```julia
# Function with optional argument
function greet(name="User")
    println("Hello, $name")
end

# struct and its usage
struct Point{T}
    x::T
    y::T
end
p1 = Point(1.0, 3.7)
println("Point: [$(p1.x), $(p1.y)]")

# Array and array comprehension
numbers = [1, 2, 3, 4, 5]
squared_numbers = [x^2 for x in numbers]

# Dictionaries and their usage
person = Dict("name" => "John", "age" => 30)
for (key, value) in person
    println("$key: $value")
end

# Try-catch block
try
    result = 10 / 0
catch e
    result = "Error: $e"
end

# While loop with break statement
i = 1
while i <= 5
    if i == 3
        break
    end
    println(i)
    i += 1
end

# Anonymous function (lambda) and map function
double = x -> 2x
numbers = [1, 2, 3, 4, 5]
doubled_numbers = map(double, numbers)

# Set data structure
fruits = Set(["apple", "banana", Nothing])

# Tuple unpacking and multi-line string
(x, y) = (10, 20)
multi_line_string = """
This is a multi-line
string in Julia.
"""
```

Listing 1: Example - Julia (`TreeSitter` powered highlighting)

```julia
# Function with optional argument
function greet(name="User")
    println("Hello, $name")
end

# struct and its usage
struct Point{T}
    x::T
    y::T
end
p1 = Point(1.0, 3.7)
println("Point: [$(p1.x), $(p1.y)]")

# Array and array comprehension
numbers = [1, 2, 3, 4, 5]
squared_numbers = [x^2 for x in numbers]

# Dictionaries and their usage
person = Dict("name" => "John", "age" => 30)
for (key, value) in person
    println("$key: $value")
end

# Try-catch block
try
    result = 10 / 0
catch e
    result = "Error: $e"
end

# While loop with break statement
i = 1
while i <= 5
    if i == 3
        break
    end
    println(i)
    i += 1
end

# Anonymous function (lambda) and map function
double = x -> 2x
numbers = [1, 2, 3, 4, 5]
doubled_numbers = map(double, numbers)

# Set data structure
fruits = Set(["apple", "banana", Nothing])

# Tuple unpacking and multi-line string
(x, y) = (10, 20)
multi_line_string = """
This is a multi-line
string in Julia.
"""
```

Listing 1: Example - Julia (`minted` powered highlighting)

```python
from math import cos
# Class with magic methods and inheritance
class Shape:
    """
    This is doc comment for Shape with some escape sequence \t\t in it.
    Second line here.
    """
    def __init__(self, size: int, color: str = "black"):
        self.color = color
        self.size = abs(size)
        print(f"creating {self.color} shape of size: ", self.size)
        if self.size < 5:
            print("shape is quite small")

    def __str__(self) -> str:
        return f"{self.color} shape" # here some direct string formatting

    def truthy(self, a: int) -> bool:
        print(f"foo\t{a}\tbar") # here some more string shenanigans
        return True

    def is_larger_than(self, other: "Shape") -> bool:
        # yes, this could be single line
        if self.size > other.size:
            return True
        else:
            return False

def foo(i: int = 5) -> None:
    """
    Here is some random function with comment string
    """
    SOME_CONSTANT = 42 # constants are recognized too!
    print(f"the constant is {SOME_CONSTANT}.", end="\n\n")
    print("input i: ", i)
    return None

foo(8) # actually call the function
circle = Shape(size=3, color="red") # call some constructors
square = Shape(size=5)
if circle.is_larger_than(square):
    print(f"the {circle.color} circle is larger than {square.color} square")
else:
    print(f"the {circle.color} circle is smaller than {square.color} square")
```

