Университет Сириус Колледж

Объектно-ориентированное программирование
Лекция 5: `dataclass`, `mixin` и декорирование классов

`dataclass`: упрощение создания классов данных. `mixin`-классы для повторного использования поведения. Декорирование классов: применение декораторов к классам.

Университет Сириус Колледж

Dataclasses

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


p = Point(0, 0)
print(p)
# <__main__.Point object at 0x7f45e98f4590>
```

Dataclasses

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

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


p = Point(0, 0)
print(p)
# point(0, 0)
```

Университет Сириус Колледж

Dataclasses

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

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


    def __eq__(self, other):
        if not isinstance(other, Point):
            return NotImplemented()
        return self.x == other.x and self.y == other.y
```

Университет Сириус Колледж

Dataclasses

Dataclasses призваны автоматизировать генерацию кода классов, которые используются для хранения данных. Не смотря на то, что они используют другие механизмы работы, их можно сравнить с "изменяемыми именованными кортежами со значениями по умолчанию".

Университет Сириус Колледж

Dataclasses

Dataclasses призваны автоматизировать генерацию кода классов, которые используются для хранения данных. Не смотря на то, что они используют другие механизмы работы, их можно сравнить с "изменяемыми именованными кортежами со значениями по умолчанию".

```python
from dataclasses import dataclass

@dataclass
class Point:
    x: float
    y: float

p1 = Point(1.0, 2.0)
p2 = Point(1.0, 2.0)
print(p1) # Point(x=1.0, y=2.0)
print(p1 == p2) # True
```

Университет Сириус Колледж

Dataclasses: Добавляем возможность сравнения

По умолчанию `dataclass` не знает, как сравнивать объекты. Мы можем включить эту возможность с помощью параметра `order=True`.

> *`order=True` автоматически генерирует методы `__lt__`, `__le__`, `__gt__`, `__ge__`. Сравнение происходит поэлементно, как у кортежей.*

```python
from dataclasses import dataclass

@dataclass(order=True)
class Student:
    gpa: float  # Сортировка будет в первую очередь по среднему баллу
    name: str

s1 = Student(4.5, "Анна")
s2 = Student(4.9, "Иван")
```

Университет Сириус Колледж

Dataclasses: Добавляем возможность сравнения

По умолчанию `dataclass` не знает, как сравнивать объекты. Мы можем включить эту возможность с помощью параметра `order=True`.

> *`order=True` автоматически генерирует методы `__lt__`, `__le__`, `__gt__`, `__ge__`. Сравнение происходит поэлементно, как у кортежей.*

```
5       gpa: float  # Сортировка будет в первую очередь по среднему баллу
6       name: str
7
8   s1 = Student(4.5, "Анна")
9   s2 = Student(4.9, "Иван")
10  s3 = Student(4.5, "Борис")
11
12  print(s1 < s2)  # True
13
```

Университет Сириус Колледж

Dataclasses: Добавляем возможность сравнения

По умолчанию `dataclass` не знает, как сравнивать объекты. Мы можем включить эту возможность с помощью параметра `order=True`.

> *`order=True` автоматически генерирует методы `__lt__`, `__le__`, `__gt__`, `__ge__`. Сравнение происходит поэлементно, как у кортежей.*

```
8  s1 = Student(4.5, "Анна")
9  s2 = Student(4.9, "Иван")
10 s3 = Student(4.5, "Борис")
11
12 print(s1 < s2)  # True
13
14 # Если gpa одинаковый, сравнение идет по второму полю (name)
15 print(s3 < s1)  # False
16
```

Университет Сириус Колледж

Dataclasses: Добавляем возможность сравнения

По умолчанию `dataclass` не знает, как сравнивать объекты. Мы можем включить эту возможность с помощью параметра `order=True`.

> *`order=True` автоматически генерирует методы `__lt__`, `__le__`, `__gt__`, `__ge__`. Сравнение происходит поэлементно, как у кортежей.*

```
11
12 print(s1 < s2)  # True
13
14 # Если gpa одинаковый, сравнение идет по второму полю (name)
15 print(s3 < s1)  # False
16
17 students = [s1, s2, s3]
18 print(sorted(students))
19 # [Student(gpa=4.5, name='Анна'), Student(gpa=4.5, name='Борис'), Student(gpa
```

Университет Сириус Колледж

Dataclasses: Добавляем возможность сравнения

По умолчанию `dataclass` не знает, как сравнивать объекты. Мы можем включить эту возможность с помощью параметра `order=True`.

> *`order=True` автоматически генерирует методы `__lt__`, `__le__`, `__gt__`, `__ge__`. Сравнение происходит поэлементно, как у кортежей.*

