Университет Сириус
Колледж

Объектно-ориентированное программирование
Лекция 2. Принципы проектирования чистого кода

Принципы SOLID (SRP, OCP, LSP, ISP, DIP). Принципы DRY (Don’t Repeat Yourself) и KISS (Keep It Simple, Stupid).

Университет Сириус Колледж

Zen of Python

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.

- Красивое лучше уродливого.
- Явное лучше неявного.
- Простое лучше сложного.
- Сложное лучше запутанного.

Университет Сириус Колледж

DRY

Университет Сириус Колледж

DRY

Don’t Repeat Yourself

Не повторяйся

Университет Сириус Колледж

DRY - Don’t Repeat Yourself
Не повторяйся

Принцип гласит, что каждая часть знания в системе должна иметь единственное, однозначное и авторитетное представление. Проще говоря: **избегайте дублирования кода**.

```
1  def process_user_data(data):
2      # Валидация email
3      email = data.get('email', '')
4      if '@' not in email or '.' not in email:
5          print("Ошибка: Некорректный email")
6          return
7      # ... какая-то обработка ...
8      print(f"Обработка данных для {email}")
9
10 def register_new_user(email, password):
11     # Повторная валидация email
12     if '@' not in email or '.' not in email:
```

Университет Сириус Колледж

DRY - Don’t Repeat Yourself
Не повторяйся

Принцип гласит, что каждая часть знания в системе должна иметь единственное, однозначное и авторитетное представление. Проще говоря: **избегайте дублирования кода**.

```
1  def process_user_data(data):
2      # Валидация email
3      email = data.get('email', '')
4      if '@' not in email or '.' not in email:
5          print("Ошибка: Некорректный email")
6          return
7      # ... какая-то обработка ...
8      print(f"Обработка данных для {email}")
9
10 def register_new_user(email, password):
11     # Повторная валидация email
12     if '@' not in email or '.' not in email:
```

Университет Сириус Колледж

DRY - Don’t Repeat Yourself
Не повторяйся

Принцип гласит, что каждая часть знания в системе должна иметь единственное, однозначное и авторитетное представление. Проще говоря: **избегайте дублирования кода**.

```
1  def process_user_data(data):
2      # Валидация email
3      email = data.get('email', '')
4      if '@' not in email or '.' not in email:
5          print("Ошибка: Некорректный email")
6          return
7      # ... какая-то обработка ...
8      print(f"Обработка данных для {email}")
9
10 def register_new_user(email, password):
11     # Повторная валидация email
12     if '@' not in email or '.' not in email:
```

Университет Сириус Колледж

DRY - Don’t Repeat Yourself
Не повторяйся

Принцип гласит, что каждая часть знания в системе должна иметь единственное, однозначное и авторитетное представление. Проще говоря: **избегайте дублирования кода**.

```
        print("Ошибка: Некорректный email")
        return
    # ... какая-то обработка ...
    print(f"Обработка данных для {email}")

def register_new_user(email, password):
    # Повторная валидация email
    if '@' not in email or '.' not in email:
        print("Ошибка: Некорректный email для регистрации")
        return
    # ... логика регистрации ...
    print(f"Регистрация пользователя с email: {email}")
```

Университет Сириус Колледж

DRY - Don’t Repeat Yourself
Не повторяйся

Принцип гласит, что каждая часть знания в системе должна иметь единственное, однозначное и авторитетное представление. Проще говоря: **избегайте дублирования кода**.

```
        print("Ошибка: Некорректный email")
        return
    # ... какая-то обработка ...
    print(f"Обработка данных для {email}")

def register_new_user(email, password):
    # Повторная валидация email
    if '@' not in email or '.' not in email:
        print("Ошибка: Некорректный email для регистрации")
        return
    # ... логика регистрации ...
    print(f"Регистрация пользователя с email: {email}")
```

