

---

# Como Projetar Mundos

(projetando programas interativos)

Prof. Helio H. L. C. Monte-Alto

---

”Cristo padeceu uma vez pelos pecados, o justo pelos injustos, para levar-nos a Deus”

”Porque Deus amou o mundo de tal maneira que deu o seu Filho unigênito, para que todo aquele que nele crê não pereça, mas tenha a vida eterna.”

”Crê no Senhor Jesus Cristo e serás salvo”

(1 Pedro 3:18, João 3:16, Atos 16:31)

---

# O que vimos até agora...

---

- O que é programação?
  - Introdução à linguagem Racket
    - Tipos de dados primitivos (Number, String, Image, etc)
    - Expressões (if, cond)
    - Avaliação de expressões (stepper)
    - Definição de funções e constantes
  - “Como projetar funções” (How to Design Functions)
    - Test Driven Development
  - “Como projetar dados” (How to Design Data)
    - Data Driven Templates
-

# O que vamos aprender agora

---

- Projetar programas interativos
    - Funções + Dados + Interação = Mundos
    - Exemplos:
      - Animações
      - Jogos
      - Editor de texto
      -
-

# O que vamos aprender agora

---

- Projetar programas interativos
    - Funções + Dados + Interação = Mundos
    - Exemplos:
      - Animações
      - Jogos
      - Editor de texto
    - Na verdade qualquer coisa
-

# Exemplo do gato

---

... exemplo em código ...

---

# Exemplo do gato

---

O que temos aí?

---

# Exemplo do gato

---








O que temos aí?

- Estados que mudam
  - Tela que muda
  - Teclado e/ou mouse que afetam o comportamento
-

# Exemplo do gato

---

- Nos bastidores:








tick #	x	image
0	0	
1	3	
2	6	
3	9	
...		
312	936	
313	939	
314	942	

28 ticks per second



# Exemplo do gato

- Nos bastidores:








tick #	x	image
0	0	
1	3	
2	6	
3	9	
...		
312	936	
313	939	
314	942	

28 ticks per second

Informação de domínio

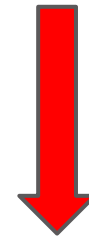
# Exemplo do gato

- Nos bastidores:

tick #	x	image
0	0	
1	3	
2	6	
3	9	
...		
312	936	
313	939	
314	942	

28 ticks per second

Informação de domínio










Definição de dados

# Exemplo do gato

- Nos bastidores:

```
;; =====  
;; Data definitions:  
  
;; Cat is Number  
;; interp. x coordinate of cat  
;;  
(define C1 0)  
(define C2 (/ WIDTH 2))  
#;  
(define (fn-for-cat c)  
  (... c))  
;; Template rules used:  
;; - atomic non-distinct: Number  
  
;; =====  
;; Functions:  
  
;; Cat -> Cat  
;; increase cat x position by SPEED  
(check-expect (next-cat 0) SPEED)  
(check-expect (next-cat 100) (+ 100 SPEED))  
#;  
(define (next-cat c) 1) ; stub  
  
;; <use template from Cat>  
  
(define (next-cat c)  
  (+ c SPEED))  
  
;; Cat -> Image  
;; add CAT-IMG to MTS at proper x coordinate and CTR-Y  
(check-expect (render-cat 100)  
  (place-image CAT-IMG 100 CTR-Y MTS))  
#;  
(define (render-cat c) MTS) ; stub  
  
;; <use template from Cat>  
  
(define (render-cat c)  
  (place-image CAT-IMG c CTR-Y MTS))
```

