Estruturas de Linguagem

Francisco Sant'Anna

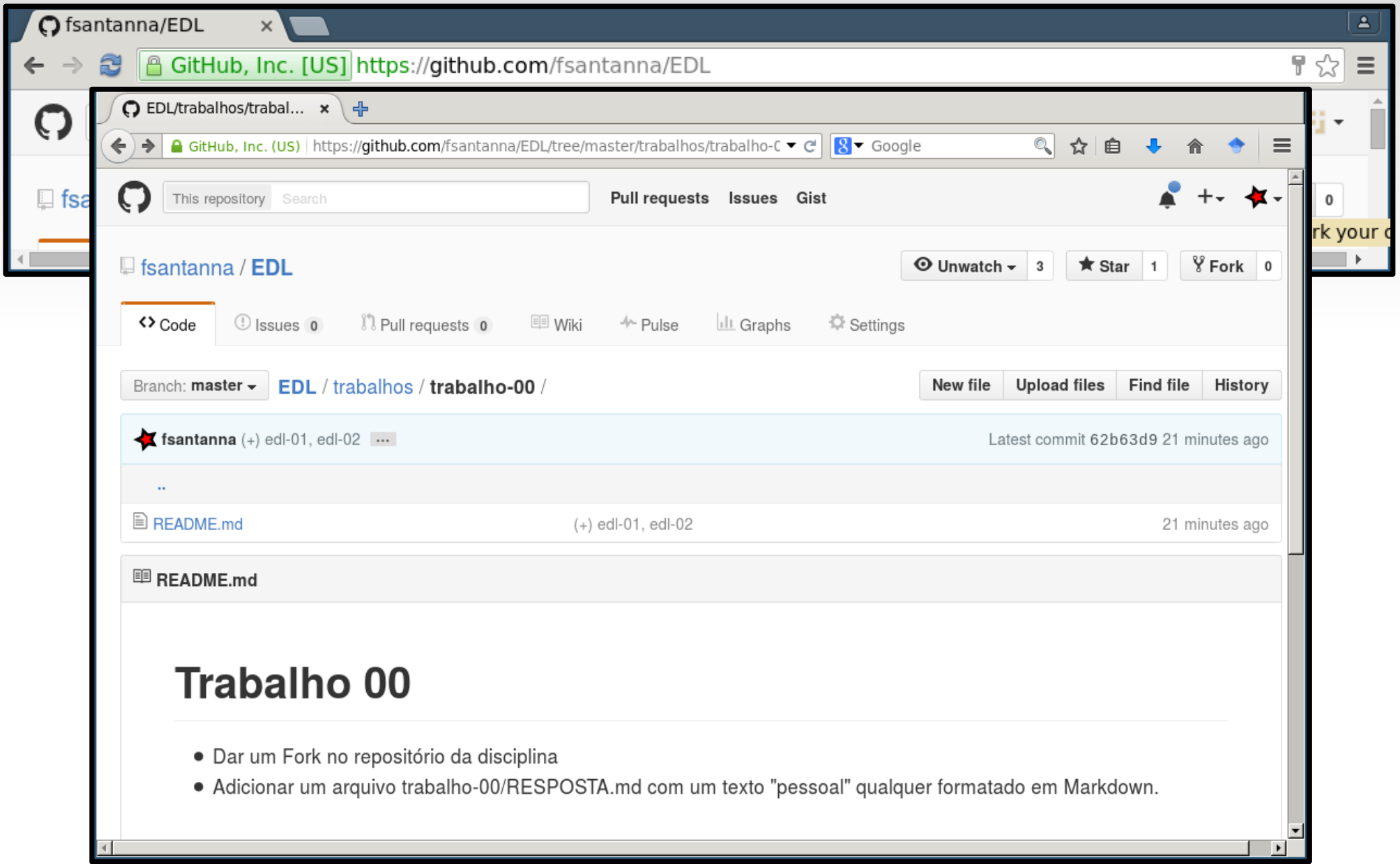
francisco.santanna@gmail.com

http://github.com/fsantanna/EDL

Trabalho 00 - GitHub
(até quarta-feira 09/03)

- Trabalho
  - Dar um *Fork* no repositório da disciplina
  - Adicionar um arquivo `trabalho-00/RESPOSTA.md` com um texto "pessoal" qualquer formatado em *Markdown*
- Links
  - `http://github.com/fsantanna/EDL`
  - `https://help.github.com/articles/good-resources-for-learning-git-and-github/`
  - `https://help.github.com/articles/basic-writing-and-formatting-syntax/`

