

Learning Python in the Age of AI

Part V

We made already the first step!

- In times of nicely working **generative AI**, the syntax learning becomes secondary
- We now know when to use python and what to avoid
 - Of course, we need the syntax, but getting concepts and best practices is way more important to create good code nowadays
 - **THIS DOES NOT MEAN THAT I ENCOURAGE YOU TO VIBE CODE!**
- Hence, **understanding** the language and its strengths and weaknesses is key
- In the next part, the possible learning paths are discussed

Very often, I forget parts of the syntax and nice tricks. But knowing them in combination of AI is a good basis to get things running fast.

Different Learning Paths

- The learning process can be boiled down to essentially three different learning paths:
 - The **Classic**: Take a Python book and read, try, improve
 - The **Hands-On**: Millions of nice hands-on tutorials are out there
 - The **Modern**: Let AI help you and explain concepts and the language
- All three paths are absolutely valid, also the **Modern**, if done correctly

Fundamentally rejecting AI is absolutely wrong in my opinion! It is a nice tool and should be used at best – and can be super helpful.

My Philosophy: AI is a nice **tool**, however, it should **NOT** do the work for you but assist you in doing it right.

The Classic

- Of course, still until today it is absolutely valid to learn Python the theoretical way and take book and read
- However, to get really good, imho, it is extremely important to get your hands dirty!
 - Try out, fail, find new ways, experiment, improve
 - In parallel, keep your eyes open for new things, best practices etc

I started essentially with this approach ~11 years ago. For me, this worked in the beginning well, but soon, I reached a point, where I got the impression that I can only improve with doing actual coding, trying out things, and so.
This is what I therefore also recommend: Get your hands dirty!

If you are interested in this approach, there is currently a [HumbleBundle](#) available. General recommendations are all O'Reilly books, such as [Fluent Python](#)

The Hands-On

- Since Python is the perfect trial&error programming language, this is a very nice approach to learn it and gain experience
- With hands-on projects, it is often more easy to incorporate good use practices on the run, instead of theoretically learning them
- Practice is key! And here, it is part of the processes

From time to time (when I have time) I am looking for nice tutorials and try them out myself - often in adapted manners. But I find this a nice practice. My last project was an AI meme generator, oriented on [this](#) tutorial.

The Modern

- Nowadays, there are thousands of ways how AI can help you, increase productivity, or even do the work for you
 - Not all studies show that this is actually true, see heise.de
- The most important things when using AI are:
 - Know what you are doing
 - **Don't blindly trust it, always crosscheck**
 - Know your tools and review, if they really profit you
- Your learning process (as well as the quality of the outcome) here mainly depends on **how** you use AI

The Modern – An Example

Exercise:

Create a small interactive program that manages a list of books. The user should be able to:

- 1) Add a new book by typing its title and author.
- 2) List all added books with a running index.
- 3) Search for books by a word in the title.
- 4) Quit the program.

The program should keep running until the user chooses to quit.

Hint: Think in advance about how to store the books, how to loop over user input, and how to make the output readable.

The Modern – An Example

Tip: ChatGPT (and others as well) have different modes. You can tell them, how they should answer to you. This helps to get away with the annoying emojis and other stuff.

- The **worst** thing to do:
 - “Please provide me a solution to this exercise”
 - Even if it works as expected: you will likely not learn much, as reading code is an important part of coding, but more importantly is writing code to get used to it
- A better start: See the AI as a **mentor** and **coding buddy**
 - At first, think about the problem yourself! Do you actually need AI? Can you solve it yourself, if yes, at which parts AI will be helpful to do nasty (but simple) implementation work?
 - If you do not have an idea yourself, it can be valuable to try: “Please help me create an implementation concept for this exercise”

The

Tip: ChatGPT (and others as well) have

which chatgpt modes exist except absolute mode

- The
 -
 -
- AI
 - 1. **Balanced mode** – default conversational mode; prioritizes clarity and helpfulness over maximal reasoning depth.
 2. **Focused mode** – designed for technical or academic precision (more concise, step-by-step, less stylistic).
 3. **Creative mode** – emphasizes narrative, brainstorming, and open-ended generation.
 4. **Code mode** – optimized for programming, debugging, and data workflows.
 5. **Data Analysis / Advanced Data mode** (formerly "Code Interpreter") – allows file uploads, plotting, and computation.
 6. **Memory mode** – retains long-term context between sessions.
 7. **Canvas / Document mode** – used for long-form editing or coding projects.
 -

Would you like a table summarizing how these modes differ in reasoning depth, verbosity, and use cases?

