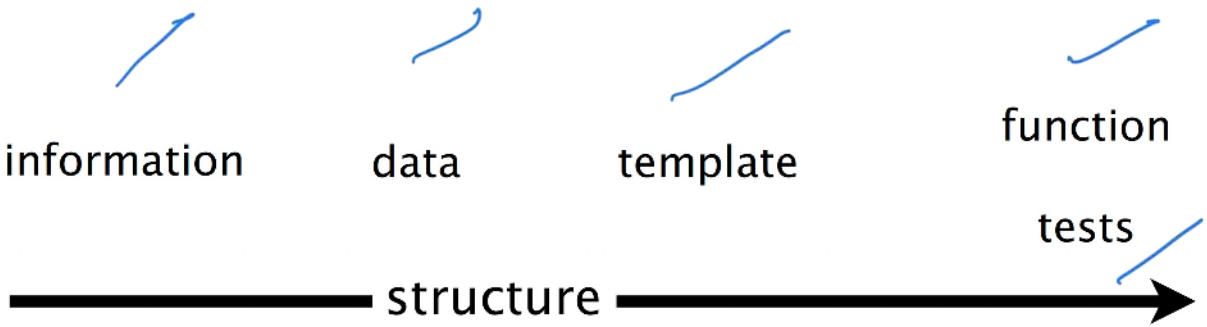
L04-pad - htdf + htdw

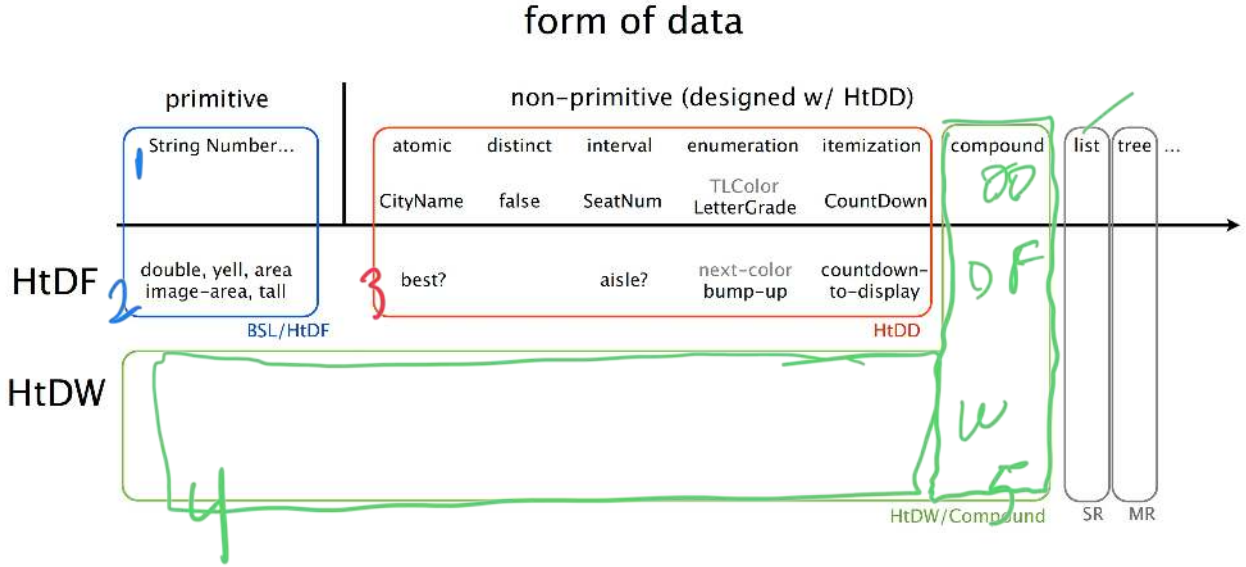
Computation Programs And Programming (The University of British Columbia)



identifying the structure of the information is a key step in program design

as data definitions get more sophisticated you will see that choosing the structure to use is a point of leverage in designing the overall program

keep calm and design the data



I want a function that ???=

one big problem?

