Dark Code Highlighting Test

This document tests the dark theme syntax highlighting for various programming languages.

Rust Code

```rust
use std::collections::HashMap;

fn main() {
    let mut scores = HashMap::new();

    scores.insert("Blue", 10);
    scores.insert("Yellow", 50);

    // Print all scores
    for (team, score) in &scores {
        println!("{}: {}", team, score);
    }

    match scores.get("Blue") {
        Some(score) => println!("Blue team score: {}", score),
        None => println!("Blue team not found"),
    }
}
```

JavaScript Code

```
class GameEngine {
    constructor(canvas) {
        this.canvas = canvas;
        this.ctx = canvas.getContext('2d');
        this.entities = [];
        this.running = false;
    }

    start() {
        this.running = true;
        this.gameLoop();
    }

    gameLoop = () => {
        if (!this.running) return;

        this.update();
        this.render();

        requestAnimationFrame(this.gameLoop);
    }

    update() {
        this.entities.forEach(entity => entity.update());
    }

    render() {
        this.ctx.clearRect(0, 0, this.canvas.width, this.canvas.height);
        this.entities.forEach(entity => entity.render(this.ctx));
    }
}
```

Python Code

```python
import asyncio
import aiohttp
from typing import List, Dict, Optional

class DataProcessor:
    def __init__(self, base_url: str):
        self.base_url = base_url
        self.session: Optional[aiohttp.ClientSession] = None

    async def __aenter__(self):
        self.session = aiohttp.ClientSession()
        return self

    async def __aexit__(self, exc_type, exc_val, exc_tb):
        if self.session:
            await self.session.close()

    async def fetch_data(self, endpoint: str) -> Dict:
        """Fetch data from API endpoint"""
        if not self.session:
            raise RuntimeError("Session not initialized")

        url = f"{self.base_url}/{endpoint}"
        async with self.session.get(url) as response:
            return await response.json()

    def process_batch(self, items: List[Dict]) -> List[Dict]:
        """Process a batch of items with list comprehension"""
        return [
            {
                'id': item['id'],
                'name': item['name'].title(),
                'score': item.get('score', 0) * 1.1
            }
            for item in items
            if item.get('active', False)
        ]

# Usage example
async def main():
    async with DataProcessor("https://api.example.com") as processor:
        data = await processor.fetch_data("users")
        processed = processor.process_batch(data['users'])
```

```
        print(f"Processed {len(processed)} items")

if __name__ == "__main__":
    asyncio.run(main())
```

Go Code

```
package main

import (
    "context"
    "fmt"
    "log"
    "net/http"
    "sync"
    "time"
)

type Server struct {
    mu    sync.RWMutex
    data  map[string]interface{}
    port  string
}

func NewServer(port string) *Server {
    return &Server{
        data: make(map[string]interface{}),
        port: port,
    }
}

func (s *Server) handleGet(w http.ResponseWriter, r *http.Request) {
    s.mu.RLock()
    defer s.mu.RUnlock()

    key := r.URL.Query().Get("key")
    if value, exists := s.data[key]; exists {
        fmt.Fprintf(w, "Value: %v\n", value)
    } else {
        http.Error(w, "Key not found", http.StatusNotFound)
    }
}

func (s *Server) Start(ctx context.Context) error {
    mux := http.NewServeMux()
    mux.HandleFunc("/get", s.handleGet)

    server := &http.Server{
        Addr:    ":" + s.port,
        Handler: mux,
```

```
        ReadTimeout:  5 * time.Second,
        WriteTimeout: 5 * time.Second,
    }

    go func() {
        <-ctx.Done()
        server.Shutdown(context.Background())
    }()

    log.Printf("Server starting on port %s", s.port)
    return server.ListenAndServe()
}
```

Inline Code Examples

Here are some `inline code` examples with different languages:

- JavaScript: `const result = await fetch('/api/data')`
- Python: `data = [x**2 for x in range(10)]`
- Rust: `let mut vec = Vec::new()`
- Go: `defer file.Close()`

Complex Nested Code

```
interface GameState {
    players: Player[];
    currentTurn: number;
    board: Cell[][];
}

class GameManager<T extends GameState> {
    private state: T;
    private observers: Array<(state: T) => void> = [];

    constructor(initialState: T) {
        this.state = { ...initialState };
    }

    subscribe(callback: (state: T) => void): () => void {
        this.observers.push(callback);
        return () => {
            const index = this.observers.indexOf(callback);
            if (index > -1) {
                this.observers.splice(index, 1);
            }
        };
    }

    private notify(): void {
        this.observers.forEach(observer => observer(this.state));
    }

    updateState(updates: Partial<T>): void {
        this.state = { ...this.state, ...updates };
        this.notify();
    }
}
```

This test document should showcase the improved dark theme syntax highlighting with better contrast and readability.