ITERATION WITH RECURSION

Generate n random numbers

```
import scala.util.Random

def randomNumbers(length: Int): List[Int] =
  if(length <= 0) Nil
  else {
    val number = Random.nextInt(100)
    number :: randomNumbers(length - 1)
  }
```

```
randomNumbers(2)
// res0: List[Int] = List(88, 29)
randomNumbers(5)
// res1: List[Int] = List(30, 49, 89, 93, 98)
randomNumbers(-1)
// res2: List[Int] = List()
```

Termination conditions

```
import scala.util.Random

def randomNumbers(length: Int): List[Int] =
  if(length == 0) Nil
  else {
    val number = Random.nextInt(100)
    number :: randomNumbers(length - 1)
  }
```

```
randomNumbers(2)
// res4: List[Int] = List(52, 0)
```

```
randomNumbers(-1)
// error: java.lang.StackOverflowError
```

Termination conditions

```
import scala.util.Random

def randomNumbers(length: Int): List[Int] =
  if(length == 0) Nil
  else {
    val number = Random.nextInt(100)
    number :: randomNumbers(length - 1)
  }
```

```
randomNumbers(2)
// res6: List[Int] = List(68, 12)
```

```
randomNumbers(-1)
// error: java.lang.StackOverflowError
```

Infinite recursion

```
def doSomething: String = doSomething
```

```
def doSomething: String = {
  while(true) { /* do nothing */ }
  ???
}
```

Generalise with Function0

```
def repeat[A](length: Int, action: () => A): List[A] =
  if(length <= 0) Nil
  else {
    val result = action()
    result :: repeat(length - 1, action)
  }
```

Generalise with Function0

```
def repeat[A](length: Int, action: () => A): List[A] =
  if(length <= 0) Nil
  else {
    val result = action()
    result :: repeat(length - 1, action)
  }
```

```
repeat(5, () => Random.nextInt(100))
// res8: List[Int] = List(94, 75, 57, 7, 0)

repeat(2, () => println("Hello"))
// Hello
// Hello
// res9: List[Unit] = List((), ())
```

Generalise with call-by-name parameter

```
def repeat[A](length: Int, action: => A): List[A] =
  if(length <= 0) Nil
  else {
    val result = action
    result :: repeat(length - 1, action)
  }
```

```
repeat(5, Random.nextInt(100))
// res11: List[Int] = List(31, 47, 63, 94, 28)

repeat(2, println("Hello"))
// Hello
// Hello
// res12: List[Unit] = List((), ())
```

Generalise with call-by-name parameter

```
def repeat[A](length: Int, action: => A): List[A] =
  if(length <= 0) Nil
  else
    action :: repeat(length - 1, action)
```

```
repeat(5, Random.nextInt(100))
// res14: List[Int] = List(23, 94, 93, 30, 8)

repeat(2, println("Hello"))
// Hello
// Hello
// res15: List[Unit] = List((), ())
```

Retry

```
import scala.util.{Try, Success, Failure}

def retry[A](remainingRetries: Int, action: => A): A =
  Try(action) match {
    case Success(value)     => value
    case Failure(exception) =>
      if(remainingRetries <= 0) throw exception
      else retry(remainingRetries - 1, action)
  }
```

Recursive data structure

- List
- Tree: JSON, file system
- case class Person(name: String, children: List[Person])

