

CMPU 101 §3 · Computer Science I

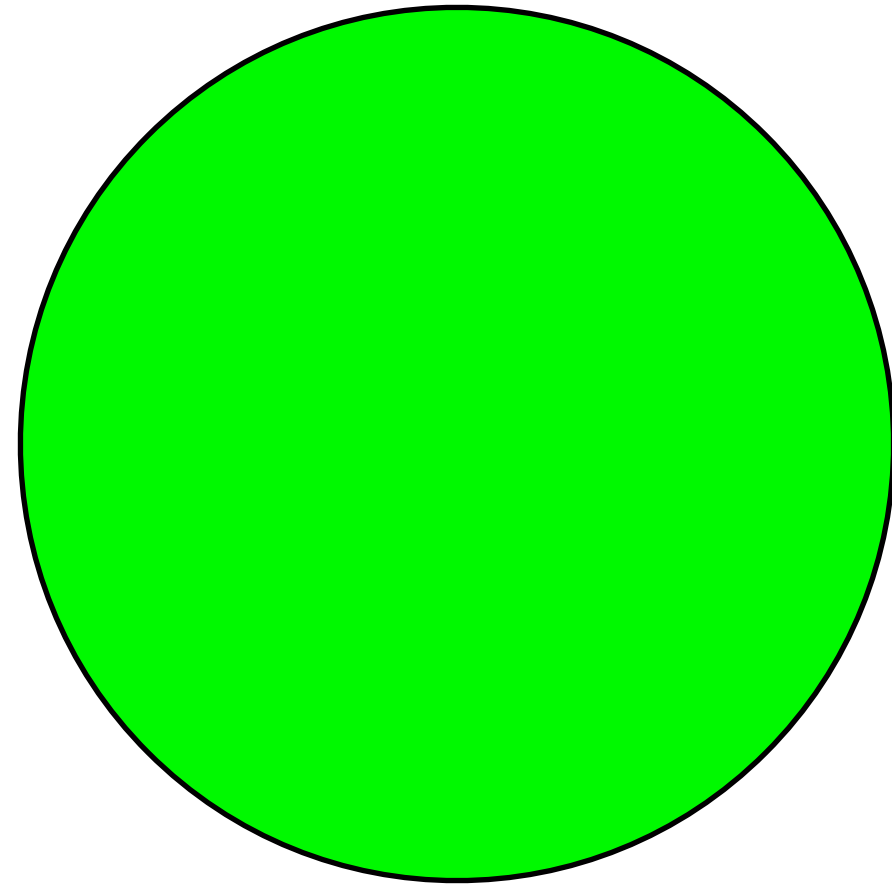
Reactors

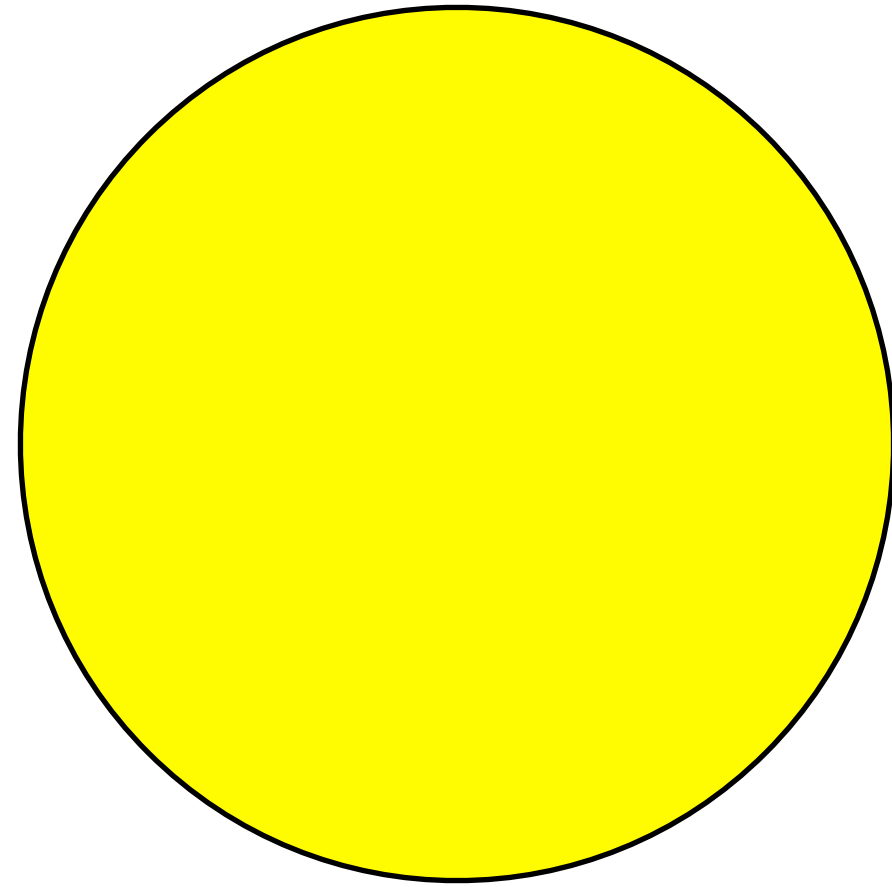
31 October 2022

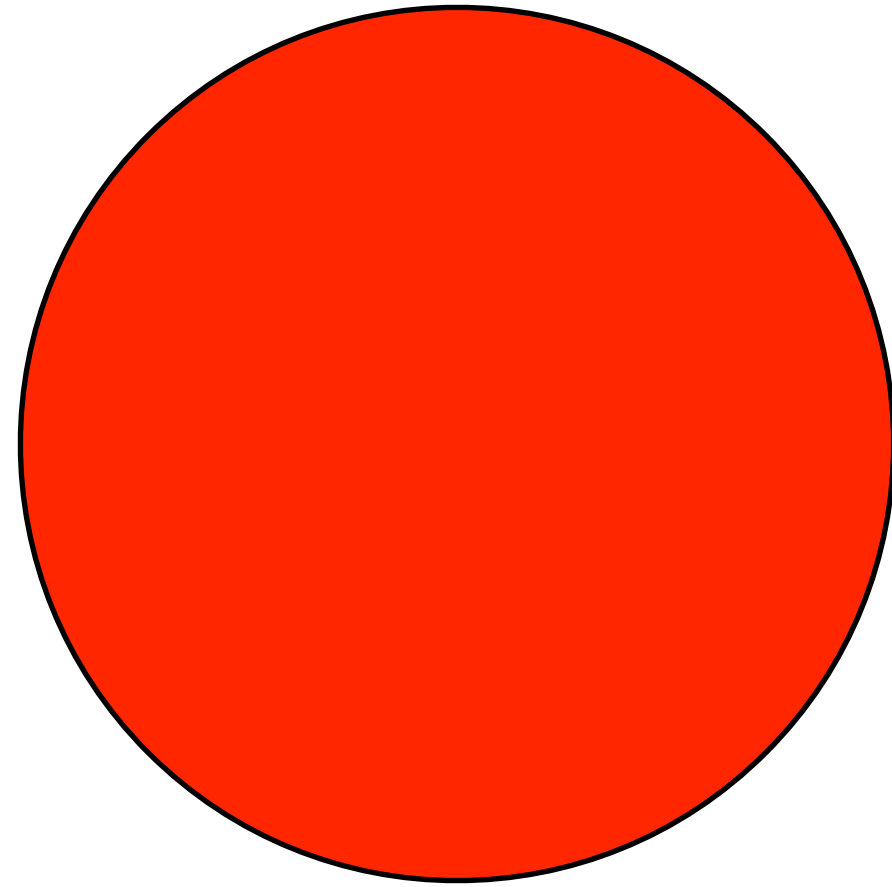


Where are we?

Where are we?
Traffic-light world







All traffic lights are the same size and position on the screen.

What distinguishes them?

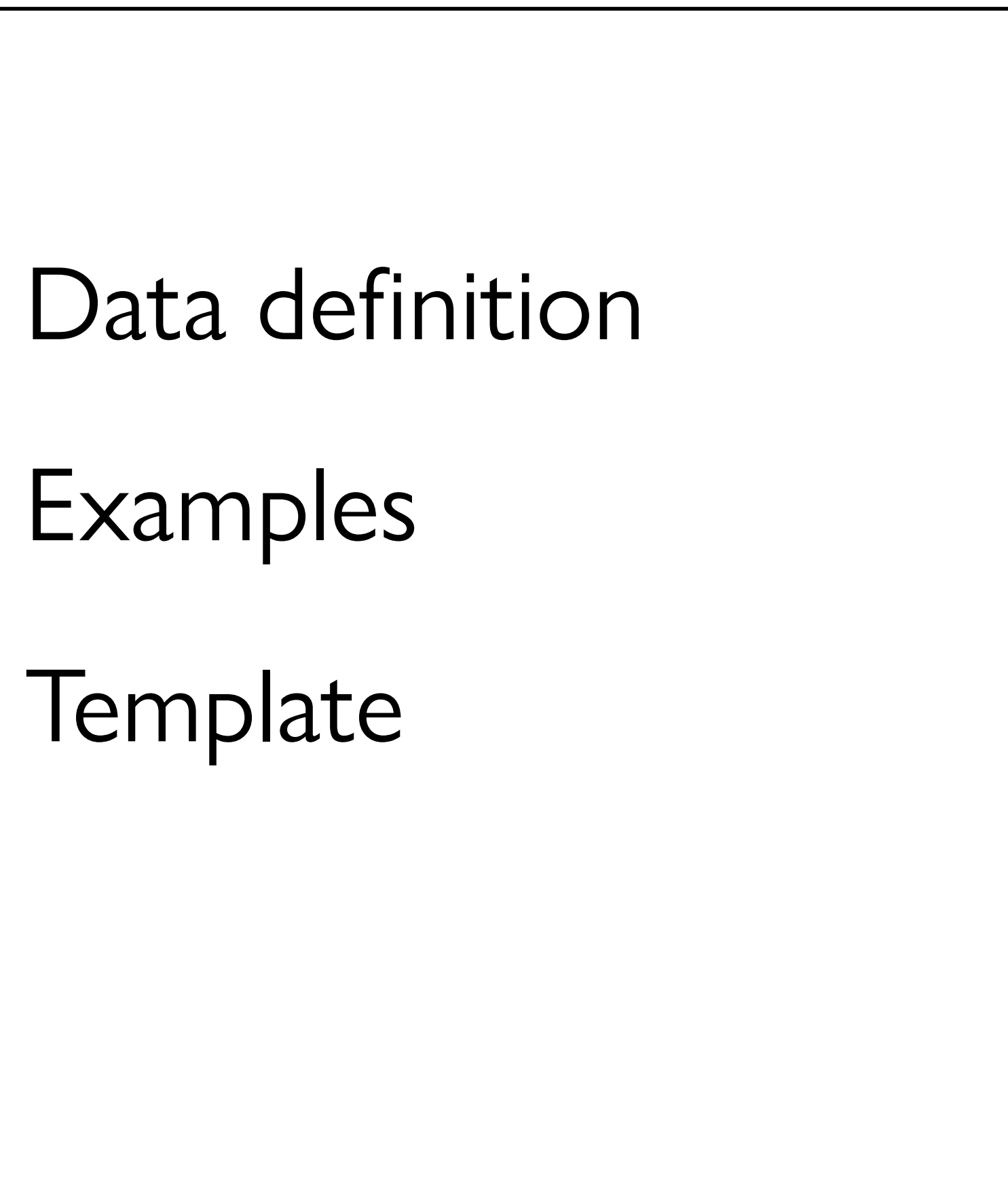
*Asking this helps us think about **data***

All traffic lights are the same size and position on the screen.

