Prompt Engineering with GitHub Copilot

Best practices for producing better code results

$ whoami

My name is Eyar Zilberman

🎩 I'm leading the product @ Datree

🎩 I'm a GitHub Star member since 2022

🎩 I'm organizer of the biggest GitHub Users Group in the world!

🐐 I'm also a podcaster, open-source contributor, and I really love goats

Start with the basic - what is GitHub Copilot?

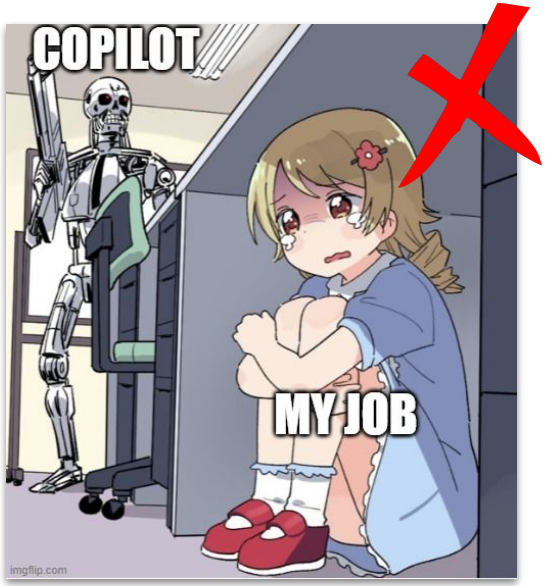
Wikipedia:

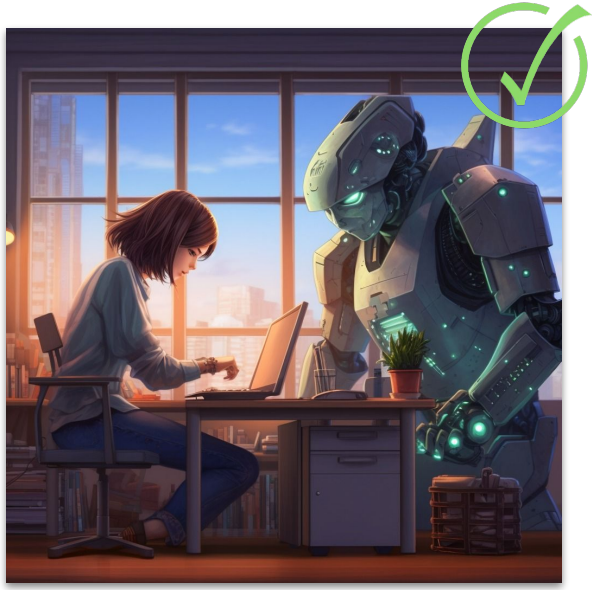
> "GitHub Copilot is a cloud-based AI tool … [to] **assist users** of Visual Studio Code [and other IDEs] … by autocompleting code."

datree

Start with the basic - how it works?

GitHub Copilot is your AI pair programmer



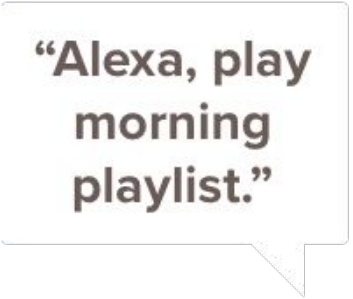


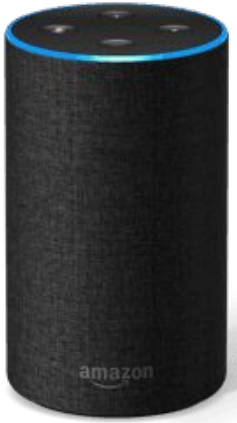
> We need a way to explain Copilot how to assist us…

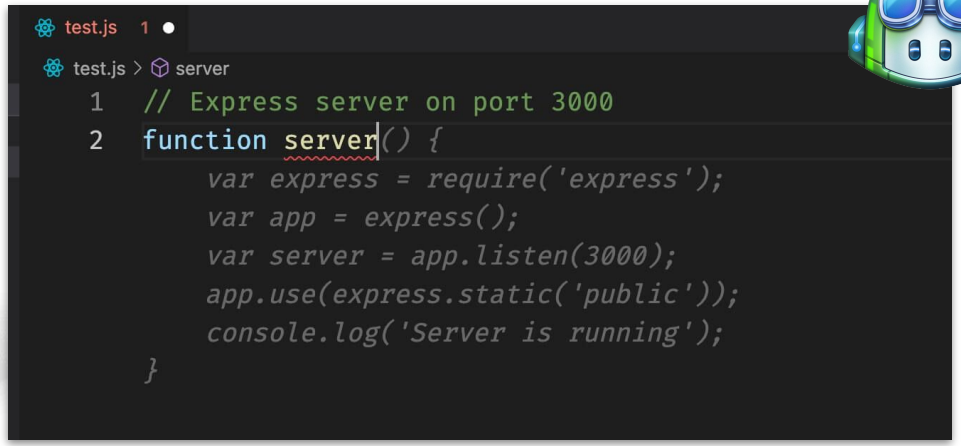
datree

Prompt engineering!

