общей суммы денег
112     print(m.total())
113
114     # вывод содержимого объекта
115     print(m)
116     # добавление и удаление купюр
117     m.add(20, 5)
118     print(m)
119
120     m.remove(100, 5)
121     print(m)
122
123     # использование операции индексирования
124     print(m[0])
125     # print(m[12])
126     print(m[2])
127
128     print('-' * 50)
129
130     k = Money()
131     print(k.get_size())
132     print(k)
133
```

Рисунок 4.2.5 – Проработка программы



```
Ind_2 x
"C:\Users\kuvsh\Desktop\СКФУ\2_4_семестр\Основы Программной инженерии\Git\OPI_4.2\F
4
1650
10: 20, 50: 5, 100: 10, 200: 1
10: 20, 50: 5, 100: 10, 200: 1, 20: 5
10: 20, 50: 5, 100: 5, 200: 1, 20: 5
{'denomination': '10', 'number': 20}
{'denomination': '100', 'number': 5}
-----
0
Process finished with exit code 0
```

Рисунок 4.2.6 – Результат работы программы

9. Зафиксируйте сделанные изменения в репозитории.
10. Выполните слияние ветки для разработки с веткой main (master).
11. Отправьте сделанные изменения на сервер GitHub.

Контрольные вопросы

1. Какие средства существуют в Python для перегрузки операций?

Для перегрузки операций в Python используются специальные методы, которые начинаются и заканчиваются двойным подчеркиванием. Например, для перегрузки оператора сложения используется метод `add`, для оператора равенства - метод `eq` и т.д.

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 ^ y`).
- `__or__(self, other)` - битовое ИЛИ (`x | y`).

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

Метод `add` вызывается при использовании оператора `+` для объектов данного класса. Метод `iadd` вызывается при использовании оператора `+=` для

того же класса. Метод `radd` вызывается, когда объект данного класса слева от оператора "+". Примеры:

```
class Number:
    def __init__(self, value):
        self.value = value

    def __add__(self, other):
        return Number(self.value + other.value)

    def __iadd__(self, other):
        self.value += other.value
        return self

    def __radd__(self, other):
        return Number(self.value + other)
```

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

Метод `new` предназначен для создания экземпляров класса, он возвращает новый объект класса. Метод `init` вызывается после создания экземпляра класса и выполняет инициализацию экземпляра. Отличие в том, что метод `new` создает объект, а `init` инициализирует его.

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

Можно сказать, что методы `repr()` и `__repr__` взаимозаменяемы. Функция `__str__` в Python делает то же самое, но ее поведение всё же немного отличается. Она предназначена для создания удобочитаемой версии, полезной для отслеживания или отображения информации об объекте. А метод `__repr__` предназначен для предоставления «официального» текстового образа объекта, который можно использовать для воссоздания этого объекта.