Switch Case Statement

This lesson explains the key features of switch case statements and their use as an alternative to multiple if-else statements

WE'LL COVER THE FOLLOWING

Alternative to Multiple `ifelse` Statements

We covered `if else` statements in the previous lesson. However, most programming languages also have some sort of switch case statement to allow developers to avoid doing complex and ugly series of `if else` statements.

Here's an example demonstrating the concept:

```
package main

import (
        "fmt"
        "time"
)

func main() {
        now := time.Now().Unix()
        mins := now % 2
        switch mins {
        case 0:
                fmt.Println("even")
        case 1:
                fmt.Println("odd")
        }
}
```

Key Features

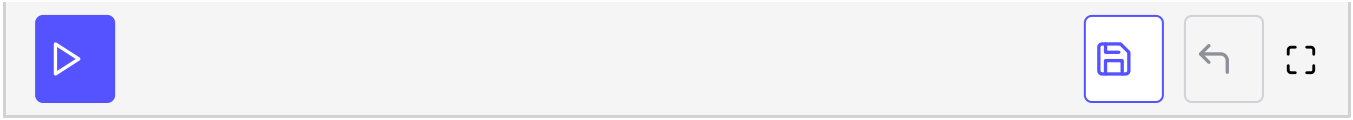
There are a few interesting things to know about this statement in Go:

- You can only compare values of the same type.
- You can set an optional `default` statement to be executed if all the others fail.
- You can use an expression in the `case` statement, for instance you can calculate a value to use in the case:

```
package main

import "fmt"

func main() {
        num := 3
        v := num % 2
        switch v {
        case 0:
                fmt.Println("even")
        case 3 - 2:
                fmt.Println("odd")
        }
}
```



- You can have multiple values in a `case` statement:

Environment Variables

Key:	Value:
GOPATH	/go

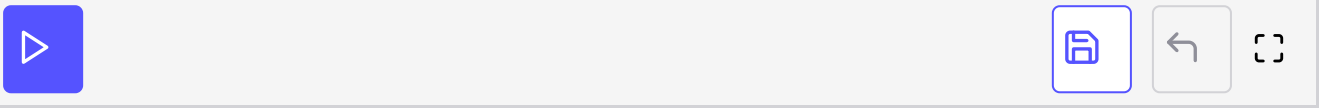
```
package main

import "fmt"

func main() {
        score := 7
        switch score {
        case 0, 1, 3:
                fmt.Println("Terrible")
        case 4, 5:
                fmt.Println("Mediocre")
        case 6, 7:
                fmt.Println("Not bad")
        case 8, 9:
                fmt.Println("Almost perfect")
        case 10:
```

```
                fmt.Println("hmm did you cheat?")
        default: //to be executed if no other case matches
                fmt.Println(score, " off the chart")

        }
}
```



- You can execute all the following statements after a match using the `fallthrough` statement:

Environment Variables

Key:	Value:
GOPATH	/go

```
package main

import "fmt"

func main() {
        n := 4
        switch n {
        case 0:
                fmt.Println("is zero")
                fallthrough //if case matches, all following conditions will be executed as w
        case 1:
                fmt.Println("is <= 1")
                fallthrough
        case 2:
                fmt.Println("is <= 2")
                fallthrough
        case 3:
                fmt.Println("is <= 3")
                fallthrough
        case 4:
                fmt.Println("is <= 4")
                fallthrough
        case 5:
                fmt.Println("is <= 5")
                fallthrough
        case 6:
                fmt.Println("is <= 6")
                fallthrough
        case 7:
                fmt.Println("is <= 7")
                fallthrough
        case 8:
                fmt.Println("is <= 8")
                fallthrough
        default:
                fmt.Println("Try again!")
        }
}
```

You can use a `break` statement inside your matched statement to exit the switch processing:

Environment Variables

Key:	Value:
GOPATH	/go

```
package main

import (
        "fmt"
        "time"
)

func main() {
        n := 1
        switch n {
        case 0:
                fmt.Println("is zero")
                fallthrough
        case 1:
                fmt.Println("<= 1")
                fallthrough
        case 2:
                fmt.Println("<= 2")
                fallthrough
        case 3:
                fmt.Println("<= 3")
                if time.Now().Unix()%2 == 0 {
                        fmt.Println("un pasito pa lante maria")
                        break //execution stops here if this case matches i.e. no other case
                }
                fallthrough
        case 4:
                fmt.Println("<= 4")
                fallthrough
        case 5:
                fmt.Println("<= 5")
        }
}
```

This concludes the discussion on all the available control flow features in Go. The following lesson contains an exercise that tests a combination of these control flow features.