

# Crafting Your Custom Logger in Python: A Step-by-Step Guide



Emanuele · [Follow](#)

3 min read · Jan 9, 2024



63



Python's built-in logging module is a powerful tool for tracking and understanding the flow of your application. However, there are situations where a custom logger tailored to your specific needs can add significant value. In this article, we'll explore the process of creating a custom logger in Python, providing you with more control and flexibility over your application's logging mechanism.

## Why a Custom Logger?

While Python's default logging capabilities cover a wide range of scenarios, a custom logger allows you to fine-tune logging behavior to meet your project's unique requirements. Whether you need to format log messages in a specific way, filter messages based on criteria, or integrate with an external logging service, a custom logger can be your go-to solution.

This one is a basic version where I am overriding a method. Basic concepts of object-oriented programming. I can use the same behavior of info from the module library and add some functionalities. I am setting the value extra in the new Logger class, so I don't need to pass every time.

We can override in the same way the methods debug, error, exception, etc.

```
import logging

class Logger(logging.Logger):
    def __init__(self, name, level=logging.NOTSET):
        super().__init__(name, level)
        self.extra_info = None

    def info(self, msg, *args, xtra=None, **kwargs):
        extra_info = xtra if xtra is not None else self.extra_info
        super().info(msg, *args, extra=extra_info, **kwargs)
```

Storing runtime information within the Logger class proves beneficial, eliminating the need to pass it explicitly each time the logger is utilized.

## Add Databricks information to the logger

If we use this logger in Databricks we can add in this way information related to the single runtime:

```
import logging
from datetime import datetime

class Logger(logging.Logger):
    def __init__(self, name, level=logging.NOTSET):
        super().__init__(name, level)
        self.extra_info = None
        self.dbutils_info()

    def info(self, msg, *args, xtra=None, **kwargs):
        extra_info = xtra if xtra is not None else self.extra_info
        super().info(msg, *args, extra=extra_info, **kwargs)

    def dbutils_info(self):
        # Get the notebook path
        self.notebook_path = dbutils.notebook.entry_point.getDbutils().notebook().
        # Extract the notebook name from the path
        self.notebook_name = self.notebook_path.split("/")[-1]
        self.notebook_id = json.loads(dbutils.notebook.entry_point.getDbutils().no

        self.extra_exc = {
            'notebook_name': self.notebook_name,
```

```
        'notebook_id': self.notebook_id
    }
```

## Add telemetry to our logger: opencensus and Azure

Another interesting use is additional functionality related to distributed tracing and integrates with Azure Application Insights for logging and tracing purposes. This code utilizes the OpenCensus library for tracing and Azure-specific exporters for logging. This method will be deprecated in September 2024, more info [here](#).

```
from opencensus.trace.tracer import Tracer
from opencensus.ext.azure.log_exporter import AzureLogHandler
from opencensus.ext.azure.trace_exporter import AzureExporter
from opencensus.trace.samplers import ProbabilitySampler
import logging

class Logger(logging.Logger):
    def __init__(self, name, level=logging.NOTSET):
        super().__init__(name, level)
        self.extra_info = None

    # Add handlers (e.g., ConsoleHandler, FileHandler, etc.)
    handler = logging.StreamHandler()
    formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(message)s')
    handler.setFormatter(formatter)
    self.addHandler(handler)
```

```

self.addHandler(
    AzureLogHandler(
        connection_string=f'InstrumentationKey={appinsight_key}'
    )
)

self.tracer = Tracer(
    exporter=AzureExporter(
        connection_string=f'InstrumentationKey={appinsight_key};IngestionEndpo
        sampler=ProbabilitySampler(1.0),
    )

def info(self, msg, *args, xtra=None, **kwargs):
    extra_info = xtra if xtra is not None else self.extra_info
    super().info(msg, *args, extra=extra_info, **kwargs)

```

## Usage example

```

#Instantiate the class
logger = Logger('logger_name',appinsight_key, level=logging.DEBUG) # level=loggi

logger.info("This is an info message.") # 2024-01-09 02:02:03,042 - INFO - This
logger.error("This is an error message.") # 2024-01-09 02:02:03,042 - ERROR - Th
logger.exception("This is an exception message.")
logger.debug("This is an exception message.") # you need to set up level=logging

```

## Further Customizations

Feel free to extend and adapt your custom logger based on your project's needs. You can explore features like custom filters, and log levels, or even integrate with third-party logging services.

