

PARADYGMATY PROGRAMOWANIA

CZY ISTNIEJE NAJLEPSZY?

@akashivskyy

PARADYGMATY PROGRAMOWANIA

SPOSÓB PATRZENIA NA PRZEPŁYW STEROWANIA I WYKONYWANIE PROGRAMU

1. PROGRAMOWANIE OBIEKTOWE

PROGRAM DEFINIUJĄ **OBIEKTY**
ŁĄCZĄCE **STAN** I **ZACHOWANIE**

3 ZAŁOŻENIA

1. ABSTRAKCJA
2. ENKAPSULACJA
3. DZIEDZICZNOŚĆ

1. ABSTRAKCJA
2. ENKAPSULACJA
3. DZIEDZICZNOŚĆ



```
protocol Shape {  
    var area: Double  
}
```

```
func printShapeArea(shape: Shape) {  
    println("area = \(shape.area)")  
}
```

```
struct Square: Shape {  
    let side: Double  
    let area: Double {  
        return side * side  
    }  
}
```

```
printShapeArea(Square(side: 4)) // 16.0
```

```
struct Circle: Shape {  
    var radius: Double  
    var area: Double {  
        return M_PI * radius * radius  
    }  
}
```

```
printShapeArea(Circle(radius: 2)) // 12.56
```

```
struct Plane: Shape {  
    var area: Double {  
        return Double.infinity  
    }  
}
```

```
printShapeArea(Plane()) // infinity
```

~~1. ABSTRAKCJA~~

2. ENKAPSULACJA

3. DZIEDZICZNOŚĆ

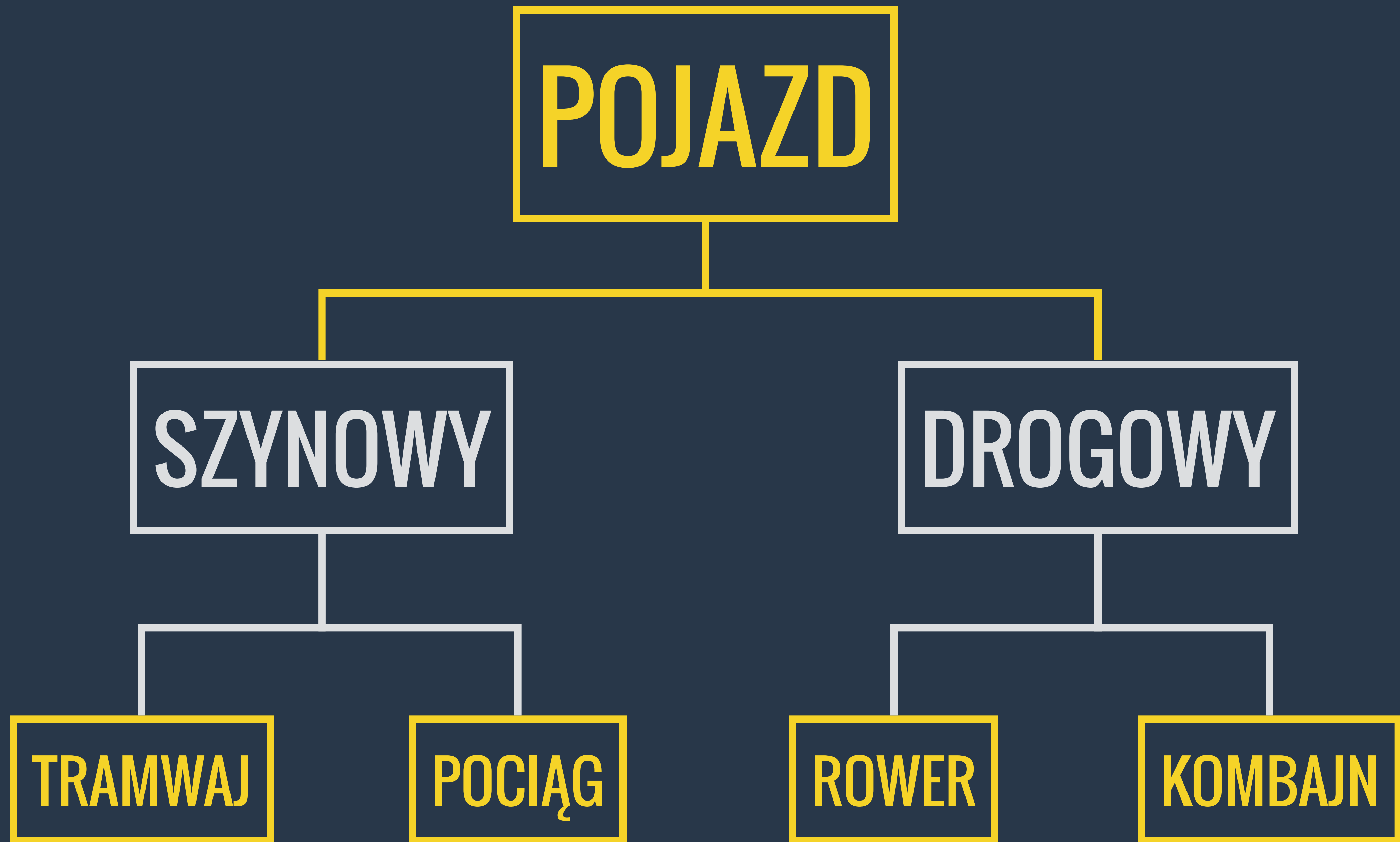
```
class EncryptionAssistant {  
    private var key = "420m1g$crub"  
    public func encrypt(pass: String) -> String {  
        return rsaEncrypt(pass, key)  
    }  
}
```

```
let assistant = EncryptionAssistant()  
assistant.encrypt("secret") // 1L100Myn4RtY  
assistant.key // compile error!
```