Definição dos dados. Números como 0, 3 e 6 representam a posição x do gato

tick #	x	image
0	0	
1	3	
2	6	
3	9	
...		
312	936	
313	939	
314	942	








28 ticks per second

# Exemplo do gato

- Nos bastidores:

```
;; =====  
;; Data definitions:  
  
;; Cat is Number  
;; interp. x coordinate of cat  
;;  
(define C1 0)  
(define C2 (/ WIDTH 2))  
#;  
(define (fn-for-cat c)  
  (... c))  
;; Template rules used:  
;; - atomic non-distinct: Number  
  
;; =====  
;; Functions:  
  
;; Cat -> Cat  
;; increase cat x position by SPEED  
(check-expect (next-cat 0) SPEED)  
(check-expect (next-cat 100) (+ 100 SPEED))  
#;  
(define (next-cat c) 1) ; stub  
  
;; <use template from Cat>  
  
(define (next-cat c)  
  (+ c SPEED))  
  
;; Cat -> Image  
;; add CAT-IMG to MTS at proper x coordinate and CTR-Y  
(check-expect (render-cat 100)  
  (place-image CAT-IMG 100 CTR-Y MTS))  
#;  
(define (render-cat c) MTS) ; stub  
  
;; <use template from Cat>  
  
(define (render-cat c)  
  (place-image CAT-IMG c CTR-Y MTS))
```

Uma função que, dada a atual posição do gato, produz a próxima posição

tick #	x	image
0	0	
1	3	
2	6	
3	9	
...		
312	936	
313	939	
314	942	








28 ticks per second

# Exemplo do gato

- Nos bastidores:

```
;; =====  
;; Data definitions:  
  
;; Cat is Number  
;; interp. x coordinate of cat  
;;  
(define C1 0)  
(define C2 (/ WIDTH 2))  
#;  
(define (fn-for-cat c)  
  (... c))  
;; Template rules used:  
;; - atomic non-distinct: Number  
  
;; =====  
;; Functions:  
  
;; Cat -> Cat  
;; increase cat x position by SPEED  
(check-expect (next-cat 0) SPEED)  
(check-expect (next-cat 100) (+ 100 SPEED))  
#;  
(define (next-cat c) 1) ; stub  
  
;; <use template from Cat>  
  
(define (next-cat c)  
  (+ c SPEED))  
  
;; Cat -> Image  
;; add CAT-IMG to MTS at proper x coordinate and CTR-Y  
(check-expect (render-cat 100)  
  (place-image CAT-IMG 100 CTR-Y MTS))  
#;  
(define (render-cat c) MTS) ; stub  
  
;; <use template from Cat>  
  
(define (render-cat c)  
  (place-image CAT-IMG c CTR-Y MTS))
```

Uma função que põe a imagem do gato no lugar certo em uma cena vazia

tick #	x	image
0	0	
1	3	
2	6	
3	9	
...		
312	936	
313	939	
314	942	

28 ticks per second

# Exemplo do gato

---

- Nos bastidores:
    - Logo temos:
      - Definição dos dados
      - Uma função que muda o estado
      - Uma função que representa o estado na tela
    - O que falta?
-

