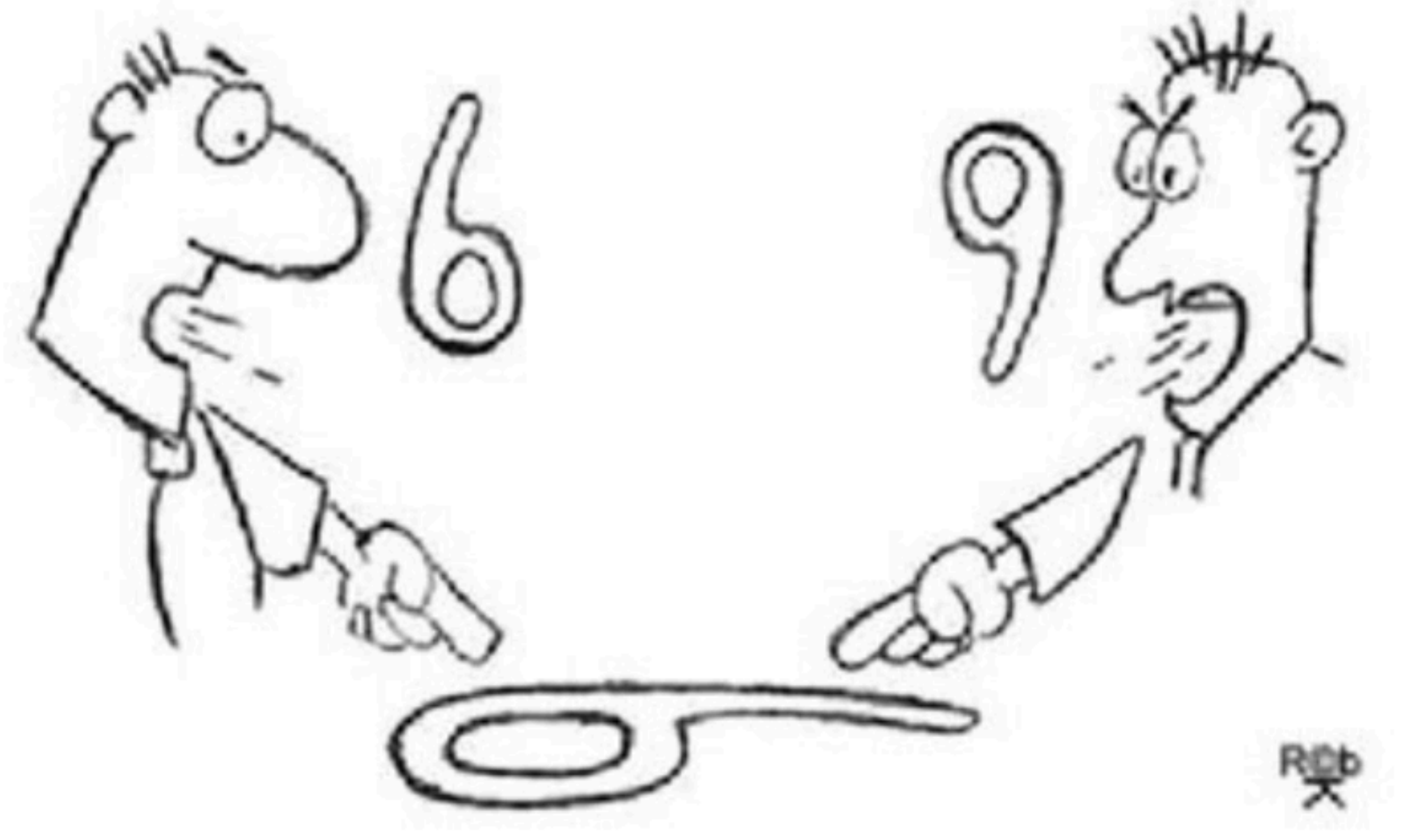
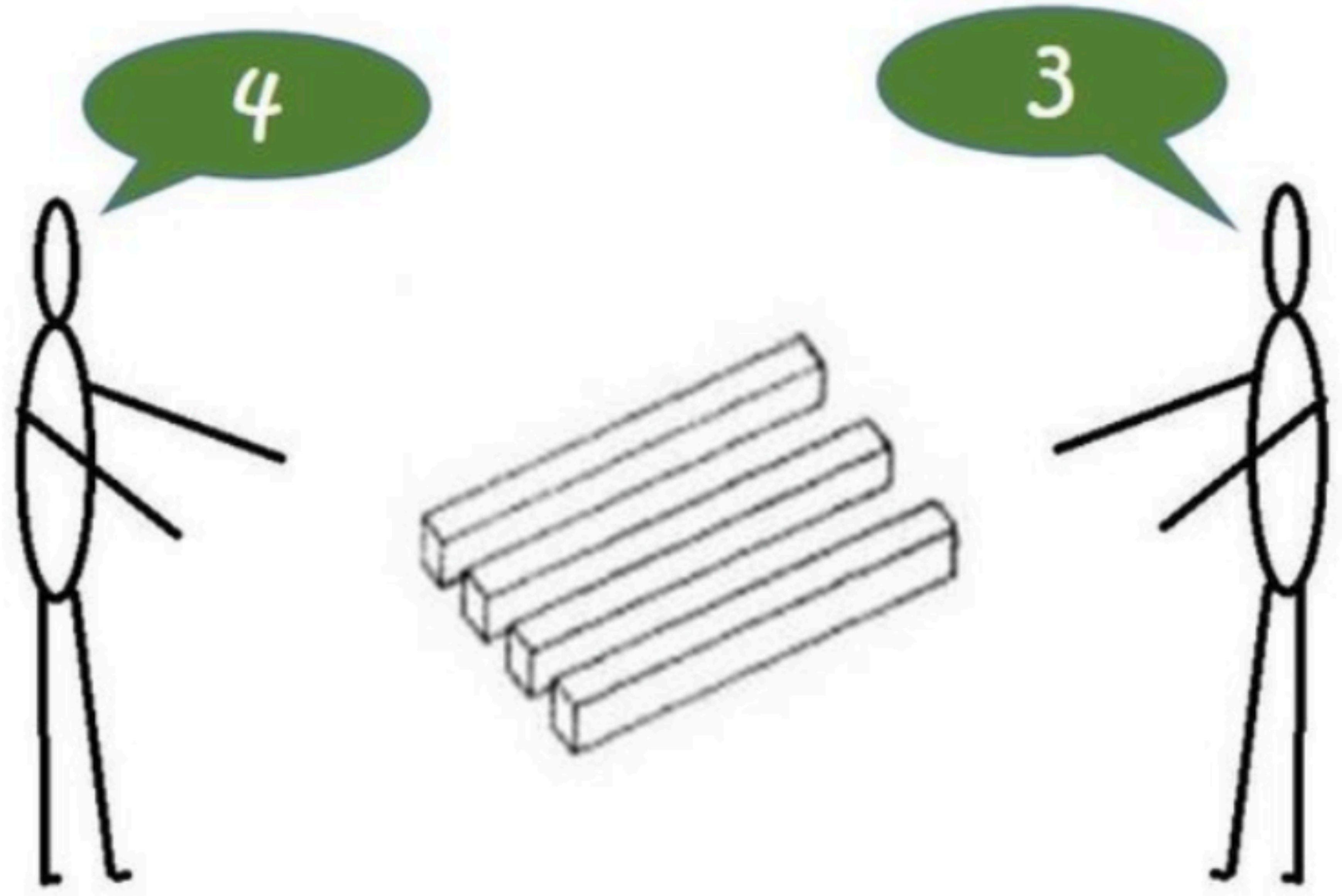
CSC1002 - AI-Assisted Programming

Design - Tradeoff





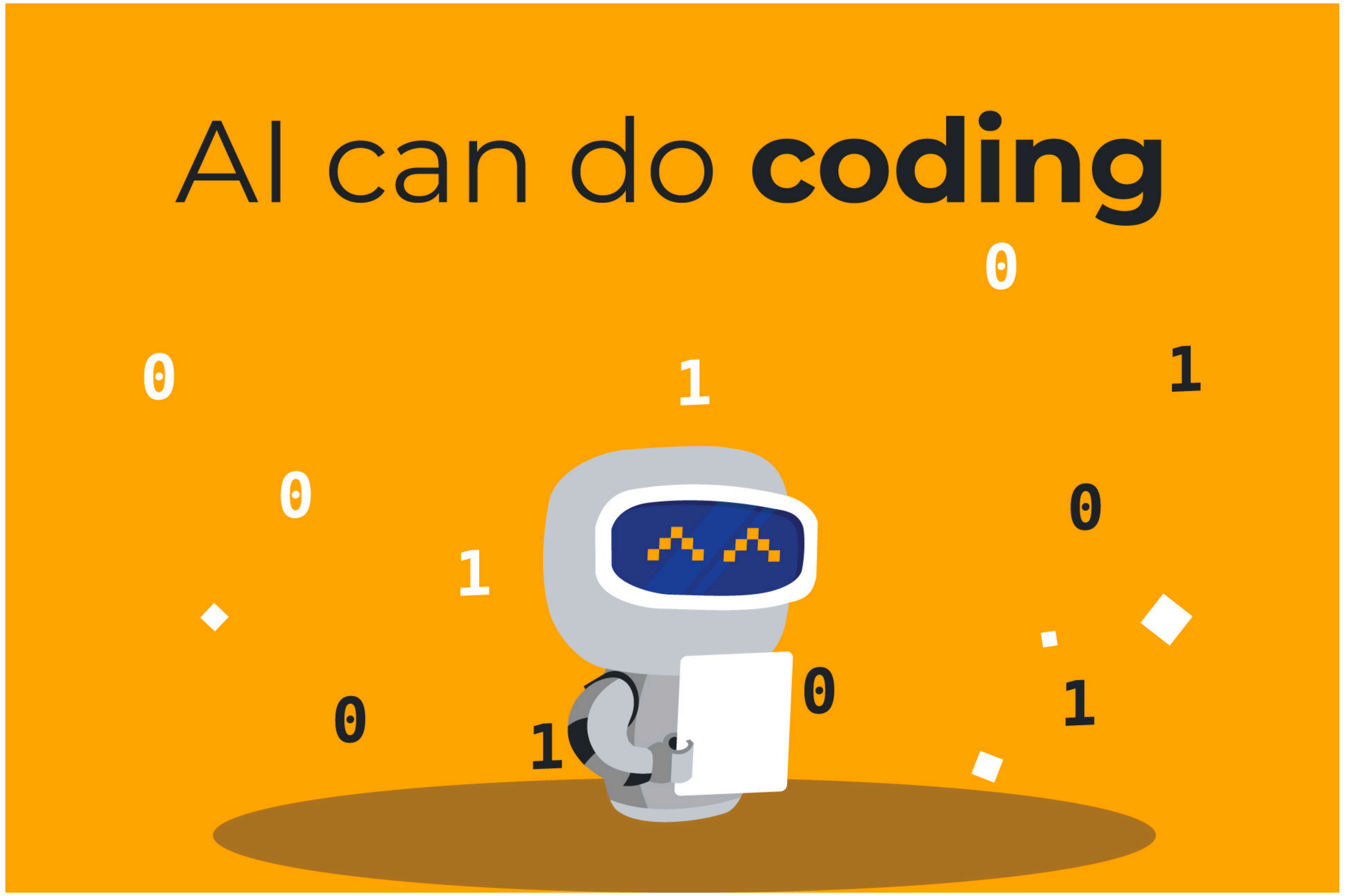
Problem Decomposition - Function

Clean Code - Single Responsibilty

- Clearly defined behaviour, with one clear task to perform
- Short in number of lines of code, say < 15 lines
- Clear Input and Output
    - Prefer a function with fewer parameters, no more than three

Goal #1 - AI Assisted Coding

CHAT



Future Programming - AI as Programming Copilot

- When first learning to program, learners often dedicated a significant amount of time working with the syntax and fundamental structure of programs.
- As AI technology advances, the focus for learners will shift towards design, coding strucrture and testing (Clean Code)
- In today's lecture, we will introduce students to the use of Open AI technology as a programming copilot to aid in the development of a game program called Number Guessing.
- Through guided interaction called prompt, we will demonstrate how the AI Assistant facilitates the completion of coding in each function based on the descirbed behavior from the prompt,
- By the end of the lecture, students will have gained a good understanding of how AI technology is reshaping the future of programming development, including how to integrate AI technology as a programming helper and how to leverage the AI Assistant to bootstrap software development.
- Additionally, we will present instances where the AI Assistant falls short.

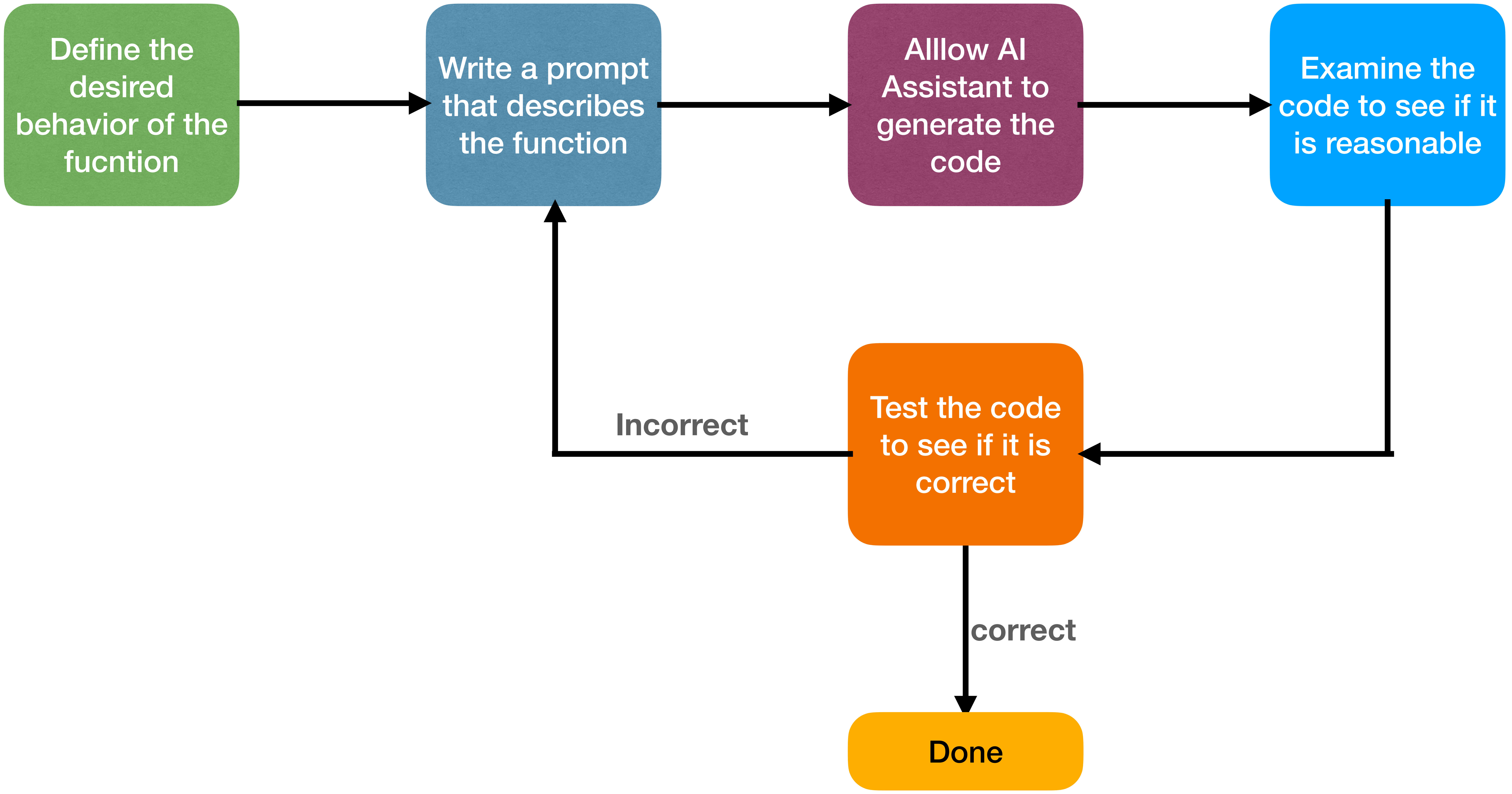
AI Assisted Programming Tasks

- Code Completion - Auto complete logic as programmers type to reduce errors and to save time by providing suggestions for method names, parameters, and even entire code segment (for, if, while, …).
- Code Generation - Generate logic for the entire function/Program based on user-provided specification and user-specified constraints using latest generative AI technology.
- Code Refinement - Refine logic with specific requirements
- Code Summarization - Generate a text description at function level or a selected code segment; the description can be used to understand the logic structure and to make further refinement as needed.
- Code Translation - Translate logic from one language to another, or one format to another (ex: JSON to Class)
- Defeat Detection - Complexity tends to increase due to CHANGES, it aims to detect unknown errors and error-prone logic.
- Clone Detection - Identify similar code fragments (repeated logic)
- Testing - Generate test cases for your logic

