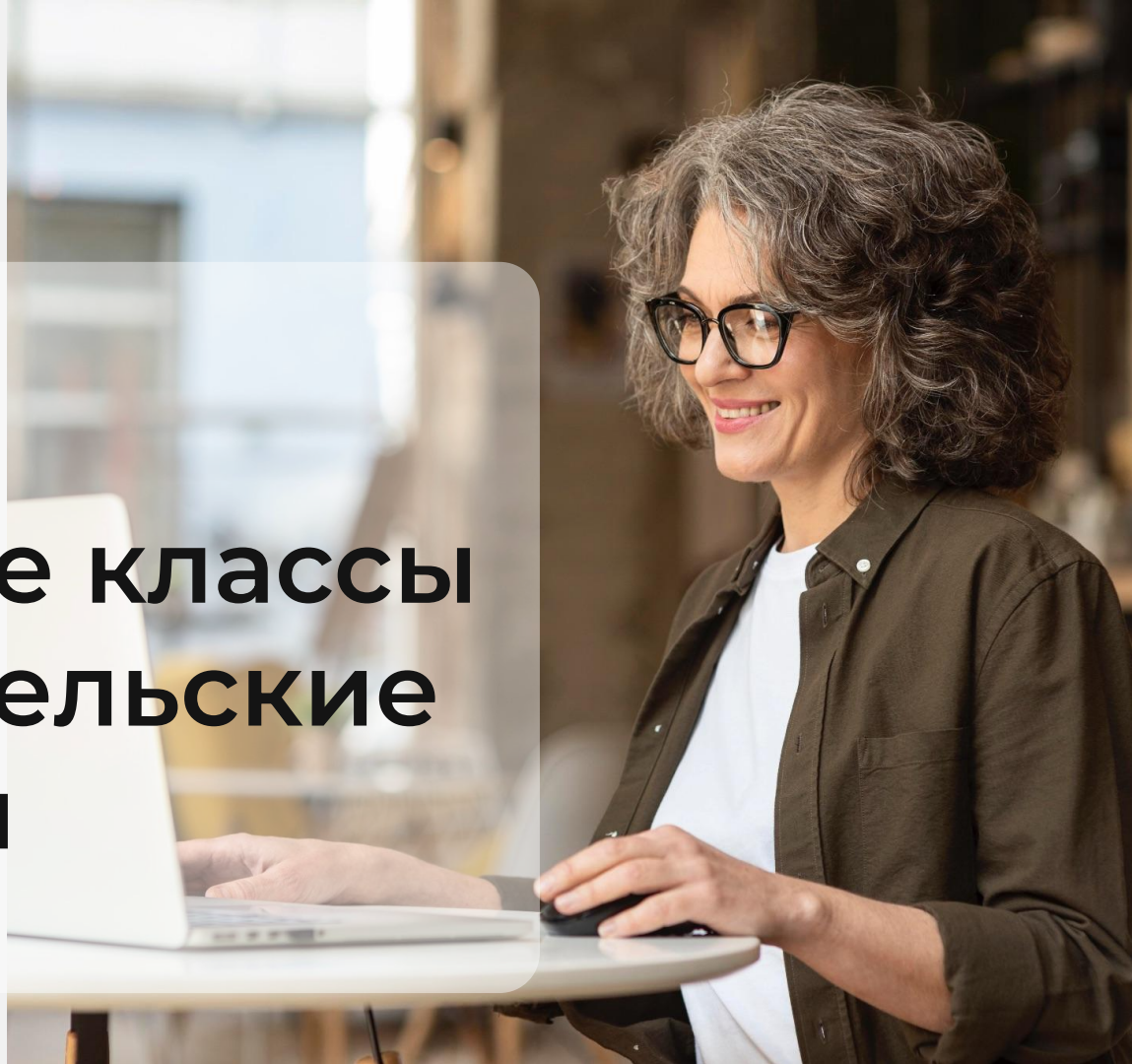


Python

Абстрактные классы и пользовательские исключения



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

-  Инкапсуляция
-  Уровни доступа
-  Защищённые и приватные методы
-  Посмотреть ответ
-  Геттеры и сеттеры
-  Декоратор @property

План занятия

- Абстрактные классы
- Документация классов
- Пользовательские исключения
- Магические методы
- Магические методы итерации



ОСНОВНОЙ БЛОК





Абстрактные классы



Абстрактный класс

Это класс, который не предназначен для создания объектов напрямую. Он служит как шаблон для других классов, задавая структуру и обязательные методы, которые должны быть реализованы в дочерних классах