# Exemplo do gato

---

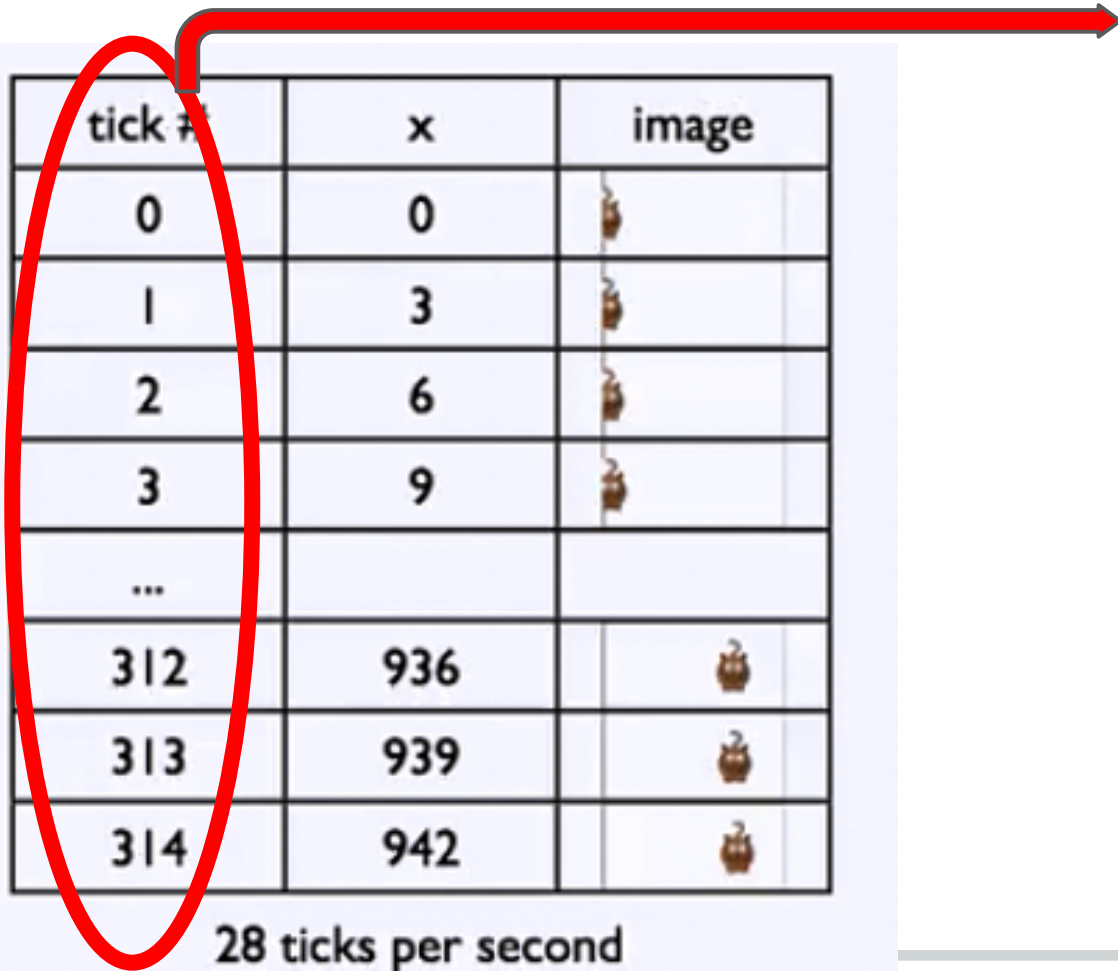
- Nos bastidores:
    - Logo temos:
      - Definição dos dados
      - Uma função que muda o estado
      - Uma função que representa o estado na tela
    - O que falta?
      - Como combiná-los em um programa
      - Como implementar as interações
-








# Exemplo do gato

---

- Nos bastidores:

Interação do clock



tick #	x	image
0	0	
1	3	
2	6	
3	9	
...		
312	936	
313	939	
314	942	

28 ticks per second



# Exemplo do gato

---

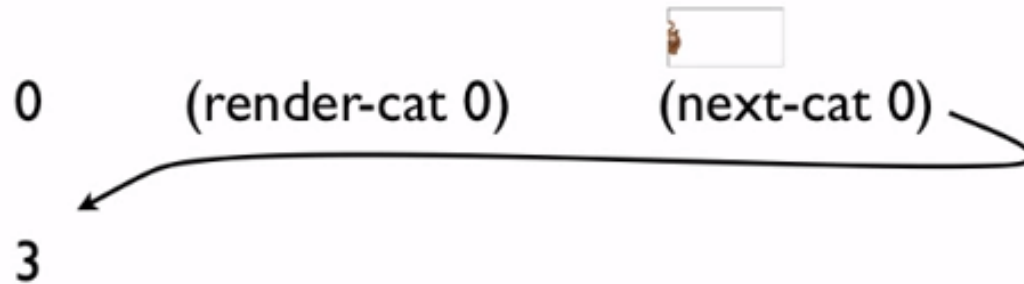


0

(render-cat 0)