Trabalho 00 - GitHub



Trabalho 01 - Artigo

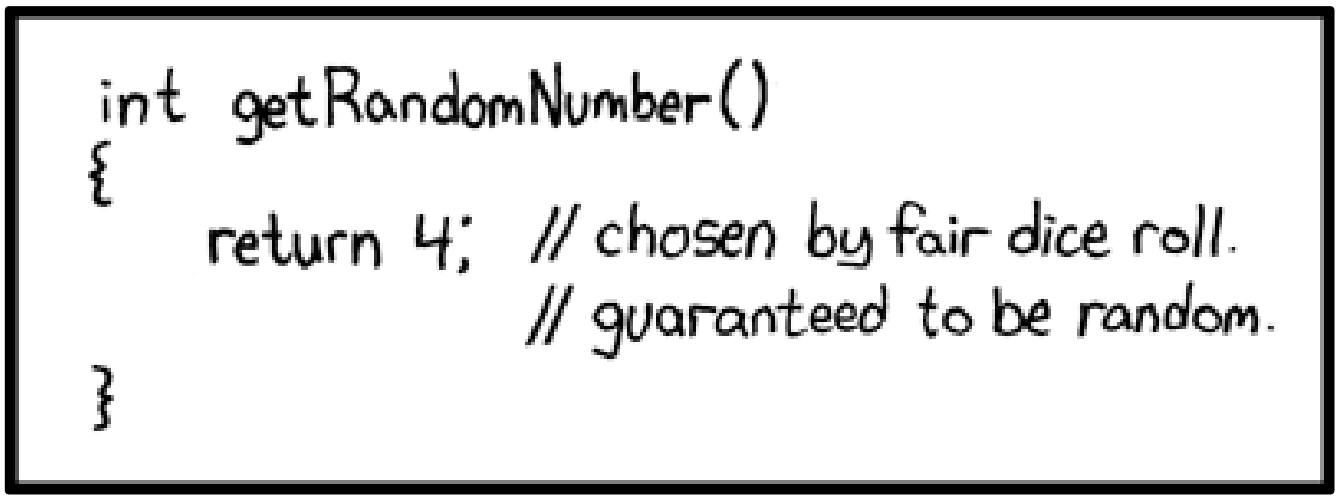
(até quarta-feira 16/03)

- Escolher uma linguagem com a qual você **não está familiarizado**.
  - evitar duplicatas com outros colegas
- Escrever um pequeno artigo (estilo *Wikipedia*):
  - origens e influências (linha do tempo)
  - classificação (imp/func/log/oo, est/din, usos)
  - avaliação em comparação com linguagens que você conhece (read/write, expressividade)
  - exemplos de código representativos
  - `trabalho-01/ARTIGO.md`
- Slides de apresentação (4-6 slides)
  - `trabalho-01/slides.pdf`