> Prompt engineering is the practice of giving an AI model the right instructions to produce the results you expect.





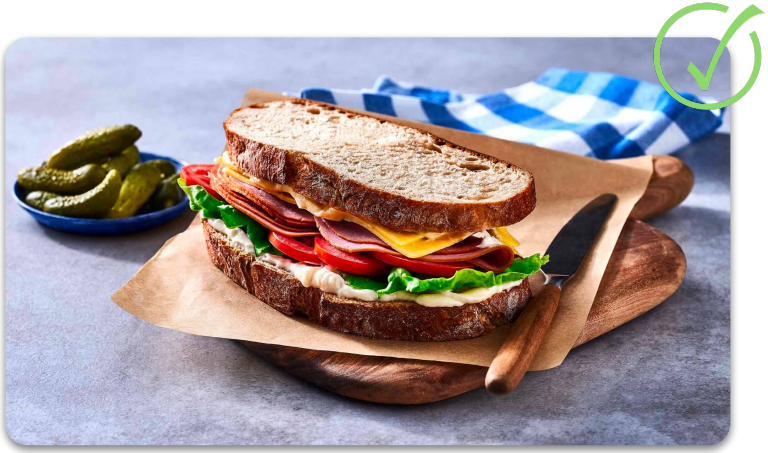


What's the catch here?

Sand·wich

Definitions from Oxford Languages

> An item of food consisting of two pieces of bread with meat, cheese, or other filling between them.



datree

Before we begin

- I'm using Python in my examples

  You don't need to be familiar with python at all

- I'm going to use API, regex and cron in my examples

  You don't need to be familiar with none of them

now we are set to start!

datree

0. High level description with detailed instructions

```
# get user data from api
```

prompt

```
def get_data():
    url = 'https://api.covid19api.com/summary'
    response = requests.get(url)
    data = response.json()
    return data
```

Copilot

datree

0. High level description with detailed instructions

```
# get user data from api v1/copilot-demo,
# then return it as a JSON object
```

datree

0. High level description with detailed instructions

```
# get user data from api v1/copilot-demo,
# then return it as a JSON object

app = Flask(__name__)

@app.route('/v1/copilot-demo', methods=['GET'])
def get_data():
    url = 'https://api.copilot.com/v1/copilot-demo'
    response = requests.get(url)
    return jsonify(response.json())
```

1. Provide specific details

```
# write a unittest for the function get_formatted_name()
# in the module name_function.py
```

datree

1. Provide specific details

```
# write a unittest for the function get_formatted_name()
# in the module name_function.py
import unittest
from name_function import get_formatted_name

class NamesTestCase(unittest.TestCase):
    """Tests for 'name_function.py'."""

    def test_first_last_name(self):
        """Do names like 'Janis Joplin' work?"""
        formatted_name = get_formatted_name('janis','joplin')
        self.assertEqual(formatted_name,'Janis Joplin')
```

datree

1. Provide specific details

```
# use pytest to write a unittest for the function
# get_formatted_name()in the module name_function.py
```

datree

1. Provide specific details

```
# use pytest to write a unittest for the function
# get_formatted_name()in the module name_function.py
import pytest
from name_function import get_formatted_name

def test_get_formatted_name_middle():
    """Do names like 'Wolfgang Mozart' work?"""
    formatted_name = get_formatted_name('wolfgang','amadeus')
    assert formatted_name == 'Wolfgang Amadeus Mozart'
```

datree

2. Explain yourself with examples

```
# write a function to validate cron expression from
# user input by using regex
```

datree

Pause!

Regular expression (aka regex):
Sequence of characters that specifies a match pattern in text

Cron:
Job scheduler pattern for computers

- *5 4 * * sun* => “At 04:05 on Sunday.”
- *@weekly* => “At 00:00 on Sunday.”

