# Задание на ООП

## Задание 1

Опишите класс `Song` со следующими свойствами: `title`, `artist`, `release_year`. Создайте экземпляр этого класса.

```
class Song:
  def __init__(self, title, artist, release_year) -> None:
    ...

some_song = Song('Какая-то песня', 'Кто-то', 2024)
print(some_song)
```

Выведите экземпляр класса в консоль. Информативный ли получился вывод?

## Задание 2

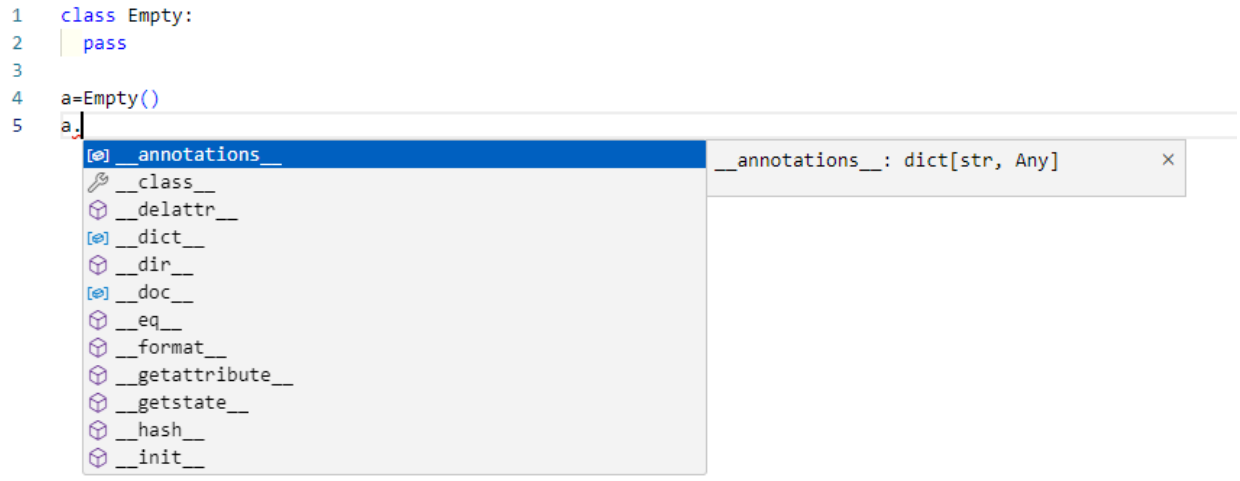
Добавьте классу `Song` метод `get_info`, который будет возвращать свойства класса в виде словаря.

```
class Song:
    def __init__(self, title, artist, release_year) -> None:
       ...

    def get_info(self):
       info = {"title": ...,
                    "artist": ...,
                    "release_year": ...}
        return info

some_song = Song('Какая-то песня', 'Кто-то', 2024)
print(some_song.get_info())
```

Выведите результат, возвращаемый методом `get_info` в консоль. Информативный ли получился вывод?

## Задание 3

На самом деле, в Python любой пользовательский класс неявно наследуется от базового класса object, из-за чего даже экземпляр «пустого» класса будет обладать набором «базовых» свойств и методов:




Некоторые из этих свойств и методов могут быть довольно полезными.

Например, свойство `__dict__` как раз хранит в себе словарь пользовательских свойств объекта класса. В прошлом задании мы сами описали словарь `info`, а могли использовать базовое свойство `__dict__`.

А когда мы пытаемся вывести объект класса в консоль, на самом деле мы вызываем метод `__str__`. Важно, что этот метод всегда должен возвращать строку!

Переопределите метод `__str__`, используя свойство `__dict__`, таким образом, чтобы команда `print(some_song)` выводила в консоль то же самое, что `print(some_song.get_info())` в предыдущем задании.

```
class Song:
    def __init__(self, title, artist, release_year) -> None:
        ...

    def __str__(self) -> str:
        string_info = ...
        return string_info


some_song = Song('Какая-то песня', 'Кто-то', 2024)

print(some_song)
```

Если всё получилось, метод `get_info` можно удалить.

## Задание 4

Опишите класс `Podcast_Episode` со следующими свойствами: `title`, `artist`, `duration`, `guest`, `over_30_min`. Свойство `duration` хранит информацию о длительности подкаста в секундах.

Опишите метод `check_is_over_30_min`, который будет вызываться в конструкторе для определения свойства `over_30_min`. Метод должен принимать в качестве параметра длительность подкаста в секундах и возвращает `True`, если длительность больше 30 минут, и `False` в ином случае.

Переопределите метод `__str__` аналогично тому, как это было сделано в классе `Song`.

Создайте экземпляр этого класса.

```
class Podcast_Episode:
    def __init__(self, title, artist, duration, guest) -> None:
        ...
        self.over_30_min = self.check_is_over_30_min(duration)

    def check_is_over_30_min(self, duration):
        ...

    def __str__(self) -> str:
        string_info = str(self.__dict__)
        return string_info


some_podcast = Podcast_Episode ('Какой-то подкаст', 'Кто-то', 123456, 'Кто-то другой')

print(some_podcast)
```

Выведите экземпляр класса в консоль. Всё работает. Но копировать код — не очень хорошая практика. Гораздо лучше пользоваться наследованием!

## Задание 5

Вынесите общие свойства и методы классов `Song` и `Podcast_Episode` в отдельный класс `Audio_Item`. Унаследуйте от него классы `Song` и `Podcast_Episode`. Создайте экземпляры классов `Song` и `Podcast_Episode`.

```
class Audio_Item:
    def __init__(self,...) -> None:
       ...

class Song(Audio_Item):
    def __init__(self, title, artist, release_year) -> None:
       super().__init__(...)
       ...

class Podcast_Episode(Audio_Item):
    def __init__(self, title, artist, duration, guest) -> None:
        super().__init__(...)
        ...

some_song = Song('Какая-то песня', 'Кто-то', 2024)
print(some_song)

some_podcast = Podcast_Episode('Какой-то подкаст', 'Кто-то', 123456, 'Кто-то другой')
print(some_podcast)
```

Выведите экземпляры классов `Song` и `Podcast_Episode` в консоль.

## Задание 6

Опишите новое свойство is_liked в классе Audio_Item. По умолчанию данное свойство имеет значение False.

Опишите новый метод like в классе Audio_Item. Данный метод должен уметь инвертировать значение свойства is_liked (используйте логическое «не», ключевое слово not). Метод не имеет параметров.

```
class Audio_Item:
    def __init__(self,...) -> None:
       ...
    def like(self):
        ...


class Song(Audio_Item):
    def __init__(self, title, artist, release_year) -> None:
       super().__init__(...)
       ...

class Podcast_Episode(Audio_Item):
    def __init__(self, title, artist, duration, guest) -> None:
        super().__init__(...)
        ...

some_song = Song('Какая-то песня', 'Кто-то', 2024)
print(some_song)

some_podcast = Podcast_Episode('Какой-то подкаст', 'Кто-то', 123456, 'Кто-то другой')
print(some_podcast)
```

Выведите экземпляры классов Song и Podcast_Episode в консоль. Убедитесь, что у объектов обоих классов появились новые свойство и метод.