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 №4.2**

**дисциплины «Объектно-ориентированное программирование»**

**Вариант 9**

|  |  |  |  |
| --- | --- | --- | --- |
|  | | 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**: «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:

Пример. Изменить класс Rational из примера 1 лабораторной работы 4.1, используя перегрузку операторов.

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("Illegal value of the denominator")  self.\_\_numerator = a  self.\_\_denominator = 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)  sign = 1  if (self.\_\_numerator > 0 and self.\_\_denominator < 0) or \  (self.\_\_numerator < 0 and self.\_\_denominator > 0):  sign = -1  a, b = abs(self.\_\_numerator), abs(self.\_\_denominator)  c = gcd(a, b)  self.\_\_numerator = sign \* (a // c)  self.\_\_denominator = b // c  # Клонировать дробь.  def \_\_clone(self):  return Rational(self.\_\_numerator, self.\_\_denominator)  @property  def numerator(self):  return self.\_\_numerator  @numerator.setter  def numerator(self, value):  self.\_\_numerator = int(value)  self.\_\_reduce()  @property  def denominator(self):  return self.\_\_denominator  @denominator.setter  def denominator(self, value):  value = int(value)  if value == 0:  raise ValueError("Illegal value of the denominator")  self.\_\_denominator = value  self.\_\_reduce()  # Привести дробь к строке.  def \_\_str\_\_(self):  return f"{self.\_\_numerator} / {self.\_\_denominator}"  def \_\_repr\_\_(self):  return self.\_\_str\_\_()  # Привести дробь к вещественному значению.  def \_\_float\_\_(self):  return self.\_\_numerator / self.\_\_denominator  # Привести дробь к логическому значению.  def \_\_bool\_\_(self):  return self.\_\_numerator != 0  # Сложение обыкновенных дробей.  def \_\_iadd\_\_(self, rhs): # +=  if isinstance(rhs, Rational):  a = self.numerator \* rhs.denominator + \  self.denominator \* rhs.numerator  b = self.denominator \* rhs.denominator  self.\_\_numerator, self.\_\_denominator = a, b  self.\_\_reduce()  return self  else:  raise ValueError("Illegal type of the argument")    def \_\_add\_\_(self, rhs): # +  return self.\_\_clone().\_\_iadd\_\_(rhs)  # Вычитание обыкновенных дробей.  def \_\_isub\_\_(self, rhs): # -=  if isinstance(rhs, Rational):  a = self.numerator \* rhs.denominator - \  self.denominator \* rhs.numerator  b = self.denominator \* rhs.denominator  self.\_\_numerator, self.\_\_denominator = a, b  self.\_\_reduce()  return self  else:  raise ValueError("Illegal type of the argument")  def \_\_sub\_\_(self, rhs): # -  return self.\_\_clone().\_\_isub\_\_(rhs)  # Умножение обыкновенных дробей.  def \_\_imul\_\_(self, rhs): # \*=  if isinstance(rhs, Rational):  a = self.numerator \* rhs.numerator  b = self.denominator \* rhs.denominator  self.\_\_numerator, self.\_\_denominator = a, b  self.\_\_reduce()  return self  else:  raise ValueError("Illegal type of the argument")  def \_\_mul\_\_(self, rhs): # \*  return self.\_\_clone().\_\_imul\_\_(rhs)  # Деление обыкновенных дробей.  def \_\_itruediv\_\_(self, rhs): # /=  if isinstance(rhs, Rational):  a = self.numerator \* rhs.denominator  b = self.denominator \* rhs.numerator  if b == 0:  raise ValueError("Illegal value of the denominator")  self.\_\_numerator, self.\_\_denominator = a, b  self.\_\_reduce()  return self  else:  raise ValueError("Illegal type of the argument")  def \_\_truediv\_\_(self, rhs): # /  return self.\_\_clone().\_\_itruediv\_\_(rhs)  # Отношение обыкновенных дробей.  def \_\_eq\_\_(self, rhs): # ==  if isinstance(rhs, Rational):  return (self.numerator == rhs.numerator) and \  (self.denominator == rhs.denominator)  else:  return False    def \_\_ne\_\_(self, rhs): # !=  if isinstance(rhs, Rational):  return not self.\_\_eq\_\_(rhs)  else:  return False  def \_\_gt\_\_(self, rhs): # >  if isinstance(rhs, Rational):  return self.\_\_float\_\_() > rhs.\_\_float\_\_()  else:  return False  def \_\_lt\_\_(self, rhs): # <  if isinstance(rhs, Rational):  return self.\_\_float\_\_() < rhs.\_\_float\_\_()  else:  return False  def \_\_ge\_\_(self, rhs): # >=  if isinstance(rhs, Rational):  return not self.\_\_lt\_\_(rhs)  else:  return False  def \_\_le\_\_(self, rhs): # <=  if isinstance(rhs, Rational):  return not self.\_\_gt\_\_(rhs)  else:  return False  if \_\_name\_\_ == '\_\_main\_\_':  r1 = Rational(3, 4)  print(f"r1 = {r1}")  r2 = Rational(5, 6)  print(f"r2 = {r2}")  print(f"r1 + r2 = {r1 + r2}")  print(f"r1 - r2 = {r1 - r2}")  print(f"r1 \* r2 = {r1 \* r2}")  print(f"r1 / r2 = {r1 / r2}")    print(f"r1 == r2: {r1 == r2}")  print(f"r1 != r2: {r1 != r2}")  print(f"r1 > r2: {r1 > r2}")  print(f"r1 < r2: {r1 < r2}")  print(f"r1 >= r2: {r1 >= r2}")  print(f"r1 <= r2: {r1 <= r2}") |

