PolyGen. Инструкция по применению

Шаблонизатор PolyGen предназначен для генерации однотипных html-документов с различным контентом. Разработка - Хыиуду, 2015 г. Вопросы, замечания, предложения - hyyudu@gmail.com

# Что на этой штуке можно делать

Можно - любую несложную (или сложную, в зависимости от опыта верстальщика) полиграфию. Аскольдокарточки, аусвайсы, расписания, письма/телеграммы, карточки для настольных игр и что угодно еще.

# Что этой штуке надобно

Для работы PolyGen требуется установленный Python версии 2.х или 3.х (рекомендуется 3.х).

Для создания верстки нужен html-верстальщик, умеющий в html и css.   
Для изменения контента нужно вносить данные в файлы .csv - значит, требуется либо соответствующая программа (например, Excel), либо просто текстовый редактор (не советую).

Для внесения в шаблоны более сложной логики требуются базовые знания языка Python.

# Структура папок и файлов комплекса

**content** - содержит csv-файлы с контентом (текстами), который должен быть внесен в шаблоны. Файлы должны быть в формате, который Ворд почему-то называет "CSV - разделители запятые", хотя там разделитель точка с запятой.

**images** - сюда обычно складываются картинки, нужные для оформления.

**results** - здесь находятся html-файлы с результатами генерации. Эти результаты можно распечатать непосредственно или перевести в pdf для дальнейшей печати (средства перевода в pdf в PolyGen не входят).

**styles** - здесь находятся css-стили для отображения html-документов

**templates** - а это шаблоны будущих html-документов.

**dicts.py** - файл, содержащий словари, которые можно использовать в шаблонах

**gen.py и gen\_python2.py** - обертка основного файл генератора в двух версиях - для Python 3.x и Python 2.x соответственно. Именно ее надо запускать для генерации документов. Изменять содержимое не нужно.

**gen.bat и gen\_python2.bat** - пакетный файл для запуска предыдущего файла (генератора), опять же, для двух разных версий. Позволяет видеть сообщение об ошибке, если оно вдруг может вам помочь.

**gen\_main.py** - основной файл генератора.

**gen\_config.py** - файл конфигурации, в котором описывается, какие именно документы и как именно надо генерировать.

**in\_project.py** - сюда нужно вписывать функции, специфичные для вашего проекта и нужные для генерации. Если вы не умеете программировать на Python, для вас этот файл бесполезен.

**Readme.docx** - этот документ

**replacements.py** - список автозамен для шаблонов и контента. Например, использование двойных кавычек (") в файлах csv может вызвать неверное их прочтение, поэтому в csv вместо кавычек используется автозамена {quot}. Аналогично с символом ";" (точка с запятой), которая точно нарушит читаемость csv, поэтому, если вам в ваших текстах нужна точка с запятой, используйте в csv вместо нее {semi}

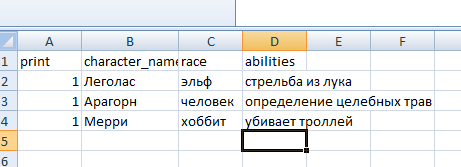
**tools.py, tools\_python3.py и tools\_python2.py** - библиотеки служебных функций генератора (основная, отдельно для Python 3 и отдельно для Python 2). Туда тоже лазить не надо.

**PolyGen-7d3d8b4db102.p12 -** файл ключа, необходимый для доступа генератора к Гугл-документам.

# Можно какой-нибудь самый простой пример?

Допустим, нам надо создать много паспортов персонажей. Размер - А7, ориентация книжная (8 штук на лист А4). На паспорте - имя персонажа, раса и список умений. Называться это будет, допустим, character.

1. Создаем в папке content файл character.csv. В нем делаем нужные нам колонки с произвольными названиями (например, character\_name, race и abilities), а также колонку print - в ней указывается число экземпляров этой карточки. Наполняем файл информацией.



2. Создаем в папке templates файл character.main.htm. В нем делаем html-верстку, размечая семантически структуру шаблона. В нужных местах вставляем метки - названия колонок из csv, заключенные в фигурные скобки.

<div class="character a6\_vert">

<div class=name>{character\_name}</div>

<div class=race>Раса персонажа: {race}</div>

<div class=abilities\_title>Особые способности:</div>

<div class=abilities>{abilities}</div>

</div>

3. Создаем в папке styles файл character.css с css-стилями

.character {

background-image: url(../images/arch.jpg);

float: left;

padding: 5mm;

border: 1px solid black;

}

.name {

width: 100%;

text-align: center;

font-size: 16mm;

}

.abilities\_title {

font-weight: bold;

}

Обратите внимание, в автоматически подключаемом файле main.css есть определения классов a4\_vert, a4\_hor (формат А4 вертикальной/горизонтальной ориентации), аналогично a5\_vert, a5\_hor и т.д.