Университет Сириус Колледж

DRY - Don’t Repeat Yourself
Не повторяйся

Принцип гласит, что каждая часть знания в системе должна иметь единственное, однозначное и авторитетное представление. Проще говоря: **избегайте дублирования кода**.

```python
def is_valid_email(email: str) -> bool:
    """Проверяет, является ли строка похожей на email."""
    return '@' in email and '.' in email


def process_user_data(data):
    email = data.get('email', '')
    if not is_valid_email(email):
        print("Ошибка: Некорректный email")
        return
    # ... какая-то обработка ...
    print(f"Обработка данных для {email}")
```

Университет Сириус Колледж

DRY - Don’t Repeat Yourself
Не повторяйся

Принцип гласит, что каждая часть знания в системе должна иметь единственное, однозначное и авторитетное представление. Проще говоря: **избегайте дублирования кода**.

```python
def is_valid_email(email: str) -> bool:
    """Проверяет, является ли строка похожей на email."""
    return '@' in email and '.' in email


def process_user_data(data):
    email = data.get('email', '')
    if not is_valid_email(email):
        print("Ошибка: Некорректный email")
        return
    # ... какая-то обработка ...
    print(f"Обработка данных для {email}")
```

Университет Сириус Колледж

DRY - Don’t Repeat Yourself
Не повторяйся

Принцип гласит, что каждая часть знания в системе должна иметь единственное, однозначное и авторитетное представление. Проще говоря: **избегайте дублирования кода**.

```python
def is_valid_email(email: str) -> bool:
    """Проверяет, является ли строка похожей на email."""
    return '@' in email and '.' in email


def process_user_data(data):
    email = data.get('email', '')
    if not is_valid_email(email):
        print("Ошибка: Некорректный email")
        return
    # ... какая-то обработка ...
    print(f"Обработка данных для {email}")
```

Университет Сириус Колледж

DRY - Don’t Repeat Yourself
Не повторяйся

Принцип гласит, что каждая часть знания в системе должна иметь единственное, однозначное и авторитетное представление. Проще говоря: **избегайте дублирования кода**.

```python
def is_valid_email(email: str) -> bool:
    """Проверяет, является ли строка похожей на email."""
    return '@' in email and '.' in email


def process_user_data(data):
    email = data.get('email', '')
    if not is_valid_email(email):
        print("Ошибка: Некорректный email")
        return
    # ... какая-то обработка ...
    print(f"Обработка данных для {email}")
```

Университет Сириус Колледж

DRY - Don’t Repeat Yourself
Не повторяйся

Принцип гласит, что каждая часть знания в системе должна иметь единственное, однозначное и авторитетное представление. Проще говоря: **избегайте дублирования кода**.

```python
def is_valid_email(email: str) -> bool:
    """Проверяет, является ли строка похожей на email."""
    return '@' in email and '.' in email


def process_user_data(data):
    email = data.get('email', '')
    if not is_valid_email(email):
        print("Ошибка: Некорректный email")
        return
    # ... какая-то обработка ...
    print(f"Обработка данных для {email}")
```

Университет Сириус Колледж

KISS

Университет Сириус Колледж

KISS

Keep it stupid simple

Пусть оно будет простым до безобразия

Университет Сириус Колледж

KISS

Keep it stupid simple

Пусть оно будет простым до безобразия

(Keep it simple, stupid)

Университет Сириус Колледж

KISS - Keep It Simple, Stupid
Пусть оно будет простым

Принцип призывает выбирать самые простые решения, которые работают. Не нужно усложнять код без необходимости, даже если сложное решение кажется "умнее" или "гибче". Простота облегчает чтение, понимание и поддержку кода.

```python
def get_day_of_week(day_number: int) -> str:
    """Возвращает название дня недели по его номеру."""
    if day_number == 1:
        return "Понедельник"
    elif day_number == 2:
        return "Вторник"
    elif day_number == 3:
        return "Среда"
    elif day_number == 4:
        return "Четверг"
    elif day_number == 5:
```

