**МИНИCTEPCTBO НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ**

**ФЕДЕРАЦИИ**

# Федеральное государственное автономное образовательное учреждение высшего образования

**«СЕВЕРОКАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

# Кафедра инфокоммуникаций Институт цифрового развития

**ОТЧЁТ**

# по лабораторной работе

Дисциплина: «Объектно – ориентированное программирование»

|  |
| --- |
| Выполнил: студент 3 курса |
| группы ИВТ-б-о-21-1 |
| Криворот Владимир Геннадьевич |

Ставрополь 2023

# Перегрузка операторов в языке Python

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

# Порядок выполнения работы:

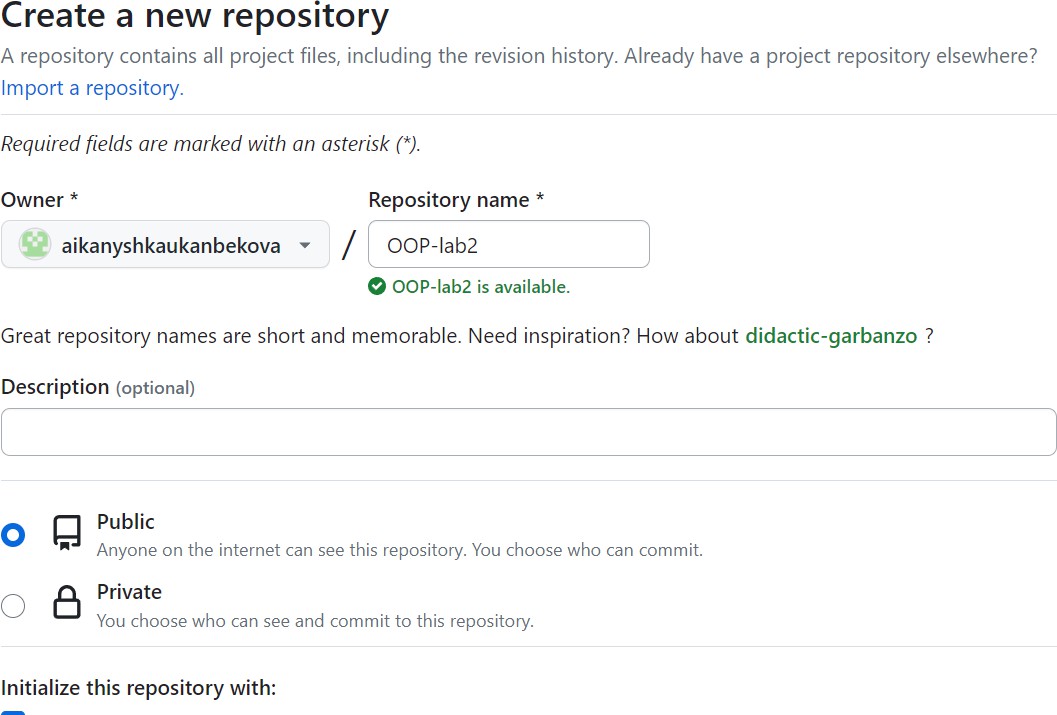
1.Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python.

Рисунок 1. Создание репозитория 2.Выполните клонирование созданного репозитория.

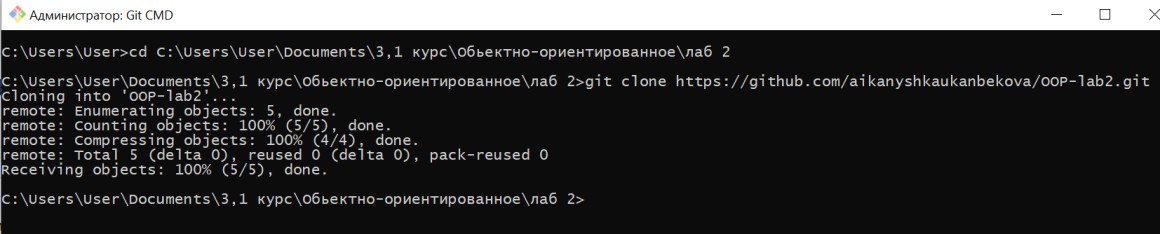


Рисунок 2. Клонирование репозитория

3.Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm.

