# Code Highlighting Test

## JavaScript Example

**JAVASCRIPT**

// This is a comment

function fibonacci(n) {

if (n <= 1) return n;

return fibonacci(n - 1) + fibonacci(n - 2);

}

const result = fibonacci(10);

console.log(`Fibonacci of 10 is: ${result}`);

class Calculator {

constructor() {

this.value = 0;

}

add(x) {

this.value += x;

return this;

}

multiply(x) {

this.value \*= x;

return this;

}

getValue() {

return this.value;

}

}

const calc = new Calculator();

const finalValue = calc.add(5).multiply(3).getValue();

## TypeScript Example

**TYPESCRIPT**

interface User {

id: number;

name: string;

email?: string;

}

type UserRole = 'admin' | 'user' | 'guest';

class UserService {

private users: Map<number, User> = new Map();

async getUser(id: number): Promise<User | null> {

return this.users.get(id) || null;

}

async createUser(user: User): Promise<void> {

this.users.set(user.id, user);

}

}

const service = new UserService();

await service.createUser({ id: 1, name: 'John Doe' });

## Python Example

**PYTHON**

import asyncio

from typing import List, Optional

class DataProcessor:

"""A class to process data asynchronously"""

def \_\_init\_\_(self, batch\_size: int = 100):

self.batch\_size = batch\_size

self.data: List[dict] = []

async def process\_batch(self, items: List[dict]) -> List[dict]:

"""Process a batch of items"""

results = []

for item in items:

# Simulate async processing

await asyncio.sleep(0.01)

results.append({

\*\*item,

'processed': True,

'timestamp': datetime.now()

})

return results

async def run(self):

"""Main processing loop"""

tasks = []

for i in range(0, len(self.data), self.batch\_size):

batch = self.data[i:i + self.batch\_size]

task = asyncio.create\_task(self.process\_batch(batch))

tasks.append(task)

results = await asyncio.gather(\*tasks)

return [item for batch in results for item in batch]

# Usage

processor = DataProcessor(batch\_size=50)

processed\_data = await processor.run()

## SQL Example

**SQL**

-- Create users table

CREATE TABLE users (

id SERIAL PRIMARY KEY,

username VARCHAR(50) UNIQUE NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

-- Complex query with JOIN

SELECT

u.username,

u.email,

COUNT(o.id) as order\_count,

SUM(o.total\_amount) as total\_spent

FROM users u

LEFT JOIN orders o ON u.id = o.user\_id

WHERE u.created\_at >= '2024-01-01'

GROUP BY u.id, u.username, u.email

HAVING COUNT(o.id) > 0

ORDER BY total\_spent DESC

LIMIT 10;

## JSON Example

**JSON**

{

"name": "markdown-docx",

"version": "1.2.0",

"description": "Convert Markdown to DOCX with syntax highlighting",

"scripts": {

"build": "tsup",

"test": "vitest"

},

"dependencies": {

"shiki": "^3.12.2",

"marked": "^15.0.8",

"docx": "^9.3.0"

}

}

## Bash Example

**BASH**

#!/bin/bash

# Function to check if command exists

command\_exists() {

command -v "$1" >/dev/null 2>&1

}

# Install dependencies

if command\_exists npm; then

echo "Installing with npm..."

npm install

elif command\_exists yarn; then

echo "Installing with yarn..."

yarn install

else

echo "No package manager found!"

exit 1

fi

# Build the project

npm run build

# Run tests

npm test -- --coverage

## Plain Text (No Language Specified)

This is plain text without syntax highlighting.

It should still be displayed in a code block format.

But without any color coding.

## Inline Code

Here's some inline code: const x = 42 and function() { return true; }.