Университет Сириус Колледж

KISS - Keep It Simple, Stupid
Пусть оно будет простым

Принцип призывает выбирать самые простые решения, которые работают. Не нужно усложнять код без необходимости, даже если сложное решение кажется "умнее" или "гибче". Простота облегчает чтение, понимание и поддержку кода.

```python
def get_day_of_week(day_number: int) -> str:
    """Возвращает название дня недели по его номеру."""
    if day_number == 1:
        return "Понедельник"
    elif day_number == 2:
        return "Вторник"
    elif day_number == 3:
        return "Среда"
    elif day_number == 4:
        return "Четверг"
    elif day_number == 5:
```

Университет Сириус Колледж

KISS - Keep It Simple, Stupid
Пусть оно будет простым

Принцип призывает выбирать самые простые решения, которые работают. Не нужно усложнять код без необходимости, даже если сложное решение кажется "умнее" или "гибче". Простота облегчает чтение, понимание и поддержку кода.

```python
def get_day_of_week(day_number: int) -> str:
    """Возвращает название дня недели по его номеру."""
    if day_number == 1:
        return "Понедельник"
    elif day_number == 2:
        return "Вторник"
    elif day_number == 3:
        return "Среда"
    elif day_number == 4:
        return "Четверг"
    elif day_number == 5:
```

Университет Сириус Колледж

KISS - Keep It Simple, Stupid
Пусть оно будет простым

Принцип призывает выбирать самые простые решения, которые работают. Не нужно усложнять код без необходимости, даже если сложное решение кажется "умнее" или "гибче". Простота облегчает чтение, понимание и поддержку кода.

```
10          return "Четверг"
11      elif day_number == 5:
12          return "Пятница"
13      elif day_number == 6:
14          return "Суббота"
15      elif day_number == 7:
16          return "Воскресенье"
17      else:
18          class InvalidDayError(Exception):
19              pass
20          raise InvalidDayError("Неверный номер дня")
```

Университет Сириус Колледж

KISS - Keep It Simple, Stupid
Пусть оно будет простым

Принцип призывает выбирать самые простые решения, которые работают. Не нужно усложнять код без необходимости, даже если сложное решение кажется "умнее" или "гибче". Простота облегчает чтение, понимание и поддержку кода.

```python
def get_day_of_week(day_number: int) -> str:
    """Возвращает название дня недели по его номеру."""
    days = {
        1: "Понедельник",
        2: "Вторник",
        3: "Среда",
        4: "Четверг",
        5: "Пятница",
        6: "Суббота",
        7: "Воскресенье"
    }
```

Университет Сириус Колледж

KISS - Keep It Simple, Stupid
Пусть оно будет простым

Принцип призывает выбирать самые простые решения, которые работают. Не нужно усложнять код без необходимости, даже если сложное решение кажется "умнее" или "гибче". Простота облегчает чтение, понимание и поддержку кода.

```
def get_day_of_week(day_number: int) -> str:
    """Возвращает название дня недели по его номеру."""
    days = {
        1: "Понедельник",
        2: "Вторник",
        3: "Среда",
        4: "Четверг",
        5: "Пятница",
        6: "Суббота",
        7: "Воскресенье"
    }
```

Университет Сириус Колледж

KISS - Keep It Simple, Stupid
Пусть оно будет простым

Принцип призывает выбирать самые простые решения, которые работают. Не нужно усложнять код без необходимости, даже если сложное решение кажется "умнее" или "гибче". Простота облегчает чтение, понимание и поддержку кода.

```python
def get_day_of_week(day_number: int) -> str:
    """Возвращает название дня недели по его номеру."""
    days = {
        1: "Понедельник",
        2: "Вторник",
        3: "Среда",
        4: "Четверг",
        5: "Пятница",
        6: "Суббота",
        7: "Воскресенье"
    }
```

Университет Сириус Колледж

SOLID

Университет Сириус Колледж

SOLID

Single responsibility principle (принцип единственной ответственности)

Университет Сириус Колледж

SOLID

Single responsibility principle (принцип единственной ответственности)

Open-closed principle (принцип открытости/закрытости)

Университет Сириус Колледж

SOLID

Single responsibility principle (принцип единственной ответственности)

Open-closed principle (принцип открытости/закрытости)

