

## Seminararbeit

---

# Python: Apache Tomcat Cookbook (Nutshell Examples)

---

**Verfasser:** Luka Benzia  
**Matrikel-Nr.:** 11778695  
**Studienrichtung:** Wirtschaftsinformatik  
**Kurs:** 4152 Seminar aus BIS  
**Textsprache:** Englisch  
**Betreuer:** Prof. Mag. Dr. Rony G. Flatscher  
**Unternehmen:** Wirtschaftsuniversität Wien

## Declaration of Authorship

I assure:

- To have individually written, to not have used any other sources or tools than referenced and to have used any other unauthorized tools for the writing of this seminar paper.
- To never have submitted this seminar paper topic to an advisor neither in this, nor in any foreign country.
- That this seminar paper matches the seminar paper reviewed by the advisor.

Date: June 01<sup>st</sup>, 2021

Signature Luka Benzia:

A handwritten signature in black ink, appearing to read 'Luka Benzia', written in a cursive style.

## **Abstract**

Apache Tomcat has become one of the most important open source web servers on the market. One of the strengths of Apache Tomcat is its compatibility with the Java programming language, which has become one of the leading languages in the IT sector in recent years. Besides Java, there are of course many other programming languages, including Python which is now one of the most popular programming languages in the world, used by many people every day. The seminar paper brings these two worlds together and makes it possible to use Apache Tomcat with Python. After a theoretical introduction on the relevant topics, various small nutshell examples are presented to demonstrate the possibilities of Apache tomcat in connection with Python. The nutshell examples are some Java Servlets which will be written in Python. The seminar paper concludes with a short summary of the most important findings as well as an outlook with a focus on further research possibilities regarding this specific topic.

# Table of Contents

1	Introduction.....	6
1.1	Objective.....	6
1.2	Methodology .....	6
1.3	Structure .....	7
2	Background.....	8
2.1	Apache Tomcat .....	8
2.1.1	History .....	8
2.1.2	Description .....	8
2.1.3	Open Source .....	9
2.1.4	Lightweight Application.....	9
2.1.5	Stability.....	9
2.1.6	Documentation.....	9
2.2	Python.....	10
2.2.1	History .....	10
2.2.2	Description .....	10
2.2.3	Difference to Java.....	11
2.3	Jython.....	11
2.3.1	History .....	11
2.3.2	Description .....	12
2.3.3	Usage .....	12
3	Nutshell Examples .....	13
3.1	Nutshell context.....	13

3.2	Precautions .....	13
3.3	Python Servlet Nutshell Examples .....	13
3.3.1	Hello world .....	13
3.3.2	Request Info .....	15
3.3.3	Request Header .....	18
3.3.4	Request Parameter.....	20
3.3.5	Cookies .....	24
4	Conclusion .....	30
4.1	Summary.....	30
4.2	Outlook .....	30
	Bibliography.....	32
	Appendix.....	33
	Installation Guide .....	33
	Step 1: Download prerequisite Software .....	33
	Step 2: Create directories and files .....	34
	Step 3: Test the server.....	38

## List of Figures

Figure 1: Google Trends: Python vs. Java .....	11
Figure 2: HelloWorld Servlet Code .....	14
Figure 3: HelloWorld Servlet output .....	15
Figure 4: RequestInfo Servlet Code.....	16
Figure 5: RequestInfo Servlet output .....	17
Figure 6: RequestHeader Servlet Code .....	18
Figure 7: RequestHeader Servlet output.....	19
Figure 8: RequestParam Servlet code .....	21
Figure 9: RequestParam Servlet output without data .....	23
Figure 10: RequestParam Servlet output with data.....	24
Figure 11: Cookies Servlet code .....	25
Figure 12: Cookies Servlet output without cookies .....	27
Figure 13: Cookie Plug-In without cookies.....	27
Figure 14: Cookie Servlet output with cookies .....	28
Figure 15: Cookies Plug-In with cookies.....	29
Figure 16: Apache Tomcat Download .....	34
Figure 17: ApacheTomcatProjekt directory in Windows (C:).....	34
Figure 18: Apache Tomcat unpacked .....	35
Figure 19: webapps directory.....	35
Figure 20: JythonTemplate directory .....	36
Figure 21: WEB-INF and META-INF directory.....	36
Figure 22: lib directory .....	37
Figure 23: jython-standalone-2.7.2.jar.....	37
Figure 24: web.xml content .....	38
Figure 25: WEB-INF directory with web.xml file .....	38
Figure 26: jythonTemplate.py content.....	39
Figure 27: jythonTemplate.py path.....	39
Figure 28: Apache Tomcat Server startup.....	40
Figure 29: Java Window for Server start .....	40
Figure 30: Tomcat Servlet Test.....	41

# 1 Introduction

The seminar paper deals with the implementation of Python code in Apache Tomcat web server. This is made possible by the use of Jython, an implementation of the Python programming language created to run on Java platforms. The theoretical background covers the relevant topics of the seminar paper and gives a deeper insight into how Apache Tomcat and Python work. The functionality of Jython is also explained in this paper and how it is possible to integrate Python into Java concepts with the help of Jython. It is possible to run Java Servlets in Apache Tomcat to automate processes of the web server. With the use of simple Nutshell examples, these Java servlets are taken up and rewritten in the Python programming language with the help of Jython. Based on these nutshell examples, the advantages and disadvantages between these two programming languages are shown in the following. The seminar paper concludes with a short summary of the most important findings as well as an outlook with a focus on further research possibilities regarding this specific topic. A detailed installation guide can be found at the end of the seminar paper, in the Appendix

## 1.1 Objective

The objective of this seminar paper is the integration of the programming language Python into the Apache Tomcat web servers so it is possible to write Python Servlets which will work exactly like the Java Servlets in Apache Tomcat. These Servlets will consequently be present as Nutshell Examples. Furthermore another goal is to create a detailed installation guide to make it easy for anyone to try out the Nutshell Examples themselves.

## 1.2 Methodology

The seminar paper starts with a theoretical part explaining the fundamental elements of the required software. A series of nutshell examples are then provided using pieces of code and images of the Servlet outcome. In addition, there is a detailed installation guide in the

Appendix, which includes every step of the installation in detail with the help of download links and pictures.

### **1.3 Structure**

The seminar paper consists of 4 sections. The first section contains the theoretical part, explaining the fundamental knowledge of the required software. The second section contains the practical part, where different Nutshell examples are explained in detail and presented together with code lines and pictures. The third section provides a brief summary of the outcomes and an outlook on further research within this topic. The fourth and last section offers a detailed installation guide.



## **2 Background**

In this chapter the theoretical Background of the seminar paper will be covered. The first point is an introduction and explanation of the web server Apache Tomcat together with the historical background of this web server. Next the programming language Python and its historical background will be explained. Third, the theoretical background of Jython will be covered. Lastly, an overview of the Servlets and the HTML language which is used in the servlets will be given.

### **2.1 Apache Tomcat**

#### **2.1.1 History**

Originally, the development of Apache Tomcat started as a project by James Duncan Davidson at Sun Microsystems. It started as reference implementation for the Java Servlet and Java Server Pages specifications. In 1999, Sun Microsystems transferred the Tomcat code base to the Apache Software Foundation, a volunteer organisation that promotes Apache software projects. They continued the project as an Open-Source-Project under the umbrella of their Top-Level-Project Jakarta. Due to its great success, it finally became an Apache top-level project on it's own in 2005 and has had its own management structure ever since. (OREILLY, 2021)

#### **2.1.2 Description**

“Apache Tomcat is an application server designed to execute Java servlets and render web pages that use Java Server page coding. It' definitely one of the more popular servlet containers available.”(Davis, 2014)

“The Apache Tomcat® software is an open source implementation of the Jakarta Servlet, Jakarta Server Pages, Jakarta Expression Language, Jakarta WebSocket, Jakarta Annotations

and Jakarta Authentication specifications. These specifications are part of the Jakarta EE platform.” (The Apache Software Foundation, 2014)

### **2.1.3 Open Source**

Apache Tomcat is open source software, it is free for everyone to use and the source code can be downloaded by anyone who is interested. This gives the user complete freedom over his web server and he can change and customise everything to his liking. (Davis, 2014)

### **2.1.4 Lightweight Application**

Tomcat is a very lightweight application which means it offers the most basic functions for a server. This means that there are almost no loading times and the redeploy time is much faster than other applications of this type. “This lightweight nature also allows it to enjoy a significantly faster development cycle.” (Davis, 2014)

### **2.1.5 Stability**

Another reason to use Apache Tomcat is the stability of the platform, which comes from the fact that Tomcat runs independently of the Apache installation. This means that even a major error in Tomcat that stops the application does not cause the server to crash and it continues to run just fine. (Davis, 2014)

### **2.1.6 Documentation**

The application is very well documented on the official web page and there are many tutorials on the internet on how to use the application and how to create the first web application. (Davis, 2014)

## **2.2 Python**

### **2.2.1 History**

The work on the programming language Python began at the end of 1980. A short time later Guido Van Rossum, the Founder of Python, began doing its application based work in December of 1989 by at Centrum Wiskunde & Informatica (CWI). The Python programming language started as a successor to ABC programming language, which has the interfacing with the Amoeba Operating System and has the feature of excepting handling. Van Rossum did already take part at the ABC project earlier in his career. He used the syntax of ABC, some of the best features and fixed some issues of the language. Van was the lead developer until 12 July 2018 when he announced his permanent vacation from his responsibilities of the python language. Python was finally released in 1991 and it used a lot fewer codes to express the concepts compared to Java. (Pramanick, 2019)

### **2.2.2 Description**

“Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.” (Python, 2021)

### 2.2.3 Difference to Java

In the last years Python has become more popular than Java which you can see in the figure 1. The Figure shows the Google Trends and that Python's fame rose above Java in 2017. This trend might come from Python's great use for experimentation code, which is more used nowadays, while Java is better for Production code. (Rowe; Johnson, 2017)

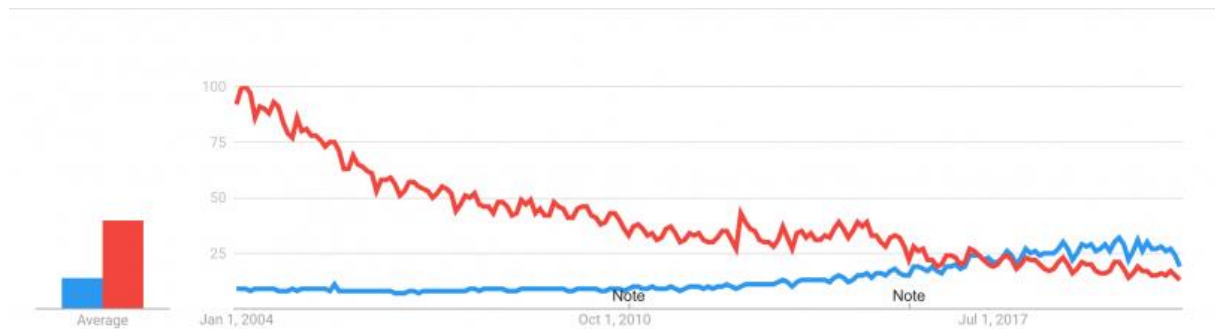


Figure 1: Google Trends: Python vs. Java

While Java is a statically typed and compiled language, Python is a dynamically typed and interpreted language. The difference is that the programming language Java is faster at runtime and also easier to debug, but Python is much easier to read and to use. (Rowe; Johnson, 2017)

## 2.3 Jython

### 2.3.1 History

Jython was created in 1997 by Jim Hugunin. It was originally made to replace C with Java for performance-intensive code accessed by Python programs. In October 2000 they moved to SourceForge and in January 2005 Jython was awarded a grant by the Python Software Foundation. (Jython, 2021)

### **2.3.2 Description**

“Jython is a Java implementation of Python that combines expressive power with clarity. Jython is freely available for both commercial and non-commercial use and is distributed with source code under the PSF License v2.” (Jython, 2021)

### **2.3.3 Usage**

Jython is needed to create Python servlets that can be used in Apache Tomcat. This extension makes it for Tomcat possible to understand and execute the written Python code. So with Jython it is possible to program in Apache Tomcat not only with the programming language Java but also with Python.