```
11
12 print(s1 < s2)  # True
13
14 # Если gpa одинаковый, сравнение идет по второму полю (name)
15 print(s3 < s1)  # False
16
17 students = [s1, s2, s3]
18 print(sorted(students))
19 # [Student(gpa=4.5, name='Анна'), Student(gpa=4.5, name='Борис'), Student(gpa
```

Университет Сириус Колледж

Dataclasses: Создание неизменяемых объектов

Иногда важно гарантировать, что данные в объекте не будут случайно изменены после его создания. Параметр `frozen=True` делает объект **иммутабельным (неизменяемым)**.

> *`frozen=True` запрещает присваивание значений атрибутам после инициализации. При попытке изменения будет вызвана ошибка `FrozenInstanceError`.*

Университет Сириус Колледж

Dataclasses: Создание неизменяемых объектов

> *Это делает объекты-конфигурации или ключи словарей более безопасными и предсказуемыми.*

```
1  from dataclasses import dataclass, FrozenInstanceError
2
3  @dataclass(frozen=True)
4  class Config:
5      host: str
6      port: int
7      user: str
8
9  db_config = Config("localhost", 5432, "admin")
10
11 print(db_config.host)  # 'localhost'
12
```

Университет Сириус Колледж

Dataclasses: Создание неизменяемых объектов

Это делает объекты-конфигурации или ключи словарей более безопасными и предсказуемыми.

```python
from dataclasses import dataclass, FrozenInstanceError

@dataclass(frozen=True)
class Config:
    host: str
    port: int
    user: str

db_config = Config("localhost", 5432, "admin")

print(db_config.host)  # 'localhost'

```

Университет Сириус Колледж

Dataclasses: Создание неизменяемых объектов

Это делает объекты-конфигурации или ключи словарей более безопасными и предсказуемыми.

```python
from dataclasses import dataclass, FrozenInstanceError

@dataclass(frozen=True)
class Config:
    host: str
    port: int
    user: str

db_config = Config("localhost", 5432, "admin")

print(db_config.host)  # 'localhost'

```

Университет Сириус Колледж

Dataclasses: Создание неизменяемых объектов

Это делает объекты-конфигурации или ключи словарей более безопасными и предсказуемыми.

```python
from dataclasses import dataclass, FrozenInstanceError

@dataclass(frozen=True)
class Config:
    host: str
    port: int
    user: str

db_config = Config("localhost", 5432, "admin")

print(db_config.host)  # 'localhost'

```

Университет Сириус Колледж

Dataclasses: Создание неизменяемых объектов

Это делает объекты-конфигурации или ключи словарей более безопасными и предсказуемыми.

```
@dataclass(frozen=True)
class Config:
    host: str
    port: int
    user: str

db_config = Config("localhost", 5432, "admin")

print(db_config.host)  # 'localhost'

try:
    # Попытка изменить атрибут "замороженного" объекта
    db_config.host = "127.0.0.1"
```

Университет Сириус Колледж

Dataclasses: Создание неизменяемых объектов

Это делает объекты-конфигурации или ключи словарей более безопасными и предсказуемыми.

```
    host: str
    port: int
    user: str

db_config = Config("localhost", 5432, "admin")

print(db_config.host)  # 'localhost'

try:
    # Попытка изменить атрибут "замороженного" объекта
    db_config.host = "127.0.0.1"
except FrozenInstanceError as e:
    print(f"Ошибка: {e}")
```

Университет Сириус Колледж

Dataclasses: Создание неизменяемых объектов

> *Это делает объекты-конфигурации или ключи словарей более безопасными и предсказуемыми.*

```
8
9  db_config = Config("localhost", 5432, "admin")
10
11 print(db_config.host)  # 'localhost'
12
13 try:
14     # Попытка изменить атрибут "замороженного" объекта
15     db_config.host = "127.0.0.1"
16 except FrozenInstanceError as e:
17     print(f"Ошибка: {e}")
18
19 # Ошибка: cannot assign to field 'host'
```

Университет Сириус Колледж

Dataclasses: Создание неизменяемых объектов

> *Это делает объекты-конфигурации или ключи словарей более безопасными и предсказуемыми.*