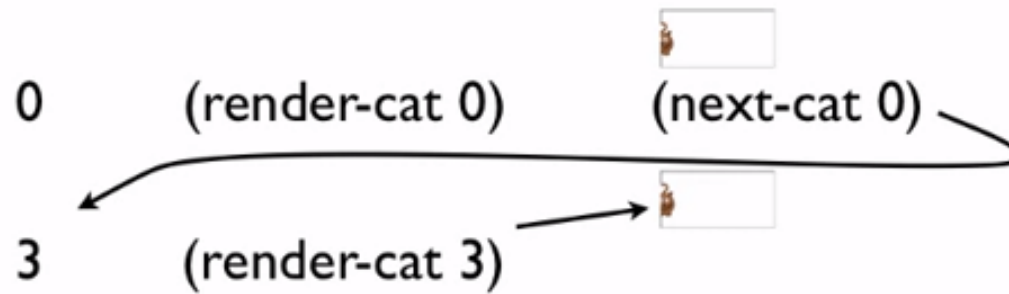


# Exemplo do gato



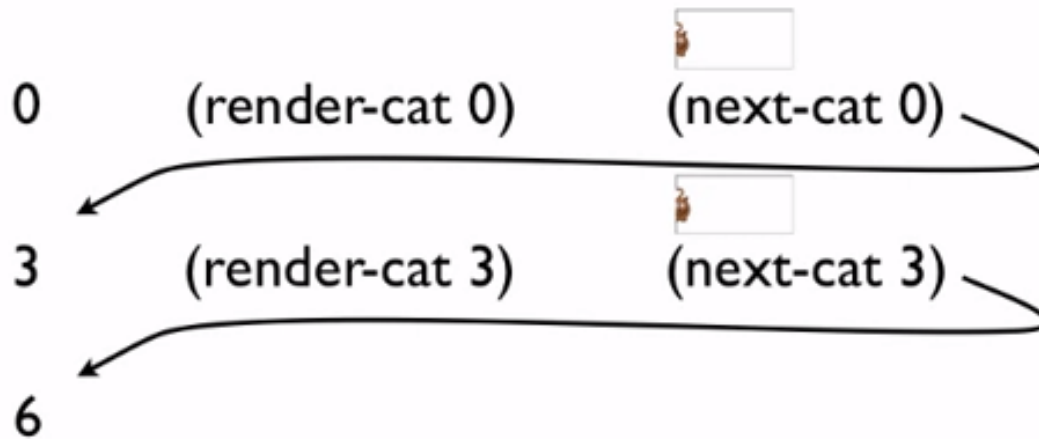
# Exemplo do gato

---



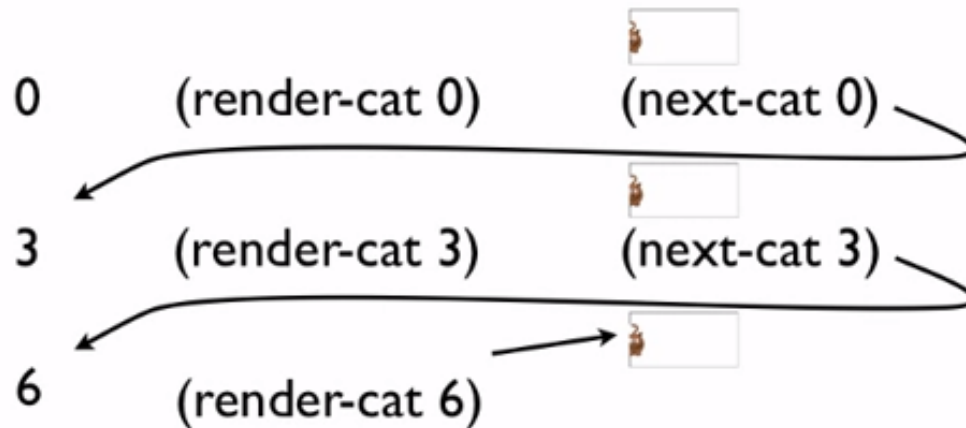
# Exemplo do gato

---