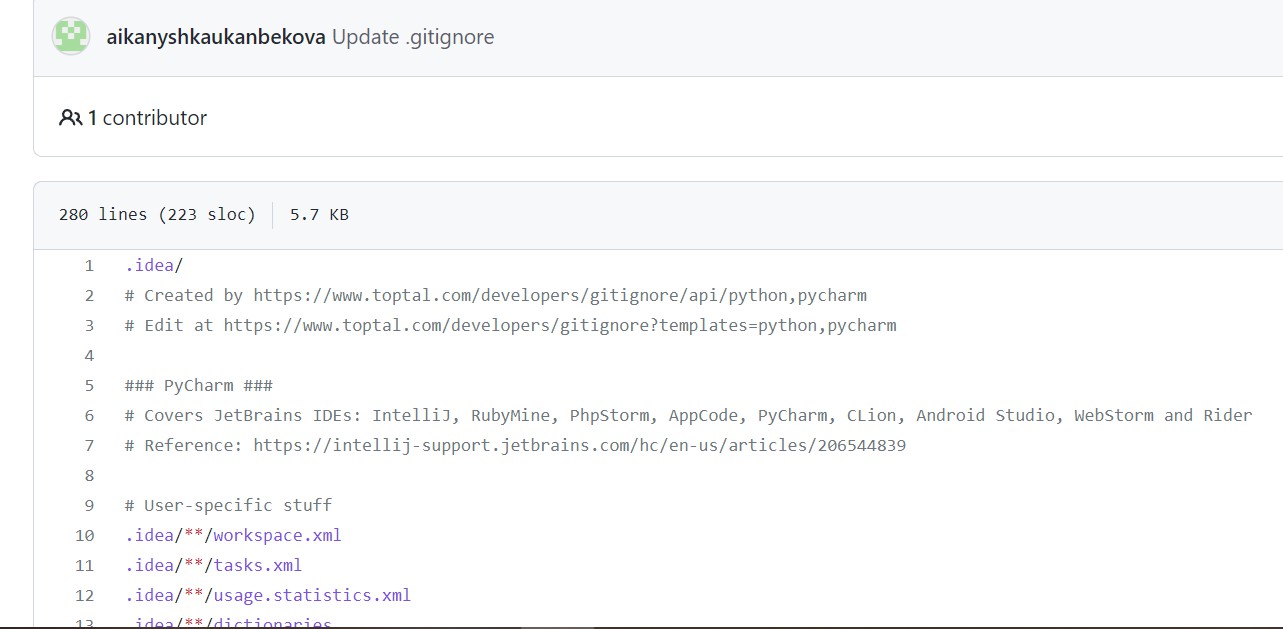


Рисунок 3. Дополнение файла .gitignore

# Практическая часть:

Вариант 19

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

Код программы:

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

import math class Pair:

*"""*

*Класс, хранящий введенные числа k и n в полях first и second*

*"""*

def init (self, first, second):

*"""*

*Конструктор класса, принимает два параметра, валидирует их и сохраняет в поля*

*"""*

# Удостоверимся, что first является целым числом

if not isinstance(first, int):

raise TypeError("Значение first должно быть целым положительным

числом")

# Удостоверимся, что second является целым числом

if not isinstance(second, int):

raise TypeError("Значение second должно быть целым положительным

числом")

# Удостоверимся, что first является положительным числом

if first < 0:

raise ValueError("Значение first должно быть положительным")

# Удостоверимся, что second является целым числом

if second < 0:

raise ValueError("Значение second должно быть положительным")

# Удостоверимся, что число first больше second

if first > second:

raise ValueError("Значение first должно быть меньше либо равно

чем second")

# Записываем значения в поля self.first = first self.second = second

def combination(self):

*"""*

*Метод высчитывает значения по формуле, с использованием переданных в конструкторе значений*

*"""*

return math.factorial(self.second) / (math.factorial(self.second - self.first) \* math.factorial(self.first))

def display(self):

*"""*

*Метод выводит экземпляра класса в консоль с указанием переданных в конструктор значений полей first и second*

*"""*

print(f"({self.first}, {self.second})")

def str (self):

*"""*

*Перегрузка оператора приведения к строке*

*"""*

return f"({self.first}, {self.second})"

def eq (self, other):

*"""*

*Перегрузка оператора сравнения '=='*

*"""*

return self.first == other.first and self.second == other.second

def ne (self, other):

*"""*

*Перегрузка оператора неравенства '!='*

*"""*

return self.first != other.first or self.second != other.second

def add (self, other):