```
1  from dataclasses import dataclass, FrozenInstanceError
2
3  @dataclass(frozen=True)
4  class Config:
5      host: str
6      port: int
7      user: str
8
9  db_config = Config("localhost", 5432, "admin")
10
11 print(db_config.host)  # 'localhost'
12
```

Университет Сириус Колледж

Dataclasses: Настройка полей с помощью `field`

Функция `field` из модуля `dataclasses` позволяет тонко настраивать каждое поле класса.

Университет Сириус Колледж

Dataclasses: Настройка полей с помощью `field`

Функция `field` из модуля `dataclasses` позволяет тонко настраивать каждое поле класса.

Основные параметры `field`:

Университет Сириус Колледж

Dataclasses: Настройка полей с помощью `field`

Функция `field` из модуля `dataclasses` позволяет тонко настраивать каждое поле класса.

Основные параметры `field`:

- `default`: Задает простое значение по умолчанию.
- `default_factory`: Задает **функцию**, которая будет вызвана для создания значения по умолчанию (например, `list` для пустого списка).
- `repr=False`: Исключает поле из метода `__repr__`.
- `init=False`: Исключает поле из метода `__init__`. Полезно для вычисляемых полей.

Университет Сириус Колледж

Dataclasses: Настройка полей с помощью `field`

```python
from dataclasses import dataclass, field
import uuid

@dataclass
class User:
    username: str
    # default_factory нужен для изменяемых типов, чтобы избежать
    # использования одного и того же списка для всех объектов
    friends: list[str] = field(default_factory=list)

    # Исключаем пароль из вывода для безопасности
    password_hash: str = field(repr=False)

    # Это поле не будет в __init__, оно вычисляется позже
    user_id: str = field(init=False)
```

Университет Сириус Колледж

Dataclasses: Настройка полей с помощью `field`

```python
from dataclasses import dataclass, field
import uuid

@dataclass
class User:
    username: str
    # default_factory нужен для изменяемых типов, чтобы избежать
    # использования одного и того же списка для всех объектов
    friends: list[str] = field(default_factory=list)

    # Исключаем пароль из вывода для безопасности
    password_hash: str = field(repr=False)

    # Это поле не будет в __init__, оно вычисляется позже
    user_id: str = field(init=False)
```

Университет Сириус Колледж

Dataclasses: Настройка полей с помощью `field`

```python
from dataclasses import dataclass, field
import uuid

@dataclass
class User:
    username: str
    # default_factory нужен для изменяемых типов, чтобы избежать
    # использования одного и того же списка для всех объектов
    friends: list[str] = field(default_factory=list)

    # Исключаем пароль из вывода для безопасности
    password_hash: str = field(repr=False)

    # Это поле не будет в __init__, оно вычисляется позже
    user_id: str = field(init=False)
```

Университет Сириус Колледж

Dataclasses: Настройка полей с помощью `field`

```python
class User:
    username: str
    # default_factory нужен для изменяемых типов, чтобы избежать
    # использования одного и того же списка для всех объектов
    friends: list[str] = field(default_factory=list)

    # Исключаем пароль из вывода для безопасности
    password_hash: str = field(repr=False)

    # Это поле не будет в __init__, оно вычисляется позже
    user_id: str = field(init=False)

    def __post_init__(self):
        # Специальный метод, вызывается после __init__
        self.user_id = str(uuid.uuid4())
```

Университет Сириус Колледж

Dataclasses: Настройка полей с помощью `field`

```python
    # использования одного и того же списка для всех объектов
    friends: list[str] = field(default_factory=list)

    # Исключаем пароль из вывода для безопасности
    password_hash: str = field(repr=False)

    # Это поле не будет в __init__, оно вычисляется позже
    user_id: str = field(init=False)

    def __post_init__(self):
        # Специальный метод, вызывается после __init__
        self.user_id = str(uuid.uuid4())

user = User("Alice", password_hash="a1b2c3d4")
print(user)
```

Университет Сириус Колледж

Dataclasses: Настройка полей с помощью `field`

```
    # Исключаем пароль из вывода для безопасности
    password_hash: str = field(repr=False)

