

Módulo 03 - Ceu

Tópicos em Linguagens

<https://github.com/fsantanna-uerj/LPX/>

Francisco Sant'Anna

francisco@ime.uerj.br



1. Hello World!

Tópicos em Linguagens

<https://github.com/fsantanna-uerj/LPX/>

Francisco Sant'Anna

francisco@ime.uerj.br

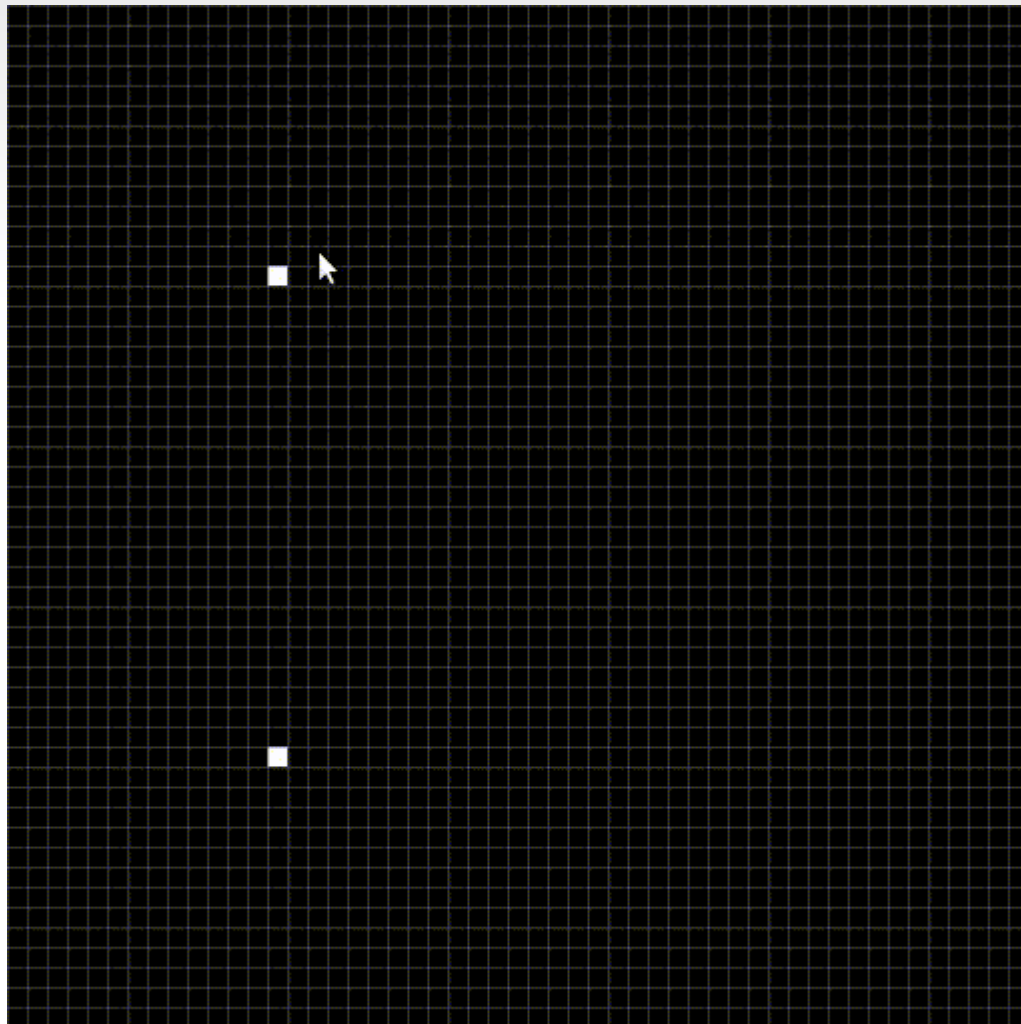


Instalação

- Ceu + pico-ceu
 - <https://github.com/fsantanna/dceu>

Hello World!

```
$ ceu --lib=pico 01-hello.ceu
```



Hello World!

```
data :XY = [x,y]

task Line (pos:XY, vel:XY) {
  par {
    every :Pico.Draw {
      pico-output-draw-pixel(pos)
    }
  } with {
    every 50:ms {
      set pos.x = pos.x + vel.x
      set pos.y = pos.y + vel.y
    }
  }
}

spawn {
  spawn Line ([-25,25],[1,-1])
  spawn Line ([-25,-25],[1,1])
  await 5:s
  set pico-quit = true
}
```

