

Лабораторная работа

Понятие шаблона, модуль JINJA2

JINJA2 это язык шаблонов, который используется для генерации документов на основе одного или нескольких шаблонов. Шаблоны используются для разных целей, в том числе для генерации html-страниц.

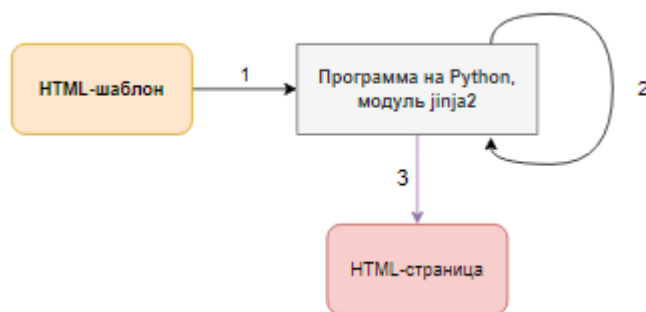
Идея JINJA2 очень проста: модуль поддерживает разделение данных и шаблона. Это позволяет использовать один и тот же шаблон, но подставлять в него разные данные. В самом простом случае шаблон - это просто текстовый файл, в котором указаны места подстановки значений с помощью переменных.

Например, это шаблон страницы `test_template.html`:

```
<html>
  <head>
    <title> Пример JINJA2 </title>
  </head>
  <body>
    <p>
      Привет, {{name}}!
    </p>
  </body>
</html>
```

В этом HTML коде в фигурных скобках указывается имя переменной, вместо которой будет подставляться ее значение.

Технология использования шаблонов для генерации html-страниц следующая (рисунок 1):



- 1 – чтение шаблона
- 2 – генерация HTML-страницы
- 3 – создание HTML-страницы

Рисунок 1. Генерация страниц на основе шаблона

1. Создается файл-шаблон, в котором в самом простом случае в html-код в фигурных скобках вставляются переменные, которые должны быть заменены на конкретные значения (например, `test_template.html`).

2. Создается программа на Python (`test.py`), в которой:

- импортируется метод `Template` библиотеки JINJA2;

```
from jinja2 import Template
```

- читается информация из файла-шаблона в переменную `html`:

```
f_template = open('test_template.html', 'r', encoding='utf-8-sig')
html = f_template.read()
f_template.close()
```

- создается объект-шаблон с помощью метода `Template()` на основе информации, прочитанной из файла-шаблона:

```
template = Template(html)
```

- с помощью метода `render()` генерируется html-код, при этом в качестве параметров метода указываются имена переменных, используемые в шаблоне, которым присваиваются конкретные значения:

```
result_html = template.render(name = "Алина")
```

В переменной `result_html` хранится код, полученный из шаблона, вместо переменной `name` подставляется имя Алина:

Например, страница `test_template.html` после генерации будет выглядеть следующим образом:

```
<html>
  <head>
    <title> Пример JINJA2 </title>
  </head>
  <body>
    <p>
      Привет, Алина!
    </p>
  </body>
</html>
```

- полученный html-код можно сохранить в файл (`test.html`), который затем открыть в браузере:

```
#создадим файл для HTML-страницы
f = open('test.html', 'w', encoding='utf-8-sig')

# выводим сгенерированную страницу в файл
f.write(result_html)
f.close()
```

3. В браузере можно просмотреть полученную страницу (рисунок 2):

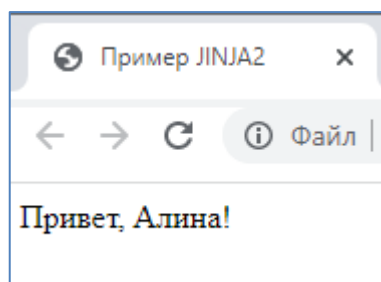


Рисунок 2. Пример использования шаблона

Переменные шаблона

Все подставляемые переменные в шаблоне указываются в двойных фигурных скобках (`{{...}}`). В качестве переменных можно использовать простые переменные (целого, вещественного и др. типов), списки, массивы и пр.