    # Это поле не будет в __init__, оно вычисляется позже
    user_id: str = field(init=False)

    def __post_init__(self):
        # Специальный метод, вызывается после __init__
        self.user_id = str(uuid.uuid4())

user = User("Alice", password_hash="a1b2c3d4")
print(user)
# User(username='Alice', friends=[])
# (password_hash и user_id не попали в repr)
```

Университет Сириус Колледж

Dataclasses: Настройка полей с помощью `field`

```
    # Исключаем пароль из вывода для безопасности
    password_hash: str = field(repr=False)

    # Это поле не будет в __init__, оно вычисляется позже
    user_id: str = field(init=False)

    def __post_init__(self):
        # Специальный метод, вызывается после __init__
        self.user_id = str(uuid.uuid4())

user = User("Alice", password_hash="a1b2c3d4")
print(user)
# User(username='Alice', friends=[])
# (password_hash и user_id не попали в repr)
```

Университет Сириус Колледж

Dataclasses

```
@dataclasses.dataclass(*, init=True, repr=True, eq=True, order=False,
unsafe_hash=False, frozen=False, match_args=True, kw_only=False,
slots=False, weakref_slot=False)
```

Dataclasses

```
@dataclasses.dataclass(*, init=True, repr=True, eq=True, order=False,
unsafe_hash=False, frozen=False, match_args=True, kw_only=False,
slots=False, weakref_slot=False)
```

- `unsafe_hash` — создаёт `__hash__`, даже если объект изменяемый (использовать с осторожностью).

Университет Сириус Колледж

Dataclasses

```
@dataclasses.dataclass(*, init=True, repr=True, eq=True, order=False,
unsafe_hash=False, frozen=False, match_args=True, kw_only=False,
slots=False, weakref_slot=False)
```

- `unsafe_hash` — создаёт `__hash__`, даже если объект изменяемый (использовать с осторожностью).
- `frozen` — делает экземпляры неизменяемыми (аналог immutable).

Университет Сириус Колледж

Dataclasses

```
@dataclasses.dataclass(*, init=True, repr=True, eq=True, order=False,
unsafe_hash=False, frozen=False, match_args=True, kw_only=False,
slots=False, weakref_slot=False)
```

- `unsafe_hash` — создаёт `__hash__`, даже если объект изменяемый (использовать с осторожностью).
- `frozen` — делает экземпляры неизменяемыми (аналог immutable).
- `match_args` — позволяет использовать позиционное сопоставление в `match`.

Университет Сириус Колледж

Dataclasses

```
@dataclasses.dataclass(*, init=True, repr=True, eq=True, order=False,
unsafe_hash=False, frozen=False, match_args=True, kw_only=False,
slots=False, weakref_slot=False)
```

- `unsafe_hash` — создаёт `__hash__`, даже если объект изменяемый (использовать с осторожностью).
- `frozen` — делает экземпляры неизменяемыми (аналог immutable).
- `match_args` — позволяет использовать позиционное сопоставление в `match`.
- `kw_only` — делает поля только именованными аргументами в конструкторе.

Университет Сириус Колледж

Dataclasses

```
@dataclasses.dataclass(*, init=True, repr=True, eq=True, order=False,
unsafe_hash=False, frozen=False, match_args=True, kw_only=False,
slots=False, weakref_slot=False)
```

- `unsafe_hash` — создаёт `__hash__`, даже если объект изменяемый (использовать с осторожностью).
- `frozen` — делает экземпляры неизменяемыми (аналог immutable).
- `match_args` — позволяет использовать позиционное сопоставление в `match`.
- `kw_only` — делает поля только именованными аргументами в конструкторе.
- `slots` — создаёт класс с `__slots__`, экономит память и запрещает новые атрибуты.

Университет Сириус Колледж

Dataclasses