Problems with recursions

to understand what is recursion, we need to first understand recursion

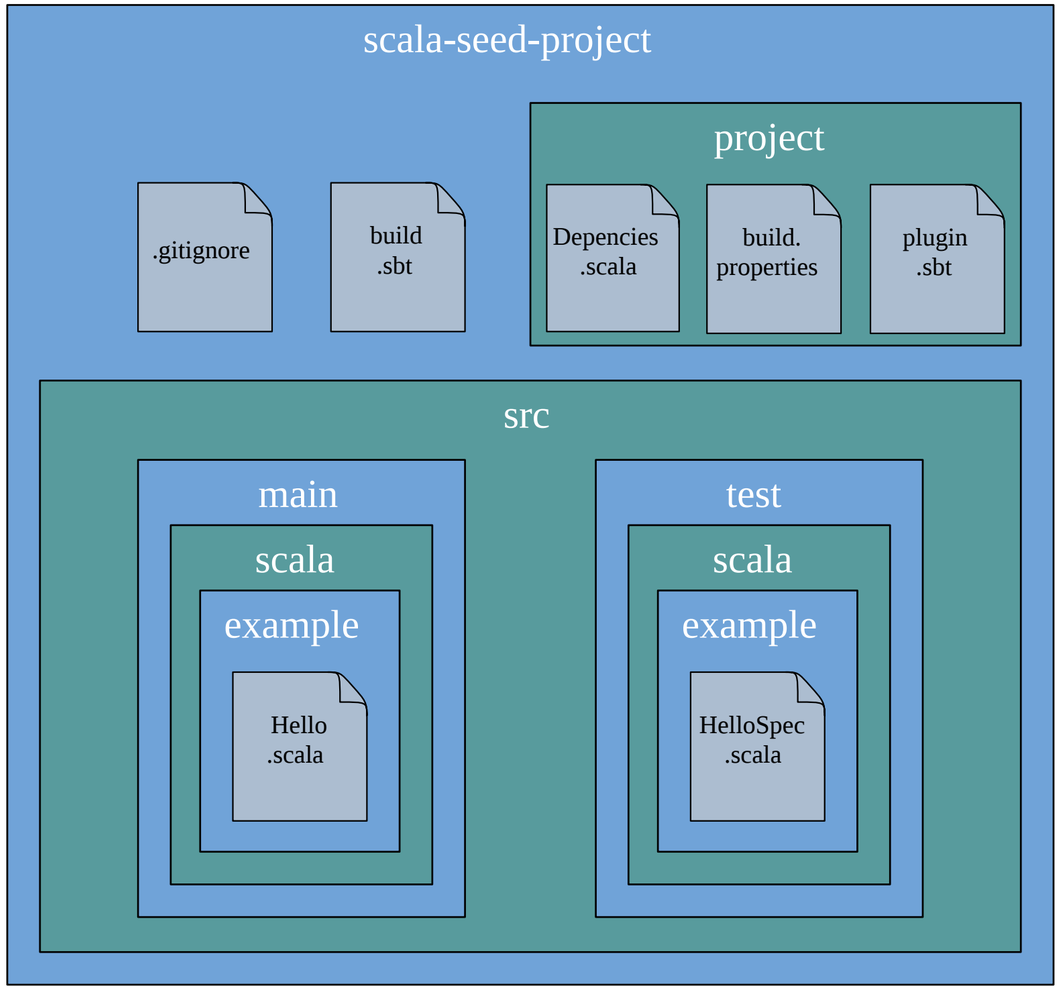
```
def doSomething: String = doSomething
```

```
def doSomething: String = {
  while(true) { /* do nothing */ }
  ???
}
```

Disk Usage

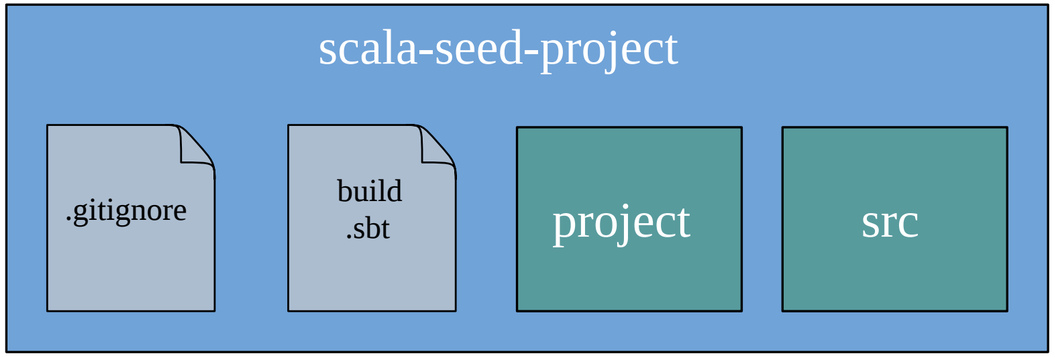
```
sbt new scala/scala-seed.g8
cd scala-seed-project
```

```
diskUsage(".")
```