Например, пусть информация (имя, группа, успеваемость) о студентах хранится в списке (его необходимо включить в программу `test.py`):

```
student =[
    ["Алина", "Бизнес-информатика", ["Базы данных",
                                     "Программирование", "Статистика"]],
    ["Вадим", "Экономика", ["Информатика", "Теория игр",
                             "Статистика"]],
    ["Ксения", "Экономика", ["Информатика", "Теория игр",
                             "Статистика"]]
]
```

Тогда, чтобы вывести информацию об одном студенте на страницу будет использоваться шаблон (`test_template.html`):

```
<html>
<head>
    <title> Пример JINJA2 </title>
</head>
<body>
    <p>
        Привет, {{user[ind][0]}}!
    </p>
    <p>
        Ваша образовательная программа {{user[ind][1]}}.
    </p>
    <p>
        Вы изучаете дисциплины:<br>
        {{user[ind][2][0]}}, {{user[ind][2][1]}}, {{user[ind][2][2]}}.
    </p>
</body>
</html>
```

В этом шаблоне используется две переменные:

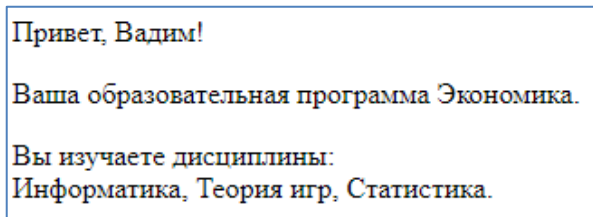
`user` – это список, в котором хранится информация о студенте, обращение к элементам списка осуществляется также как в программе Python;

`ind` – это номер студента в списке, информацию о котором нужно вывести.

Для генерации страницы в программе `test.py` будет использоваться оператор:

```
result_html = template.render(user = student,
                               ind = 1)
```

Результат генерации приведен на рисунке 3.



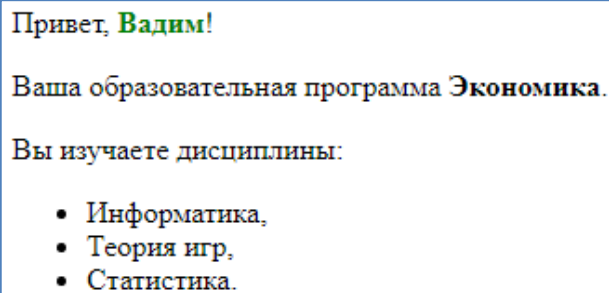
Привет, Вадим!

Ваша образовательная программа Экономика.

Вы изучаете дисциплины:
Информатика, Теория игр, Статистика.

Рисунок 3. Пример страницы

Самостоятельное задание. Сохранить шаблон `test_template.html` под именем `ind_test_template.html`, затем исправить последний так, чтобы информация на html-странице выводилась в виде, показанном на рисунке 4.



```
Привет, Вадим!  
Ваша образовательная программа Экономика.  
Вы изучаете дисциплины:  
• Информатика,  
• Теория игр,  
• Статистика.
```

Рисунок 4. Страница, сгенерированная по шаблону

Конструкции программирования в шаблоне

При создании шаблонов можно использовать конструкции программирования: оператор присваивания, условный оператор, цикл и др. Эти конструкции вставляются в шаблон в операторные скобки `{% ... %}`, обязательно включают начало конструкции и ее конец.

Оператор присваивания

Синтаксис оператора присваивания:

```
{% set переменная = выражение %}
```

В результате выполнения этого оператора будет вычислено значение выражение справа и занесено в переменную. После чего эту переменную можно использовать в любом месте шаблона.

Условный оператор

Синтаксис условного оператора:

```
{% if условие %}  
...  
{% elif условие %}  
...  
{% else %}  
...  
{% endif %}
```

Разделы `elif` и `if` являются необязательными, `elif` может повторяться любое количество раз.

Условие – это логическое выражение, допустимое в языке Python. В нем можно использовать переменные, передаваемые в шаблон.