Liskov substitution principle (принцип подстановки Лисков)

Университет Сириус Колледж

SOLID

Single responsibility principle (принцип единственной ответственности)

Open-closed principle (принцип открытости/закрытости)

Liskov substitution principle (принцип подстановки Лисков)

Interface segregation principle (принцип разделения интерфейса)

Университет Сириус Колледж

SOLID

Single responsibility principle (принцип единственной ответственности)

Open-closed principle (принцип открытости/закрытости)

Liskov substitution principle (принцип подстановки Лисков)

Interface segregation principle (принцип разделения интерфейса)

Dependency inversion principle (принцип инверсии зависимостей)

Университет Сириус Колледж

S - Single responsibility principle
Принцип единственной ответственности

Каждый блок вашего кода должен выполнять одну задачу

```
class TicketAndCardChecker:
    def check_ticket(self, ticket: str):
        if ticket.isdigit() and len(ticket) == 10:
            ...
        return False

    def check_card (self, card: str):
        card = card. replace(' ', '')
        if card.isdigit() and len(card) <= 17:
            ...
        return False
```

Университет Сириус Колледж

S - Single responsibility principle
Принцип единственной ответственности

Каждый блок вашего кода должен выполнять одну задачу

```python
class TicketChecker:
    def check(self, ticket: str):
        if ticket. isdigit() and len(ticket) == 10:
            ...
        return False

class CardChecker:
    def check(self, card: str):
        card = card. replace(' ', '')
        if card.isdigit() and len(card) <= 17:
            ...
        return False
```

Университет Сириус Колледж

O - Open-closed principle
Принцип открытости/закрытости

Ваши модули или библиотеки должны быть открыты для расширения, но закрыты для модификации.

```python
class DiscountCalculator:
    """Класс для расчета скидки в зависимости от типа клиента."""
    def get_discount(self, customer_type: str, price: float) -> float:
        if customer_type == 'standard':
            return price * 0.05  # 5% скидка
        elif customer_type == 'premium':
            return price * 0.1   # 10% скидка

        return 0.0
```

Университет Сириус Колледж

O - Open-closed principle
Принцип открытости/закрытости

Ваши модули или библиотеки должны быть открыты для расширения, но закрыты для модификации.

```python
class DiscountCalculator:
    """Класс для расчета скидки в зависимости от типа клиента."""
    def get_discount(self, customer_type: str, price: float) -> float:
        if customer_type == 'standard':
            return price * 0.05  # 5% скидка
        elif customer_type == 'premium':
            return price * 0.1   # 10% скидка

        return 0.0
```

Университет Сириус Колледж

O - Open-closed principle
Принцип открытости/закрытости

Ваши модули или библиотеки должны быть открыты для расширения, но закрыты для модификации.

```python
from abc import ABC, abstractmethod

class DiscountStrategy(ABC):
    @abstractmethod
    def apply_discount(self, price: float) -> float:
        pass

class StandardDiscount(DiscountStrategy):
    def apply_discount(self, price: float) -> float:
        return price * 0.05

class PremiumDiscount(DiscountStrategy):
```

Университет Сириус Колледж

O - Open-closed principle
Принцип открытости/закрытости

Ваши модули или библиотеки должны быть открыты для расширения, но закрыты для модификации.

```python
from abc import ABC, abstractmethod

class DiscountStrategy(ABC):
    @abstractmethod
    def apply_discount(self, price: float) -> float:
        pass

class StandardDiscount(DiscountStrategy):
    def apply_discount(self, price: float) -> float:
        return price * 0.05

class PremiumDiscount(DiscountStrategy):
```

Университет Сириус Колледж

O - Open-closed principle
Принцип открытости/закрытости

Ваши модули или библиотеки должны быть открыты для расширения, но закрыты для модификации.

```python
from abc import ABC, abstractmethod

class DiscountStrategy(ABC):
    @abstractmethod
    def apply_discount(self, price: float) -> float:
        pass

class StandardDiscount(DiscountStrategy):
    def apply_discount(self, price: float) -> float:
        return price * 0.05

class PremiumDiscount(DiscountStrategy):
```