4. Складываем в папку images нужные картинки (например, в данном случае arch.jpg для фона картинки)

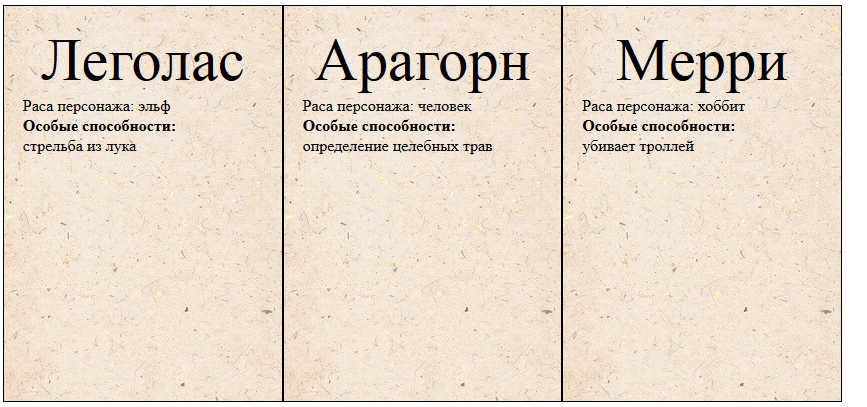
5. Открываем текстовым редактором файл gen\_config.py и добавляем туда по аналогии с остальными запись о том, что нужно обрабатывать и генерировать еще и персонажей, а также информацию о том, как именно генерировать. Как минимум, надо указать, по сколько страниц на лист должно умещаться. :

'character': {

'items\_per\_page': 8

},

6. Запускаем файл gen.py или gen\_python2.py, смотря какая версия Python у вас установлена. Он сформирует файл characters.htm в папке results



После этого никаких изменений в шаблоне и css не нужно - теперь достаточно изменять файл контента и запускать скрипт генерации, чтобы получать аусвайсы новых персонажей.

# Усложняем, ч. 1 - файл конфигурации

Какие еще настройки есть в файле gen\_config.py.

В тех же фигурных скобках, где записан 'items\_per\_page', можно вставлять другие опции, влияющие на генерацию. Название каждой опции заключается в апострофы, а ее значение пишется через двоеточие после названия. Пары "опция-значение" разделяются запятыми

**output\_start** - надо указать название файла в папке templates. Этот текст вставляется в самое начало файла результата. Если не указан, в качестве его используется templates/output\_start.htm

**extra\_head, extra\_body** - текст, которым надо заменять соответствующие метки ({extra\_head} или {extra\_body}) в файле, указанном в output\_start. Вряд ли эта возможность будет применяться часто, но пусть будет.

**wrapper\_class** - css-класс для блока-обертки. Собственно, для этого класса вам в стилях потребуется указать только его ширину (в зависимости от формата бумаги, на которой собираетесь печатать). По умолчанию (если не указан) будет подставлен класс a4\_vert (т.е. 210 х 297мм).

**css** - список css-файлов в папке styles, которые надо подключать. Название каждого отдельного файла заключается в апострофы, а весь список целиком - в квадратные скобки. По умолчанию список состоит из одного файла, одноименного с шаблоном и файлом контента. Если бы в этом примере мы хотели бы, чтобы к файлу characters.htm подключались вместо character.css другие файлы стилей (например, a.css и b.css), нужно было написать в конфиге:

'character': {

'items\_per\_page': 8,

'css': ['a.css', 'b.css']

},

**datafile** - название файла контента (без .csv). Позволяет из одного и того же файла контента создавать несколько разных документов (например, отдельно аусвайсы и отдельно список имен персонажей). Например, если бы мы хотели, чтобы информация о персонажах бралась не из content/characters.csv, а из content/chars.csv, нужно указать 'datafile': 'chars'

**template\_file** - имя файла в папке шаблонов, который будет использоваться для генерации. По умолчанию это <имя типа>.main.htm. Это позволяет, к примеру, генерировать различные документы с одинаковым шаблоном, но разными стилями.

**googlesheet\_url** - если данные хранятся не в csv, а в гуглдоке, здесь надо указать его url. Гуглдок должен быть доступен без авторизации (по ссылке), не иметь более одной страницы, данные должны начинаться с 1 столбца и 1 строки (строка названий полей), названия полей очень желательно писать латиницей и без пробелов. В остальных строках (кроме первой) должно быть НЕ БОЛЬШЕ столбцов, чем в первой. В документе нужно прописать разрешенный доступ с почтового ящика **polygraph.gen@gmail.com**.

Функция работает в режиме бета-теста.

