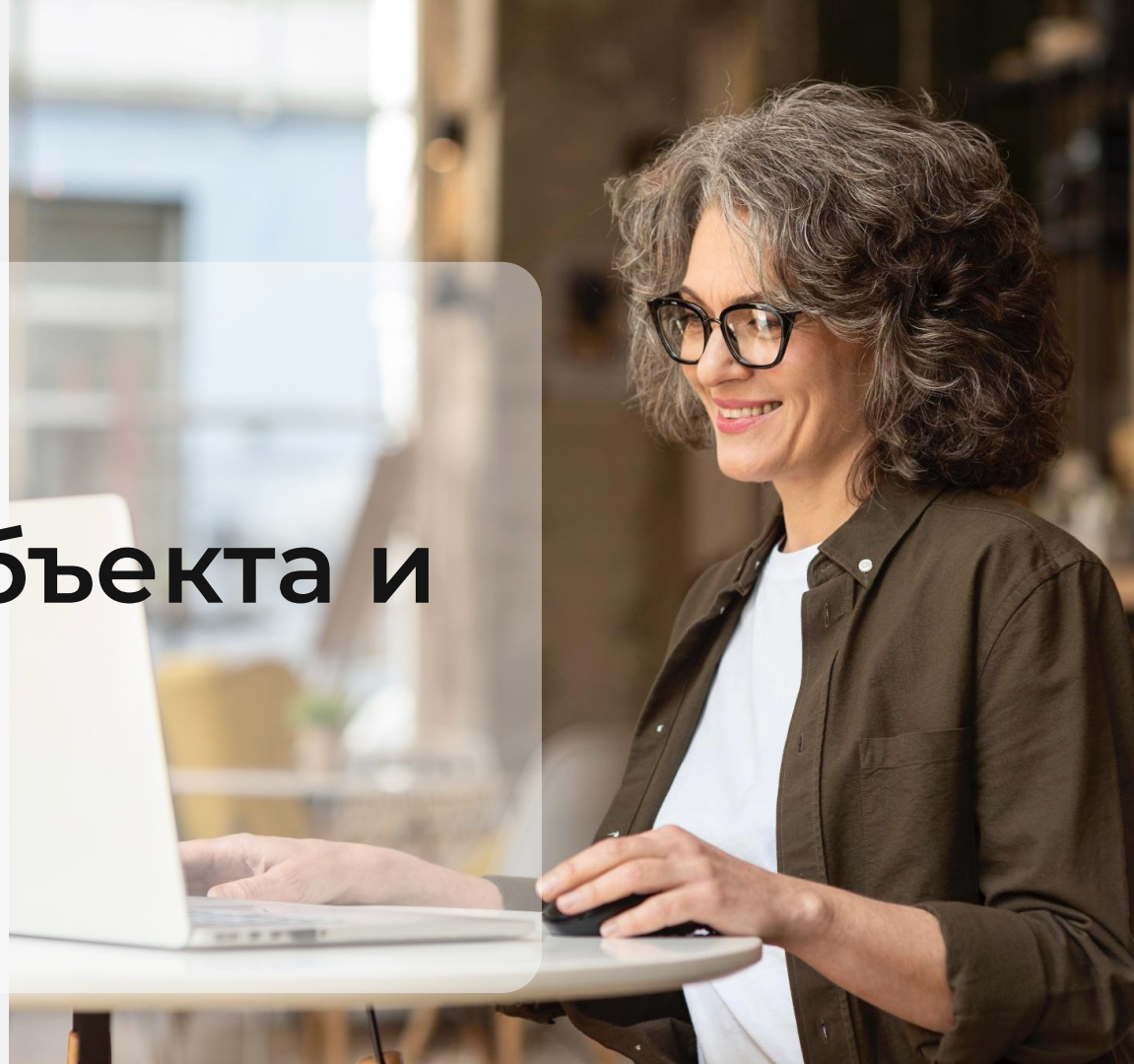
IT CAREER HUB

Python

Атрибуты объекта и класса



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] ООП
- [x] ООП vs функциональный подход
- [x] Создание класса
- [x] Метод __init__
- [x] Поля, методы и атрибуты
- [x] self
- [x] Создание объекта
- [x] Доступ к полям объекта
- [x] Методы класса

План занятия

- Поля класса
- Доступ к полям
- Поля объекта по умолчанию
- Классовые методы
- Статические методы
- Магический метод __str__
- Магический метод __repr__

ICH IT CAREER HUB

ОСНОВНОЙ БЛОК

IT CAREER HUB

Поля класса

Полезно знать

ICH
IT CAREER HUB

Поле класса (атрибут класса)

Это переменная, которая принадлежит самому классу, а не отдельному объекту. Все экземпляры класса могут получить к ней доступ, и при этом значение хранится в одном месте — внутри класса.

