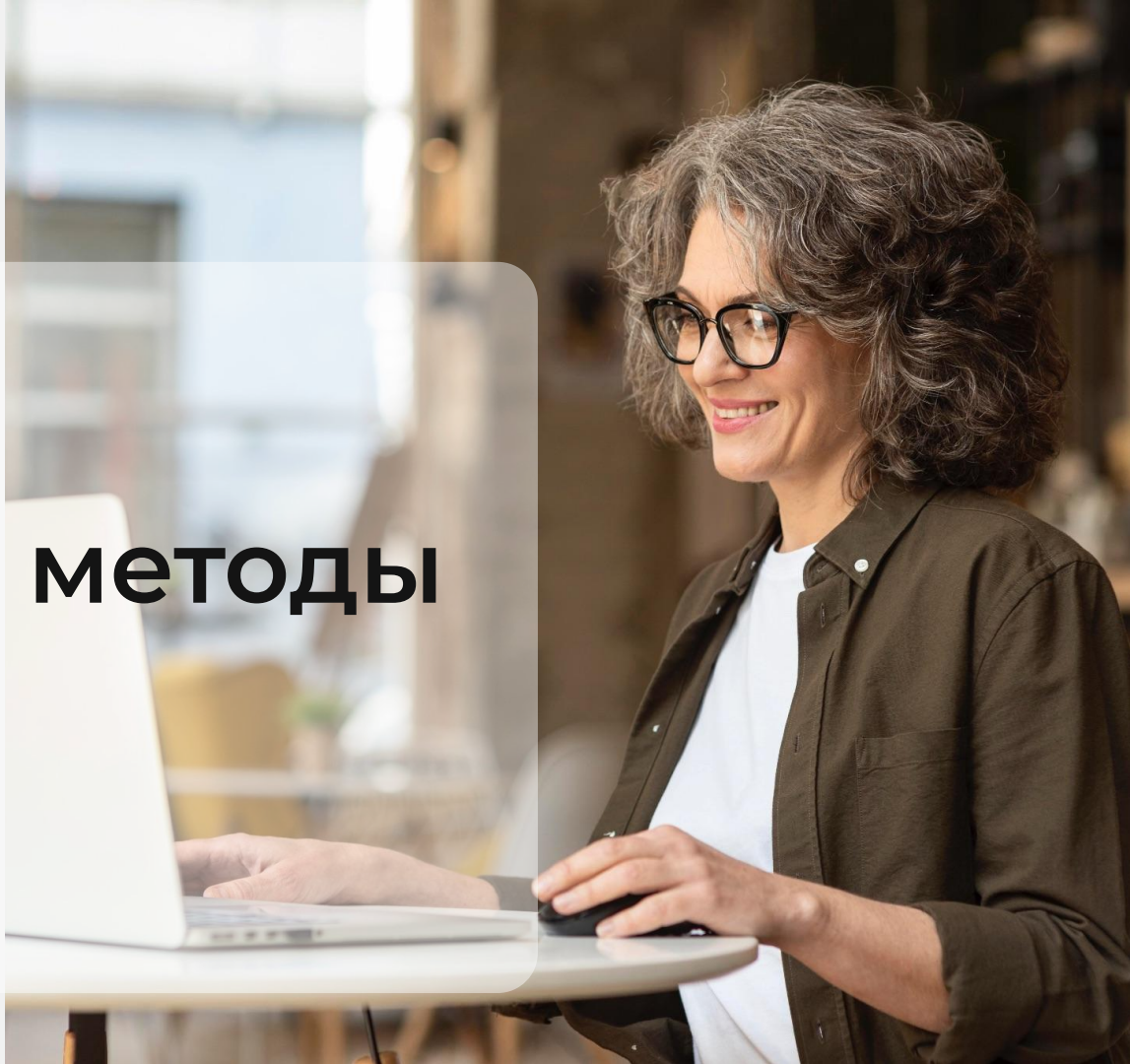
IT CAREER HUB

Python

Магические методы



Преподаватель

Портрет

Имя Фамилия

Текущая должность

Количество лет опыта

Какой у Вас опыт - ключевые кейсы

Самые яркие проекты

Дополнительная информация по вашему усмотрению

Корпоративный e-mail

Социальные сети (по желанию)

