QUIZ

Fill in the blank:

An indefinite loop repeats while a condition is `true` .

You got it!

You are taking candies from a bowl one by one. When you find the chocolate, you want to stop searching. Fill in the code below to make this possible:

```
candies := []string{"Jellybean", "Chocolate", "Gummies", "Licorice"}
for index, candy := range candies {
  if candy == "Chocolate" {
    fmt.Println("I'll take candy number", index)
    // Stop searching the array
    break
  }
}
```

You got it!

What is the purpose of the `for` keyword in Go?

- The `for` keyword only declares indefinite loops, but not definite and infinite loops.
- The `for` keyword only declares definite loops, but not indefinite and infinite loops.
- The `for` keyword is used to declare definite, indefinite, and infinite loops.

Correct! In Go, the language is simplified by using only the for keyword for all types of loops.

- The `for` keyword only declares infinite loops, but not definite and indefinite loops.

Fill in the blanks:

In a definite loop, the **initial statement** creates a new variable to be used within the for loop code block. The **conditional expression** stops the loop at a certain point. And the **post statement** is a piece of code that runs each time that loop completes.

You got it!

What is a loop best used for in programming?

To write more modular code.

To check the condition of a statement.

To automate repetitive tasks.

Correct! With loops a repetitive task can be programmed easily with a simple syntax.

To give names to certain values.

Fill in the blanks:

An infinite loop runs **forever** and has **no** condition.

You got it!

You have an address book that maps the names of your friends to their address. Fill in the code below to iterate over this map and print your friend's information.

```
addressBook := map[string]string{
  "Jannet": "22 Water St",
  "Joe": "241 North Rd",
  "Robert": "86 Stone St",
}
for name, address := range addressBook {
  fmt.Println(name, "lives at", address)
}
```

You got it!

Which keyword skips to the next iteration of the loop?

`continue`

Correct! The `continue` keyword skips to the next iteration of the loop.

`range`

`if`

`break`