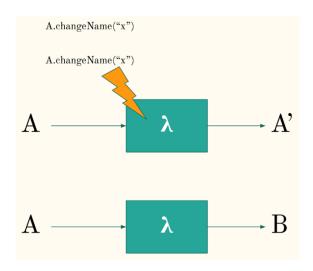
Chapter 1: Introducing Functional Programming



Chapter 2: Treating Functions as First-Class Citizens

No images

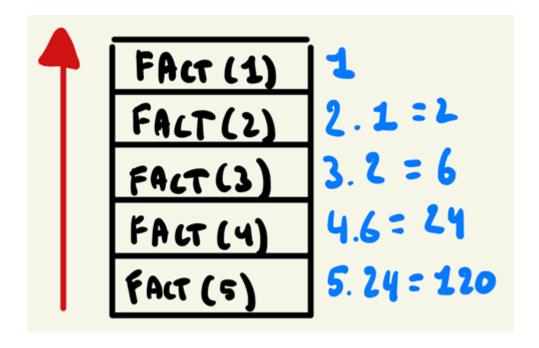
Chapter 3: Higher -Order Functions

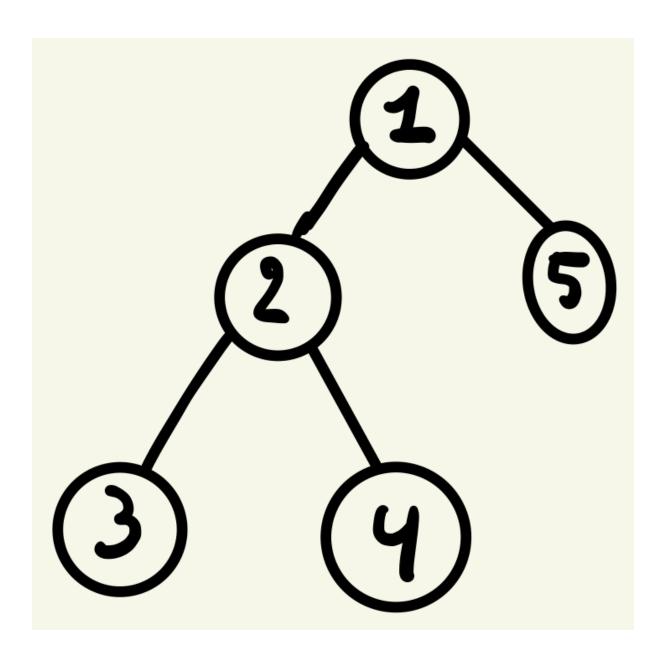
Chapter 4: Write Testable Codes with Pure Functions

Chapter 5: Immutability

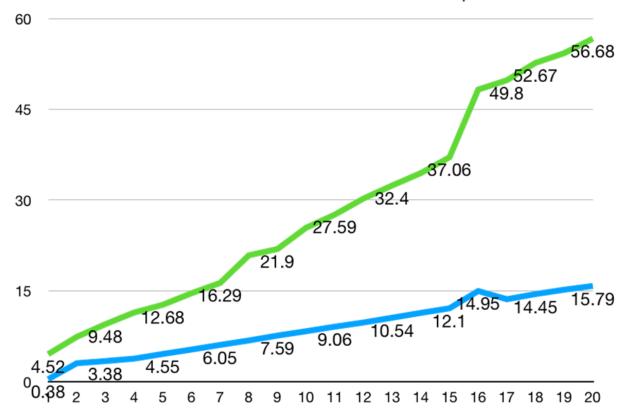
Chapter 6: Three Common Categories of Functions

Chapter 7: Recursion









Chapter 8: Readable Function Composition with Fluent Programming

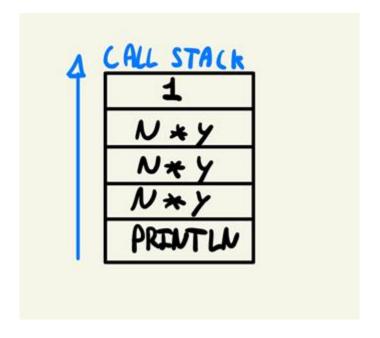
```
func main() {
    x := 3
    y := 4
    z := x + y
    fmt.Println(z) \
}
```

```
func main() {
    x := 3
    y := 4
    z := add(x, y)
    fmt.Println(z)
}
```

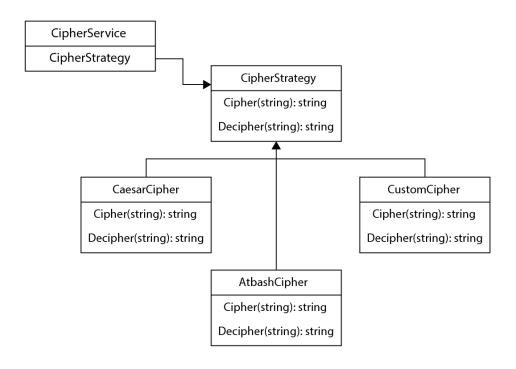
```
func main() {
    input := [jint{1, 2, 3, 4, 5, 6} ]
    isEven := functi int) bool {
        return i%2 == 0
    }
    numberPrinter(pkg.Filter(input, isEven))
}

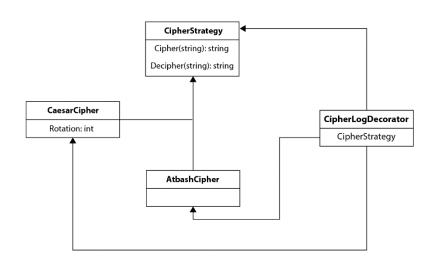
func numberPrinter(input [jint) {
        for _, in := range input {
            fmt.Println(in)
        }
    }
}

func Filter[A any](input []A, pred Predicate[A]) []A {
        output := []A(]
        for _, element := range input {
            if pred(element) {
                output = append(output, element)
            }
        }
        return output
}
```

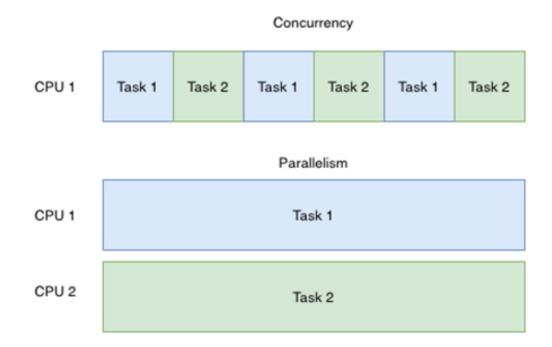


Chapter 9: Functional Design Patterns





Chapter 10: Thinking functionally to solve problems



Chapter 11: Functional Programming Libraries