- **Variants with Methods**
- **More Java Syntax**

Functions with Variants

```
; An animal is either
;  - snake
;  - dillo
;  - ant

; animal-is-lighter? : animal num -> bool
(define (animal-is-lighter? a n)
  (cond
   [(snake? a) (snake-is-lighter? s n)]
   [(dillo? a) (dillo-is-ligheter? s n)]
   [(ant? a) (ant-is-lighter? s n)]))

; snake-is-lighter? : snake num -> bool
(define (snake-is-lighter? s n) ...)

; dillo-is-lighter? : dillo num -> bool
(define (dillo-is-lighter? d n) ...)

; ant-is-lighter? : ant num -> bool
(define (ant-is-lighter? a n) ...)
```

Methods with Variants

```
interface IAnimal {
  boolean isLighter(double n);
}

class Snake implements IAnimal {
  ...
  boolean isLighter(double n) { ... }
}

class Dillo implements IAnimal {
  ...
  boolean isLighter(double n) { ... }
}

class Ant implements IAnimal {
  ...
  boolean isLighter(double n) { ... }
}
```

Translating Functions to Methods

```
; An animal is either
;  - snake
;  - dillo
;  - ant

; animal-is-lighter? : animal num -> bool
(define (animal-is-lighter? a n)
  (cond
   [(snake? a) (snake-is-lighter? s n)]
   [(dillo? a) (dillo-is-ligheter? s n)]
   [(ant? a) (ant-is-lighter? s n)]))

; snake-is-lighter? : snake num -> bool
(define (snake-is-lighter? s n) ...)

; dillo-is-lighter? : dillo num -> bool
(define (dillo-is-lighter? d n) ...)

; ant-is-lighter? : ant num -> bool
(define (ant-is-lighter? a n) ...)
```

```
interface IAnimal {
  boolean isLighter(double n);
}

class Snake implements IAnimal {
  ...
  boolean isLighter(double n) { ... }
}

class Dillo implements IAnimal {
  ...
  boolean isLighter(double n) { ... }
}

class Ant implements IAnimal {
  ...
  boolean isLighter(double n) { ... }
}
```

Translating Functions to Methods

```
; An animal is either
;   - snake
;   - dillo
;   - ant

; animal-is-lighter? : animal num -> bool
(define (animal-is-lighter? a n)
  (cond
   [(snake? a) (snake-is-lighter? s n)]
   [(dillo? a) (dillo-is-ligheter? s n)]
   [(ant? a) (ant-is-lighter? s n)]))

; snake-is-lighter? : snake num -> bool
(define (snake-is-lighter? s n) ...)

; dillo-is-lighter? : dillo num -> bool
(define (dillo-is-lighter? d n) ...)

; ant-is-lighter? : ant num -> bool
(define (ant-is-lighter? a n) ...)
```

Data definition turns into class declarations

```
interface IAnimal {
  boolean isLighter(double n);
}

class Snake implements IAnimal {
  ...
  boolean isLighter(double n) { ... }
}

class Dillo implements IAnimal {
  ...
  boolean isLighter(double n) { ... }
}

class Ant implements IAnimal {
  ...
  boolean isLighter(double n) { ... }
}
```

Translating Functions to Methods

```
; An animal is either
;  - snake
;  - dillo
;  - ant

; animal-is-lighter? : animal num -> bool
(define (animal-is-lighter? a n)
  (cond
   [(snake? a) (snake-is-lighter? s n)]
   [(dillo? a) (dillo-is-ligheter? s n)]
   [(ant? a) (ant-is-lighter? s n)]))

; snake-is-lighter? : snake num -> bool
(define (snake-is-lighter? s n) ...)

; dillo-is-lighter? : dillo num -> bool
(define (dillo-is-lighter? d n) ...)

; ant-is-lighter? : ant num -> bool
(define (ant-is-lighter? a n) ...)
```