## 3 Nutshell Examples

Various Nutshell examples are explained in detail in this chapter and presented with the help of lines of code and pictures. The examples contain fundamental functions of a web server and provide a basic understanding of the functionality of servlets that can be used in Tomcat.

### 3.1 Nutshell context

Apache Tomcat offers a variety of Java servlet examples that can be found under the path "<CATALINA\_HOME>\webapps\examples\WEB-INF\classes" together with the compiled classes. In this seminar paper, these examples serve as the basis for the Nutshell examples.

### 3.2 Precautions

In order to be able to carry out the following Nutshell examples, some precautions must be taken. The download and installation of some software is necessary, including Apache Tomcat, Java and Jython. Some detailed installation instructions can be found in the appendix of this seminar paper. In addition a suitable development environment is required to create the Python servlets. A very popular and often used environment is PyCharm, which also offers a free version:

- **PyCharm download:**

<https://www.jetbrains.com/de-de/pycharm/download/#section=windows>

### 3.3 Python Servlet Nutshell Examples

#### 3.3.1 Hello world

The first servlet is a simple one to show the general functionality of Python servlets.

The first step is to create a new Python file called HelloWorld.py, and save it in the web-apps folder of the Tomcat installation. The code for the first Python servlet is stored in this file and can be retrieved later from the Tomcat server. The Python file should then be filled with the following code. (see figure 2)

```

World.py
HelloWorld.py
1  ### HelloWorld
2
3  from javax.servlet.http import HttpServlet
4
5  class HelloWorld(HttpServlet):
6      def doGet(self, req, res):
7          res.setContentType("text/html")
8          out = res.getWriter()
9
10         out.println("<html><body>"
11                    "<sp><h1>Hello World!</h1></p>"
12                    "<sp>Welcome to the nutshell examples of Python meets Apache Tomcat!</p>"
13                    "<sp>I really hope these examples are useful for you and that you can create awesome things with the wonderful Python language. </p>"
14                    "<hr>"
15                    "<sp>" + u"\u00A9" + " Luka Benzia</p></body></html>"
16                    "</body></html>")
17
18
19  # Luka Benzia
20
21
  
```

Figure 2: HelloWorld Servlet Code

First, the package `java.servlet.http` must be imported (see Code Line 3). Then a new class is created, which must have the same name as the PythonServlet, HelloWorld. In addition, the previously imported HttpServlet must be retrieved from this class. (see Code Line 5) Within this class, a function must now be created, called `doGet`, which contains the variables `self`, `req`, `res`. (see Code line 6).

- **req** = request
- **res** = response

The next step is to tell the server that the response message should be considered as an HTML file (see Code line 7) and then allocate a output writer to write the response message into the network socket. (see Code line 8) As a last point, a response message must be written to the server in an HTML document. (see Code line 10-16) Anything can be sent to the server in this form, it is not necessary to use the exact text from figure 2. Because of the implementation of HTML it is possible to use all the commands that exist in HTML and the server will understand and display them correctly. The HelloWorld servlet is now ready and can be retrieved from

the server. To do this, the server must be started and the file which was just created must be retrieved in a browser.

- **HelloWorld Servlet:** <http://localhost:8080/JythonTemplate/HelloWorld.py>

If everything has been done correctly, the browser should display the following content. (see figure 3)

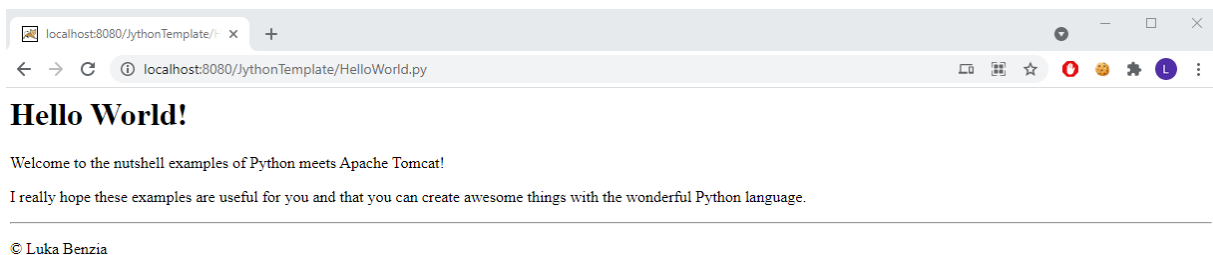


Figure 3: HelloWorld Servlet output

Now the first servlet has been created and successfully retrieved from the server.

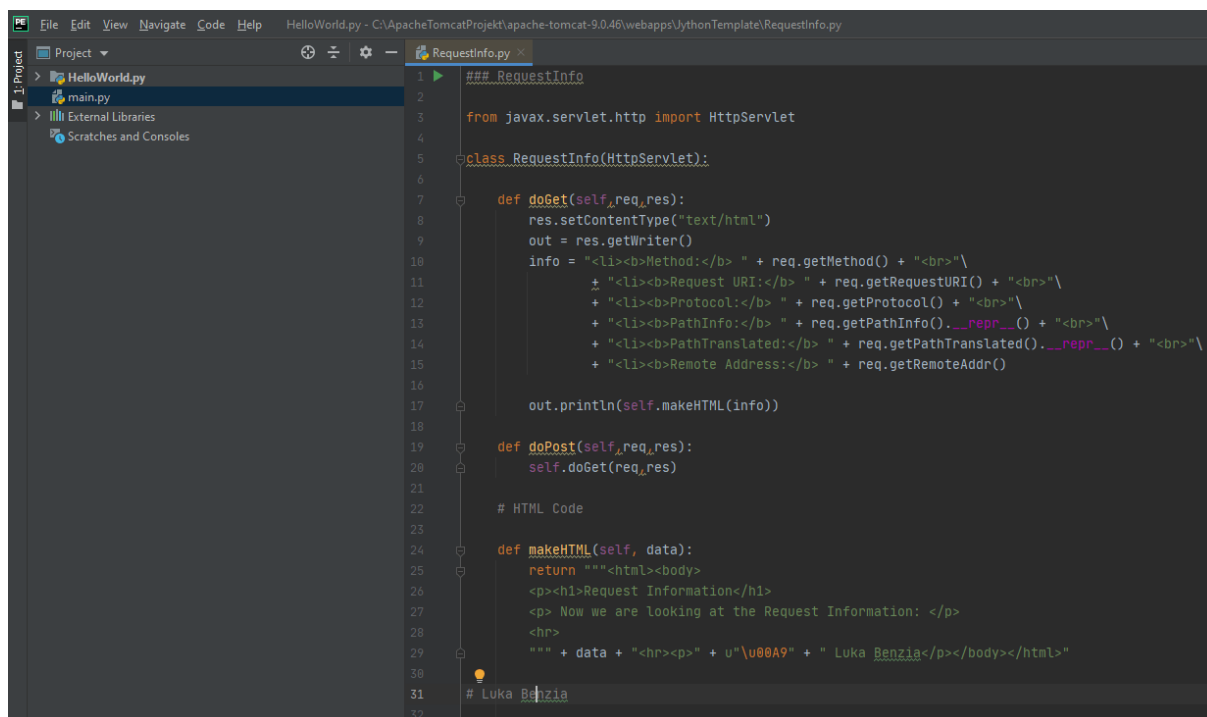
### 3.3.2 Request Info

In the next example some request information of the client will be viewed. The servlet will show us the method, the request URL, the protocol, the path and the remote IP address of the client.

At the beginning, a new file with the name RequestInfo.py is created in the web-apps folder.

Now this file is filled with a few lines of code to get the servlet running. (see figure 4)





```

1  ### RequestInfo
2
3  from javax.servlet.http import HttpServlet
4
5  class RequestInfo(HttpServlet):
6
7      def doGet(self, req, res):
8          res.setContentType("text/html")
9          out = res.getWriter()
10         info = "<li><b>Method:</b> " + req.getMethod() + "<br>"\
11             + "<li><b>Request URI:</b> " + req.getRequestURI() + "<br>"\
12             + "<li><b>Protocol:</b> " + req.getProtocol() + "<br>"\
13             + "<li><b>PathInfo:</b> " + req.getPathInfo().__repr__() + "<br>"\
14             + "<li><b>PathTranslated:</b> " + req.getPathTranslated().__repr__() + "<br>"\
15             + "<li><b>Remote Address:</b> " + req.getRemoteAddr()
16
17         out.println(self.makeHTML(info))
18
19     def doPost(self, req, res):
20         self.doGet(req, res)
21
22     # HTML Code
23
24     def makeHTML(self, data):
25         return """<html><body>
26             <p><h1>Request Information</h1>
27             <p> Now we are looking at the Request Information: </p>
28             <hr>
29             """ + data + "<hr><p> " + u"\u00A9 " + " Luka Benzia</p></body></html>"
30
31     # Luka Benzia
32
  
```

Figure 4: RequestInfo Servlet Code

The first steps are exactly the same as in the first HelloWorld example, so import the package again (see code line 3), create the class (see code line 5) and the function (see code line 7). In addition, a new variable is now defined in the function with the name info. In this variable, the required information is obtained through various commands. (see code line 10-15)

- **getMethod()** = “HTTP defines a set of request methods to indicate the desired action to be performed for a given resource. Although they can also be nouns, these request methods are sometimes referred as HTTP verbs. Each of them implements a different semantic, but some common features are shared by a group of them:” (MDN contributors, 2021a)
- **getRequestURI()** = prints out the complete URI of the client
- **getProtocol()** = returns the protocol of the client with the version
- **getPathInfo()** = returns the path passed to the servlet
- **getRemoteAddr()** = returns the IP Address of the client

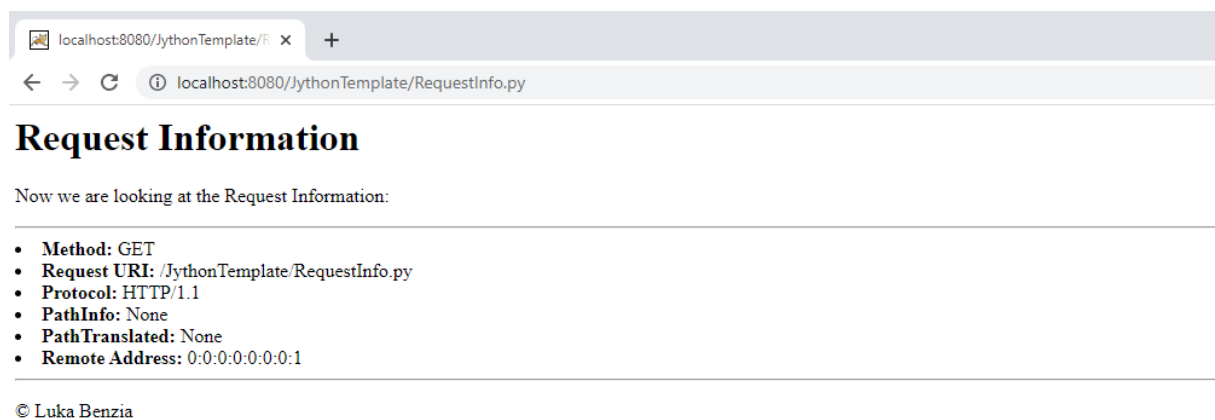
Unlike in the HelloWorld example, the HTML file is not output directly in the next step, but a new function called makeHTML is created for better clarity. This function returns the HTML

file, which can be adapted as desired. (see code line 24) Another advantage of this function is that it is easier to adapt, i.e. you can edit the HTML file and use it anywhere in the code. Furthermore, there is a doPost function for the first time, but it is not yet used in this example. (see figure 4)

The servlet is ready to go. If the server is still running, the file name is now simply changed in the URL.