Важно

- Камера должна быть включена на протяжении всего занятия
- В течение занятия вопросы задавать в чате или когда преподаватель спрашивает, есть ли у Вас вопросы
- Вести себя уважительно и этично по отношению к остальным участникам занятия
- Организационные вопросы по обучению решаются с кураторами, а не на тематических занятиях
- Во время занятия будут интерактивные задания, будьте готовы включить камеру или демонстрацию экрана по просьбе преподавателя

Повторение

- [x] Абстрактные классы
- [x] Документация классов
- [x] Пользовательские исключения
- [x] Магические методы
- [x] Магические методы итерации

План занятия

- Методы сравнения
- Декоратор functools.total_ordering
- Методы индексации и доступа к элементам
- Арифметические методы
- Магический метод __bool__
- Магический метод __call__

ICH IT CAREER HUB

ОСНОВНОЙ БЛОК

IT CAREER HUB

Методы сравнения

Важно

Магические методы сравнения позволяют определить, как объекты **сравниваются между собой** при использовании операторов `==`, `!=`, `<`, `<=`, `>`, `>=`.

Применение магических методов сравнения

- Сравнение объектов по содержимому, а не по адресу в памяти
- Сортировка и поиск в списках
- Работа с множествами и словарями

Методы и операторы

Метод	Оператор	Назначение
`__eq__`	`==`	Равенство
`__ne__`	`!=`	Неравенство
`__lt__`	`<`	Меньше
`__le__`	`<=`	Меньше или равно
`__gt__`	`>`	Больше
`__ge__`	`>=`	Больше или равно

Особенности

- Если не реализован `__ne__`, Python использует `not __eq__()`
- Если реализован только `__lt__`, то `sort()` всё равно будет работать

Пример: сравнение книг по названию

1

```
class Book:
    def __init__(self, title):
        self.title = title

    def __eq__(self, other):
        if not isinstance(other, Book):
            return NotImplemented
        return self.title == other.title

    def __lt__(self, other):
        return self.title < other.title

    def __repr__(self):
        return f"Book(title={self.title})"
```

2

```
b1 = Book("1984")
b2 = Book("1984")
b3 = Book("Brave New World")

print(b1 == b2)
print(b1 < b3)
# сортировка по названию
print(sorted([b3, b1, b2]))
```

Особенности

- Всегда проверяйте тип `other` внутри метода, чтобы избежать ошибок (`isinstance`)
- Если метод возвращает `NotImplemented`, Python попробует сравнить наоборот или выбросит исключение

ICH IT CAREER HUB

Декоратор functools.total_ordering

Важно

Для **сравнения объектов** вместо написания **всех шести методов сравнения**, можно воспользоваться **декоратором** `@total_ordering` из модуля `functools`.

Что делает @total_ordering

- Позволяет **сократить количество кода**
- Вы реализуете только `__eq__` **и один из методов:** `__lt__, __le__, __gt__, __ge__`
- Остальные сравнения Python сгенерирует **автоматически**

Пример

1

```
from functools import total_ordering

@total_ordering
class Book:
    def __init__(self, title):
        self.title = title

    def __eq__(self, other):
      if not isinstance(other, Book):
          return NotImplemented
      return self.title == other.title

    def __lt__(self, other):
        return self.title < other.title
```

2

```
b1 = Book("1984")
b2 = Book("Brave New World")

print(b1 < b2)    # < реализовано вручную
print(b1 >= b2)   # >= создано
автоматически
```

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

Методы индексации и доступа к элементам

Методы индексации и доступа к элементам

Метод	Цель	Возвращает	Использование
`__getitem__`	Получать элемент по ключу или индексу	Значение	`value = obj[key]`
`__setitem__`	Устанавливать значение по ключу	`None`	`obj[key] = value`
`__contains__`	Проверять наличие элемента	`True` или `False`	`key in obj`
`__len__`	Возвращать количество элементов	Целое число	`len(obj)`