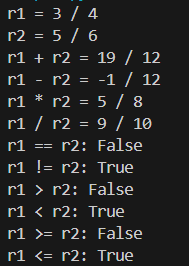


Рисунок 1 – Результат выполнения программы

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

Листинг 2 – Код к программе индивидуального задания 1

|  |
| --- |
| #!/usr/bin/env python3  # -\*- coding: utf-8 -\*-  '''  Поле first — целое положительное число, часы; поле second — целое  положительное число, минуты. Реализовать метод  minutes() — приведение времени в минуты.  Выполнил студент группы ИВТ-б-о-22-1 Иващенко О.А.  '''  class Time:  def \_\_init\_\_(self, hours=0, minutes=0):  '''  Метод инициализации значений.  '''  if not self.is\_valid(hours) or not self.is\_valid(minutes):  raise ValueError("Часы и минуты должны быть неотрицательными целыми числами")  self.hours = hours  self.minutes = minutes  @staticmethod  def is\_valid(value):  '''  Проверяет аргумент на правильность введённых данных.  '''  return isinstance(value, int) and value >= 0  def total\_minutes(self):  '''  Метод приведения времени в минуты.  '''  return self.hours \* 60 + self.minutes  def \_\_add\_\_(self, other):  '''  Перегрузка оператора сложения.  '''  if isinstance(other, Time):  total\_minutes = self.total\_minutes() + other.total\_minutes()  elif isinstance(other, int):  total\_minutes = self.total\_minutes() + other  else:  return NotImplemented  return Time(total\_minutes // 60, total\_minutes % 60)  def \_\_sub\_\_(self, other):  '''  Перегрузка оператора вычитания.  '''  if isinstance(other, Time):  total\_minutes = self.total\_minutes() - other.total\_minutes()  elif isinstance(other, int):  total\_minutes = self.total\_minutes() - other  else:  return NotImplemented  return Time(total\_minutes // 60, total\_minutes % 60)  def \_\_int\_\_(self):  '''  Приведение объекта к целому числу (общее количество минут).  '''  return self.total\_minutes()  def \_\_str\_\_(self):  '''  Строковое представление объекта.  '''  return f"{self.hours} ч. {self.minutes} мин."  def \_\_eq\_\_(self, other):  '''  Перегрузка оператора равенства.  '''  if isinstance(other, Time):  return self.total\_minutes() == other.total\_minutes()  return False  def \_\_lt\_\_(self, other):  '''  Перегрузка оператора меньше.  '''  if isinstance(other, Time):  return self.total\_minutes() < other.total\_minutes()  return NotImplemented  def \_\_le\_\_(self, other):  '''  Перегрузка оператора меньше или равно.  '''  return self == other or self < other  if \_\_name\_\_ == "\_\_main\_\_":  try:  hours = int(input("[1] Введите количество часов: "))  minutes = int(input("[1] Введите количество минут: "))  time1 = Time(hours, minutes)  hours = int(input("[2] Введите количество часов: "))  minutes = int(input("[2] Введите количество минут: "))  time2 = Time(hours, minutes)  print(f"\nВремя 1: {time1.\_\_str\_\_()}, всего минут: {int(time1)}")  print(f"Время 2: {time2.\_\_str\_\_()}, всего минут: {int(time2)}\n")  print(f"[=] Эквивалентно ли время: {time1.\_\_eq\_\_(time2)}")  print(f"[+] Сумма времён: {time1.\_\_add\_\_(time2).\_\_str\_\_()}")  if time1.total\_minutes() > time2.total\_minutes():  print(f"[-] Разность времён: {time1.\_\_sub\_\_(time2).\_\_str\_\_()}")  else:  print(f"[-] Разность времён: {time2.\_\_sub\_\_(time1).\_\_str\_\_()}")  print(f"[<] Время 1 < Время 2: {time1.\_\_lt\_\_(time2).\_\_str\_\_()}")  print(f"[<=] Время 1 <= Время 2: {time1.\_\_le\_\_(time2).\_\_str\_\_()}")  except ValueError as e:  print(f"Ошибка: {e}") |

