Университет Сириус Колледж

Объектно-ориентированное программирование Лекция 4. Полиморфизм, Абстрактные классы и Исключения

Понятие полиморфизма, переопределение методов. Утиная типизация. Абстрактные классы и методы (модуль `abc`), интерфейсы. Обработка исключений: `try-except-finally`, пользовательские исключения.

Университет Сириус Колледж

Принципы ООП

1. Абстракция
2. Инкапсуляция
3. Наследование
4. Полиморфизм

Университет Сириус Колледж

Полиморфизм

Полиморфизм - это свойство объектов разных классов реагировать на одинаковые вызовы методов по-разному.

> *Суть: один интерфейс — множество реализаций.*

```
    def area(self):
        return 3.14 * self.r ** 2

class Rectangle:
    def area(self):
        return self.w * self.h

# Полиморфный вызов
for shape in [Circle(), Rectangle()]:
```

Университет Сириус Колледж

Полиморфизм

Полиморфизм - это свойство объектов разных классов реагировать на одинаковые вызовы методов по-разному.

> *Суть: один интерфейс — множество реализаций.*

```
    def area(self):
        return 3.14 * self.r ** 2

class Rectangle:
    def area(self):
        return self.w * self.h

# Полиморфный вызов
for shape in [Circle(), Rectangle()]:
```

Университет Сириус Колледж

Полиморфизм

Полиморфизм - это свойство объектов разных классов реагировать на одинаковые вызовы методов по-разному.

> *Суть: один интерфейс — множество реализаций.*

```
        def area(self):
            return 3.14 * self.r ** 2

class Rectangle:
    def area(self):
        return self.w * self.h

# Полиморфный вызов
for shape in [Circle(), Rectangle()]:
    print(shape.area())
```

Университет Сириус Колледж

Переопределение методов

Переопределение (overriding) — это замещение метода базового класса в подклассе новой реализацией.

```
    def speak(self):
        print("Some sound")

class Dog(Animal):
    def speak(self):
        print("Bark")

animal = Animal()
dog = Dog()

animal.speak()  # Some sound
```

Университет Сириус Колледж

Переопределение методов

Переопределение (overriding) — это замещение метода базового класса в подклассе новой реализацией.

```
class Animal:
    def speak(self):
        print("Some sound")

class Dog(Animal):
    def speak(self):
        print("Bark")

animal = Animal()
dog = Dog()

animal.speak()  # Some sound
```

Университет Сириус Колледж

Переопределение методов

Переопределение (overriding) — это замещение метода базового класса в подклассе новой реализацией.

```
class Animal:
    def speak(self):
        print("Some sound")

class Dog(Animal):
    def speak(self):
        print("Bark")

animal = Animal()
dog = Dog()

animal.speak()  # Some sound
```

Университет Сириус Колледж

Переопределение методов

Переопределение (overriding) — это замещение метода базового класса в подклассе новой реализацией.

```
    def speak(self):
        print("Some sound")

class Dog(Animal):
    def speak(self):
        print("Bark")

animal = Animal()
dog = Dog()

animal.speak()  # Some sound
```

Университет Сириус Колледж

Переопределение методов

Переопределение (overriding) — это замещение метода базового класса в подклассе новой реализацией.

```
    def speak(self):
        print("Some sound")

class Dog(Animal):
    def speak(self):
        print("Bark")

animal = Animal()
dog = Dog()

animal.speak()  # Some sound
dog.speak()     # Bark
```

Университет Сириус Колледж

Динамическая диспетчеризация

Python выбирает реализацию метода во время выполнения (runtime). Это обеспечивает гибкость, но требует осознанного проектирования интерфейсов.

Реализуется через поиск метода по цепочке наследования (**MRO — Method Resolution Order**).