Пример: коллекция заметок

```python
class Notes:
    def __init__(self):
        self._data = {}

    def __getitem__(self, key):
        # доступ по ключу (в нижнем регистре)
        return self._data[key.lower()]

    def __setitem__(self, key, value):
        if value.strip():  # сохраняем только непустые строки
            self._data[key.lower()] = value
        else:
            raise ValueError("Empty notes are not allowed")

    def __contains__(self, key):
        return key.lower() in self._data  # проверка in

    def __len__(self):
        return len(self._data)  # длина коллекции
```

1

создать класс `Notes`, в котором:

- автоматически фильтруются пустые строки,
- ключи автоматически приводятся к нижнему регистру,
- можно использовать `len()`, `in`, индексацию и присваивание.

Пример: коллекция заметок

2

```
    def __str__(self):
        result = "Notes:\n"
        for key, value in self._data.items():
            result += f"- {key}: {value}\n"
        return result.strip()

notes = Notes()
notes["Idea"] = "Build a game"         # присваивание по ключу
notes["TODO"] = "Finish the project"   # присваивание по ключу

print(notes["idea"])       # доступ по ключу
print("todo" in notes)
# вхождение элемента игнорируя регистр
print(len(notes))          # количество элементов
print(notes)               # строковое представление
```

создать класс `Notes`, в котором:

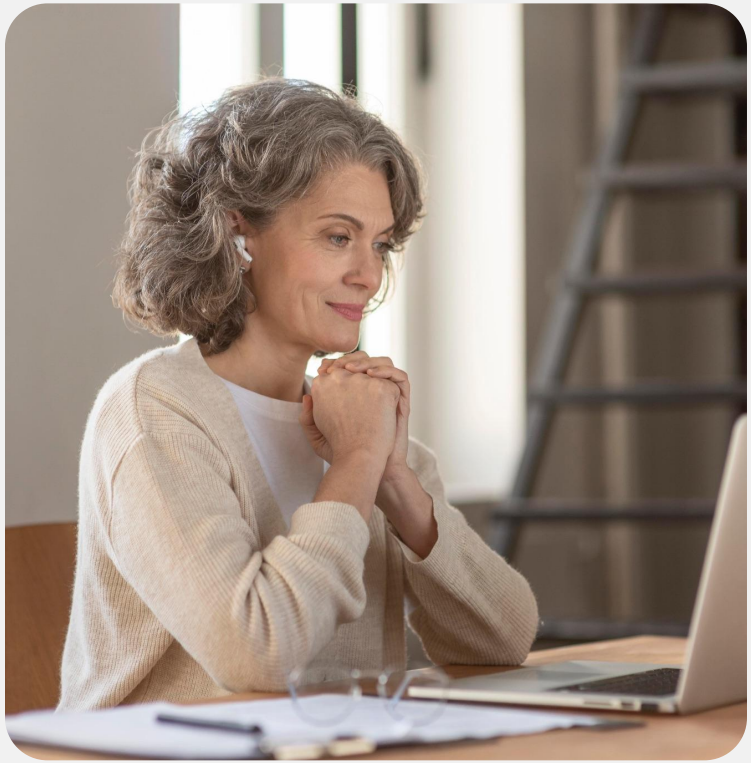
- автоматически фильтруются пустые строки,
- ключи автоматически приводятся к нижнему регистру,
- можно использовать `len()`, `in`, индексацию и присваивание.