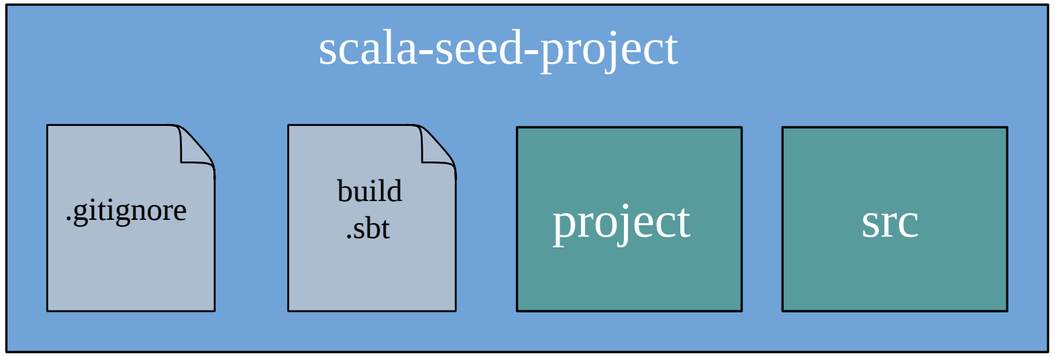
Disk Usage

```
diskUsage(".") =
  diskUsage(".gitignore") +
  diskUsage("build.sbt")  +
  diskUsage("project")    +
  diskUsage("src")
```



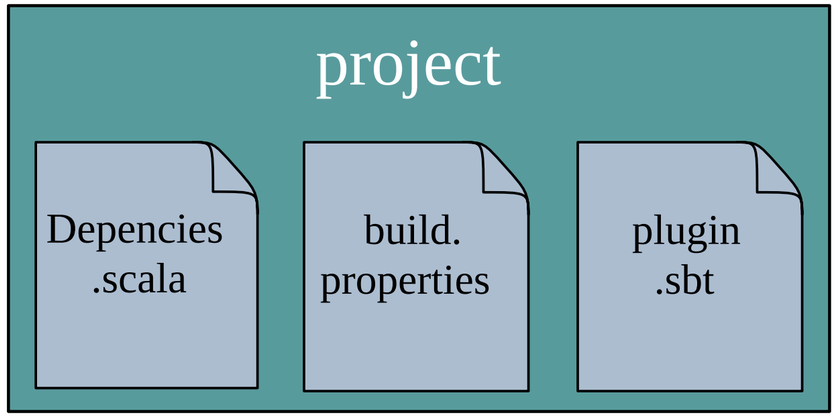
Disk Usage

```
diskUsage(".") =
  4 Kb + // diskUsage(".gitignore")
  4 Kb + // diskUsage("build.sbt")
  diskUsage("project") +
  diskUsage("src")
```