signature

6 smaller pieces

purpose

examples

```
(@htdf topple)
(@signature Image -> Image)
;; produce image rotated by 90 degrees
(check-expect (topple (rectangle 10 20 "solid" "red"))
              (rectangle 20 10 "solid" "red"))
(check-expect (topple (triangle 20 "solid" "red"))
              (rotate 90 (triangle 20 "solid" "red")))

;(define (topple img) empty-image) ;stub

(@template-origin Image)

(@template
 (define (topple img)
   (... img)))

(define (topple img)
  (rotate 90 img))
```

how do I build template?

template

function definition

I want data that ???=

one big problem?

type comment

interpretation

examples

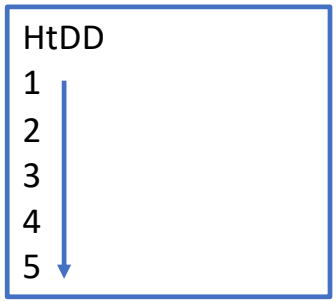
how to produce template

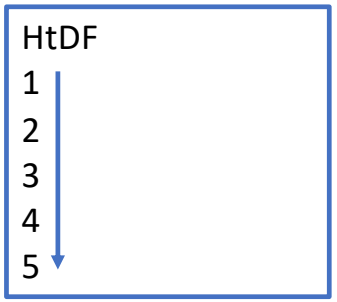
template

5 smaller pieces

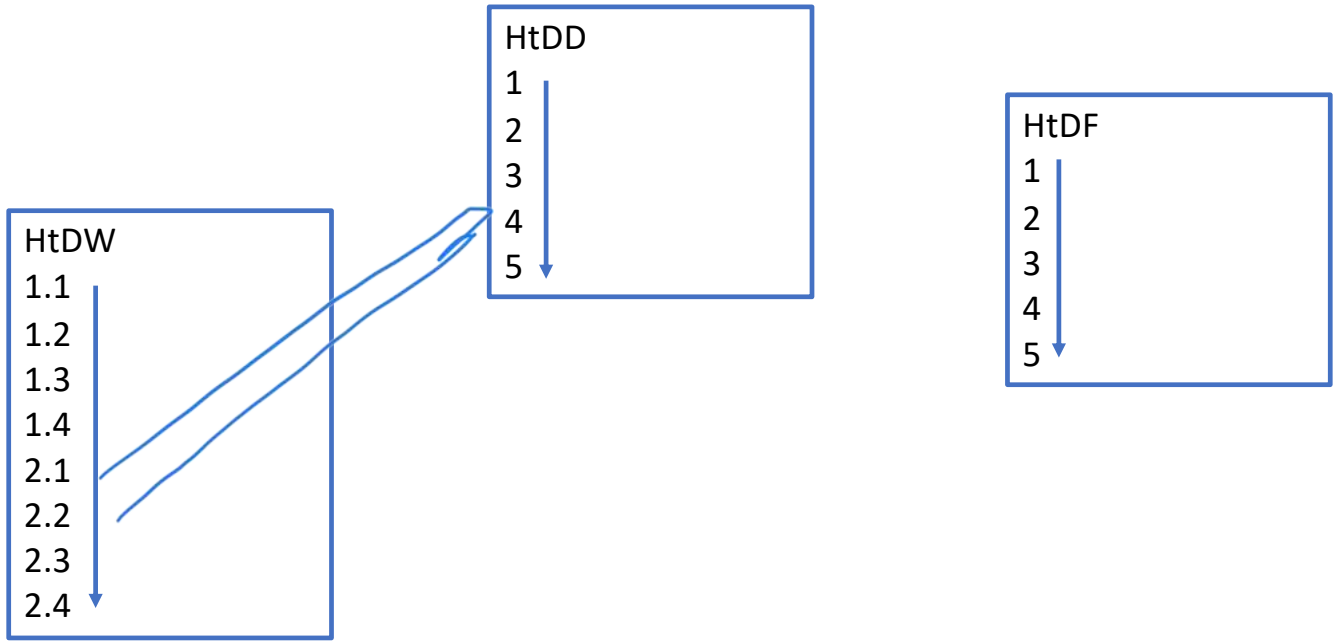
```
(@problem 1)
(@htdd GradeStanding)
;; GradeStanding is one of:
;;  - Natural
;;  - "H"
;;  - "P"
;;  - "F"
;;  - "T"
;; interp. a percent grade OR standing
;; CONSTRAINT: If natural is in [0, 100]
(define GS1 10)
(define GS2 "H")

(@dd-template-rules one-of                ;5 cases
                    atomic-non-distinct   ;Natural
                    atomic-distinct       ;"H"
                    atomic-distinct       ;"P"
                    atomic-distinct       ;"F"
                    atomic-distinct)      ;"T"

(define (fn-for-grade-standing gs)
  (cond [(number? gs) (... gs)]
        [(string=? gs "H") (...)]
        [(string=? gs "P") (...)]
        [(string=? gs "F") (...)]
        [else (...)]))
```