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

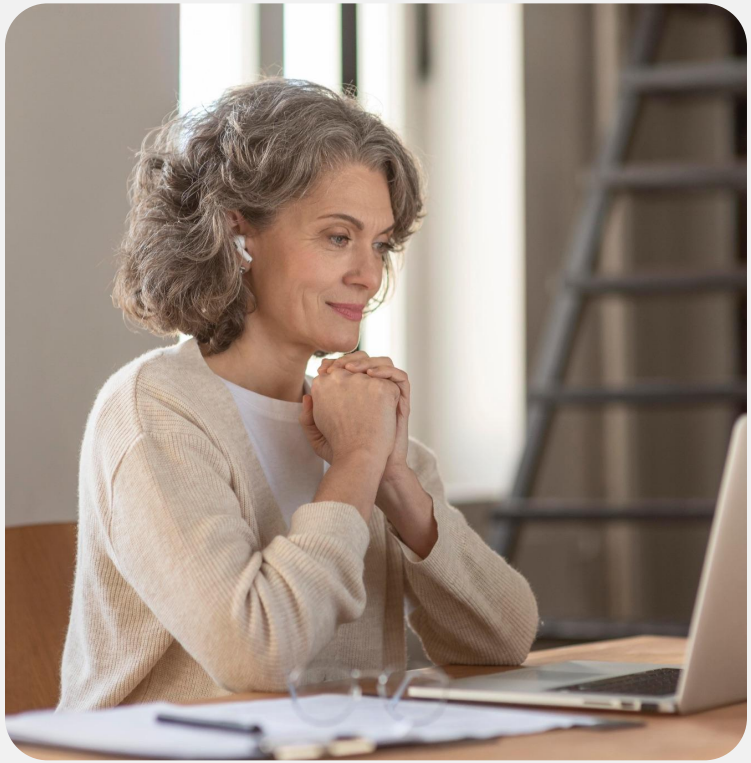
ЗАДАНИЯ



Выберите верный вариант ответа

Укажите верные утверждения о @total_ordering:

a. Нужно реализовать все шесть методов сравнения вручную
b. Достаточно реализовать `__eq__` и один из `__lt__`, `__le__`, `__gt__`, `__ge__`
c. `@total_ordering` автоматически создаёт недостающие методы сравнения
d. Работает только с числами



Выберите верный вариант ответа

Укажите верные утверждения о @total_ordering:

a. Нужно реализовать все шесть методов сравнения вручную
b. Достаточно реализовать `__eq__` и один из `__lt__`, `__le__`, `__gt__`, `__ge__`
c. `@total_ordering` автоматически создаёт недостающие методы сравнения
d. Работает только с числами

IT CAREER HUB

ВОПРОСЫ

ICH IT CAREER HUB

Арифметические методы

Важно

Магические методы позволяют переопределить поведение **арифметических операторов** (`+`, `-`, `*`, `/` и т.д.) для объектов пользовательских классов. Это удобно, когда объект представляет **число, вектор, матрицу** или любую другую структуру, с которой логично выполнять арифметические действия.

Магические методы для арифметических операторов

Метод	Оператор	Назначение
`__add__`	`+`	Сложение
`__sub__`	`-`	Вычитание
`__mul__`	`*`	Умножение
`__truediv__`	`/`	Деление
`__floordiv__`	`//`	Целочисленное деление
`__mod__`	`%`	Остаток от деления
`__pow__`	`**`	Возведение в степень

Пример: класс для 2D-вектора

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

    def __add__(self, other):
        if not isinstance(other, Vector):
            return NotImplemented
        return Vector(self.x + other.x, self.y + other.y)

    def __str__(self):
        return f"({self.x}, {self.y})"


a = Vector(2, 3)  # координаты смещения
b = Vector(1, 4)  # координаты смещения
c = a + b         # объединение координат смещения
print(c)
```

Особенности

- Метод получает **два аргумента**: `self` (до оператора) и `other` (после оператора)
- Можно проверять тип `other`, чтобы обрабатывать только нужные случаи

Инкрементные арифметические методы

Если объект поддерживает **изменение на месте**, можно реализовать специальные методы для работы с операторами `+=, -=, *=, /=` и др.

Они называются **инкрементными** и начинаются с `__i...__`. Метод должен **изменять объект и возвращать его**

Инкрементные арифметические методы

Метод	Оператор	Назначение
`__iadd__`	`+=`	Сложение с присваиванием
`__isub__`	`-=`	Вычитание с присваиванием
`__imul__`	`*=`	Умножение с присваиванием
`__itruediv__`	`/=`	Деление с присваиванием
`__ifloordiv__`	`//=`	Целочисленное деление с присваиванием
`__imod__`	`%=`	Остаток от деления с присваиванием
`__ipow__`	`**=`	Возведение в степень с присваиванием

Пример

