**实验2 HTTP服务器与HTML文档**

**【实验目的】**

1. 深入理解HTTP协议原理及应用。
2. 熟练掌握HTML的基本标签使用，理解HTML文档的结构。
3. 掌握CSS样式表的应用，能够通过不同方式为HTML文档设置样式。
4. 掌握Flask框架中函数装饰器@app.route()的使用，理解其在定义路由和处理请求中的作用。
5. 掌握Jinja2模板引擎在Flask框架中的使用，包括变量传递、控制结构和模板继承。
6. 初步了解Web开发基础和框架。

**【实验环境】**

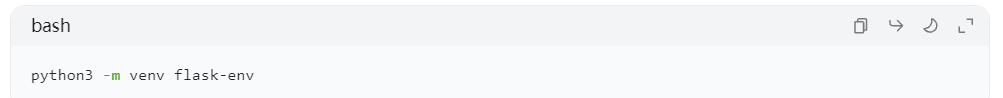
1. **硬件环境：普通计算机一台，具备网络连接功能。**
2. **软件环境：安装有 Python 编程语言环境（建议 Python 3.x 版本），文本编辑器（如 Notepad++、PyCharm 等），以及 Flask 框架。若使用 Windows 系统，还需确保 PowerShell 或命令提示符可正常使用；若为 Linux 或 Mac 系统，需具备相应的终端环境。**

**【实验内容】**

### **一、HTTP服务器及Flask框架应用**

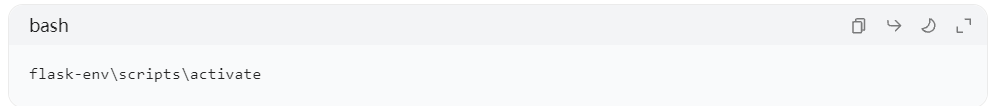
**（一）搭建 Flask 开发环境并创建基础应用**

1. **创建虚拟环境：**打开终端（Windows 系统为 PowerShell 或命令提示符，Linux/Mac 系统为终端），执行以下命令创建名为flask-env的虚拟环境（也可以使用实验室中已经安装好的环境 network）：