Университет Сириус Колледж

Динамическая диспетчеризация

```python
class Animal:
    def speak(self):
        print("Неопределенный звук животного")

class Dog(Animal):
    def speak(self):
        print("Гав!")

class Cat(Animal):
    def speak(self):
        print("Мяу!")

def make_animal_speak(animal: Animal):
    print(f"Пришел объект типа {type(animal).__name__}. Он говорит:")
    animal.speak()
```

Университет Сириус Колледж

Динамическая диспетчеризация

```python
class Animal:
    def speak(self):
        print("Неопределенный звук животного")

class Dog(Animal):
    def speak(self):
        print("Гав!")

class Cat(Animal):
    def speak(self):
        print("Мяу!")

def make_animal_speak(animal: Animal):
    print(f"Пришел объект типа {type(animal).__name__}. Он говорит:")
    animal.speak()
```

Университет Сириус Колледж

Динамическая диспетчеризация

```python
class Animal:
    def speak(self):
        print("Неопределенный звук животного")

class Dog(Animal):
    def speak(self):
        print("Гав!")

class Cat(Animal):
    def speak(self):
        print("Мяу!")

def make_animal_speak(animal: Animal):
    print(f"Пришел объект типа {type(animal).__name__}. Он говорит:")
    animal.speak()
```

Университет Сириус Колледж

Динамическая диспетчеризация

```
        print("Неопределенный звук животного")

class Dog(Animal):
    def speak(self):
        print("Гав!")

class Cat(Animal):
    def speak(self):
        print("Мяу!")

def make_animal_speak(animal: Animal):
    print(f"Пришел объект типа {type(animal).__name__}. Он говорит:")
    animal.speak()
    print("-" * 20)

```

Университет Сириус Колледж

Динамическая диспетчеризация

```
8
9  class Cat(Animal):
10     def speak(self):
11         print("Мяу!")
12
13 def make_animal_speak(animal: Animal):
14     print(f"Пришел объект типа {type(animal).__name__}. Он говорит:")
15     animal.speak()
16     print("-" * 20)
17
18 generic_animal = Animal()
19 my_dog = Dog()
20 my_cat = Cat()
21
22 make_animal_speak(generic_animal)
```

Университет Сириус Колледж

Динамическая диспетчеризация

```
6       def speak(self):
7           print("Гав!")
8
9   class Cat(Animal):
10      def speak(self):
11          print("Мяу!")
12
13  def make_animal_speak(animal: Animal):
14      print(f"Пришел объект типа {type(animal).__name__}. Он говорит:")
15      animal.speak()
16      print("-" * 20)
17
18  generic_animal = Animal()
19  my_dog = Dog()
20  my_cat = Cat()
```

Университет Сириус Колледж

Динамическая диспетчеризация

```
8
9  class Cat(Animal):
10     def speak(self):
11         print("Мяу!")
12
13 def make_animal_speak(animal: Animal):
14     print(f"Пришел объект типа {type(animal).__name__}. Он говорит:")
15     animal.speak()
16     print("-" * 20)
17
18 generic_animal = Animal()
19 my_dog = Dog()
20 my_cat = Cat()
21
22 make_animal_speak(generic_animal)
```

Университет Сириус Колледж

Динамическая диспетчеризация

```
10      def speak(self):
11          print("Мяу!")
12
13  def make_animal_speak(animal: Animal):
14      print(f"Пришел объект типа {type(animal).__name__}. Он говорит:")
15      animal.speak()
16      print("-" * 20)
17
18  generic_animal = Animal()
19  my_dog = Dog()
20  my_cat = Cat()
21
22  make_animal_speak(generic_animal)
23  make_animal_speak(my_dog)
24  make_animal_speak(my_cat)
```

Университет Сириус Колледж

Утиная типизация

В Python важна не принадлежность к классу, а наличие требуемых методов.