*"""*

*Перегрузка оператора сложения '+'*

*"""*

self.first += other.first self.second += other.second return self

def sub (self, other):

*"""*

*Перегрузка оператора вычитания '-' """*

self.first -= other.first self.second -= other.second return self

@classmethod def read(cls):

*"""*

*Статичный метод для создания экземпляра класса с запрашиваеним значений в консоли*

*"""*

k = int(input("Введите число k: ")) n = int(input("Введите число n: "))

return cls(k, n)

def make\_pair(first, second):

*"""*

*Функция создания экземпляра класса Pair, принимая значения полей как аргументы*

*"""*

return Pair(first, second)

if name == ' main ':

# Создаем 2 экземпляра класса Pair

pair1 = Pair(5, 10) pair2 = Pair(6, 15)

# Операции сравнения print(pair1 == pair2) print(pair1 != pair2)

# Складываем первый со вторый

print(pair1 + pair2)

# Вычитаем второй из первого

print(pair1 - pair2)

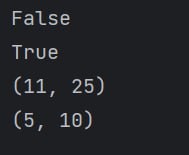


Рисунок 1. Результат задания 1

Задание 2. Создать класс String для работы со строками, аналогичными строкам Turbo Pascal (строка представляется как список 255 байт, длина — в первом байте). Максимальный размер строки должен задаваться. Обязательно должны быть реализованы: определение длины строки, поиск подстроки в строке, удаление подстроки из строки, вставка подстроки в строку, сцепление двух строк.

Код программы:

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

class String:

*"""*

*Класс String по примеру реализации класса из TurboPascal*

*"""*

MAX\_SIZE = 255 # Максимальное количество символов в строке

def init (self, string):

*"""*

*Конструктор класса, обязательный параметр - исходная строка*

*"""*

if len(string) > self.MAX\_SIZE:

raise ValueError("Строка слишком длинная")

# Текущее количество символов в строке

self.count = len(string)

# Массив символов, первым элементом является длина

self.data = [self.count] \* (self.MAX\_SIZE + 1)

# Записываем все символы из строки в массив символов data

for i, char in enumerate(string): self.data[i + 1] = char

def size(self):

*"""*

*Длина строки*

*"""*

return self.count

def len (self):

*"""*

*Перегруженный оператор для функции len, возвращает длину строки*

*"""*

return self.count

def str (self):

*"""*

*Преобразование экземпляра класса в строку*

*"""*

string = ""

for i in range(1, self.count + 1): string += self.data[i]

return string

def find(self, substring):

*"""*

*Поиск подстроки в строке, возвращает индекс, если подстрока не входит в строку - возвращает -1*

*"""*

sub\_length = len(substring)

for i in range(1, self.count - sub\_length + 2): found = True

for j in range(sub\_length):

if self.data[i + j] != substring[j]: found = False

break

if found:

return i

return -1

def remove(self, substring):

*"""*

*Удаление подстроки из строки*

*"""*

sub\_length = len(substring) index = self.find(substring)

if index != -1:

for i in range(index + sub\_length, self.count + 1): self.data[i - sub\_length] = self.data[i]

self.count -= sub\_length self.data[0] = self.count

def sub (self, other):

*"""*

*Перегруженный оператор вычитания '-', удаляет подстроку из строки*

*"""*

self.remove(other) return self

def insert(self, substring, index):

*"""*

*Вставка подстроки в строку в определенный индекс*

*"""*

sub\_length = len(substring)

if self.count + sub\_length > self.MAX\_SIZE:

raise ValueError(f"Строка не может быть больше чем

{self.MAX\_SIZE}")

for i in range(self.count, index - 1, -1): self.data[i + sub\_length] = self.data[i]

for i, char in enumerate(substring): self.data[index + i] = char

self.count += sub\_length self.data[0] = self.count

def concatenate(self, other):

*"""*

*Конкатенация переданной строки в конец текущей строки*

*"""*

other\_len = len(other)

if self.count + other\_len > self.MAX\_SIZE:

raise ValueError(f"Строка не может быть больше чем

{self.MAX\_SIZE}")

for i in range(0, other\_len): self.data[self.count + i + 1] = other[i]

self.count += other\_len self.data[0] = self.count

def add (self, other):

*"""*

*Перегрузка оператора сложения, конкатенируют подстроку в конец текущей строки*

*"""*

self.concatenate(other) return self

def getitem (self, item):