The Modern – An Example

Tip: ChatGPT (and others as well) have different modes. You can tell them, how they should answer to you. This helps to get away with the annoying emojis and other stuff.

- The **worst** thing to do:
 - “Please provide me a solution to this exercise”
 - Even if it works as expected: you will likely not learn much, as reading code is an important part of coding, but more importantly is writing code to get used to it
- A better start: See the AI as a **mentor** and **coding buddy**
 - At first, think about the problem yourself! Do you actually need AI? Can you solve it yourself, if yes, at which parts AI will be helpful to do nasty (but simple) implementation work?
 - If you do not have an idea yourself, it can be valuable to try:
“Please help me create an implementation concept for this exercise”

The Modern – An Example

- I'd say, the concept is solid, however, I would probably change minor things. For example:
 - I'd have a look, if there are tools for interactive applications:
 - Of course, there are! I have good experience with questionary
 - This for sure will change the interface etc. So I'd ask AI to adapt the concept accordingly: Better now.
- Now, we can start implementing:
 - Option 1: Can you please provide the boilerplate code for the project?
 - Can be dangerous, as AI might force you in a certain direction
 - Option 2: Start small with single functionalites

Directly provides the solution...
No learning effect, no practice



Small, atomic instructions are a best practice hear!

The Modern – An Example

- “Let’s start with the `add_book` function. It should take our DB dict as an argument and ask for the user input.”
- “Now, integrate it into the questionnaire main call.”
- And so on.
- The smaller the tasks, the better, as you are practicing more
 - Concept: You are the architect that understands the problem and explains the desired task algorithmic to AI that then only does the implementation work
 - Of course, you can ask it also for improvements and how you can do better, but again: review!

Small, atomic instructions are a best practice here!

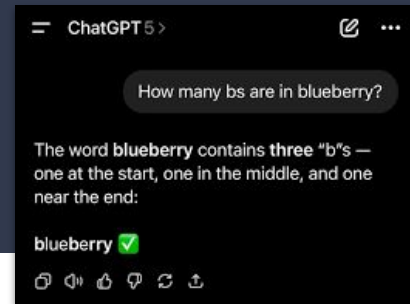
Using AI: Best Practices for Learning

- Let it explain things to you rather than just implementing it
 - Think of it: if I don't get it myself, why I am not just replaced by AI?!
 - This way, you learn the most sustainable
- Provide small tasks, not monolithic ones:
 - “Implement this new feature and change the backend accordingly”
 - Many things might change and you'll lose track
 - You'll reach the point where you do not cross-check anymore if the changes are too big and just c&p
 - “Add another input variable to function XYZ” <- better
 - + May ask, why it changed smth in a certain way

Using AI: Best Practices for Coding

- Know your problem, don't be vague. If you do not know, may let AI explain to you, or do some research (e.g. with perplexity)
- NEVER use unreviewed code! NEVER blindly trust AI!
- Be specific, don't request monolithic tasks
- Don't use it for security-critical things
 - Backend/full-stack/Auth
 - Administration
 - User data (!!)

Do not blindly trust AI. Know what you want and understand what you use. Rather see AI as a codemonkey to save you some work. But try to keep thinking yourself!



VIBE CODING

**VULNERABILITY
AS A SERVICE**

Using AI: Best Practices for Documentation

- AI is perfect for documentation:
 - Provide it a code snippet and let it explain to you what it does and let it write a docstring
- This is really valuable, since it is a simple but hated task, and AI does mostly pretty good
- => You can learn what it does (and how) + docu

Using AI: Best Practices for Debugging

- AI is often very useful for debugging: The new google...
 - Back in the days, we copied an error message to google search and hoped that someone had the same issue on stackoverflow
 - Now, AI (often) can help you to find a solution
- If used correctly, it also supports your learning:
 - Learning from errors is valuable. AI can help you with that
 - AI might find a typo or small bug in seconds, while you would be searching for long (my exp: does often not work properly...)
 - AI can (try) to explain you, what is wrong with your problem and provide you hints on how to improve/fix things

Using AI: Best Practices for Debugging II

- But it is also limited -> Best example: import sklearn (yesterday)
 - Not always helps, sometimes docu or google is more helpful
- **My recommendation:** Don't blindly c&p the entire stack trace etc!
 - This can contain sensitive information about you, API keys, and more
 - Be careful, especially for security-relevant stuff
- For security-critical things: have a look at the documentation, at least as a verification!!