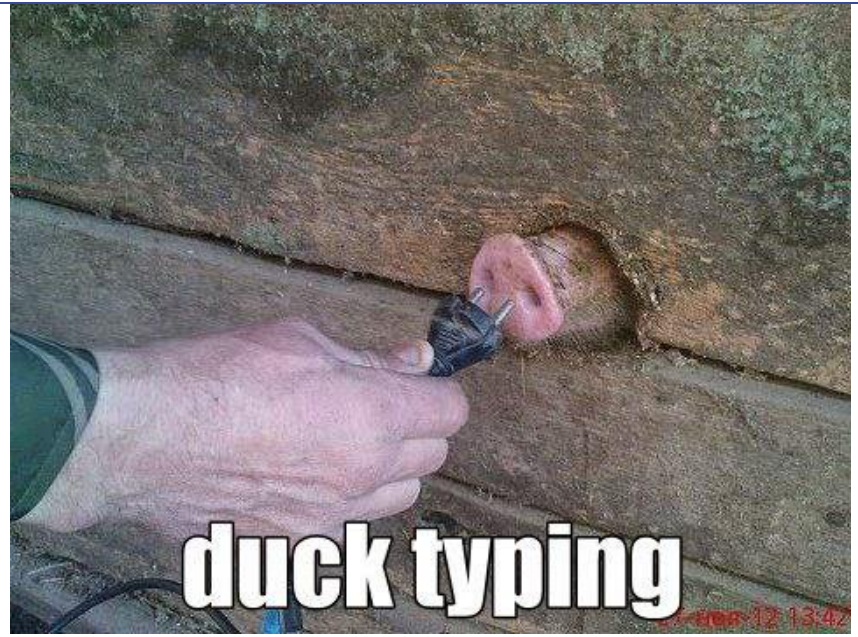
> *“Если объект ходит как утка и крякает как утка, значит, это утка.”*

Университет Сириус Колледж

Утиная типизация

В Python важна не принадлежность к классу, а наличие требуемых методов.

> *“Если объект ходит как утка и крякает как утка, значит, это утка.”*



Университет Сириус Колледж

Утиная типизация

```
1  class Duck:
2      def quack(self): print("Quack!")
3
4  class Person:
5      def quack(self): print("I'm imitating a duck!")
6
7  def make_it_quack(obj):
8      obj.quack()
9
10 make_it_quack(Duck())
11 make_it_quack(Person())
```

Университет Сириус Колледж

Утиная типизация

```
1  class Book:
2      def __init__(self, title, pages_content):
3          self.title = title
4          self.pages = pages_content
5
6      def __len__(self):
7          return len(self.pages)
8
9  class City:
10     def __init__(self, name, streets_list):
11         self.name = name
12         self.streets = streets_list
13
14     def __len__(self):
15         return len(self.streets)
```

Университет Сириус Колледж

Утиная типизация

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

    def __len__(self):
        return len(self.pages)

class City:
    def __init__(self, name, streets_list):
        self.name = name
        self.streets = streets_list

    def __len__(self):
        return len(self.streets)
```

Университет Сириус Колледж

Утиная типизация

```
5
6       def __len__(self):
7           return len(self.pages)
8
9   class City:
10      def __init__(self, name, streets_list):
11          self.name = name
12          self.streets = streets_list
13
14      def __len__(self):
15          return len(self.streets)
16
17  def print_object_length(obj):
18      print(f"Длина объекта: {len(obj)}")
19
```

Университет Сириус Колледж

Утиная типизация

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

    def __len__(self):
        return len(self.pages)

class City:
    def __init__(self, name, streets_list):
        self.name = name
        self.streets = streets_list

    def __len__(self):
        return len(self.streets)
```

Университет Сириус Колледж

Утиная типизация

```
7           return len(self.pages)
8
9   class City:
10      def __init__(self, name, streets_list):
11          self.name = name
12          self.streets = streets_list
13
14      def __len__(self):
15          return len(self.streets)
16
17  def print_object_length(obj):
18      print(f"Длина объекта: {len(obj)}")
19
20  my_book = Book("Война и мир", ["страница 1", "страница 2", "...", "страница 1
21  my city = City("Москва", ["Тверская", "Арбат", "...", "Ленинский проспект"])
```