Назначение абстрактных классов



Определить единый интерфейс для группы классов



Заставить наследников реализовать нужные методы



Описать общую логику, но запретить создание "незаконченных" объектов

Отличие от обычного класса

| Обычный класс | Абстрактный класс |
|--|--|
| Можно создавать объекты | Создание объектов запрещено |
| Все методы можно переопределить, но не обязательно | Некоторые методы обязательны к реализации |
| Используется напрямую | Используется как основа для наследования |

Создание абстрактных классов



Класс должен **наследоваться от** ABC (из модуля abc)



Абстрактные методы помечаются декоратором `@abstractmethod`

Пример

```
from abc import ABC, abstractmethod

class Employee(ABC): # Абстрактный класс

    @abstractmethod
    def work(self):
        pass # Метод без реализации

e = Employee() # TypeError: не реализован абстрактный метод
```

Пример

Создание подкласса, реализующего метод

```
class Programmer(Employee):  
    def work(self):  
        print("Write code")
```

```
p = Programmer()  
p.work()
```

Создание абстрактных классов



Абстрактный класс не может быть использован напрямую — он является **шаблоном**



Класс может **содержать как абстрактные, так и обычные (реализованные) методы**



ВОПРОСЫ





Документация классов

Важно



Документация класса упрощает чтение кода, помогает другим разработчикам и отображается в `help()`. Документация класса пишется в виде **многострочной строки** сразу **под объявлением класса** в тройных кавычках

Пример

```
class Book:
    """
    Represents a book.

    Attributes:
        title (str): The title of the book.
        author (str): The author of the book.

    Methods:
        get_info(): Returns a brief description of the book.
    """

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

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

print(Book.__doc__)
```

