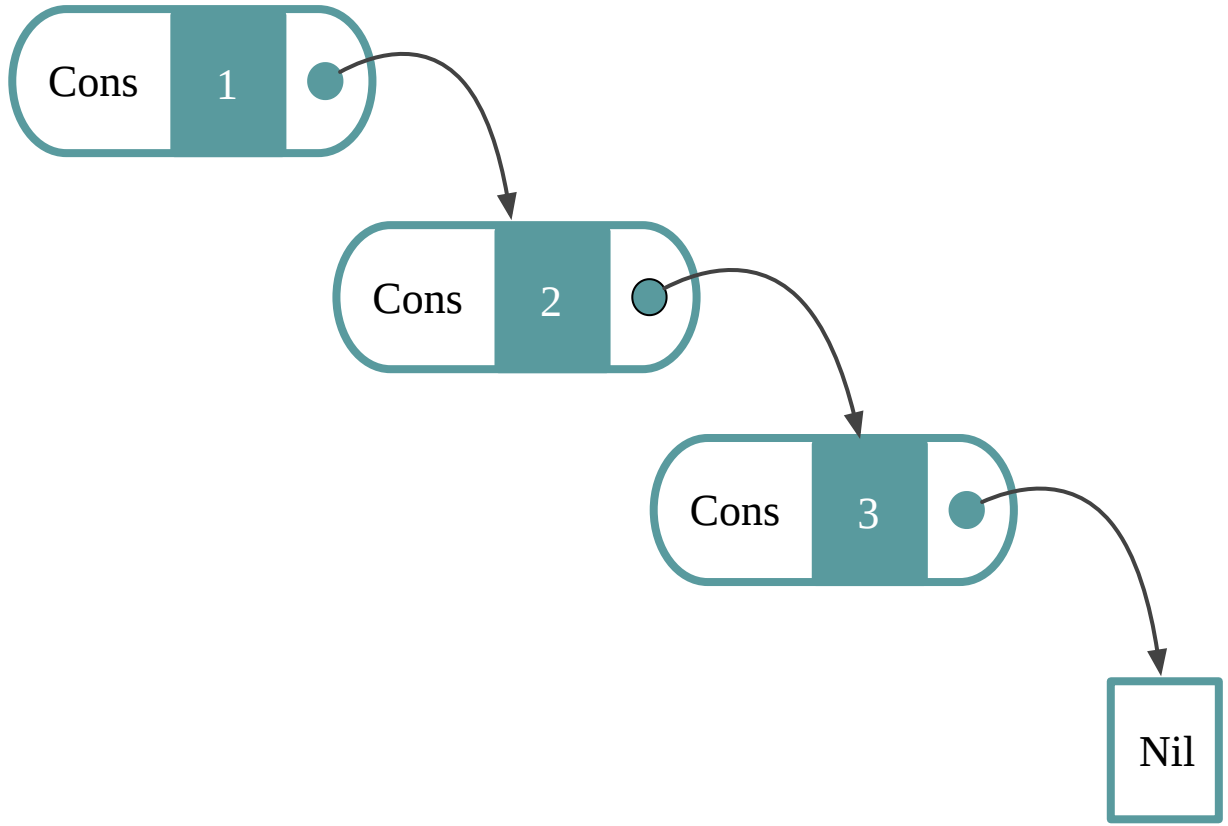
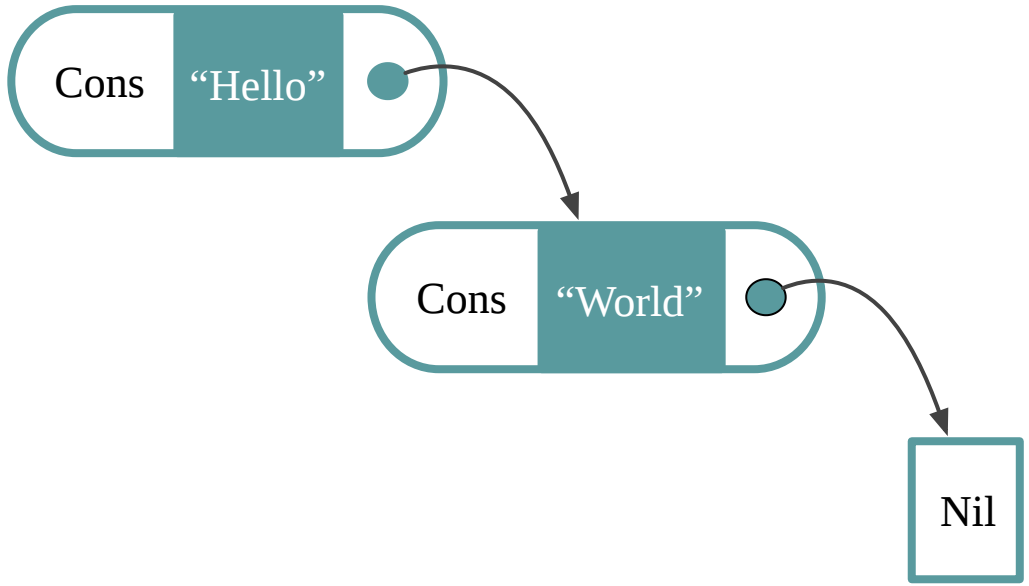

FOUNDATION

Linked List



Linked List



Linked List

```
sealed trait List[A]

case class Nil[A]() extends List[A]
case class Cons[A](head: A, tail: List[A]) extends List[A]
```

Linked List

```
sealed trait List[A]

case class Nil[A]() extends List[A]
case class Cons[A](head: A, tail: List[A]) extends List[A]
```

```
val list: List[Int] = Cons(1, Cons(2, Cons(3, Nil())))
// list: List[Int] = Cons(1, Cons(2, Cons(3, Nil())))
```

Linked List

```
sealed trait List[A] {
  def ::(head: A): List[A] =
    Cons(head, this)
}

case class Nil[A]() extends List[A]
case class Cons[A](head: A, tail: List[A]) extends List[A]
```

```
val list: List[Int] = 1 :: 2 :: 3 :: Nil()
// list: List[Int] = Cons(1, Cons(2, Cons(3, Nil())))
```

Linked List

```
sealed trait List[A]

case class Nil[A]() extends List[A]
case class Cons[A](head: A, tail: List[A]) extends List[A]

object List {
    def apply[A](xs: A*): List[A] =
      if (xs.isEmpty) Nil() else Cons(xs.head, apply(xs.tail: _*))
  }
```

```
val list: List[Int] = List(1,2,3)
// list: List[Int] = Cons(1, Cons(2, Cons(3, Nil())))
```