AI Assisted Programming - Limitations

- Wrong Logic
- Obsolete or Deprecated Logic
- Outdated Features
- Unlicensed Logic
- Missing Library/Module

AI-Assisted Programming Flow


Define the desired behavior of the fucntion
Write a prompt that describes the function
Alllow AI Assistant to generate the code
Examine the code to see if it is reasonable
Test the code to see if it is correct
Incorrect
correct
Done

AI Assisted Prompt

Prompt - Input to Generative AI

Text in natural language

- In the context of Generative AI, the manner by which we engage AI system is conversation called PROMPT, simply a piece of text in natural language.
- The QUALITY of prompt directly dictates the accuracy of the responses from the AI system.
- In summary, a well-defined prompt is the recipe for a successful conversation that facilitates understanding of the topics of interest. For examples:
    - “Tell me about python programming”, or
    - I am a freshmen student and I am about to take my first programming course in python. Give me an introduction to python programming, assumming I have no prior programming experience. Show me the top 10 useful features and describe it with examples and explanation.

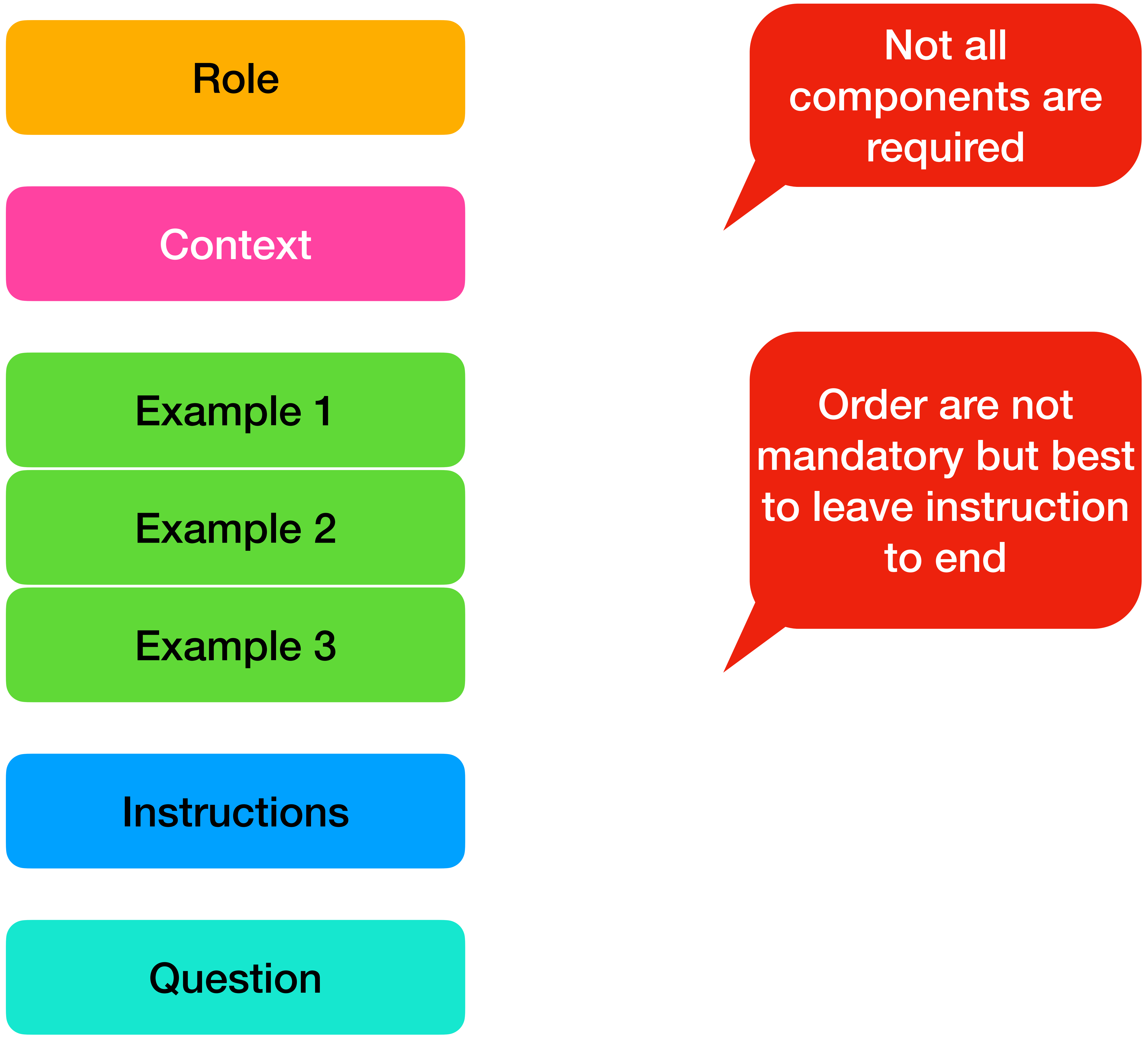
Prompt - General Guidelines

- Clarify
  - use simple sentences and instructions; favor short prompt and short instructions
  - add examples with context (background) as needed
- Conciseness
  - breakdown instructions into smaller sentences, in steps in proper sequence.
  - follow standard formats/symbols recognized by AI assistant:
    - (), [], :, “”,_, |
- Focus
  - be specific on the topics of interest
- Role
  - control tone, style and depth of the response

Prompt - AntiPatterns

- Information Overloaded
  - Too much information might reduce accuracy of the response
- Open-ended questions
  - Avoid too general questions such as “Help me to become Kinley!”
- Lack of Constraints
  - Specify any specific requirements, if any.
    - Format Ouput, Response Style, Tone, and so on.

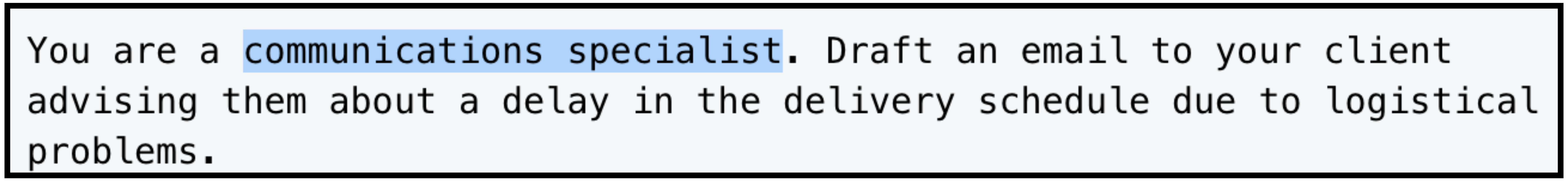
Prompt - Components



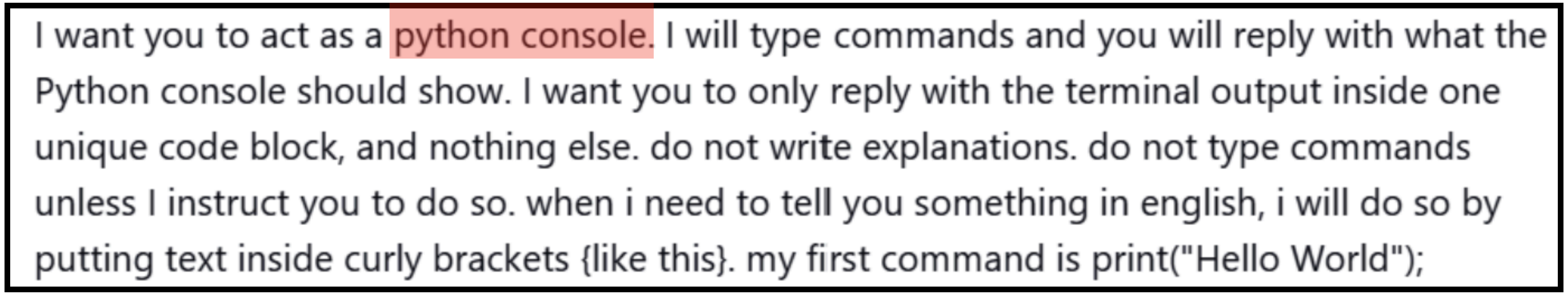
Not all components are required

Order are not mandatory but best to leave instruction to end

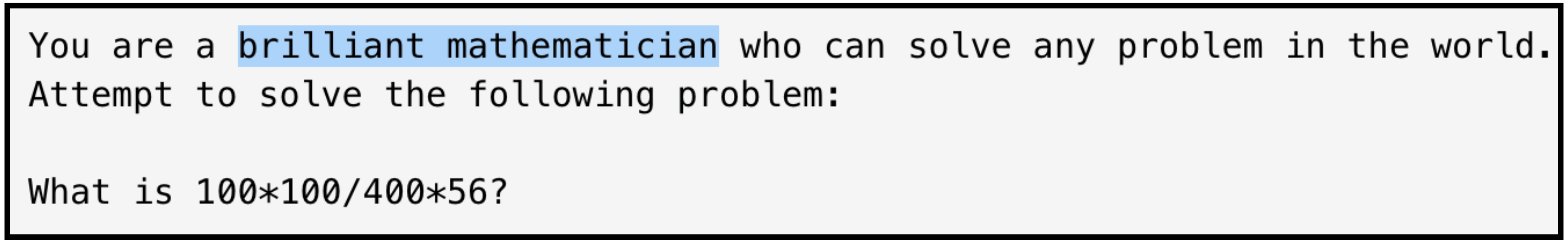
Prompt - Role



```
You are a communications specialist. Draft an email to your client
advising them about a delay in the delivery schedule due to logistical
problems.
```



I want you to act as a python console. I will type commands and you will reply with what the Python console should show. I want you to only reply with the terminal output inside one unique code block, and nothing else. do not write explanations. do not type commands unless I instruct you to do so. when i need to tell you something in english, i will do so by putting text inside curly brackets {like this}. my first command is print("Hello World");



```
You are a brilliant mathematician who can solve any problem in the world.
Attempt to solve the following problem:

What is 100*100/400*56?
```

Prompt - Examples

Great product, 10/10: positive
Didn't work very well: negative
Super helpful, worth it: positive
It doesnt work!:

Add 3+3: 6
Add 5+5: 10
Add 2+2:

1. Laura Simmons [FARMER]
2. Kevin Alvarez [DANCE INSTRUCTOR]
3. Rachel O'Connor [VOLUNTEER]
4. Kinley [?]

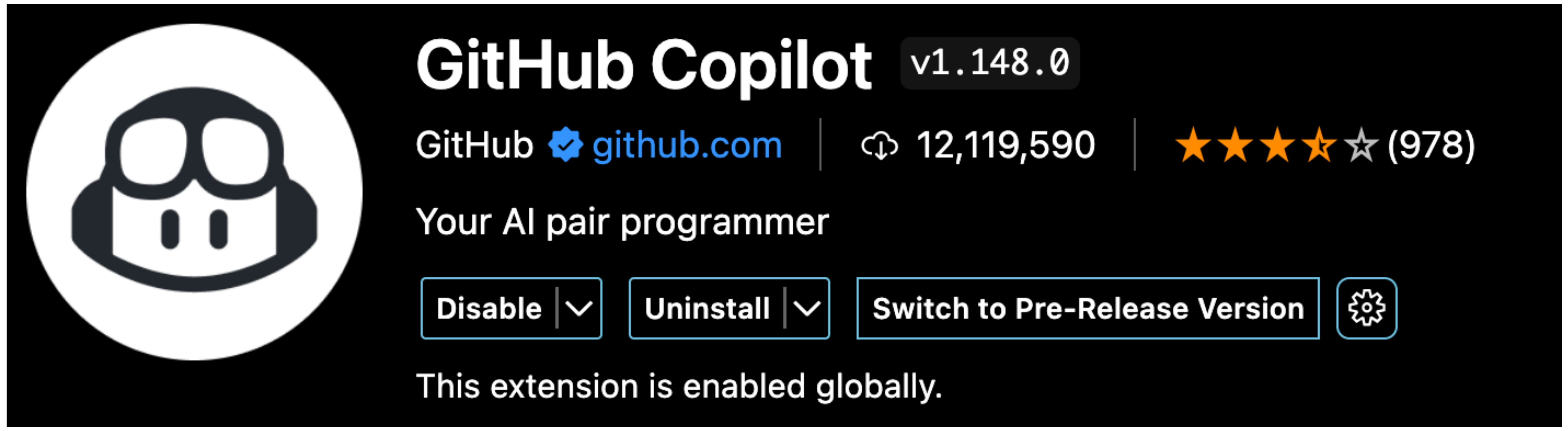
Prompt - Fine Tuning

- Start with a concise, clear, and focused prompt with specific context.
    - an overview of the topic of interest
- Elaborate further with more specific prompt
    - ask specific question based on previous response
- Stay within the context
    - Ensure AI Assistant stays within the topic of interest at all time
    - Re-prime prompt if needed

Effective Prompt - Programming

- Context
    - Open Files
    - History
    - Overall Description
- Instructions
    - Logic Decomposition Steps
    - step-by-step line comments
    - Specific Requirements
- Be Predictable
    - Consistent names, input and output
    - Parameters and Returns
    - Examples