Design methods tell us
how & when to break problem into smaller pieces

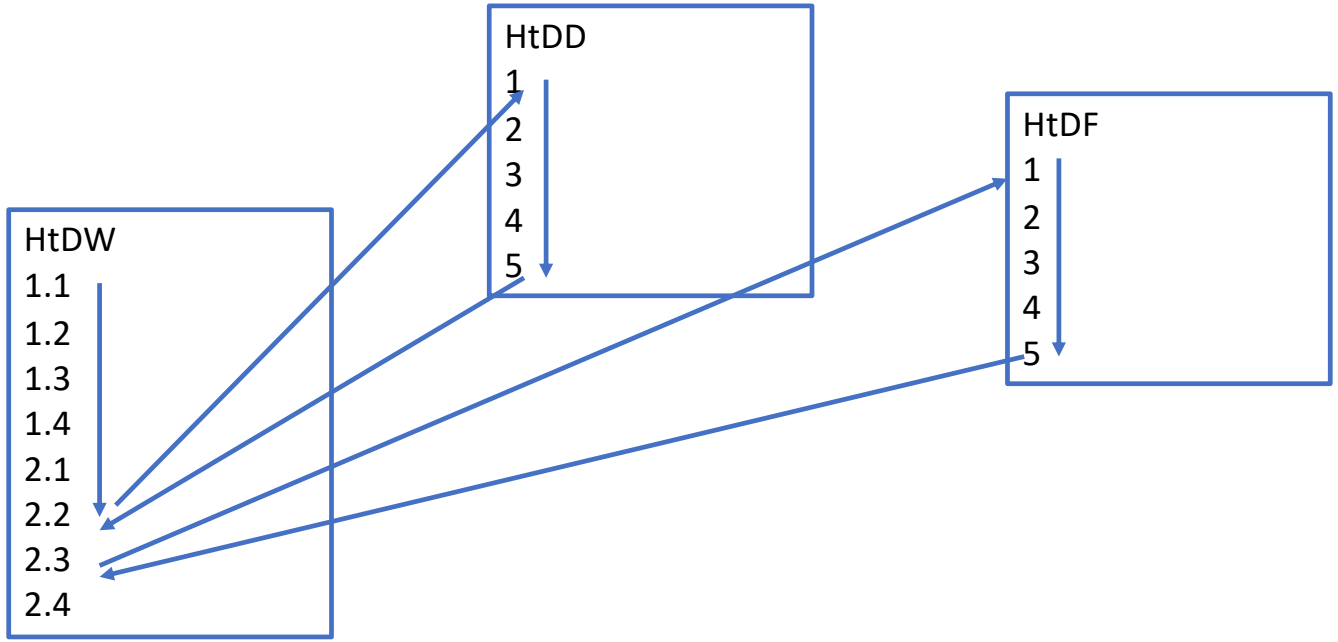




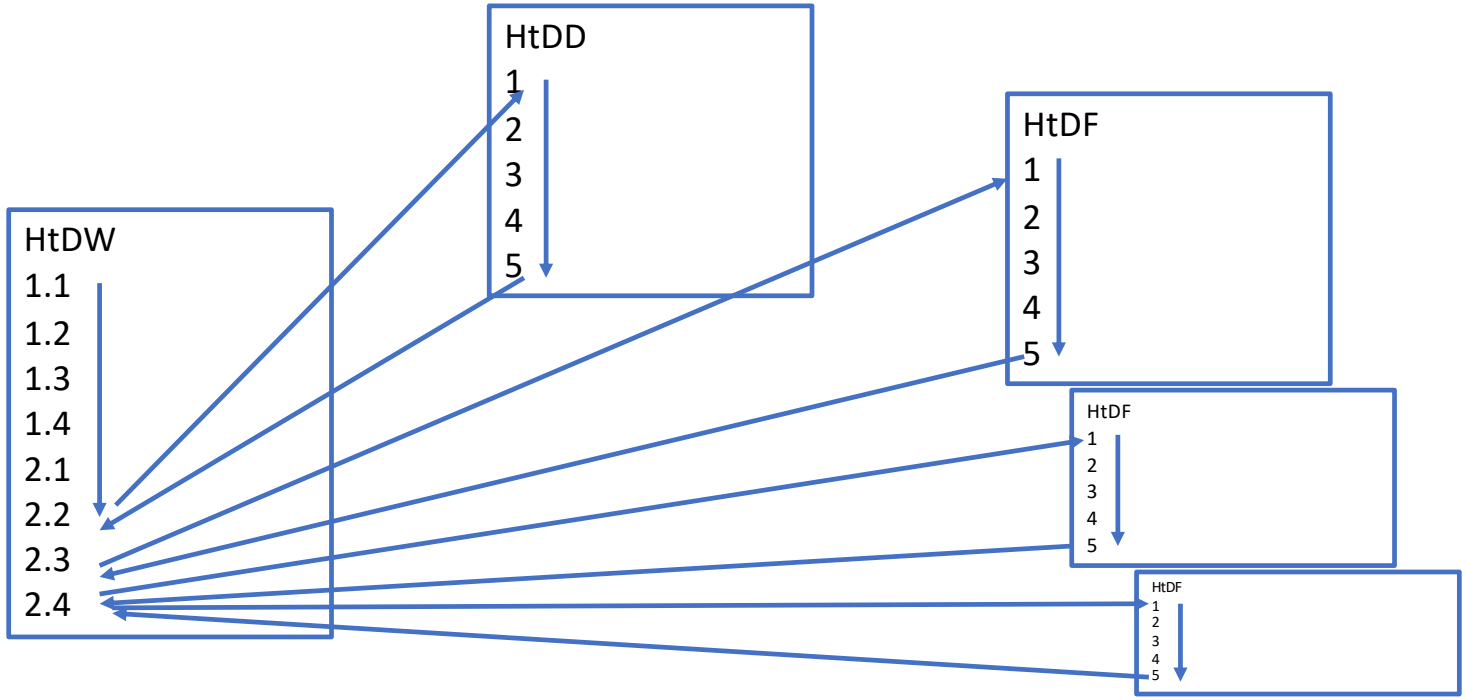
Design methods tell us
how & when to break problem into smaller pieces



Trust the recipes
to help you make progress
Always ask: what's next step of current recipe?



Trust the recipes
to help you make progress
Always ask: what's next step of current recipe?



SPD Checklists

See full recipe page for details

```
(require spd/tags)
(require 2htdp/image)
(require 2htdp/universe)

;; My world program  (make this more specific)
(@htdw WS)
;; ================
;; Constants:


;; ================
;; Data definitions:

(@htdd WS)
;; WS is ... (give WS a better name)



;; ================
;; Functions:

(@htdf main)
(@signature WS -> WS)
;; start the world with (main ...)
;;
(@template-origin htdw-main)
(define (main ws)
  (big-bang ws          ;WS
    (on-tick   tock)    ;WS -> WS
    (to-draw   render) ;WS -> Image
    (on-mouse  ...)     ;WS Integer Integer MouseEvent -> WS
    (on-key    ...)))  ;WS KeyEvent -> WS

(@htdf tock)
(@signature WS -> WS)
;; produce the next ...
;; !!!
(define (tock ws) ws)

(@htdf render)
(@signature WS -> Image)
;; render ...
;; !!!
(define (render ws) empty-image)
```