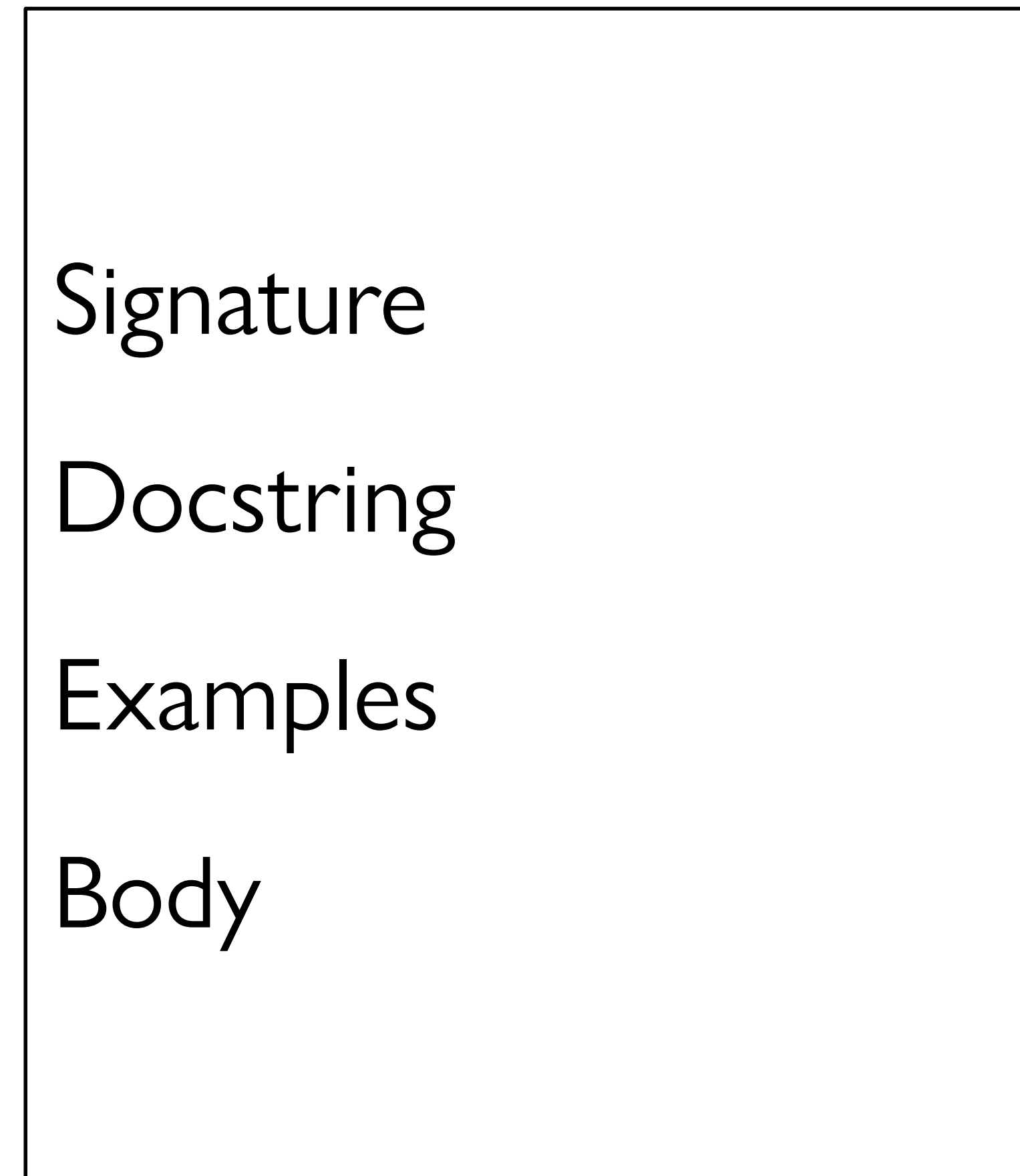
How do we get from one to the other?

*Asking this helps us think about **functions***

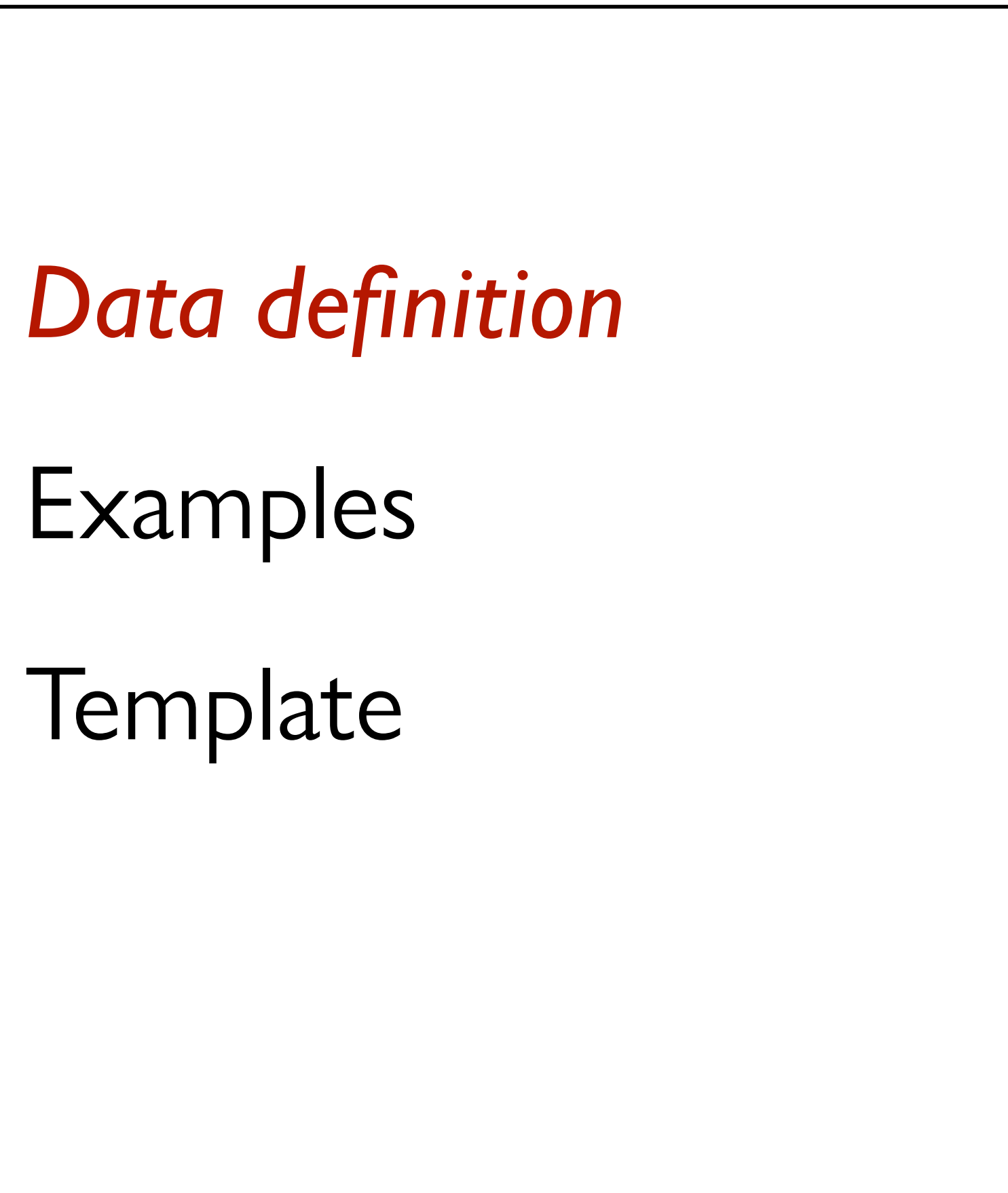
Data



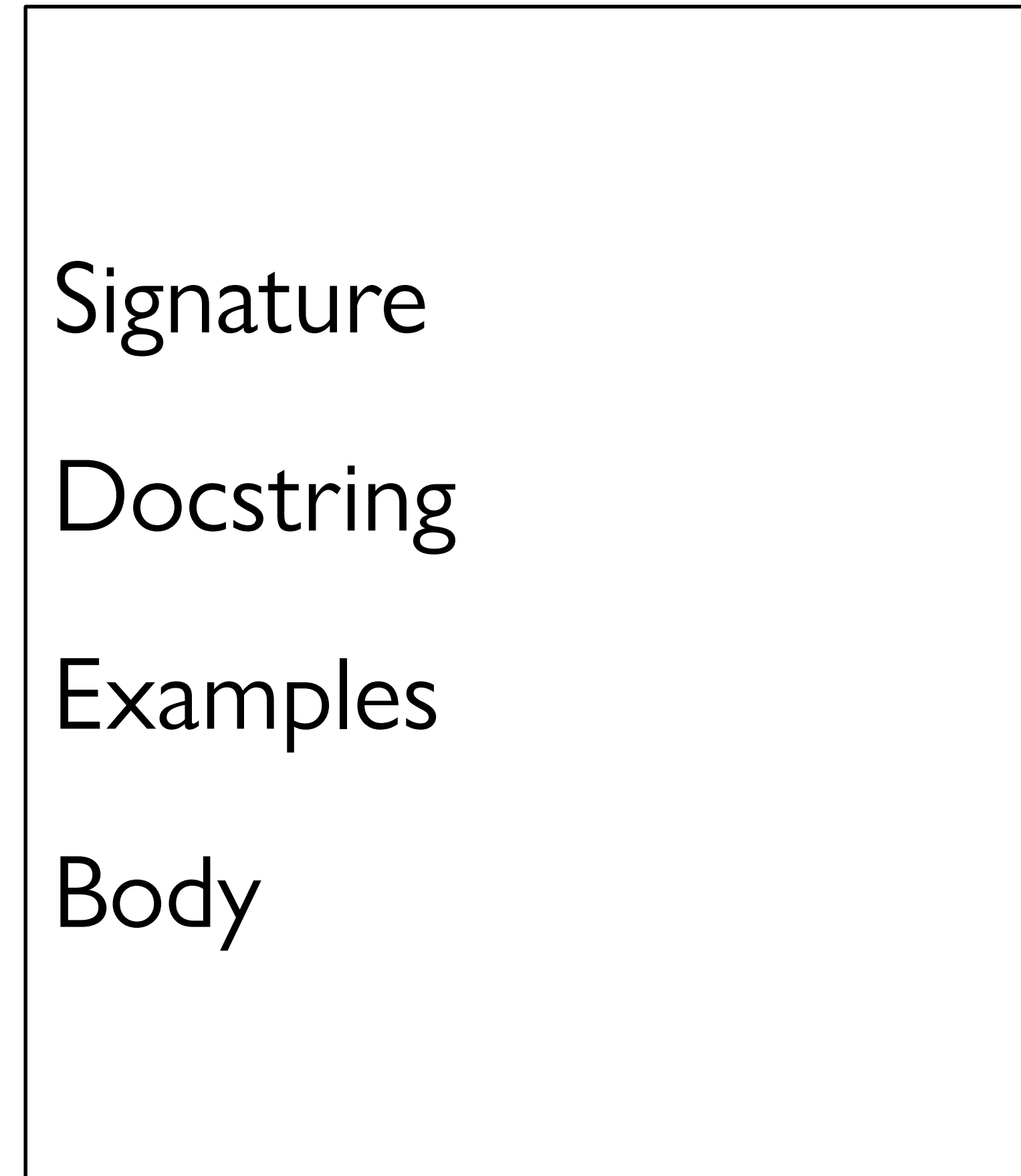
Functions



Data



Functions



```
data TrafficLight:
```

```
    . . .
```

```
end
```

```
data TrafficLight:
```