```
@dataclasses.dataclass(*, init=True, repr=True, eq=True, order=False,
unsafe_hash=False, frozen=False, match_args=True, kw_only=False,
slots=False, weakref_slot=False)
```

- `unsafe_hash` — создаёт `__hash__`, даже если объект изменяемый (использовать с осторожностью).
- `frozen` — делает экземпляры неизменяемыми (аналог immutable).
- `match_args` — позволяет использовать позиционное сопоставление в `match`.
- `kw_only` — делает поля только именованными аргументами в конструкторе.
- `slots` — создаёт класс с `__slots__`, экономит память и запрещает новые атрибуты.
- `weakref_slot` — добавляет слот для поддержки weak references.

Университет Сириус Колледж

Mixins

Mixin-классы (примеси) - это классы у которых нет данных, но есть методы. Mixin используются для добавления одних и тех же методов в разные классы.

- **Наследование** — это отношение "is-a" (Dog is an Animal).
- **Композиция** — это "has-a" (Car has an Engine).
- **Миксин** — это "provides-a" (Person provides a JsonExport capability).

Университет Сириус Колледж

Mixins

Mixin-классы (примеси) - это классы у которых нет данных, но есть методы. Mixin используются для добавления одних и тех же методов в разные классы.

- **Наследование** — это отношение "is-a" (Dog is an Animal).
- **Композиция** — это "has-a" (Car has an Engine).
- **Миксин** — это "provides-a" (Person provides a JsonExport capability).

> *Миксины не предназначены для создания экземпляров. Это просто "примесь" функциональности.*

Университет Сириус Колледж

```python
import json

class JsonMixin:
    def to_json(self):
        """Преобразуем атрибуты объекта в словарь"""
        return json.dumps(self.__dict__, indent=4, ensure_ascii=False)

class Person(JsonMixin):
    def __init__(self, name, age):
        self.name = name
        self.age = age

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

Университет Сириус Колледж

```python
import json

class JsonMixin:
    def to_json(self):
        """Преобразуем атрибуты объекта в словарь"""
        return json.dumps(self.__dict__, indent=4, ensure_ascii=False)

class Person(JsonMixin):
    def __init__(self, name, age):
        self.name = name
        self.age = age

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

Университет Сириус Колледж

```python
import json

class JsonMixin:
    def to_json(self):
        """Преобразуем атрибуты объекта в словарь"""
        return json.dumps(self.__dict__, indent=4, ensure_ascii=False)

class Person(JsonMixin):
    def __init__(self, name, age):
        self.name = name
        self.age = age

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

Университет Сириус Колледж

```
class JsonMixin:
    def to_json(self):
        """Преобразуем атрибуты объекта в словарь"""
        return json.dumps(self.__dict__, indent=4, ensure_ascii=False)

class Person(JsonMixin):
    def __init__(self, name, age):
        self.name = name
        self.age = age

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

```

Университет Сириус Колледж

```python
import json

class JsonMixin:
    def to_json(self):
        """Преобразуем атрибуты объекта в словарь"""
        return json.dumps(self.__dict__, indent=4, ensure_ascii=False)

class Person(JsonMixin):
    def __init__(self, name, age):
        self.name = name
        self.age = age

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

Университет Сириус Колледж

```
class Person(JsonMixin):
    def __init__(self, name, age):
        self.name = name
        self.age = age

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

p = Person("Иван", 30)
b = Book("Война и мир", "Л.Н. Толстой")
print(p.to_json())
print(b.to_json())
```

Университет Сириус Колледж

```
class Person(JsonMixin):
    def __init__(self, name, age):
        self.name = name
        self.age = age

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

p = Person("Иван", 30)
b = Book("Война и мир", "Л.Н. Толстой")
print(p.to_json())
print(b.to_json())
```

Университет Сириус Колледж

Декорирование классов

```python
def add_str_dunder(class_):
    def wrapper(self):
        attrs = [f'{k}={v}' for k, v in self.__dict__.items()]
        return ", ".join(attrs)
    setattr(class_, "__str__", wrapper)
    return class_

@add_str_dunder
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

print(Person('Petya', 20))
# name=Petya, age=20
```

Университет Сириус Колледж

```python
PLUGINS = {}

def register_plugin(name):
    def decorator(cls):
        PLUGINS[name] = cls
        return cls
    return decorator

@register_plugin("video")
class VideoPlugin:
    def play(self): print("Playing video")

@register_plugin("audio")
class AudioPlugin:
    def play(self): print("Playing audio")
```

Университет Сириус Колледж

```python
PLUGINS = {}

def register_plugin(name):
    def decorator(cls):
        PLUGINS[name] = cls
        return cls
    return decorator

@register_plugin("video")
class VideoPlugin:
    def play(self): print("Playing video")

@register_plugin("audio")
class AudioPlugin:
    def play(self): print("Playing audio")
```

Университет Сириус Колледж

```python
PLUGINS = {}

def register_plugin(name):
    def decorator(cls):
        PLUGINS[name] = cls
        return cls
    return decorator