~~1. ABSTRAKCJA~~

~~2. ENKAPSULACJA~~

3. DZIEDZICZNOŚĆ



```
class Car {  
    var color: String = "red"  
    var name: String {  
        return "\($color) car"  
    }  
}
```

```
class BlueCar: Car {  
    override var color = "blue"  
}
```

```
Car().name // red car
```

```
BlueCar().name // blue car
```

2. PROGRAMOWANIE IMPERATYWNE

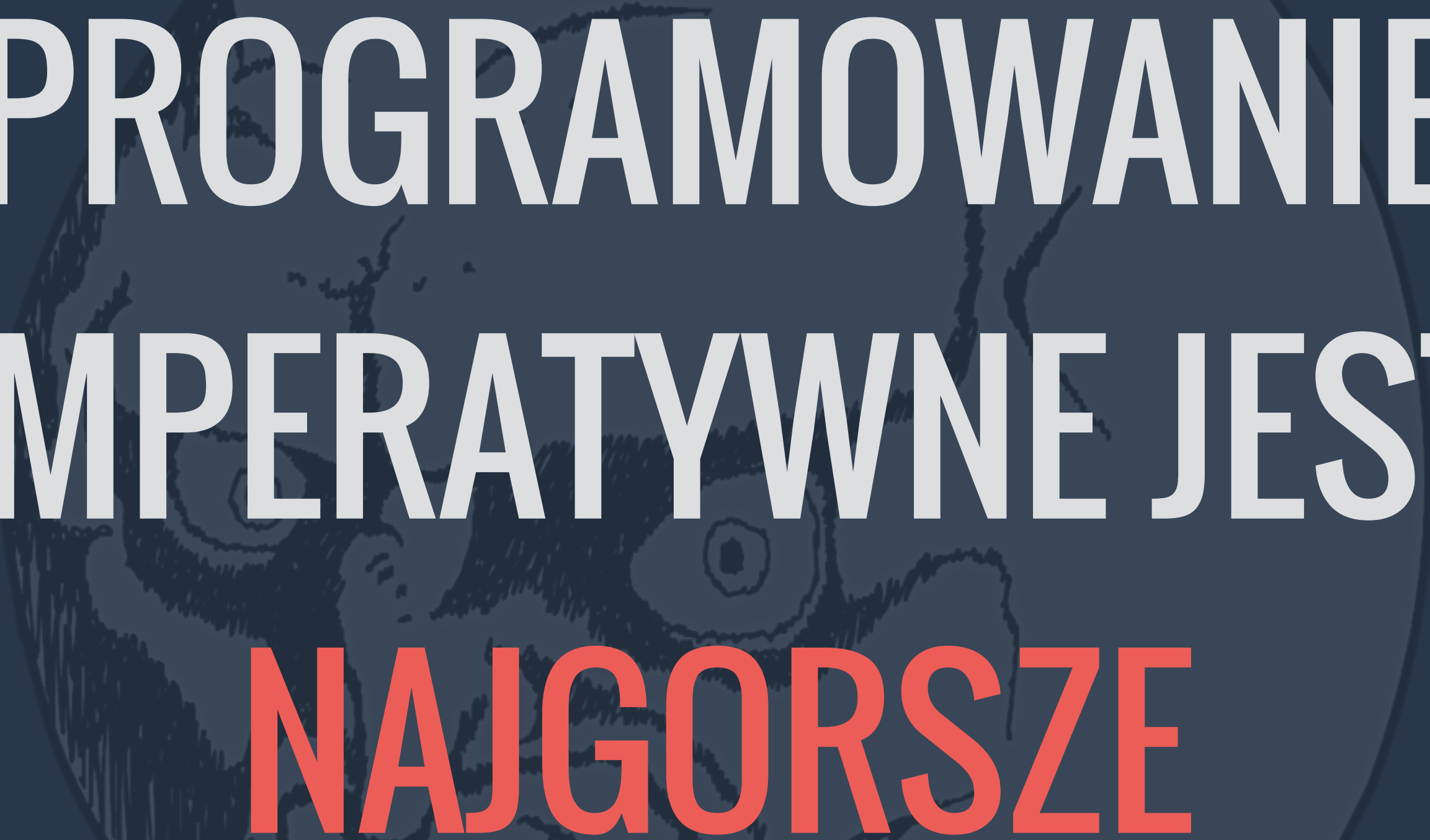
ZDANIA ROZKAZUJĄCE,
ZMIENIAJĄCE GLOBALNY STAN
PROGRAMU

```
let numbers = [1, 2, 3, 4, 5, 6]
var sum = 0
var odds: [Int] = []
for number in numbers {
    sum += number
    if number % 2 == 1 {
        odds.append(number)
    }
}
```