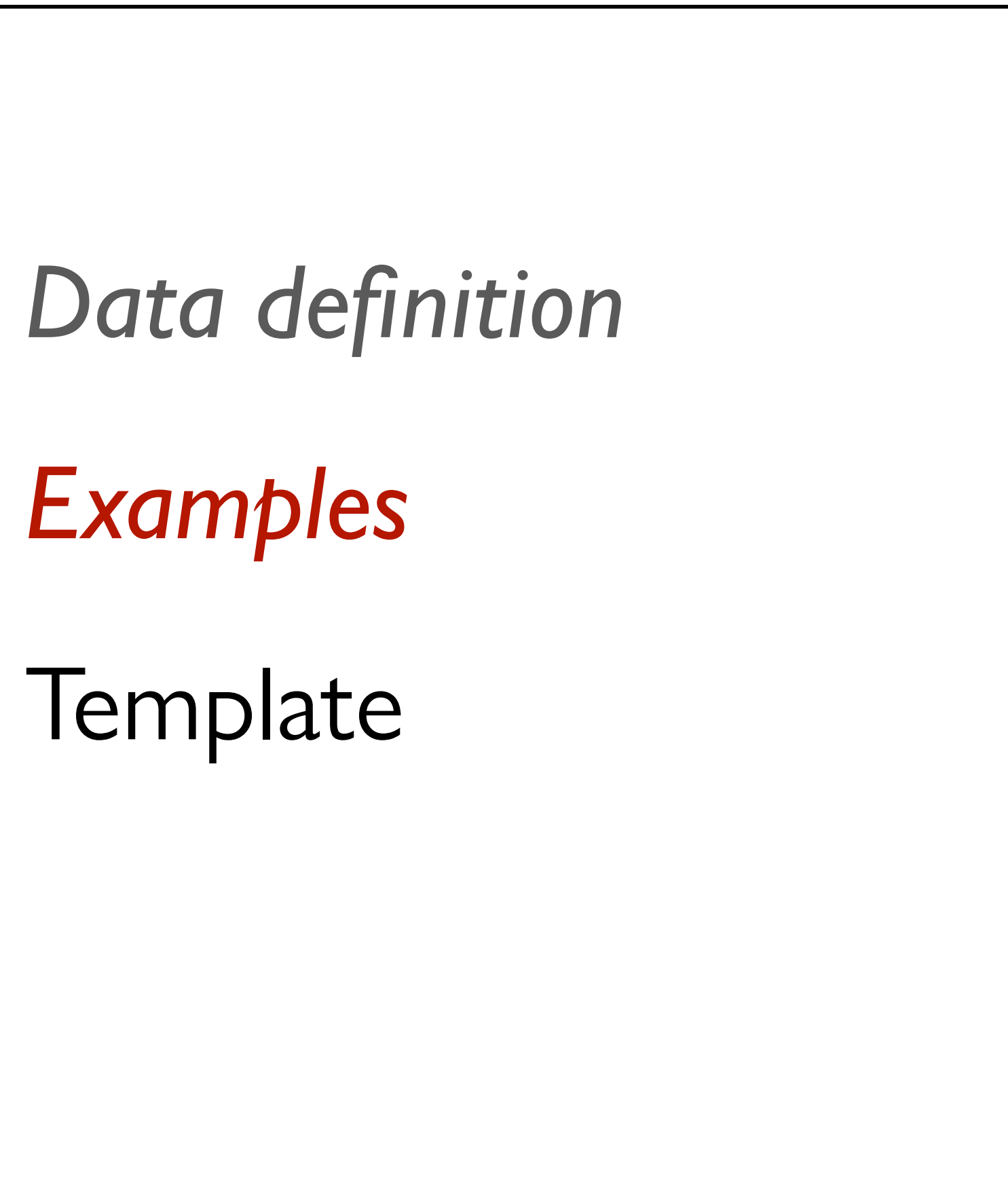
```
  | green
```

```
  | yellow
```

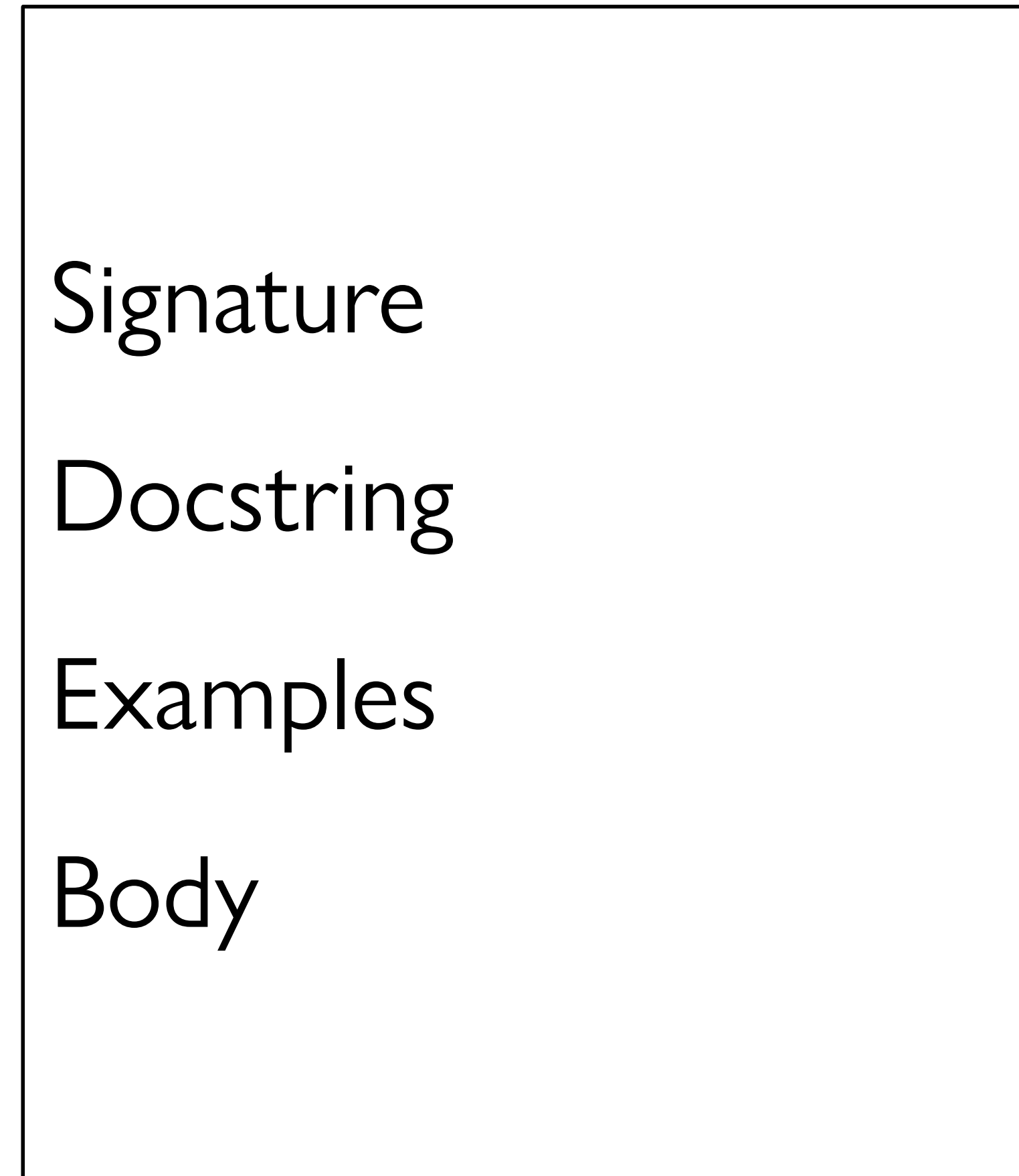
```
  | red
```

```
end
```

Data



Functions



```
data TrafficLight:  
  | green  
  | yellow  
  | red  
end
```

```
TL-GREEN = green  
TL-YELLOW = yellow  
TL-RED = red
```

For this data definition, the examples are so trivial we can skip them, but you saw in lab on Friday how helpful it can be to have examples when you have a lot of possibilities!

```
data TrafficLight:
```

```
  | green
```

```
  | yellow
```

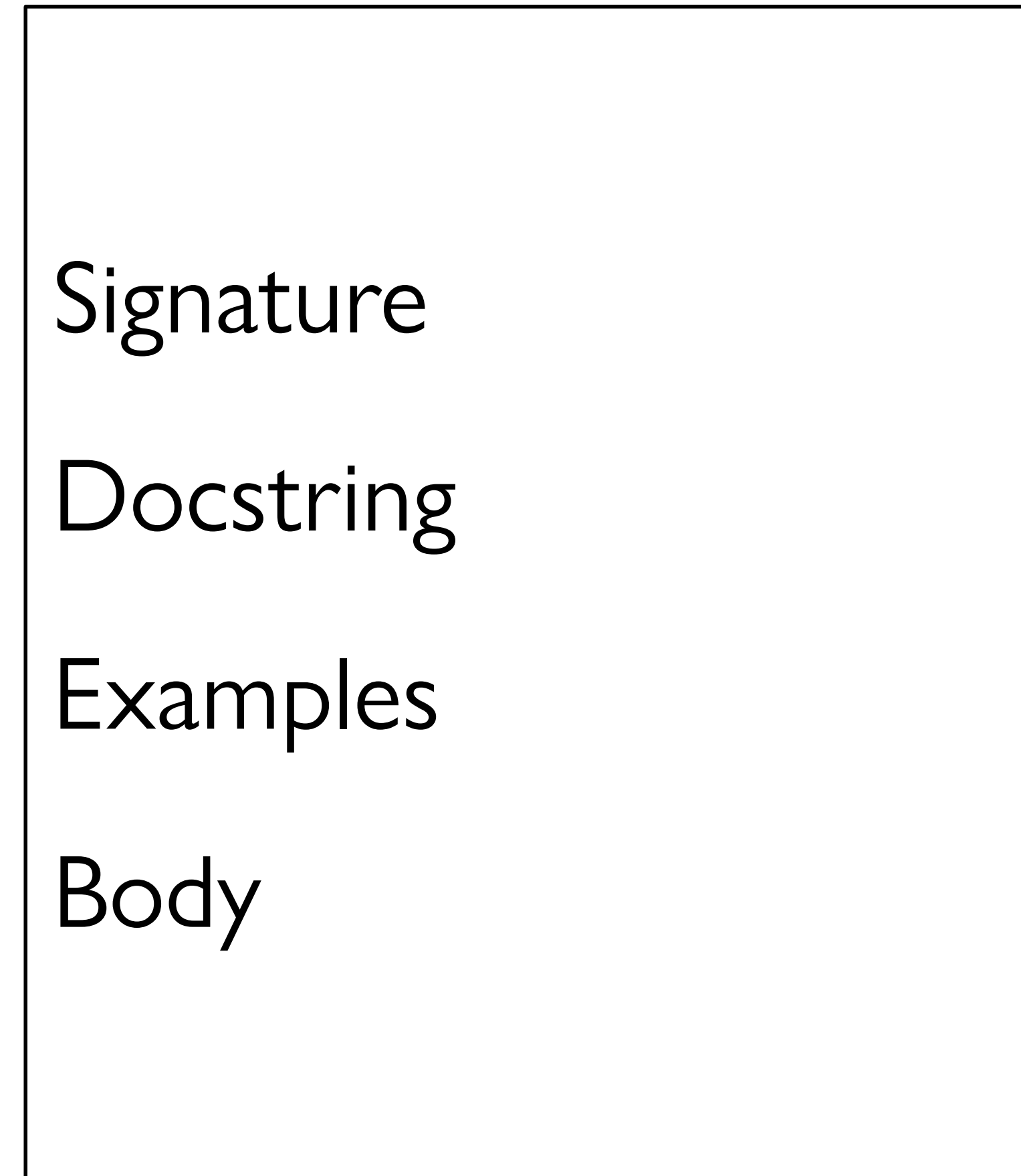
```
  | red
```

```
end
```

Data



Functions




```
data TrafficLight:
```

```
  | green
```

```
  | yellow
```

```
  | red
```

```
end
```

```
data TrafficLight:
```

```
  | green
```

```
  | yellow
```

```
  | red
```

```
end
```

```
#|
```

```
fun trafficlight-fun(tl :: TrafficLight) -> ...:
```

```
|#
```

```
data TrafficLight:
```

```
  | green  
  | yellow  
  | red
```

```
end
```

```
#|
```

```
fun trafficlight-fun(tl :: TrafficLight) -> ...:
```

```
  doc: "TrafficLight template"
```

```
  cases (TrafficLight) tl:
```

```
    | green => ...
```

```
    | yellow => ...
```

```
    | red => ...
```

```
  end
```

```
where:
```