@register_plugin("video")
class VideoPlugin:
    def play(self): print("Playing video")

@register_plugin("audio")
class AudioPlugin:
    def play(self): print("Playing audio")
```

```python
def register_plugin(name):
    def decorator(cls):
        PLUGINS[name] = cls
        return cls
    return decorator

@register_plugin("video")
class VideoPlugin:
    def play(self): print("Playing video")

@register_plugin("audio")
class AudioPlugin:
    def play(self): print("Playing audio")
```

Университет Сириус Колледж

```
7       return decorator
8
9   @register_plugin("video")
10  class VideoPlugin:
11      def play(self): print("Playing video")
12
13  @register_plugin("audio")
14  class AudioPlugin:
15      def play(self): print("Playing audio")
16
17
18  plugin_name = "video"
19  plugin_class = PLUGINS[plugin_name]
20  plugin_instance = plugin_class()
21  plugin_instance.play()
```

Университет Сириус Колледж

```
    return decorator

@register_plugin("video")
class VideoPlugin:
    def play(self): print("Playing video")

@register_plugin("audio")
class AudioPlugin:
    def play(self): print("Playing audio")


plugin_name = "video"
plugin_class = PLUGINS[plugin_name]
plugin_instance = plugin_class()
plugin_instance.play()
```

Университет Сириус Колледж

```
    instances = {}
    def get_instance(*args, **kwargs):
        if cls not in instances:
            instances[cls] = cls(*args, **kwargs)
        return instances[cls]
    return get_instance

@singleton
class DatabaseConnection:
    def __init__(self):
        print("Создано новое подключение к БД")

conn1 = DatabaseConnection()
conn2 = DatabaseConnection()
print(conn1 is conn2) # True
```

Университет Сириус Колледж

```python
def singleton(cls):
    instances = {}
    def get_instance(*args, **kwargs):
        if cls not in instances:
            instances[cls] = cls(*args, **kwargs)
        return instances[cls]
    return get_instance

@singleton
class DatabaseConnection:
    def __init__(self):
        print("Создано новое подключение к БД")

conn1 = DatabaseConnection()
conn2 = DatabaseConnection()
```

Университет Сириус Колледж

```python
def singleton(cls):
    instances = {}
    def get_instance(*args, **kwargs):
        if cls not in instances:
            instances[cls] = cls(*args, **kwargs)
        return instances[cls]
    return get_instance

@singleton
class DatabaseConnection:
    def __init__(self):
        print("Создано новое подключение к БД")

conn1 = DatabaseConnection()
conn2 = DatabaseConnection()
```

Университет Сириус Колледж

```python
def singleton(cls):
    instances = {}
    def get_instance(*args, **kwargs):
        if cls not in instances:
            instances[cls] = cls(*args, **kwargs)
        return instances[cls]
    return get_instance

@singleton
class DatabaseConnection:
    def __init__(self):
        print("Создано новое подключение к БД")

conn1 = DatabaseConnection()
conn2 = DatabaseConnection()
```

Университет Сириус Колледж

```python
def singleton(cls):
    instances = {}
    def get_instance(*args, **kwargs):
        if cls not in instances:
            instances[cls] = cls(*args, **kwargs)
        return instances[cls]
    return get_instance

@singleton
class DatabaseConnection:
    def __init__(self):
        print("Создано новое подключение к БД")

conn1 = DatabaseConnection()
conn2 = DatabaseConnection()
```

Университет Сириус Колледж

```
    instances = {}
    def get_instance(*args, **kwargs):
        if cls not in instances:
            instances[cls] = cls(*args, **kwargs)
        return instances[cls]
    return get_instance

@singleton
class DatabaseConnection:
    def __init__(self):
        print("Создано новое подключение к БД")

conn1 = DatabaseConnection()
conn2 = DatabaseConnection()
print(conn1 is conn2) # True
```

Университет Сириус Колледж

```
    instances = {}
    def get_instance(*args, **kwargs):
        if cls not in instances:
            instances[cls] = cls(*args, **kwargs)
        return instances[cls]
    return get_instance

@singleton
class DatabaseConnection:
    def __init__(self):
        print("Создано новое подключение к БД")

conn1 = DatabaseConnection()
conn2 = DatabaseConnection()
print(conn1 is conn2) # True
```