Listing 2: Example - Python (TreeSitter powered highlighting)

```python
from math import cos
# Class with magic methods and inheritance
class Shape:
    """
    This is doc comment for Shape with some escape sequence \t\t in it.
    Second line here.
    """
    def __init__(self, size: int, color: str = "black"):
        self.color = color
        self.size = abs(size)
        print(f"creating {self.color} shape of size: ", self.size)
        if self.size < 5:
            print("shape is quite small")

    def __str__(self) -> str:
        return f"{self.color} shape" # here some direct string formatting

    def truthy(self, a: int) -> bool:
        print(f"foo\t{a}\tbar") # here some more string shenanigans
        return True

    def is_larger_than(self, other: "Shape") -> bool:
        # yes, this could be single line
        if self.size > other.size:
            return True
        else:
            return False

def foo(i: int = 5) -> None:
    """
    Here is some random function with comment string
    """
    SOME_CONSTANT = 42 # constants are recognized too!
    print(f"the constant is {SOME_CONSTANT}.", end="\n\n")
    print("input i: ", i)
    return None

foo(8) # actually call the function
circle = Shape(size=3, color="red") # call some constructors
square = Shape(size=5)
if circle.is_larger_than(square):
    print(f"the {circle.color} circle is larger than {square.color} square")
else:
    print(f"the {circle.color} circle is smaller than {square.color} square")
```

Listing 2: Example - Python (`minted` powered highlighting)

```rust
use std::collections::HashMap;
use std::ops::{Add, Mul};

#[derive(Debug)]
struct Point<T> {
    x: T,
    y: T,
}

impl<T> Point<T> {
    fn new(x: T, y: T) -> Self {
        Point { x, y }
    }
}

#[derive(Debug)]
enum Shape {
    Circle(f64),
    Rectangle(f64, f64),
    Triangle(f64, f64, f64),
}

fn area(shape: Shape) -> f64 {
    match shape {
        Shape::Circle(radius) => std::f64::consts::PI * radius.powi(2),
        Shape::Rectangle(width, height) => width * height,
        Shape::Triangle(a, b, c) => {
            let s = (a + b + c) / 2.0;
            (s * (s - a) * (s - b) * (s - c)).sqrt()
        }
    }
}

fn main() {
    let mut scores = HashMap::new();
    scores.insert("Alice", 95);
    scores.insert("Charlie", 85);
    println!("Scores: {:?}", scores); // here is some comment

    let point = Point::new(10, 20);
    println!("Point: {:?}", point);

    let circle = Shape::Circle(5.0);
    let rectangle = Shape::Rectangle(4.0, 6.0);
    let triangle = Shape::Triangle(3.0, 4.0, 5.0);

    println!("Circle area: {}", area(circle));
    println!("Rectangle area: {}", area(rectangle));

    let numbers = [1, 2, 3, 4, 5];
    println!("Sum: {}", numbers.iter().sum::<i32>());
}
```

Listing 3: Example - Rust (`TreeSitter` powered highlighting)

```rust
use std::collections::HashMap;
use std::ops::{Add, Mul};

#[derive(Debug)]
struct Point<T> {
    x: T,
    y: T,
}

impl<T> Point<T> {
    fn new(x: T, y: T) -> Self {
        Point { x, y }
    }
}

#[derive(Debug)]
enum Shape {
    Circle(f64),
    Rectangle(f64, f64),
    Triangle(f64, f64, f64),
}

fn area(shape: Shape) -> f64 {
    match shape {
        Shape::Circle(radius) => std::f64::consts::PI * radius.powi(2),
        Shape::Rectangle(width, height) => width * height,
        Shape::Triangle(a, b, c) => {
            let s = (a + b + c) / 2.0;
            (s * (s - a) * (s - b) * (s - c)).sqrt()
        }
    }
}

fn main() {
    let mut scores = HashMap::new();
    scores.insert("Alice", 95);
    scores.insert("Charlie", 85);
    println!("Scores: {:?}", scores); // here is some comment

    let point = Point::new(10, 20);
    println!("Point: {:?}", point);

    let circle = Shape::Circle(5.0);
    let rectangle = Shape::Rectangle(4.0, 6.0);
    let triangle = Shape::Triangle(3.0, 4.0, 5.0);

    println!("Circle area: {}", area(circle));
    println!("Rectangle area: {}", area(rectangle));

    let numbers = [1, 2, 3, 4, 5];
    println!("Sum: {}", numbers.iter().sum::<i32>());
}

```

Listing 3: Example - Rust (`minted` powered highlighting)