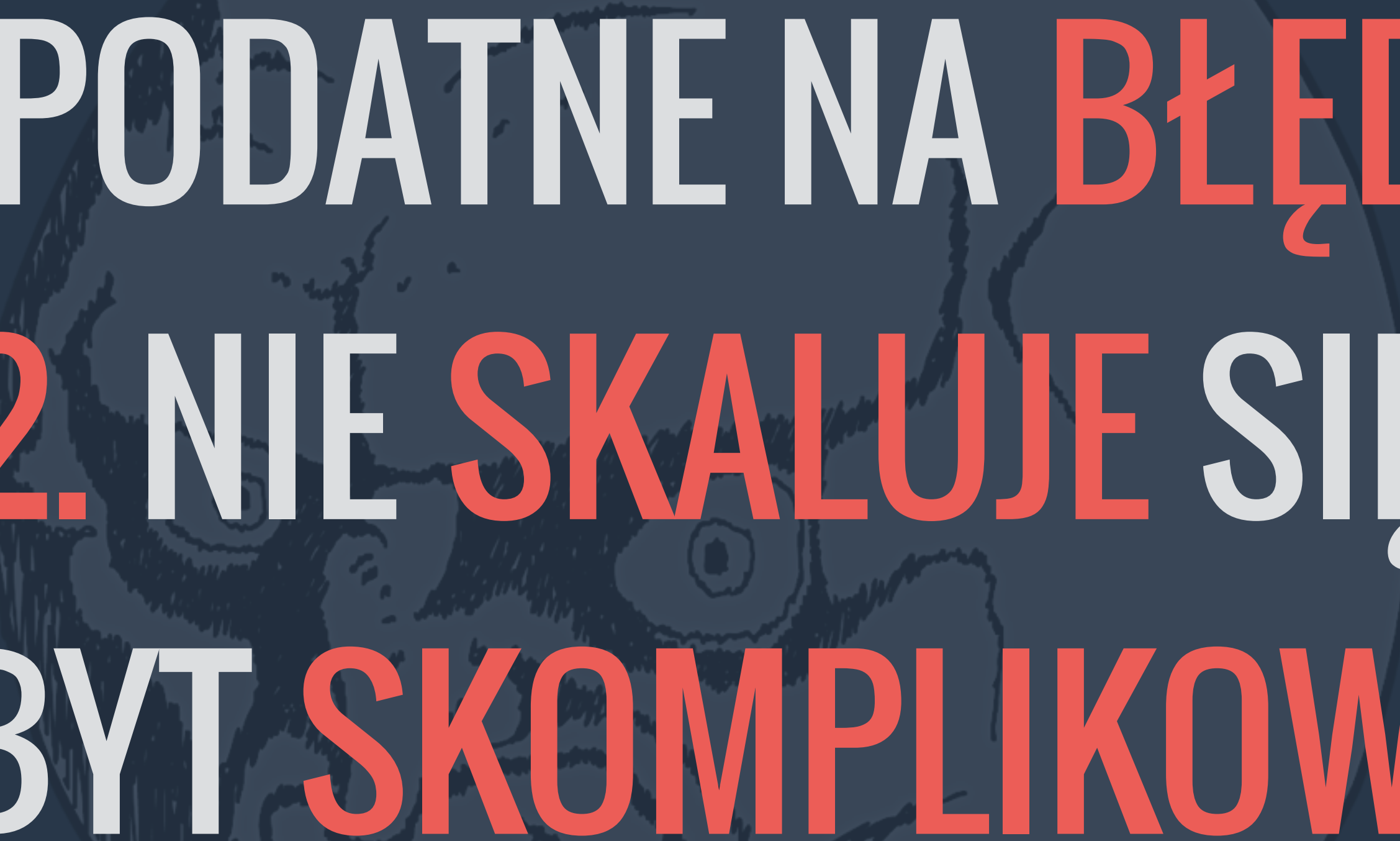
```
getRemoteData("url", { data, error in
    if error == nil {
        parseData(data, { parsed, error in
            if error == nil {
                handleParsedData(parsed)
            } else {
                displayError(error)
            }
        })
    } else {
        displayError(error)
    }
})
```