- **RequestInfo Servlet:** <http://localhost:8080/JythonTemplate/RequestInfo.py>

The browser should display the following content. (see figure 5)



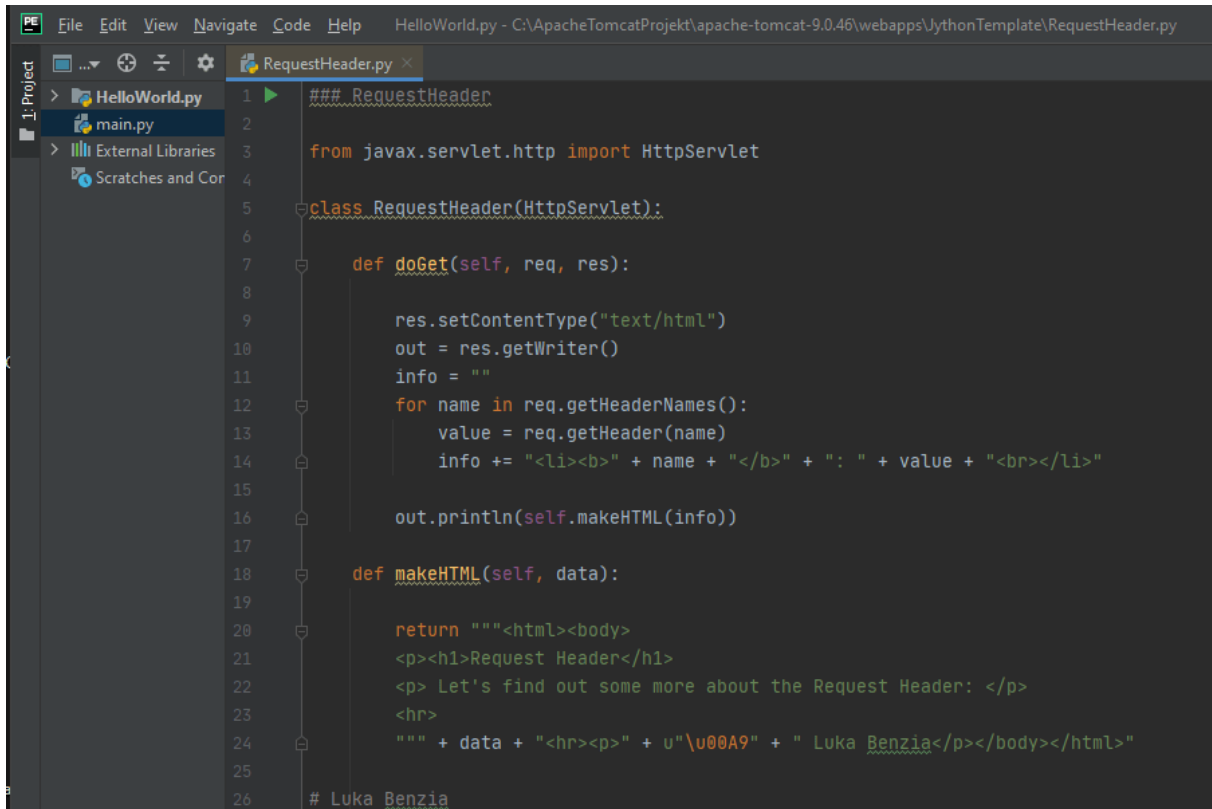
*Figure 5: RequestInfo Servlet output*

The request information is now fetched by the commands of the servlet and displayed in the browser. Thus, the second example is also ready and can be called up at any time.

### 3.3.3 Request Header

The following example highlights further information of the request header. HTTP headers allow the client and server to pass additional information to a request or response.

As in the previous examples, a new file called RequestHeader.py is created at the beginning and filled with some lines of code. (see Figure 6)



```

1  ### RequestHeader
2
3  from javax.servlet.http import HttpServlet
4
5  class RequestHeader(HttpServlet):
6
7      def doGet(self, req, res):
8
9          res.setContentType("text/html")
10         out = res.getWriter()
11         info = ""
12         for name in req.getHeaderNames():
13             value = req.getHeader(name)
14             info += "<li><b>" + name + "</b>" + ": " + value + "<br></li>"
15
16         out.println(self.makeHTML(info))
17
18     def makeHTML(self, data):
19
20         return """<html><body>
21         <p><h1>Request Header</h1>
22         <p> Let's find out some more about the Request Header: </p>
23         <hr>
24         """ + data + "<hr><p>" + u"\u00A9" + " Luka Benzia</p></body></html>"
25
26     # Luka Benzia
  
```

Figure 6: RequestHeader Servlet Code

As in the first two examples, the first steps are exactly the same, they will be skipped from this example onwards. Again, a variable called info is created in the doGet function but this time it is left empty. (see code line 11) Now a loop is formed to store each name and value from the header as its own variable with the help of the getHeaderNames() command. (see code line 12) Then the names are stored together with their value in the previously created variable 'info'. In addition, some HTML code is added to display the output more nicely. (see code line

14) Again, a function is created that contains the HTML file that is to be sent to the Tomcat web server. (see code line 18-24) Now the browser is opened again to try out the servlet.

- **RequestHeader Servlet:** <http://localhost:8080/JythonTemplate/RequestHeader.py>

When everything has been successfully implemented again, the following output is generated. (see Figure 7)

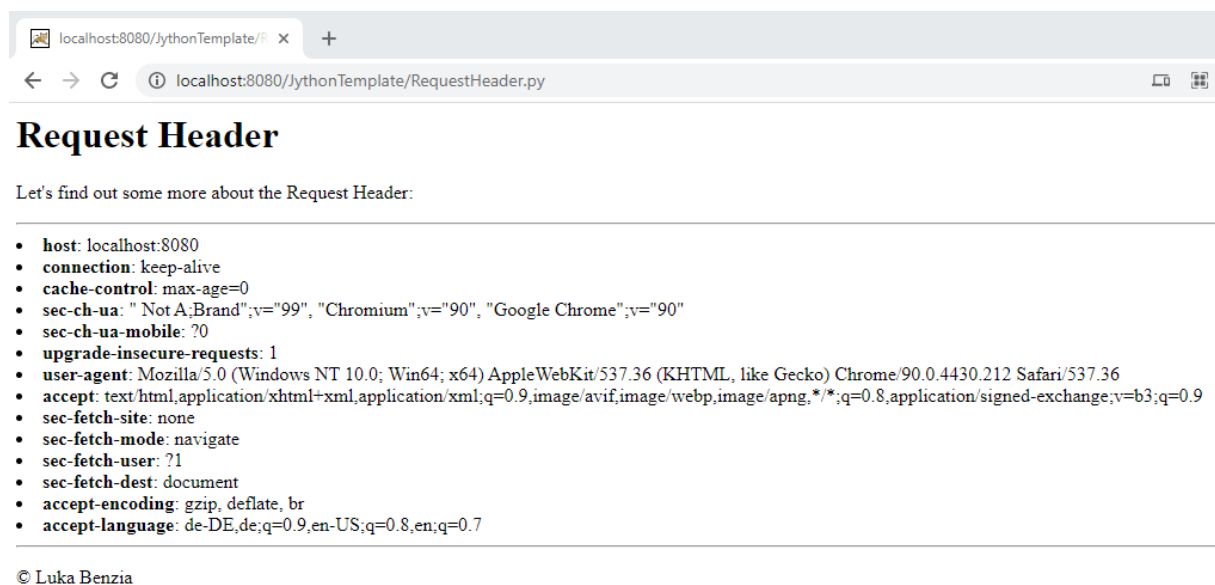


Figure 7: RequestHeader Servlet output

To better understand the output generated, here is an overview:

- **Host:** Specifies the domain name of the server and the TCP port number on which the server is listening. (MDN contributors, 2021b)
- **Connection:** Controls whether the network connection should remain open after the current transaction has ended. (MDN contributors, 2021b)
- **Cache-control:** Specifies instructions for cache mechanisms in requests and responses. (MDN contributors, 2021b)
- **Sec-ch-ua(-mobile):** "The set of Sec-CH-UA-\* client hints aims to deprecate and replace the User-Agent header in order to reduce the passive fingerprinting surface we expose via HTTP requests." (Chrome Platform Status, 2021)

- **Upgrade-insecure-requests:** Sends a signal to the server that the client prefers an encrypted and authenticated response and that the upgrade-insecure-request instruction can be processed successfully. (MDN contributors, 2021b)
- **User-agent:** Contains a characteristic string with which the network protocol partners can determine the application type, operating system, software provider or software version of the requesting user agent. (MDN contributors, 2021b)
- **Accept:** Informs the server what kind of data can be returned. (MDN contributors, 2021b)
- **Sec-fetch-site:** Indicates the relationship between a request initiator's origin and the origin of the resource. (MDN contributors, 2021b)
- **Sec-fetch-mode:** indicates the request's mode. (MDN contributors, 2021b)
- **Sec-fetch-user:** is only sent for requests initiated by user activation, and its value will always be '?1'. (MDN contributors, 2021b)
- **Sec-fetch-dest:** indicates the request's destination, that is how the fetched data will be used. (MDN contributors, 2021b)
- **Accept-encoding:** Notifies the server of the encoding algorithm, usually a compression algorithm, that can be used when returning a resource. (MDN contributors, 2021b)
- **Accept-language:** Notifies the server of the language in which it should return. This is a hint and not necessarily under full control of the user: the server should always be careful not to override an explicit user selection. (MDN contributors, 2021b)

### 3.3.4 Request Parameter

In the next example, data is requested from the user and sent back to the server.