Например, в нашем примере в список `student` для каждого студента включим его пол:

```
student =[  
    ["Алина", "Бизнес-информатика", ["Базы данных",  
                                         "Программирование", "Статистика"], "ж"],  
    ["Вадим", "Экономика", ["Информатика", "Теория игр",  
                              "Статистика"], "м"],  
    ["Ксения", "Экономика", ["Информатика", "Теория игр",  
                              "Статистика"], "ж"]  
]
```

И в шаблоне исправим приветствие:

```
<html>
  <head>
    <title> Пример JINJA2 </title>
  </head>
  <body>
    <p>
      Привет,
      {% if user[ind][3] == "ж" %}
        студентка
      {% else %}
        студент
      {% endif %}
      {{user[ind][0]}}!
    </p>
    <p>
      Ваша образовательная программа {{user[ind][1]}}.
    </p>
    <p>
      Вы изучаете дисциплины:<br>
      {{user[ind][2][0]}}, {{user[ind][2][1]}}, {{user[ind][2][2]}}.
    </p>
  </body>
</html>
```

В результате, в зависимости от значения параметра `ind` будет выведено разное приветствие (рисунок 5).

Привет, студент Вадим!	Привет, студентка Алина!
Ваша образовательная программа Экономика.	Ваша образовательная программа Бизнес-информатика.
Вы изучаете дисциплины: Информатика, Теория игр, Статистика.	Вы изучаете дисциплины: Базы данных, Программирование, Статистика.

Рисунок 5 Результат генерации html-страницы с разными параметрами

Самостоятельное задание. Исправить шаблон `ind_test_template.html` так, чтобы перед названием образовательной программы выводился ее код (рисунок 6).

Привет, студентка Алина !	Привет, студент Вадим !
Ваша образовательная программа 38.03.05 Бизнес-информатика .	Ваша образовательная программа 38.03.01 Экономика .
Вы изучаете дисциплины: <ul style="list-style-type: none">• Базы данных,• Программирование,• Статистика.	Вы изучаете дисциплины: <ul style="list-style-type: none">• Информатика,• Теория игр,• Статистика.

Рисунок 6. Результат генерации html-страницы с разными параметрами

Оператор цикла

Синтаксис оператора цикла:

```
{% for переменная in range(...) %}
...
{% endfor %}
```

или

```
{% for переменная in список %}  
...  
{% endfor %}
```

Внутри операторных скобок можно использовать переменные, передаваемые в шаблон.

Например, изменим список `student` так, чтобы количество изучаемых дисциплин для каждого студента было разным:

```
student =[  
    ["Алина", "Бизнес-информатика", ["Базы данных",  
                                        "Программирование", "Эконометрика", "Статистика"], "ж"],  
    ["Вадим", "Экономика", ["Информатика", "Теория игр",  
                             "Экономика", "Эконометрика", "Статистика"], "м"],  
    ["Ксения", "Экономика", ["Информатика", "Теория игр",  
                             "Статистика"], "ж"]  
]
```

Тогда для вывода дисциплин необходимо использовать цикл:

```
<html>  
  <head>  
    <title> Пример JINJA2 </title>  
  </head>  
  <body>  
    <p>  
      Привет,  
      {% if user[ind][3] == "ж" %}  
        студентка  
      {% else %}  
        студент  
      {% endif %}  
      {{user[ind][0]}}!  
    </p>  
    <p>  
      Ваша образовательная программа {{user[ind][1]}}.  
    </p>  
    <p>  
      Вы изучаете дисциплины:<br>  
      {% for dis in user[ind][2] %}  
        {{dis}},  
      {% endfor %}  
    </p>  
  </body>  
</html>
```

В результате, в зависимости от значения параметра `ind` будет сформирована разная страница (рисунок 7).

