Closures in Swift are self-contained blocks of code that can be passed around and used in your code. They are often used to encapsulate functionality or to delay the execution of code.

Closures are defined using curly braces {} and can contain any valid Swift code. They can also capture variables from the context in which they are defined. This allows closures to access and modify values of variables even after the original scope has ended.

Closures can be passed as arguments to functions, returned from functions, and assigned to variables. They can also be used to create objects, such as arrays and dictionaries.

Here is an example of a simple closure:

Swift

```swift
let closure = { (name: String) -> String in
  return "Hello, \(name)!"
}
```

This closure takes a string as an argument and returns a string. The closure captures the name variable from the context in which it is defined.

Closures can be used to encapsulate functionality, such as the following example:

Swift

```swift
func greet(name: String) {
  print(closure(name))
}

greet(name: "John Doe") // Prints "Hello, John Doe!"
```

Closures can also be used to delay the execution of code, such as the following example:

Swift

```swift
let timer = Timer.scheduledTimer(withTimeInterval: 1.0, repeats:
false, block: { () -> Void in
  print("Hello, world!")
})
```

This timer will execute the closure after 1 second.

Closures are a powerful tool that can be used to write more flexible and reusable code in Swift.

Here are some additional benefits of using closures in Swift:

- **Readability:** Closures can make your code more readable and easier to understand.
- **Maintainability:** Closures can make your code more maintainable by encapsulating functionality and delaying the execution of code.
- **Performance:** Closures can improve the performance of your code by allowing the compiler to optimize them.

Overall, closures are a valuable feature of Swift that can help you to write better code.