As always, it starts with a new file called RequestParam.py. (see figure 8)

```

PE File Edit View Navigate Code Help HelloWorld.py - C:\ApacheTomcatProjekt\apache-tomcat-9.0.46\webapps\JythonTemplate\RequestParam.py
RequestParam.py x
> ...
> HelloWorld.py
  > main.py
  > External Libraries
  > Scratches and Cor
1  ### Request Parameter
2
3  from javax.servlet.http import HttpServlet
4
5  class RequestParam(HttpServlet):
6
7      def doGet(self, req, res):
8          res.setContentType("text/html")
9          out = res.getWriter()
10         params = ""
11
12         firstName = req.getParameter("firstname")
13         lastName = req.getParameter("lastname")
14         Age = req.getParameter("age")
15         Nation = req.getParameter("nation")
16
17         if firstName != None and lastName != None and Age != None and Nation != None:
18             params = params + \
19                 "<br><b>First Name:</b> " + firstName + \
20                 "<br><br><b>Last Name:</b> " + lastName + \
21                 "<br><br><b>Age:</b> " + Age + \
22                 "<br><br><b>Nation:</b> " + Nation
23         else:
24             params = "No Parameters, Please enter some"
25
26         out.println(self.makeHTML(params))
27
28     def doPost(self, req, res):
29         doGet(self, req, res)
30
31     def makeHTML(self, data):
32         return """<html><body>
33             <p><h1>Request Parameter</h1>
34             <p>Tell me a little bit about yourself!</p>
35             <hr>
36             <h3>Please enter your data:</h3>
37             <form method='get'>
38             <p>
39             First Name:
40             <input type='text' name='firstname'><br />
41             <p>
42             Last Name:
43             <input type='text' name='lastname'><br />
44             <p>
45             Age:
46             <input type='text' name='age'><br />
47             <p>
48             Nation:
49             <input type='text' name='nation'><br />
50             <p>
51             <input type='submit' value='SUBMIT'>
52             <hr>
53             <p>
54             <h3>Check your data:</h3>
55             """ + data + """
56             """<hr><p> + u"\u00A9" + " Luka Benzia</p></body></html>"""
57
58     # Luka Benzia
  
```

Figure 8: RequestParam Servlet code

The first lines contain the usual code, see 'HelloWorld' example. In the doGet function, a variable with an empty set is created again, called params. (see code line 10)

After that, some variables are requested with the getParameter() function and the empty variables are filled with them. The requested data is transmitted by the user in the browser, more on this later. (see code line 12-15)

In the next step, a if-else statement is made that says: If the variables to be filled are not empty, they are sent back to the server and displayed to the user, otherwise a message is displayed saying that no parameters were entered. The empty variable params is then filled with HTML code and the requested variables in the loop. (see code line 17-22)

The empty variable params is then filled with HTML code and the requested variables in the loop. This is called later by the HTML code and then returns the doGet function. ( see code line 28-29)

Then the function makeHTML is filled with HTML code. This time objects that the user can edit are used in this code. (see code line 31-56)

Now the HTML code has to be printed in the doGet function. (see code line 26)

The servlet is now ready and can be accessed in the browser

- **RequestParam Servlet:** <http://localhost:8080/JythonTemplate/RequestParam.py>

The servlet output should look like this: (see figure 9)

localhost:8080/JythonTemplate/R x +

localhost:8080/JythonTemplate/RequestParam.py

## Request Parameter

Tell me a little bit about yourself!

---

**Please enter your data:**

First Name:

Last Name:

Age:

Nation:

---

**Check your data:**

No Parameters, Please enter some

---

© Luka Benzia

Figure 9: RequestParam Servlet output without data

As can be seen in the output, the server requests various data from the user, the First Name, the Last Name, the Age and the Nation. Currently, no data has been transferred yet, therefore the previously created variables are empty and the previously defined text is displayed.

Now the data is filled and submitted. (see figure 10)



localhost:8080/JythonTemplate/F x +

localhost:8080/JythonTemplate/RequestParam.py?firstname=Max&lastname=Muster&age=29&nation=Austria

## Request Parameter

Tell me a little bit about yourself!

---

**Please enter your data:**

First Name:

Last Name:

Age:

Nation:

---

**Check your data:**

**First Name:** Max

**Last Name:** Muster

**Age:** 29

**Nation:** Austria

---

© Luka Benzia

Figure 10: RequestParam Servlet output with data

The requested data has been sent to the server, which now presents this data to the user. So the servlet is complete and can be used at any time.

### 3.3.5 Cookies

This example deals with the so-called cookies.

Cookies are data packets that are exchanged between computer programs. In general, the term is usually used to describe HTTP cookies, which websites use to store user data locally and on the server in order to make individual functions and web applications such as online shops, social networks and forums more user-friendly. (Digital Guide IONOS, 2020)

A new file with code, called Cookies.py, is created. (see figure 10)

```

PE File Edit View Navigate Code Help HelloWorld.py - C:\ApacheTomcatProjekt\apache-tomcat-9.0.46\webapps\JythonTemplate\Cookies.py
1: Project
  > HelloWorld.py
  > main.py
  > External Libraries
  > Scratches and Cor

1  ### Cookies Example
2
3  from javax.servlet import *
4
5  class Cookies(http.HttpServlet):
6
7      def doGet(self, req, res):
8
9          res.setContentType("text/html")
10         out = res.getWriter()
11         params = ""
12
13         # get the cookies
14         cookies = req.getCookies()
15         if cookies != None:
16             for cookie in req.cookies:
17                 name = cookie.getName()
18                 value = cookie.getValue()
19                 params = params + "<br><b>Name: </b> " + name + \
20                     "<br><b>Value: </b>" + value + "<br>"
21
22         # set a cookie using the parameters
23         name = req.getParameter("cookieName")
24         if name != None and len(name) > 0:
25             value = req.getParameter("cookieValue")
26             res.addCookie(http.Cookie(name, value))
27             params = params + "<p><br><b> New cookie: </b>" + name + \
28                 " = " + value
29
30         out.println(self.makeHTML(params))
31
32     def doPost(self, req, res):
33         doGet(self, req, res)
34
35     def makeHTML(self, data):
36         return """<html><body>
37             <p><h1>Cookies</h1>
38             <p> Let's bake some cookies together!</p>
39             <form method='get'>
40             <p>
41             <b>Cookie Name:</b>
42             <input type='text' name='cookieName'><br />
43             <p>
44             <b>Cookie Value:</b>
45             <input type='text' name='cookieValue'><br />
46             <p>
47             <input type='submit' value='SEND'>
48             <hr>
49             <p><b>Your browser is sending the following cookies:</b> </p>
50             """ + data + """
51             """<hr><p>" + u"\u00A9" + " Luka Benzia</p></body></html>"""
52
53     # Luka Benzia
  
```

Figure 11: Cookies Servlet code

As always, the first lines of code remain the same. (see HelloWorld Servlet)

In the doGet function, cookies are requested with the help of the command `getCookies()` (see code line 14)

The next step is to create an if-else statement that says: If the name of the cookie is not empty a loop is created that stores the corresponding value for each name together with some HTML code in a new variable called `params` . (see code line 15-20)

Now a code is written that creates a new cookie that is entered by the user. The name of the cookie is queried and stored in a variable. If this name is not empty and longer than 0, the corresponding value is retrieved and stored. Then, with the help of the command `addCookie()`, a new cookie with name and value is created and sent to the server.(see code line 23-26)

The variable `params` is also filled with HTML code, the name and the value of the cookie to show the user the cookie. ( see code line 27-28)

Again, the `doPost` function is used to send the `doGet` function to the server. ( see code line 32) in the function `makeHTML` the necessary HTML code is saved again, which contains some fields that the user can fill in. (see code line 35-51)

Then the HTML code must be sent to the server with the `params` variable. (see code line 30)

The servlet is ready and can be called up in the browser.

- **Cookies Servlet:** <http://localhost:8080/JythonTemplate/Cookies.py>

The following output is displayed: (see figure 12)

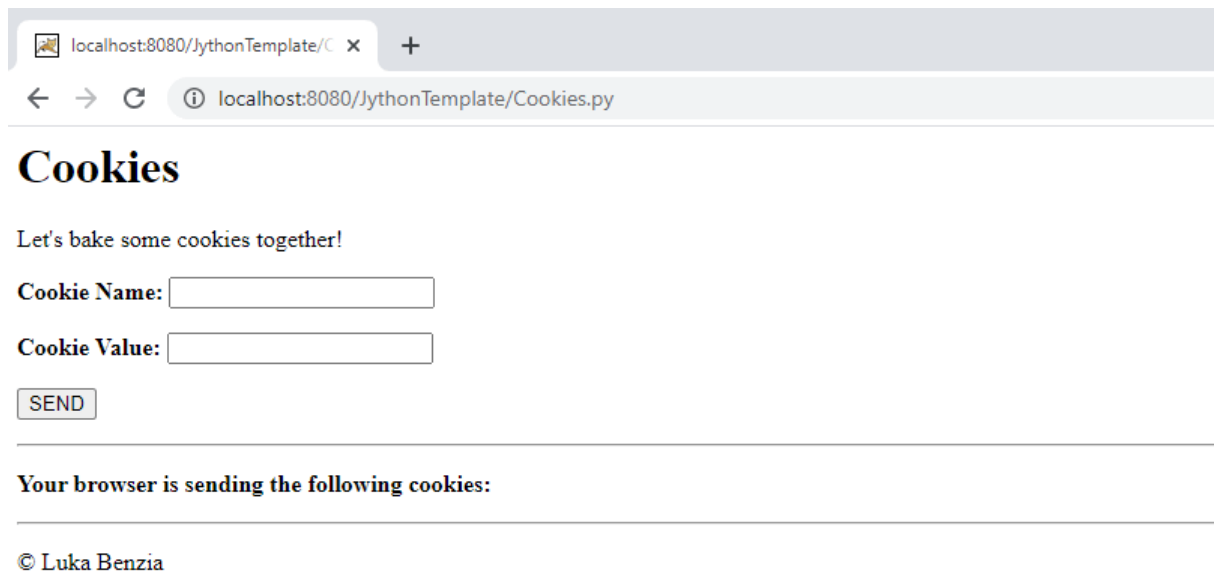


Figure 12: Cookies Servlet output without cookies

Since no cookies have been sent to the server yet, none are displayed. There are some plug-ins for the browser that recognise and display cookies. If such a plug-in is used, this message appears, which says: No cookies available. (see figure 13)

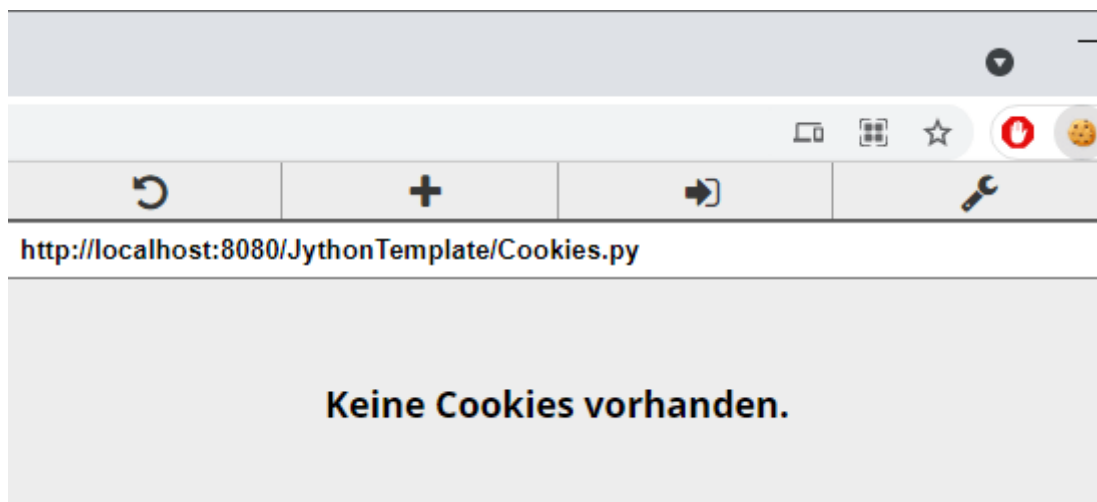


Figure 13: Cookie Plug-In without cookies

New cookies can now be created by entering the name and value of the cookie in the empty fields and then sending them. (see figure 14)