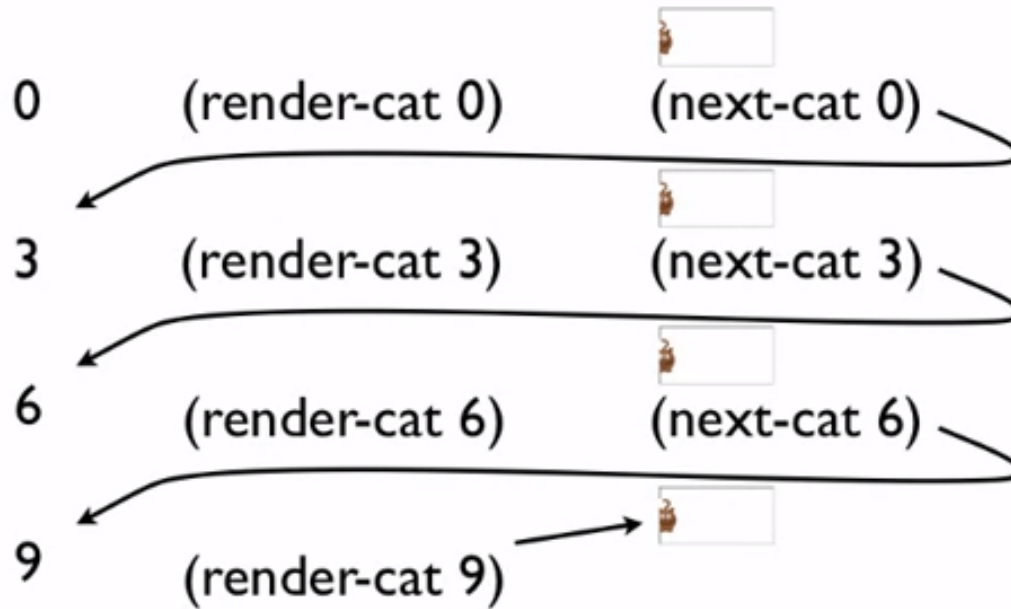


# Exemplo do gato

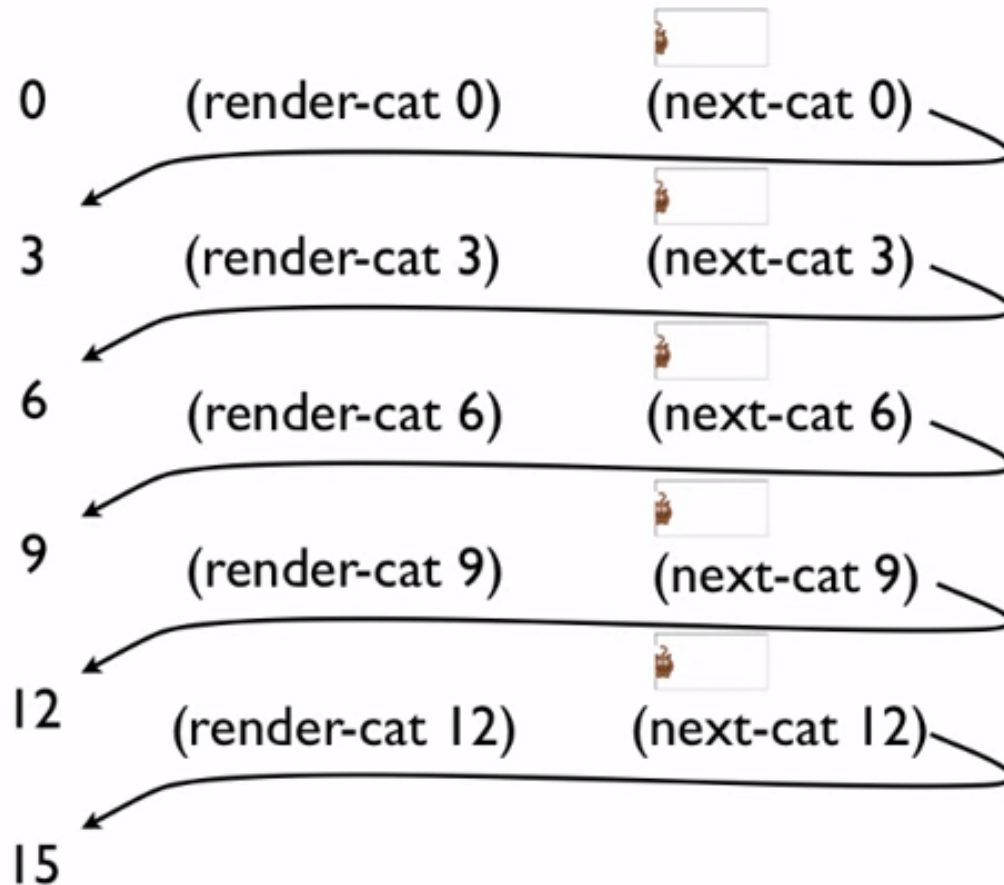
---



# Exemplo do gato

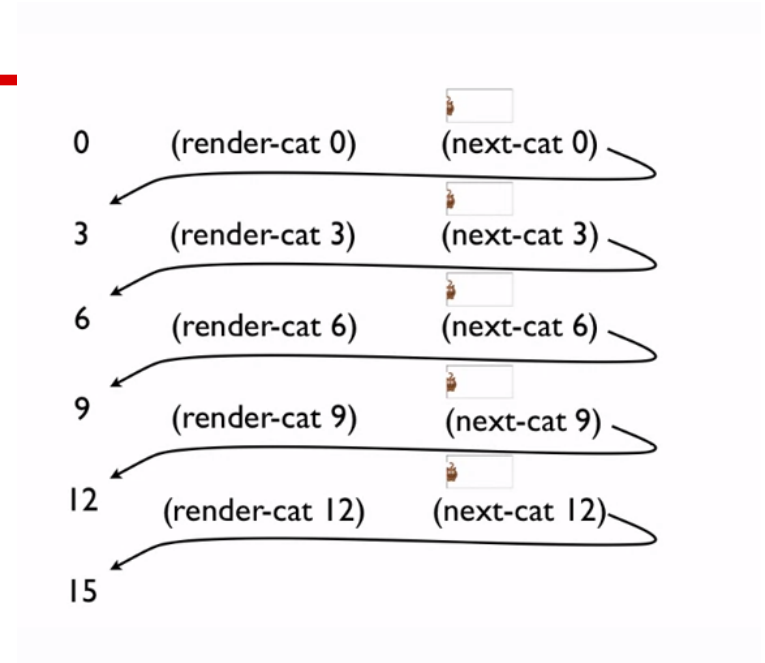


# Exemplo do gato



# Exemplo do gato

- Como??