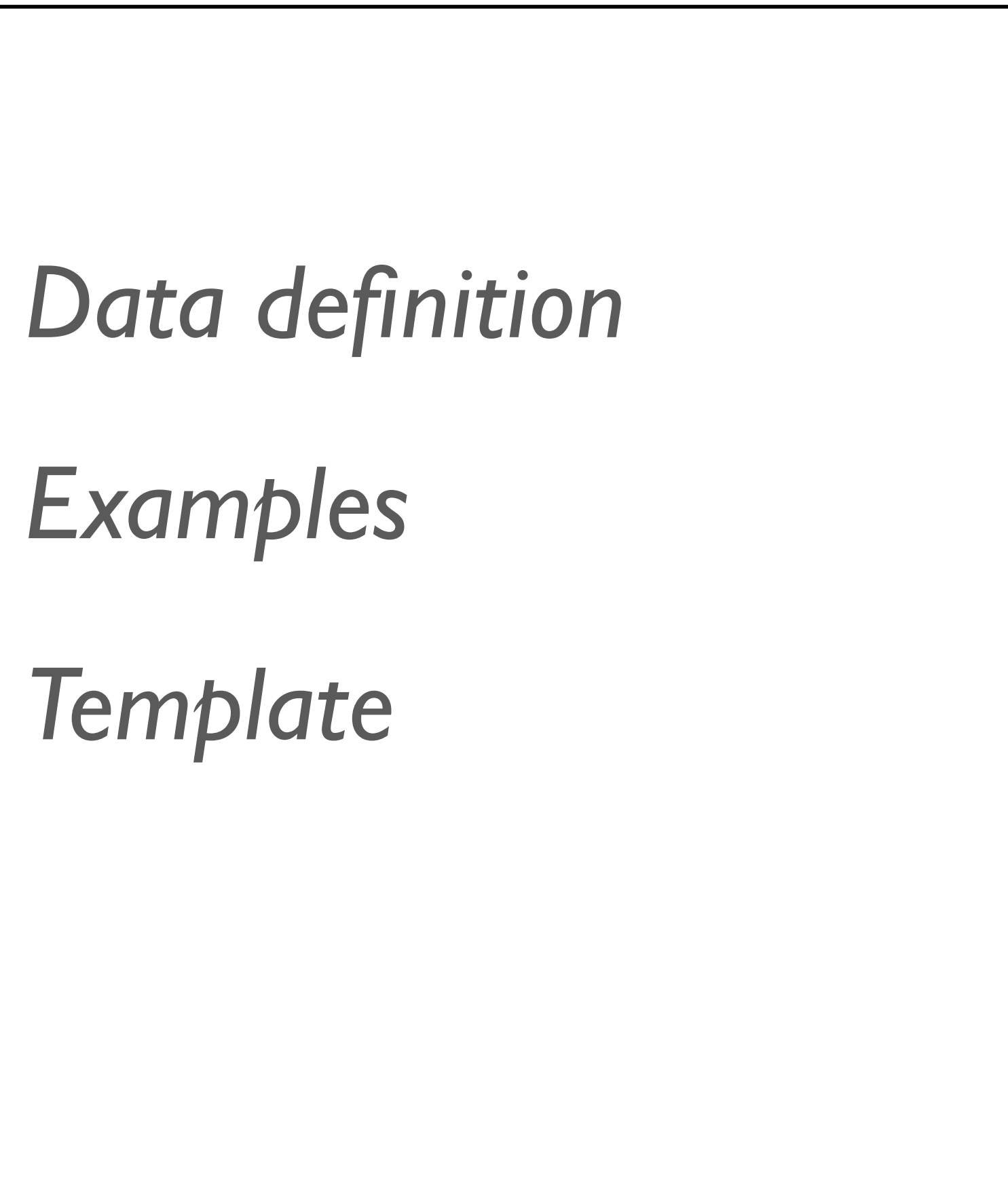
```
  trafficlight-fun(green) is ...
```

```
  trafficlight-fun(yellow) is ...
```

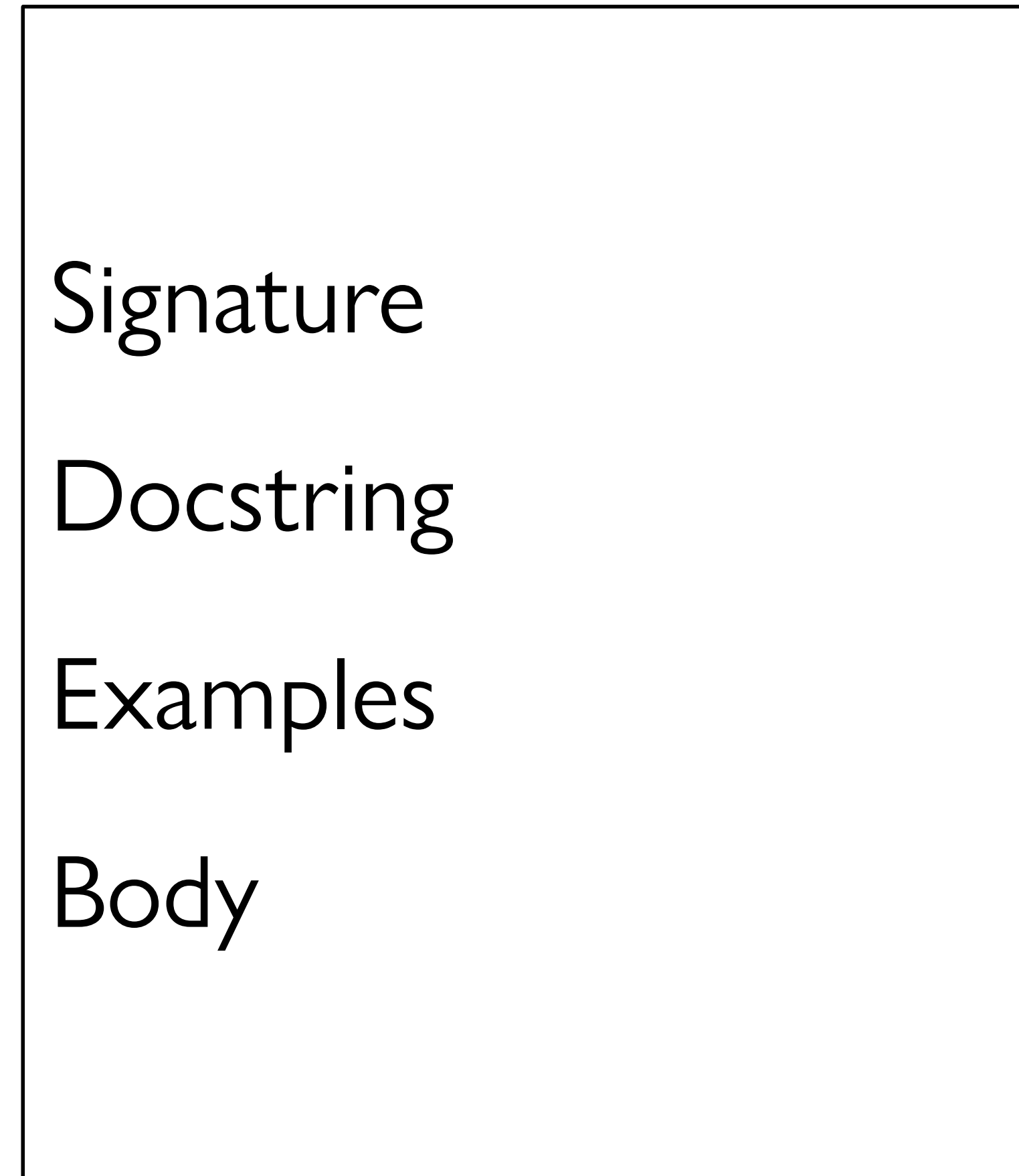
```
  trafficlight-fun(red) is ...
```

```
end |#
```

Data



Functions



Pyret has a mechanism for supporting interactive programs, called a **reactor**.

To use it, first write

```
include reactors
```

```
reactor:  
  init: initial-state  
  to-draw: draw-function  
  event-type: event-function  
end
```

reactor:

init: *initial-state*

to-draw: *draw-function*

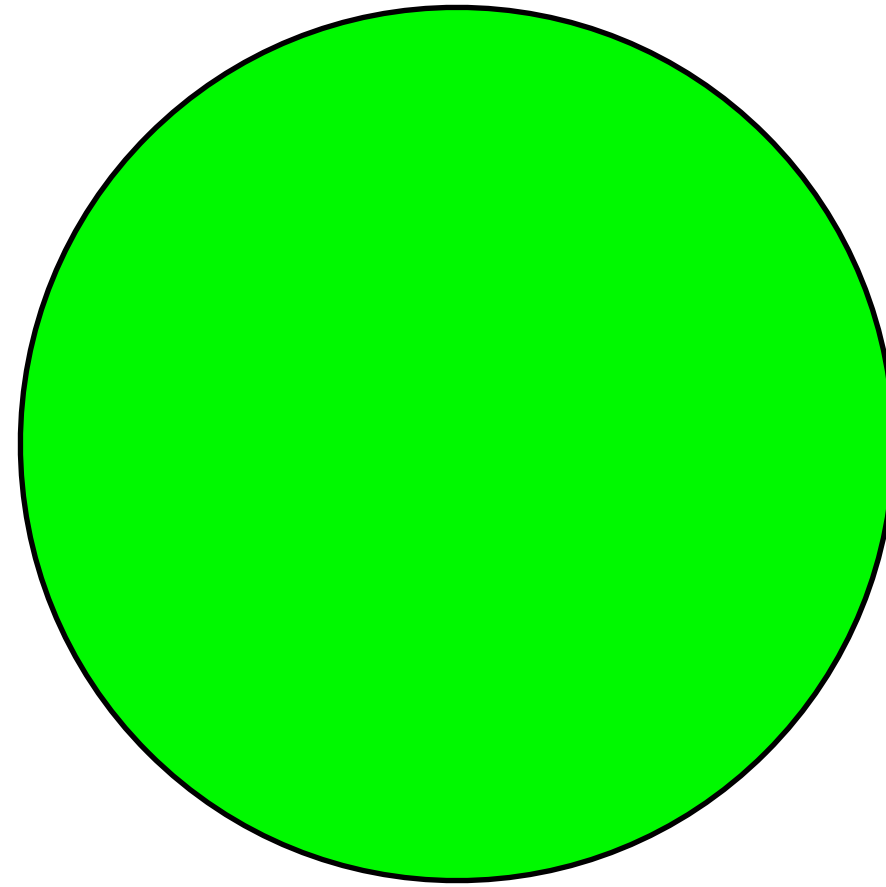
event-type: *event-function*

end



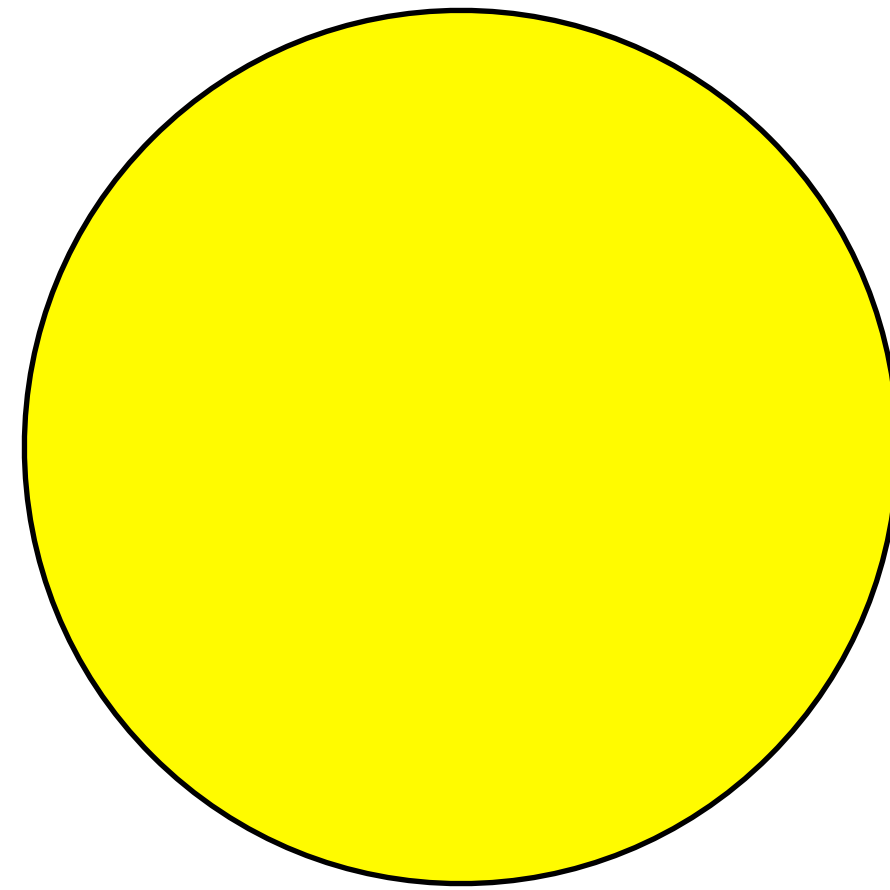
Less nuclear reactor; more person-that-reacts to something.

reactor puts all the pieces together to start things going.