**types**  - список подтипов для родительского типа (см. следующий раздел)

**pass** - если там стоит True, то этот тип документов не генерируется

# Усложняем, ч.2 - подтипы.

Может случиться так, что нужно в одном документе собрать несколько однотипных документов. Например, в один файл telegrams собрать телеграммы сразу двух разновидностей - телеграммы для курьеров и телеграммы для поставщиков. У них одинаковая по формату шапка, но разное внутреннее наполнение. В таком случае действуем так:

1. Создаем два файла в папке content - telegram.courier.csv и telegram.delivery.csv (т.е. название файла состоит из названия типа и названия подтипа через точку). В них есть некоторые совпадающие поля (для шапки, которая есть и там, и там) и некоторые уникальные для каждого подтипа.

2. В папке templates создаем файл telegram.main.htm, где описываем общую часть (шапку). В том месте, где надо вставлять нужную конкретную часть (курьерскую или доставку), ставим метку {main\_part}

3. В той же папке создаем файлы telegram.delivery.htm и telegram.courier.htm, в них делается верстка внутренней части.

4. В css остается один файл telegrams.css, с картинками тоже ничего не меняется.

5. В gen\_config в опции types прописывается список подтипов (аналогично тому, как прописывается список css):

'telegram': {

'items\_per\_page': 4,

'types' : [ 'courier', 'delivery' ]

},

# Усложняем, ч.3 - специальные метки в шаблонах

В файлах шаблонов можно использовать некоторые специальные метки. Допустим, у наших персонажей есть колонка stars, в которой находится некоторое целое число.

**Шаблон повтора строки**  {повторяемая строка \*\*поле csv с числом повторов или просто число}

Например, мы хотим, чтобы у персонажа с 3 звездочками эти самые 3 звездочки и рисовались. А у персонажа с 5 звездочками рисовались, соответственно, пять.

{<img src=../images/star.jpg>\*\*stars}.

Либо просто повторение: {A\*\*5}

**Шаблон окончания числительного** {поле csv : первый вариант окончания : второй вариант окончания : третий вариант окончания}

{stars:звездочка:звездочки:звездочек} - в этом случае, если в колонке stars стоит 3, в этом месте шаблона будет "3 звездочки", а если 1 - "1 звездочка".

Если первым параметром будет не поле контента, а число - оно и будет склоняться. {3:звездочка:звездочки:звездочек} - получим "3 звездочки".

**Шаблон условия**. Допустим, у вас любой персонаж с 3 и более звездочками может делать угабуга. И информацию об этом надо вынести в карточку. Выглядеть это будет, например, так:  
Может ли делать угабуга: {if:item['stars'] >= 3}да{else}нет{endif}.

А если мы не хотим, чтобы информация об угебуге вообще была в карточке того, кто ее не может. тогда лучше сделать так, чтобы текст появился только у того, кто может.  
{if:item['stars'] >= 3}Персонаж может делать угабуга {endif}

Вставлять одну проверку в другую можно, но тогда для каждого условия надо ставить идентификатор, который повторяется в endif:

{if: item['stars'] >= 3}Персонаж может делать угабуга {if: item['stars'] == 5} и даже матумбу! {endif} {endif} - вот так не сработает.

{if ugabuga: item['stars'] >= 3}Персонаж может делать угабуга {if matumba: item['stars'] == 5} и даже матумбу! {endif matumba} {endif ugabuga} - а вот так сработает

**Шаблон словаря**. Допустим, у персонажа есть поле level со значениями от 1 до 5, а мы хотим выводить его на карточке в виде "Уровень: первый" или "Уровень: четвертый". Тогда открываем dicts.py и добавляем новый словарь (если ключами словаря являются числа, их не надо заключать в кавычки, а вот любой текст - нужно):

character\_levels = {1: "Первый", 2: "Второй", 3: "Третий", 4: "Четвертый", 5: "Пятый"}

А в шаблоне надо писать {d~имя словаря в dicts.py:поле в csv}

Т.е. в нашем случае " Персонаж имеет {d~character\_levels:level} уровень"

# Усложняем, ч.4 - добавление собственных процедур на Python

Все программирование идет в файле in\_project.py.

Для каждого отдельного элемента генерации, указанного в конфиге, выполняется get\_aux\_data(procedure\_key), которая должна возвращать словарь. Все эти словари собираются воедино и потом могут использоваться в process\_item\_text. procedure\_key - это ключ словаря в конфиге (для простых типов), либо тип+точка+подтип (например telegram.courier) для сложных.

process\_item\_text позволяет вносить новые ключи в item (объект, содержащий данные из строки файла csv, названия полей - из первой строки файла). Переменная text содержит весь текст шаблона (из templates). Позволяет делать обработки вида "заменить все запятые на ЗПТ, а точки на ТЧК"

# Специальные классы в css

.hyphens - включает переносы  
.a4\_hor, .a4\_vert, .a5\_hor, ..., .a8\_vert - горизонтальные и вертикальные размеры, соответствующие размерам бумаги с А4 по А8  
.prot\_medium\_hor, .prot\_medium\_vert - размеры стандартного среднего протектора (93х66мм)  
.prot\_small\_hor, .prot\_small\_vert - размеры стандартного малого протектора (42х64мм)