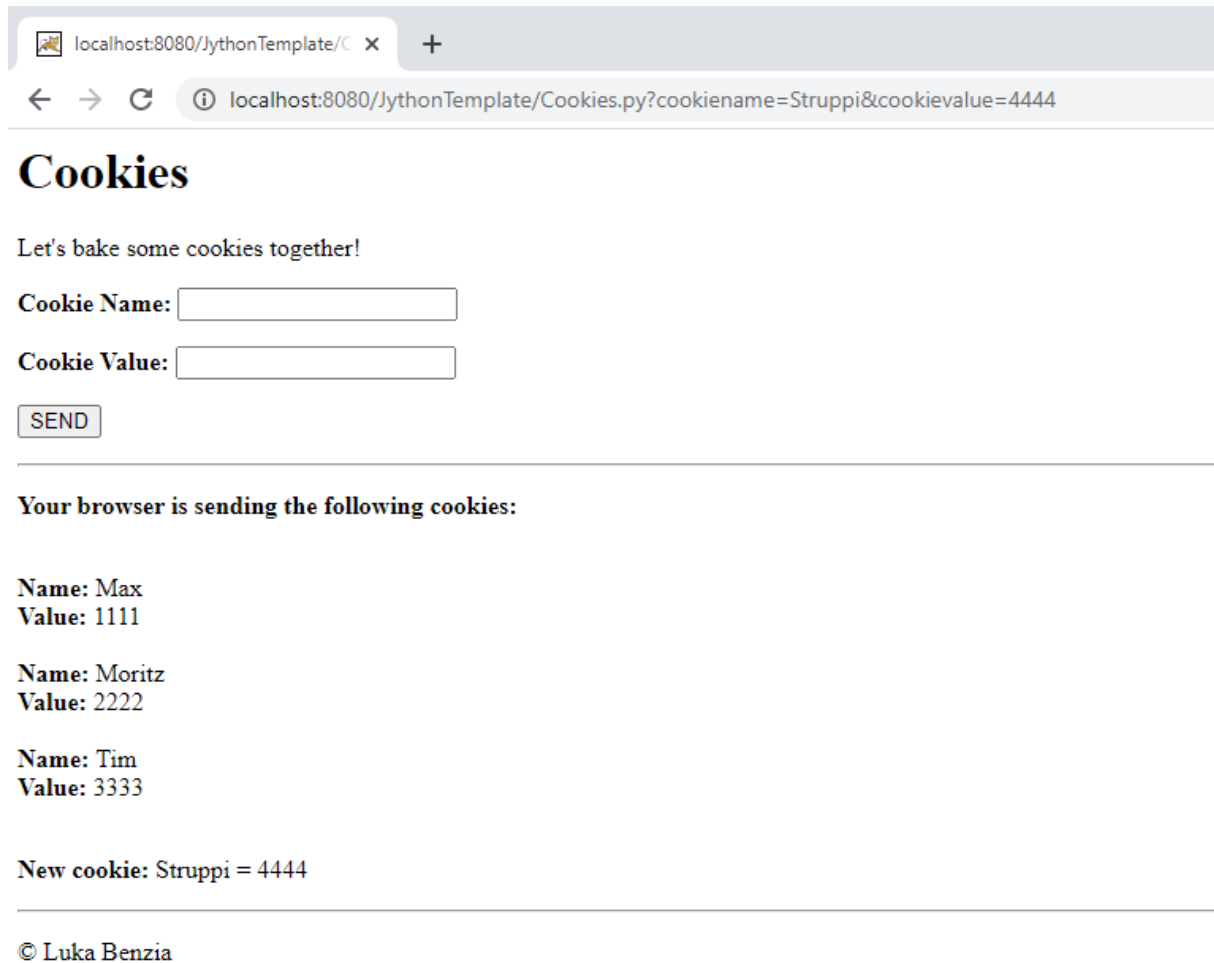


Figure 14: Cookie Servlet output with cookies

The cookies created are now displayed to the user. Previously created cookies are displayed further up and the newest cookie in a separate field below. When the browser's cookie plug-in is used again, the following content appears. (see Figure 15)

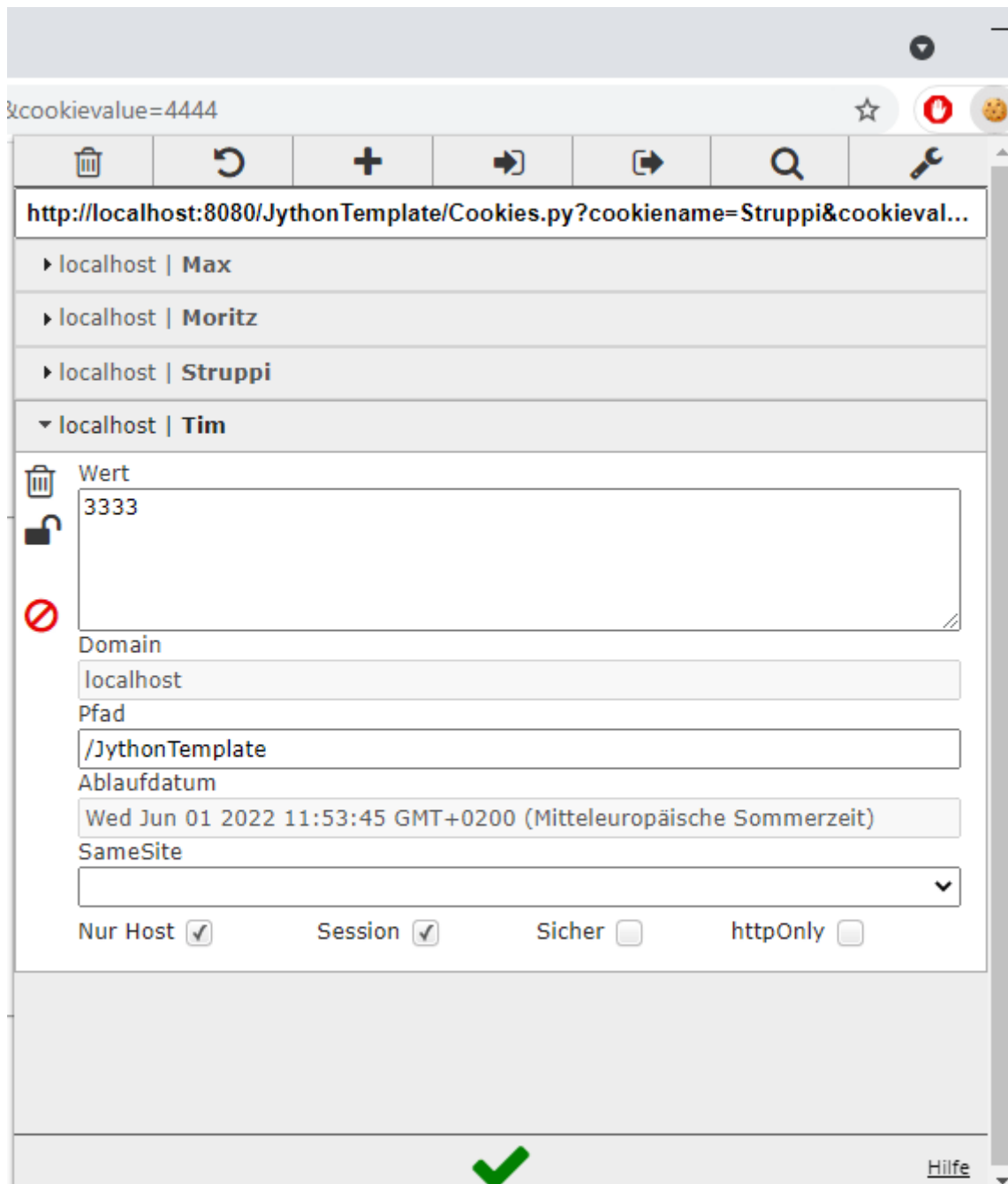


Figure 15: Cookies Plug-In with cookies

The plug-in shows the cookies, so the cookies were actually created and saved in the browser. The servlet is complete and can be used at any time.

## **4 Conclusion**

To conclude the seminar paper, a concise summary is provided to give an overview of the main findings of this work. It also provides an outlook that can be used as an approach for further research in this topic area.

### **4.1 Summary**

In the first part of the thesis, the basic theoretical concepts of the web application Apache Tomcat were discussed together with the historical aspect. Likewise, the theoretical aspects of the Python programming language were drawn in comparison to Java. Finally, the theoretical background of Jython was examined in more detail. This laid the foundation for the practical part, which shows various Nutshell examples that present the functionality and use of Python servlets. It was not only described in detail how to create these Nutshell examples but also how to execute them and to which outcome the respective servlets lead. The Nutshell examples show that using Python simplifies the creation of servlets, as less typing is required. The language is also more forgiving. Another finding of the practical part is that it is easy for anyone to create their own servlets and thus get their own web server up and running. Apache Tomcat is still a powerful tool for web servers and due to the open source aspect it continues to be developed and improved by many people. It is also very well documented and if problems arise, the internet offers many solutions quickly and easily. In comparison with other competing products, Apache Tomcat offers numerous features. The configuration progress is very simple as well as the installation of Plug-Ins like Jython.

### **4.2 Outlook**

The seminar paper was an insight into the world of web servers. It is intended to be a basis for the use of Python for servlets that can be used in Apache Tomcat. Positively, the implementation of Python using Jython was working accurately and had no problem being implemented. Further work could focus on implementing other programming languages and adopting the functionality of Java Servlets. The seminar paper offers only a small collection of

nutshell examples that explain and present the basic concepts of Servlets. Thus, another interesting aspect for future work would be the creation of more complex Nutshell examples that exploit the multitude of functions that Apache Tomcat offers. In principle, further approaches in this thematic area are unlimited, and the seminar work provides the foundation for this.



## Bibliography

- Chrome Platform Status (2021). Feature: Sec-CH-UA Client Hints. Retrieved May 31, 2021, from <https://www.chromestatus.com/feature/5995832180473856>
- Davis, M. (2014, June 17). Five Reasons You Should Use Tomcat (Updated for 2020). Retrieved May 29, 2021, from Future Hosting website: <https://www.futurehosting.com/blog/five-reasons-you-should-use-tomcat/>
- Digital Guide IONOS (2020). Was sind Cookies? Retrieved May 31, 2021, from <https://www.ionos.at/digitalguide/hosting/hosting-technik/was-sind-cookies/>
- Jython (2021). HISTORY OF THE SOFTWARE. Retrieved May 29, 2021, from <https://www.jython.org/jython-old-sites/archive/22/history.html>
- MDN contributors (2021a). HTTP request methods. Retrieved May 31, 2021, from <https://developer.mozilla.org/de/docs/Web/HTTP/Methods>
- MDN contributors (2021b). HTTP Header. Retrieved May 31, 2021, from <https://developer.mozilla.org/de/docs/Web/HTTP/Headers>
- OREILLY (2021). Where Did Tomcat Come From? Retrieved May 29, 2021, from <https://www.oreilly.com/library/view/tomcat-the-definitive/9780596101060/ch01s05.html>
- Pramanick, S. (2019, May 06). History of Python. Retrieved May 29, 2021, from <https://www.geeksforgeeks.org/history-of-python/>
- Python. (2021). What is Python? Executive Summary. Retrieved May 29, 2021, from <https://www.python.org/doc/essays/blurb/#:~:text=Python%20is%20an%20interpreted%2C%20object,programming%20language%20with%20dynamic%20semantics.&text=Python's%20simple%2C%20easy%20to%20learn,program%20modularity%20and%20code%20reuse.>
- Rowe, W.; Johnson, J. (2017). Python vs Java: What's the difference? Retrieved May 29, 2021, from <https://www.bmc.com/blogs/python-vs-java/#:~:text=Java%20is%20a%20statically%20typed,use%20and%20easier%20to%20read.>

# Appendix

## Installation Guide

### Step 1: Download prerequisite Software

The first installation step is to download the necessary software, which is the basic requirement for your first own web server. You need to download either Java Runtime Environment (JRE) or Java Development Kit (JDK) because these tools help you to add Java to your Windows environment variables what makes it possible to use Apache Tomcat. After the software has been downloaded, it must be installed.