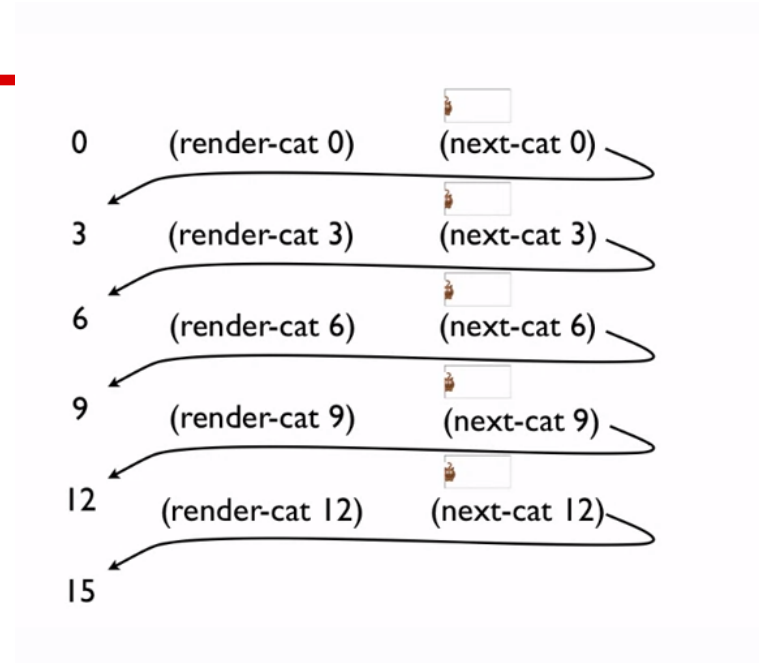
```
(big-bang 0  
  (on-tick next-cat)  
  (to-draw render-cat))
```

```
; Cat  
; Cat -> Cat  
; Cat -> Image
```



# Exemplo do gato

- Como??



