Cheatsheets / **Learn Go: Loops and Arrays**

Learn Go: Loops

A loop repeats a block of code until a certain condition is met.

A loop repeats a block of code until a certain condition is met.

A definite loop repeats a fixed number of times. It has: - an initial statement which creates a new variable, - a conditional expression that determines if the loop runs, - and a post statement that runs each time the loop completes.

A definite loop repeats a fixed number of times. It has:

- an initial statement which creates a new variable,
- a conditional expression that determines if the loop runs,
- and a post statement that runs each time the loop completes.

```
for number := 0; number < 5; number++ {
  fmt.Print(number)
}
```

An indefinite loop repeats while a condition remains true.

An indefinite loop repeats while a condition remains true.

```
number := 0 // Initialize a variable to be used inside the loop
for number < 5 {
  fmt.Println(number)
  number++ // Update the variable being used
}
```

In Go, the language is simplified by using only the `for` keyword for both definite and indefinite loops.

In Go, the language is simplified by using only the `for` keyword for both definite and indefinite loops.

When a condition is true forever, then a special type of indefinite loop is created, called an infinite loop.

When a condition is true forever, then a special type of indefinite loop is created, called an infinite loop.

```
for {
  // Loop body logic
  // This repeats forever
}
// This is never reached
```

The break keyword stops the loop at the current iteration.

The break keyword stops the loop at the current iteration.

```
animals := []string{""Cat"", ""Dog"",
""Fish"", ""Turtle""}
for index := 0; index < len(animals);
index++ {
  if animals[index] == ""Dog"" {
    fmt.Println(""Found the perfect
animal!"")
    break // Stop searching the array
  }
}
```

The continue keyword skips the loop to the next iteration.

The continue keyword skips the loop to the next iteration.

```
jellybeans := []string{""green"",
""blue"", ""yellow"", ""red"", ""green"",
""yellow"", ""red""}
for index := 0; index < len(jellybeans);
index++ {
  if jellybeans[index] == ""green"" {
    continue
  }
  fmt.Println(""You ate the"",
jellybeans[index], ""jellybean!"")
}
```

In Go, the range keyword can be used in a map or array to work through each contained item one at a time within a loop.

In Go, the range keyword can be used in a map or array to work through each contained item one at a time within a loop.

```
letters := []string{""A"", ""B"", ""C"",
""D""}
for index, value := range letters {
  fmt.Println(""Index:"", index,
""Value:"", value)
}
```

Print Share