1. **激活虚拟环境：**

* Windows 系统：

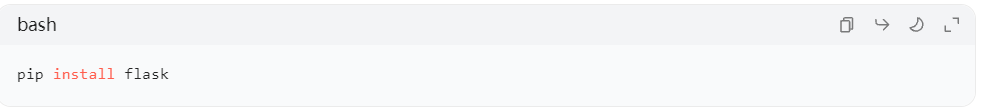


* Linux/Mac 系统：

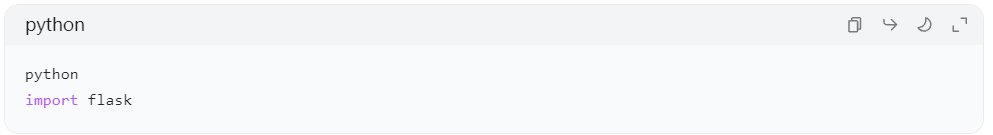


* 如果使用 conda，执行：

1. **安装 Flask 包：**在激活的虚拟环境中，运行以下命令安装 Flask 框架：

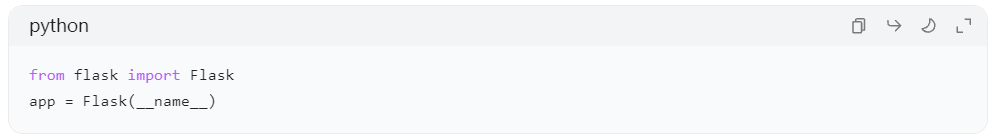


1. **验证安装：**在虚拟环境中进入 Python 交互环境，执行以下命令验证 Flask 是否安装成功：

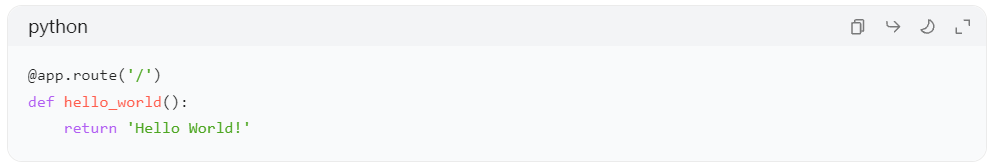


若未报错，则表明 Flask 安装成功，可退出 Python 交互环境（使用quit()命令）。

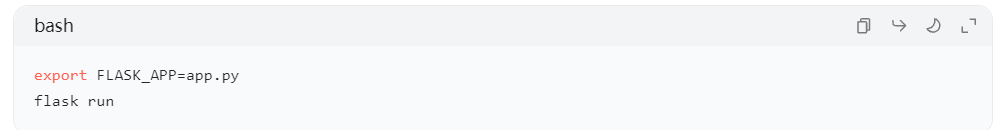
1. **创建 Flask 应用实例：**使用文本编辑器创建一个名为app.py的文件，在文件中编写以下代码创建 Flask 应用实例：



1. **定义基本路由：**在app.py文件中继续编写代码，使用@app.route()装饰器定义根路由（/），当访问该路由时返回 “Hello World!”：



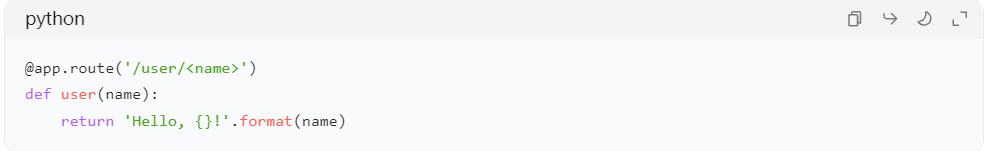
1. **运行 Flask 应用：**在终端中确保处于虚拟环境且位于app.py文件所在目录，执行以下命令运行 Flask 应用：



若为 Windows 系统，使用set FLASK\_APP=app.py命令设置环境变量。运行成功后，在浏览器地址栏输入http://127.0.0.1:5000，可看到页面显示 “Hello World!”，表明 Flask 应用正常运行。

### **（二）深入理解 Flask 函数装饰器@app.route()**

1. **添加带参数的路由：**在app.py文件中添加以下代码，定义一个带参数的路由/user/<name>：



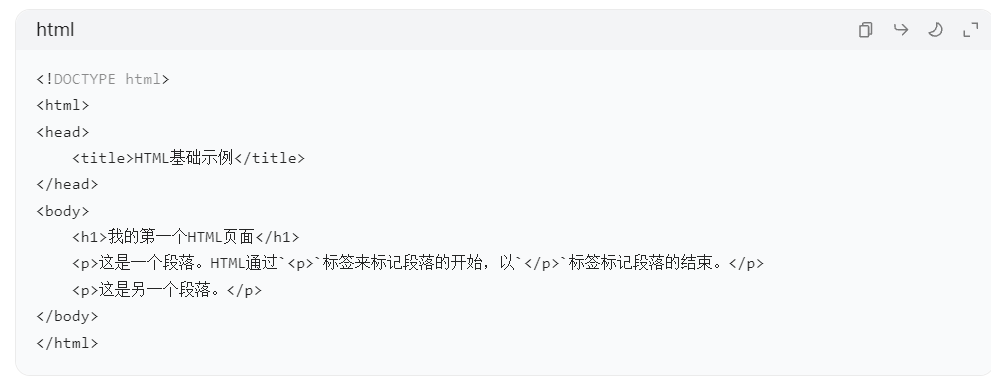
1. **运行并测试新路由：**保存app.py文件，重新运行 Flask 应用（在终端中直接执行flask run，因为已设置好FLASK\_APP环境变量）。在浏览器地址栏中输入http://127.0.0.1:5000/user/你的名字（将 “你的名字” 替换为实际姓名），观察页面显示内容，应看到 “Hello, 你的名字！” 的问候信息，说明@app.route()装饰器能够正确处理带参数的路由请求，并将参数传递给相应的函数进行处理。
2. **分析装饰器作用：**结合代码和实验结果，理解@app.route()装饰器的作用：它用于将一个函数绑定到指定的 URL 路径上，当客户端访问该 URL 时，Flask 框架会调用与之绑定的函数，并将函数的返回值作为响应内容返回给客户端。通过这种方式，开发者可以方便地定义不同的路由和对应的处理逻辑，构建复杂的 Web 应用。