Рекомендации по документации



Кратко опишите, что делает класс



Перечислите основные атрибуты и методы



Используйте одинаковый стиль по всему проекту



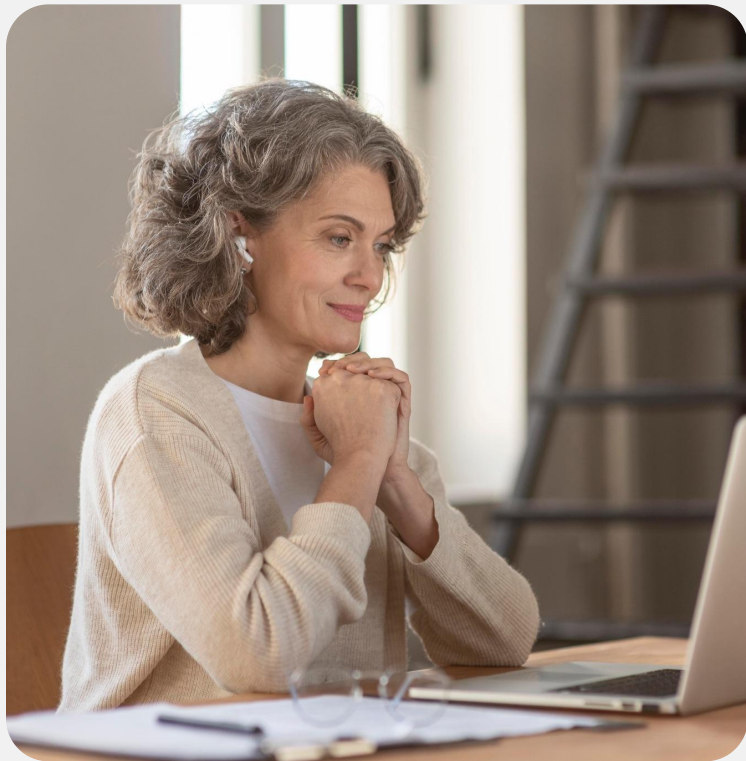
ВОПРОСЫ





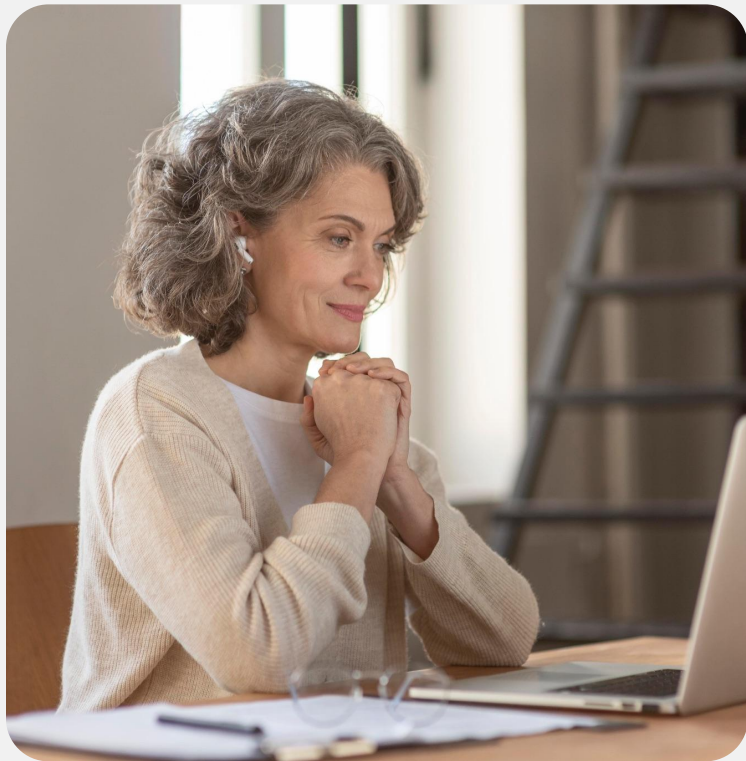
ЗАДАНИЯ





Выберите верные варианты ответа

1. Укажите верные утверждения об абстрактных классах:
 - a. Абстрактный класс можно использовать для создания объектов напрямую
 - b. Абстрактный класс задаёт структуру для наследников
 - c. Абстрактный класс может содержать как реализованные, так и абстрактные методы
 - d. Абстрактный класс запрещает наследование



Выберите верные варианты ответа

1. Укажите верные утверждения об абстрактных классах:
 - a. Абстрактный класс можно использовать для создания объектов напрямую
 - b. Абстрактный класс задаёт структуру для наследников
 - c. Абстрактный класс может содержать как реализованные, так и абстрактные методы
 - d. Абстрактный класс запрещает наследование



Выполните задание

Что произойдет при выполнении следующего кода?

```
from abc import ABC, abstractmethod
```

```
class Animal(ABC):
    @abstractmethod
    def speak(self):
        pass
```

```
class Dog(Animal):
    pass
```

```
d = Dog()
d.speak()
```



Выполните задание

Что произойдет при выполнении следующего кода?

```
from abc import ABC, abstractmethod
```

```
class Animal(ABC):
    @abstractmethod
    def speak(self):
        pass
```

```
class Dog(Animal):
    pass
```

```
d = Dog()
d.speak()
```

Ответ: Произойдет ошибка при попытке создать объект Dog



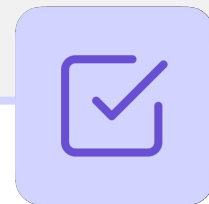
ВОПРОСЫ





Пользовательские исключения

Важно



Пользовательские исключения позволяют точно указать, что именно пошло не так, и делают код более читаемым и контролируемым.

Назначение пользовательских исключений



Ясно отделять ошибки своей логики от стандартных



Точно указывать причину ошибки



Отлавливать нужные исключения в try/except

Создание пользовательских исключений







Синтаксис

```
class CustomError(Exception):  
    pass  
  
raise CustomError("Error message")
```

Пояснение

- CustomError — имя нового класса ошибки (по соглашению заканчивается на Error)
- Exception — базовый класс, от которого наследуется логика

Особенности пользовательских исключений

- 
 Пользовательские исключения должны наследоваться от `Exception` или его подклассов
- 
 Не следует наследоваться от `BaseException` (предназначен для системных исключений)
- 
 Класс может быть пустым (`pass`) или содержать собственную логику
- 
 Названия пользовательских исключений принято заканчивать на `Error`

Пример

Ограничение доступа по возрасту

```
class AccessDeniedError(Exception):
    """Вызывается, если пользователь слишком молод для доступа."""
    pass

def check_age(age):
    if age < 18:
        raise AccessDeniedError("Access denied: age must be at least 18")
    print("Access granted")

try:
    check_age(16)
except AccessDeniedError as e: # можно отловить собственную ошибку
    print("Ошибка:", e)
```

- Класс `AccessDeniedError` **наследуется от** `Exception` — это базовый подход для пользовательских ошибок
- Внутри `try/except` можно **перехватить собственную ошибку**, не затрагивая другие

Дополнительные поля в исключениях



Позволяют делать исключения более информативными



Дают возможность передавать в обработчик **контекст ошибки**



Могут облегчить **логирование и отладку**

Пример

Пользовательская ошибка с данными

```
class AccessDeniedError(Exception):
    """Вызывается, если пользователь слишком молод для доступа."""

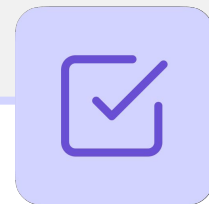
    def __init__(self, age):
        self.age = age
        super().__init__(f"Access denied: age {age} is too low")

def check_age(age):
    if age < 18:
        raise AccessDeniedError(age)
    print("Access granted")

try:
    check_age(15)
except AccessDeniedError as e:
    print("Error:", e)
    print("Age:", e.age)
```