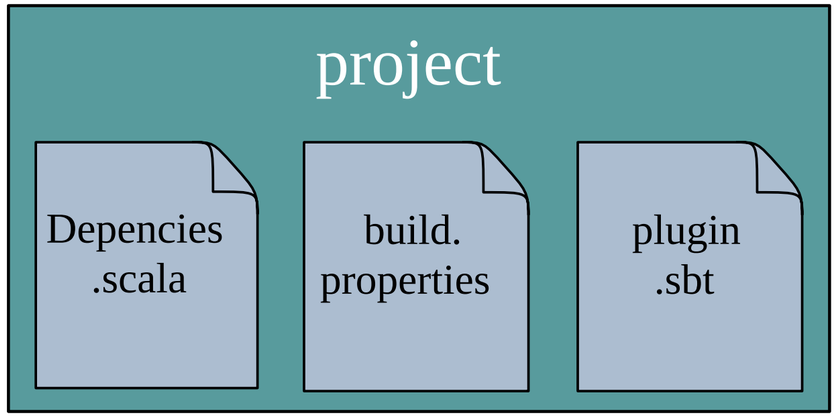
Disk Usage

```
diskUsage("project") =
  diskUsage("project/Dependencies.scala") +
  diskUsage("project/build.properties")   +
  diskUsage("project/plugin.sbt")
```



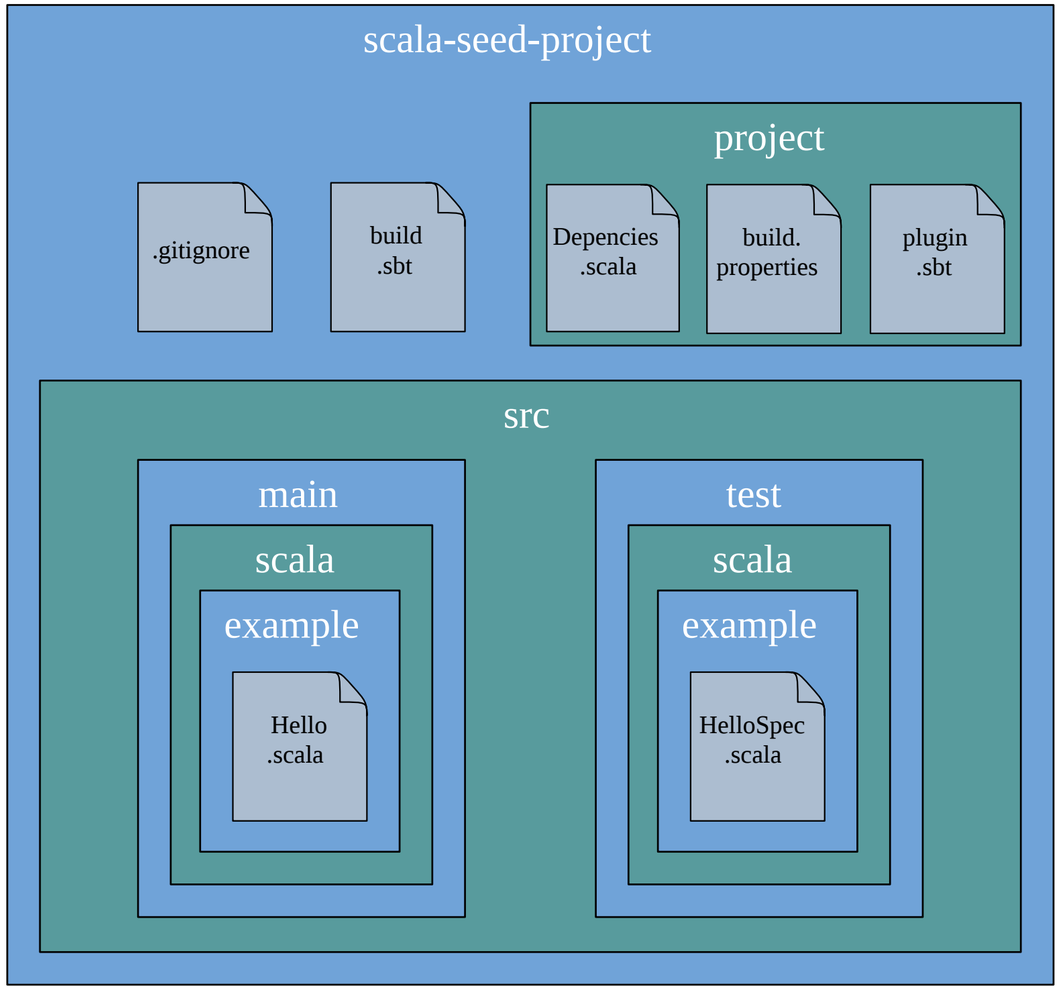
Disk Usage

```
diskUsage("project") =
  4 Kb + // diskUsage("project/Dependencies.scala
  4 Kb + // diskUsage("project/build.properties")
  4 Kb   // diskUsage("project/plugin.sbt")
```