03-ceu/01-hello.ceu

Alterações

Exercício 3.1

- Altere o exemplo 01-hello.ceu...
 - Adicione mais dois pixels em direções opostas.
 - Adicione um novo parâmetro de cor às tarefas e modifique suas cores.
 - Crie uma nova task que desenhe uma figura de vários pixels que se movimenta por 1s como um todo em cada direção (dir,baixo,esq,cima).

2. Design

Tópicos em Linguagens

<https://github.com/fsantanna-uerj/LPX/>

Francisco Sant'Anna

francisco@ime.uerj.br



Design

- Structured Concurrency + Event-Driven Programming
 - Structured Deterministic Concurrency
 - Event Signaling Mechanisms
 - Lexical Memory Management

Design

- Dynamic typing
- Statements as expressions
- Dynamic collections (tuples, vectors, and dicts)
- Stackless coroutines
- Restricted closures
- Deferred statements (for finalization)
- Exception handling (throw & catch)
- Hierarchical tuple templates (for data description)
- Seamless integration with C

Manual

- <https://github.com/fsantanna/dceu/blob/main/doc/manual-out.md>

Exemplos

- Birds
- Pingus
- Rocks

3. Iteradores e Geradores

Tópicos em Linguagens

<https://github.com/fsantanna-uerj/LPX/>

Francisco Sant'Anna

francisco@ime.uerj.br



Iteradores

```
loop in iter([10,20,30]), v {  
    println(v)  
}
```

```
val it = iter([10,20,30])  
println(it)  
println(it[0](it))  
println(it[0](it))  
println(it[0](it))  
println(it[0](it))
```

03-ceu/02-iter.ceu

- Iteração / Iteration
 - `loop, it[0](it)`
- Iterável / Iterable
 - `[10,20,30]`
- Iterador / Iterator
 - `[f, ...]`
- Função `iter` transforma um *tupla* em um *iterador*

Python vs Lua vs Ceu

- Iteradores em Python são *stateful*, pois dependem de um objeto para manter o estado interno da iteração.
- Iteradores em Lua são *stateless*, pois não mantêm nenhum estado interno para controle da iteração.
- Iteradores em Lua podem ser *stateful*, por exemplo usando *closures*.
- Iteradores em Ceu são *stateful*, pois dependem de uma tupla para manter o estado interno da iteração.

Exercício 3.3.1

- Refaça o exercício 1.3.1 em Ceu.
- Crie um iterador em Ceu que receba uma árvore e retorne todas as folhas, da esquerda para a direita.
 - **Variante 1:** transforme a árvore em um vetor

```
val a = [  
  :aaa,  
  [  
    :xxx,  
    :yyy,  
  ],  
  :bbb,  
]
```

```
      +  
     / | \  
  aaa + bbb  
     / \  
    xxx yyy
```

```
loop in Arvore(a), f {  
  println(f)  
}  
  
;; saida  
:aaa  
:xxx  
:yyy  
:bbb
```

01-iter/ex-1.3.1.py

Geradores

```
loop in iter([10,20,30]), v {  
    println(v)  
}
```

```
coro F () {  
    yield(1)  
    yield(2)  
    Yield(3)  
}  
  
val f = coroutine(F)  
loop in iter(f), v {  
    println(v)  
}
```

```
val it = iter(coroutine(F))  
println(it)  
println(it[0](it))  
println(it[0](it))  
println(it[0](it))  
println(it[0](it))
```

- Iteração
 - **loop, resume -> resume**
- Gerador → Iterador
 - **iter** empacota co-rotina
 - **resume** acorda gerador
 - **yield** “produz” um valor
- **f vs F vs resume F ()**

Exercício 3.3.2

- Refaça o exercício 2.3.2 em Ceu.
- Considere um personagem que se move em um mapa nas 4 direções: “dir”, “esq”, “cima”, “baixo”.
- Crie um gerador em Ceu em que o personagem faz um movimento contínuo “em quadrado”, com 10 passos seguidos em cada direção.