Университет Сириус Колледж

Утиная типизация

```
11          self.name = name
12          self.streets = streets_list
13
14      def __len__(self):
15          return len(self.streets)
16
17  def print_object_length(obj):
18      print(f"Длина объекта: {len(obj)}")
19
20  my_book = Book("Война и мир", ["страница 1", "страница 2", "...", "страница 1
21  my_city = City("Москва", ["Тверская", "Арбат", "...", "Ленинский проспект"])
22  my_string = "Это просто строка"
23  my_list = [1, 2, 3, 4, 5]
24
25  print object length(my book)
```

Университет Сириус Колледж

Утиная типизация

```
14      def __len__(self):
15          return len(self.streets)
16
17  def print_object_length(obj):
18      print(f"Длина объекта: {len(obj)}")
19
20  my_book = Book("Война и мир", ["страница 1", "страница 2", "...", "страница 1
21  my_city = City("Москва", ["Тверская", "Арбат", "...", "Ленинский проспект"])
22  my_string = "Это просто строка"
23  my_list = [1, 2, 3, 4, 5]
24
25  print_object_length(my_book)
26  print_object_length(my_city)
27  print_object_length(my_string)
28  print_object_length(my_list)
```

Университет Сириус Колледж

Утиная типизация и интерфейсы

- Не требует наследования от общего базового класса.
- Поддерживает структурную типизацию — поведение определяет тип.
- Позже формализовано в typing.Protocol (PEP 544).

Сравнение:

Подход	Требует наследования	Проверка
Номинальная типизация	Да	По имени класса
Утиная типизация	Нет	По набору методов

Университет Сириус Колледж

Абстрактные классы

Абстрактный класс имеет некоторые особенности, а именно:

- Абстрактный класс не содержит всех реализаций методов, необходимых для полной работы, это означает, что он содержит один или несколько абстрактных методов. Абстрактный метод - это только объявление метода, без его подробной реализации.
- Абстрактный класс предоставляет интерфейс для подклассов, чтобы избежать дублирования кода. Нет смысла создавать экземпляр абстрактного класса.
- Производный подкласс должен реализовать абстрактные методы для создания конкретного класса, который соответствует интерфейсу, определенному абстрактным классом. Следовательно, экземпляр не может быть создан, пока не будут переопределены все его абстрактные методы.

Университет Сириус
Колледж

Абстрактный класс определяет общий интерфейс для набора подклассов. Он предоставляет общие атрибуты и методы для всех подклассов, чтобы уменьшить дублирование кода. Он также заставляет подклассы реализовывать абстрактные методы, чтобы избежать каких-либо несоответствий.

Университет Сириус Колледж

Определение абстрактного класса

```
1  class Animal:
2      def move(self): pass
3
4  a = Animal()
```

Университет Сириус Колледж

Определение абстрактного класса

```
1  class Animal:
2      def move(self): pass
3
4  a = Animal()
```

Класс Animal не является в полной мере абстрактным, так как может быть инициализирован.

Университет Сириус Колледж

Определение абстрактного класса

```
1  from abc import ABC, abstractmethod
2
3  class Animal(ABC):
4      @abstractmethod
5      def move(self): pass
6
7  a = Animal()
8  # TypeError: Can't instantiate abstract class Animal with abstract metho
```

Университет Сириус Колледж

Реализация абстрактного метода

```
1  from abc import ABC, abstractmethod
2
3  class Animal(ABC):
4      @abstractmethod
5      def move(self): pass
6
7  class Cat(Animal):
8      def move(self):
9          print('Кошка грациозно крадется')
10
11 class Fish(Animal):
12     def move(self):
13         print('Рыба плывет')
14
15 cat = Cat()
```

Университет Сириус Колледж

Реализация абстрактного метода

```python
from abc import ABC, abstractmethod

class Animal(ABC):
    @abstractmethod
    def move(self): pass

class Cat(Animal):
    def move(self):
        print('Кошка грациозно крадется')

class Fish(Animal):
    def move(self):
        print('Рыба плывет')

cat = Cat()
```