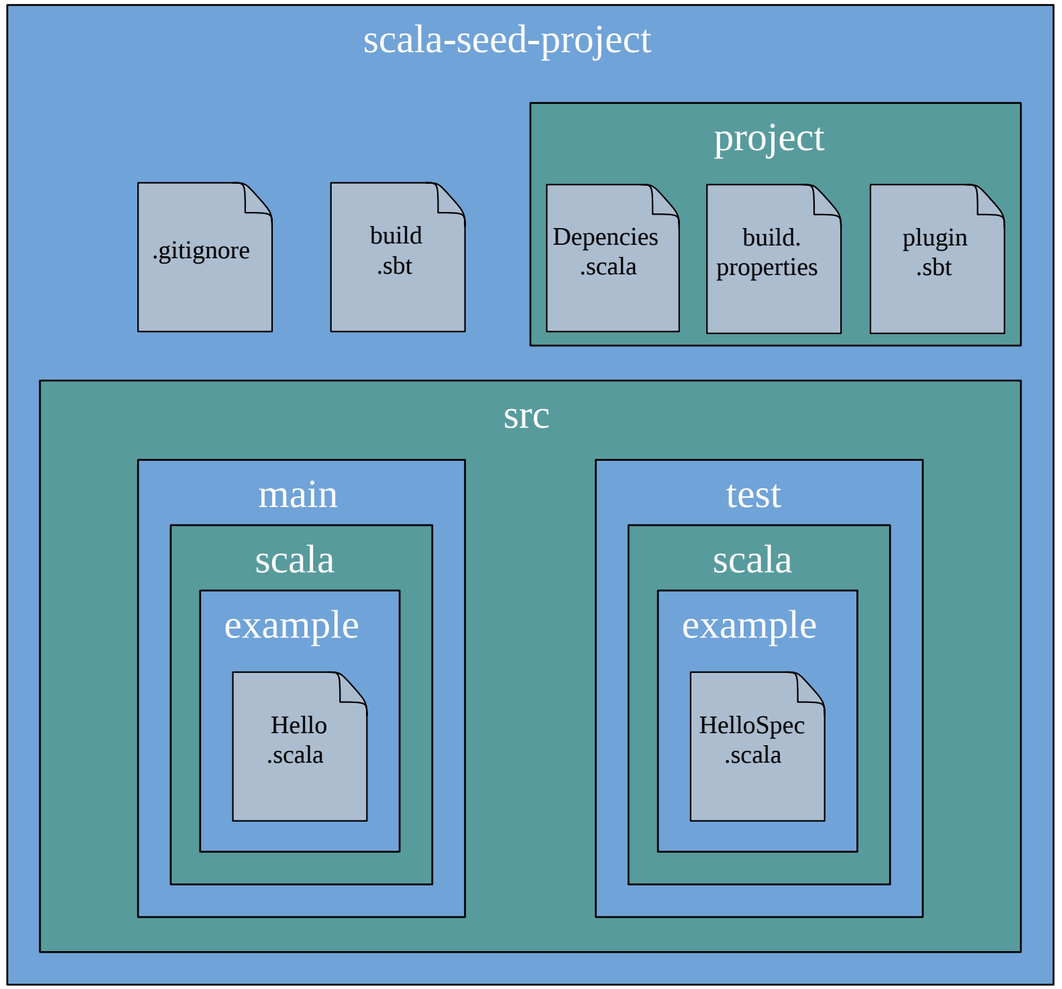
Disk Usage

```
diskUsage(".") =
    4 Kb + // diskUsage(".gitignore")
    4 Kb + // diskUsage("build.sbt")
   12 Kb + // diskUsage("project")
   diskUsage("src")
```