Университет Сириус Колледж

O - Open-closed principle
Принцип открытости/закрытости

Ваши модули или библиотеки должны быть открыты для расширения, но закрыты для модификации.

```python
from abc import ABC, abstractmethod

class DiscountStrategy(ABC):
    @abstractmethod
    def apply_discount(self, price: float) -> float:
        pass

class StandardDiscount(DiscountStrategy):
    def apply_discount(self, price: float) -> float:
        return price * 0.05

class PremiumDiscount(DiscountStrategy):
```

Университет Сириус Колледж

O - Open-closed principle
Принцип открытости/закрытости

Ваши модули или библиотеки должны быть открыты для расширения, но закрыты для модификации.

```python
from abc import ABC, abstractmethod

class DiscountStrategy(ABC):
    @abstractmethod
    def apply_discount(self, price: float) -> float:
        pass

class StandardDiscount(DiscountStrategy):
    def apply_discount(self, price: float) -> float:
        return price * 0.05

class PremiumDiscount(DiscountStrategy):
```

Университет Сириус Колледж

O - Open-closed principle
Принцип открытости/закрытости

Ваши модули или библиотеки должны быть открыты для расширения, но закрыты для модификации.

```python
from abc import ABC, abstractmethod

class DiscountStrategy(ABC):
    @abstractmethod
    def apply_discount(self, price: float) -> float:
        pass

class StandardDiscount(DiscountStrategy):
    def apply_discount(self, price: float) -> float:
        return price * 0.05

class PremiumDiscount(DiscountStrategy):
```

Университет Сириус Колледж

L - Liskov substitution principle
Принцип подстановки Лисков

- Функции (и классы), которые используют указатели или ссылки на базовые классы, должны иметь возможность использовать подтипы базового типа, ничего не зная об их существовании.
- Подкласс не должен создавать новых мутаторов свойств базового класса.

Университет Сириус Колледж

L - Liskov substitution principle
Принцип подстановки Лисков

```python
class Email:
    def __init__(self, username: str, host: str):
        self.email = '{0}@{1}'. format(username, host)

    def isvalid(self):
        known_hosts = ['yandex.ru', 'gmail.com']
        for host in known_hosts:
            if self.email.endswith(host):
                return True
        return False

class SiriusEmail(Email):
    def __init__(self, username: str) :
        super().__init__(username, 'talantiuspeh.ru')
```

Университет Сириус Колледж

L - Liskov substitution principle
Принцип подстановки Лисков

```python
class Email:
    def __init__(self, username: str, host: str):
        self.email = '{0}@{1}'. format(username, host)

    def isvalid(self):
        known_hosts = ['yandex.ru', 'gmail.com']
        for host in known_hosts:
            if self.email.endswith(host):
                return True
        return False

class SiriusEmail(Email):
    def __init__(self, username: str) :
        super().__init__(username, 'talantiuspeh.ru')
```

Университет Сириус Колледж

I - Interface segregation principle
Принцип разделения интерфейса

Программные сущности не должны зависеть от методов, которые они не используют.

Отделяйте и разделяйте методы, не заставляйте пользователей (вашего кода) использовать ненужные или навязанные методы.

Университет Сириус Колледж

I - Interface segregation principle
Принцип разделения интерфейса

```python
from abc import ABC, abstractmethod

class IMultiFunctionDevice(ABC):
    @abstractmethod
    def print_document(self, document):
        pass

    @abstractmethod
    def scan_document(self, document):
        pass

    @abstractmethod
    def fax_document(self, document):
        pass

```

Университет Сириус Колледж

I - Interface segregation principle
Принцип разделения интерфейса

```
2
3  class IMultiFunctionDevice(ABC):
4      @abstractmethod
5      def print_document(self, document):
6          pass
7
8      @abstractmethod
9      def scan_document(self, document):
10         pass
11
12     @abstractmethod
13     def fax_document(self, document):
14         pass
15
16 class SimplePrinter(IMultiFunctionDevice):
```

