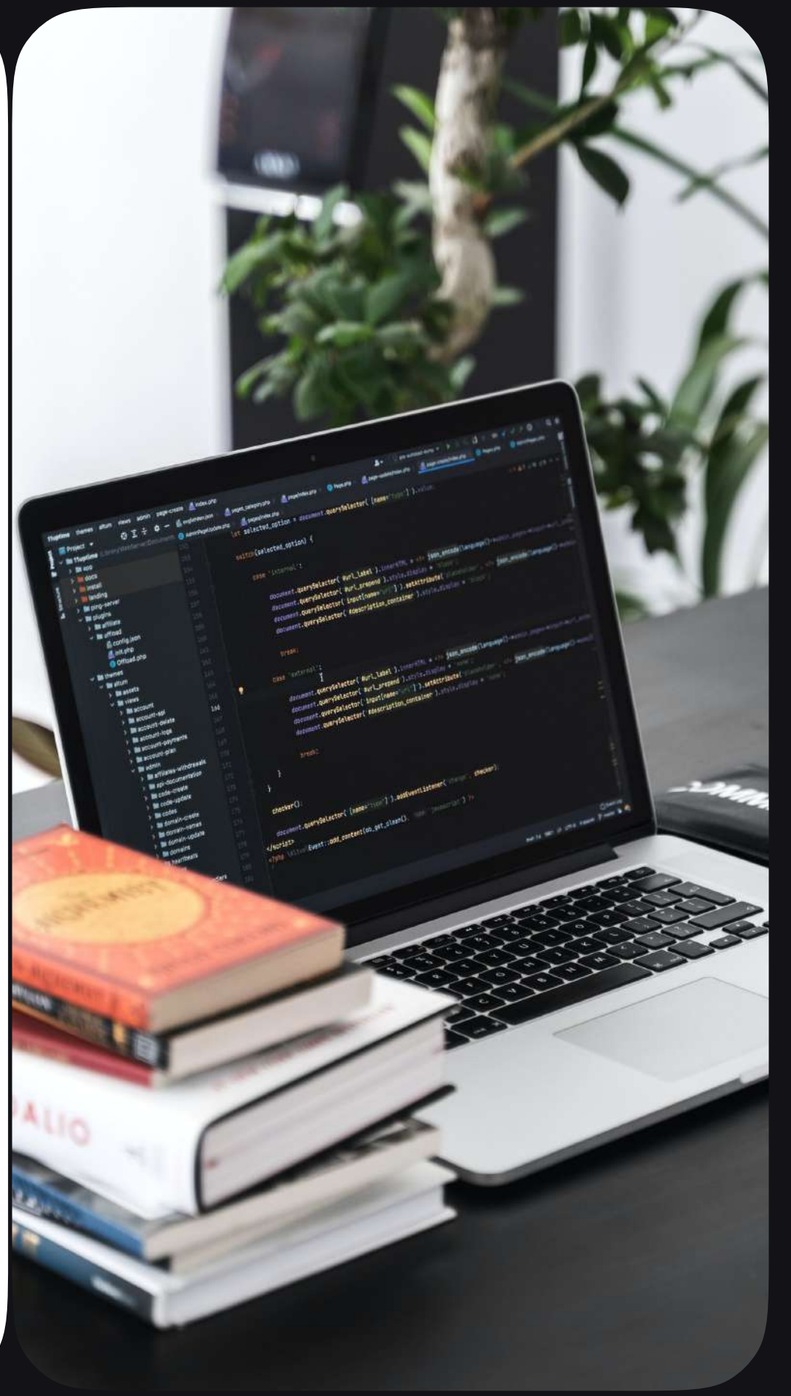
Модуль 2 Занятие 5

Переопределение методов



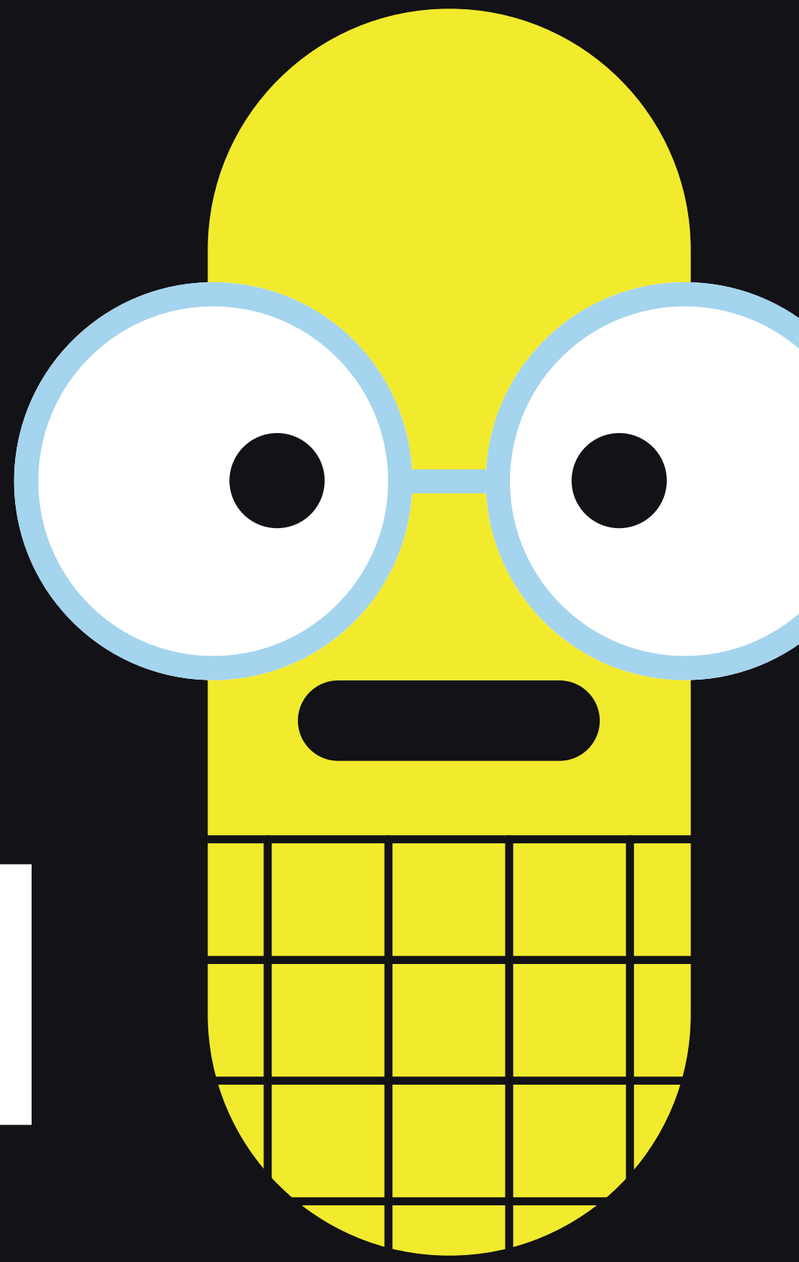
Переопределение методов

Магические методы арифметических операций

Магические методы сравнения

Метод __call__

Теория



На чем мы остановились

```
class Point:
    def __init__(self, x):
        self.x = x

    def __str__(self):
        return f'Точка с координатой {self.x}'

my_point1 = Point(2)
my_point2 = Point(3)
my_point1 + my_point2
```

Можно ли выполнить арифметическую операцию с нашими объектами?

Ошибка:

```
TypeError: unsupported operand type(s) for +: 'Point' and 'Point'
```

Перегрузка операторов

Один и тот же оператор в Python, например +, ведет себя с разными типами по-разному:

```
a = 2 + 3
b = '2' + '3'
print(a)
print(b)
```

Вывод:

```
5
23
```

Возможность переопределять различные операторы в классах называют **перегрузкой операторов**

Класс Fraction

```python
import math

class Fraction:
    def __init__(self, num, den):
        self.num, self.den = self.get_reduced_fraction(num, den)

    # Статические методы get_reduced_fraction # и get_common_denominator

    def __str__(self):
        return f'Дробь {self.num}/{self.den}'

fraction1 = Fraction(1, 2)
fraction2 = Fraction(3, 9)
print(fraction1)
print(fraction2)
```

Создадим новый класс Fraction — обыкновенная дробь

- numerator — числитель
- denominator — знаменатель

Вывод:

Дробь 1/2

Дробь 1/3

Класс Fraction

Методы get_reduced_fraction и get_common_denominator:

```
@staticmethod
def get_reduced_fraction(num, den):
    """Принимает числитель и знаменатель дроби
    и возвращает кортеж: числитель и
    знаменатель сокращенной дроби."""
    gcd = math.gcd(num, den)
    return num // gcd, den // gcd

@staticmethod
def get_common_denominator(den1, den2):
    """Принимает знаменатель первой и
    знаменатель второй дроби и
    возвращает общий знаменатель."""
    common_den = den1 * den2 // math.gcd(den1, den2)
    return common_den
```

Сложение дробей

```
class Fraction:
    ...

fraction1 = Fraction(1, 2)
fraction2 = Fraction(3, 9)
print(fraction1 + fraction2)
```

Чтобы выполнять операцию сложения, необходимо переопределить поведение оператора + в классе Fraction. Выполним перегрузку оператора +.

Ошибка:

```
TypeError: unsupported operand type(s) for +: 'Fraction' and 'Fraction'
```

Арифметические операции

- def __add__(self, other): … — сложение self + other
- def __sub__(self, other): … — вычитание self – other
- def __mul__(self, other): … — умножение self * other
- def __truediv__(self, other): … — деление self / other
- def __floordiv__(self, other): … - деление нацело self // other
- и другие…

Полный список методов
можно найти на странице
официальной документации

Сложение дробей

```
class Fraction:
    ...

    def __add__(self, other):
        common_den = self.get_common_denominator(self.den, other.den)
        num = common_den // self.den * self.num + common_den // other.den * other.num
        num, den = self.get_reduced_fraction(num, common_den)
        return num, den

fraction1 = Fraction(1, 2)
fraction2 = Fraction(3, 9)
print(fraction1 + fraction2)
```

Результатом сложения двух дробей будет кортеж: числитель и знаменатель дроби, полученной в результате арифметической операции

Вывод:

```
(5, 6)
```

Сложение дробей

```
class Fraction:
    ...

    def __add__(self, other):
        common_den = self.get_common_denominator(self.den, other.den)
        num = common_den // self.den * self.num + common_den // other.den * other.num
        num, den = self.get_reduced_fraction(num, common_den)
        return Fraction(num, den)

fraction1 = Fraction(1, 2)
fraction2 = Fraction(3, 9)
fraction3 = fraction1 + fraction2
print(fraction1)
```

Но в некоторых случаях было бы полезно получить в результате новый объект класса Fraction

Вывод:

```
Дробь 5/6
```

Расширенные присваивания

- def __iadd__(self, other): … — self += other
- def __isub__(self, other): … — self -= other
- def __imul__(self, other): … — self *= other
- def __itruediv__(self, other): … — self /= other
- def __ifloordiv__(self, other): … — self //= other
- и другие…

Эти методы должны выполнять операцию на месте, то есть изменять объект self, и обязательно возвращать результат

Сложение дробей

```
class Fraction:
    ...

    def __add__(self, other):
        common_den = self.get_common_denominator(self.den, other.den)
        num = common_den // self.den * self.num + common_den // other.den * other.num
        num, den = self.get_reduced_fraction(num, common_den)
        return Fraction(num, den)

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

fraction1 = Fraction(1, 2)
fraction2 = Fraction(3, 9)
fraction1 += fraction2
print(fraction1)
```

Метод __iadd__ уже
не создает новый объект,
а меняет существующий.
В результате возвращает
этот же самый объект

Вывод:

Дробь 5/6

Арифметические операции (отраженные)

- def __radd__(self, other): … — сложение other + self
- def __rsub__(self, other): … — вычитание other – self
- def __rmul__(self, other): … умножение other * self
- def __rtruediv__(self, other): … деление other / self
- def __rfloordiv__(self, other): … деление нацело other // self
- и другие…

Эти методы также вызываются для арифметических операций, но с переставленными операндами, например, если надо получить разное поведение операций **self + other** и **other + self**

Пример

```
class Test:
    def __init__(self, x):
        self.x = x

    def __add__(self, other):
        return 'Вызов метода __add__'

    def __radd__(self, other):
        return 'Вызов метода __radd__'

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

    def __str__(self):
        return f'x = {self.x}'

a = Test(1)
print(a + 2)
print(2 + a)
a += 2
print(a)
```