```
class Counter:
    def __init__(self, value=0):
        self.value = value

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

    def __str__(self):
        return f"Counter({self.value})"

c = Counter(5)
print(id(c))
c += 3
print(id(c))  # id объекта не изменился
print(c)
```

IT CAREER HUB

Магический метод __bool__

Полезно знать

ICH
IT CAREER HUB

Метод __bool__

Этот метод определяет, как объект ведёт себя в логических выражениях — например, при проверке в `if, while`

Пример: считается «пустым» при нуле и меньше

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

    def __bool__(self):
        return self.value > 0


c1 = Counter(-5)
c2 = Counter(3)

print(bool(c1))
print(bool(c2))

if c1:
    print("Есть элементы")
else:
    print("Пусто")
```

IT CAREER HUB

Магический метод __call__

Полезно знать

ICH
IT CAREER HUB

Метод __call__

Позволяет делать объект вызываемым, то есть обращаться к нему как к функции — через круглые скобки: `obj()`.
Он позволяет использовать объект как функцию с внутренним состоянием

Пример: объект-конвертер

```
class CurrencyConverter:
    def __init__(self, rate):
        self.rate = rate  # сколько единиц нужно за 1 доллар

    def __call__(self, dollars):
        return dollars * self.rate

# Курс: 1 доллар = 0.88 евро
euro_converter = CurrencyConverter(0.88)

print(euro_converter(10))
print(euro_converter(5.5))
```

Особенности

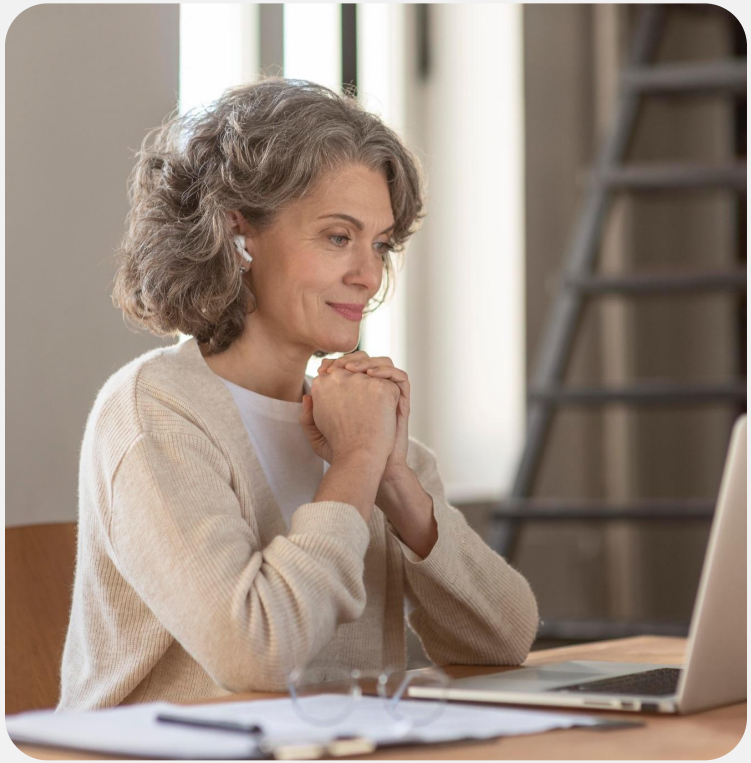
- Метод вызывается при `obj(...)`
- Может принимать аргументы, как обычная функция