Университет Сириус Колледж

I - Interface segregation principle
Принцип разделения интерфейса

```
10          pass
11
12      @abstractmethod
13      def fax_document(self, document):
14          pass
15
16  class SimplePrinter(IMultiFunctionDevice):
17      def print_document(self, document):
18          print(f"Печатаю документ: {document}")
19
20      def scan_document(self, document):
21          pass  # Нечего делать
22
23      def fax_document(self, document):
24          raise NotImplementedError("Этот принтер не поддерживает факс")
```

Университет Сириус Колледж

I - Interface segregation principle
Принцип разделения интерфейса

```
9       def scan_document(self, document):
10          pass
11
12      @abstractmethod
13      def fax_document(self, document):
14          pass
15
16  class SimplePrinter(IMultiFunctionDevice):
17      def print_document(self, document):
18          print(f"Печатаю документ: {document}")
19
20      def scan_document(self, document):
21          pass  # Нечего делать
22
23      def fax_document(self, document):
```

Университет Сириус Колледж

I - Interface segregation principle
Принцип разделения интерфейса

```
10          pass
11
12      @abstractmethod
13      def fax_document(self, document):
14          pass
15
16  class SimplePrinter(IMultiFunctionDevice):
17      def print_document(self, document):
18          print(f"Печатаю документ: {document}")
19
20      def scan_document(self, document):
21          pass  # Нечего делать
22
23      def fax_document(self, document):
24          raise NotImplementedError("Этот принтер не поддерживает факс")
```

Университет Сириус Колледж

I - Interface segregation principle
Принцип разделения интерфейса

```
10          pass
11
12      @abstractmethod
13      def fax_document(self, document):
14          pass
15
16  class SimplePrinter(IMultiFunctionDevice):
17      def print_document(self, document):
18          print(f"Печатаю документ: {document}")
19
20      def scan_document(self, document):
21          pass  # Нечего делать
22
23      def fax_document(self, document):
24          raise NotImplementedError("Этот принтер не поддерживает факс")
```

Университет Сириус Колледж

I - Interface segregation principle
Принцип разделения интерфейса

```
10          pass
11
12      @abstractmethod
13      def fax_document(self, document):
14          pass
15
16  class SimplePrinter(IMultiFunctionDevice):
17      def print_document(self, document):
18          print(f"Печатаю документ: {document}")
19
20      def scan_document(self, document):
21          pass  # Нечего делать
22
23      def fax_document(self, document):
24          raise NotImplementedError("Этот принтер не поддерживает факс")
```

Университет Сириус Колледж

I - Interface segregation principle
Принцип разделения интерфейса

```python
from abc import ABC, abstractmethod

class IPrinter(ABC):
    @abstractmethod
    def print_document(self, document):
        pass

class IScanner(ABC):
    @abstractmethod
    def scan_document(self, document):
        pass

class SimplePrinter(IPrinter):
    def print_document(self, document):
        print(f"Печатаю документ: {document}")
```

Университет Сириус Колледж

I - Interface segregation principle
Принцип разделения интерфейса

```python
from abc import ABC, abstractmethod

class IPrinter(ABC):
    @abstractmethod
    def print_document(self, document):
        pass

class IScanner(ABC):
    @abstractmethod
    def scan_document(self, document):
        pass

class SimplePrinter(IPrinter):
    def print_document(self, document):
        print(f"Печатаю документ: {document}")
```

Университет Сириус Колледж

I - Interface segregation principle
Принцип разделения интерфейса

```python
from abc import ABC, abstractmethod

class IPrinter(ABC):
    @abstractmethod
    def print_document(self, document):
        pass

class IScanner(ABC):
    @abstractmethod
    def scan_document(self, document):
        pass

class SimplePrinter(IPrinter):
    def print_document(self, document):
        print(f"Печатаю документ: {document}")
```