HtDW

1. Domain analysis (use a piece of paper!)
   1. Sketch program scenarios
   2. Identify constant information
   3. Identify changing information
   4. Identify big-bang options
2. Build the actual program
   1. Constants (based on 1.2 above)
   2. Data definitions (based on 1.3 above)
   3. Functions
      1. main first (based on 1.4 and 2.2 above)
      2. wish list entries for big-bang handlers
   4. Work through wish list until done

```
on-tick
to-draw
on-key
on-mouse
```

HtDD

First identify form of information, then write:

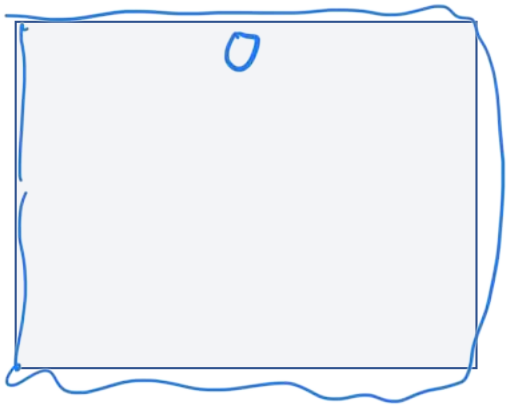
1. A possible structure definition (not until compound data)
2. A type comment that defines type name and describes how to form data
3. An interpretation to describe correspondence between information and data.
4. One or more examples of the data.
5. A template for a 1 argument function operating on data of this type.

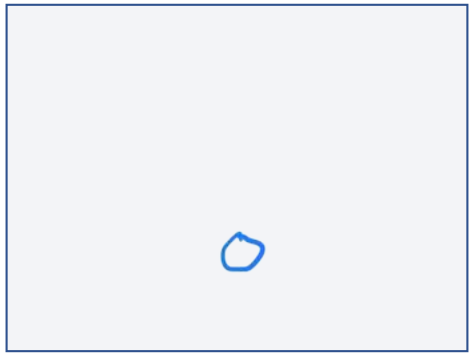
HtDF

1. Signature, purpose and stub.
2. Define examples, wrap each in check-expect.
3. Template and inventory.
4. Code the function body.
5. Test and debug until correct

Test guidelines

1. at least 2
2. different argument/field values
3. code coverage
4. points of variation in behavior
5. 2 long / 2 deep





Don’t worry, handwriting will be magically cleaned up!

Constant

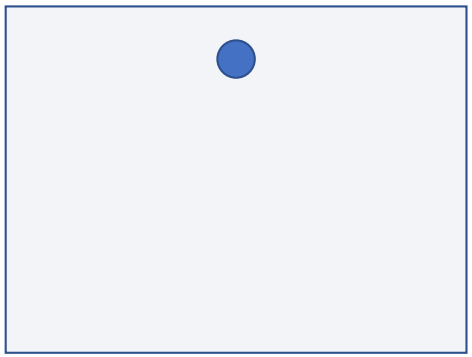
width
height
ctr-x
speed
mts dir
spider image