You could add a method in the error method to raise a message to your slack or Teams channel

```
def error(self, msg):  
    super().error(msg)  
    self.slack.send(msg)
```

## Conclusion

Creating a custom logger in Python provides you with the flexibility to tailor your logging experience to match your project's requirements. By following the steps outlined in this guide, you can easily craft a logger that suits your needs, enhancing the overall manageability and understanding of your application's behavior.

Happy logging! 📖🐍

Python

Logging

Azure

Opencensus

Databricks



## Written by Emanuele

13 Followers

LA-based robotics engineer specializing in Azure technologies. Passionate about system design, AI, and entrepreneurship, dedicated to driving tech innovation

Follow



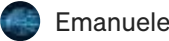
---

More from Emanuele

```
7
8 app = FastAPI()
9
10 @app.get("/")
11 async def root():
12     logger.debug('this is a debug message')
13     return {"message": "Hello World"}
14
15 @app.post("/")
16 async def post():
17     return {"message": "Hello from the post route"}
18
19
20 @app.get("/deprecated", description="This is the root route", deprecated=True)
21 async def base_get_route():
22     return {"message": "Hello World"}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS PROMPT FLOW AZURE

(env) C:\workspace\python\fastapi\_login\udicorn-main>app --log-level debug

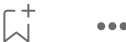


Emanuele

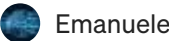
## Print log messages in the terminal using Python FastAPI and uvicorn

Short article, just to share a common issue when someone starts to work in Python...

Apr 5 🖱️ 10



initialize job	<1s
Checkout LearningAssis...	5s
Print Pipeline Informati...	3s
Print Directory Structu...	<1s



Emanuele

## Azure DevOps Pipelines: Displaying Agent Environment...

1s

output

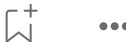


Emanuele

## Write a Databricks dataframe in just one file in a azure blob storage

Databricks provides a powerful platform for big data analytics and machine learning, an...

Jan 5 🖱️ 12



Put Item

FastAPI Item API Documentation

This API allows you to update an item.

PUT /items/{item\_id}

Updates an item.

Parameters

- `{item_id}` (path, required): The ID of the item to update. Must be an integer.
- `{body}` (body, required): The item to update. Must be a JSON object with the following properties:
  - `name` (string, required): The name of the item.
  - `description` (string, optional): The description of the item. If provided, must be a string of maximum 300 characters.
  - `price` (number, required): The price of the item. Must be a number greater than zero.
  - `tax` (number, optional): The tax of the item. If provided, must be a number.

Example

Request:

```
{
  "name": "item 1",
  "description": "This is item 1",
  "price": 9.99,
  "tax": 0.99
}
```

Response:

```
HTTP/1.1 200 OK Content-Type: application/json
{"item_id": 1, "name": "item 1", "description": "This is item 1", "price": 9.99, "tax": 0.99}
```

Request samples

Copy

Expand all

Collapse all

```
{
  "name": "item 1",
  "description": "This is item 1",
  "price": 9.99,
  "tax": 0.99
}
```

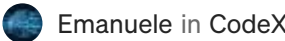
Response samples

Copy

Expand all

Collapse all