**PROGRAMOWANIE
IMPERATYWNE JEST
NAJPOPULARNIEJSZE**

**PROGRAMOWANIE
IMPERATYWNE JEST
NAJŁATWIEJSZE**



**PROGRAMOWANIE
IMPERATYWNE JEST
NAJGORSZE**

- 
1. PODATNE NA **BŁĘDY**
 2. **NIE SKALUJE SIĘ**
 3. **ZBYT SKOMPLIKOWANE**

```
getRemoteData("example.com", { data, error in
    if error == nil {
        parseData(data, { parsed, error in
            if error == nil {
                handleParsedData(parsed)
            } else {
                displayError(error)
            }
        })
    } else {
        displayError(error)
    }
})
```

```
getRemoteData("example.com", { data, error in
    if error == nil {
        parseData(data, { parsed, error in
            if error == nil {
                if parsedDataValid(parsed) {
                    handleParsedData(parsed)
                }
            } else {
                displayError(error)
            }
        })
    } else {
        displayError(error)
    }
})
```

```
getRemoteData("example.com", { data, error in
    if error == nil {
        parseData(data, { parsed, error in
            if error == nil {
                if parsedDataValid(parsed) {
                    saveParsedDataInCache(parsed, { error in
                        if error == nil {
                            handleParsedData(parsed)
                        } else {
                            displayError(error)
                        }
                    })
                }
            }
        })
    } else {
        displayError(error)
    }
})
```

```
getRemoteData("example.com", { data, error in
    if error == nil {
        parseData(data, { parsed, error in
            if error == nil {
                if parsedDataValid(parsed) {
                    saveParsedDataInCache(parsed, { error in
                        if error == nil {
                            handleParsedData(parsed, { error in
                                if error == nil {
                                    displaySuccess()
                                } else {
                                    displayError(error)
                                }
                            })
                        }
                    })
                } else {
                    // Handle invalid parsed data
                }
            }
        })
    }
})
}
```






3. PROGRAMOWANIE DEKLARATYWNE

DEKLARUJEMY CO CHCEMY
OSIĄGNAĆ, A NIE JAK TO
ZROBIĆ

```
let numbers = [1, 2, 3, 4, 5, 6]
var sum = 0
var odds: [Int] = []
for number in numbers {
    sum += number
    if number % 2 == 1 {
        odds.append(number)
    }
}
```