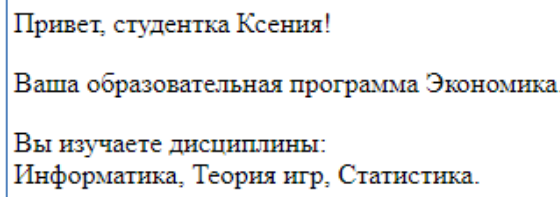
Привет, студент Вадим! Ваша образовательная программа Экономика. Вы изучаете дисциплины: Информатика, Теория игр, Экономика, Эконометрика, Статистика.	Привет, студентка Ксения! Ваша образовательная программа Экономика. Вы изучаете дисциплины: Информатика, Теория игр, Статистика,
---	---

Рисунок 7. Результат генерации html-страницы с разными параметрами

На рисунке 7 видно, что при такой организации цикла после последней дисциплины выводится запятая, а не точка. То есть нужно в цикле проверить, является ли очередной элемент из списка последним, если да – то вывести после названия дисциплины точку, в противном случае – запятую (ниже приведен фрагмент изменяемого шаблона):

```
<p>
    Вы изучаете дисциплины:<br>
    {% for dis in user[ind][2] %}
        {% if dis == user[ind][2][-1] %}
            {{dis}}.
        {% else %}
            {{dis}},
        {% endif %}
    {% endfor %}
</p>
```

На рисунке 8 показан результат выполнения программы с новым шаблоном.



```
Привет, студентка Ксения!

Ваша образовательная программа Экономика.

Вы изучаете дисциплины:
Информатика, Теория игр, Статистика.
```

Рисунок 8. Результат генерации html-страницы

Для вывода дисциплин каждого студента можно передавать в шаблон не весь список со студентами, а информацию об одном студенте, а также количество дисциплин, которое изучает этот студент. Генерация шаблона в этом случае будет выглядеть следующим образом:

```
result_html = template.render( user = student[2],
                                n = len(student[2][2])
)
```

А сам шаблон будет иметь вид:

```
<html>
<head>
    <title> Пример JINJA2 </title>
</head>
<body>
    <p>
        Привет,
        {% if user[3] == "ж" %}
            студентка
        {% else %}
            студент
        {% endif %}
        {{user[0]}}!
    </p>
    <p>
        Ваша образовательная программа {{user[1]}}.
    </p>
    <p>
        Вы изучаете дисциплины:<br>
        {% for i in range(n) %}
            {% if i == n - 1 %}
                {{user[2][i]}}.
            
```

```

        {% else %}
            {{user[2][i]}} ,
        {% endif %}
    {% endfor %}

```

</p>
 </body>
 </html>

Результат генерации html-страницы будет совпадать с результатом предыдущего способа (рисунок 7).

Самостоятельное задание. Исправить шаблон `ind_test_template.html` так, чтобы результат генерации выглядел, как показано на рисунке 9.

Привет, студентка Ксения! Ваша образовательная программа 38.03.01 Экономика . Вы изучаете дисциплины: <ul style="list-style-type: none"> • Информатика, • Теория игр, • Статистика. 	Привет, студент Вадим! Ваша образовательная программа 38.03.01 Экономика . Вы изучаете дисциплины: <ul style="list-style-type: none"> • Информатика, • Теория игр, • Экономика, • Эконометрика, • Статистика.
--	--

Рисунок 9. Результат генерации html-страницы с разными параметрами

Использование функций

Если в шаблоне необходимо использовать функции (из библиотек или созданные в программе Python), перед генерацией их необходимо указать с помощью метода `globals` объекта-шаблона:

```

template.globals['имя_функции_в_шаблоне'] = имя_функции_в_программе
переменная = template.render(...)

```

Другим способом передачи функции в шаблон является возможность указать имя функции в списке параметров метода `render()`:

```

переменная = template.render(...,
                               имя_функции_в_шаблоне = имя_функции_в_программе
                              )

```

Рекомендуется использовать первый способ, чтобы разделить передаваемые параметры и функции.