## **（三）实验拓展**

1. **增加更多路由和功能：**在app.py中继续添加不同的路由，如/about路由用于返回关于应用的介绍信息，/contact路由用于返回联系方式等，进一步熟悉@app.route()装饰器的使用。
2. **研究 Flask 请求对象：**在app.py中引入request对象，通过from flask import request导入。在路由函数中使用request对象获取请求的相关信息，如请求方法、请求头、请求参数等，并返回包含这些信息的响应内容，深入了解 Flask 处理请求的机制。

### **二、HTML文档**

1. **HTML 基础实验**
2. ****创建 HTML 基本文档****：打开文本编辑器，创建一个新的文本文件，将其保存为.html后缀的文件，例如basic\_html.html。在文件中编写以下代码：



保存文件后，使用浏览器打开该文件，观察页面显示效果，理解 HTML 文档的基本结构以及段落标签的使用。

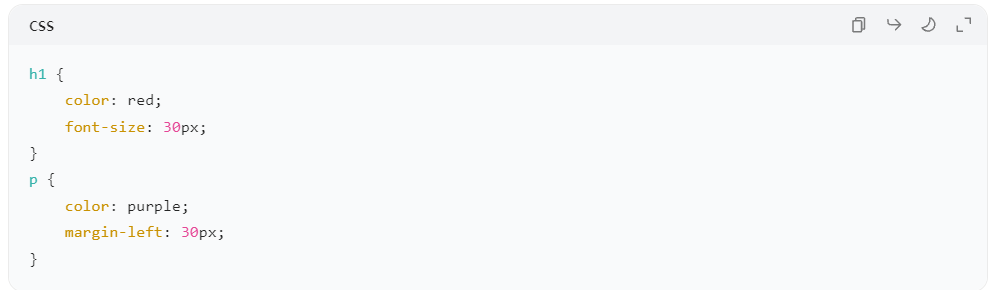
1. ****添加图像和超链接：**在上述basic\_html.html文件的body部分添加图像和超链接，代码如下：**

确保example.jpg图片文件与 HTML 文件在同一目录下（若图片不在同一目录，需修改src属性的路径），保存文件后刷新浏览器，查看图像和超链接的显示效果，理解<img>和<a>标签的使用方法。

1. **CSS 样式应用实验**
2. ****内部样式表****：在basic\_html.html文件的head部分添加内部样式表，修改代码如下：



保存文件后刷新浏览器，观察页面中标题和段落的样式变化，理解内部样式表的作用和语法。  
****2. 外部样式表：**创建一个新的文本文件，保存为.css后缀的文件，例如styles.css。在styles.css文件中编写以下代码：**



然后在basic\_html.html文件的head部分引入外部样式表，修改代码如下：



保存两个文件后刷新浏览器，观察页面样式的变化，对比外部样式表与内部样式表的效果，理解外部样式表的引入方式和优势。

****3. 内联样式：****在basic\_html.html文件的body部分，为某个段落添加内联样式，修改代码如下：



保存文件后刷新浏览器，查看该段落的样式变化，理解内联样式的使用场景和语法。

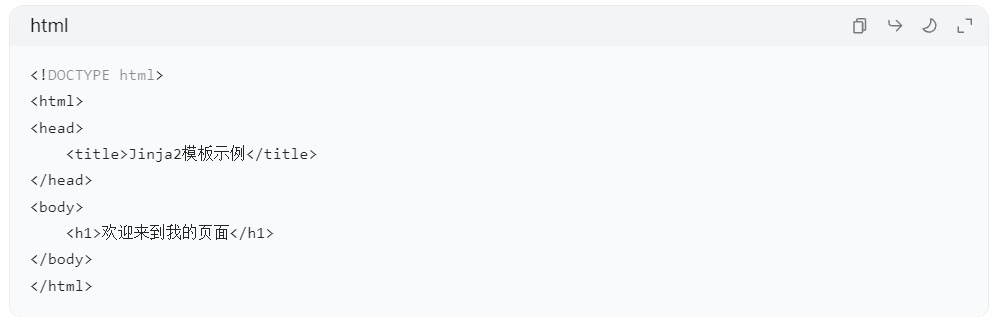
1. **动态文档与模板技术实验（需安装 Python 和 Flask 库）**
2. ****创建 Flask 项目****：打开命令行界面，创建一个新的目录作为 Flask 项目目录，例如flask\_project。进入该目录后，创建一个虚拟环境（可选，但推荐），并安装 Flask 库：