VSC - Open AI tools



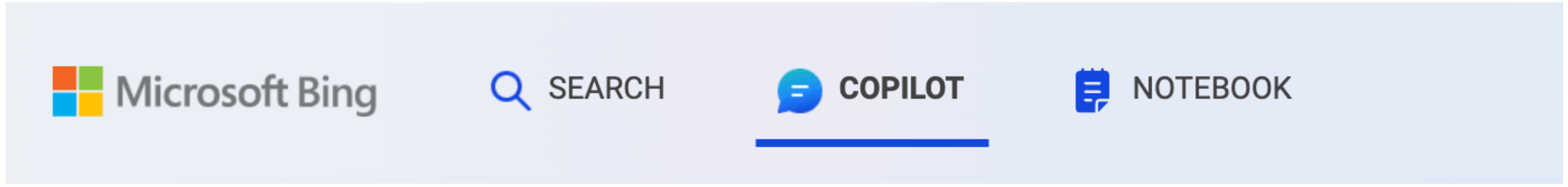
GitHub Copilot v1.148.0

GitHub github.com | 12,119,590 | ★★★★☆ (978)

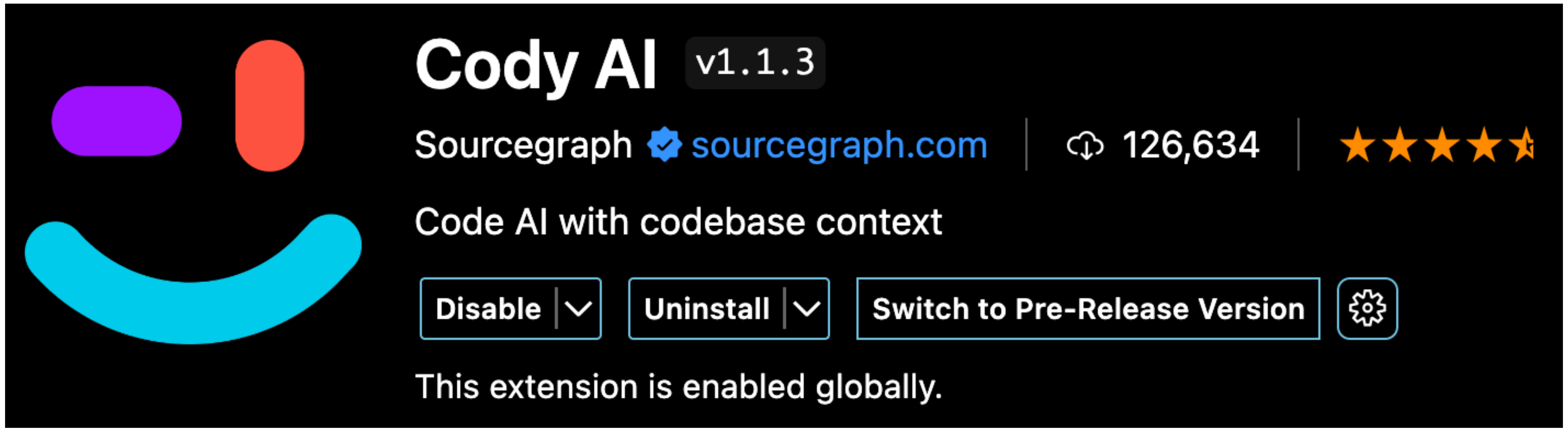
Your AI pair programmer

This extension is enabled globally.

Microsoft Copilot



Cody AI



Cody AI v1.1.3

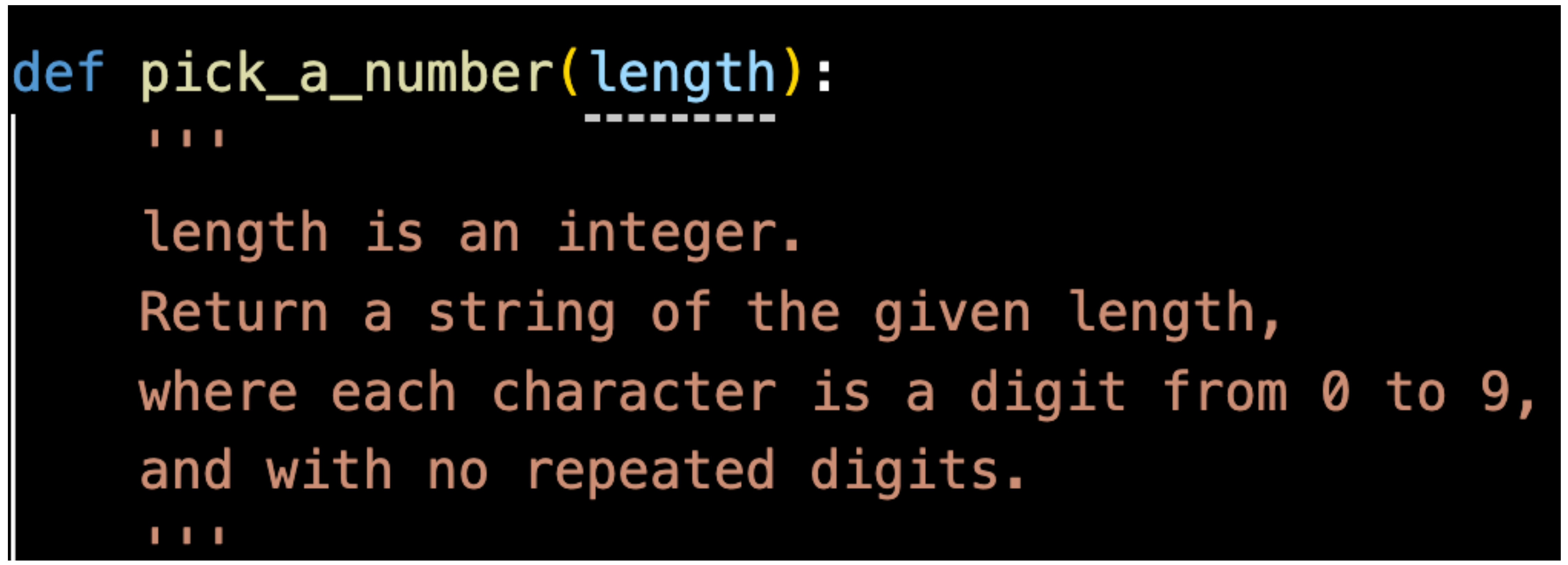
Sourcegraph sourcegraph.com | 126,634 | ★★★★★

Code AI with codebase context

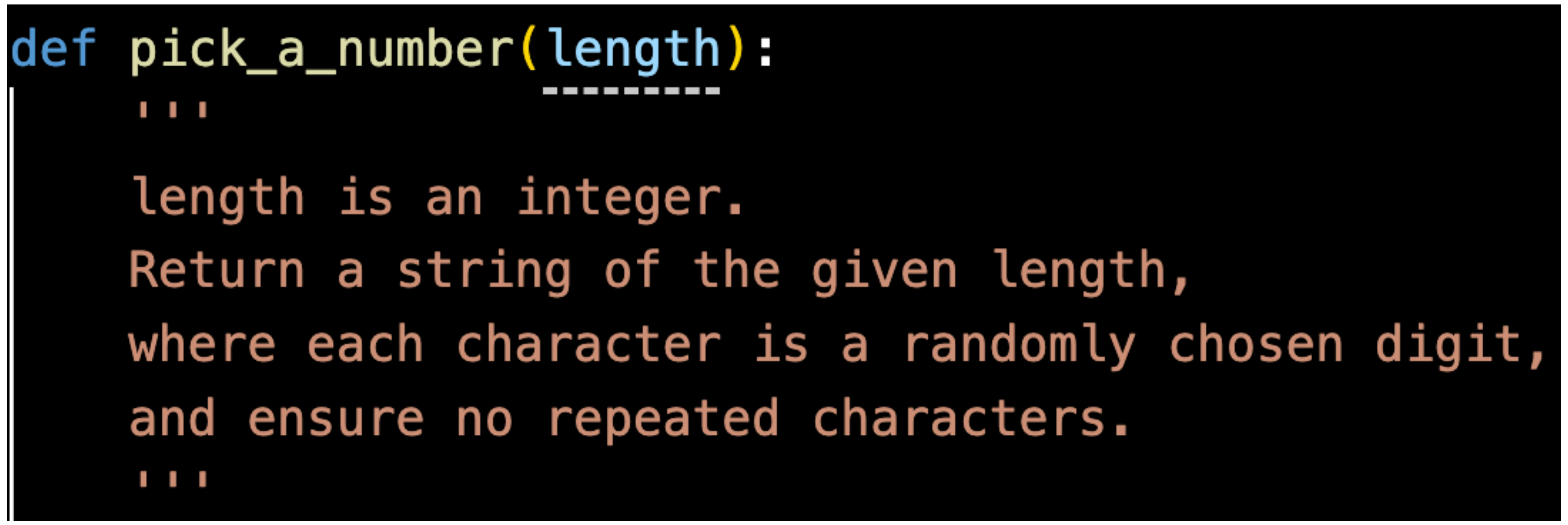
This extension is enabled globally.

Prompt - pick_a_number()

Prompt 1 - general description

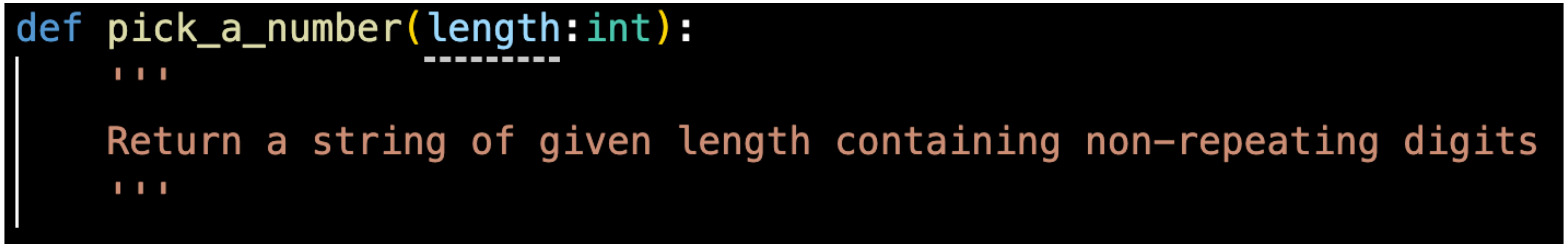


```python
def pick_a_number(length):
    '''
    length is an integer.
    Return a string of the given length,
    where each character is a digit from 0 to 9,
    and with no repeated digits.
    '''
```

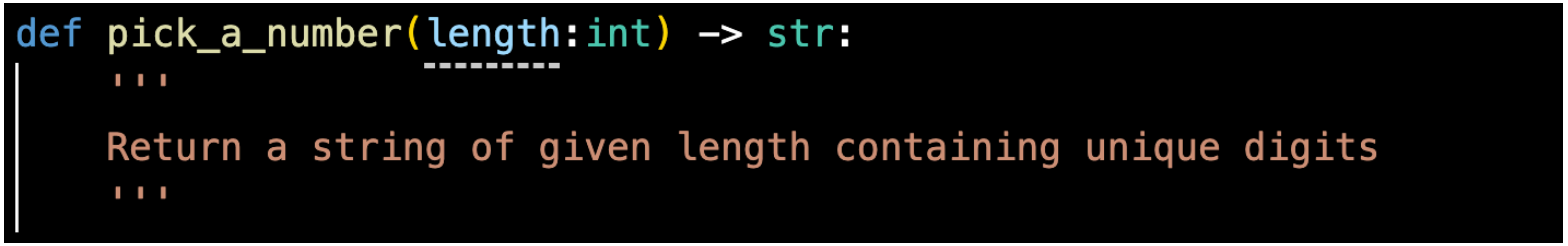


```python
def pick_a_number(length):
    '''
    length is an integer.
    Return a string of the given length,
    where each character is a randomly chosen digit,
    and ensure no repeated characters.
    '''
```

Prompt 2 - with typing hint



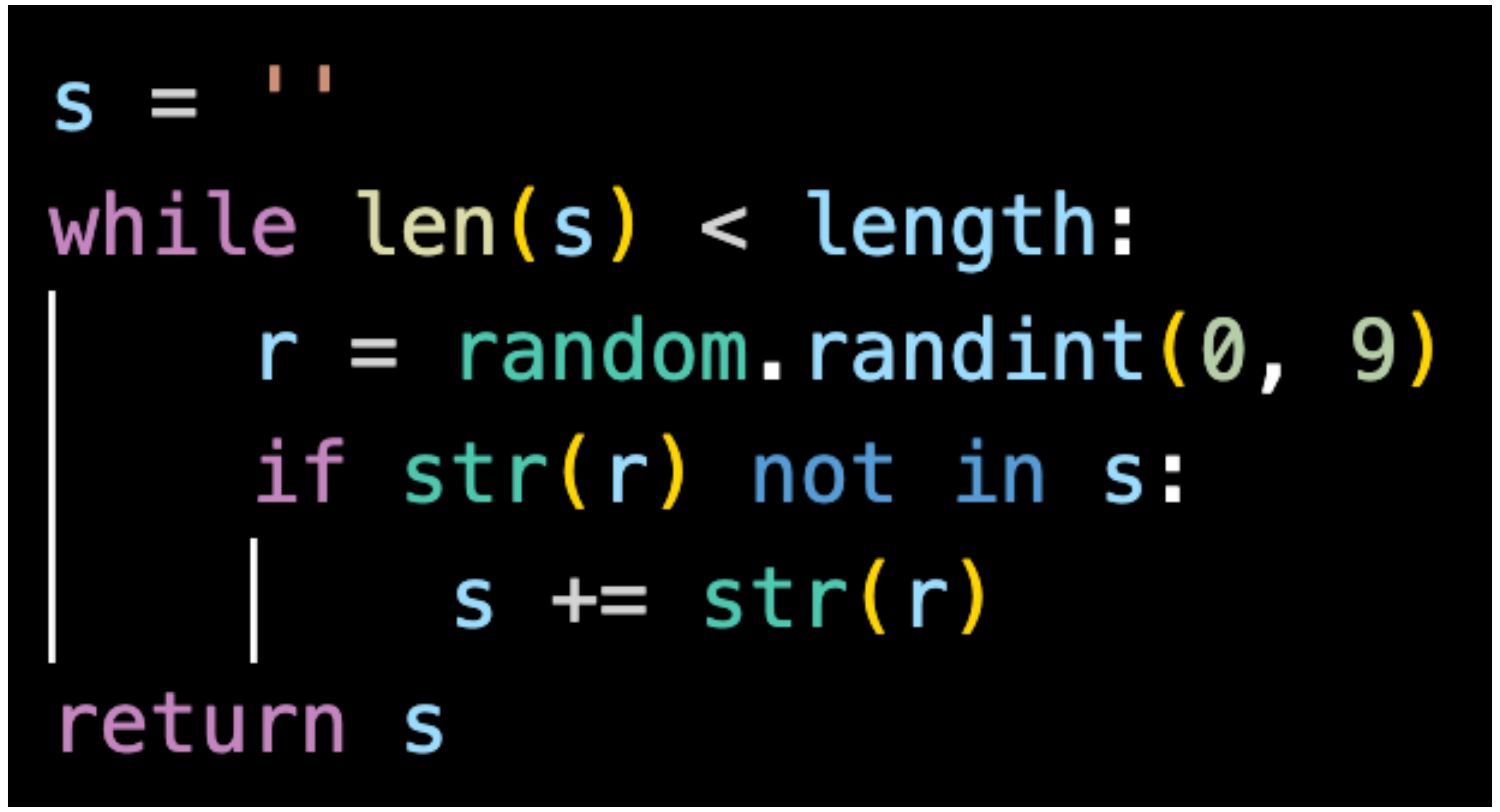
```python
def pick_a_number(length:int):
    '''
    Return a string of given length containing non-repeating digits
    '''
```



```python
def pick_a_number(length:int) -> str:
    '''
    Return a string of given length containing unique digits
    '''
```

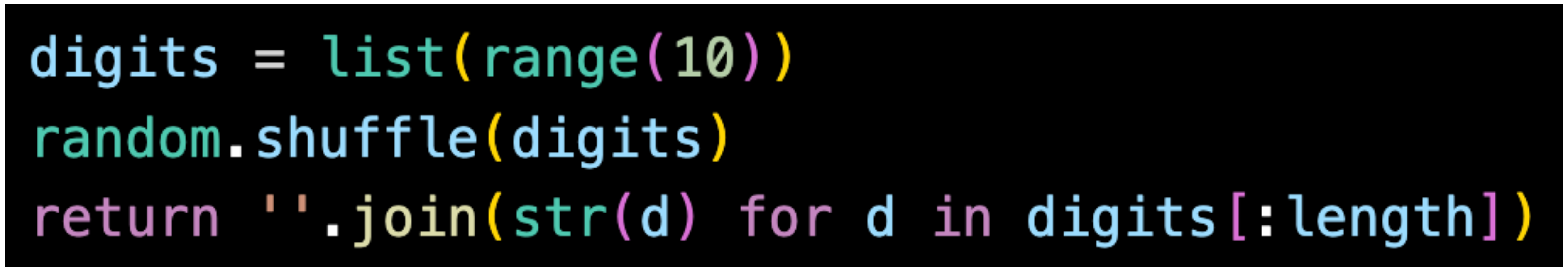
Sample AI-generated source code

Try calling these functions with length > 10 ????