- Можно сохранять дополнительные данные внутри исключения
- Удобно при логировании, отладке и тестировании
- `super().__init__()` передаёт сообщение в базовый Exception, чтобы его можно было отобразить обычным `print(e)`

Важно



Исключения, которые относятся к определённому типу ошибки, лучше делать **наследником подходящего встроенного исключения**, а не просто Exception

Пример

Собственная ошибка значения (ValueError)

```
class TemperatureTooLowError(ValueError):
    pass

def set_temperature(value):
    if value < -273.15:
        raise TemperatureTooLowError("Temperature cannot be below absolute zero")
    print(f"Temperature set to {value}°C")

set_temperature(15)
set_temperature(-300) # вызовет ошибку
```



ВОПРОСЫ





ЗАДАНИЯ

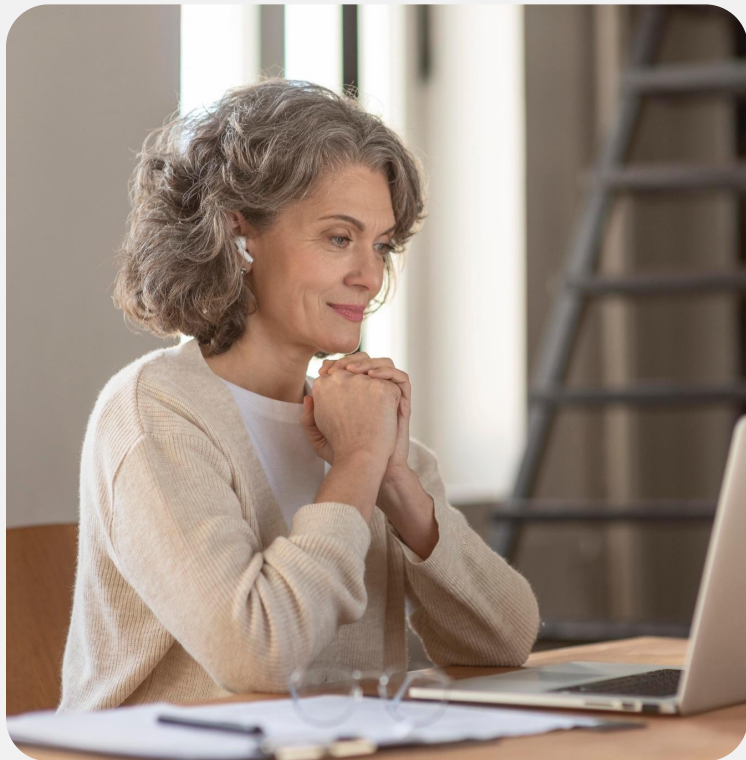




Выберите верный вариант ответа

Укажите, в каких случаях имеет смысл создавать пользовательские исключения:

- a. Чтобы обработать ошибку, связанную с некорректным типом
- b. Чтобы явно разделить ошибки бизнес-логики от стандартных ошибок
- c. Чтобы ускорить выполнение программы



Выберите верный вариант ответа

Укажите, в каких случаях имеет смысл создавать пользовательские исключения:

- a. Чтобы обработать ошибку, связанную с некорректным типом
- b. Чтобы явно разделить ошибки бизнес-логики от стандартных ошибок
- c. Чтобы ускорить выполнение программы



ВОПРОСЫ





Магические методы

Магические методы



Управляют созданием и инициализацией объектов



Отвечают за представление объекта



Определяют поведение при сравнении, арифметике, в коллекциях и т.п.



Позволяют делать объекты итерируемыми, вызываемыми и т.п.