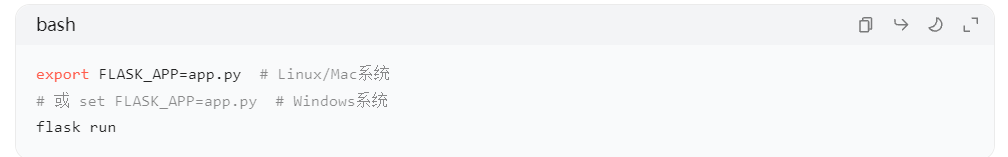
1. ****创建简单的 Flask 应用并使用 Jinja2 模板：****在flask\_project目录下创建一个 Python 文件，例如app.py，编写以下代码：



在flask\_project目录下创建一个名为templates的文件夹，在该文件夹中创建index.html文件，编写以下代码：



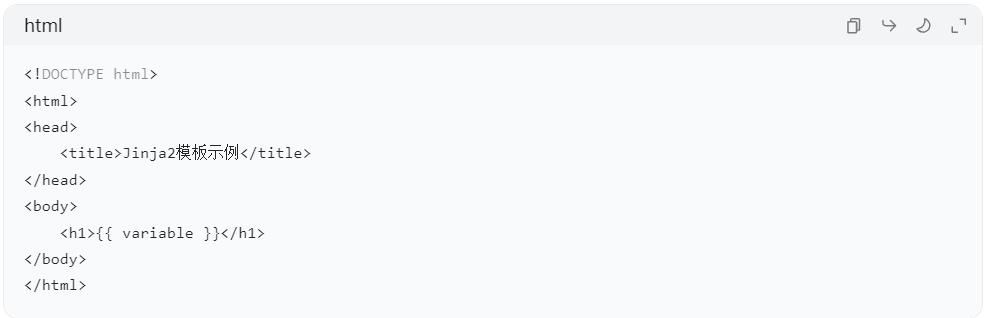
在命令行中运行 Flask 应用



打开浏览器，访问http://127.0.0.1:5000，观察页面显示效果，理解 Flask 框架中模板的基本使用。  
3. ****传递变量到模板****：修改app.py文件，传递变量到模板，代码如下：



修改index.html文件，接收并显示变量：



保存文件后，重新运行 Flask 应用，刷新浏览器，观察变量在模板中的显示效果，理解 Jinja2 模板中变量的传递和使用。

 4.**使用控制结构：**在app.py文件中定义一个列表，传递给模板并使用 Jinja2 的控制结构进行遍历显示，修改代码如下：



修改index.html文件，使用for循环遍历列表：



保存文件后，重新运行 Flask 应用，刷新浏览器，观察列表在模板中的显示效果，理解 Jinja2 模板中控制结构的使用。  
5. ****模板继承****：在templates文件夹中创建一个基础模板base.html，代码如下：



创建一个继承自base.html的模板child.html，代码如下：



修改app.py文件，使其渲染child.html模板：



保存文件后，重新运行 Flask 应用，刷新浏览器，观察页面的布局和样式，理解模板继承的原理和优势。

1. **实验拓展**
2. ****深入研究 CGI****：尝试使用 Python 编写一个简单的 CGI 程序，处理浏览器发送的数据并生成动态文档。在万维网服务器上配置 CGI 环境（可使用 Apache 等服务器），将编写的 CGI 程序部署到服务器上进行测试。
3. ****复杂模板应用****：在 Flask 项目中，进一步探索 Jinja2 模板引擎的高级功能，如自定义过滤器、宏等。使用模板继承构建更复杂的页面布局，提高代码的复用性。