Если не определены методы расширенных или отраженных арифметических операций, то в этом случае будут вызываться обычные методы арифметических операций

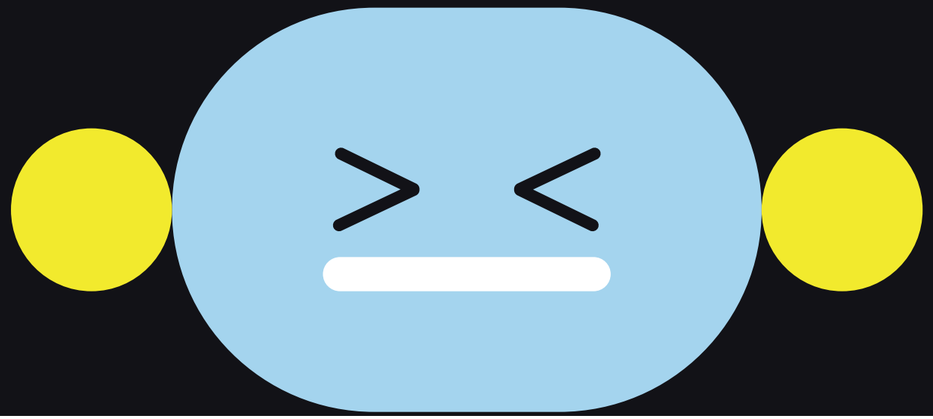
Вывод:

```
Вызов метода __add__
Вызов метода __radd__
x = 3
```

перерыв физкультминутка

Смотрим вверх–вниз, вправо–влево

Вращаем по кругу туда–обратно



Крепко зажмуриваемся

Быстро моргаем

Сравнение дробей

```
class Fraction:
...

fraction1 = Fraction(1, 2)
fraction2 = Fraction(2, 3)
print(fraction1 < fraction2)
```

Чтобы выполнять операцию сравнения необходимо переопределить поведение оператора сравнения < в классе Fraction. Выполним перегрузку оператора <.

Ошибка:

```
TypeError: '<' not supported between instances of 'Fraction' and 'Fraction'
```

Методы сравнения

- def __eq__(self, other): … — сравнение на равенство self == other
- def __ne__(self, other): … — не равно self != other
- def __lt__(self, other): … — меньше self < other
- def __gt__(self, other): … — больше self > other
- def __le__(self, other): … — меньше или равно self <= other
- def __ge__(self, other): … — больше или равно self >= other

Представленные методы определяют поведение соответствующих им операторов сравнения

Практика

Создай класс Fraction (обыкновенная дробь). Добавь инициализатор, который будет принимать два целых числа и добавлять объекту атрибуты — числитель и знаменатель. Добавь метод __str__, который будет возвращать текстовое представление объекта в формате: «Дробь {числитель}/{знаменатель}», например, «Дробь 1/2». Переопредели методы сравнения ==, !=, >, >=, <, <=. Методы должны возвращать True или False.

Считай с клавиатуры две строки, содержащие два целых числа через пробел — числитель и знаменатель первой дроби и числитель и знаменатель второй дроби. Создай экземпляры класса Fraction с параметрами, считанными с клавиатуры. Выведи результаты сравнения ==, !=, >, >=, <, <= этих дробей.

Входные данные:

Вводится две строки, каждая строка содержит два целых числа через пробел — числитель и знаменатель первой дроби и числитель и знаменатель второй дроби

Выходные данные:

Выводится 6 строк True или False

Вызов объекта

```
class Fraction:
...

fraction = Fraction(1, 2)
fraction()
```

Чтобы вызвать объект как функцию, необходимо определить магический метод __call__.

Ошибка:

```
TypeError: 'Fraction' object is not callable
```

Пример

```
class Fraction:
    ...

    def __call__(self):
        num = int(input('Введите новый числитель дроби: '))
        den = int(input('Введите новый знаменатель дроби: '))
        self.num, self.den = num, den

fraction = Fraction(1, 2)
fraction()
print(fraction)
```

Если экземпляр будет вызван как функция, то можно будет ввести новый числитель и новый знаменатель дроби и изменить существующую дробь

Вывод:

```
Введите новый числитель дроби: 2
Введите новый знаменатель дроби: 3
Дробь 2/3
```