- **Java Runtime Environment:** <https://www.java.com/de/download/manual.jsp>
- **Java Development Kit:** <https://www.oracle.com/java/technologies/javase-jdk16-downloads.html>

After the installation is done you have to download Apache Tomcat, which you can find on the homepage of the software. Apache Tomcat version 9 was used in this seminar paper:

- **Apache Tomcat:** <https://tomcat.apache.org/download-90.cgi>

On the website you can find various files for download, including a 64-bit Windows zip file, which is required. (See Figure 16)

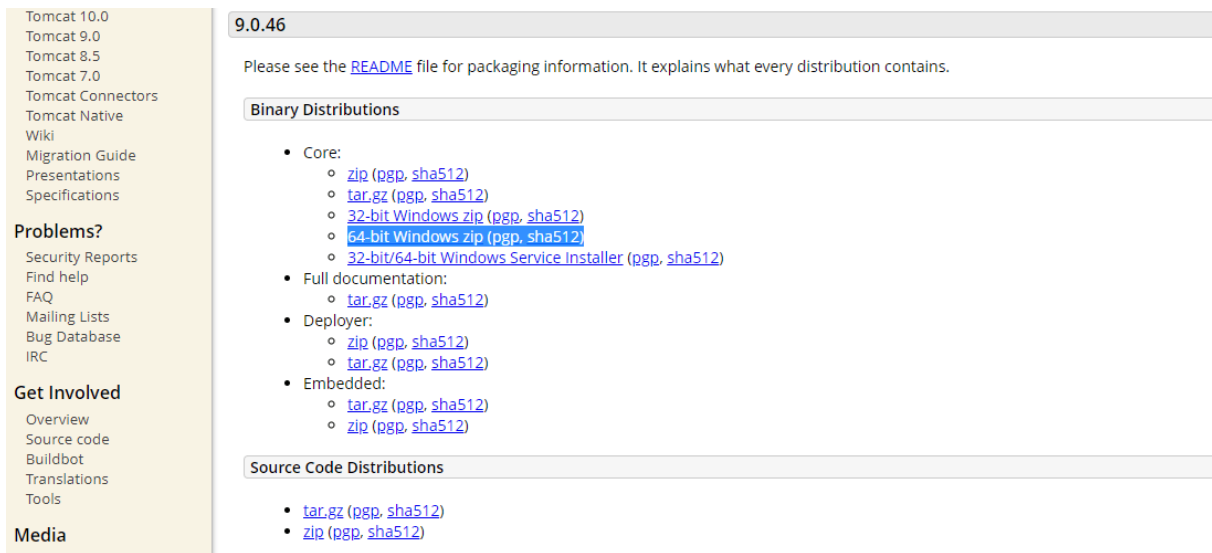


Figure 16: Apache Tomcat Download

## Step 2: Create directories and files

After the required software has been downloaded and installed, a new folder must be created in the Windows (C:) directory. The folder can be named individually, in this seminar work it is called ApacheTomcatProjekt. (see Figure 17)

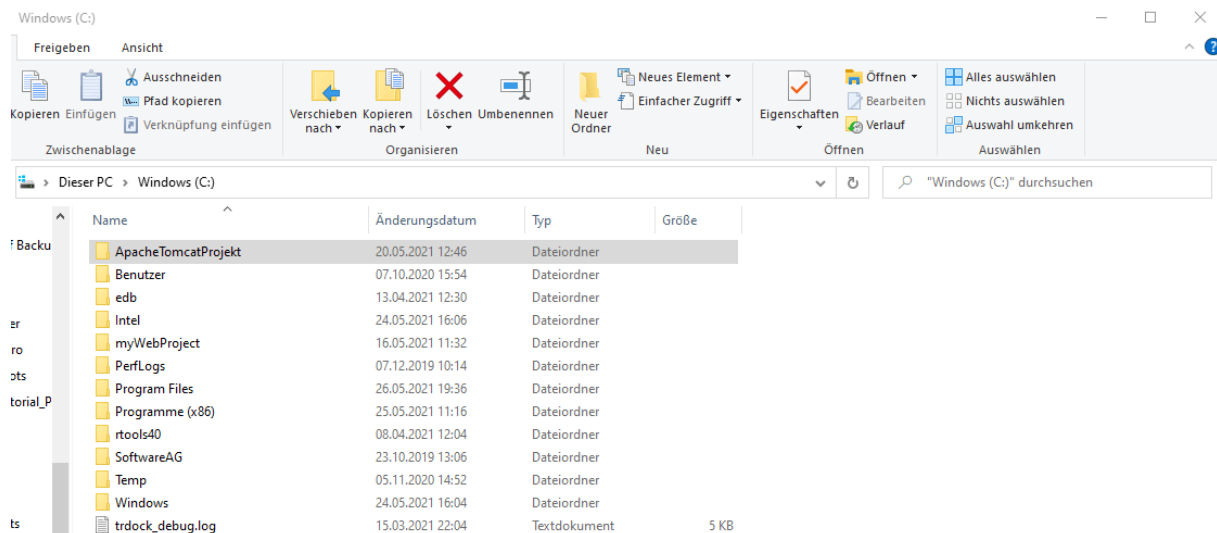


Figure 17: ApacheTomcatProjekt directory in Windows (C:)

Now the previously downloaded ZIP file containing Apache Tomcat must be unpacked in this folder. In order to unpack the file, you need another software such as WinRAR. However, there are also various other programmes with which ZIP files can be unpacked.

- **WinRAR download:** <https://winrar.de/downld.php>

After the ZIP file has been successfully unpacked in the folder, the order should look like this: (see Figure 18)

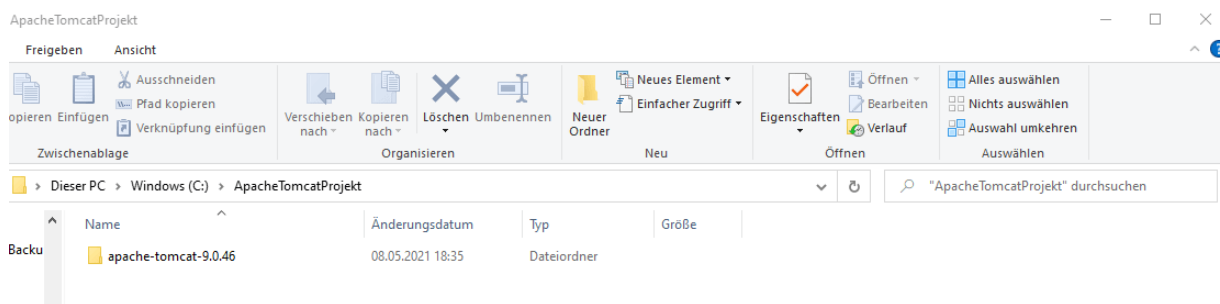


Figure 18: Apache Tomcat unpacked

This folder can also be named individually. In this example, the original name is retained. If the folder has been unzipped correctly, it should contain several other folders and files. One of these folders is called webapps. (see Figure 19)

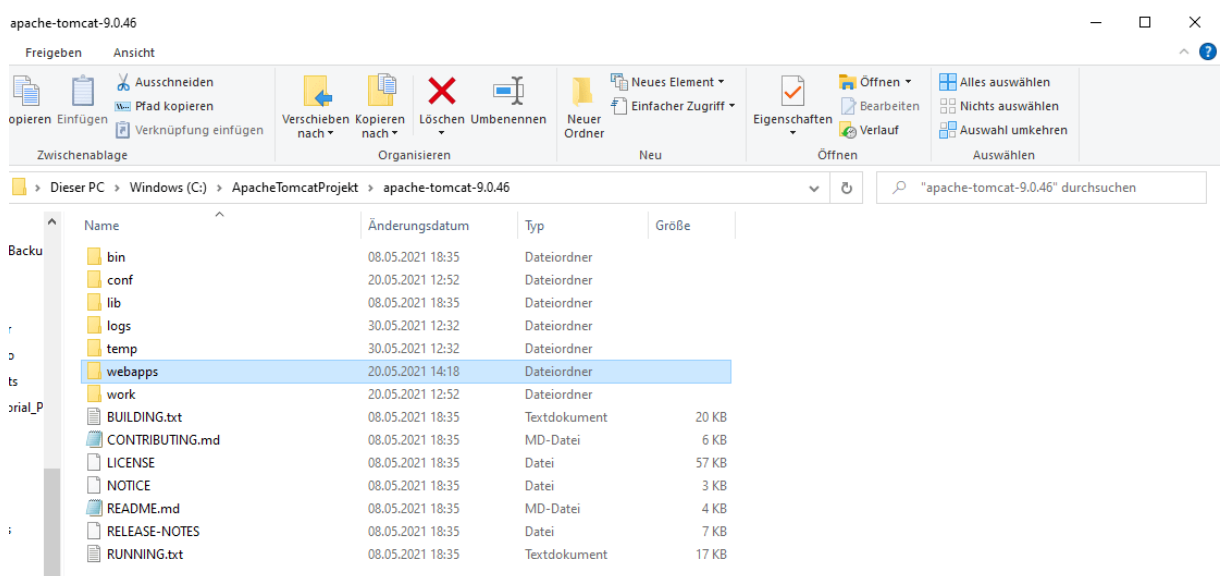


Figure 19: webapps directory

Now create a new folder in the webapps folder. The name of this folder can again be freely chosen, here it is called Jython Template. (see Figure 20)

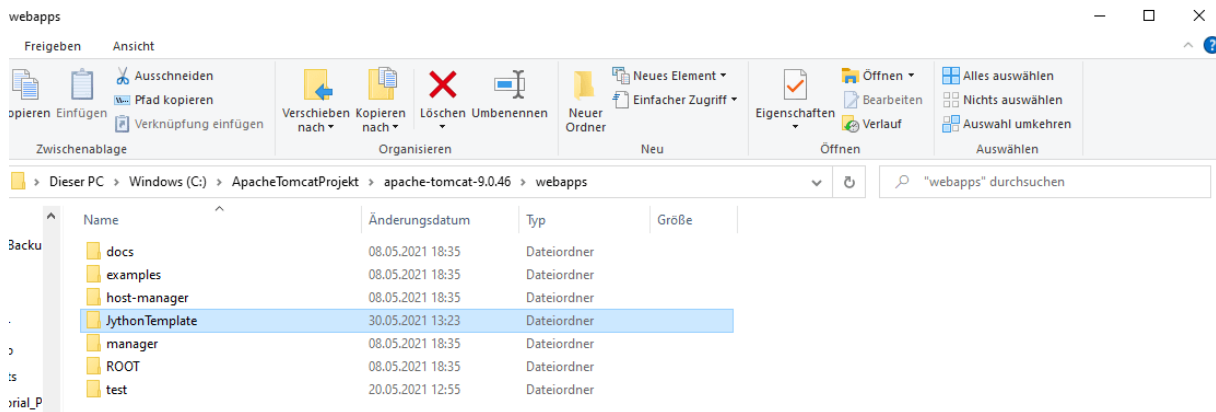


Figure 20: JythonTemplate directory

Now two more folders have to be created in JythonTemplate, WEB-INF and META-INF. It is very important to use exactly these names. (see Figure 21)