> **It’s possible to use regex (sequence) to validate cron patterns**

2. Explain yourself with examples

```
# write a function to validate cron expression from
# user input by using regex
```

datree

2. Explain yourself with examples

```
# write a function to validate cron expression from
# user input by using regex

def validate_cron_expression(cron_expression):
    regex = r'^((\d{1,2}|\*)\s){4}(\d{1,2}|\*)$'
    if re.match(regex, cron_expression):
        return True

if __name__ == '__main__':
```

➜ copilot Users/eyar/Dev/copilot/demo.py
Please input cron expression: @weekly
None

2. Explain yourself with examples

```
# write a function to validate cron expression from
# user input by using regex
# the regex should also catch cron patterns like @daily, @weekly, etc.
```

datree

2. Explain yourself with examples

```
# write a function to validate cron expression from
# user input by using regex
# the regex should also catch cron patterns like @daily, @weekly, etc.
def validate_cron_expression(cron_expression):
    regex = r'^((\d{1,2}|\*)\s){4}(\d{1,2}|\*)$|^(@\w+)$'
    if re.match(regex, cron_expression):
        return True


if __name__ == '__main__':
```

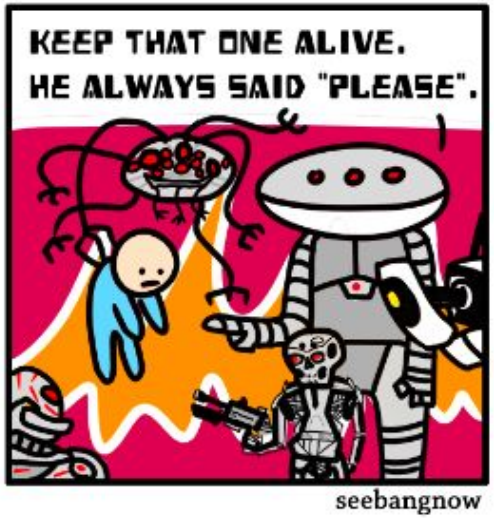
➜ copilot Users/eyar/Dev/copilot/demo.py
Please input cron expression: @weekly
True

Other tips

- **Prefer common languages**
  - Python, JavaScript, TypeScript, Ruby, Go, C# and C++
- **Be always explicit and not implicite**
  - Implicite: don’t use package X
  - Explicit: use package Y
- **Use consistent naming conventions for variables**
  - Don’t mix camelCase together with snake_case
- **Open relevant files in neighboring tabs**

Most important tip - be polite!

- “Please write…”
- “I would like to…”



Although Copilot is awesome, it should not replace:

- Code review
- Running integration tests
- Manually testing your code
- Checking for security issues
- Any other coding practices…

A look into the future - Copilot X

- Copilot for Docs
- Copilot for Pull Requests
- Copilot Chat
- Copilot for CLI
- Copilot Voice

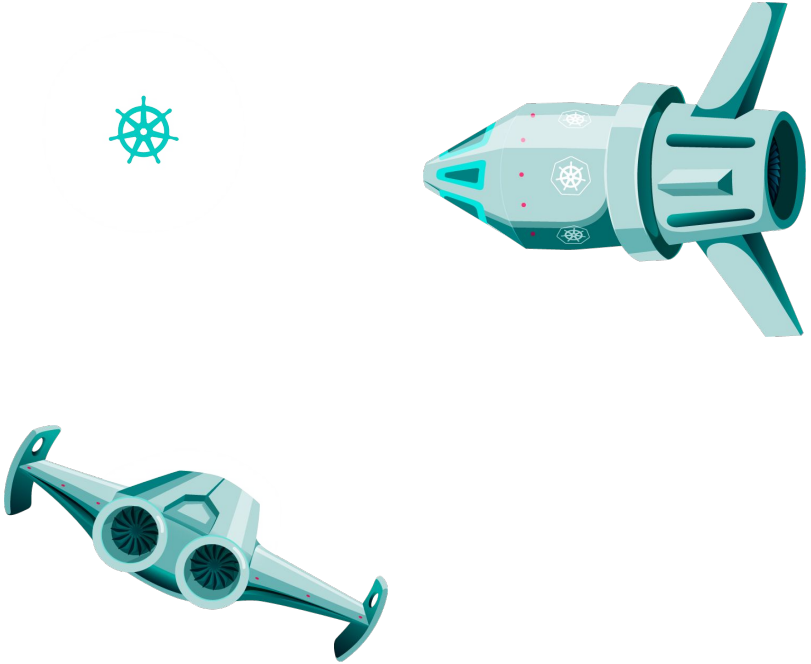
datree

Where to find me

GitHub: eyarz

Email: eyar@datree.io

Linkedin: eyar-zilberman



datree

datree

Thank You

links

https://dev.to/github/a-beginners-guide-to-prompt-engineering-with-github-copilot-3ibp

https://microsoft.github.io/prompt-engineering/

https://github.blog/2023-05-17-how-github-copilot-is-getting-better-at-understanding-your-code/

https://docs.github.com/en/copilot/overview-of-github-copilot