Variant functions turn into variant methods — all with the same contract after the implicit argument

```
interface IAnimal {
  boolean isLighter(double n);
}

class Snake implements IAnimal {
  ...
  boolean isLighter(double n) { ... }
}

class Dillo implements IAnimal {
  ...
  boolean isLighter(double n) { ... }
}

class Ant implements IAnimal {
  ...
  boolean isLighter(double n) { ... }
}
```

Translating Functions to Methods

```
; An animal is either
;  - snake
;  - dillo
;  - ant

; animal-is-lighter? : animal num -> bool
(define (animal-is-lighter? a n)
  (cond
   [(snake? a) (snake-is-lighter? s n)]
   [(dillo? a) (dillo-is-ligheter? s n)]
   [(ant? a) (ant-is-lighter? s n)]))

; snake-is-lighter? : snake num -> bool
(define (snake-is-lighter? s n) ...)

; dillo-is-lighter? : dillo num -> bool
(define (dillo-is-lighter? d n) ...)

; ant-is-lighter? : ant num -> bool
(define (ant-is-lighter? a n) ...)
```

Function with variant-based **`cond`** turns into just an **`interface`** method declaration

```
interface IAnimal {
  boolean isLighter(double n);
}

class Snake implements IAnimal {
  ...
  boolean isLighter(double n) { ... }
}

class Dillo implements IAnimal {
  ...
  boolean isLighter(double n) { ... }
}

class Ant implements IAnimal {
  ...
  boolean isLighter(double n) { ... }
}
```

Lists of Things

```
interface IListOfThing {
  int length();
}

class EmptyListOfThing implements IListOfThing {
   EmptyListOfThing() { }
   public int length() { return 0; }
}

class ConsListOfThing implements IListOfThing {
   Thing first;
   IListOfThing rest;
   ConsListOfThing(Thing first, IListOfThing rest) {
     this.first = first;
     this.rest = rest;
   }
   public int length() { return 1 + this.rest.length(); }
}
```


Copy

Trees of Things

```
interface ITreeOfThing {
  int count();
}

class EmptyTreeOfThing implements ITreeOfThing {
   EmptyTreeOfThing() { }
   public int count() { return 0; }
}

class ConsTreeOfThing implements ITreeOfThing {
   Thing v;
   TreeOfThing left;
   TreeOfThing right;
   ConsTreeOfThing(Thing v, ITreeOfThing left, ITreeOfThing right) {
     this.v = v;
     this.left = left;
     this.right = right;
   }
   public int count() { return 1 + this.left.count()
                              + this.right.count(); }
}
```

Copy

- Variants with Methods
- **More Java Syntax**

Conditionals

Some uses of **cond** where not based on the data definition:

```
(define (posn-big-part p)
  (cond
   [(> (posn-x p) (posn-y p)) (posn-x p)]
   [else (posn-y p)]))
```

For these, we use **if** in Java:

```
class Posn { ...
  double bigPart() {
    if (this.x > this.y)
      return this.x;
    else
      return this.y;
  }
}
```

Conditionals

In general:

```
(cond
 [question1 answer1]
 [question2 answer2]
 ...
 [else answerN])
```

⇒

```
if question1
   return answer1;
else if question2
   return answer2;
...
else
   return answerN;
```

Primitive Operations on Numbers

```
1 + 2   → 3
1 - 2   → -1
1 * 2   → 2
1 / 2   → 0
1.0 / 2.0   → 0.5

1 < 2   → true
1 > 2   → false
1 <= 2   → true
1 >= 2   → false
1 == 2   → false
1 == 1   → true
```

Primitive Operations on Booleans

```
!true → false
!false → true

true && true → true
true && false → false
true || false → true
false || false → false
```

Primitive Operations on Strings

```
"hello".equals("bye") → false
"hello".equals("hello") → true

"good".concat(" bye") → "good bye"

"good bye".startsWith("good") → true
"good bye".startsWith("bye") → false

"good bye".endsWith("good") → false
"good bye".endsWith("bye") → true
```

These operations are really method calls, and that's why **`String`** is capitalized