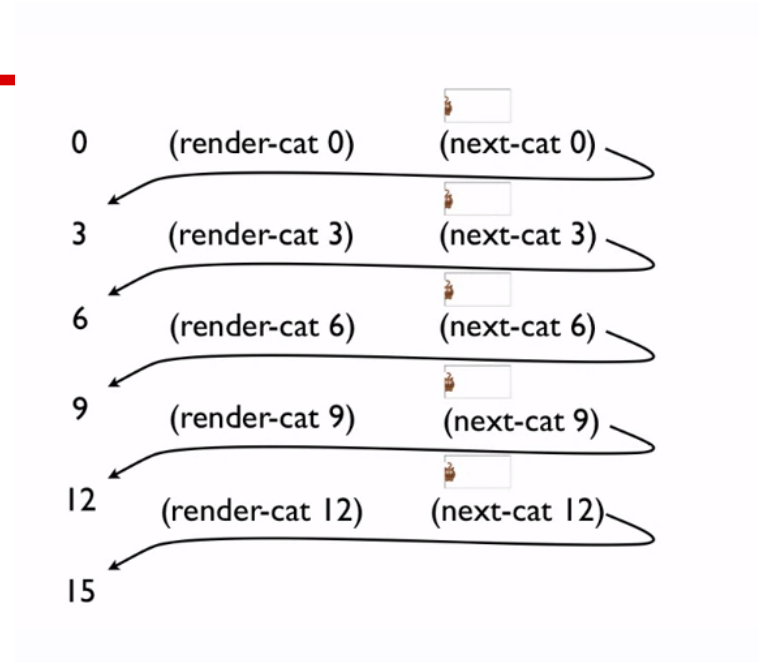
Estado inicial do  
mundo (que é Cat)

```
(big-bang 0  
  (on-tick next-cat)  
  (to-draw render-cat))
```

```
; Cat  
; Cat -> Cat  
; Cat -> Image
```

# Exemplo do gato

- Como??



Estado inicial do mundo (que é Cat)

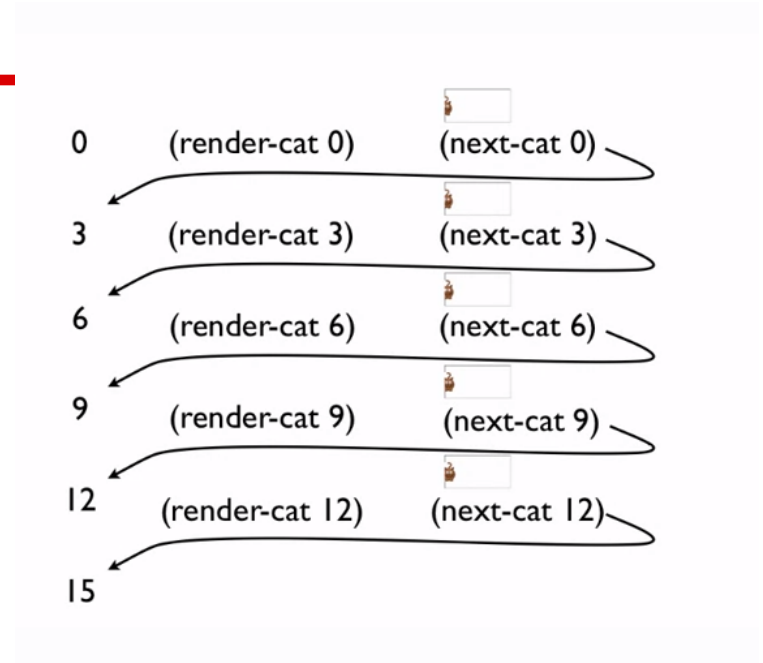
```
(big-bang 0  
  (on-tick next-cat)  
  (to-draw render-cat))
```

```
; Cat  
; Cat -> Cat  
; Cat -> Image
```

A cada tick do clock, chama next-cat com o estado atual do mundo para pegar o próximo estado

# Exemplo do gato

- Como??



Estado inicial do mundo (que é Cat)

```
(big-bang 0  
  (on-tick next-cat)  
  (to-draw render-cat))
```

```
; Cat  
; Cat -> Cat  
; Cat -> Image
```

A cada tick do clock, chama *next-cat* com o estado atual do mundo para pegar o próximo estado

A cada tick do clock, chama *render-cat* com o estado atual do mundo para desenhar o estado atual do mundo



# Receita de “Como Criar Mundos”

---

1. Analise de domínio (no papel!)
    1. Desenhe cenários
    2. Identifique informações constantes
    3. Identifique informações que mudam
    4. Identifique opções de big-bang
  2. Construa o programa
    1. Constantes (baseadas no passo 1.2)
    2. Definições de dados (baseadas no passo 1.3)
    3. Funções
      1. Função *main* (baseada nos passos 1.3, 1.4 e 2.2)
      2. Itens da lista de desejos baseado nos chamados do big-bang
    4. Trabalhar na lista de desejos ate terminar
-