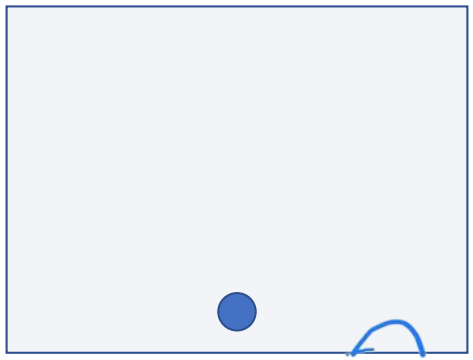
Changing

spider y

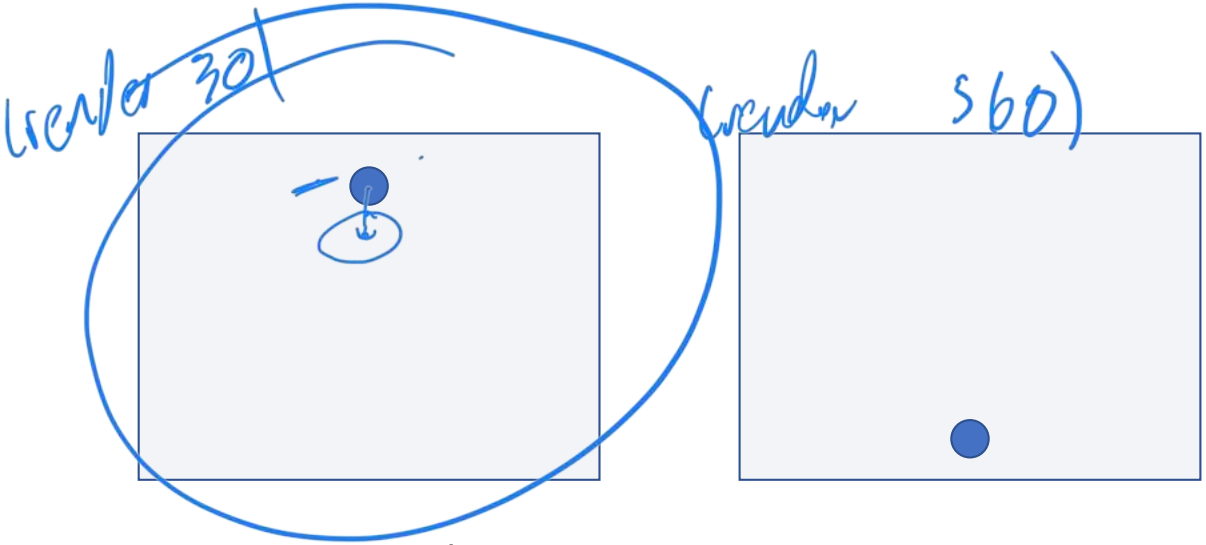
BB options

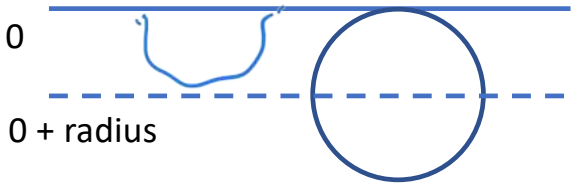
on-tick
to-draw
~~on-key~~
~~on-mouse~~



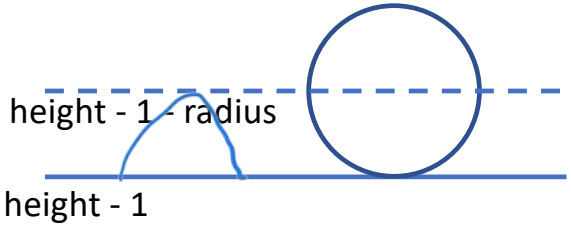


Constant	Changing	BB options
width	spider y	on-tick
height		to-draw
center x		~~on-key~~
speed		~~on-mouse~~
spider radius		
spider image		
mts		





Constant	Changing	BB options
width	spider y	on-tick
height		to-draw
center x		~~on-key~~
speed		~~on-mouse~~
spider radius		
spider image		
mts		



SPD Checklists

See full recipe page for details

```
(require spd/tags)
(require 2htdp/image)
(require 2htdp/universe)

;; My world program  (make this more specific)
(@htdw WS)
;; ================
;; Constants:


;; ================
;; Data definitions:

(@htdd WS)
;; WS is ... (give WS a better name)



;; ================
;; Functions:

(@htdf main)
(@signature WS -> WS)
;; start the world with (main ...)
;;
(@template-origin htdw-main)
(define (main ws)
  (big-bang ws          ;WS
    (on-tick   tock)    ;WS -> WS
    (to-draw   render) ;WS -> Image
    (on-mouse  ...)     ;WS Integer Integer MouseEvent -> WS
    (on-key    ...)))  ;WS KeyEvent -> WS

(@htdf tock)
(@signature WS -> WS)
;; produce the next ...
;; !!!
(define (tock ws) ws)

(@htdf render)
(@signature WS -> Image)
;; render ...
;; !!!
(define (render ws) empty-image)
```