Примеры ситуаций, где вызываются магические методы

| Выражение | Вызываемый метод |
|--|-----------------------------|
| <code>str(obj)</code> или <code>print()</code> | <code>__str__()</code> |
| <code>len(obj)</code> | <code>__len__()</code> |
| <code>obj1 == obj2</code> | <code>__eq__()</code> |
| <code>obj1 + obj2</code> | <code>__add__()</code> |
| <code>item in obj</code> | <code>__contains__()</code> |
| <code>obj()</code> | <code>__call__()</code> |
| <code>bool(obj)</code> | <code>__bool__()</code> |

Магические методы итерации

| Метод | Назначение |
|-------------------------|---|
| <code>__iter__()</code> | Возвращает итератор — объект с <code>__next__()</code> |
| <code>__next__()</code> | Возвращает следующее значение или выбрасывает <code>StopIteration</code> |

Пример

Итератор, с нарастающей суммой элементов

```
class CumulativeSum:
    def __init__(self, numbers):
        self.numbers = numbers
        self.index = 0
        self.total = 0

    def __iter__(self):
        return self

    def __next__(self):
        if self.index >= len(self.numbers):
            raise StopIteration
        self.total +=
self.numbers[self.index]
        self.index += 1
        return self.total
```

1

```
data = [3, 5, 2, 4]
acc = CumulativeSum(data)
```

```
for value in acc:
    print(value)
```

2



ВОПРОСЫ





ЗАДАНИЯ

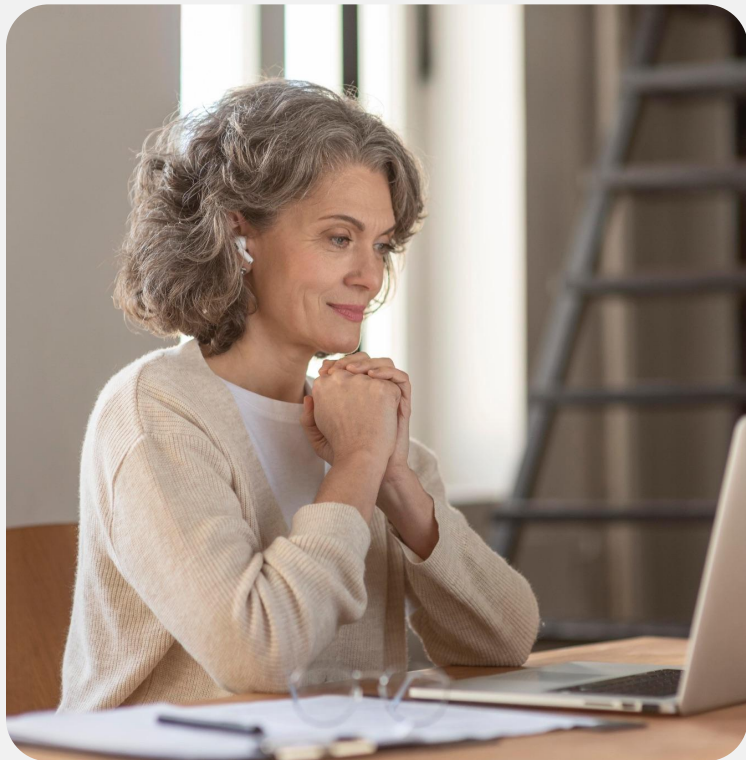




Выберите верный вариант ответа

В каких случаях необходимо определить метод `__iter__()`?

- a. Когда объект должен поддерживать итерацию через `for`
- b. Чтобы объект можно было передавать в функцию `len()`



Выберите верный вариант ответа


В каких случаях необходимо определить метод `__iter__()`?

- а. Когда объект должен поддерживать итерацию через `for`
- б. Чтобы объект можно было передавать в функцию `len()`




ВОПРОСЫ

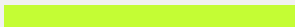




ПРАКТИЧЕСКАЯ РАБОТА



1. Онлайн-платёжные системы



Создайте абстрактный класс `PaymentProcessor`.

- В классе должен быть метод `pay(amount)`.
- Реализуйте два класса:
 - `PaypalPayment`, который печатает `"Paid <amount> via PayPal"`.
 - `CreditCardPayment`, который печатает `"Paid <amount> via Credit Card"`.

2. Проверка платежей

Доработайте систему:

- Создайте пользовательское исключение `InvalidPaymentError`.
- В каждом платёжном классе метод `pay(amount)` должен проверять сумму:
 - Если сумма меньше или равна нулю, выбрасывать `InvalidPaymentError`.
 - Иначе проводить платёж.



ДОМАШНЕЕ ЗАДАНИЕ



Домашнее задание

1. Фигуры и площади

Создайте абстрактный класс Shape.

- В классе должен быть метод `area()`, который возвращает площадь фигуры.
- Реализуйте два класса:
 - Circle, который принимает радиус.
 - Rectangle, который принимает ширину и высоту.

Домашнее задание

2. Проверка размеров фигур

Доработайте фигуры:

- Добавьте проверку в конструкторы `Circle` и `Rectangle`, чтобы значения были **положительными**.
- Если передано отрицательное или нулевое значение, выбрасывайте пользовательское исключение `InvalidSizeError`.

Заключение