In general: Do not provide too much personal information, dont just c&p everything. Never share API tokens etc...

Using AI: Best Practices for Plotting

- AI makes plotting so incredibly easy and saves a lot of time
- The most important thing: make sure, the correct data is used and it is handled correctly, and of course NOT CHANGED!
- Let's try it out with a small exercise

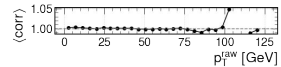
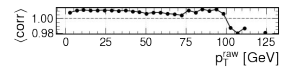
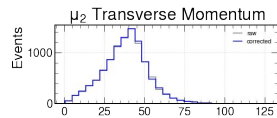
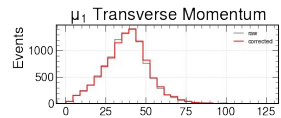
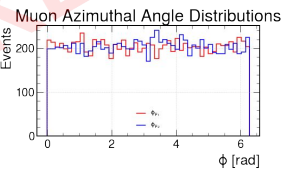
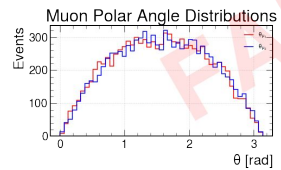
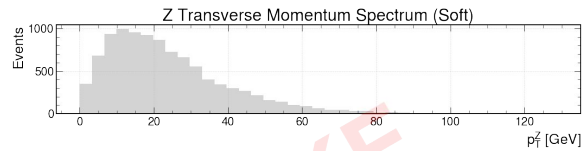
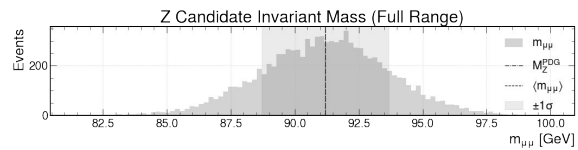
Exercise:

Let's generate some LHC mock data of a $Z \rightarrow \mu\mu$ analysis and visualize it.

Using AI: Best Practices for Plotting – Exercise

- The data should contain:
pT mu1, pT mu2, raw pT mu1, raw pT mu2, corr1, corr2, theta mu1, theta mu2, phi mu1 phi mu2, pT Z, eta Z
- Make sure that the distributions make sense (e.g. by visualizing them!)
 - + Verify with some statistics checks
- Now, please create plots:
 - 1) The histogram of the Z- > mu mu candidate invariant mass, overlayed with mu1/2 distributions
 - 2) Below, the pT Z distribution
 - 3) Next line: polar (left) and azimuth (right) distributions of both muons
 - 4) In the next line, on the left, the mu1 pT distribution and the raw pT with a subplot of the average correction factor per bin below. The same on the right for mu2.
 - Optional: Use the **mplhep** package to make them look like CMS plots
 - Overlay ALL plots with “FAKE”

Note: If you are not familiar with the physics, ask AI ;-)



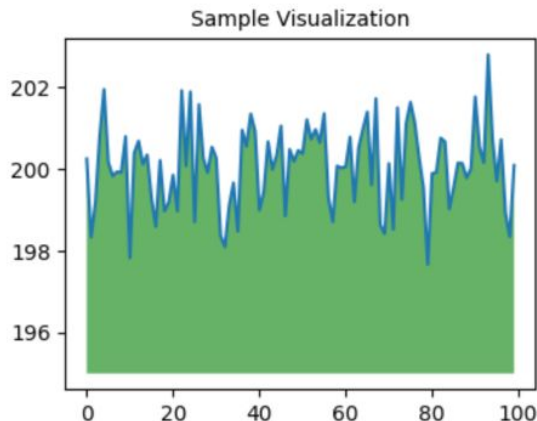
Using AI: Some Useful Tools

- More and more tools become available
- Chatbots: ChatGPT, LeChat, Claude, DeepSeek, Gemini, ...
- Integrations: [CoPilot](#), [JetBrains AI assist](#), [cursor](#)...
- Perplexity: Deep (re)search
- Next level: Agents [*I am not (yet?) a fan of*]
- More and more specialized tools become available, collections can be found e.g. [here](#)
- Jupyter with nicely made AI integration: [Google Colab](#)
 - You can perfectly use it as a coding buddy