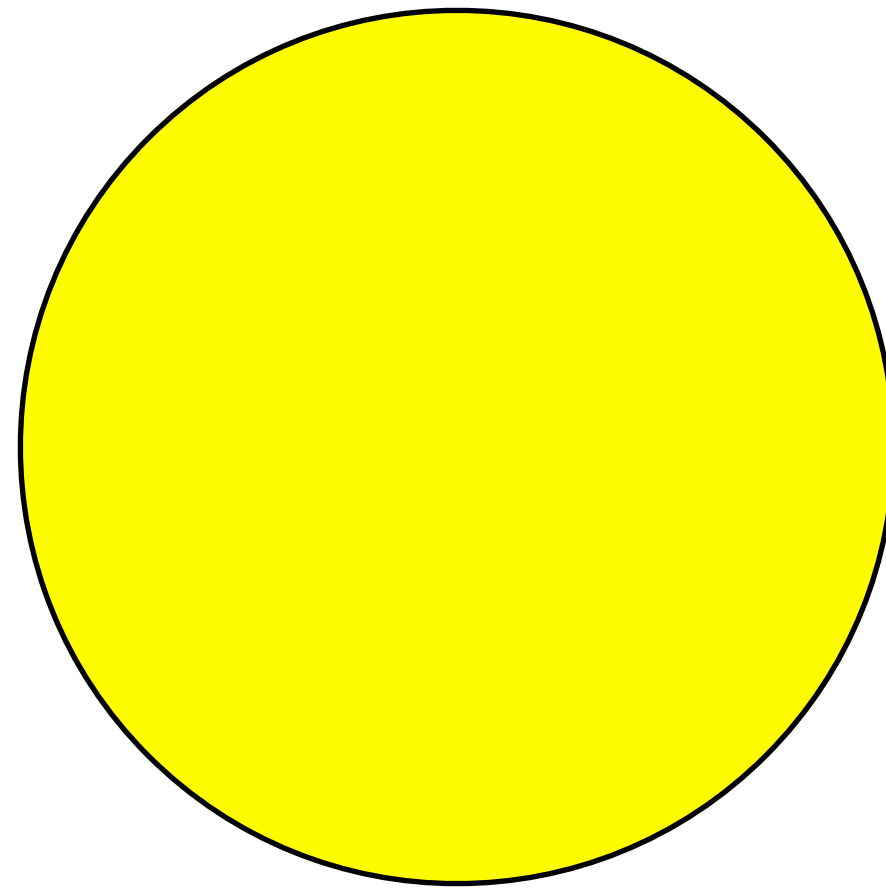


initial state

some event happens...



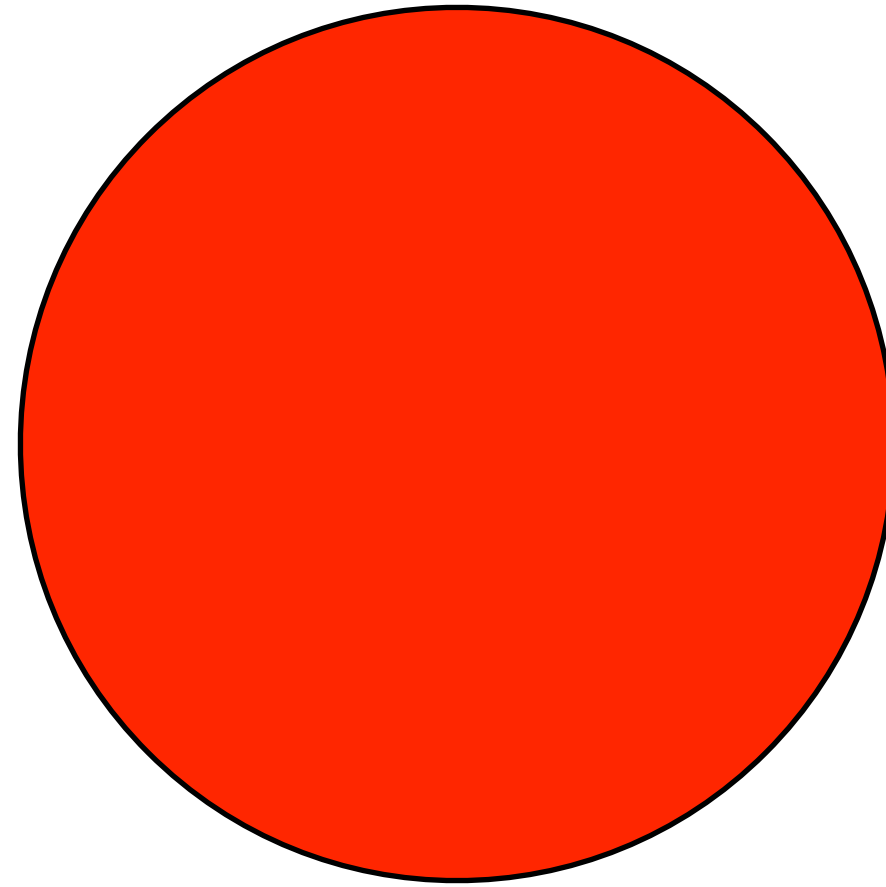
next state



~~next state~~

now the current state

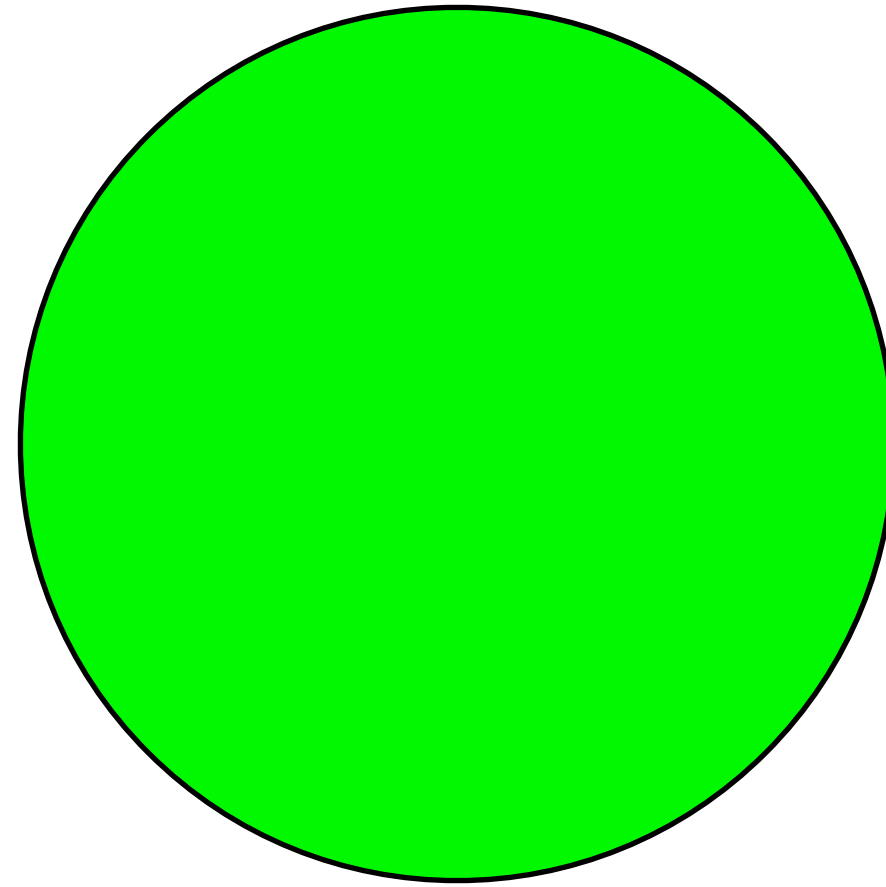
some event happens...



~~next state~~

now the current state

some event happens...



~~next state~~

now the current state

```
reactor:  
  init: initial-state,  
  to-draw: draw-function,  
  event-type: event-function  
end
```

```
reactor:  
  init: green,  
  to-draw: draw-function,  
  event-type: event-function  
end
```

```
reactor:  
  init: green,  
  to-draw: draw-light,  
  event-type: event-function  
end
```

*We haven't written this;
add it to our wishlist!*

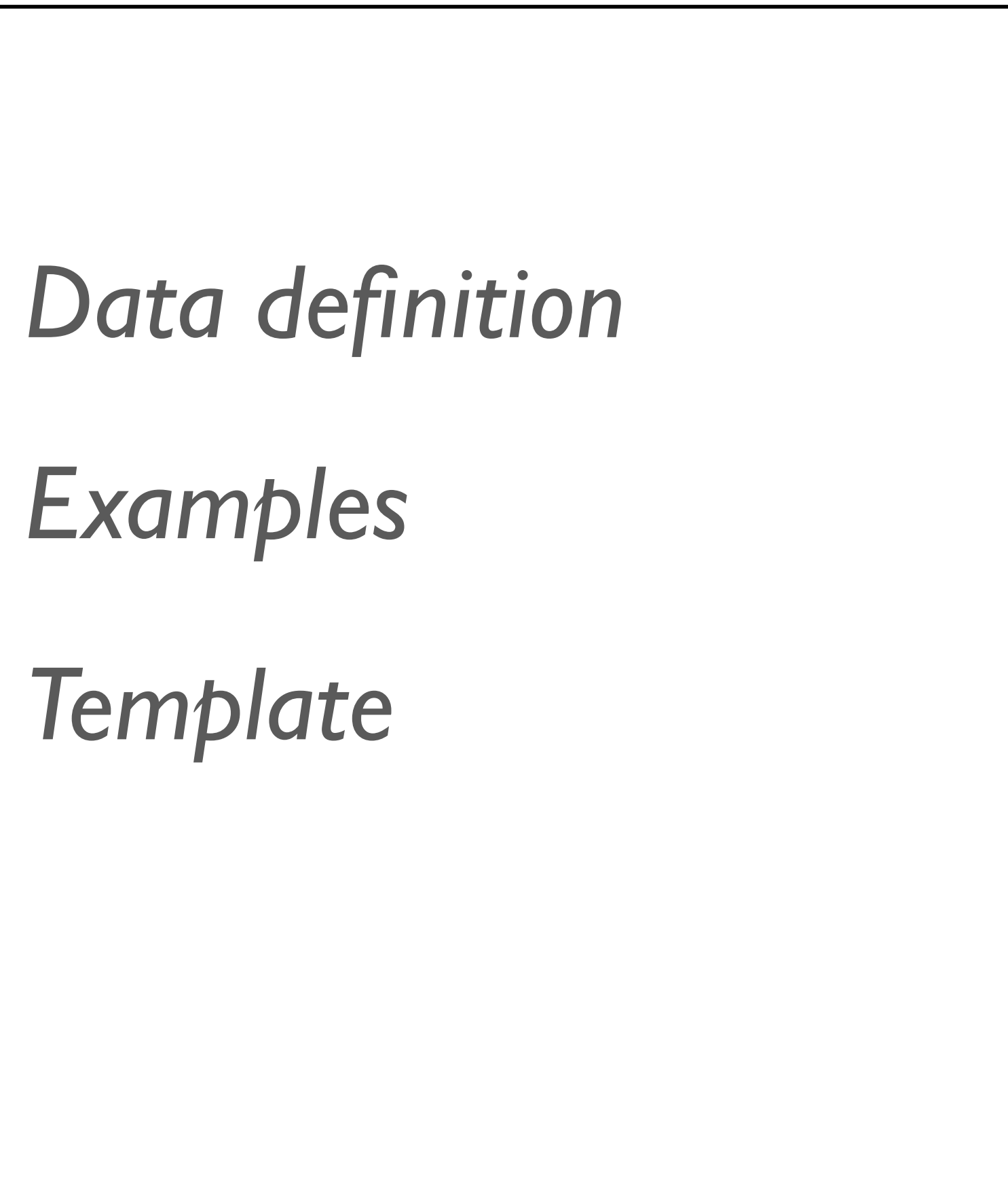
```
reactor:  
  init: green,  
  to-draw: draw-light,  
  on-tick: next-light  
end
```