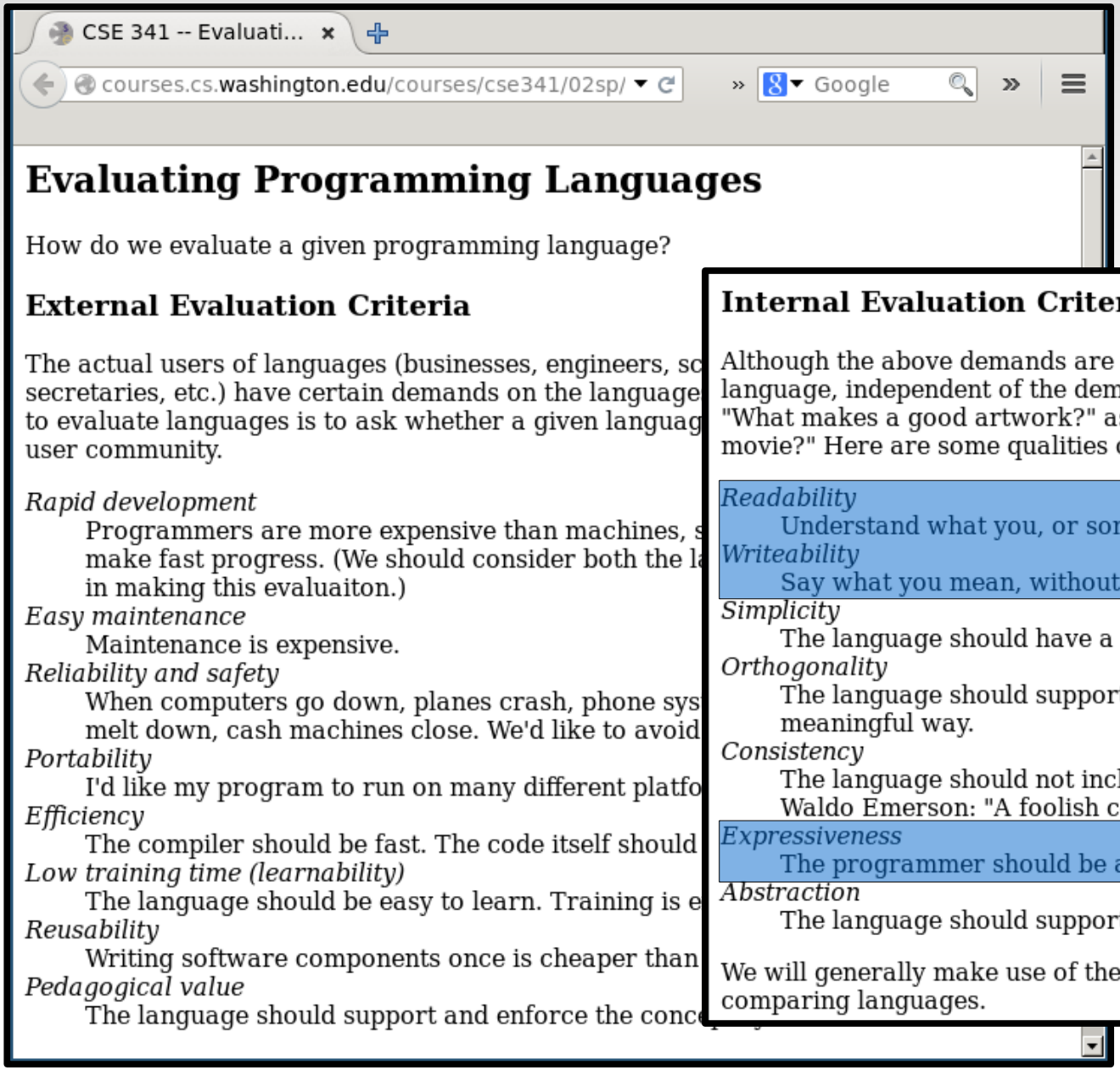
Critério de Avaliação

- Trabalhos e Apresentações

- Prova(s)



Avaliando Linguages



Internal Evaluation Criteria

Although the above demands are all important, we should still ask what makes a *good* language, independent of the demands of its users. This is a little like the question "What makes a good artwork?" as opposed to "What makes a good Hollywood movie?" Here are some qualities of a good language.

Readability
: Understand what you, or someone else has written.

Writeability
: Say what you mean, without excessive verbiage.

Simplicity
: The language should have a minimal number of primitive concepts/features.

Orthogonality
: The language should support the combination of its concepts/features in a meaningful way.

Consistency
: The language should not include needless inconsistencies. (But remember Ralph Waldo Emerson: "A foolish consistency is the hobgoblin of small minds.")

Expressiveness
: The programmer should be able to express their algorithm naturally.

Abstraction
: The language should support a high level of data and control abstraction.

We will generally make use of these and other internal evaluation criteria when comparing languages.