HtDW

1. Domain analysis (use a piece of paper!)
    1. Sketch program scenarios
    2. Identify constant information
    3. Identify changing information
    4. Identify big-bang options
2. Build the actual program
    1. Constants (based on 1.2 above)
    2. Data definitions (based on 1.3 above)
    3. Functions
        1. main first (based on 1.4 and 2.2 above)
        2. wish list entries for big-bang handlers
    4. Work through wish list until done

```
on-tick
to-draw
on-key
on-mouse
```

HtDD

First identify form of information, then write:

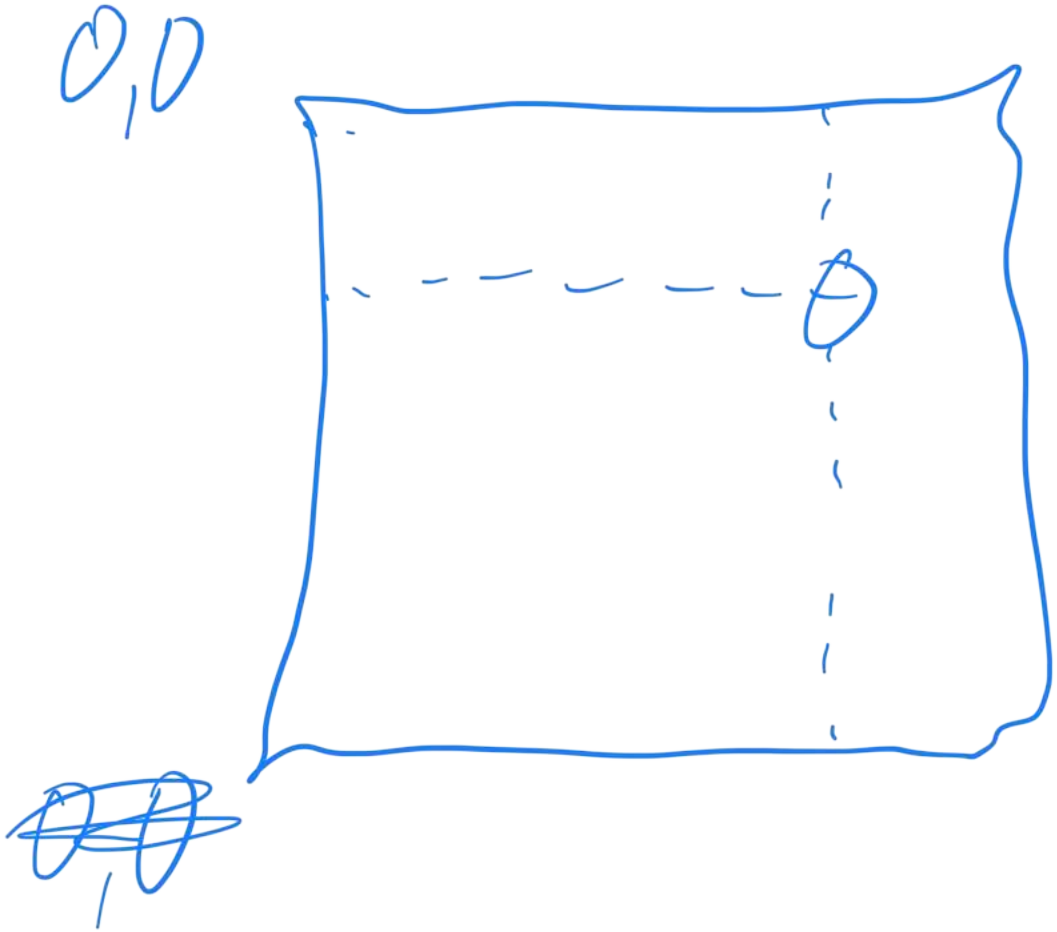
1. A possible structure definition (not until compound data)
2. A type comment that defines type name and describes how to form data
3. An interpretation to describe correspondence between information and data.
4. One or more examples of the data.
5. A template for a 1 argument function operating on data of this type.