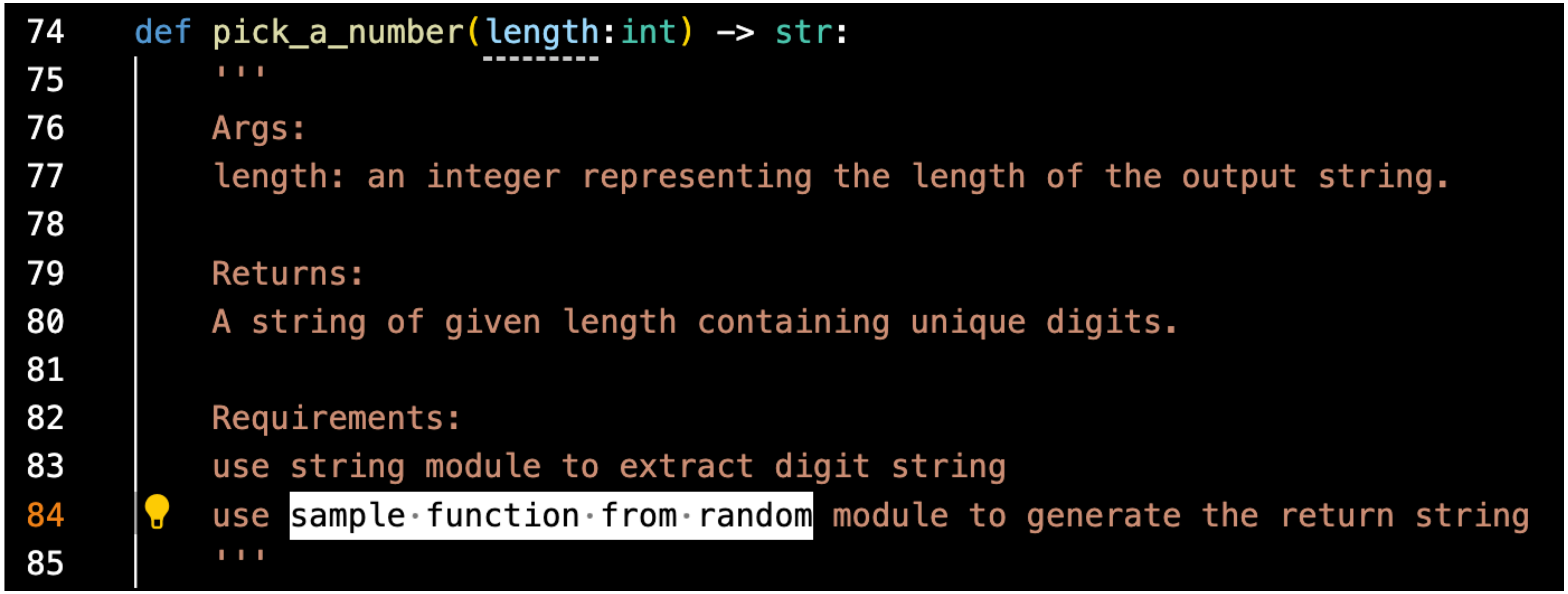
```
s = ''
while len(s) < length:
    r = random.randint(0, 9)
    if str(r) not in s:
        s += str(r)
return s
```

Functionally they are identical



```
digits = list(range(10))
random.shuffle(digits)
return ''.join(str(d) for d in digits[:length])
```

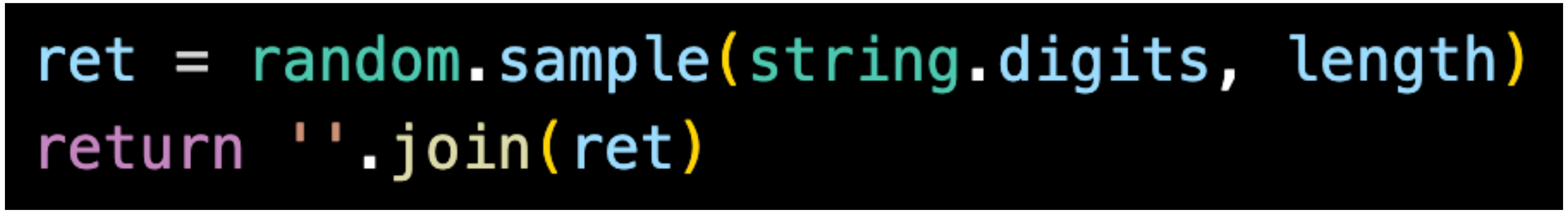
Prompt 3 - specific requirements



```python
def pick_a_number(length:int) -> str:
    '''
    Args:
    length: an integer representing the length of the output string.

    Returns:
    A string of given length containing unique digits.

    Requirements:
    use string module to extract digit string
    use sample function from random module to generate the return string
    '''
```

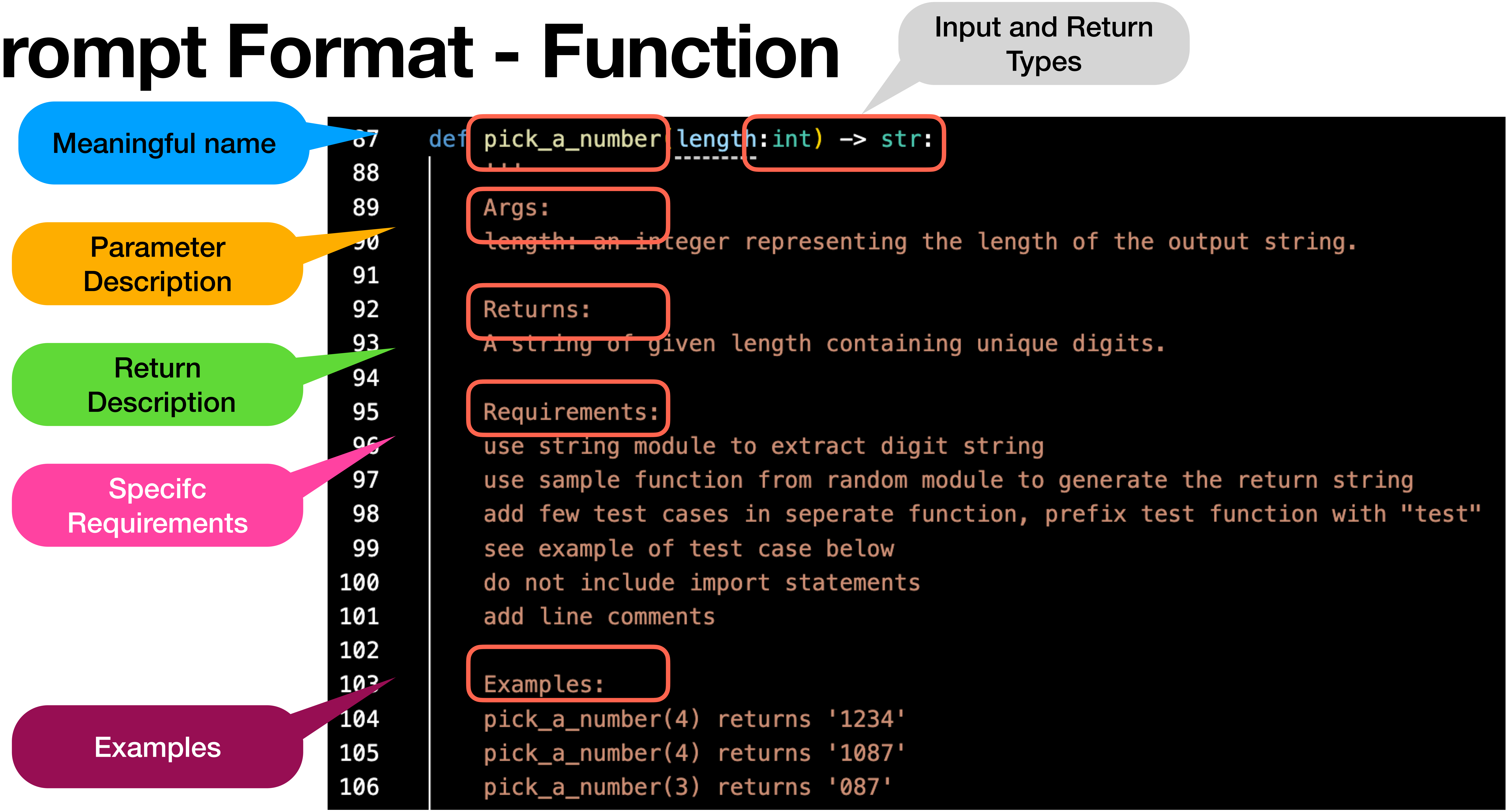
Requirements: Specific Method

Guide AI to use Sample() method to generate the random string



```
ret = random.sample(string.digits, length)
return ''.join(ret)
```

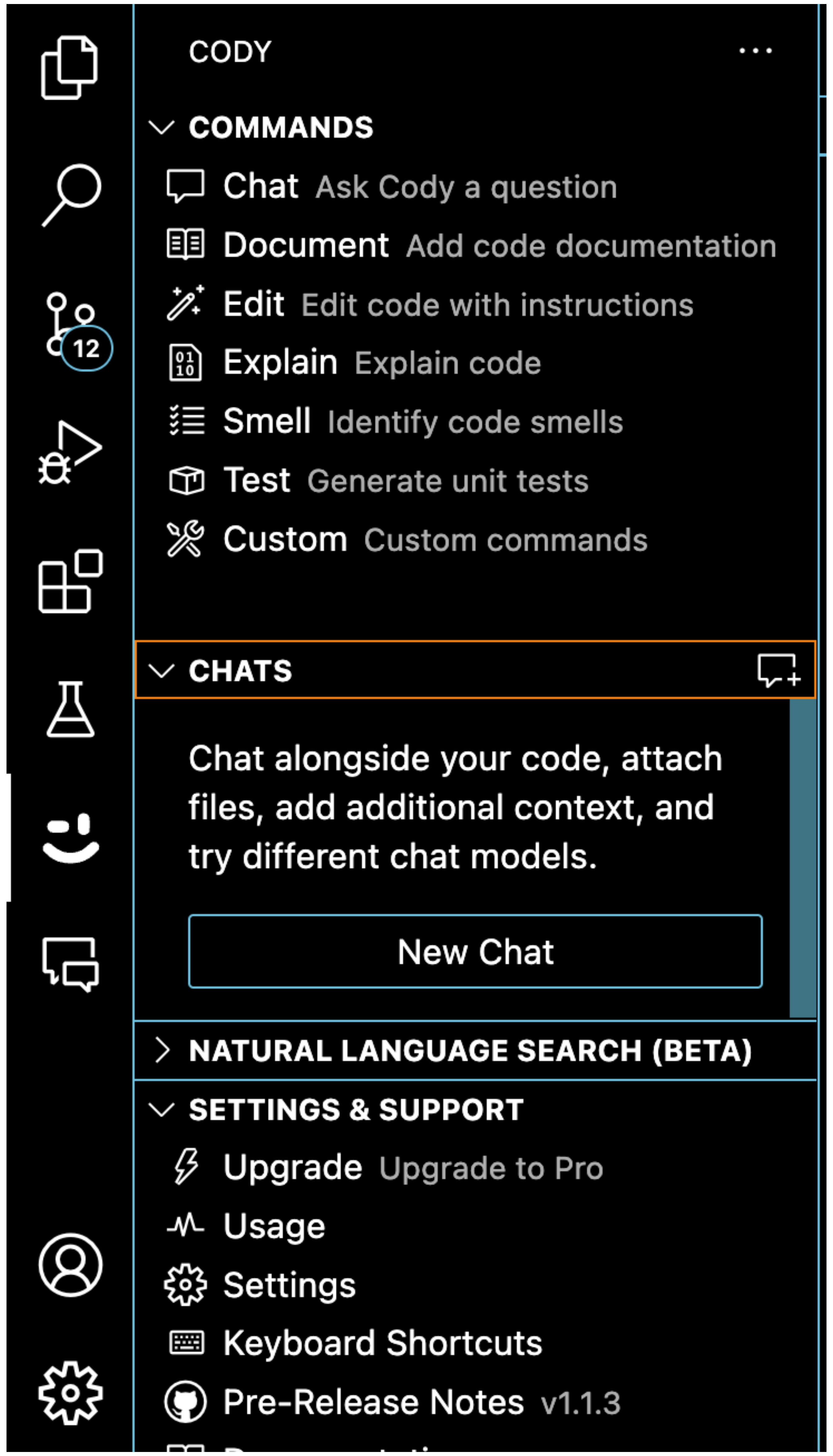
Prompt Format - Function



```
def pick_a_number(length:int) -> str:
    '''
    Args:
    length: an integer representing the length of the output string.

    Returns:
    A string of given length containing unique digits.

    Requirements:
    use string module to extract digit string
    use sample function from random module to generate the return string
    add few test cases in seperate function, prefix test function with "test"
    see example of test case below
    do not include import statements
    add line comments

    Examples:
    pick_a_number(4) returns '1234'
    pick_a_number(4) returns '1087'
    pick_a_number(3) returns '087'
```

Implementation - AI-Assisted

Cody AI - Demo



CODY

COMMANDS

- Chat Ask Cody a question
- Document Add code documentation
- Edit Edit code with instructions
- Explain Explain code
- Smell Identify code smells
- Test Generate unit tests
- Custom Custom commands

CHATS

Chat alongside your code, attach files, add additional context, and try different chat models.

