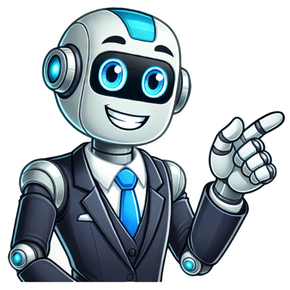
Continue



Jinja is a fast, expressive, extensible templating engine. Special placeholders in the template allow writing code similar to Python syntax. Then the template is passed data to render the final document. It includes: Template inheritance and inclusion. Define and import macros within templates. HTML templates can use autoescaping to prevent XSS from untrusted user input. A sandboxed environment can safely render untrusted templates. AsyncIO support for generating templates and calling async functions. I18N support with Babel. Templates are compiled to optimized Python code just-in-time and cached, or can be compiled ahead-of-time. Exceptions point to the correct line in templates to make debugging easier. Extensible filters, tests, functions, and even syntax. Jinja's philosophy is that while application logic belongs in Python if possible, it shouldn't make the template designer's job difficult by restricting functionality too much. {% extends "base.html" %} {% block title %}Members{% endblock %} {% block content %} {% for user in users %} {{ user.username }} {% endfor %} {% endblock %} Jinja is a fast, expressive, extensible templating engine. Special placeholders in the template allow writing code similar to Python syntax. Then the template is passed data to render the final document. It includes: Template inheritance and inclusion. Define and import macros within templates. HTML templates can use autoescaping to prevent XSS from untrusted user input. A sandboxed environment can safely render untrusted templates. AsyncIO support for generating templates and calling async functions. I18N support with Babel. Templates are compiled to optimized Python code just-in-time and cached, or can be compiled ahead-of-time. Exceptions point to the correct line in templates to make debugging easier. Extensible filters, tests, functions, and even syntax. Jinja's philosophy is that while application logic belongs in Python if possible, it shouldn't make the template designer's job difficult by restricting functionality too much. {% extends "base.html" %} {% block title %}Members{% endblock %} {% block content %} {% for user in users %} {{ user.username }} {% endfor %} {% endblock %} The Pallets organization develops and supports Jinja and other popular packages. In order to grow the community of contributors and users, and allow the maintainers to devote more time to the projects, please donate today. See our detailed contributing documentation for many ways to contribute, including reporting issues, requesting features, asking or answering questions, and making PRs. If you don’t know some of the other frequently used terms in data science. Then click here.It is the degree of distortion from the symmetrical bell curve or the normal distribution. It measures the lack of symmetry in data distribution.It differentiates extreme values in one versus the other tail. A symmetrical distribution will have a skewness of 0.There are two types of Skewness: Positive and NegativePositive Skewness means when the tail on the right side of the distribution is longer or fatter. The mean and median will be greater than the mode.Negative Skewness is when the tail of the left side of the distribution is longer or fatter than the tail on the right side. The mean and median will be less than the mode.So, when is the skewness too much?The rule of thumb seems to be:If the skewness is between -0.5 and 0.5, the data are fairly symmetrical.If the skewness is between -1 and -0.5(negatively skewed) or between 0.5 and 1(positively skewed), the data are moderately skewed.If the skewness is less than -1(negatively skewed) or greater than 1(positively skewed), the data are highly skewed.ExampleLet us take a very common example of house prices. Suppose we have house values ranging from $100k to $1,000,000 with the average being $500,000.If the peak of the distribution was left of the average value, portraying a positive skewness in the distribution. It would mean that many houses were being sold for less than the average value, i.e. $500k. This could be for many reasons, but we are not going to interpret those reasons here.If the peak of the distributed data was right of the average value, that would mean a negative skew. This would mean that the houses were being sold for more than the average value.KurtosisKurtosis is all about the tails of the distribution — not the peakedness or flatness. It is used to describe the extreme values in one versus the other tail. It is actually the measure of outliers present in the distribution.High kurtosis in a data set is an indicator that data has heavy tails or outliers. If there is a high kurtosis, then, we need to investigate why do we have so many outliers. It indicates a lot of things, maybe wrong data entry or other things. Investigate!Low kurtosis