*Another function for the
wishlist!*

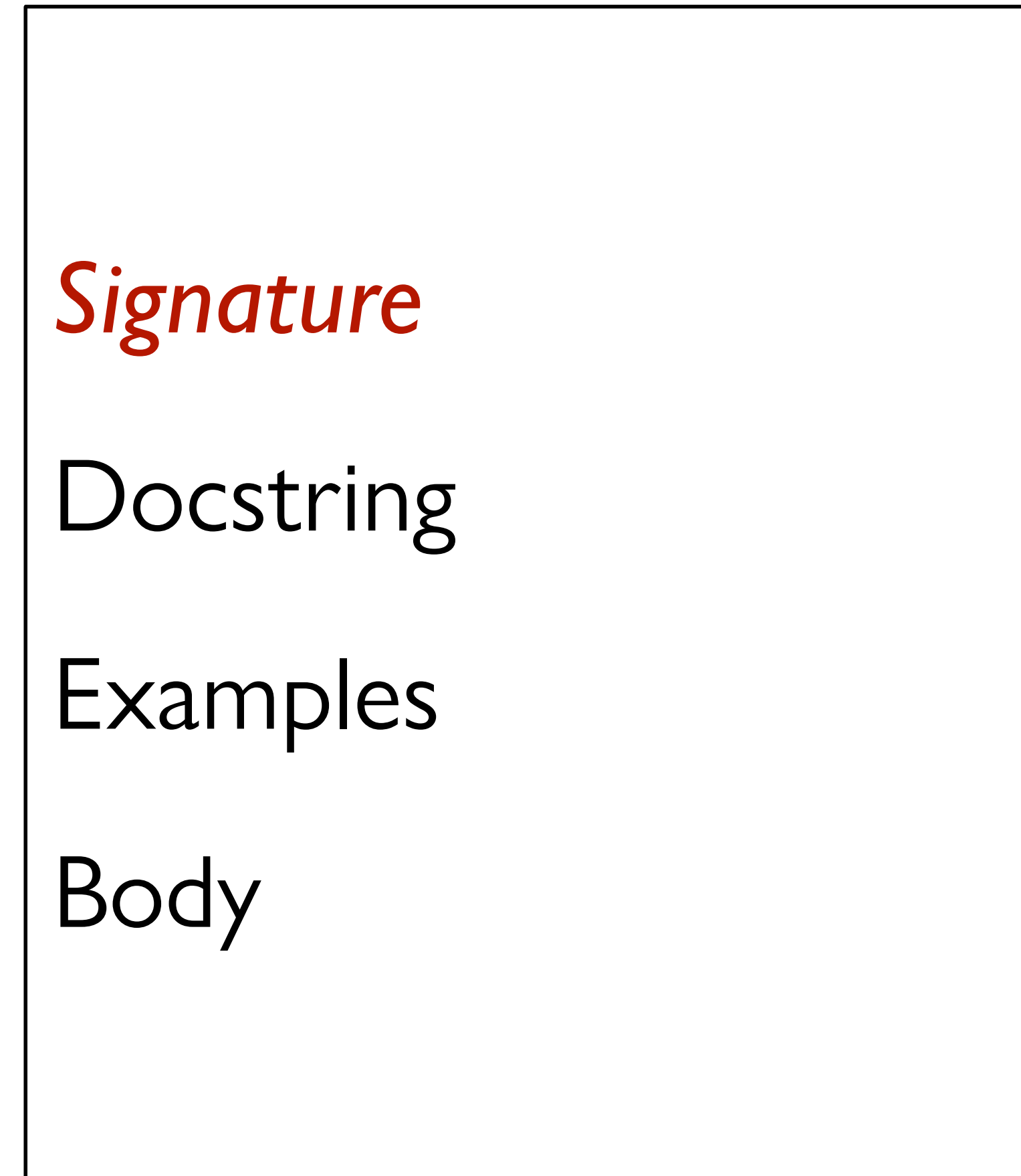
So far...

```
# TrafficLight data  
# - definition  
# - examples  
# - template  
  
# define reactor  
  
# Wishlist:  
# - fun draw-light...  
# - fun next-light...
```

Data



Functions



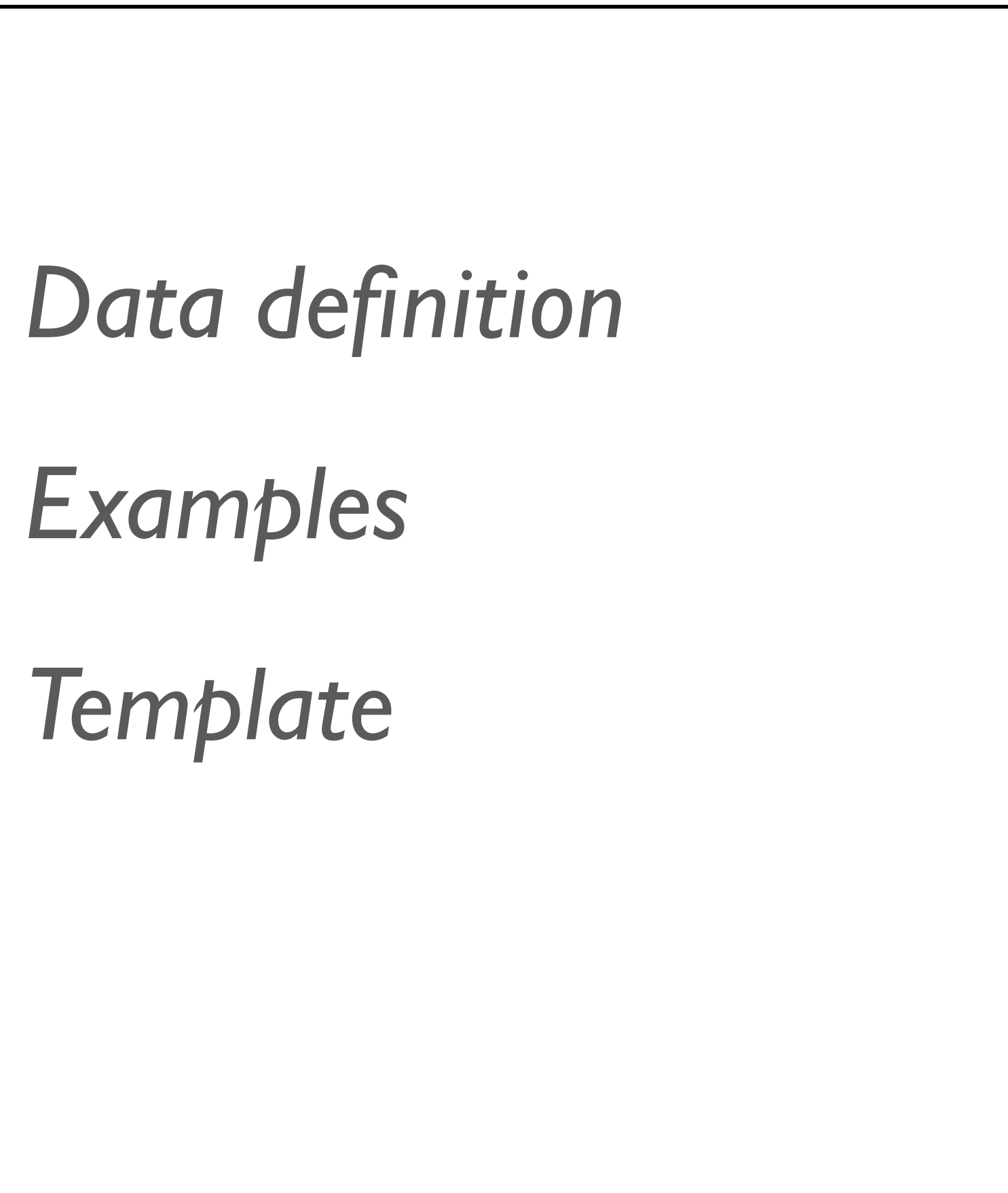
```
fun draw-light(tl :: TrafficLight) -> Image:  
  ...  
end
```



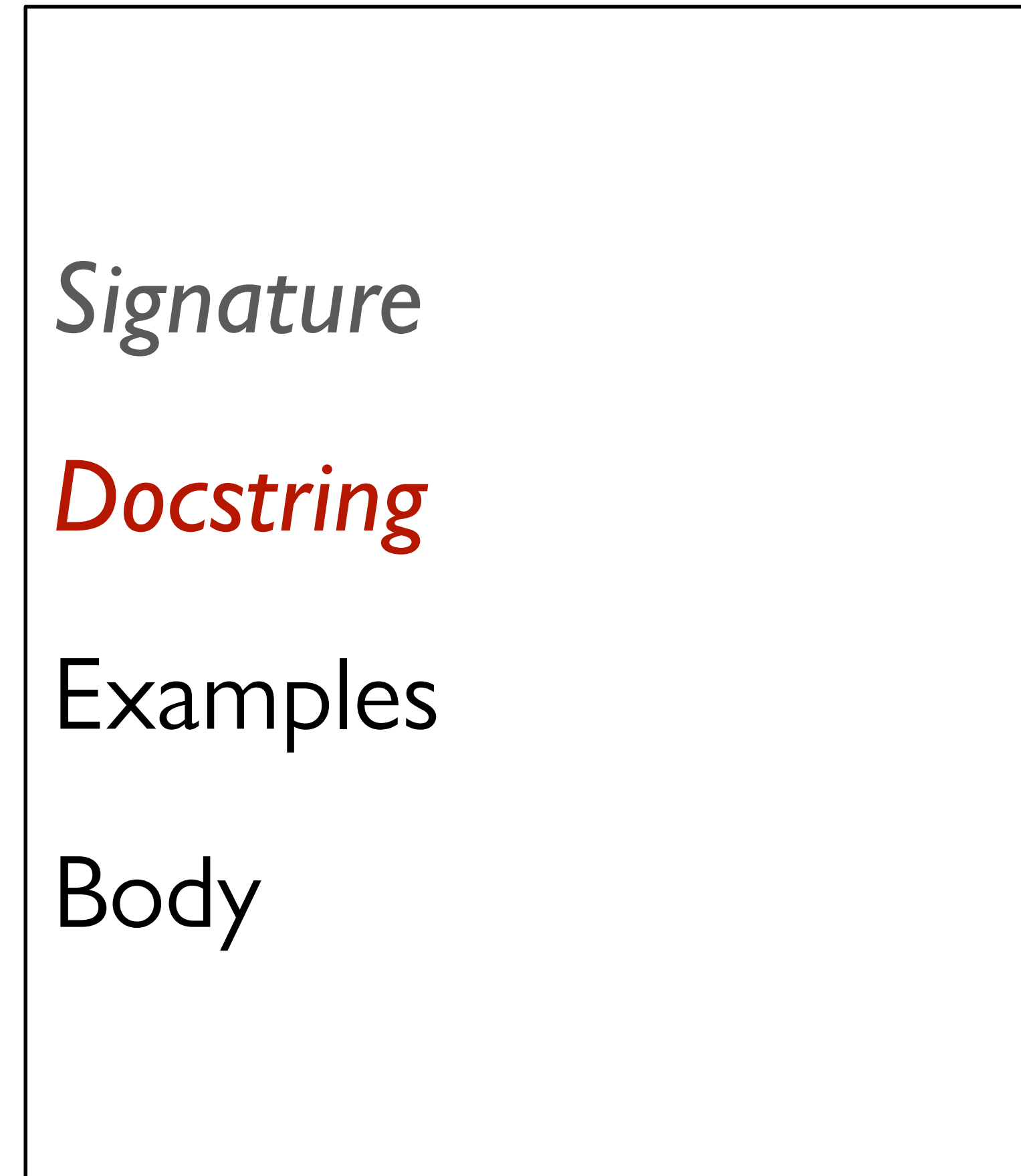
```
fun draw-light(tl :: TrafficLight) -> Image:  
  ...  
end
```

```
fun next-light(tl :: TrafficLight) -> TrafficLight:  
  ...  
end
```