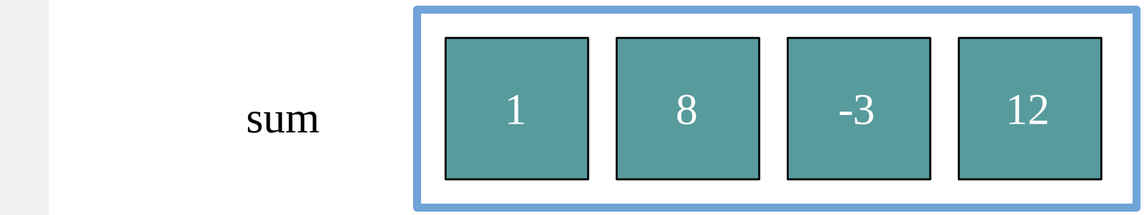
Disk Usage

```
diskUsage(".") =
    4 Kb + // diskUsage(".gitignore")
    4 Kb + // diskUsage("build.sbt")
   12 Kb + // diskUsage("project")
    8 Kb   // diskUsage("src")
```



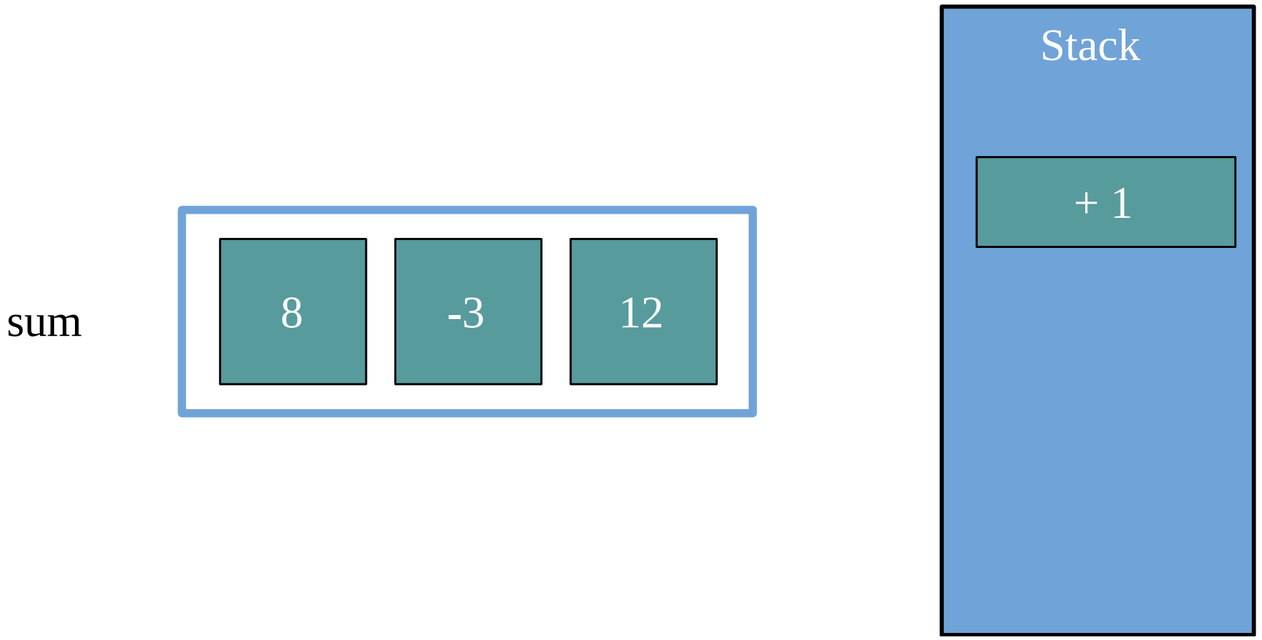
Recursion

```
def sum(numbers: List[Int]): Int =
  numbers match {
    case Nil =>
      0
    case head :: tail =>
      head + sum(tail)
  }
```