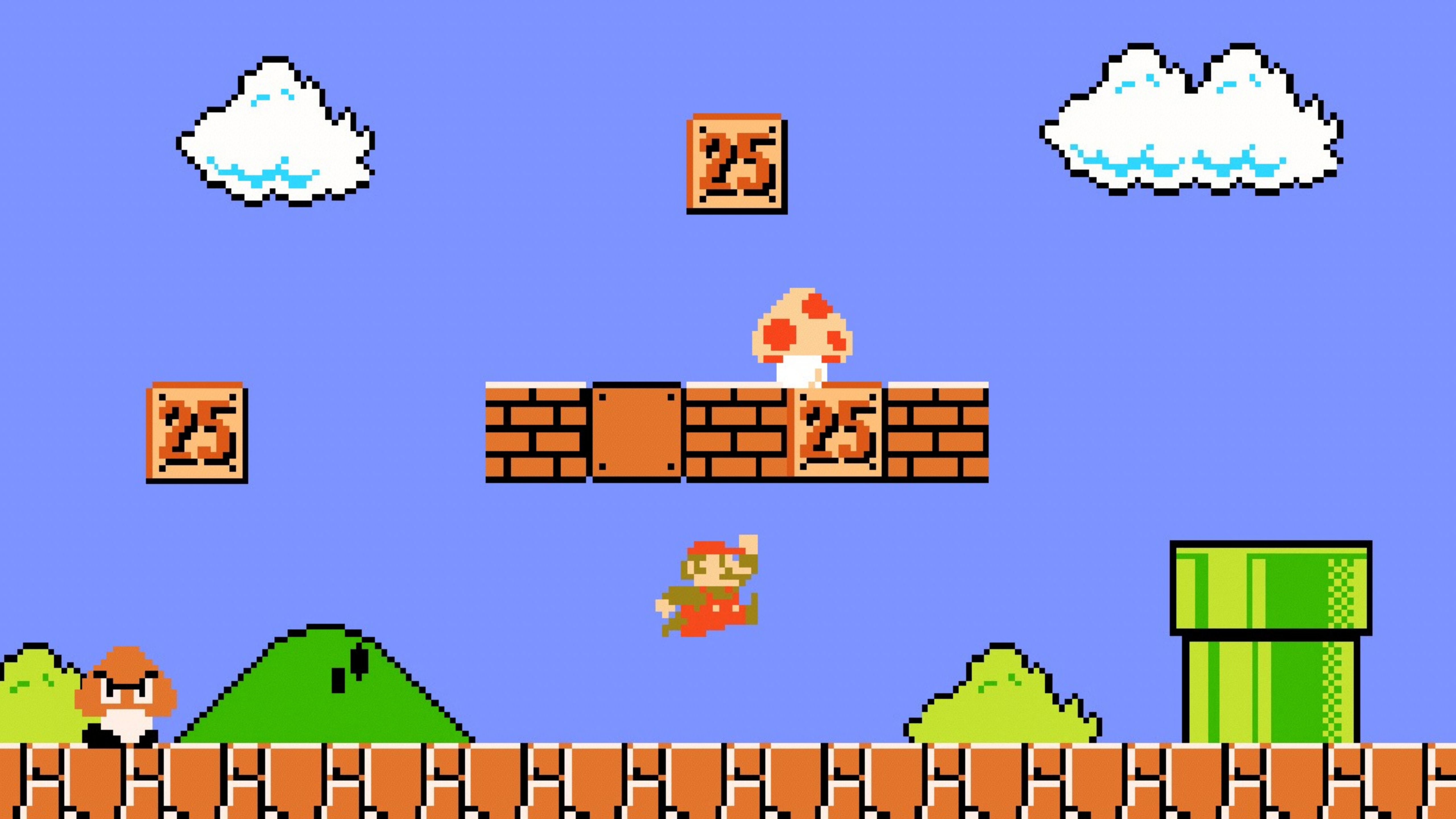
```
var sum = 0
var odds: [Int] = []
let numbers = [1, 2, 3, 4, 5, 6]
for number in numbers {
    sum += number // redukcja
    if number % 2 == 1 { // filtracja
        odds.append(number)
    }
}
```

```
let numbers = [1, 2, 3, 4, 5, 6]
let sum = reduce(numbers, 0, { memo, number in
    return memo + number
})
let odds = filter(numbers, { number in
    return number % 2 == 1
})
```

```
let numbers = [1, 2, 3, 4, 5, 6]
```

```
let sum = reduce(numbers, 0, +)
```

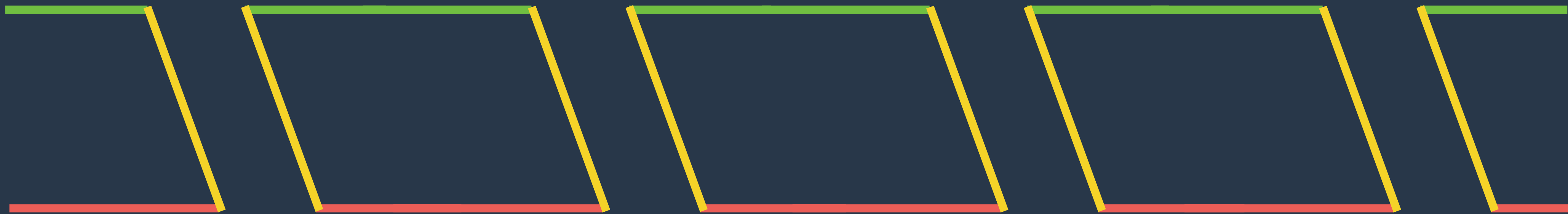
```
let odds = filter(numbers, { $0 % 2 == 1 })
```