HtDF

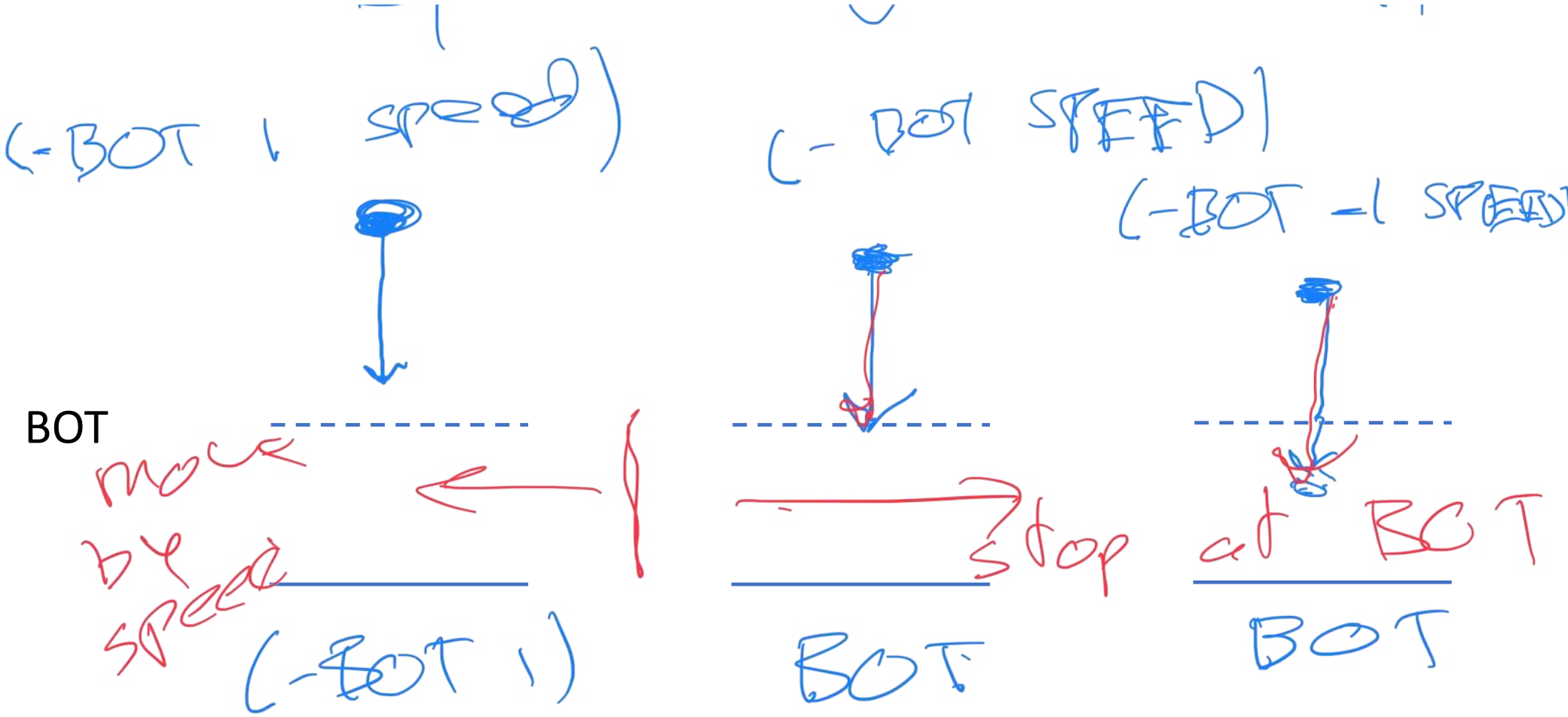
1. Signature, purpose and stub.
2. Define examples, wrap each in check-expect.
3. Template and inventory.
4. Code the function body.
5. Test and debug until correct

Test guidelines

1. at least 2
2. different argument/field values
3. code coverage
4. points of variation in behavior
5. 2 long / 2 deep



boundary case analysis for tock



boundary case analysis for tock