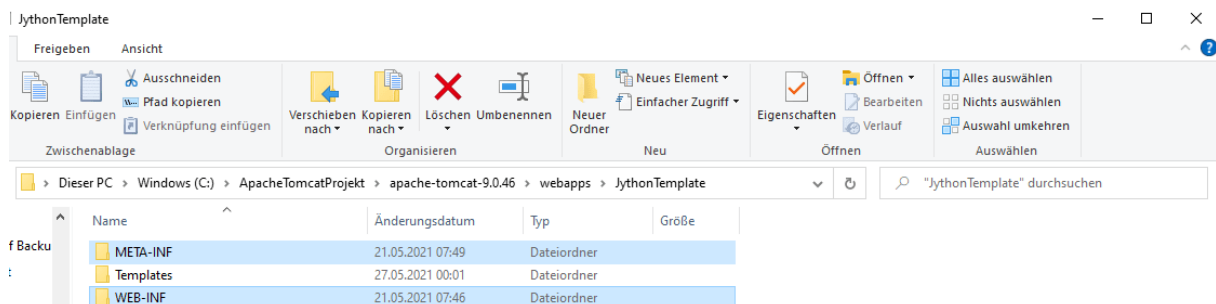


Figure 21: WEB-INF and META-INF directory

Another folder is now created in the WEB-INF folder with the name lib. Here, too, the exact naming is important. (see Figure 22)

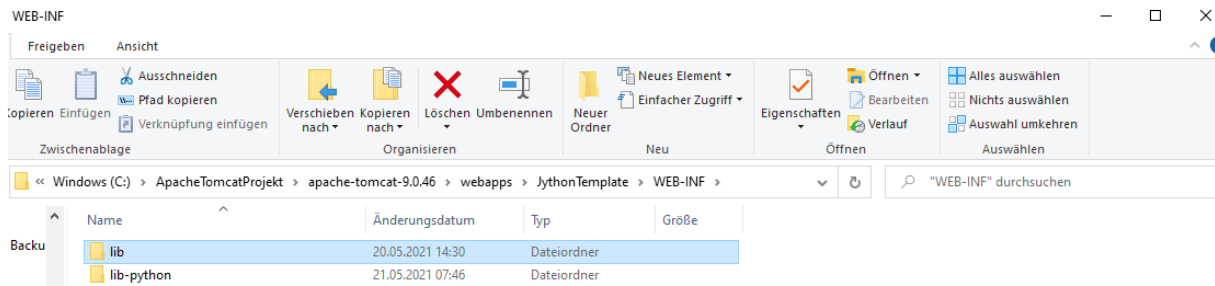


Figure 22: lib directory

The next step is to download another important piece of software, Jython. On the website of Jython you can find the required file.

- **Jython download:** <https://www.jython.org/download.html>

Under this link you can find several versions of Jython. Choose the Jython standalone file of the latest version, jython-standalone-2.7.2.jar. After the download is complete, copy this file into the WEB-INF/lib folder. (see Figure 23)

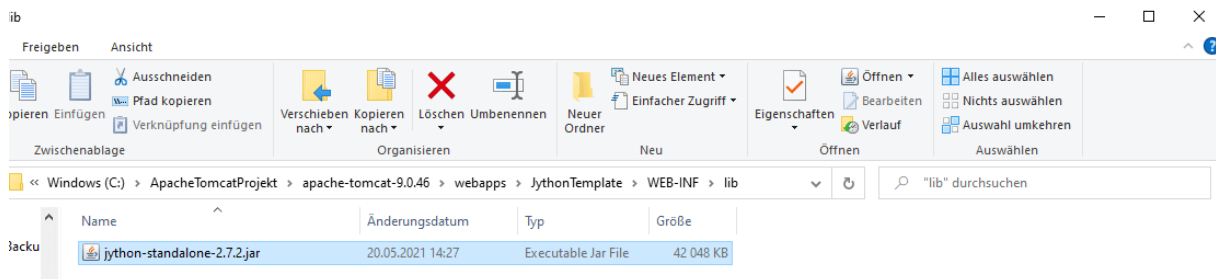
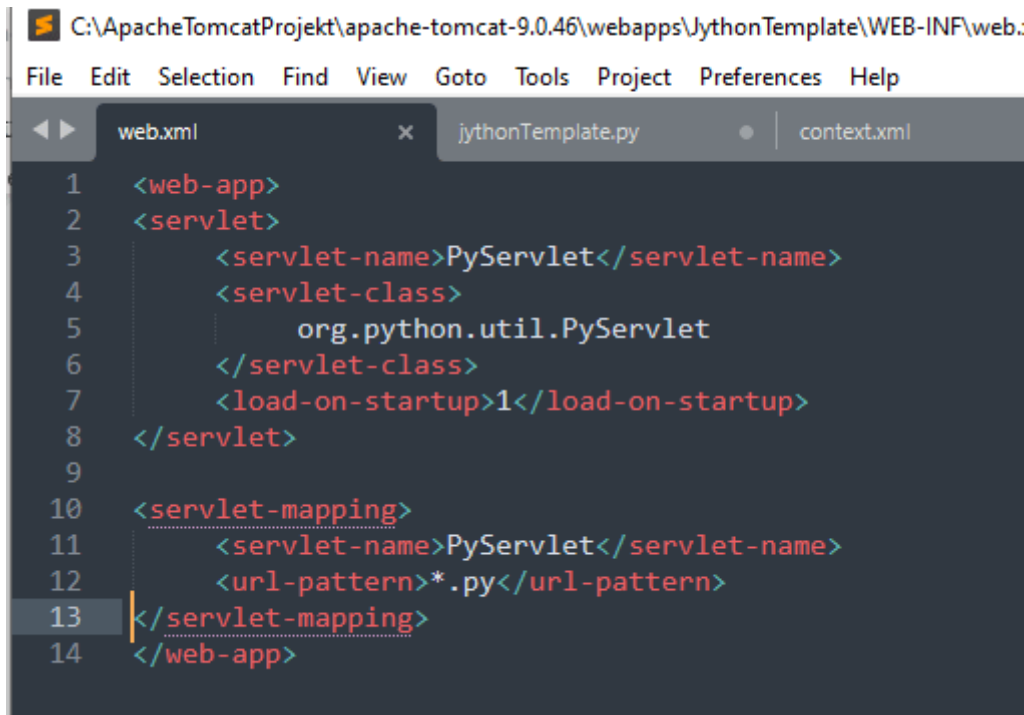


Figure 23: jython-standalone-2.7.2.jar

This library is now loaded into memory when Apache Tomcat is started.

The Jython servlets work with the help of an intermediate Java servlet called PyServlet and this is the servlet in which Tomcat runs Jython servlets. In the following, Tomcat needs to know that it should run the PyServlet when it receives a request for a resource with the extension \*.py. We achieve this with the help of the web.xml file. To do this, you must again access the WEB-INF folder, in which a file with the name web.xml is inserted, which contains the following content. (see Figure 24) You can work with any programme that can edit \*.xml files. One recommendation is the programme Sublime Text, which can be found under this link.

- **Sublime text download:** <https://www.sublimetext.com/download>



```

1  <web-app>
2  <servlet>
3      <servlet-name>PyServlet</servlet-name>
4      <servlet-class>
5          org.python.util.PyServlet
6      </servlet-class>
7      <load-on-startup>1</load-on-startup>
8  </servlet>
9
10 <servlet-mapping>
11     <servlet-name>PyServlet</servlet-name>
12     <url-pattern>*.py</url-pattern>
13 </servlet-mapping>
14 </web-app>
  
```

Figure 24: web.xml content

Once the required file has been successfully created, the directory WEB-INF should look like this. (see Figure 25)

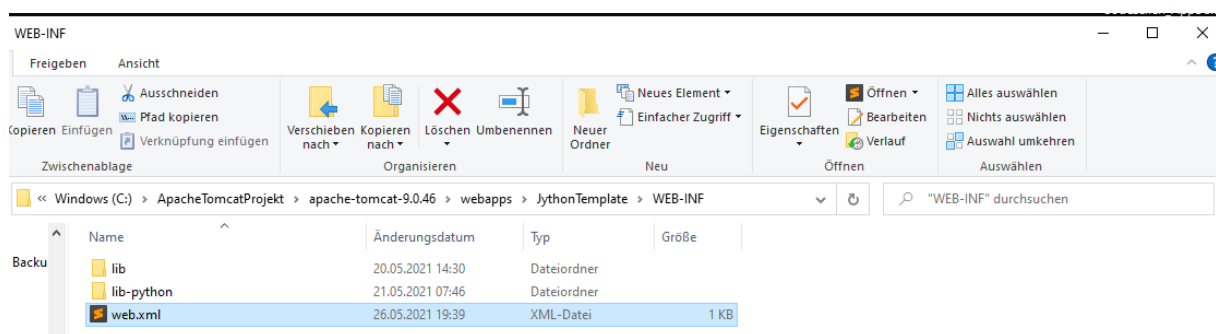
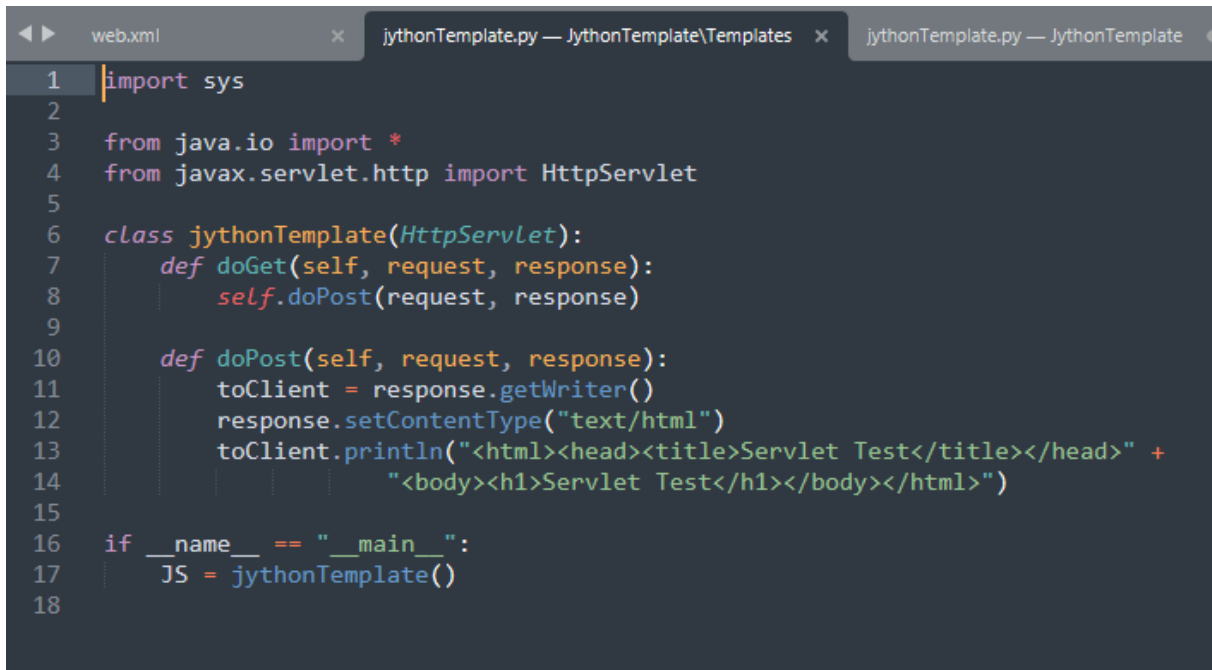


Figure 25: WEB-INF directory with web.xml file

### Step 3: Test the server

The next step is to test whether everything works as it should. To do this, we have to create a servlet with which we can test whether Apache Tomcat recognises everything. To do this, a

Python file must be created that can be named individually, in this example it is called `jythonTemplate.py`, as it is to serve as a template for further servlets. It must contain the following content. (see Figure 26)



```

1 import sys
2
3 from java.io import *
4 from javax.servlet.http import HttpServlet
5
6 class jythonTemplate(HttpServlet):
7     def doGet(self, request, response):
8         self.doPost(request, response)
9
10    def doPost(self, request, response):
11        toClient = response.getWriter()
12        response.setContentType("text/html")
13        toClient.println("<html><head><title>Servlet Test</title></head>" +
14                        "<body><h1>Servlet Test</h1></body></html>")
15
16 if __name__ == "__main__":
17     JS = jythonTemplate()
18
  
