ДЕТСКАЯ ОНЛАЙН-ШКОЛА ПРОГРАММИРОВАНИЯ
HELLO WORLD

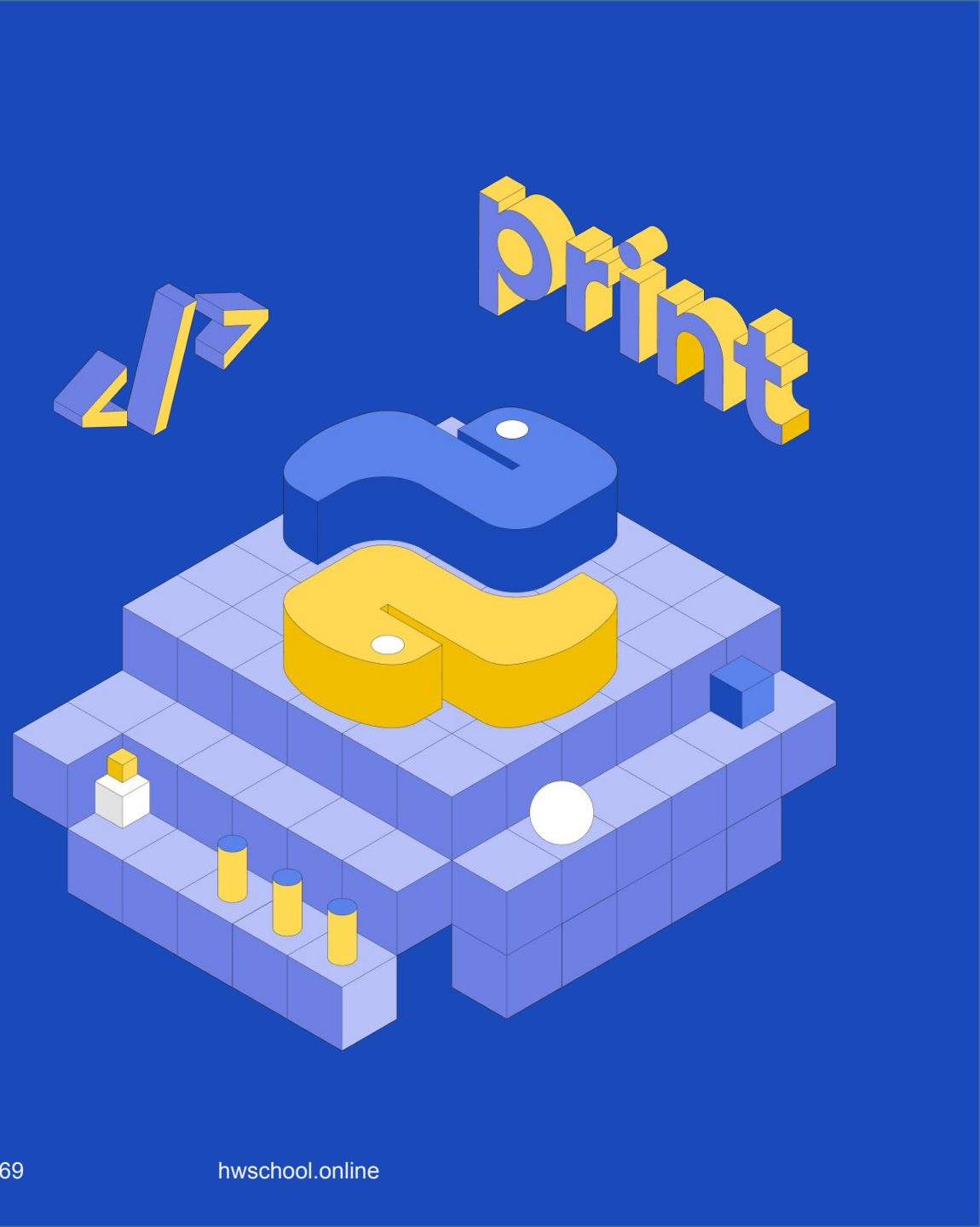
#ШПАРГАЛОЧКИ

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ PYTHON

Продвинутый уровень

Материалы подготовлены отделом методической разработки

Больше полезных материалов и общения в нашем комьюнити в Telegram: https://t.me/hw_school



Книга отзывов. Часть 4

ModelForm - встроенный класс в Django, который позволяет автоматически создать форму на основании полей модели.

Внутри унаследованного от ModelForm класса нужно прописать еще один класс - **Meta**, и уже внутри него связать форму с нужной моделью, а также указать, какие поля нужно создать.

```
from django import forms
from reviews.models import Review

class ReviewForm(forms.ModelForm):
    class Meta:
        model = Review
        fields = ['name', 'email', 'rating', 'review']
```

Некоторым полям может понадобиться валидация - проверка того, что пользователь ввел допустимые значения:

в **models.py**

```
import MinValueValidator, MaxValueValidator
class Review(models.Model):
    ...
    rating = models.IntegerField(validators=[MinValueValidator(1), MaxValueValidator(10)])
```

в **forms.py**

```
class Meta:
    ...
    widgets = { 'rating': forms.NumberInput(attrs={'min': 1, 'max': 10}) }
```

В Django есть встроенная административная часть сайта, которая позволяет взаимодействовать с базами данных через веб-интерфейс. Она доступна по пути **admin/** (нужно добавить к адресу сайта).

Чтобы зарегистрироваться на сайте администрирования, нужно запустить команду

python manage.py createsuperuser

Появится предложение ввести **email** и создать пароль, что и нужно сделать.

Чтобы взаимодействовать с моделью через административный сайт, ее нужно зарегистрировать в файле **admin.py** нашего приложения:

```
from reviews.models import Review

admin.site.register(Review)
```

А чтобы записи из базы данных отображались на странице красивее, нужно переопределить у модели метод **__str__**

```
class Review(models.Model):
    ...
    def __str__(self):
        return f'Review by {self.name} <{self.email}>'
```

Function-Based Views - представления в виде функций

Class-Based Views - представления в виде классов

В классе-преставлении нужно определить методы **get** и **post** для обработки **GET** и **POST** запросов

```
from django.views import View

class ReviewsView(View):
    def get(self, request):
        ...
    def post(self, request):
        ...
```