Generics

Generics allow you to write code that can be used with different types. This is useful because it can save you from writing duplicate code and make your code more flexible.

To create a generic function, you use the < and > symbols to specify the type parameters. For example, the following function is generic because it takes two type parameters, T and U:

Swift .
```swift
func sum<T, U>(a: T, b: U) -> T + U where T: Numeric, U: Numeric {
    return a + b
}
```

The sum() function can be used to sum two values of any type that conforms to the Numeric protocol. The Numeric protocol defines a set of methods that must be implemented by any type that wants to conform to the protocol. These methods include methods for adding, subtracting, multiplying, and dividing values.

To use a generic function, you need to specify the type arguments when you call the function. For example, the following code calls the sum() function with the type arguments Int and Double:

Swift .
```swift
let result1 = sum(a: 1, b: 2) // 3
let result2 = sum(a: 1.0, b: 2.0) // 3.0
```

The result1 variable will be assigned the value 3, and the result2 variable will be assigned the value 3.0.

Generics can also be used to create generic types. For example, the following code defines a generic Array type:

Swift .
```swift
struct Array<T> {
    private var elements: [T] = []
    mutating func append(_ element: T) {
        elements.append(element)
    }

    func count() -> Int {
        return elements.count
    }
}
```

The Array<T> type can be used to store an array of values of any type. To create an Array<T> object, you need to specify the type argument when you create the object. For example, the following code creates an Array<Int> object:

Swift
```swift
let array = Array<Int>()
```

The array object can be used to store an array of integers.

Generics are a powerful tool that can be used to write reusable and flexible code. By using generics, you can avoid writing duplicate code and make your code more efficient.

Swift is a strongly typed language

This means that the type of every variable and expression in Swift must be known at compile time. This helps to prevent errors and makes the code more reliable.

Type parameter

A type parameter is a placeholder for a type that is specified when the generic function or type is used. For example, the T and U type parameters in the sum() function are type parameters.

Here is an example of how to use a generic type parameter:

Swift
```swift
func printArray<T>(array: [T]) {
    for element in array {
        print(element)
    }
}
```

```
let intArray = [1, 2, 3]
let stringArray = ["Hello", "world!"]

printArray(array: intArray)
printArray(array: stringArray)
```

The printArray() function is a generic function that takes an array of any type as an argument. The T type parameter is the type of the elements in the array.

When the printArray() function is called with the intArray argument, the T type parameter is inferred to be Int. When the printArray() function is called with the stringArray argument, the T type parameter is inferred to be String.

This is just a simple example of how to use generic type parameters. Generics can be used to write more complex and powerful code.

I hope this helps!