Поле класса

Синтаксис

```
class ClassName:
    class_attribute = value  # поле
класса
```

Пояснение

- `class_attribute` — поле класса, общее для всех объектов
- `value` — значение этого поля

Пример

```
class Book:
    library_name = "Central Library"  # поле класса

    def __init__(self, title, author):
        self.title = title    # поле объекта
        self.author = author  # поле объекта
```

Использование полей класса

- Для хранения настроек, категорий, счётчиков, общих значений
- Когда значение одинаково для всех объектов

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

Доступ к полям

Изменение полей класса

Поля класса можно читать и изменять как внутри класса, так и снаружи — через класс или объект. Но при этом поведение может отличаться — особенно при изменении значений.

1. Чтение поля класса

- Напрямую через класс: `ClassName.attribute`
- Через объект: `object_name.attribute`
- Внутри методов класса — через `self.attribute` или `ClassName.attribute`

Пример чтения поля класса

```
class Book:
    library_name = "Central Library"  # поле класса

    def __init__(self, title, author):
        self.title = title
        self.author = author

    def show_library(self):
        print(self.library_name)  # доступ через self
        print(Book.library_name)  # доступ через имя класса

print(Book.library_name)  # доступ через класс
book = Book("1984", "George Orwell")
print(book.library_name)  # доступ через объект
book.show_library()       # доступ изнутри
```

2. Изменение значения через класс

```
class Book:
    library_name = "Central Library"  # поле класса

    def __init__(self, title, author):
        self.title = title
        self.author = author

book = Book("1984", "George Orwell")
book2 = Book("Brave New World", "Aldous Huxley")
Book.library_name = "City Library"
print(Book.library_name)
print(book.library_name)      # у объекта тоже изменилось
print(book2.library_name)     # у объекта тоже изменилось
```

3. Создание поля через объект

Если попытаться изменить значение поля через объект, на самом деле будет создано **новое поле в конкретном объекте**, не затронув поле класса.

Пример создания поля через объект

```
class Book:
    library_name = "Central Library"  # поле класса

    def __init__(self, title, author):
        self.title = title
        self.author = author

book = Book("1984", "George Orwell")
book2 = Book("Brave New World", "Aldous Huxley")

book.library_name = "Private Shelf"  # создаёт новое поле в book
print(book.library_name)  # Private Shelf — новое значение в объекте
print(Book.library_name)  # поле класса не изменилось
print(book2.library_name) # также прочитает поле класса
```