Университет Сириус Колледж

Реализация абстрактного метода

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

class Cat(Animal):
    def move(self):
        print('Кошка грациозно крадется')

class Fish(Animal):
    def move(self):
        print('Рыба плывет')

cat = Cat()
cat.move() # Работает
```

Университет Сириус Колледж

Реализация абстрактного метода

```
10
11 class Fish(Animal):
12     def move(self):
13         print('Рыба плывет')
14
15 cat = Cat()
16 cat.move() # Работает
17
18 fish = Fish()
19 fish.move() # Работает
20
21 # А если не реализовать, будет ошибка:
22 class Bird(Animal): pass
23
24 bird = Bird() # TypeError: Can't instantiate...
```

Университет Сириус Колледж

Реализация абстрактного метода

```
10
11 class Fish(Animal):
12     def move(self):
13         print('Рыба плывет')
14
15 cat = Cat()
16 cat.move() # Работает
17
18 fish = Fish()
19 fish.move() # Работает
20
21 # А если не реализовать, будет ошибка:
22 class Bird(Animal): pass
23
24 bird = Bird() # TypeError: Can't instantiate...
```

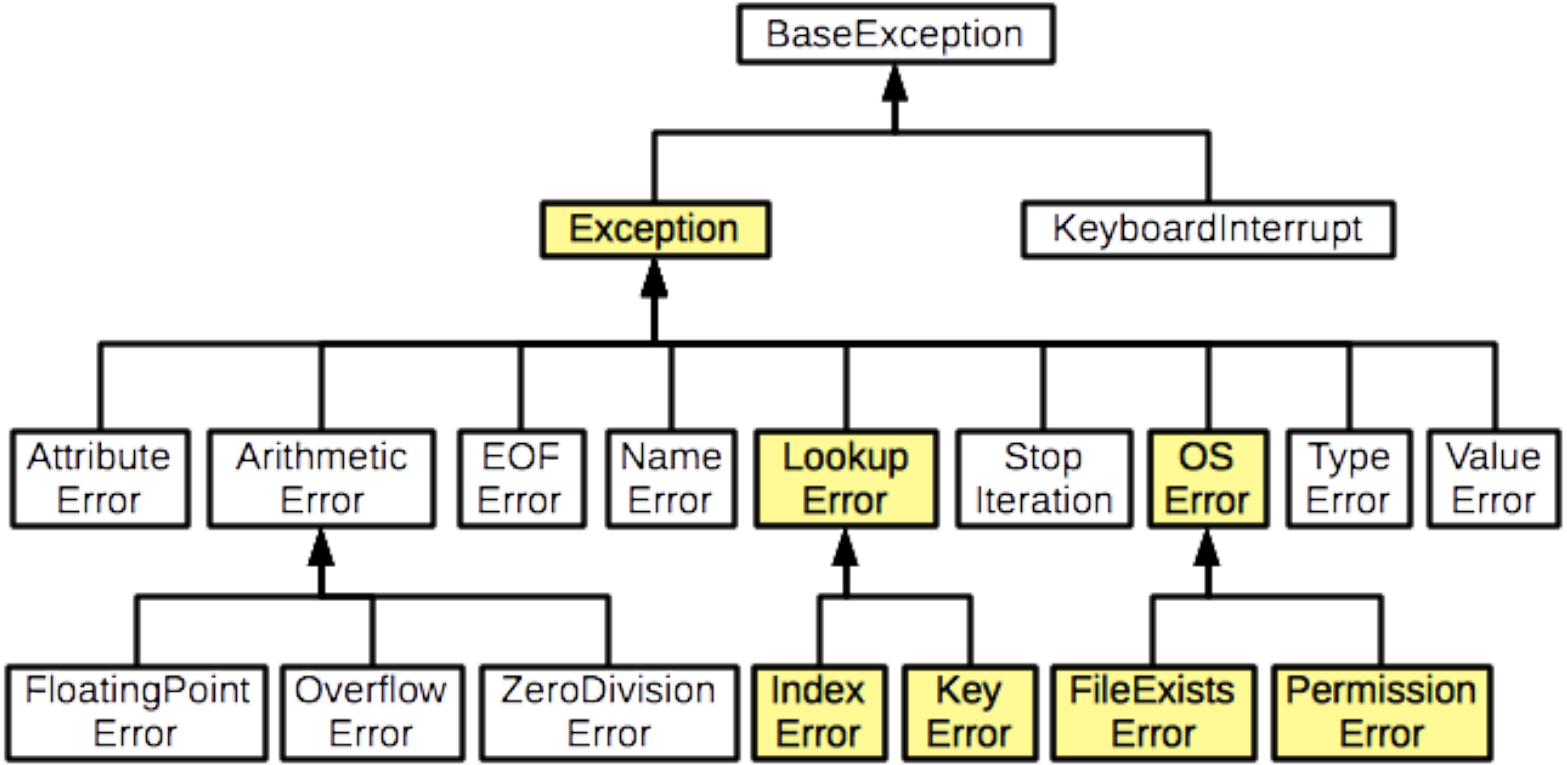
Университет Сириус Колледж

Обработка исключений