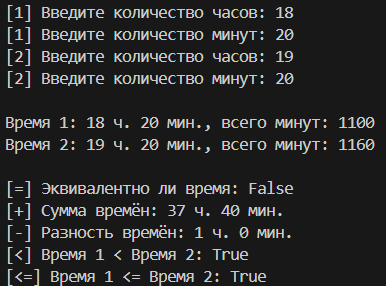


Рисунок 2 – Результат выполнения программы индивидуального задания 1

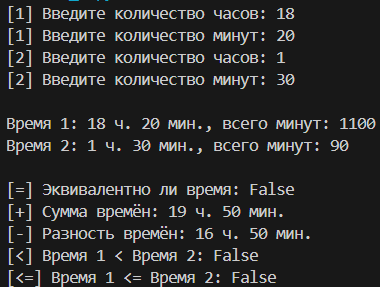


Рисунок 3 – Результат выполнения программы индивидуального задания 1

Индивидуальное 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. В тех задачах, где возможно, реализовать конструктор инициализации строкой. Карточка иностранного слова представляет собой словарейу, 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, представляющий собой карточки иностранного слова. Название словаря задаётся при создании нового словаря, но должна быть предоставлена возможность его изменения во время работы. Карточки добавляются в словарь и удаляются из него. Реализовать поиск определённого 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 исходных словари не должны изменяться. При объединении новый словарь должен содержать без повторений все слова, содержащиеся в обоих словаря-операндах. При пересечении новый словарь должен состоять из тех слов, которые имеются в обоих словарях-операндах. При вычитании новый словарь должен содержать слова первого словаря-операнда, отсутствующие во втором.

Листинг 3 – Код программы индивидуального задания 2

|  |
| --- |
| #!/usr/bin/env python3  # -\*- coding: utf-8 -\*-  """  Создать электронный словарь, который хранит карточки с иностранными  словами и их переводами, ограничивает их количество (size), поддерживает  добавление, удаление, поиск переводов, операции объединения, пересечения  и разности словарей, а также исключает дубли.  """  class WordCard:  def \_\_init\_\_(self, foreign\_word, translation):  """  Конструктор карточки иностранного слова.  """  self.foreign\_word = foreign\_word  self.translation = translation  def \_\_eq\_\_(self, other):  """  Проверяет равенство двух карточек по иностранному слову.  """  if isinstance(other, WordCard):  return self.foreign\_word == other.foreign\_word  return False  def \_\_hash\_\_(self):  """  Хеш-функция для поддержки операций с множествами.  """  return hash(self.foreign\_word)  class Dictionary:  MAX\_SIZE = 100  def \_\_init\_\_(self, name, size=MAX\_SIZE):  """  Конструктор класса Dictionary.  """  self.name = name  self.size = min(size, self.MAX\_SIZE)  self.cards = []  self.count = 0  def size(self):  """  Возвращает установленную длину словаря.  """  return self.size  def add\_card(self, foreign\_word, translation):  """  Добавляет карточку в словарь.  """  if self.count >= self.size:  raise ValueError("Словарь переполнен")  new\_card = WordCard(foreign\_word, translation)  # Проверка на наличие дублей  if new\_card in self.cards:  print(f"Слово '{foreign\_word}' уже есть в словаре")  return  self.cards.append(new\_card)  self.count += 1  def remove\_card(self, foreign\_word):  """  Удаляет карточку из словаря.  """  self.cards = [card for card in self.cards if card.foreign\_word != foreign\_word]  self.count = len(self.cards)  def find\_translation(self, foreign\_word):  """  Ищет перевод для иностранного слова.  """  for card in self.cards:  if card.foreign\_word == foreign\_word:  return card.translation  return None  def \_\_getitem\_\_(self, foreign\_word):  """  Перегрузка операции индексирования для поиска перевода.  """  return self.find\_translation(foreign\_word)  def \_\_setitem\_\_(self, foreign\_word, translation):  """  Перегрузка индексирования для добавления или обновления карточки.  """  for card in self.cards:  if card.foreign\_word == foreign\_word:  card.translation = translation  return  self.add\_card(foreign\_word, translation)  def \_\_add\_\_(self, other):  """  Операция объединения словарей.  """  if not isinstance(other, Dictionary):  return NotImplemented  new\_dict = Dictionary(f"{self.name} + {other.name}", size=self.MAX\_SIZE)  new\_dict.cards = list(set(self.cards + other.cards))  new\_dict.count = len(new\_dict.cards)  return new\_dict  def \_\_and\_\_(self, other):  """  Операция пересечения словарей.  """  if not isinstance(other, Dictionary):  return NotImplemented  new\_dict = Dictionary(f"{self.name} & {other.name}", size=self.MAX\_SIZE)  new\_dict.cards = list(set(self.cards) & set(other.cards))  new\_dict.count = len(new\_dict.cards)  return new\_dict  def \_\_sub\_\_(self, other):  """  Операция вычитания словарей.  """  if not isinstance(other, Dictionary):  return NotImplemented  new\_dict = Dictionary(f"{self.name} - {other.name}", size=self.MAX\_SIZE)  new\_dict.cards = [card for card in self.cards if card not in other.cards]  new\_dict.count = len(new\_dict.cards)  return new\_dict  def \_\_str\_\_(self):  """  Строковое представление словаря.  """  cards\_str = "\n".join(f"{card.foreign\_word}: {card.translation}" for card in self.cards)  return f"Словарь '{self.name}' ({self.count}/{self.size}):\n{cards\_str}"  # Пример использования:  if \_\_name\_\_ == "\_\_main\_\_":  dict1 = Dictionary("Еда")  dict1.add\_card("Apple", "Яблоко")  dict1.add\_card("Bread", "Хлеб")  dict1.add\_card("Milk", "Молоко")  dict2 = Dictionary("Природа")  dict2.add\_card("Apple", "Яблоко")  dict2.add\_card("Tree", "Дерево")  dict2.add\_card("River", "Река")  print(f"{dict1}\n") # Вывод словаря 1  print(f"{dict2}\n") # Вывод словаря 2  print("\nОбъединение словарей:")  print(dict1.\_\_add\_\_(dict2))  print("\nПересечение словарей:")  print(dict1.\_\_and\_\_(dict2))  print("\nРазность словарей:")  print(dict1.\_\_sub\_\_(dict2))  dict1.name = "Продовольствие" # Изменение названия словаря  print(f"\n{dict1}") |