Университет Сириус Колледж

I - Interface segregation principle
Принцип разделения интерфейса

```python
from abc import ABC, abstractmethod

class IPrinter(ABC):
    @abstractmethod
    def print_document(self, document):
        pass

class IScanner(ABC):
    @abstractmethod
    def scan_document(self, document):
        pass

class SimplePrinter(IPrinter):
    def print_document(self, document):
        print(f"Печатаю документ: {document}")
```

Университет Сириус Колледж

I - Interface segregation principle
Принцип разделения интерфейса

```python
from abc import ABC, abstractmethod

class IPrinter(ABC):
    @abstractmethod
    def print_document(self, document):
        pass

class IScanner(ABC):
    @abstractmethod
    def scan_document(self, document):
        pass

class SimplePrinter(IPrinter):
    def print_document(self, document):
        print(f"Печатаю документ: {document}")
```

Университет Сириус Колледж

I - Interface segregation principle
Принцип разделения интерфейса

```python
from abc import ABC, abstractmethod

class IPrinter(ABC):
    @abstractmethod
    def print_document(self, document):
        pass

class IScanner(ABC):
    @abstractmethod
    def scan_document(self, document):
        pass

class SimplePrinter(IPrinter):
    def print_document(self, document):
        print(f"Печатаю документ: {document}")
```

Университет Сириус Колледж

I - Interface segregation principle
Принцип разделения интерфейса

```python
from abc import ABC, abstractmethod

class IPrinter(ABC):
    @abstractmethod
    def print_document(self, document):
        pass

class IScanner(ABC):
    @abstractmethod
    def scan_document(self, document):
        pass

class SimplePrinter(IPrinter):
    def print_document(self, document):
        print(f"Печатаю документ: {document}")
```

Университет Сириус Колледж

D - Dependency inversion principle
Принцип инверсии зависимостей

- Модули верхних уровней не должны импортировать сущности из модулей нижних уровней. Оба типа модулей должны зависеть от абстракций.
- Абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций.

Проще говоря: ваш основной код ("бизнес-логика") не должен знать о конкретных "деталях" реализации (как именно что-то пишется в базу данных или отправляется по email). Он должен работать с "контрактом" (абстракцией).

Университет Сириус Колледж

D - Dependency inversion principle
Принцип инверсии зависимостей

```python
class EmailNotifier:
    """Модуль нижнего уровня (деталь)."""
    def send_email(self, message: str):
        print(f"Отправка email: {message}")

class OrderProcessor:
    """Модуль верхнего уровня (бизнес-логика)."""
    def __init__(self):
        self.notifier = EmailNotifier()

    def process_order(self, order_id: int):
        print(f"Обработка заказа #{order_id}...")
        self.notifier.send_email(f"Ваш заказ #{order_id} обработан.")
```

Университет Сириус Колледж

D - Dependency inversion principle
Принцип инверсии зависимостей

```python
class EmailNotifier:
    """Модуль нижнего уровня (деталь)."""
    def send_email(self, message: str):
        print(f"Отправка email: {message}")

class OrderProcessor:
    """Модуль верхнего уровня (бизнес-логика)."""
    def __init__(self):
        self.notifier = EmailNotifier()

    def process_order(self, order_id: int):
        print(f"Обработка заказа #{order_id}...")
        self.notifier.send_email(f"Ваш заказ #{order_id} обработан.")
```

Университет Сириус Колледж

D - Dependency inversion principle
Принцип инверсии зависимостей

```python
class EmailNotifier:
    """Модуль нижнего уровня (деталь)."""
    def send_email(self, message: str):
        print(f"Отправка email: {message}")

class OrderProcessor:
    """Модуль верхнего уровня (бизнес-логика)."""
    def __init__(self):
        self.notifier = EmailNotifier()

    def process_order(self, order_id: int):
        print(f"Обработка заказа #{order_id}...")
        self.notifier.send_email(f"Ваш заказ #{order_id} обработан.")
```