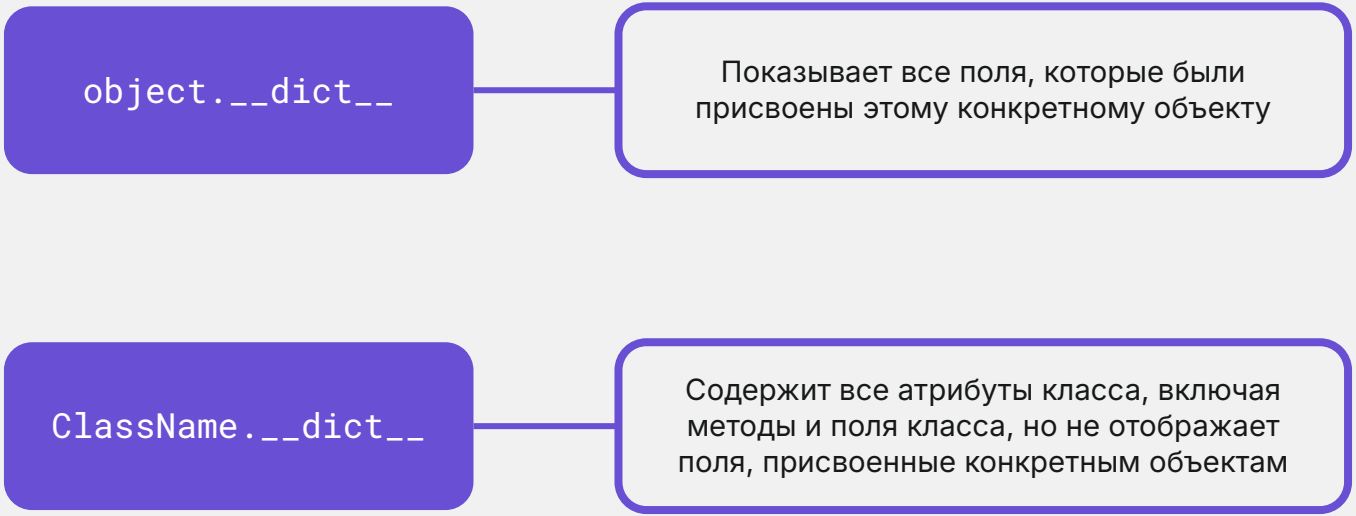
4. Механизм поиска атрибутов

При обращении к полю через объект (`object.attribute`), Python сначала **проверяет, есть ли такое поле у самого объекта**. Если не находит — продолжает поиск **в классе**, из которого объект был создан.

5. Магическое поле __dict__

`__dict__` — это **специальный атрибут** (магическое поле) объекта или класса, в котором хранятся **все явно заданные атрибуты** в виде словаря.

Магическое поле __dict__



Пример использования __dict__

```
class Book:
    library_name = "Central Library"  # поле класса

    def __init__(self, title, author):
        self.title = title
        self.author = author

book = Book("1984", "George Orwell")
book2 = Book("Brave New World", "Aldous Huxley")

book.library_name = "Private Shelf"  # создаёт новое поле в book

print(book.__dict__)  # у book1 появилось собственное поле
print(book2.__dict__) # у book2 такого поля нет
print(Book.__dict__)  # поля и методы класса
```

6. Добавление новых полей извне класса

Python позволяет **в любой момент добавить новое поле** как к классу, так и к объекту. Но **делать это не рекомендуется** — такие поля **неочевидны**, их сложно отследить, и они **могут привести к ошибкам**, особенно в больших проектах или при командной разработке.

Пример создания полей вне класса

```
class Book:
    library_name = "Central Library"  # поле класса

    def __init__(self, title, author):
        self.title = title
        self.author = author

book = Book("1984", "George Orwell")
book2 = Book("Brave New World", "Aldous Huxley")

book.book_color = "black"            # создание поля объекта вне класса
Book.default_language = "eng"        # создание поля класса вне класса
print(book.book_color)
# print(book2.book_color)            # ошибка, такого поля нет
print(book.default_language)
```

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

Поля объекта по умолчанию

Важно

В Python можно назначить полю объекта класса определенное базовое значение, которое можно изменить. Это реализуется через **значения по умолчанию** в методе `__init__` — ведь он работает как обычная функция.