```python
try:
    num = int(input("Введите число: "))
    result = 10 / num
    print(f"Результат: {result}")
except ValueError:
    print("Ошибка: Введено не число!")
except ZeroDivisionError:
    print("Ошибка: Деление на ноль!")
finally:
    print("Блок finally выполнен.")
```

Университет Сириус Колледж

Пользовательские исключения



Университет Сириус Колледж

Пользовательские исключения

```python
class BalanceTooLowError(Exception): pass

def withdraw(balance, amount):
    if amount > balance:
        raise BalanceTooLowError("Недостаточно средств на счете")
    return balance - amount

my_balance = 100
try:
    my_balance = withdraw(my_balance, 150)
except BalanceTooLowError as e:
    print(f"Ошибка операции: {e}")
```

Университет Сириус Колледж

Пользовательские исключения

```python
class BalanceTooLowError(Exception): pass

def withdraw(balance, amount):
    if amount > balance:
        raise BalanceTooLowError("Недостаточно средств на счете")
    return balance - amount

my_balance = 100
try:
    my_balance = withdraw(my_balance, 150)
except BalanceTooLowError as e:
    print(f"Ошибка операции: {e}")
```

Университет Сириус Колледж

Пользовательские исключения

```python
class BalanceTooLowError(Exception): pass

def withdraw(balance, amount):
    if amount > balance:
        raise BalanceTooLowError("Недостаточно средств на счете")
    return balance - amount

my_balance = 100
try:
    my_balance = withdraw(my_balance, 150)
except BalanceTooLowError as e:
    print(f"Ошибка операции: {e}")
```

Университет Сириус Колледж

Пользовательские исключения

```
1  class BalanceTooLowError(Exception): pass
2
3  def withdraw(balance, amount):
4      if amount > balance:
5          raise BalanceTooLowError("Недостаточно средств на счете")
6      return balance - amount
7
8  my_balance = 100
9  try:
10     my_balance = withdraw(my_balance, 150)
11 except BalanceTooLowError as e:
12     print(f"Ошибка операции: {e}")
```

Университет Сириус Колледж

Пользовательские исключения

```python
class BalanceTooLowError(Exception): pass

def withdraw(balance, amount):
    if amount > balance:
        raise BalanceTooLowError("Недостаточно средств на счете")
    return balance - amount

my_balance = 100
try:
    my_balance = withdraw(my_balance, 150)
except BalanceTooLowError as e:
    print(f"Ошибка операции: {e}")
```

Университет Сириус Колледж

Пользовательские исключения

```python
class BalanceTooLowError(Exception): pass

def withdraw(balance, amount):
    if amount > balance:
        raise BalanceTooLowError("Недостаточно средств на счете")
    return balance - amount

my_balance = 100
try:
    my_balance = withdraw(my_balance, 150)
except BalanceTooLowError as e:
    print(f"Ошибка операции: {e}")
```