Университет Сириус Колледж

D - Dependency inversion principle
Принцип инверсии зависимостей

```python
class EmailNotifier:
    """Модуль нижнего уровня (деталь)."""
    def send_email(self, message: str):
        print(f"Отправка email: {message}")

class OrderProcessor:
    """Модуль верхнего уровня (бизнес-логика)."""
    def __init__(self):
        self.notifier = EmailNotifier()

    def process_order(self, order_id: int):
        print(f"Обработка заказа #{order_id}...")
        self.notifier.send_email(f"Ваш заказ #{order_id} обработан.")
```

Университет Сириус Колледж

D - Dependency inversion principle
Принцип инверсии зависимостей

```python
class EmailNotifier:
    """Модуль нижнего уровня (деталь)."""
    def send_email(self, message: str):
        print(f"Отправка email: {message}")

class OrderProcessor:
    """Модуль верхнего уровня (бизнес-логика)."""
    def __init__(self):
        self.notifier = EmailNotifier()

    def process_order(self, order_id: int):
        print(f"Обработка заказа #{order_id}...")
        self.notifier.send_email(f"Ваш заказ #{order_id} обработан.")
```

Университет Сириус Колледж

D - Dependency inversion principle
Принцип инверсии зависимостей

```python
from abc import ABC, abstractmethod

class INotifier(ABC):
    @abstractmethod
    def notify(self, message: str):
        pass

class EmailNotifier(INotifier):
    def notify(self, message: str):
        print(f"Отправка email: {message}")

class OrderProcessor:
    def __init__(self, notifier: INotifier):
        self.notifier = notifier

```

Университет Сириус Колледж

D - Dependency inversion principle
Принцип инверсии зависимостей

```python
from abc import ABC, abstractmethod

class INotifier(ABC):
    @abstractmethod
    def notify(self, message: str):
        pass

class EmailNotifier(INotifier):
    def notify(self, message: str):
        print(f"Отправка email: {message}")

class OrderProcessor:
    def __init__(self, notifier: INotifier):
        self.notifier = notifier

```

Университет Сириус Колледж

D - Dependency inversion principle
Принцип инверсии зависимостей

```python
from abc import ABC, abstractmethod

class INotifier(ABC):
    @abstractmethod
    def notify(self, message: str):
        pass

class EmailNotifier(INotifier):
    def notify(self, message: str):
        print(f"Отправка email: {message}")

class OrderProcessor:
    def __init__(self, notifier: INotifier):
        self.notifier = notifier

```

Университет Сириус Колледж

D - Dependency inversion principle
Принцип инверсии зависимостей

```python
from abc import ABC, abstractmethod

class INotifier(ABC):
    @abstractmethod
    def notify(self, message: str):
        pass

class EmailNotifier(INotifier):
    def notify(self, message: str):
        print(f"Отправка email: {message}")

class OrderProcessor:
    def __init__(self, notifier: INotifier):
        self.notifier = notifier

```

Университет Сириус Колледж

D - Dependency inversion principle
Принцип инверсии зависимостей

```python
from abc import ABC, abstractmethod

class INotifier(ABC):
    @abstractmethod
    def notify(self, message: str):
        pass

class EmailNotifier(INotifier):
    def notify(self, message: str):
        print(f"Отправка email: {message}")

class OrderProcessor:
    def __init__(self, notifier: INotifier):
        self.notifier = notifier
```

Университет Сириус Колледж

D - Dependency inversion principle
Принцип инверсии зависимостей

```python
from abc import ABC, abstractmethod

class INotifier(ABC):
    @abstractmethod
    def notify(self, message: str):
        pass

class EmailNotifier(INotifier):
    def notify(self, message: str):
        print(f"Отправка email: {message}")

class OrderProcessor:
    def __init__(self, notifier: INotifier):
        self.notifier = notifier

```

Университет Сириус Колледж

D - Dependency inversion principle
Принцип инверсии зависимостей

```python
from abc import ABC, abstractmethod

class INotifier(ABC):
    @abstractmethod
    def notify(self, message: str):
        pass

class EmailNotifier(INotifier):
    def notify(self, message: str):
        print(f"Отправка email: {message}")

class SmsNotifier(INotifier):
    def notify(self, message: str):
        print(f"Отправка SMS: {message}")

```