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

ЗАДАНИЯ

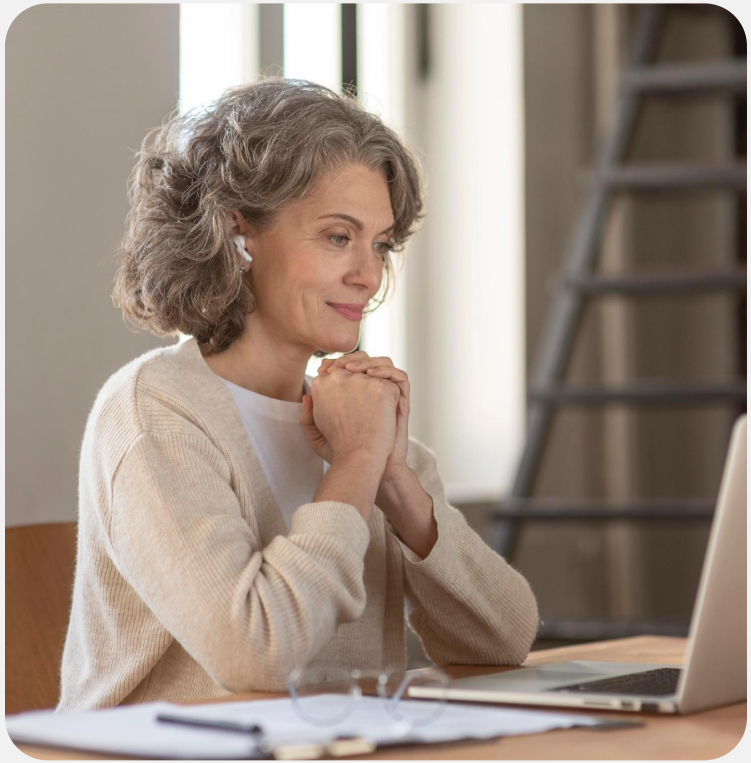


Выполните задание

Найдите ошибку в коде

```
class Box:
    def __init__(self, weight):
        self.weight = weight

    def __add__(self):
        return Box(self.weight + 1)
```



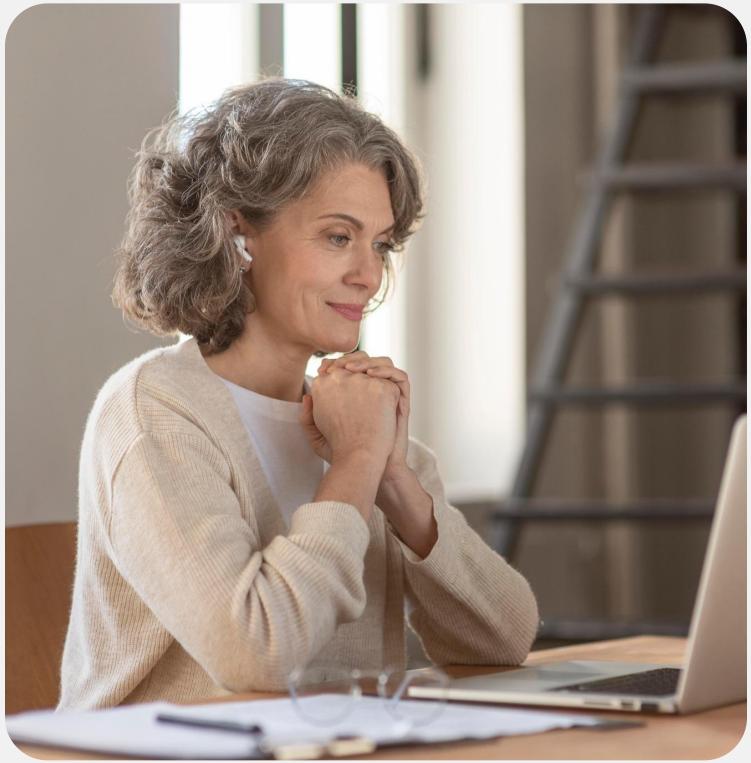
Выполните задание

Найдите ошибку в коде

```
class Box:
    def __init__(self, weight):
        self.weight = weight

    def __add__(self):
        return Box(self.weight + 1)
```