in a data set is an indicator that data has light tails or lack of outliers. If we get low kurtosis(too good to be true), then also we need to investigate and trim the dataset of unwanted results.Mesokurtic: This distribution has kurtosis statistic similar to that of the normal distribution. It means that the extreme values of the distribution are similar to that of a normal distribution characteristic. This definition is used so that the standard normal distribution has a kurtosis of three.Leptokurtic (Kurtosis > 3): Distribution is longer, tails are fatter. Peak is higher and sharper than Mesokurtic, which means that data are heavy-tailed or profusion of outliers. Outliers stretch the horizontal axis of the histogram graph, which makes the bulk of the data appear in a narrow (“skinny”) vertical range, thereby giving the “skinniness” of a leptokurtic distribution.Platykurtic: (Kurtosis < 3): Distribution is shorter, tails are thinner than the normal distribution. The peak is lower and broader than Mesokurtic, which means that data are light-tailed or lack of outliers.The reason for this is because the extreme values are less than that of the normal distribution.If you want to get an Introduction to Machine Learning, click here. Jinja2 is a template engine for Python. You can use it when rendering data to web pages. For every link you visit, you want to show the data with the formatting. By using a template engine we can seperate display logic (html, css) from the actual Python code. Let’s start with an example Related coursePython Flask: Make Web Apps with Python Create the directories:And create the file user.html in /app/templates: Then create the code app.py in /app/app.py: from flask import Flask, flash, redirect, render_template, requestfrom random import randintapp = Flask(__name__)def index():return "Flask App!"def hello():users = ["Frank", "Steve", "Alice", "Bruce"]return render_template('user.html', **locals())if __name__ == "__main__":app.run(host='0.0.0.0', port=8080) Finally execute with: python app.py* Running on You can then open in your browser. This will output the data formatted according to html: > Expressions to print to the template output Comments not included in the template output # … ## Line Statements In the example above we have two statements and one expression. We have not included any comments. Base template and child templatesA Jinja2 template can extend a base template. On a webpage with many sites you may want these pages look similar. In /templates/ create a file called base.html with this code: - My Webpage Copyright 2015 by ", line 1, in top-level template code jinja2.exceptions.UndefinedError: 'type' is undefined With strict error checking we get error straight away. It's worth noting that Ansible uses StrictUndefined by default so when you us it to render Jinja template you'll get errors whenever there's a reference to an undefined variable. Overall I suggest always enabling undefined type to StrictUndefined. If you don't do it you can get some very subtle bugs in your templates that will be difficult to find. At first it might be easy to understand why a value is missing in the output but over time, with templates growing larger, it's difficult for human eye to notice something is not right. You definitely don't want to be only realizing your templates are broken when loading config onto your devices! Before we wrap this post up I just wanted to show you how to include comments in your templates. Generally templates should be self explanatory but comments come in handy when multiple people work on the same template and your future self will probably be grateful for explaining non-obvious bits as well. You can also use comment syntax to disable parts of the template during debugging. Comments are added using {# ... #} syntax. That is anything between {# and #} is treated as a comment and will be ignored by the engine. Below is our first example with some comments added to it: from jinja2 import Template template = """hostname {{ hostname }} {# DNS configuration -#} no ip domain lookup ip domain name local.lab ip name-server {{ name_server_pri }} ip name-server {{ name_server_sec }} {# Time servers config, we should use pool.ntp.org -#} ntp server {{ ntp_server_pri }} prefer ntp server {{ ntp_server_sec }} ntp server {{ ntp_server_trd }}""" data = { "hostname": "core-sw-waw-01", "name_server_pri": "1.1.1.1", "name_server_sec": "8.8.8.8", "ntp_server_pri": "0.pool.ntp.org", "ntp_server_sec": "1.pool.ntp.org", } j2_template = Template(template) print(j2_template.render(data)) And the output: hostname core-sw-waw-01 no ip domain lookup ip domain name local.lab ip name-server 1.1.1.1 ip name-server 8.8.8.8 ntp server 0.pool.ntp.org prefer ntp server 1.pool.ntp.org ntp server No sign of comments. Though you might have noticed - (dash) character just before closing #}. Without that dash an extra empty line would have been added after each comment. Whitespace treatment in Jinja is not very intuitive and it's possibly one of the most confusing parts of the language. In one of the future posts I will discuss in detail different scenarios and techniques that will allow you to make your templates look exactly the way you want it. Conclusion This concludes first post in the Jinja Tutorial series. What I showed you here should be enough to get you to start creating your own templates. In the future posts we'll have a look at other features and we'll work through examples illustrating use cases. Stay tuned! References Jinja2 is a designer-friendly, secure, and fast templating engine for Python. If you frequently work with web frameworks like Flask and Django, you will love it. Jinja2 lets you embed Python expressions, conditionals, and loops inside templates. This helps you build scalable and maintainable applications. Today at PythonCentral, let us explain the basics, its syntax, and some real-world examples to help you integrate it with your Python projects. Get. Set. Learn! How to Install Jinja Let us take one of favourite tool Pip, to install Jinja2. Execute this command to install: pip install Jinja2 Basics of Jinja2 Templating Placeholders, control structures, and filters: these are the building blocks of templates. These allow dynamic content rendering. Sample Syntax To understand templating better, let us take up a basic Jinja2 syntax and then look at its components. Here is one: {{ title }} Welcome, {{ user }}! If you look closely, you will notice: {{ title }}: Placeholder for dynamic content. {{ user }}: Injects a variable into the template. How to Render Templates in Python with Jinja2 One of the most common applications of Jinja2 is rendering templates dynamically in Python. Here is a sample syntax to understand better: from jinja2 import Template template = Template("Hello, {{ name }}!") message = template.render(name="PythonCentral") print(message) # Output: Hello, PythonCentral! How to Use Loops and Conditionals Jinja2 supports loops and conditionals for flexible template rendering. Let us take a few examples to understand them better. Looping Through Data Here is a syntax that uses looping through data: {% for item in items %} {{ item }} {% endfor %} Using Conditionals Here is a syntax that uses conditionals: {% if user.is_admin %} Welcome, Admin! {% else %} Welcome, User! {% endif %} Template Inheritance for Code Reusability With Jinja 2 you can inherit from a base template, reducing code duplication. Let us take a base template and then a child template now. Here is a sample base template: {% block title %}Default Title{% endblock %} {% block header %}Header Content{% endblock %} {% block content %}Main Content{% endblock %} And here is the HTML code for a sample child template: {% extends "base.html" %} {% block title %}My Page{% endblock %} {% block content %} This is my custom content. {% endblock %} Using Filters for Data Formatting Jinja2 provides filters to modify variables inside templates. Let us look at some commonly used filters now: {{ name|upper }}: Converts text to uppercase {{ price|round(2) }}: Rounds a number to 2 decimal places {{ users|length }}: Returns the number of elements in a list Let us try to use these filters now. Total Users: {{ users|length }} Price: ${{ amount|round(2) }} Using Jinja2 in Flask Applications Jinja2 is the default templating engine for Flask. To integrate it into a Flask app, you can spin up your own script similar to this: from flask import Flask, render_template app = Flask(__name__) @app.route("/") def home(): return render_template("index.html", title="Home Page", user="PythonCentral") if __name__ == "__main__": app.run(debug=True) In this script, render_template("index.html"): Dynamically injects data into templates. Variables like "title" and "user" are passed to the template. Best Practices for Using Jinja2 To prevent "cross-site scripting (XSS)" and other vulnerabilities, follow these best practices: Always enable autoescaping: Jinja2 escapes HTML content by default. Use safe filter cautiously: Allows rendering raw HTML (we have provided an example just after this section). Avoid executing user-input code: Do not use "eval()" in templates. Here is an example of unsafe practice: {{ user_input|safe }} {# This can be dangerous! #} Wrapping Up Jinja2 is an essential templating engine for Python, offering powerful features like loops, conditionals, template inheritance, and filters. Whether you are using it in Flask, Django, or standalone applications, it helps generate dynamic and scalable content efficiently. Learning Jinja2 will enhance your ability to build dynamic, efficient, and maintainable Python applications with ease. Here are some more articles you might be interested in. Related Articles