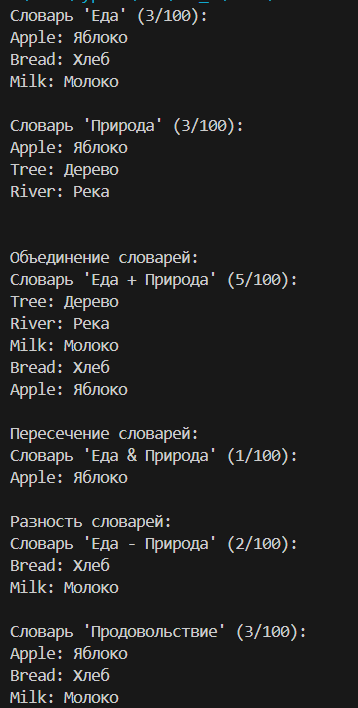


Рисунок 4 – Результат выполнения программы индивидуального задания 2

Ответы на контрольные вопросы:

1. Какие средства существуют в Python для перегрузки операций?

Для перегрузки операций в Python используются магические методы (\_\_ad\_\_, \_\_eq\_\_, \_\_len\_\_ и т.д.) для перегрузки стандартных операций.

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

Арифметические: \_\_add\_\_, \_\_sub\_\_, \_\_mul\_\_, \_\_truediv\_\_, \_\_floordiv\_\_, \_\_mod\_\_, \_\_pow\_\_.

Операции отношения: \_\_eq\_\_, \_\_ne\_\_, \_\_lt\_\_, \_\_le\_\_, \_\_gt\_\_, \_\_ge\_\_.

1. В каких случаях будут вызваны следующие методы: \_\_add\_\_ , \_\_iadd\_\_ и \_\_radd\_\_? Приведите примеры.

\_\_add\_\_ - при использовании + с объектами слева;

\_\_iadd\_\_ - при использовании +=;

\_\_radd\_\_ - при использовании + с объектами справа, если слева на определён \_\_add\_\_.

1. Для каких целей предназначен метод \_\_new\_\_ ? Чем он отличается от метода \_\_init\_\_?

\_\_new\_\_ создаёт новый объект, вызывается перед \_\_init\_\_.

\_\_init\_\_ - инициализирует объект после его создания.

1. Чем отличаются методы \_\_str\_\_ и \_\_repr\_\_?

\_\_str\_\_ - читаемое представление объекта для пользователя.

\_\_repr\_\_ - формальное представление объекта для разработчика, используется в консоли.

**Выводы**: В процессе выполнения лабораторной работы были приобретены навыки по перегрузке операторов при написании программ с помощью языка программирования Python версии 3.x.

Ссылка на репозиторий: [IUnnamedUserI/OOP\_2: Объектно-ориентированное программирование. Лабораторная работа №2](https://github.com/IUnnamedUserI/OOP_2)