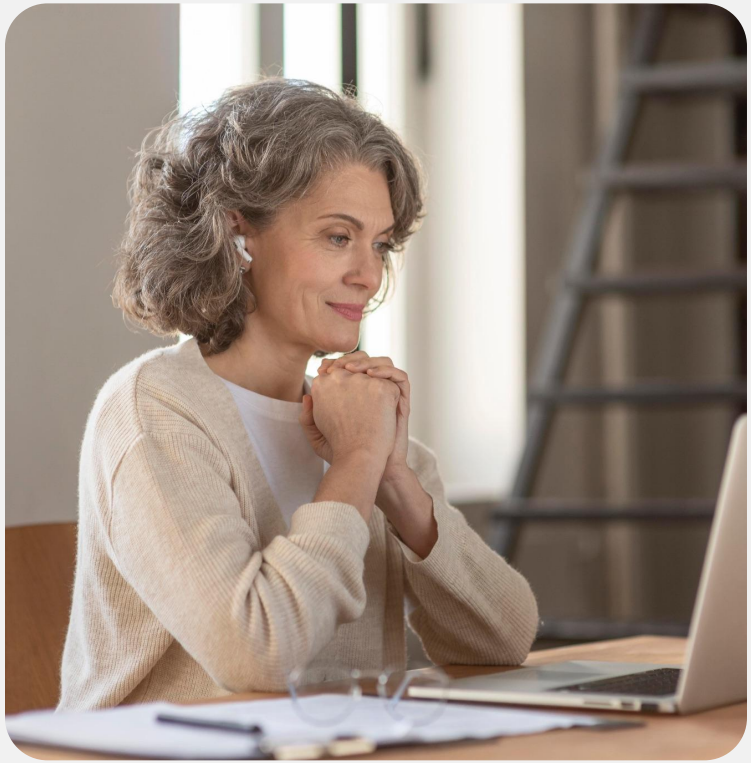
Ответ: у метода `__add__` должно быть два аргумента (`self`, `other`).



Выберите верный вариант ответа

Укажите, что верно о методе __call__:

a. Делает объект вызываемым, как функцию
b. Используется только для арифметических операций
c. Может принимать аргументы



Выберите верный вариант ответа

Укажите, что верно о методе __call__:

a. Делает объект вызываемым, как функцию
b. Используется только для арифметических операций
c. Может принимать аргументы

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

ДОМАШНЕЕ ЗАДАНИЕ

Домашнее задание

1. **Электронное письмо**

Реализуйте класс `Email`, который представляет электронное письмо. Каждое письмо должно содержать:

- `sender` — адрес отправителя
- `recipient` — адрес получателя
- `subject` — тема письма
- `body` — текст письма
- `date` — дата отправки

Класс должен поддерживать:

- Сравнение писем по дате
- Преобразование письма в строку
- Получение длины текста письма
- Проверку на наличие текста в письме или не состоит ли текст только из пробелов

Домашнее задание

1. **Электронное письмо**

Пример использования:

```
e1 = Email("alice@example.com", "bob@example.com",
"Meeting", "Let's meet at 10am", datetime(2024, 6, 10))
e2 = Email("bob@example.com", "alice@example.com",
"Report", "", datetime(2024, 6, 11))

print(e1)
print(e2)
print("Length:", len(e1))
print("Has text:", bool(e1))
print("Is newer:", e2 > e1)
```

Пример вывода:

```
From: alice@example.com
To: bob@example.com
Subject: Meeting
- Let's meet at 10am -

From: bob@example.com
To: alice@example.com
Subject: Report
-  -

Length: 18
Has text: True
Is newer: True
```

Домашнее задание

2. Класс для работы с деньгами

Создайте класс `Money`, в котором можно:

- складывать и вычитать объекты через операторы `+` и `-`
- выводить объект как строку в виде `"$<amount>"`
- при сложении и вычитании возвращается **новый объект**
- если вычитание приводит к отрицательному значению — вернуть `0`

Пример использования:

```
money1 = Money(100)
money2 = Money(50)

print(money1 + money2)
print(money1 - money2)
print(money2 - money1)
```

Пример вывода:

```
$150
$50
$0
```

Заключение