New Chat

NATURAL LANGUAGE SEARCH (BETA)

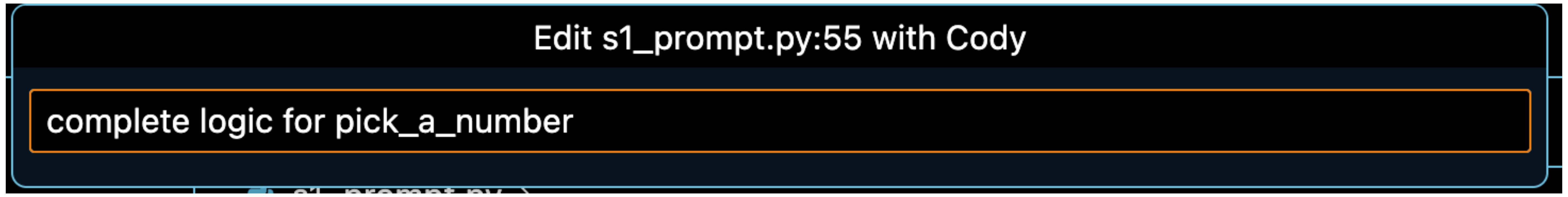
SETTINGS & SUPPORT

- Upgrade Upgrade to Pro
- Usage
- Settings
- Keyboard Shortcuts
- Pre-Release Notes v1.1.3

Preparation

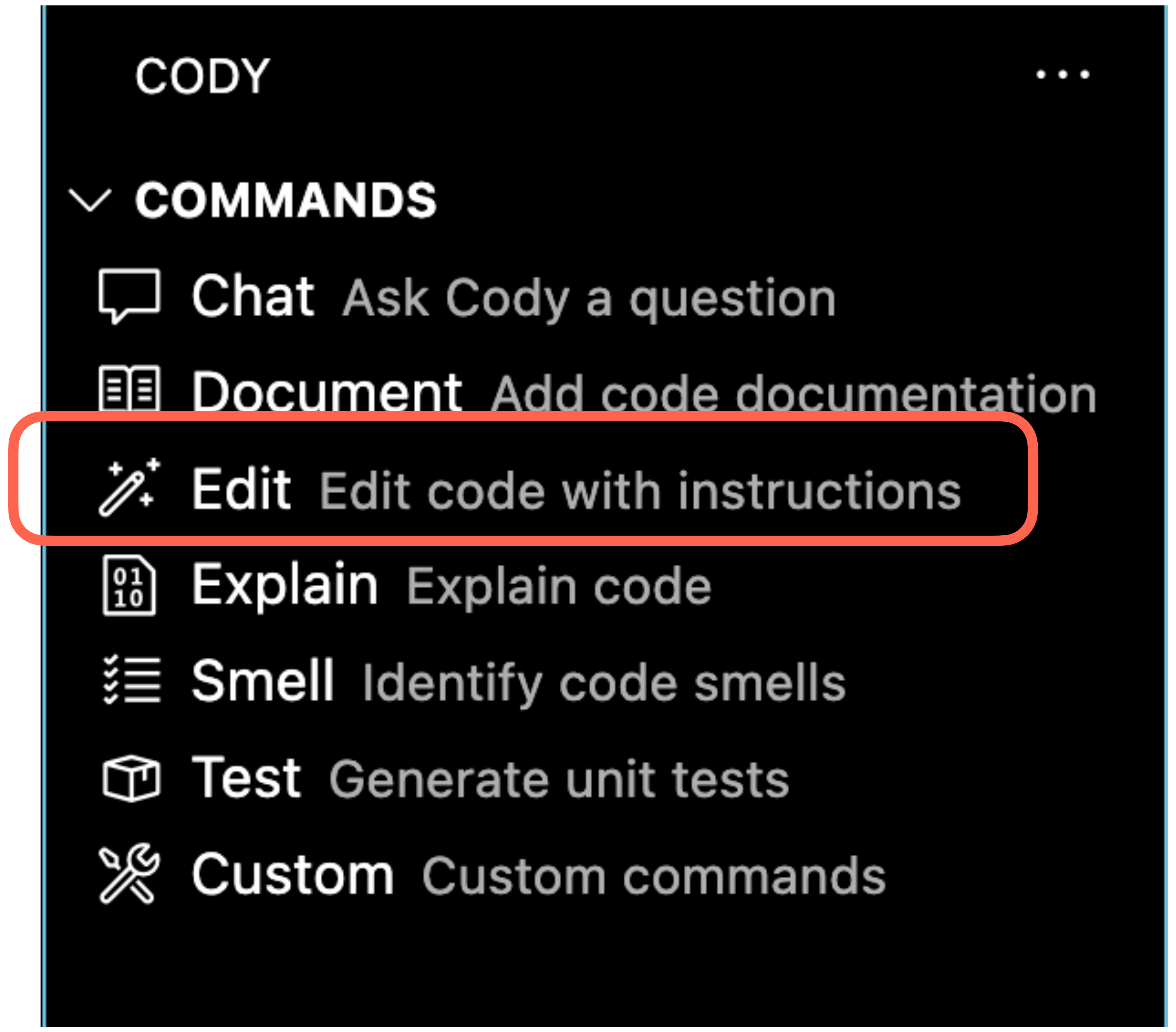
- Create an empty python file, say s1_prompt.py
- On BB, open the “prompt” file.
- Copy and Paste the generation description (top portion)
- Copy and Paste one of the pick_a_number() function, and perform Code Generation using Cody AI
    - Try different description of pick_a_number() function from the template file
    - Compare the code generated

Code Generation - Edit (without selection)