# Receita de “Como Criar Mundos”

---

1. **Analise de domínio (no papel!)**
    1. **Desenhe cenários**
    2. **Identifique informações constantes**
    3. **Identifique informações que mudam**
    4. **Identifique opções de big-bang**
  2. **Construa o programa**
    1. **Constantes (baseadas no passo 1.2)**
    2. **Definições de dados (baseadas no passo 1.3)**
    3. **Funções**
      1. **Função *main* (baseada nos passos 1.3, 1.4 e 2.2)**
      2. **Itens da lista de desejos baseado nos chamados do big-bang**
    4. **Trabalhar na lista de desejos ate terminar**
-

# Exemplo do gato

---

Ver `cat-starter.rkt`

Me desculpem se ainda está entediante... mas faz parte do aprendizado.

Em breve veremos exemplos mais interessantes!!

Consulte o material: [How to Design Worlds](#)

---

# 1. Análise de Domínio

---

Vamos lá... papel e caneta!!!

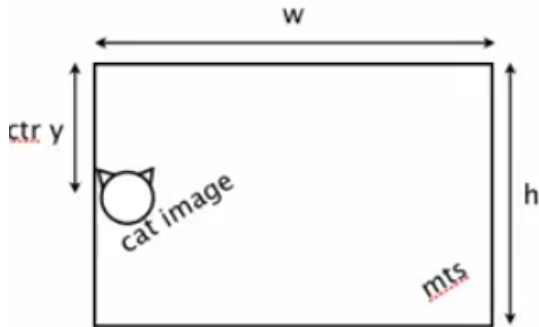
---



# 1. Análise de Domínio

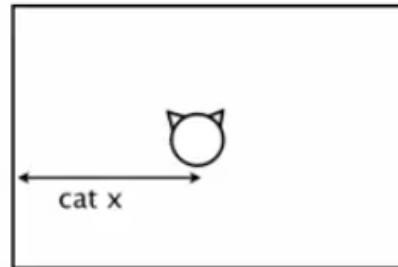
---

Vamos lá... papel e caneta!!!



## Constant

width  
height  
ctr-y  
mts  
cat image



## Changing

x coordinate of cat



## big-bang options

on-tick  
to-draw

## 2. Construa o programa

---

- Pegar template em [How To Design Worlds](#)
    1. Constantes (baseadas no passo 1.2)
    2. Definições de dados (baseadas no passo 1.3)
    3. Funções
      1. Função *main* (baseada nos passos 1.3, 1.4 e 2.2)
      2. Itens da lista de desejos baseado nos chamados do big-bang
    4. Trabalhar na lista de desejos ate terminar
-

## 2. Construa o programa

---

- Pegar template em [How To Design Worlds](#)
    1. **Constantes (baseadas no passo 1.2)**
    2. Definições de dados (baseadas no passo 1.3)
    3. Funções
      1. Função *main* (baseada nos passos 1.3, 1.4 e 2.2)
      2. Itens da lista de desejos baseado nos chamados do big-bang
    4. Trabalhar na lista de desejos ate terminar
-

## 2. Construa o programa

---

- Pegar template em [How To Design Worlds](#)
    1. Constantes (baseadas no passo 1.2)
    - 2. Definições de dados (baseadas no passo 1.3)**
    3. Funções
      1. Função *main* (baseada nos passos 1.3, 1.4 e 2.2)
      2. Itens da lista de desejos baseado nos chamados do big-bang
    4. Trabalhar na lista de desejos ate terminar
-

## 2. Construa o programa

---

- Pegar template em [How To Design Worlds](#)
    1. Constantes (baseadas no passo 1.2)
    2. Definições de dados (baseadas no passo 1.3)
    3. **Funções**
      1. **Função *main*** (baseada nos passos 1.3, 1.4 e 2.2)
      2. **Itens da lista de desejos baseado nos chamados do big-bang**
    4. Trabalhar na lista de desejos ate terminar
-

## 2. Construa o programa

---

- Pegar template em [How To Design Worlds](#)
    1. Constantes (baseadas no passo 1.2)
    2. Definições de dados (baseadas no passo 1.3)
    3. Funções
      1. Função *main* (baseada nos passos 1.3, 1.4 e 2.2)
      2. Itens da lista de desejos baseado nos chamados do big-bang
    4. **Trabalhar na lista de desejos ate terminar**
-

# Duvidas??

---