```

Figure 26: `jythonTemplate.py` content

The file `jythonTemplate.py` must now be inserted into the folder `jythonTemplate`. For a good overview, another folder called `Templates` was created in the `JythonTemplate` folder in this example, in which the Python file can be found. However, this is not necessary because you can simply leave the file in the `JythonTemplate` folder. (see Figure 27)

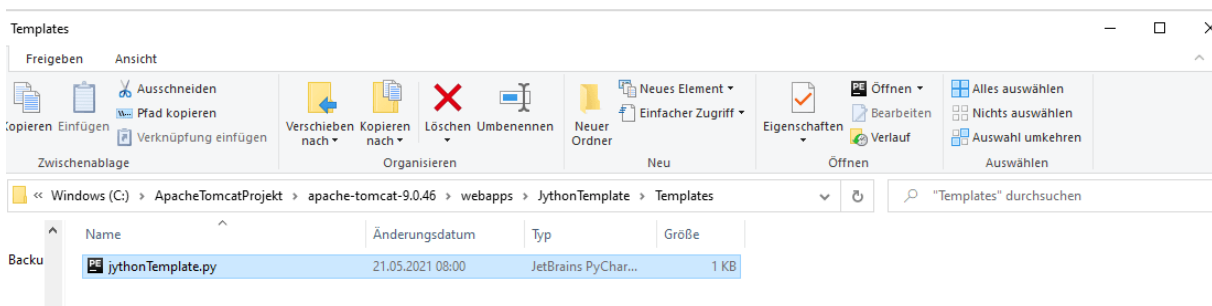
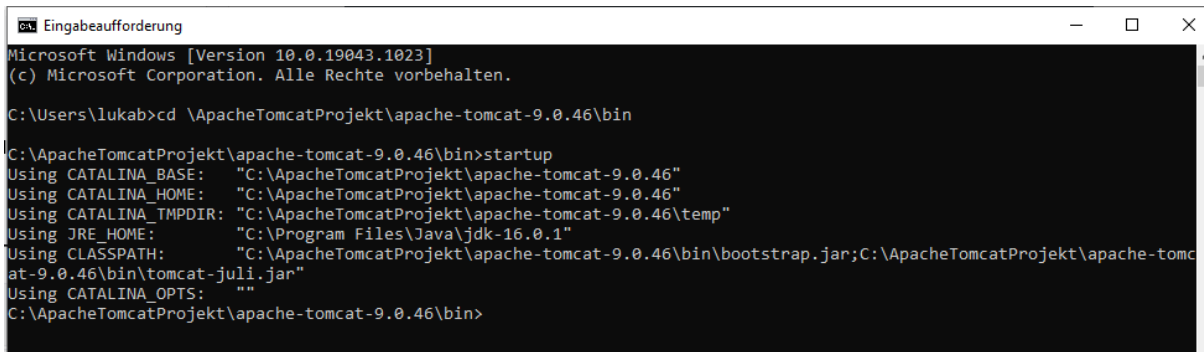


Figure 27: `jythonTemplate.py` path



Now you can start the Tomcat server and test the functionality. To do this, open the CMD shell and navigate to the required folder: \ApacheTomcatProjekt\apache-tomcat-9.0.46\bin. Then execute the file startup. (see Figure 28)



```

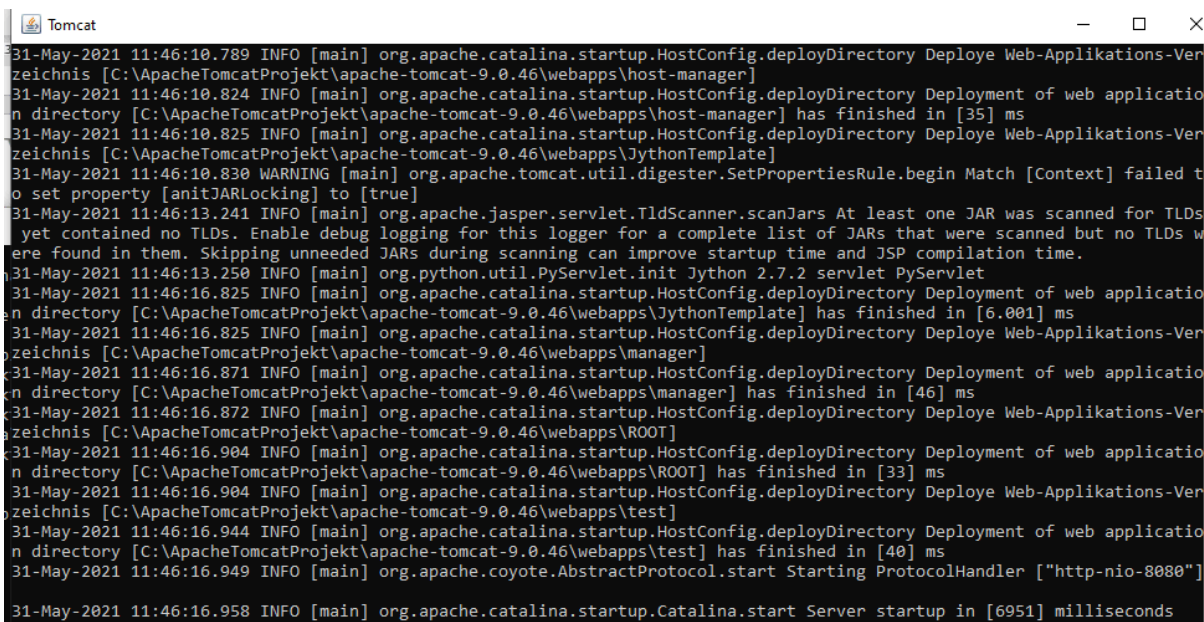
Eingabeaufforderung
Microsoft Windows [Version 10.0.19043.1023]
(c) Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\lukab>cd \ApacheTomcatProjekt\apache-tomcat-9.0.46\bin

C:\ApacheTomcatProjekt\apache-tomcat-9.0.46\bin>startup
Using CATALINA_BASE:   "C:\ApacheTomcatProjekt\apache-tomcat-9.0.46"
Using CATALINA_HOME:   "C:\ApacheTomcatProjekt\apache-tomcat-9.0.46"
Using CATALINA_TMPDIR: "C:\ApacheTomcatProjekt\apache-tomcat-9.0.46\temp"
Using JRE_HOME:        "C:\Program Files\Java\jdk-16.0.1"
Using CLASSPATH:       "C:\ApacheTomcatProjekt\apache-tomcat-9.0.46\bin\bootstrap.jar;C:\ApacheTomcatProjekt\apache-tomcat-9.0.46\bin\tomcat-juli.jar"
Using CATALINA_OPTS:   ""
C:\ApacheTomcatProjekt\apache-tomcat-9.0.46\bin>
  
```

Figure 28: Apache Tomcat Server startup

A new Java window opens in which the Web Server is started. (see Figure 29)



```

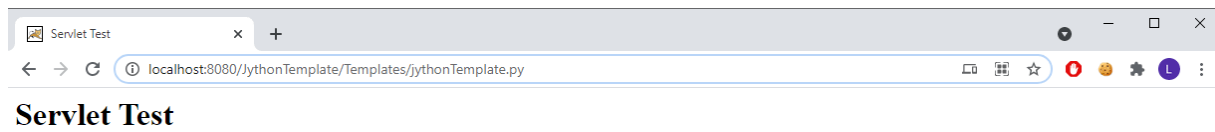
Tomcat
31-May-2021 11:46:10.789 INFO [main] org.apache.catalina.startup.HostConfig.deployDirectory Deploye Web-Applikations-Verzeichniss [C:\ApacheTomcatProjekt\apache-tomcat-9.0.46\webapps\host-manager]
31-May-2021 11:46:10.824 INFO [main] org.apache.catalina.startup.HostConfig.deployDirectory Deployment of web application directory [C:\ApacheTomcatProjekt\apache-tomcat-9.0.46\webapps\host-manager] has finished in [35] ms
31-May-2021 11:46:10.825 INFO [main] org.apache.catalina.startup.HostConfig.deployDirectory Deploye Web-Applikations-Verzeichniss [C:\ApacheTomcatProjekt\apache-tomcat-9.0.46\webapps\JythonTemplate]
31-May-2021 11:46:10.830 WARNING [main] org.apache.tomcat.util.digester.SetPropertiesRule.begin Match [Context] failed to set property [anitJARLocking] to [true]
31-May-2021 11:46:13.241 INFO [main] org.apache.jasper.servlet.TldScanner.scanJars At least one JAR was scanned for TLDs yet contained no TLDs. Enable debug logging for this logger for a complete list of JARs that were scanned but no TLDs were found in them. Skipping unneeded JARs during scanning can improve startup time and JSP compilation time.
31-May-2021 11:46:13.250 INFO [main] org.python.util.PyServlet.init Jython 2.7.2 servlet PyServlet
31-May-2021 11:46:16.825 INFO [main] org.apache.catalina.startup.HostConfig.deployDirectory Deployment of web application directory [C:\ApacheTomcatProjekt\apache-tomcat-9.0.46\webapps\JythonTemplate] has finished in [6.001] ms
31-May-2021 11:46:16.825 INFO [main] org.apache.catalina.startup.HostConfig.deployDirectory Deploye Web-Applikations-Verzeichniss [C:\ApacheTomcatProjekt\apache-tomcat-9.0.46\webapps\manager]
31-May-2021 11:46:16.871 INFO [main] org.apache.catalina.startup.HostConfig.deployDirectory Deployment of web application directory [C:\ApacheTomcatProjekt\apache-tomcat-9.0.46\webapps\manager] has finished in [46] ms
31-May-2021 11:46:16.872 INFO [main] org.apache.catalina.startup.HostConfig.deployDirectory Deploye Web-Applikations-Verzeichniss [C:\ApacheTomcatProjekt\apache-tomcat-9.0.46\webapps\ROOT]
31-May-2021 11:46:16.904 INFO [main] org.apache.catalina.startup.HostConfig.deployDirectory Deployment of web application directory [C:\ApacheTomcatProjekt\apache-tomcat-9.0.46\webapps\ROOT] has finished in [33] ms
31-May-2021 11:46:16.904 INFO [main] org.apache.catalina.startup.HostConfig.deployDirectory Deploye Web-Applikations-Verzeichniss [C:\ApacheTomcatProjekt\apache-tomcat-9.0.46\webapps\test]
31-May-2021 11:46:16.944 INFO [main] org.apache.catalina.startup.HostConfig.deployDirectory Deployment of web application directory [C:\ApacheTomcatProjekt\apache-tomcat-9.0.46\webapps\test] has finished in [40] ms
31-May-2021 11:46:16.949 INFO [main] org.apache.coyote.AbstractProtocol.start Starting ProtocolHandler ["http-nio-8080"]
31-May-2021 11:46:16.958 INFO [main] org.apache.catalina.startup.Catalina.start Server startup in [6951] milliseconds
  
```

Figure 29: Java Window for Server start

After these steps, the server is online and you can access it. To do this, go to the browser of your choice and open the following link:

- **Tomcat Server:** <http://localhost:8080/JythonTemplate/Templates/jythonTemplate.py>

If the output of the website now looks like this (see Figure 30), everything runs smoothly and you can start to create your own servlets in Python.



*Figure 30: Tomcat Servlet Test*

To shut down the server, open the Java window in which the server was started up and press the key combination Ctrl+c.