Data



Functions



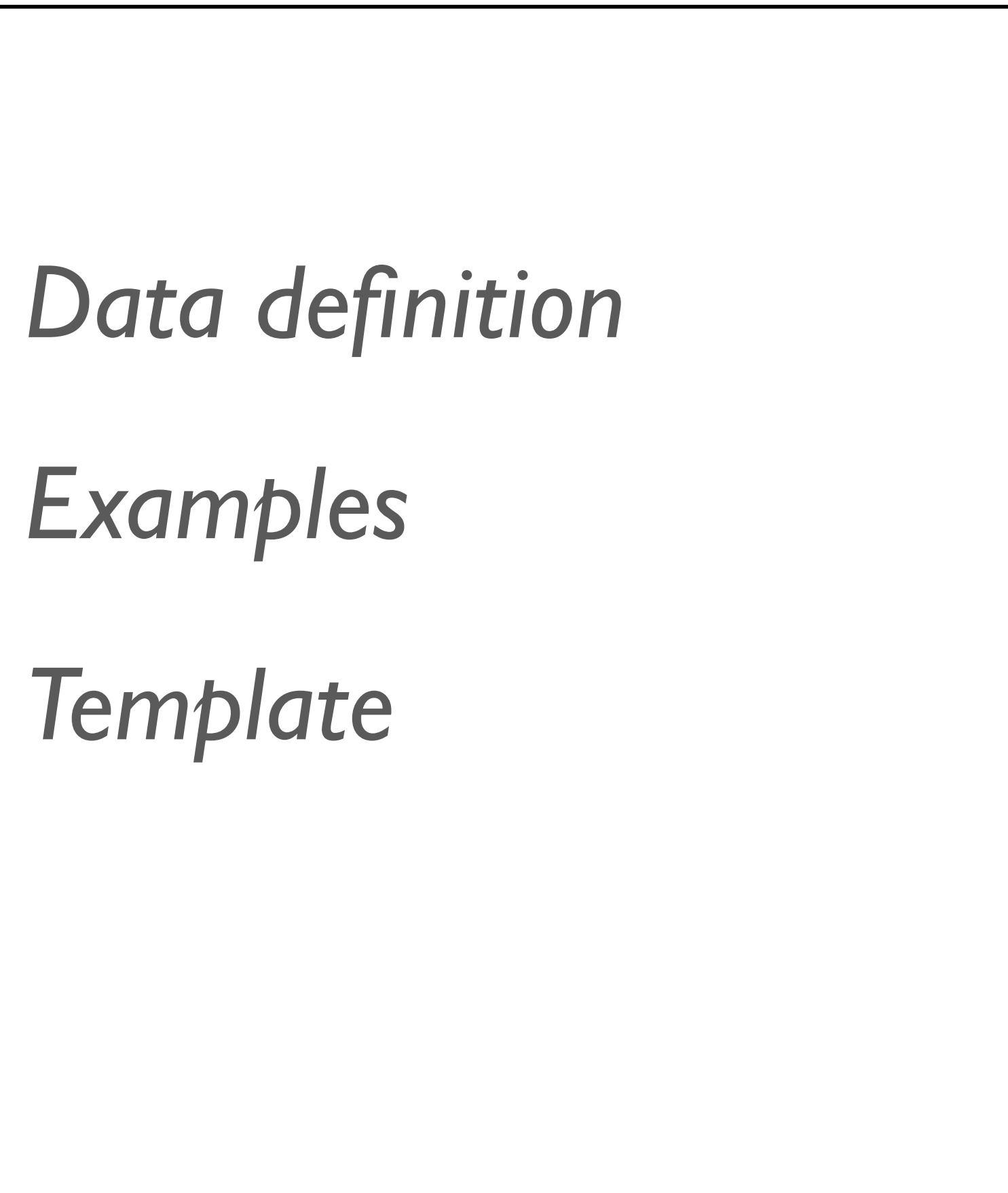
```
fun draw-light(tl :: TrafficLight) -> Image:  
  doc: "Draw a circle of the given color, rendering a traffic light"  
  ...  
end
```

```
fun next-light(tl :: TrafficLight) -> TrafficLight:  
  ...  
end
```

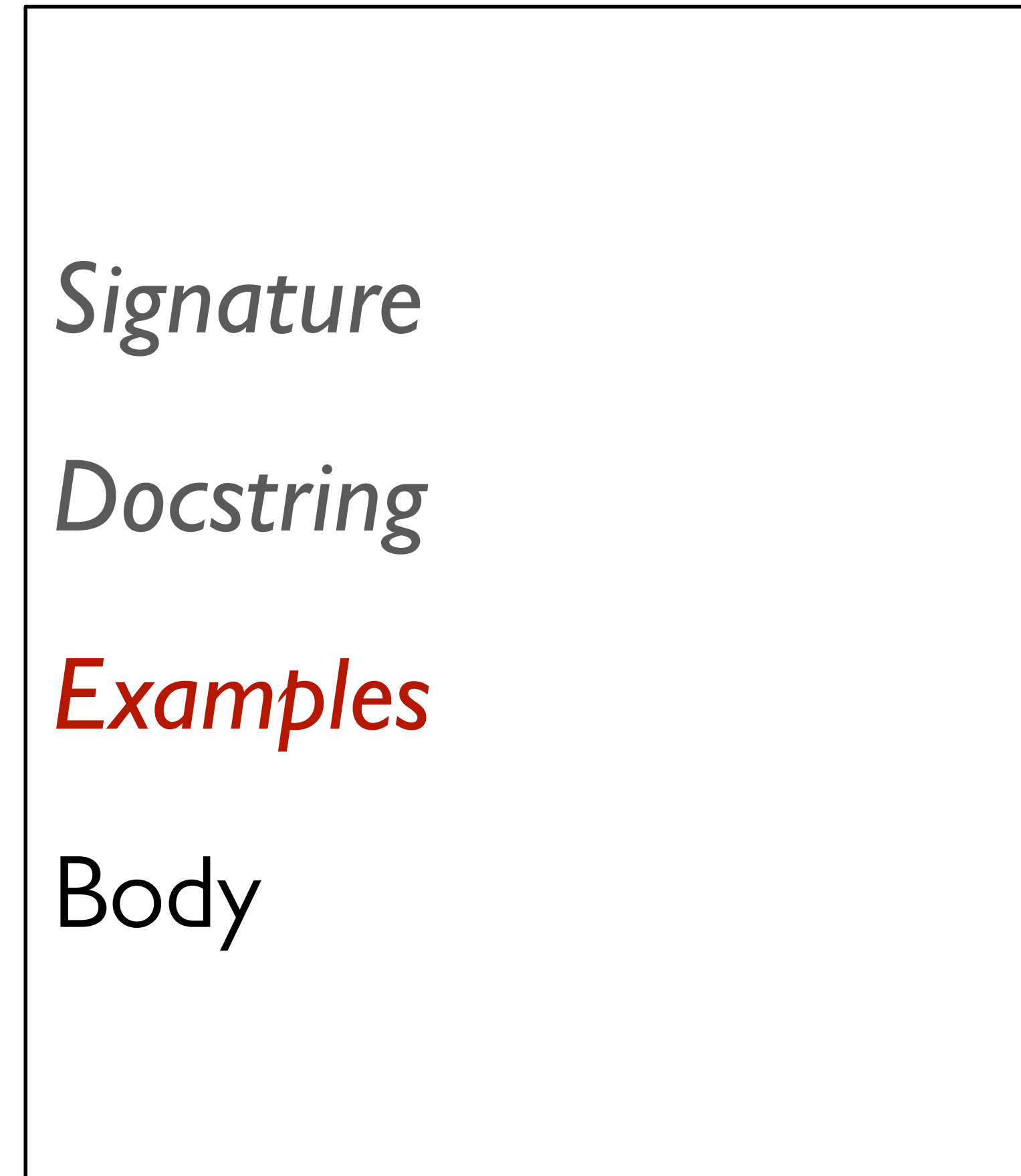
```
fun draw-light(tl :: TrafficLight) -> Image:  
  doc: "Draw a circle of the given color, rendering a traffic light"  
  ...  
end
```

```
fun next-light(tl :: TrafficLight) -> TrafficLight:  
  doc: "Produce the next light in the sequence green, yellow, red"  
  ...  
end
```

Data



Functions



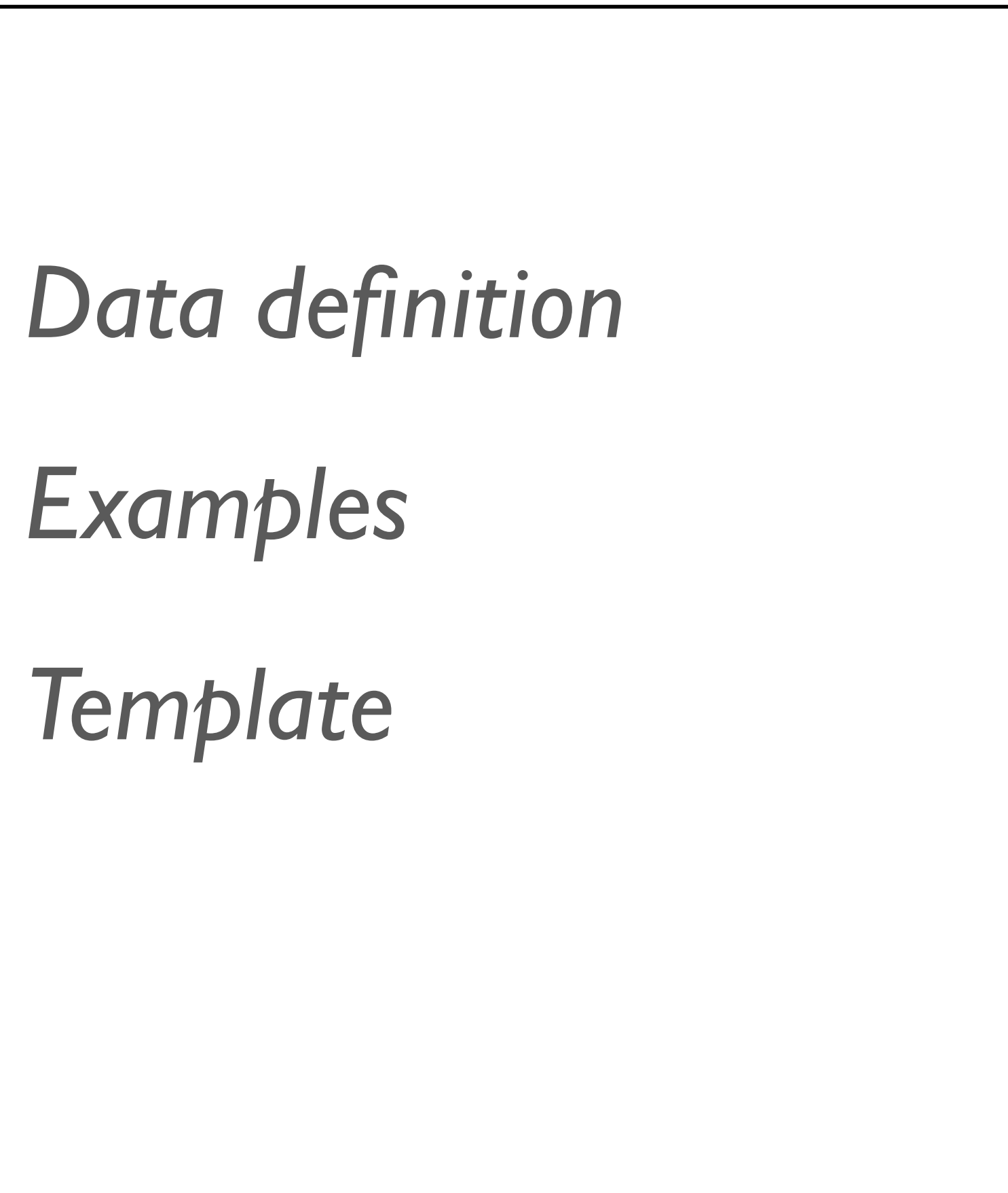
```
fun draw-light(tl :: TrafficLight) -> Image:  
  doc: "Draw a circle of the given color, rendering a traffic light"  
  ...  
where:  
  draw-light(green) is circle(20, "solid", "green")  
  draw-light(yellow) is circle(20, "solid", "yellow")  
  draw-light(red) is circle(20, "solid", "red")  
end
```

```
fun next-light(tl :: TrafficLight) -> TrafficLight:  
  doc: "Produce the next light in the sequence green, yellow, red"  
  ...  
end
```