Using AI: Some Useful Tools – Colab

```
/tmp/ipython-input-2017123786.py in <cell line: 0>()
    14
--> 15 ata = io.BytesIO()
    16 plt.savefig(data)
    17 image = F"data:image/png;base64,{base64.b64encode(c
    18 alt = "Sample Visualization"
```

NameError: name 'data' is not defined



Next steps: [Explain error](#)

```
plt.fill_between(x, ys, 195, where=(ys > 195), facecolor='g', alpha=0.6)
plt.title("Sample Visualization", fontsize=10)
```

```
ata = io.BytesIO()
```

```
+data = io.BytesIO()
```

```
plt.savefig(data)
```

```
image = F"
```

```
alt = "Sample Visualization"
```

```
display(image)
```

```
plt.close()
```

```
-----
NameError: name 'data' is not defined
```

```
/tmp/ipython-input-2017123786.py in <cell line: 0>()
    14
    15
--> 16 ata = io.BytesIO()
    17
    18
```

```
-----
NameError: name 'data' is not defined
```

Please explain this error:

NameError: name 'data' is not defined

It looks like there's a typo in the variable name `ata`. It should be `data`. I'll fix that for you.



[Accept & Run](#) [Accept](#) [Cancel](#)

What can I help you build?



Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

Sample Visualization



Don't fully trust AI!

And keep in mind: It does not guarantee you that it provides the best solution!

Knowing language and best practices is key to good code development

Very Simple Example: Sum of squares

How can I sum the squares of numbers from 1 to N in Python?

Use a list comprehension with `sum` :

python

 Copy code

```
N = 10
result = sum(i**2 for i in range(1, N + 1))
print(result)
```

Or use the closed-form formula for efficiency:

python

 Copy code

```
N = 10
result = N * (N + 1) * (2 * N + 1) // 6
print(result)
```

Very Simple Example: Sum of squares

How can I sum the squares of numbers from 1 to N in python with numpy

You can compute the sum of squares from 1 to N efficiently with NumPy as follows:

python

 Copy code

```
import numpy as np

N = 100
result = np.sum(np.arange(1, N + 1)**2)
print(result)
```

Or, using vectorized dot product (slightly faster):

python

 Copy code

```
result = np.arange(1, N + 1).dot(np.arange(1, N + 1))
```

     ...

Performance Comparison

```
%%time
N = 10000
for j in range(100):
    result = sum(i**2 for i in range(1, N + 1))

CPU times: user 67.1 ms, sys: 2.08 ms, total: 69.2 ms
Wall time: 68.1 ms
```

1st Prompt: No specification

2nd Prompt: Use NumPy

```
%%time
import numpy as np
N = 10000
for j in range(100):
    result = np.sum(np.arange(1, N + 1)**2)

CPU times: user 3.26 ms, sys: 189 µs, total: 3.45 ms
Wall time: 2.51 ms
```

Often, it helps to be more specific on the problem! But not always. Knowledge on that is important!

3rd Try – Still no Guarantee!

How can I sum the squares of numbers from 1 to N in Python? Please provide the most performant implementation, except the close formular

For maximum performance **without using the closed form** $N(N+1)(2N+1)/6$, use **NumPy vectorization** — it leverages optimized C loops:

python

 Copy code

```
import numpy as np

def sum_squares_numpy(N: int) -> int:
    arr = np.arange(1, N + 1, dtype=np.int64)
    return np.sum(arr * arr)
```

This avoids Python loops and is orders of magnitude faster for large `N`.

If NumPy is not allowed, the fastest pure-Python approach (using generator expressions) is:

python

 Copy code

```
def sum_squares_py(N: int) -> int:
    return sum(i * i for i in range(1, N + 1))
```

%%time

```
def sum_squares_numpy(N: int) -> int:
    arr = np.arange(1, N + 1, dtype=np.int64)
    return np.sum(arr * arr)
```

N = 10000

```
for j in range(100):
    sum_squares_py(N)
```

CPU times: user 47.8 ms, sys: 1.94 ms, total: 49.8 ms
Wall time: 49.3 ms

Do not blindly trust AI. Know what you want and rather use it as a codemonkey to save some work. But try to think yourself!

Always cross-check and try to know things yourself!

(The implementation work, the AI still can do!)