Edit s1_prompt.py:55 with Cody

complete logic for pick_a_number



CODY

COMMANDS

Chat Ask Cody a question

Document Add code documentation

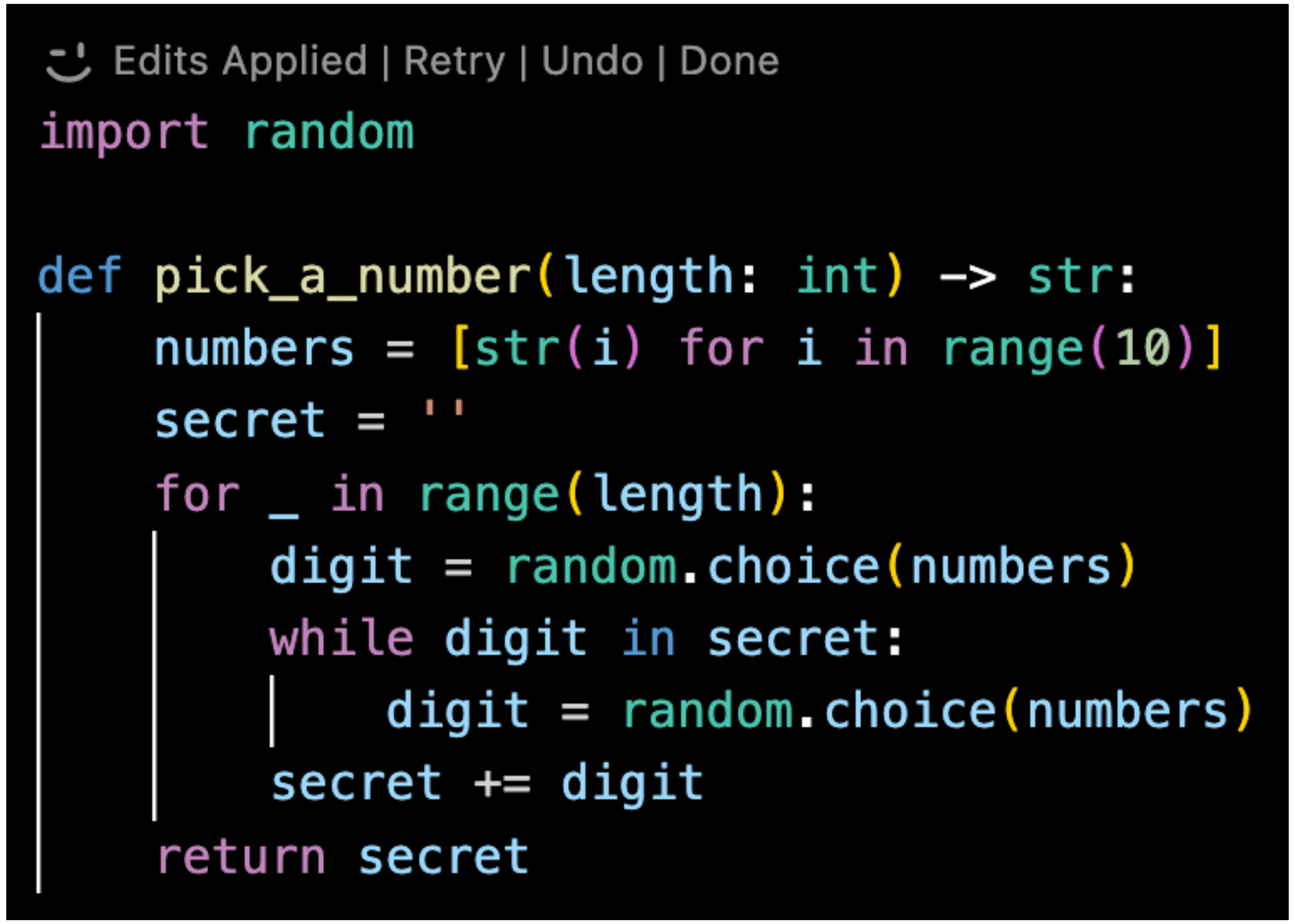
Edit Edit code with instructions

Explain Explain code

Smell Identify code smells

Test Generate unit tests

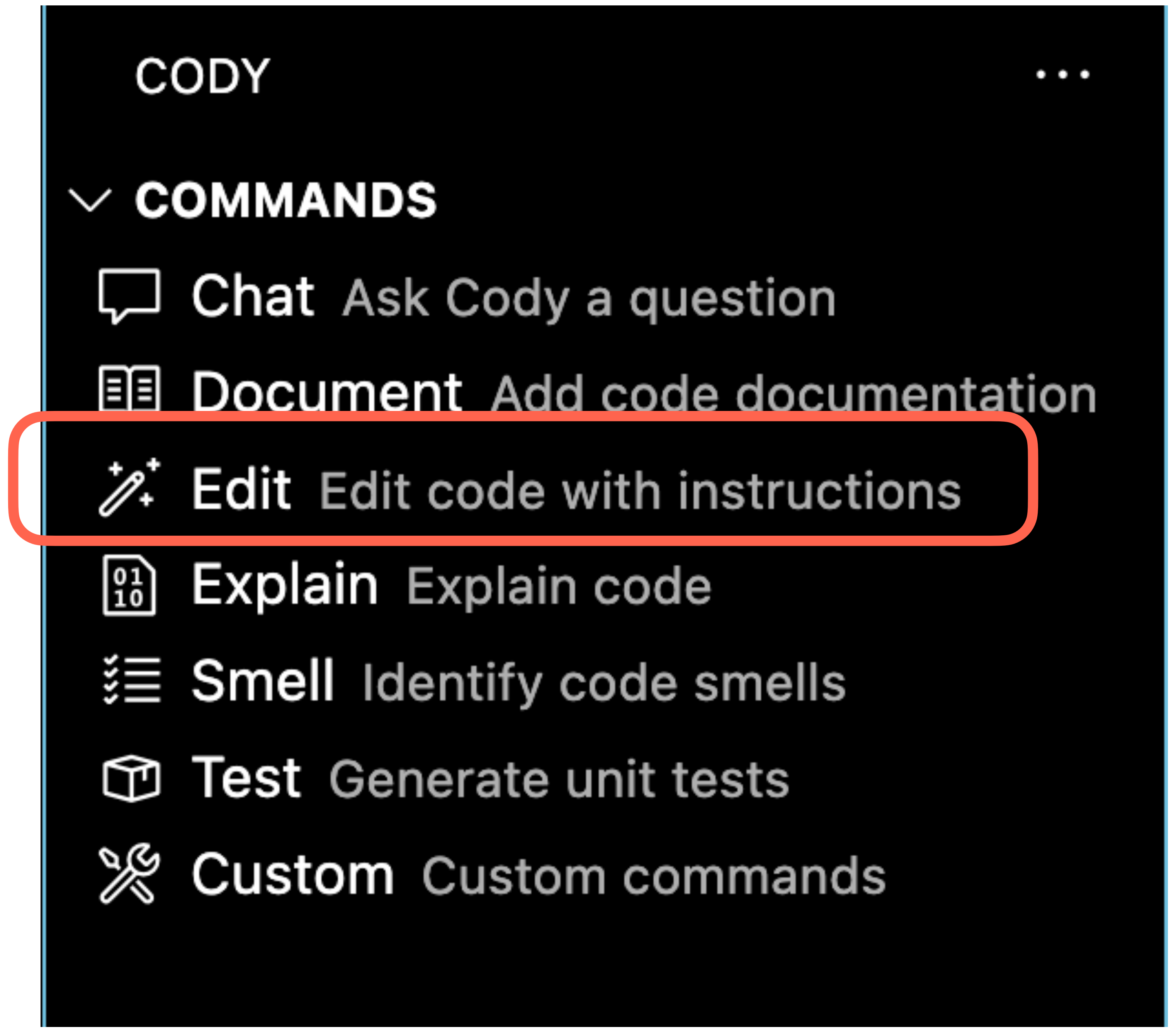
Custom Custom commands

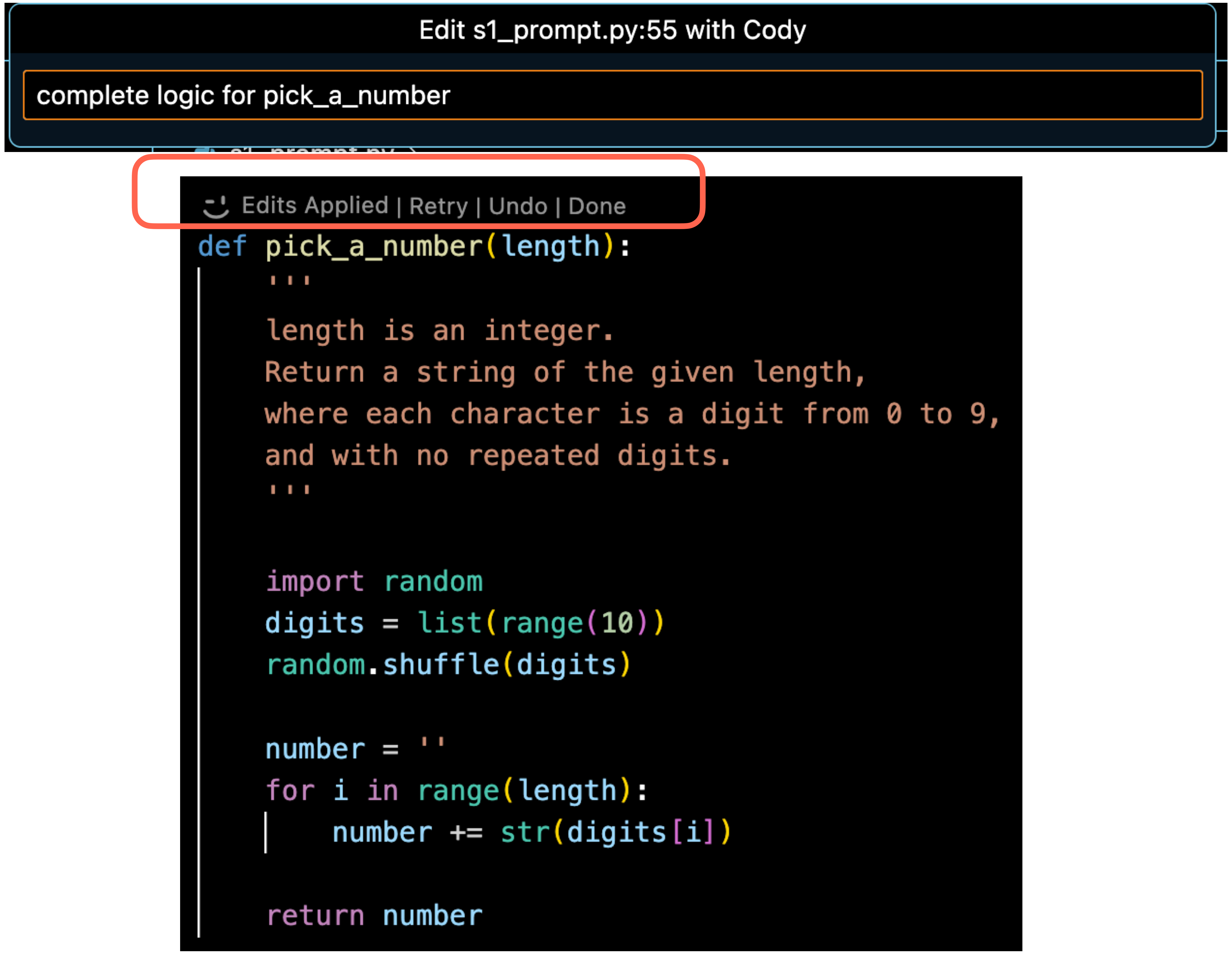


Edits Applied | Retry | Undo | Done

```python
import random

def pick_a_number(length: int) -> str:
    numbers = [str(i) for i in range(10)]
    secret = ''
    for _ in range(length):
        digit = random.choice(numbers)
        while digit in secret:
            digit = random.choice(numbers)
        secret += digit
    return secret
```

Code Generation - Edit (with selection)

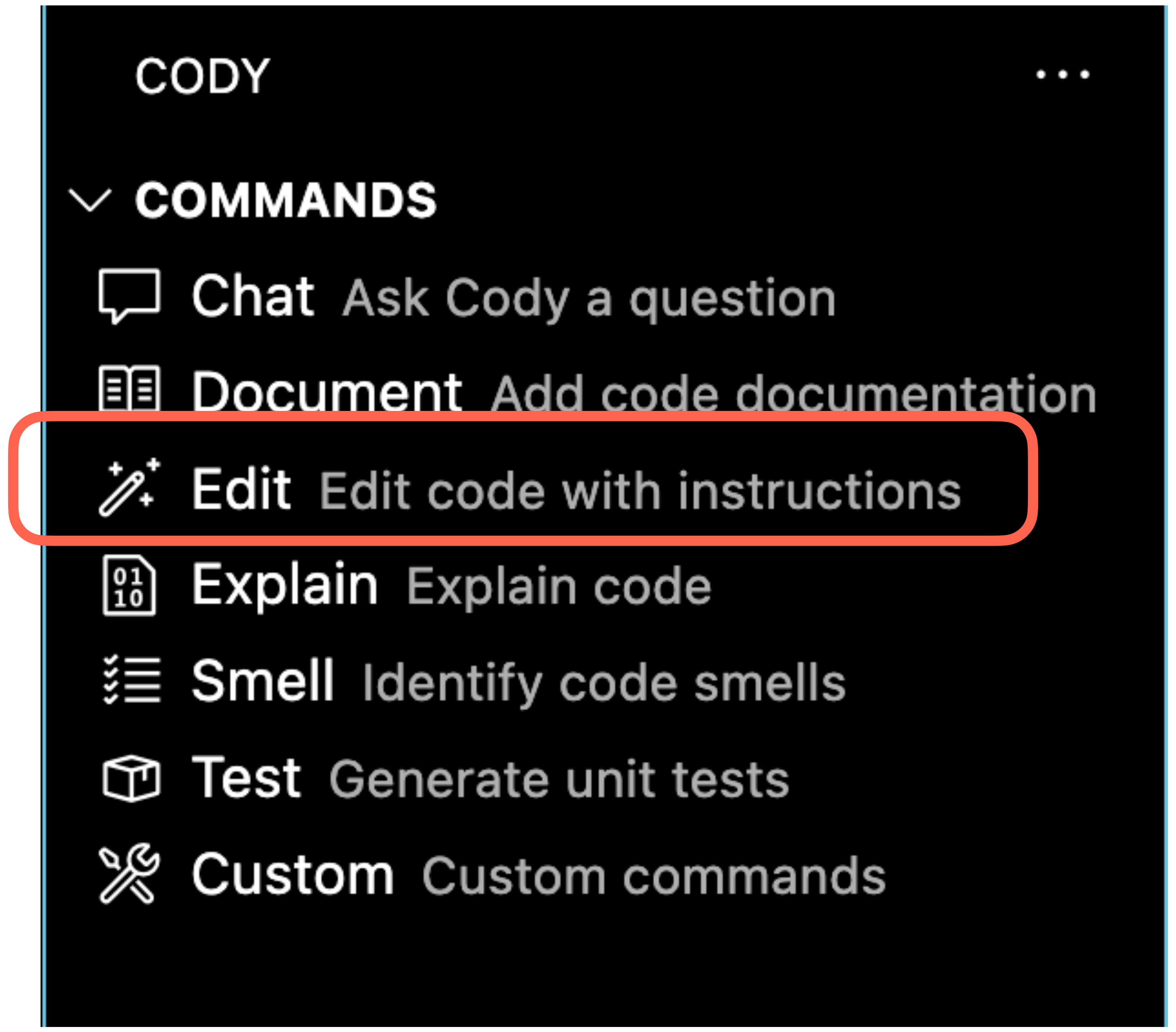


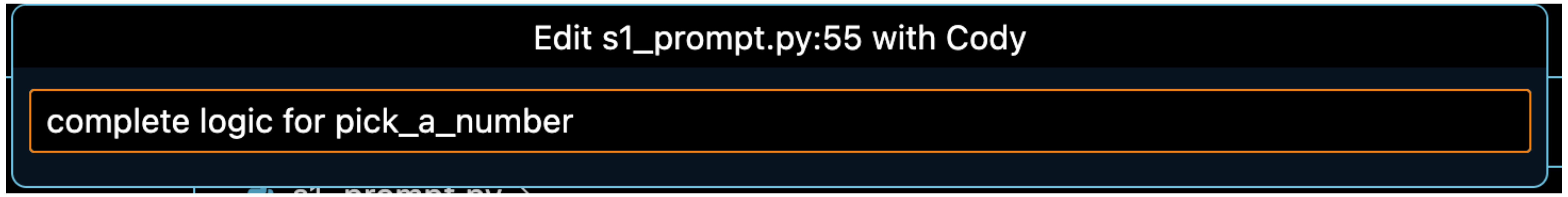

Edit s1_prompt.py:55 with Cody
Edits Applied | Retry | Undo | Done

complete logic for pick_a_number

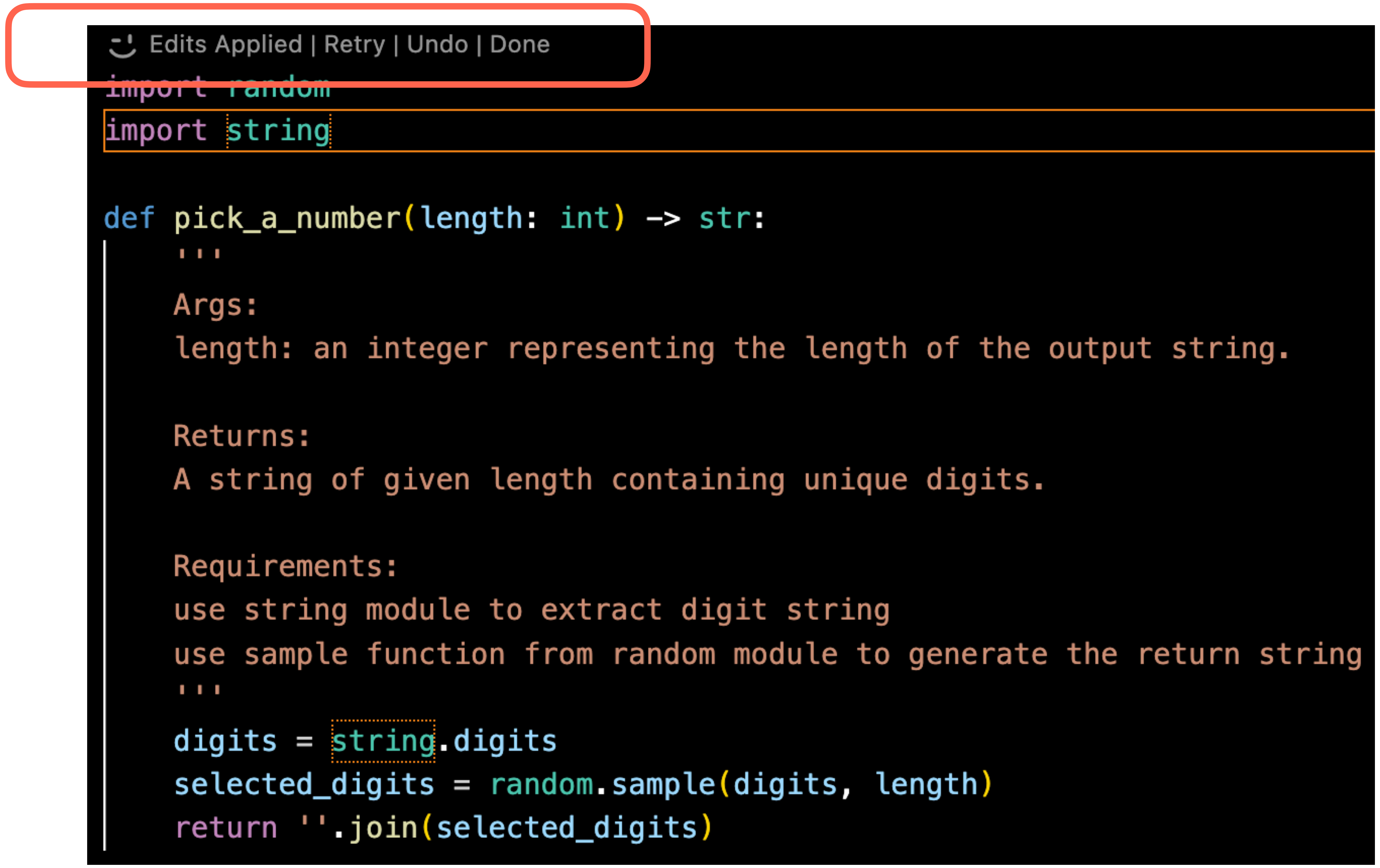
```
def pick_a_number(length):
    '''
    length is an integer.
    Return a string of the given length,
    where each character is a digit from 0 to 9,
    and with no repeated digits.
    '''

    import random
    digits = list(range(10))
    random.shuffle(digits)

    number = ''
    for i in range(length):
        number += str(digits[i])

    return number
```

Code Generation - Edit (with selection)





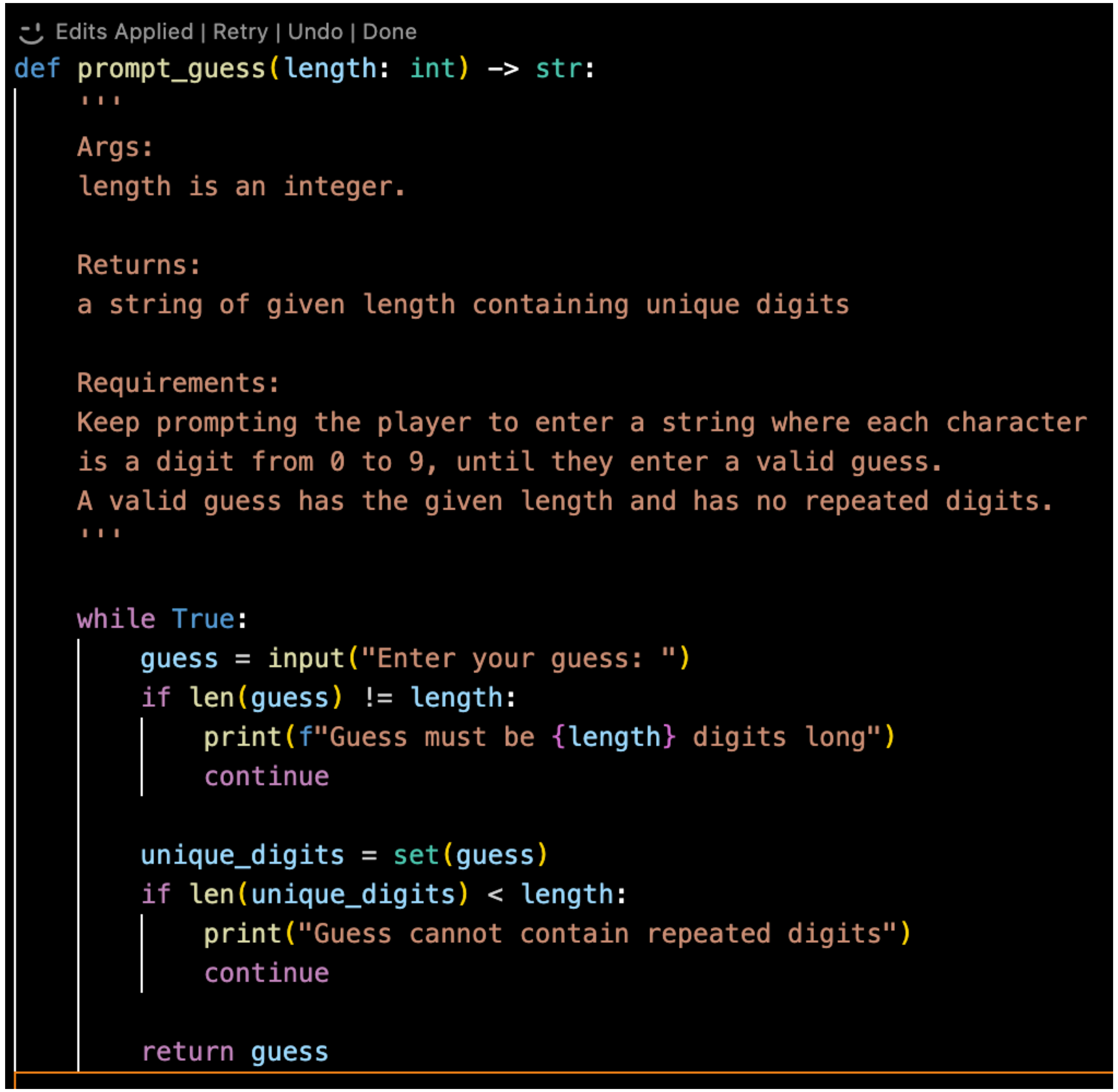
complete logic for pick_a_number



```
import random
import string

def pick_a_number(length: int) -> str:
    '''
    Args:
    length: an integer representing the length of the output string.

    Returns:
    A string of given length containing unique digits.

    Requirements:
    use string module to extract digit string
    use sample function from random module to generate the return string
    '''
    digits = string.digits
    selected_digits = random.sample(digits, length)
    return ''.join(selected_digits)
```