Пример: значение по умолчанию

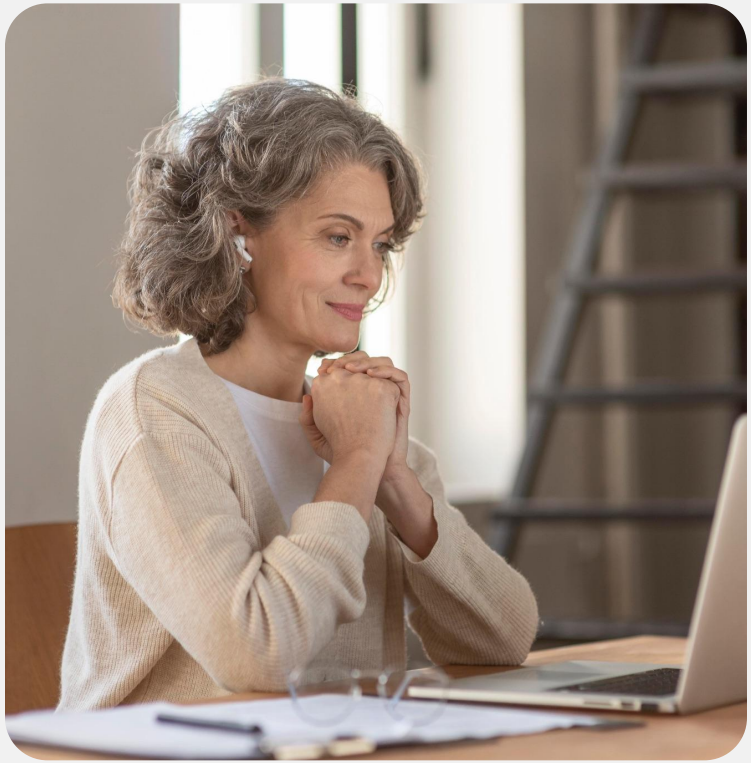
```
class Book:
    def __init__(self, title, author, language="English"):
        self.title = title
        self.author = author
        self.language = language

book1 = Book("1984", "George Orwell")
# язык по умолчанию
book2 = Book("Cien años de soledad", "Gabriel García Márquez", "Spanish")
# другой язык

print(book1.language)
print(book2.language)
```

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

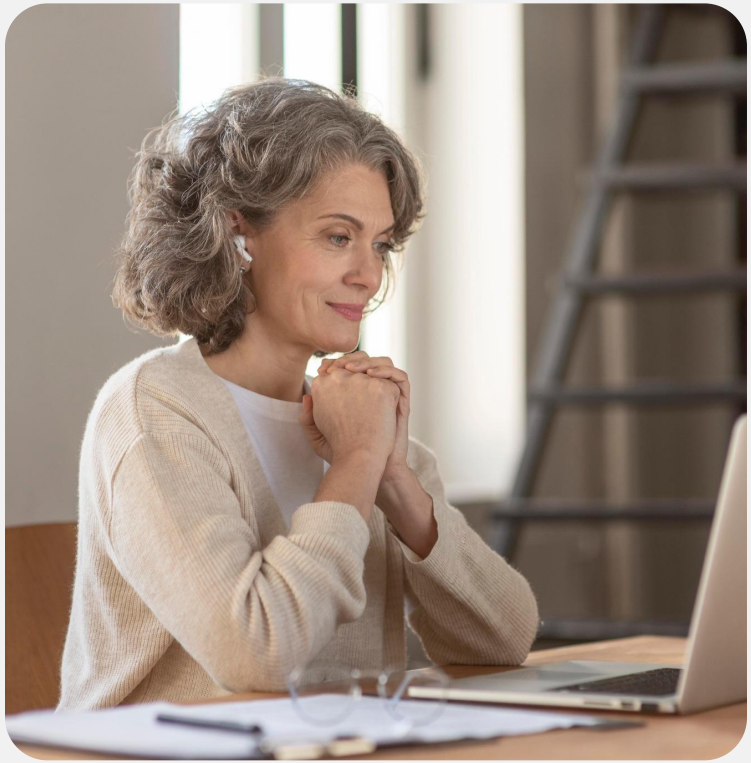
ЗАДАНИЯ



Выберите верный вариант ответа

1. **Укажи верные утверждения о полях класса:**

a. Поле класса общее для всех объектов
b. Поле класса создаётся внутри метода `__init__()`
c. Изменение поля через объект всегда меняет поле класса
d. Поле класса можно прочитать через объект



Выберите верные варианты ответа

1. **Укажи верные утверждения о полях класса:**

a. Поле класса общее для всех объектов
b. Поле класса создаётся внутри метода `__init__()`
c. Изменение поля через объект всегда меняет поле класса
d. Поле класса можно прочитать через объект

IT CAREER HUB

ВОПРОСЫ

ICH IT CAREER HUB

Классовые методы

Полезно знать

ICH
IT CAREER HUB

Классовый метод

Это метод, который работает не с конкретным объектом, а с самим классом. Он получает доступ не к `self`, а к `cls` — ссылке на класс, из которого был вызван.

Важно

Чтобы объявить классовый метод, используется декоратор `@classmethod`.