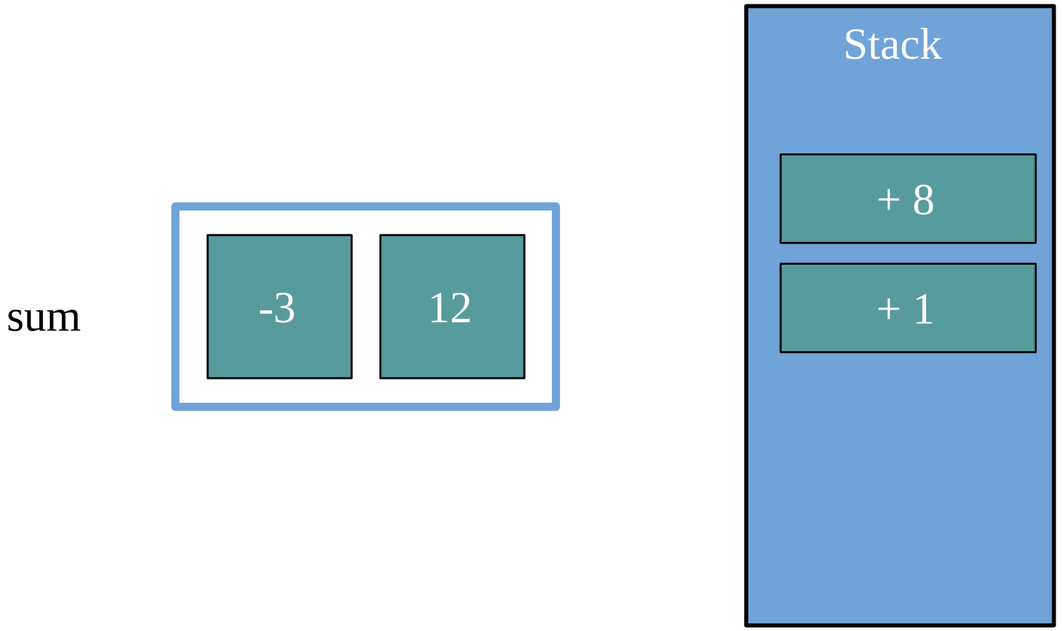
Recursion

```
def sum(numbers: List[Int]): Int =
  numbers match {
    case Nil =>
      0
    case head :: tail =>
      head + sum(tail)
  }
```



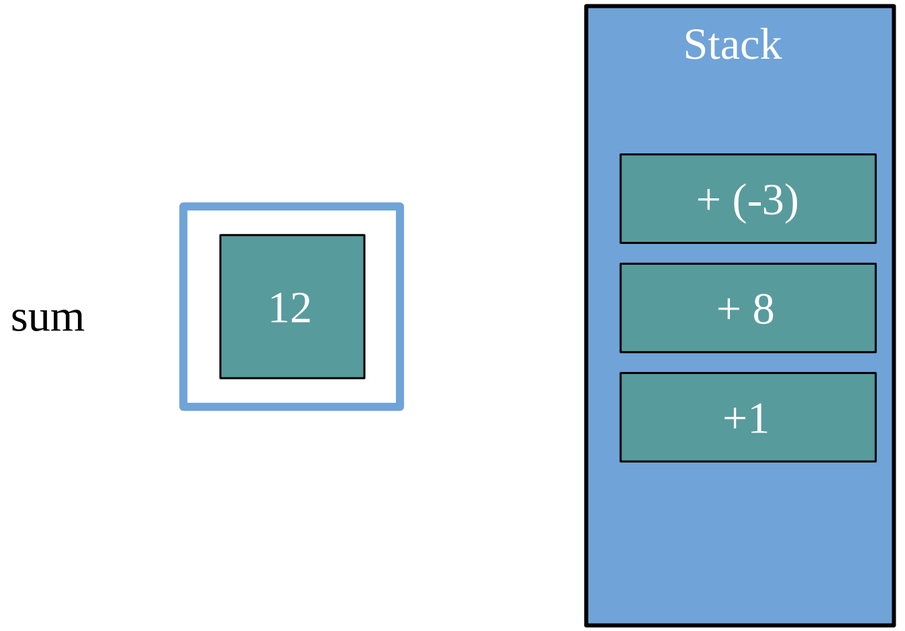
Recursion

```
def sum(numbers: List[Int]): Int =
  numbers match {
    case Nil =>
      0
    case head :: tail =>
      head + sum(tail)
  }
```