Например, напишем функцию `add_spaces()`, которая будет после каждой буквы строки вставлять пробел:

```

def add_spaces(text):
    return " ".join(text)

```

Теперь сделаем доступной эту функцию из шаблона, также добавим функцию для вычисления длины списка (чтобы не передавать в шаблон количество дисциплин, а вычислить значение непосредственно в шаблоне):

```

template.globals["add_spaces"] = add_spaces
template.globals["len"] = len
result_html = template.render(user = student[2])

```

Также функции можно передать следующим образом (результат будет точно таким же):


```
result_html = template.render(user = student[2],
                              add_spaces = add_spaces,
                              len= len
                              )
```

Тогда фрагмент шаблона с использованием функций будет выглядеть следующим образом (сохраним длину списка в переменной n):

```
<body>
<p>
    Привет,
    {% if user[3] == "ж" %}
        студентка
    {% else %}
        студент
    {% endif %}
    {{add_spaces(user[0])}}!
</p>
<p>
    Ваша образовательная программа {{user[1]}}.
</p>
<p>
    Вы изучаете дисциплины:<br>
    {% set n = len(user[2]) %}
    {% for i in range(n) %}
        {% if i == n - 1 %}
            {{user[2][i]}}.
        {% else %}
            {{user[2][i]}},
        {% endif %}
    {% endfor %}
</p>
</body>
```

Результат генерации html-страницы приведен на рисунке 10.

```
Привет, студентка К с е н и я!

Ваша образовательная программа Экономика.

Вы изучаете дисциплины:
Информатика, Теория игр, Статистика.
```

Рисунок 10. Результат генерации html-страницы с использованием функций

Самостоятельное задание. Исправить шаблон `ind_test_template.html` так, чтобы результат генерации выглядел, как показано на рисунке 11 (добавить количество изучаемых дисциплин). Реализовать функцию, которая в качестве параметра получает количество дисциплин `n`, а возвращает слово «дисциплина» в верном падеже (1 дисциплина, 2 дисциплины, 3 дисциплины, 4 дисциплины, 5 дисциплин и т.д.)

Привет, студентка К с е н и я! Ваша образовательная программа 38.03.01 Э к о н о м и к а. Вы изучаете 3 дисциплины: <ul style="list-style-type: none"> • Информатика, • Теория игр, • Статистика. 	Привет, студент В а д и м! Ваша образовательная программа 38.03.01 Э к о н о м и к а. Вы изучаете 5 дисциплин: <ul style="list-style-type: none"> • Информатика, • Теория игр, • Экономика, • Эконометрика, • Статистика.
--	--

Рисунок 11. Результат генерации html-страницы с разными параметрами

Макросы

Макросы в Jinja2 напоминают функции в Python. Суть в том, чтобы сделать код, который можно использовать повторно, просто присвоив ему название. Синтаксис:

```
{% macro имя_макроса (параметры) %}
...
{% endmacro %}
```

Для обращения к созданному макросу с параметром `x_list` используется запись:

имя_макроса(x_list)

Например:

```
{% macro render_list(list) %}
    {% set n = len(list) %}
    {% for i in range(n) %}
        {% if i == n - 1 %}
            {{list[i]}}.
        {% else %}
            {{list[i]}},
        {% endif %}
    {% endfor %}
{% endmacro %}
```

В этом примере создан макрос `render_list`, который принимает обязательный аргумент `list` и выводит в одну строку любой список Python. Элементы списка разделяются запятыми, в конце ставится точка.

В нашем примере список дисциплин, которые изучает студент, хранятся в списке `user[2]`. Для вывода генерации списка используется оператор:

```
{{ render_list(user[2]) }}
```

Определение макроса должно располагаться в файле-шаблоне до его первого вызова.

Вместо того чтобы использовать макросы прямо в шаблоне, лучше хранить их в отдельном файле и импортировать при необходимости.

Предположим, все макросы хранятся в файле `macros.html` в текущей папке. Чтобы импортировать их из файла, нужно использовать инструкцию `import`:

```
{% import "macros.html" as macros %}
```

Теперь можно ссылаться на макросы в файле `macros.html` с помощью переменной `macros`. Например:

```
{{ macros.render_list(user[2]) }}
```

Инструкция `{% import "macros.html" as macros %}` импортирует все макросы и переменные (определенные на высшем уровне) из файла `macros.html` в шаблон. Также можно импортировать определенные макросы с помощью `from`:

```
{% from "macros.html" import render_list %}
```

Самостоятельное задание. Исправить шаблон `ind_test_template.html` так, чтобы генерация списка дисциплин осуществлялась с помощью макроса.