Применение классовых методов

- Работа с полями класса (например, счётчиками или общими настройками)
- Создание объектов особым образом — через альтернативные конструкторы
- Добавление поведения, относящегося к самому классу, а не к отдельному экземпляру

Классовые методы

Синтаксис

```
class ClassName:
    @classmethod
    def method_name(cls, ...):
        ...
```

Пояснение

- `@classmethod` — декоратор, помечающий метод как классовый
- `cls` — ссылка на сам класс (аналог `self`, но для класса)

Пример: счётчик созданных объектов

```python
class Book:
    total_books = 0  # поле класса

    def __init__(self, title, author):
        self.title = title
        self.author = author
        Book.total_books += 1

    @classmethod
    def show_total(cls):
        print(f"Total books: {cls.total_books}")
        # print(cls.title)  # Нельзя обратиться к полю объекта

Book.show_total()     # Вызов через класс
book1 = Book("1984", "George Orwell")
book2 = Book("Brave New World", "Aldous Huxley")

Book.show_total()     # Вызов через класс
book1.show_total()    # Вызов через объект
```

Пример: альтернативный конструктор (создание объекта по шаблону)

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

    @classmethod
    def from_string(cls, data):
        title, author = data.split(" - ")
        # cls — это Book, поэтому cls(title, author) эквивалентно
Book(title, author)
        return cls(title, author)  # создаёт и возвращает новый объект

book = Book.from_string("1984 - George Orwell")
print(book.title)
print(book.author)
```

Пример: поведение, связанное с классом

```
class Book:
    default_format = "PDF"
    default_language = "en"

    @classmethod
    def show_defaults(cls):
        print(f"Format: {cls.default_format}, Language: {cls.default_language}")

    @classmethod
    def reset_defaults(cls):
        cls.default_format = "PDF"
        cls.default_language = "en"

Book.default_format = "EPUB"  # Изменение формата
Book.show_defaults()          # Просмотр текущих настроек
Book.reset_defaults()         # Сброс до базовых настроек
Book.show_defaults()          # Просмотр текущих настроек
```

IT CAREER HUB

ВОПРОСЫ

ICH IT CAREER HUB

Статические методы

Полезно знать

ICH
IT CAREER HUB

Статический метод

Это метод, который не зависит ни от объекта, ни от самого класса. Он работает как обычная функция, но помещён внутрь класса для логической связанности с ним.

Важно

Чтобы объявить статический метод, используется декоратор @staticmethod

Использование статических методов

- Когда функция логически относится к классу, но не использует ни `self`, ни `cls`
- Чтобы сгруппировать связанную логику внутри класса, а не выносить её в отдельные функции вне класса

Статические методы

Синтаксис

```
class ClassName:
    @staticmethod
    def method_name(...):
        ...
```

Пояснение

- `@staticmethod` — декоратор, превращающий метод в статический
- Такой метод **не получает ни** `self`**, ни** `cls`

Пример: вспомогательная функция внутри класса

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

    @staticmethod
    def is_title_valid(title):
        return isinstance(title, str) and len(title) > 0

print(Book.is_title_valid("1984"))       # True
print(Book.is_title_valid(""))           # False
print(Book.is_title_valid(123))          # False
```

Такой метод можно вызывать и через класс (`Book.is_title_valid()`), и через объект — результат будет одинаков

Пример: логически связанные, но независимые операции

```
class MathHelper:
    @staticmethod
    def square(x):
        return x * x

    @staticmethod
    def cube(x):
        return x * x * x

print(MathHelper.square(5))
print(MathHelper.cube(3))
```

Сравнение типов методов

Тип метода	Обычный метод	Классовый метод	Статический метод
Декоратор	–	`@classmethod`	`@staticmethod`
Первый аргумент	`self`	`cls`	–
Доступ	К полям и методам **объекта**	К полям и методам **класса**	–
Когда использовать	Когда метод работает с конкретным экземпляром	Когда метод работает с классом или создаёт объекты по шаблону	Когда функция логически относится к классу, но ничего не использует

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

Магический метод __str__

Полезно знать

ICH
IT CAREER HUB

Метод __str__()

Это магический метод, который Python вызывает, когда объект выводится через `print()`, чтобы получить **строковое представление объекта**, которое будет показано пользователю.