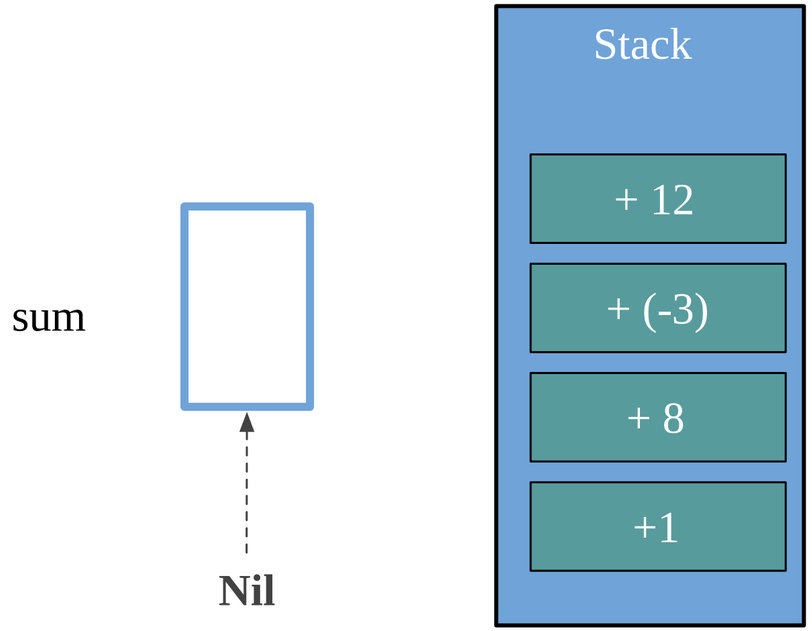
Recursion

```
def sum(numbers: List[Int]): Int =
  numbers match {
    case Nil =>
      0
    case head :: tail =>
      head + sum(tail)
  }
```



Recursion

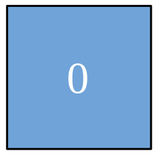
```
def sum(numbers: List[Int]): Int =
  numbers match {
    case Nil =>
      0
    case head :: tail =>
      head + sum(tail)
  }
```

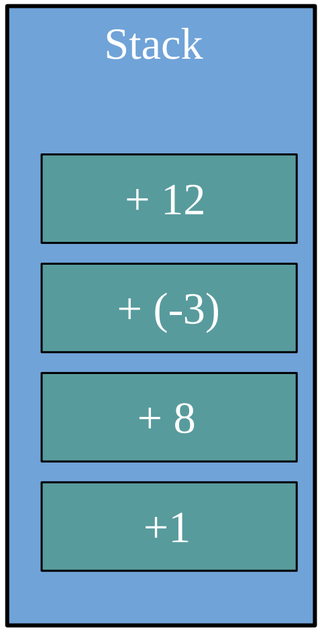


Recursion

```
def sum(numbers: List[Int]): Int =
  numbers match {
    case Nil =>
      0
    case head :: tail =>
      head + sum(tail)
  }
```

```
sum(List())
// res23: Int = 0
```





Recursion

```
def sum(numbers: List[Int]): Int =
  numbers match {
    case Nil =>
      0
    case head :: tail =>
      head + sum(tail)
  }
```