Complete Implementation - AI

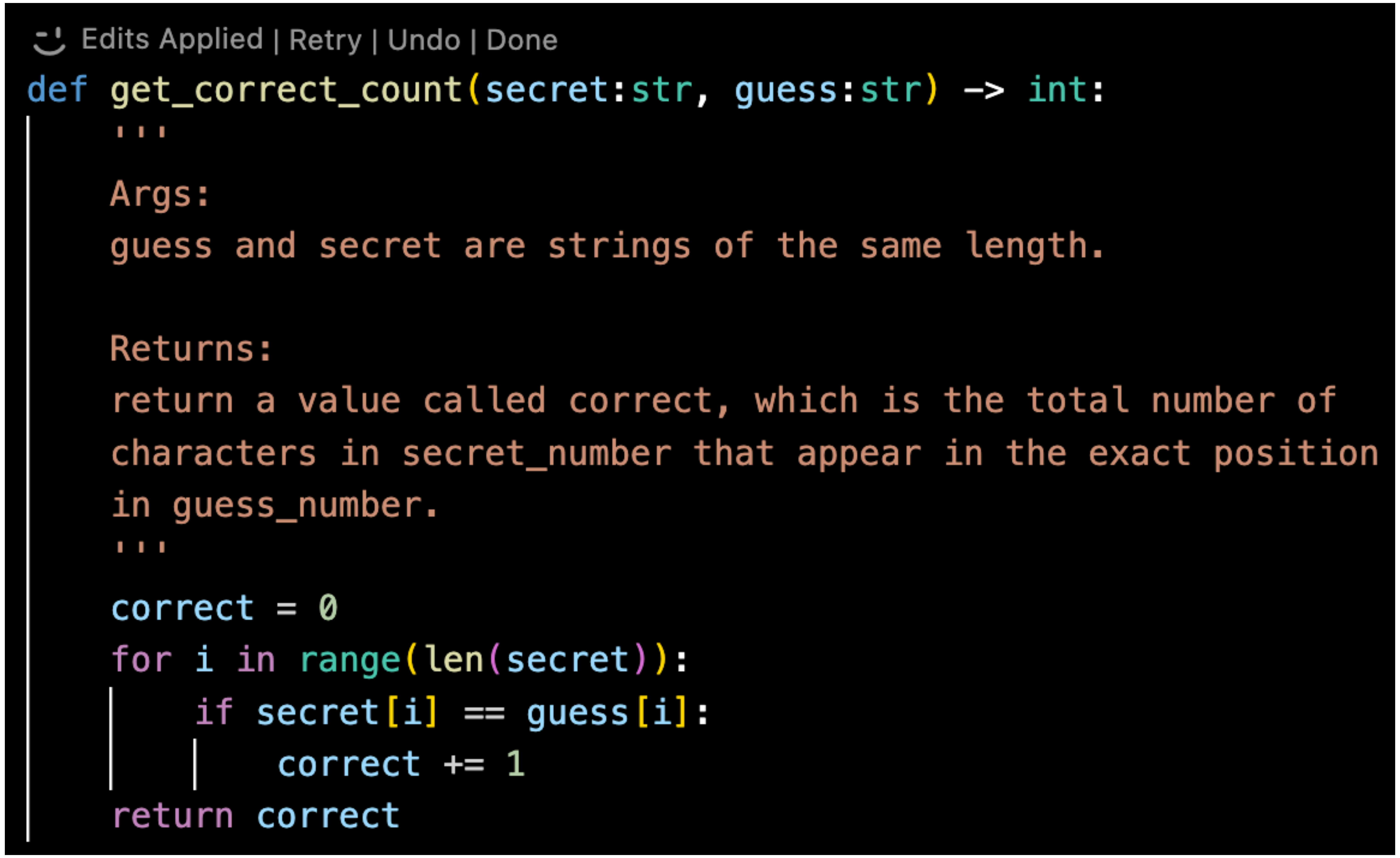
prompt_guess() - Cody AI



```
def prompt_guess(length: int) -> str:
    '''
    Args:
    length is an integer.

    Returns:
    a string of given length containing unique digits

    Requirements:
    Keep prompting the player to enter a string where each character
    is a digit from 0 to 9, until they enter a valid guess.
    A valid guess has the given length and has no repeated digits.
    '''

    while True:
        guess = input("Enter your guess: ")
        if len(guess) != length:
            print(f"Guess must be {length} digits long")
            continue

        unique_digits = set(guess)
        if len(unique_digits) < length:
            print("Guess cannot contain repeated digits")
            continue

        return guess
```

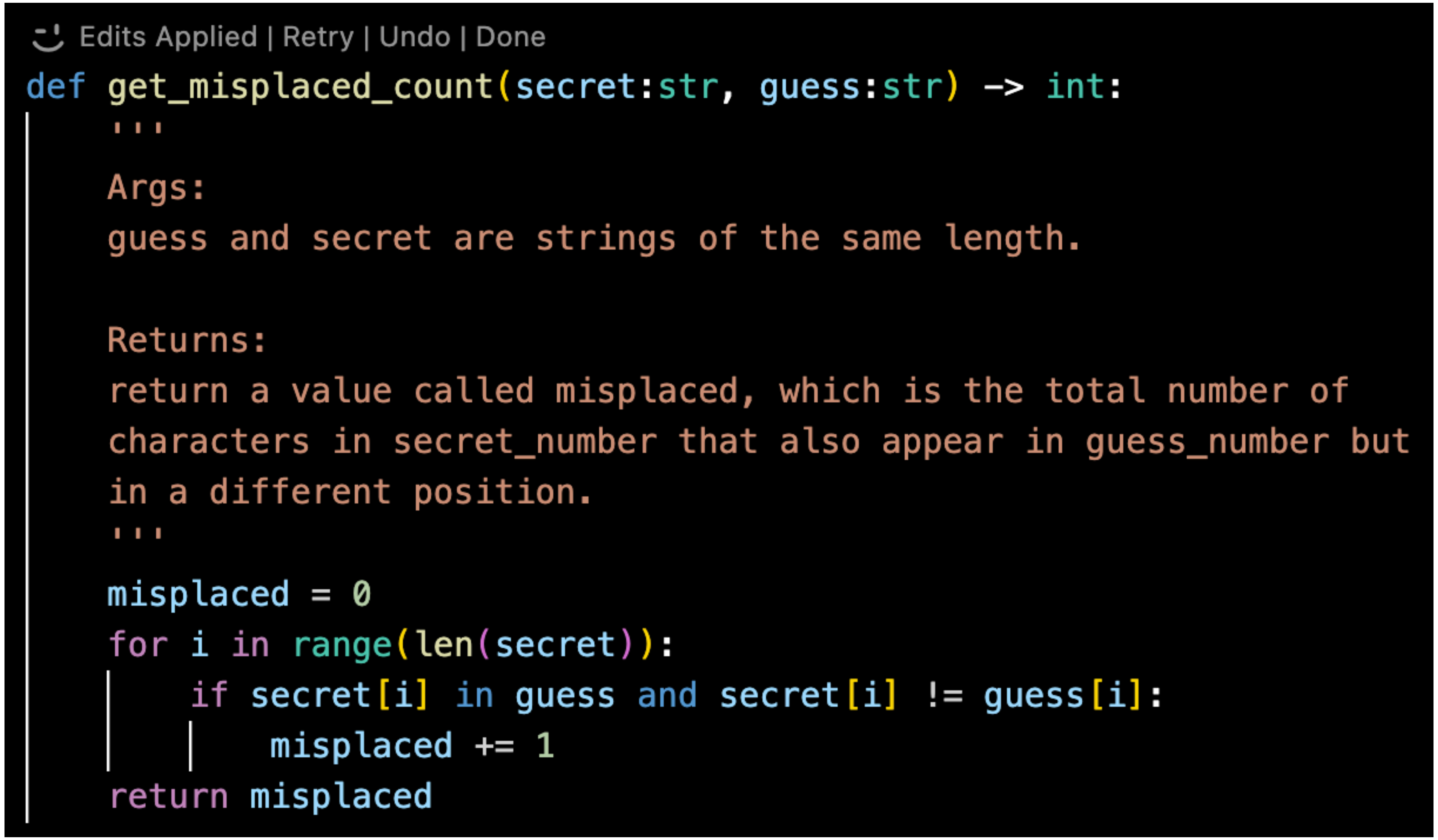
is this correct ???

get_correct_count() - Cody AI



```python
def get_correct_count(secret:str, guess:str) -> int:
    '''
    Args:
    guess and secret are strings of the same length.

    Returns:
    return a value called correct, which is the total number of
    characters in secret_number that appear in the exact position
    in guess_number.
    '''
    correct = 0
    for i in range(len(secret)):
        if secret[i] == guess[i]:
            correct += 1
    return correct
```