Важно

По умолчанию Python выводит **техническую информацию** — тип и адрес объекта в памяти, что не информативно. Определение метода `__str__()` позволяет **вернуть читаемую строку** о содержимом объекта.

Пример: техническая информация

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

book = Book("1984", "George Orwell")
print(book)
```

Пример: содержимое объекта

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

    def __str__(self):
        return f"{self.title} by {self.author}"

book = Book("1984", "George Orwell")
print(book)       # вызывает __str__
print(str(book))  # вызывает __str__
```

Особенности магического метода __str__

- Не должен ничего выводить сам — только возвращать строку
- Вызывается автоматически при `print(obj)` или `str(obj)`

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

Магический метод __repr__

Полезно знать

ICH
IT CAREER HUB

Метод __repr__

Это магический метод, который возвращает **техническое представление объекта**, удобное для **разработчика**. Оно используется при выводе в коллекциях, отладке и в интерактивной оболочке

Пример: использование __repr__

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

    def __repr__(self):
        return f"Book(title={self.title!r}, author={self.author!r})"

book1 = Book("1984", "George Orwell")
book2 = Book("Brave New World", "Aldous Huxley")
print(repr(book1))
books = [book1, book2]
print(books)
```

Особенности магического метода __repr__

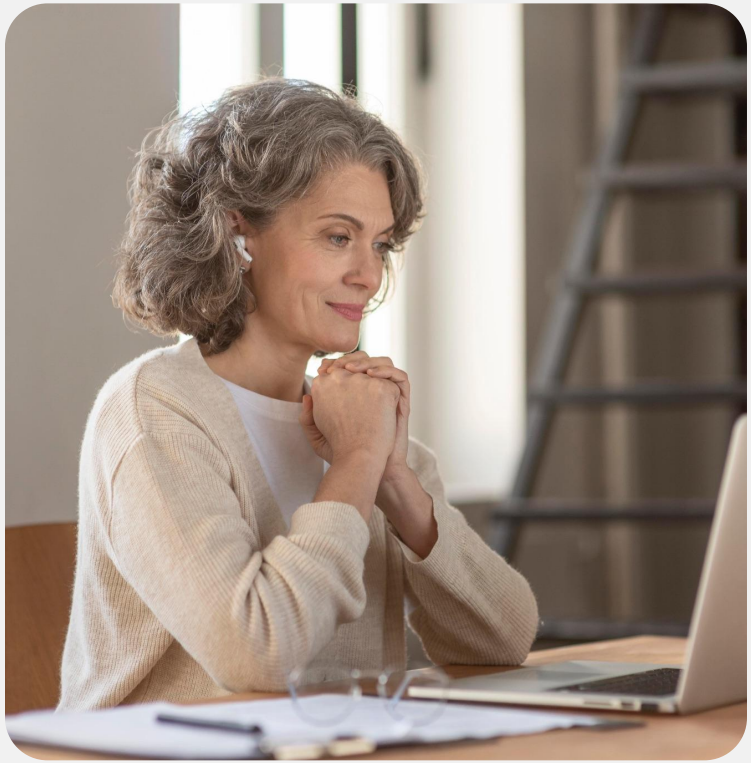
- Если `__str__()` отсутствует, Python использует `__repr__()` как запасной вариант при `print()`
- Может использоваться **в логах**, **интерактивных сессиях** и отладчиках
- В идеале, `__repr__()` должен возвращать строку, из которой **можно воссоздать объект**

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

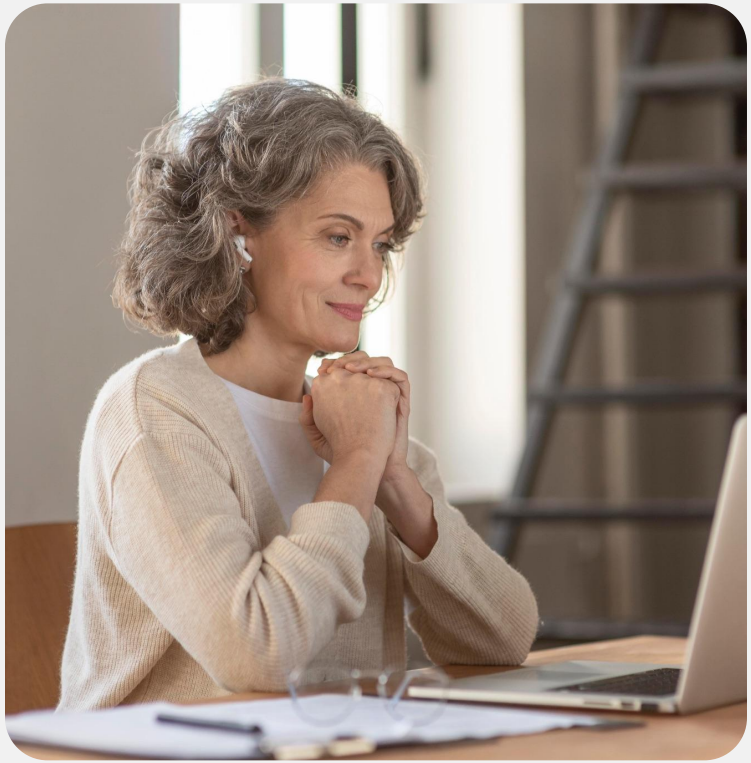
ЗАДАНИЯ



Выберите верные варианты ответа

1. **Укажи верные утверждения о @classmethod:**

a. Метод `@classmethod` получает первым аргументом `self`
b. Метод `@classmethod` может создавать объект класса
c. Метод `@classmethod` имеет доступ к атрибутам конкретного объекта
d. Метод `@classmethod` вызывается только через класс, а не через объект
e. Метод `@classmethod` имеет доступ к классу



Выберите верные варианты ответа

1. **Укажи верные утверждения о @classmethod:**

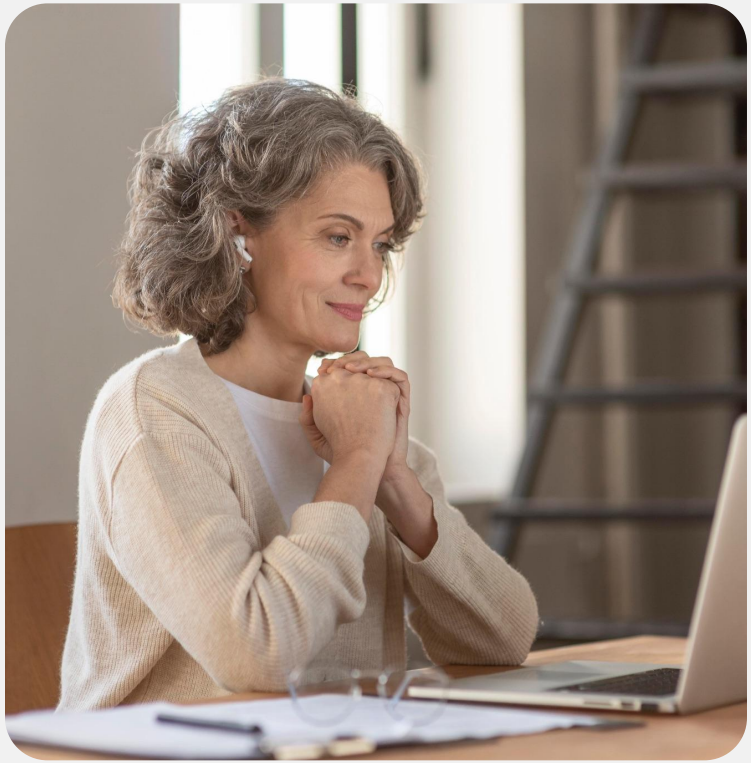
a. Метод `@classmethod` получает первым аргументом `self`

b. Метод `@classmethod` может создавать объект класса

c. Метод `@classmethod` имеет доступ к атрибутам конкретного объекта

d. Метод `@classmethod` вызывается только через класс, а не через объект

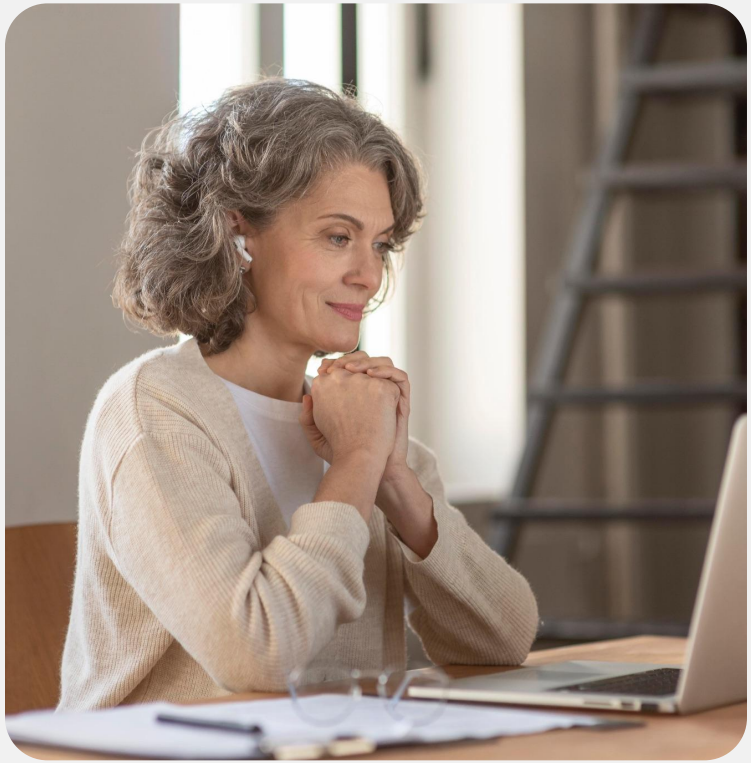
e. Метод `@classmethod` имеет доступ к классу



Выполните задание

2. Найдите ошибку в коде:

```
class Person:
    @staticmethod
    def greet(self):
        print("Hello!")
```



Выполните задание

2. Найдите ошибку в коде:

```
class Person:
    @staticmethod
    def greet(self):
        print("Hello!")
```

Ответ: у статического метода не должно быть параметра `self`

IT CAREER HUB

ВОПРОСЫ

IT CAREER HUB

ПРАКТИЧЕСКАЯ РАБОТА

1. Расстояние между городами

Создайте класс `City`, представляющий город с координатами.

- У каждого города есть поля `name`, `latitude`, `longitude`.
- Добавьте строковое представление объекта.
- Добавьте метод `distance(city1, city2)`, который возвращает кортеж `(latitude, longitude)` между двумя городами.
- Проверьте расстояние между двумя городами.

Пример вывода:

```
City: Berlin (52.52, 13.4)
City: Paris (48.85, 2.35)
Distance: 14.72
```

2. Создание объекта из строки

Доработайте класс `City`.

- Добавьте метод `from_string(data)`, который создаёт объект из строки вида `"Rome:41.89,12.51"`.
- Проверьте создание нового объекта через этот метод и выведите его.

Пример вывода:

```
City: Rome (41.89, 12.51)
```

IT CAREER HUB

ДОМАШНЕЕ ЗАДАНИЕ

Домашнее задание

1. Счётчик экземпляров

Создайте класс `User`, представляющий пользователя.

- При создании должны указываться логин (`username`) и пароль (`password`).
- У класса должно быть поле `total_users`, хранящее общее количество созданных пользователей.
- При каждом создании нового объекта `User`, счётчик должен увеличиваться.
- Добавьте метод `get_total()`, возвращающий количество пользователей.
- Проверьте, что счётчик работает.

Пример вывода:

```
Total users: 2
```

Домашнее задание

2. Проверка данных пользователя

Доработайте класс `User`.

- Добавьте валидации полей при создании.
- Имя должно быть непустой строкой.
- Пароль должен быть строкой длиной не менее 5 символов.
- Если данные некорректны — выбрасывайте `ValueError`.
- Добавьте строковое представление объекта.
- Проверьте работу класса с разными значениями.

Пример вызова:

```
user1 = User("alice", "secret")
user2 = User("bob", "qwe")
```

Пример вывода:

```
User: alice
  ...
ValueError: Invalid password:
'qwe'.
```

Заключение