RURY

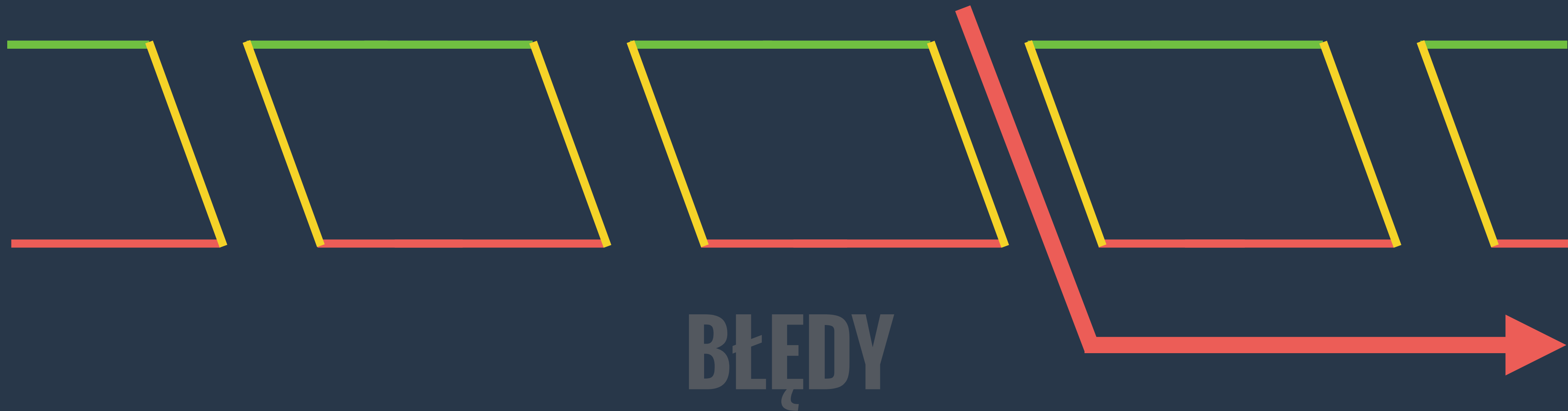


POBIERZ PRZETWÓRZ ZAPISZ WYŚWIETL



BŁĘDY

POBIERZ PRZETWÓRZ ZAPISZ WYŚWIETL



```
getRemoteData("example.com")  
  .then({ data in parseData(data) })  
  .filter({ parsed in parsedDataValid(parsed) })  
  .then({ parsed in saveInCache(parsed) })  
  .then({ parsed in handleParsedData(parsed) })  
  .error({ error in displayError(error) })
```

```
getRemoteData("example.com")
    .then({ data in parseData(data) })
    .filter({ parsed in parsedDataValid(parsed) })
    .filter({ parsed in !alreadyInCache(parsed) })
    .then({ parsed in saveInCache(parsed) })
    .then({ parsed in handleParsedData(parsed) })
    .error({ error in displayError(error) })
```



```
getRemoteData("example.com")
  .then({ data in parseData(data) })
  .filter({ parsed in parsedDataValid(parsed) })
  .filter({ parsed in !alreadyInCache(parsed) })
  .then({ parsed in saveInCache(parsed) })
  .then({ parsed in handleParsedData(parsed) })
  .error({ error in displayError(error) })
```


**PROGRAMOWANIE
DEKLARATYWNE JEST
PROSTSZE**

**PROGRAMOWANIE
DEKLARATYWNE JEST
BEZPIECZNIEJSZE**

**PROGRAMOWANIE
DEKLARATYWNE JEST
BARDZIEJ SKALOWALNE**

CZY ISTNIEJE
NAJLEPSZY PARADYGMAT?

1. OBIEKTOWY

2. IMPERATYWNY

3. DEKLARATYWNY

~~1. OBIĘKTOWY~~

2. IMPERATYWNY

3. DEKLARATYWNY



~~1. OBIĘKTOWY~~

~~2. IMPERATYWNY~~

3. DEKLARATYWNY

RAZEM

DZIĘKUJĘ
ADRIAN KASHIVSKYY

@akashivskyy

github.com/akashivskyy/talks