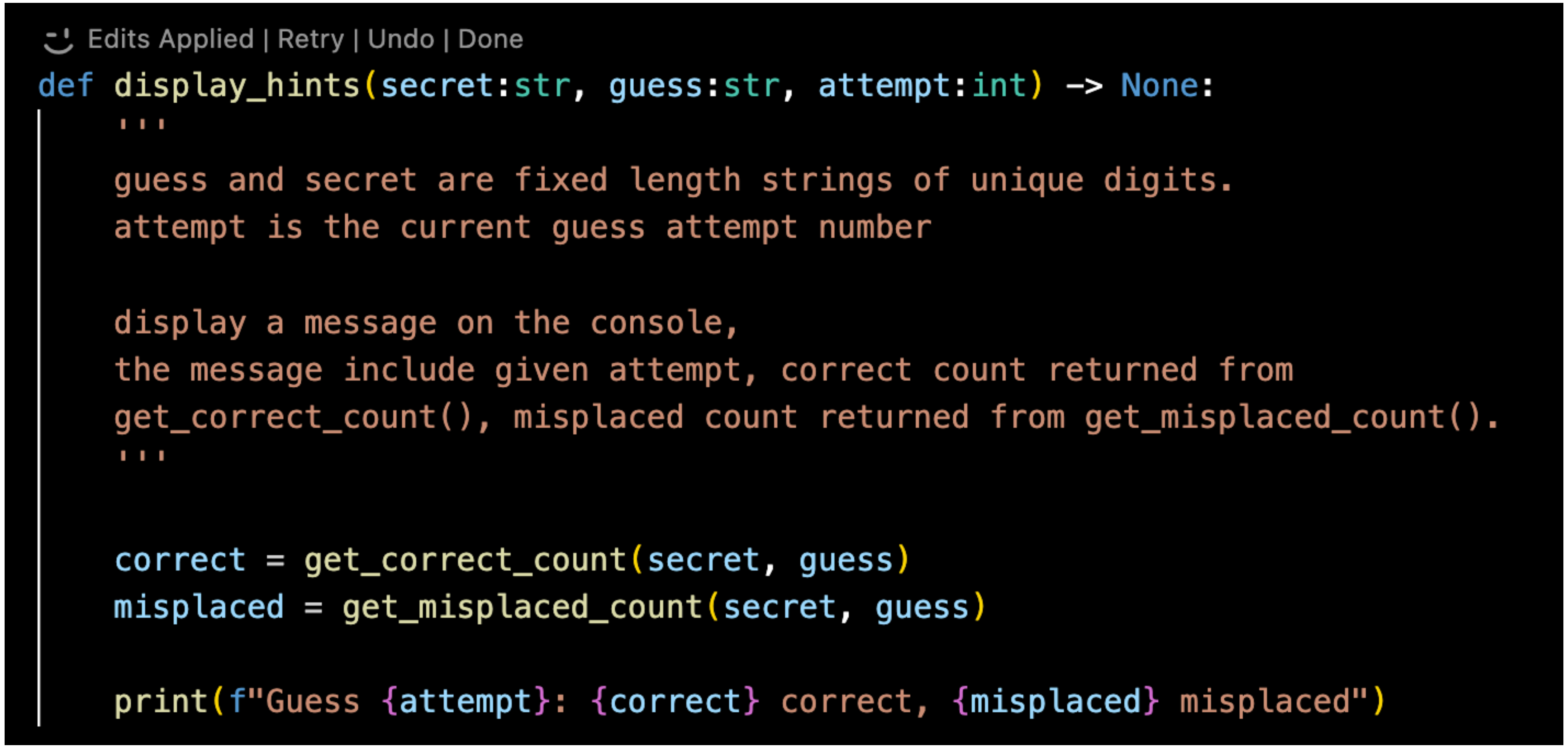
get_misplaced_count() - Cody AI



Edits Applied | Retry | Undo | Done

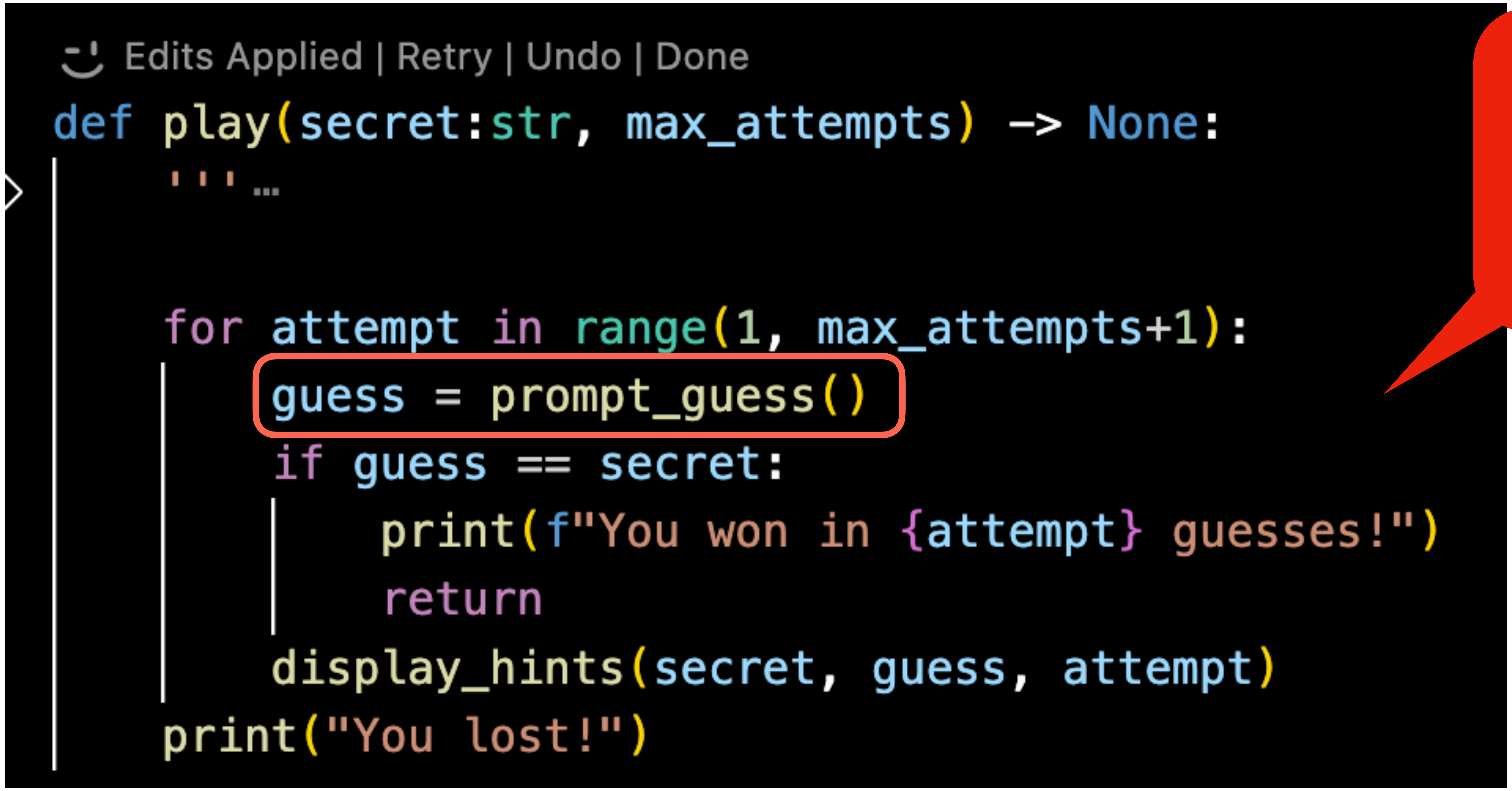
```python
def get_misplaced_count(secret:str, guess:str) -> int:
    '''
    Args:
    guess and secret are strings of the same length.

    Returns:
    return a value called misplaced, which is the total number of
    characters in secret_number that also appear in guess_number but
    in a different position.
    '''
    misplaced = 0
    for i in range(len(secret)):
        if secret[i] in guess and secret[i] != guess[i]:
            misplaced += 1
    return misplaced
```

display_hints() - Cody AI



```python
def display_hints(secret:str, guess:str, attempt:int) -> None:
    '''
    guess and secret are fixed length strings of unique digits.
    attempt is the current guess attempt number

    display a message on the console,
    the message include given attempt, correct count returned from
    get_correct_count(), misplaced count returned from get_misplaced_count().
    '''

    correct = get_correct_count(secret, guess)
    misplaced = get_misplaced_count(secret, guess)

    print(f"Guess {attempt}: {correct} correct, {misplaced} misplaced")
```

play() - Cody AI



Edits Applied | Retry | Undo | Done

```python
def play(secret:str, max_attempts) -> None:
    '''...

    for attempt in range(1, max_attempts+1):
        guess = prompt_guess()
        if guess == secret:
            print(f"You won in {attempt} guesses!")
            return
        display_hints(secret, guess, attempt)
    print("You lost!")
```

is this correct ???

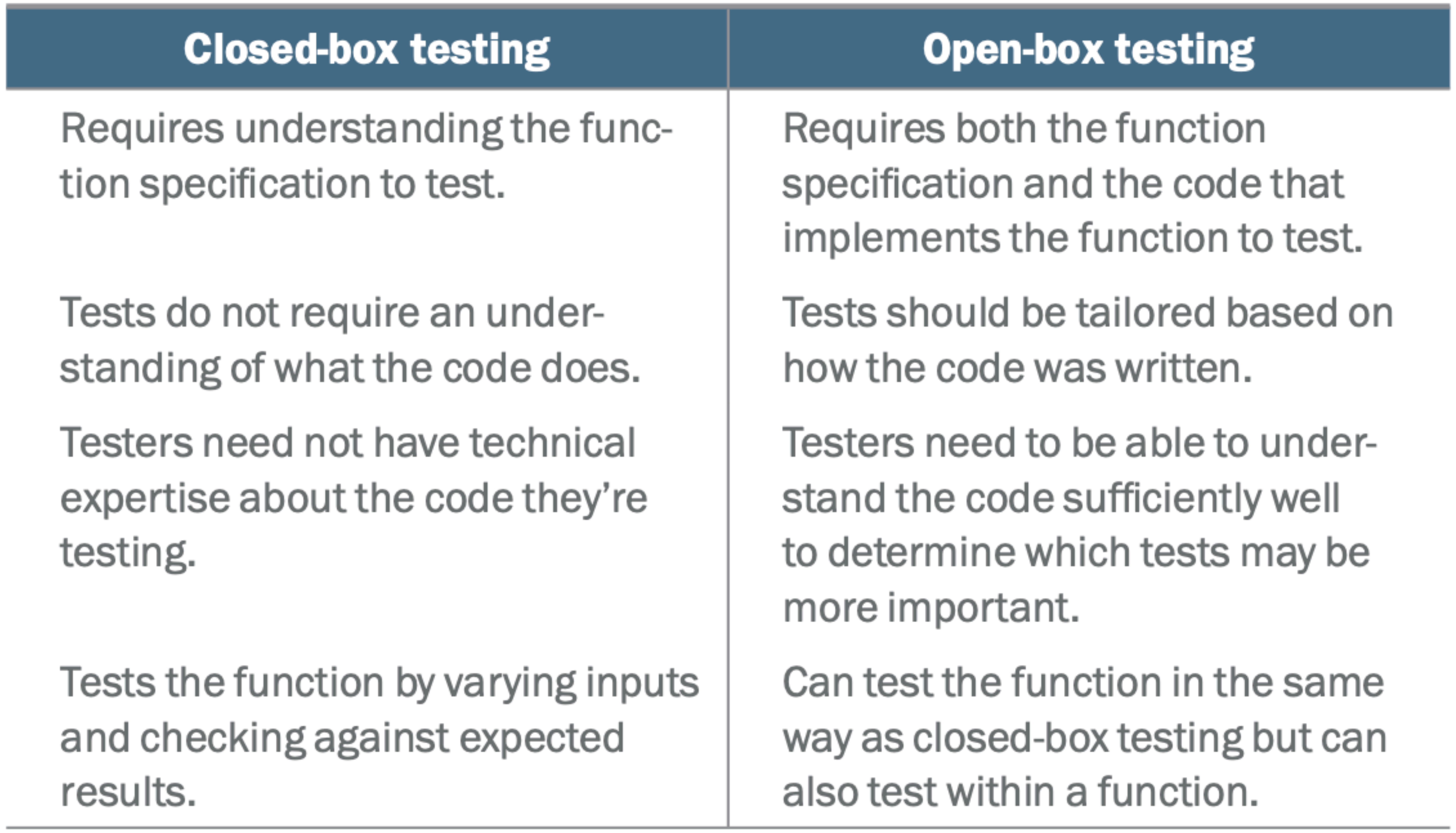
Bonus - Unit Testing

Why Testing

- Without a doubt, testing is essential especially in the software industry (cost, quality, customer satisfaction, sales, support, time, revenue, …etc)
- Types of Testing depends on Size and Scope
- Unit Testing - Small, Fast, Simple (usually on function basis, component basis)
- Integrated - Across all components to ensure end-to-end connectivity under testbed platform
- System/Application - Complete testing with business cases close to production platform
- Others - stress test, performance, benchmark, longetivity, ….etc
- For CSC1002: ONLY Unit Testing

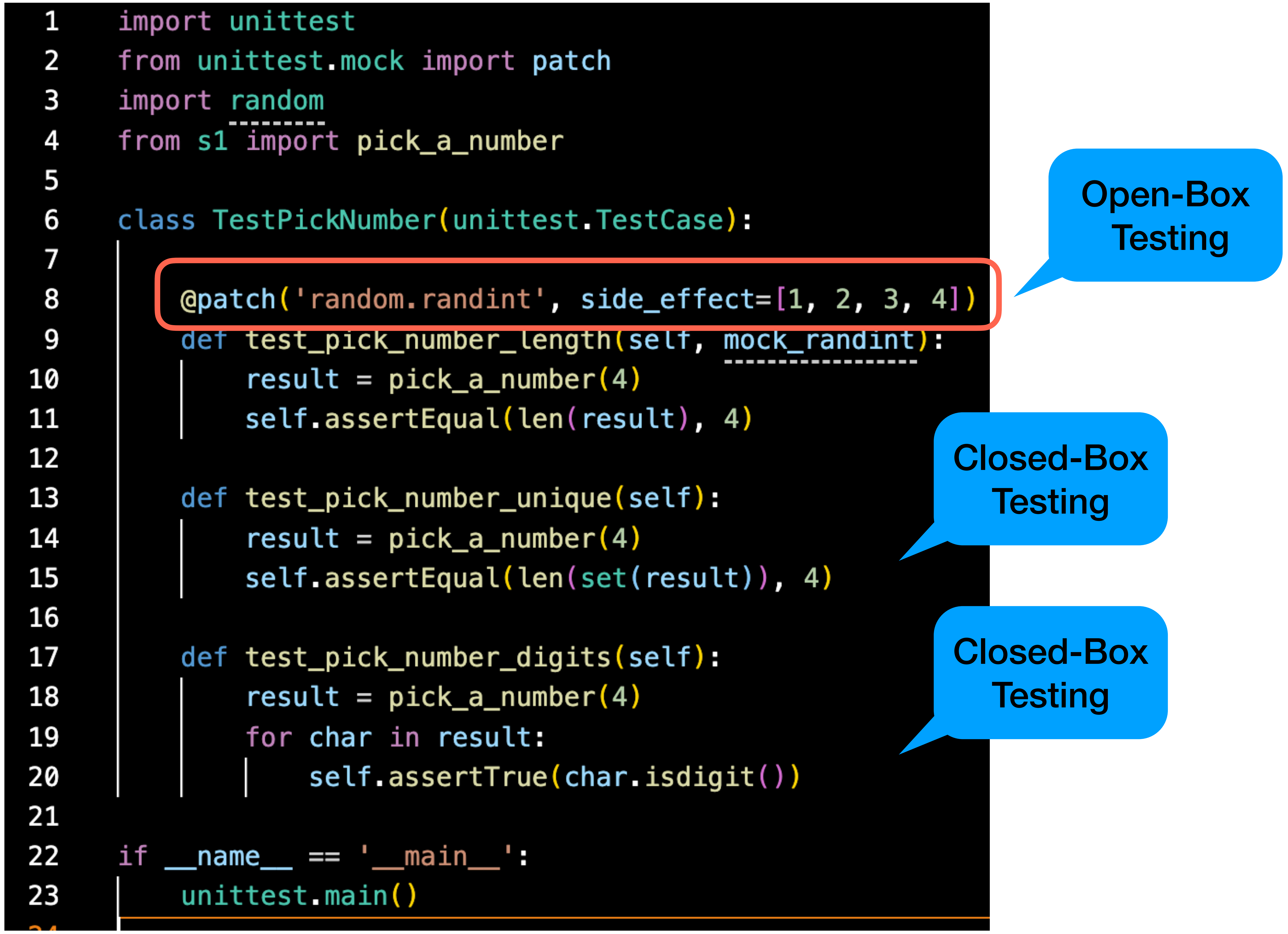
Unit Testing

Closed vs Open



Closed-box testing	Open-box testing
Requires understanding the function specification to test.	Requires both the function specification and the code that implements the function to test.
Tests do not require an understanding of what the code does.	Tests should be tailored based on how the code was written.
Testers need not have technical expertise about the code they're testing.	Testers need to be able to understand the code sufficiently well to determine which tests may be more important.
Tests the function by varying inputs and checking against expected results.	Can test the function in the same way as closed-box testing but can also test within a function.

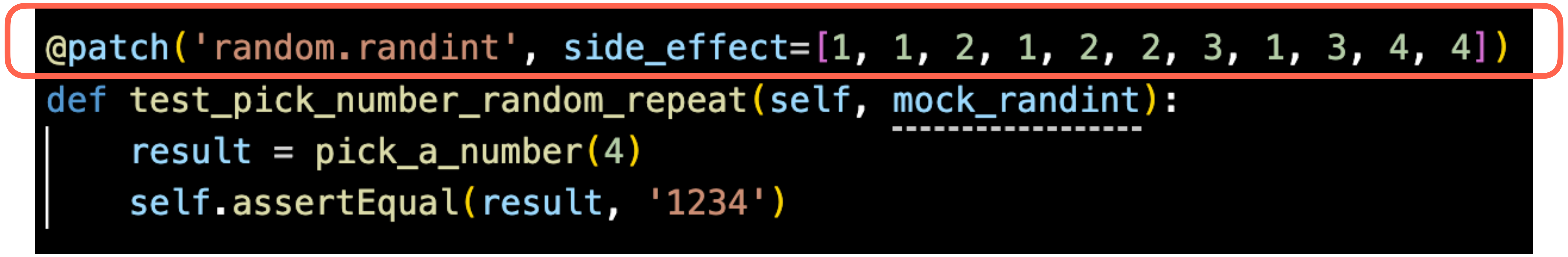
Unit Testing - Examples



```python
import unittest
from unittest.mock import patch
import random
from s1 import pick_a_number

class TestPickNumber(unittest.TestCase):

    @patch('random.randint', side_effect=[1, 2, 3, 4])
    def test_pick_number_length(self, mock_randint):
        result = pick_a_number(4)
        self.assertEqual(len(result), 4)

    def test_pick_number_unique(self):
        result = pick_a_number(4)
        self.assertEqual(len(set(result)), 4)

    def test_pick_number_digits(self):
        result = pick_a_number(4)
        for char in result:
            self.assertTrue(char.isdigit())

if __name__ == '__main__':
    unittest.main()
```

Another Open-Box Testing

To Ensure the Logic within the While loop is correct



```
@patch('random.randint', side_effect=[1, 1, 2, 1, 2, 2, 3, 1, 3, 4, 4])
def test_pick_number_random_repeat(self, mock_randint):
    result = pick_a_number(4)
    self.assertEqual(result, '1234')
```

Unit Testing - Design

- Standard Cases
  - Based on the function’s behaviour/specification
  - Common Inputs with expected results to assure the basic functionality of the function is passed
    - Ex: a function foo() that expects 2 integers as input: foo(+ve, +ve), foo(-ve, +ve), foo(-ve, ve), foo(-ve, -ve), foo(0,0), foo(0,+ve), foo(+ve, 0)
- Exception Cases
  - These are inputs that might test some of the rules for the function in more depth or contain unexpected inputs
    - Out of Range Input, Empty, Invalid Input, Large Value, List with Large Items, …etc

The End - Thank you