```
{
  "item_id": 1,
  "name": "item 1",
  "description": "This is item 1",
  "price": 9.99,
  "tax": 0.99
}
```



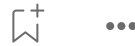
Emanuele in CodeX

## How to Document an API for Python FastAPI: Best Practices fo...



Azure DevOps Pipelines revolutionize the way software development teams build, test, an...

Mar 25 🖱️ 1



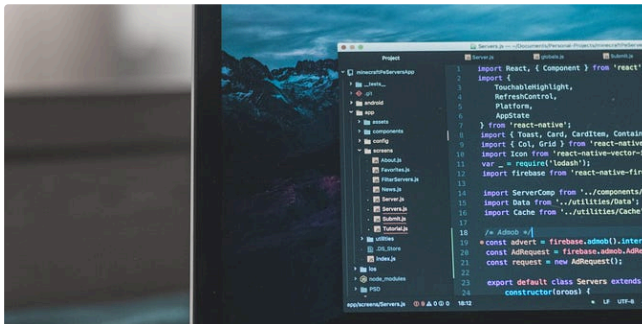
In the fast-paced world of software development, maintaining readable and...

May 21 🖱️ 13

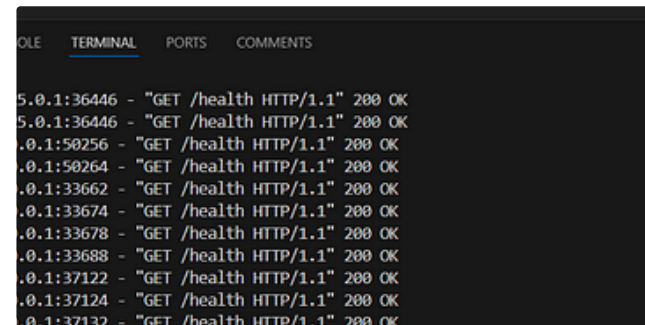


See all from Emanuele

## Recommended from Medium



Rahul Madhani in Azure Tutorials



Jegadeesh N

# A Deep Dive into Python Logging: Practical Examples for Developers

Master the Art from Basics to Advanced Techniques. Explore Configurations for...

Dec 27, 2023

👤 10

💬 1



# Implementing Health Checks and Auto-Restarts for FastAPI...

Deploying applications can often come with unexpected challenges, and our recent...

May 17

👤 6



## Lists



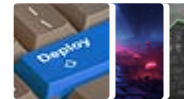
### Coding & Development

11 stories · 662 saves



### Practical Guides to Machine Learning

10 stories · 1572 saves



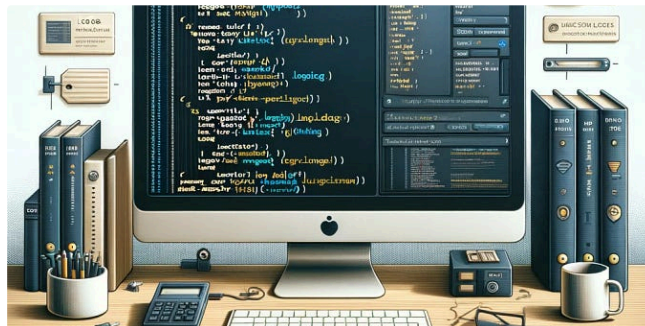
### Predictive Modeling w/ Python

20 stories · 1309 saves



### ChatGPT

21 stories · 686 saves



Software Development Engineer

Mar. 2020 – May 2021

- Developed Amazon checkout and payment services to handle traffic of 10 Million daily global transactions
- Integrated Iframes for credit cards and bank accounts to secure 80% of all consumer traffic and prevent CSRF, cross-site scripting, and cookie-jacking
- Led Your Transactions implementation for JavaScript front-end framework to showcase consumer transactions and reduce call center costs by \$25 Million
- Recovered Saudi Arabia checkout failure impacting 4000+ customers due to incorrect GET form redirection


## Projects

### NinjaPrep.io (React)

- Platform to offer coding problem practice with built in code editor and written + video solutions in React
- Utilized Nginx to reverse proxy IP address on Digital Ocean hosts
- Developed using Styled-Components for 95% CSS styling to ensure proper CSS scoping
- Implemented Docker with Seccomp to safely run user submitted code with < 2.2s runtime

### HeatMap (JavaScript)

- Visualized Google Takeout location data of location history using Google Maps API and Google Maps heatmap code with React
- Included local file system storage to reliably handle 5mb of location history data
- Implemented Express to include routing between pages and jQuery to parse Google Map and implement heatmap overlay

 Utkarsh Singh in Stackademic

## Mastering Python Logging: The Ultimate Guide

Logging is an indispensable part of Python programming, crucial for debugging,...

Mar 10  3




 Vijay Maurya

## Resolving SSLCertVerificationError:...

When working with APIs in Python, you may encounter the dreaded...

May 15  4  2



 Alexander Nguyen in Level Up Coding

## The resume that got a software engineer a \$300,000 job at Google.

1-page. Well-formatted.

★ Jun 1  7.7K  78



 Kunal in FanCode Engineering

## Open-source Log Aggregation by Loki

FanCode optimizes log aggregation with Loki and cloud services for better performance...

Feb 21  82  1



[See more recommendations](#)

---

[Help](#) [Status](#) [About](#) [Careers](#) [Press](#) [Blog](#) [Privacy](#) [Terms](#) [Text to speech](#) [Teams](#)