*"""*

*Перегрузка оператора [] - получение символа из строки по индексу*

*"""*

return self.data[item]

if name == ' main ':

s = String("Уланбекова") print(s)

print(s.size())

s += " Айканыш" print(s) print(s.size())

print(s[12])

s.remove("Уланбекова ") print(s) print(s.size())

print(s.find("ка"))

s.insert("555", 3) print(s)

s.remove("555") print(s)

s.concatenate(" Уланбекова") print(s)

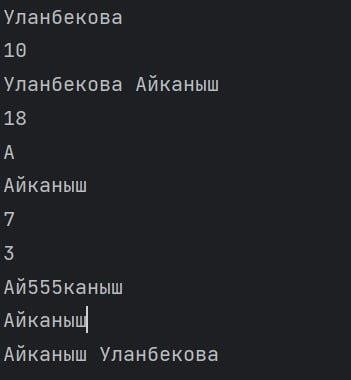


Рисунок 2. Результат задания 2

# Контрольные вопросы:

1. Какие средства существуют в Python для перегрузки операций?

Для перегрузки операций в Python используется механизм магических методов (или методов-операторов), которые начинаются и заканчиваются двумя символами подчеркивания. Например, для перегрузки операции сложения (+) используется метод add (), для перегрузки операции индексации ([]) используется метод getitem (), и т.д.

Некоторые из магических методов, которые можно перегрузить в Python:

* + add () – операция сложения
  + sub () – операция вычитания
  + mul () – операция умножения
  + truediv () – операция деления
  + mod () – операция остатка от деления
  + lt () – операция "меньше"
  + gt () – операция "больше"
  + eq () – операция "равно"
  + ne () – операция "не равно"
  + str () – преобразование объекта в строку

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

Для перегрузки арифметических операций в Python используются следующие методы:

* + add () – операция сложения (+)
  + sub () – операция вычитания (-)
  + mul () – операция умножения (\*)
  + truediv () – операция деления (/)
  + floordiv () – операция целочисленного деления (//)
  + mod () – операция остатка от деления (%)
  + pow () – операция возведения в степень (\*\*)

Для перегрузки операций отношения в Python используются следующие методы:

* + lt () – операция "меньше" (<)
  + le () – операция "меньше или равно" (<=)
  + eq () – операция "равно" (==)
  + ne () – операция "не равно" (!=)
  + gt () – операция "больше" (>)
  + ge () – операция "больше или равно" (>=)

1. В каких случаях будут вызваны следующие методы: add ,

iadd и radd ? Приведите примеры.

Метод add () будет вызван при использовании оператора сложения (+) между двумя объектами, например:

a = 5

b = 10

c = a + b # вызовется метод add () у объекта a

Метод iadd () будет вызван при использовании сокращенной операции сложения (+=) между двумя объектами, например:

a = 5

b = 10

a += b # вызовется метод iadd () у объекта a

Метод radd () будет вызван, если первый операнд не поддерживает операцию сложения (+), а второй поддерживает. Например:

a = "Hello"

b = " world"

c = b + a # вызовется метод radd () у объекта a, так как строка не поддерживает сложение с типом str

1. Для каких целей предназначен метод new ? Чем он отличается от метода init ?

Метод new() в Python используется для создания нового экземпляра класса. Этот метод вызывается перед методом init () и возвращает новый объект класса.

Метод init () же используется для инициализации созданного объекта класса. Он вызывается после метода new() и позволяет задать начальные значения атрибутов объекта.

Отличие между методами new() и init () заключается в том, что new() создает новый объект, а init () инициализирует его. Также, в отличие от

init (), метод new() не обязательно должен возвращать экземпляр класса – он может вернуть любой другой объект.

1. Чем отличаются методы str и repr ?

Метод str() используется для получения строкового представления объекта, которое будет понятно человеку. Он вызывается функцией str() или при использовании объекта в контексте, где ожидается строковое значение, например, при выводе на экран.

Метод repr() используется для получения строкового представления объекта, которое будет понятно интерпретатору Python. Он вызывается функцией repr() или при использовании объекта в интерактивной оболочке Python.

Таким образом, метод str() предназначен для отображения объекта пользователю, а метод repr() – для отображения объекта в коде Python. Обычно метод repr() возвращает строку, которую можно использовать для создания точной копии объекта с помощью функции eval().