Animals

All animals need to eat...

Define `feed-animal`, which takes an animal (snake, dillo, or ant) and feeds it (5 lbs, 2 lbs, or 0.001 lbs, respectively)

What is an `animal`?
Animal Data Definition

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

The “either” above makes this a new kind of data definition:

    data with varieties

Examples:

    (make-snake 'slinky 10 'rats)
    (make-dillo 2 true)
    (make-ant 0.002 (make-posn 3 4))
Feeding Animals

; feed-animal : animal -> animal
; To feed the animal a
(define (feed-animal a)
  ...)

(check-expect (feed-animal (make-snake "Slinky" 10 "rats"))
  (make-snake "Slinky" 15 "rats"))

(check-expect (feed-animal (make-dillo 2 true))
  (make-dillo 4 true))

(check-expect (feed-animal (make-ant 0.002 (make-posn 3 4)))
  (make-ant 0.003 (make-posn 3 4)))
For the template step...

\[
\text{(define (feed-animal a)  
  ...)}
\]

• Is \text{a} compound data?

• Technically yes, but the definition \text{animal} doesn’t have \text{make-something}, so we don’t use the compound-data template rule
Template for Varieties

Choice in the data definition

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

means cond in the template:

(define (feed-animal a)
  (cond
    [...  ...]
    [...  ...]
    [...  ...]))

Three data choices means three cond cases
Questions for Varieties

```
(define (feed-animal a)
  (cond
    [... ...]
    [... ...]
    [... ...]
    [... ...]))
```

How do we write a question for each case?

It turns out that

```
(define-struct snake (name weight food))
```

provides `snake`?

```
(snake? (make-snake 'slinky 5 'rats)) → true
(snake? (make-dillo 2 true)) → false
(snake? 17) → false
```
Template

(define (feed-animal a)
  (cond
    [(snake? a) ...]
    [(dillo? a) ...]
    [(ant? a) ...]))

New template rule: varieties ⇒ cond

Now continue template case-by-case...
(define (feed-animal a)
  (cond
   [(snake? a) ... (feed-snake a) ...]
   [(dillo? a) ... (feed-dillo a) ...]
   [(ant? a) ... (feed-ant a) ...]))

Remember: references in the data definition ⇒ template references

; An animal is either
;  - snake
;  - dillo
;  - ant
Shapes of Data and Templates

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

; A snake is
; (make-snake sym num sym)

; A dillo is
; (make-dillo num bool)

; An ant is
; (make-ant num posn)

; A posn is
; (make-posn num num)

(define (feed-animal a)
  (cond
    [(snake? a) ... (feed-snake a) ...]
    [(dillo? a) ... (feed-dillo a) ...]
    [(ant? a) ... (feed-ant a) ...]))

(define (feed-snake s)
  ... (snake-name s) ... (snake-weight s)
  ... (snake-food s) ...)

(define (feed-dillo d)
  ... (dillo-weight d)
  ... (dillo-alive? d) ...)

(define (feed-ant a)
  ... (ant-weight d)
  ... (feed-posn (ant-loc d)) ...)

(define (feed-posn p)
  ... (posn-x p) ... (posn-y p) ...)

17
Design Recipe III

**Data**

- Understand the input data

**Contract, Purpose, and Header**

- Describe (but don’t write) the function

**Examples**

- Show what will happen when the function is done

**Template**

- Set up the body based on the input data (and only the input)

**Body**

- The most creative step: implement the function body

**Test**

- Run the examples
When the problem statement mentions $N$ different varieties of a thing, write a data definition of the form

```
; A thing is
;  - variety1
;  ...
;  - varietyN
```
Examples

When the input data has varieties, be sure to pick each variety at least once.

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

(check-expect (feed-animal (make-snake "Slinky" 10 "rats"))
             (make-snake "Slinky" 15 "rats"))

(check-expect (feed-animal (make-dillo 2 true))
             (make-dillo 4 true))

(check-expect (feed-animal (make-ant 0.002 (make-posn 3 4)))
             (make-ant 0.003 (make-posn 3 4)))
Template

When the input data has varieties, start with `cond`

- \( N \) varieties \( \Rightarrow N \) `cond` lines
- Formulate a question to match each corresponding variety
- Continue template steps case-by-case

```
(define (feed-animal a)
  (cond
    [(snake? a) ...]
    [(dillo? a) ...]
    [(ant? a) ...]))
```
Template

When the input data has varieties, start with cond

- N varieties ⇒ N cond lines
- Formulate a question to match each corresponding variety
- Continue template steps case-by-case

When the data definition refers to a data definition, make the template refer to a template

```scheme
(define (ant-at-home? a)
  ... (ant-weight a)
  ... (posn-at-home? (ant-loc a)) ...)

(define (posn-at-home? p)
  ... (posn-x p) ... (posn-y p) ...)
```
Template

When the input data has varieties, start with cond

- \( N \) varieties \(\Rightarrow N \) cond lines
- Formulate a question to match each corresponding variety
- Continue template steps case-by-case

When the data definition refers to a data definition, make the template refer to a template

```
(define (feed-animal a)
  (cond
    [(snake? a) ... (feed-snake a) ...]
    [(dillo? a) ... (feed-dillo a) ...]
    [(ant? a) ... (feed-ant a) ...]]
```