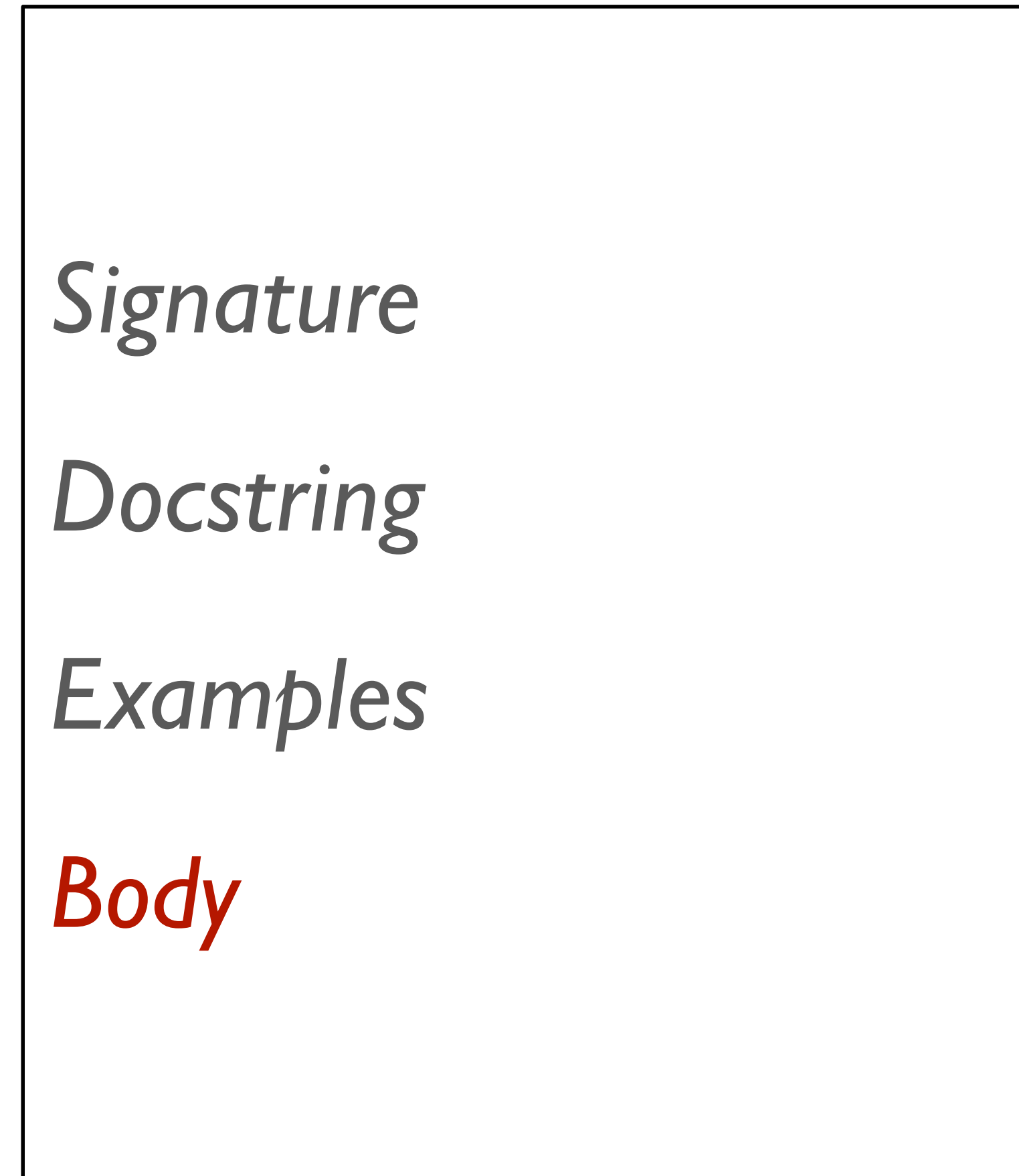
```
fun draw-light(tl :: TrafficLight) -> Image:
  doc: "Draw a circle of the given color, rendering a traffic light"
  ...
where:
  draw-light(green) is circle(20, "solid", "green")
  draw-light(yellow) is circle(20, "solid", "yellow")
  draw-light(red) is circle(20, "solid", "red")
end
```

```
fun next-light(tl :: TrafficLight) -> TrafficLight:
  doc: "Produce the next light in the sequence green, yellow, red"
  ...
where:
  next-light(green) is yellow
  next-light(yellow) is red
  next-light(red) is green
end
```

Data



Functions

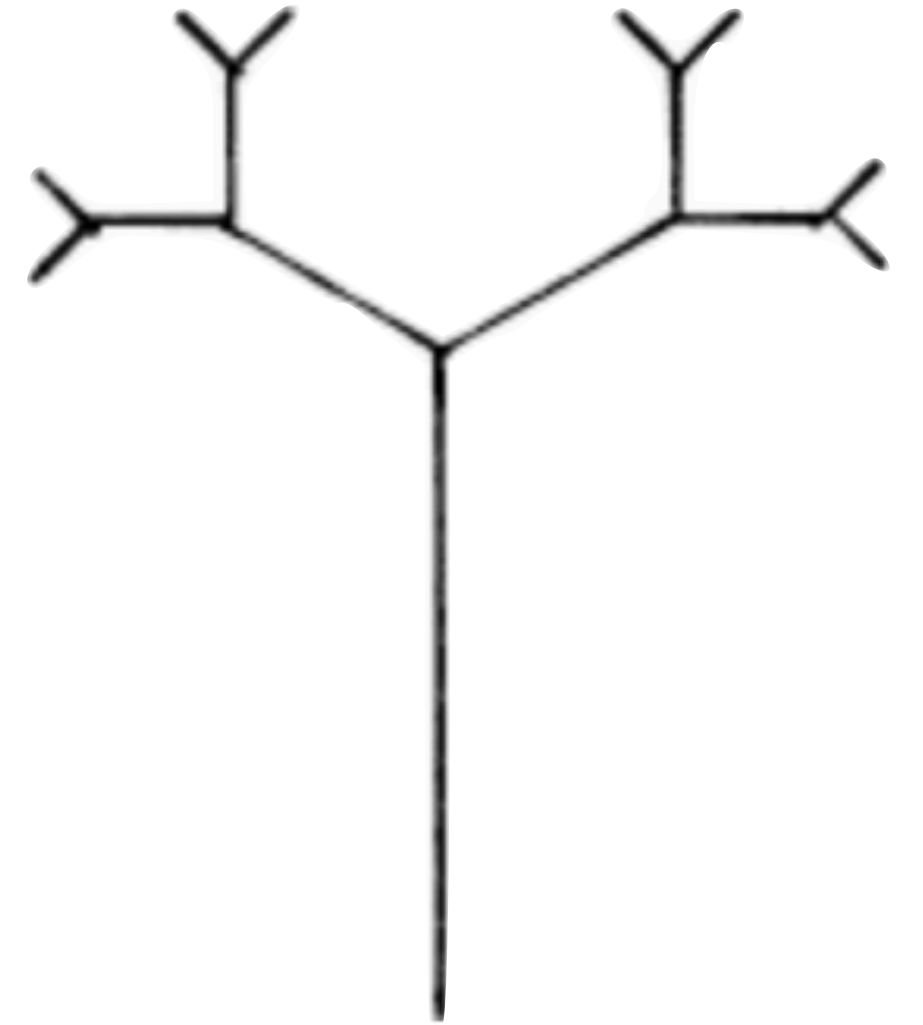


Fractal tree



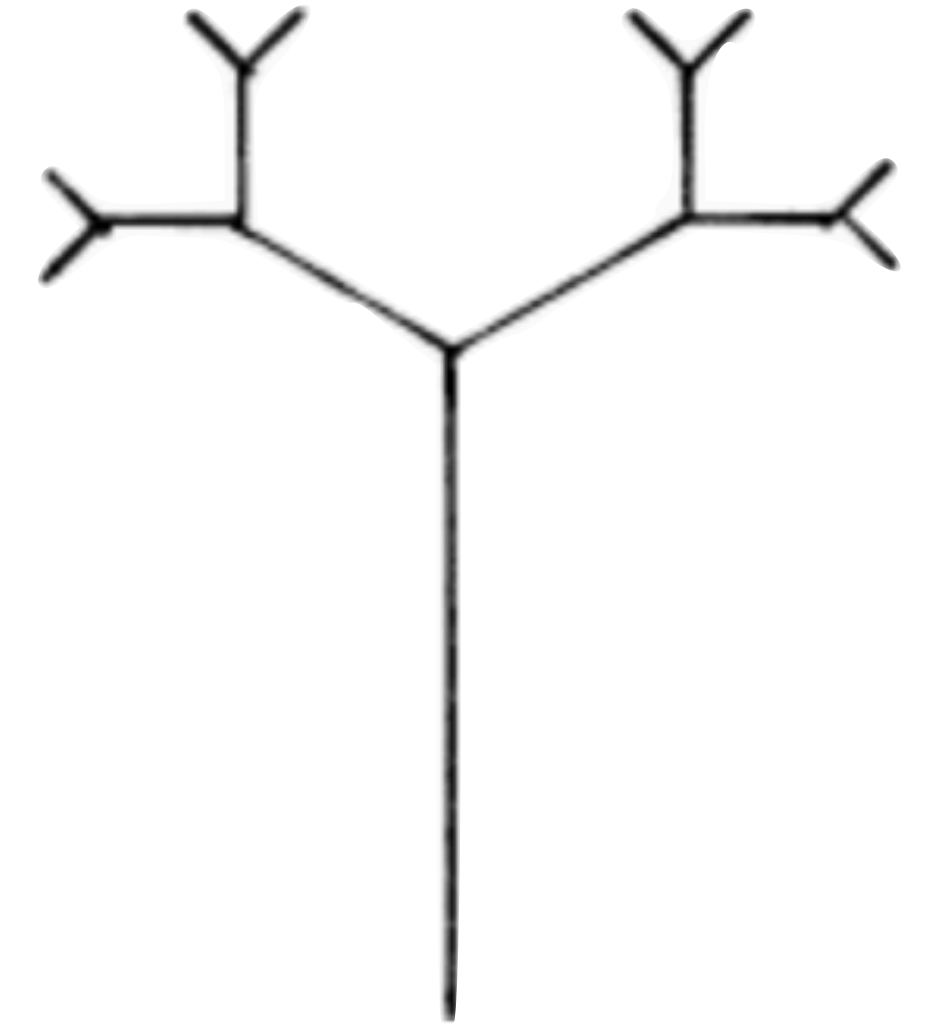
How can we draw a tree?

A big tree is a stick with two smaller trees on top;
but a little tree is just a stick.



A big tree is a stick with two smaller trees on top;
but a little tree is just a stick.

```
data Tree:  
  | stick  
  | branch(t1 :: Tree, t2 :: Tree)  
end
```



To finish this data definition, we should add examples and a template function.

Examples

```
lil-tree = branch(stick, stick)
```

```
big-tree =
```

```
  branch(  
    branch(stick, stick),  
    branch(stick, stick))
```

Template

```
#|
```

```
fun tree-fun(tree :: Tree) -> ...:
```

```
  doc: "Tree template"
```

```
  cases (Tree) tree:
```

```
    | stick => ...
```

```
    | branch(t1, t2) => ... tree-fun(t1) ... tree-fun(t2) ...
```

```
  end
```

```
where:
```

```
  tree-fun(stick) is ...
```

```
  tree-fun(lil-tree) is ...
```

```
  tree-fun(big-tree) is ...
```

```
end
```

```
|#
```

How can we draw a Tree?


```
fun draw-tree-size(tree :: Tree, size :: Number) -> Image:
  doc: "Draw a tree based on a line of the specified size"
  stick-tree = line(1, size, "black")
  cases (Tree) tree:
    | stick =>
      # A small tree is just a stick
      stick-tree
    | branch(t1, t2) =>
      # A branch is
      above(
        # Two smaller trees
        beside(
          rotate(45,
            draw-tree-size(t1, size / 2)),
          rotate(-45,
            draw-tree-size(t2, size / 2))),
        # Above a stick
        stick-tree)
  end
end
```

```
TREE-SIZE = 400
```

```
fun draw-tree(tree :: Tree) -> Image:  
  doc: "Draw a tree (and its subtrees)"  
  draw-tree-size(tree, TREE-SIZE)  
end
```

Now, let's use a reactor to animate the recursion of the fractal, starting from the simplest tree and working toward a full, leafy one.

One more reactor