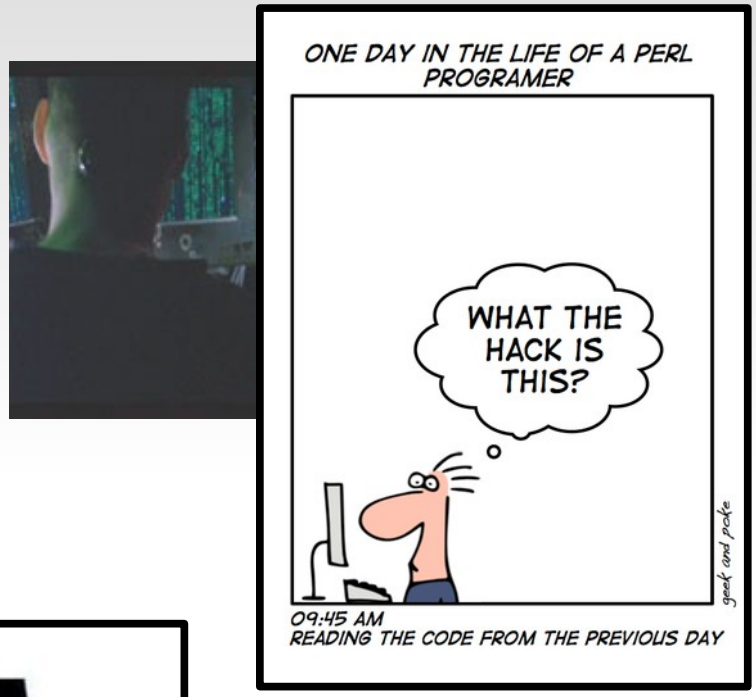
Manutenção

Prototipação

Readability vs Writability

```
while(<>) {
    split;
    print "$_[1], $_[0]\n";
}
```

```
chico@note:/data/UERJ/EDL/code$ cat names.txt
Francisco Sant'Anna
João Silva
chico@note:/data/UERJ/EDL/code$ cat names.txt | perl names.pl
Sant'Anna, Francisco
Silva, João
chico@note:/data/UERJ/EDL/code$
chico@note:/data/UERJ/EDL/code$
```



HelloWorld.java - Notepad

File Edit Format Help

```
public class HelloWorld {
  public static void main(String[] args) {
     System.out.println("Hello World!");
  }
}
```

Readability vs Writability

```c
// C

int timeOut = 1;

<...>

timeOut = 0;
```

```java
// Java

boolean timeOut = true;

<...>

timeOut = false;
```

Poder de Expressividade

18

I like Matthias Felleisen's notion of expressive power, which is *comparative*:

- Language A is strictly more expressive than language B if both of the following are true:
  - Any program written in language B can be rewritten in language A while keeping the essential structure of the program intact.
  - Some programs written in language A have to be violently restructured in order to be written in language B.

```
die ("found no solutions") unless length(solutions) > 0;
```

instead of

```
if (length(solutions) == 0) { die("found no solutions"); }
```

So you have to establish whether you're asking about expressive power of surface syntax or deeper structure.

Exemplo 1: Closures

- `code/counter-0[0-3].lua`
- Funções
  - puras vs impuras
- Estado
  - global vs encapsulado (composição?)
- Colsure como "cidadão de primeira classe"
  - Atribuição, passagem, retorno, **criação dinâmica**
- Com o que se parecem `c1` e `c2`?

Exemplo 2: Iteradores

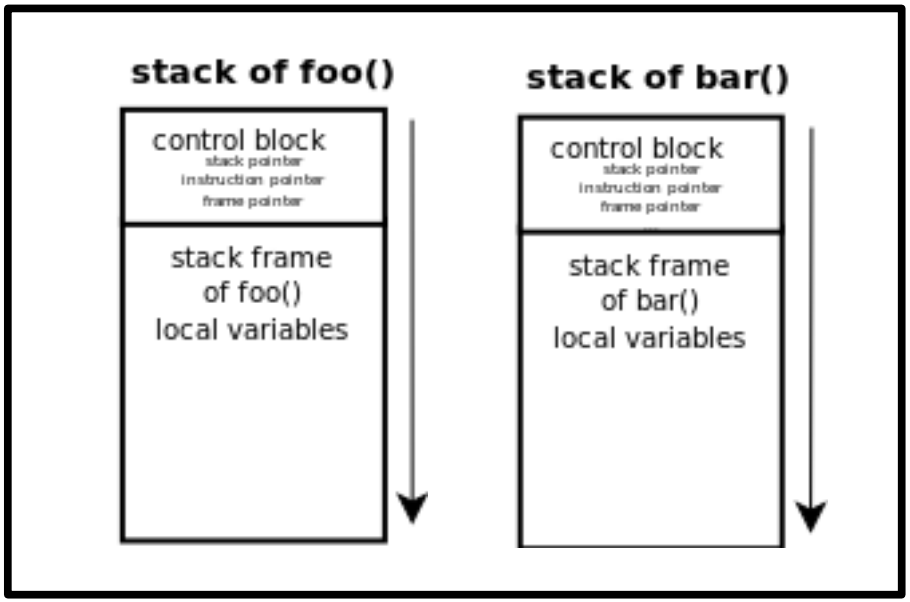
```
for i=1, 10 do
    local v = i*i
    print(i,v)
end
```

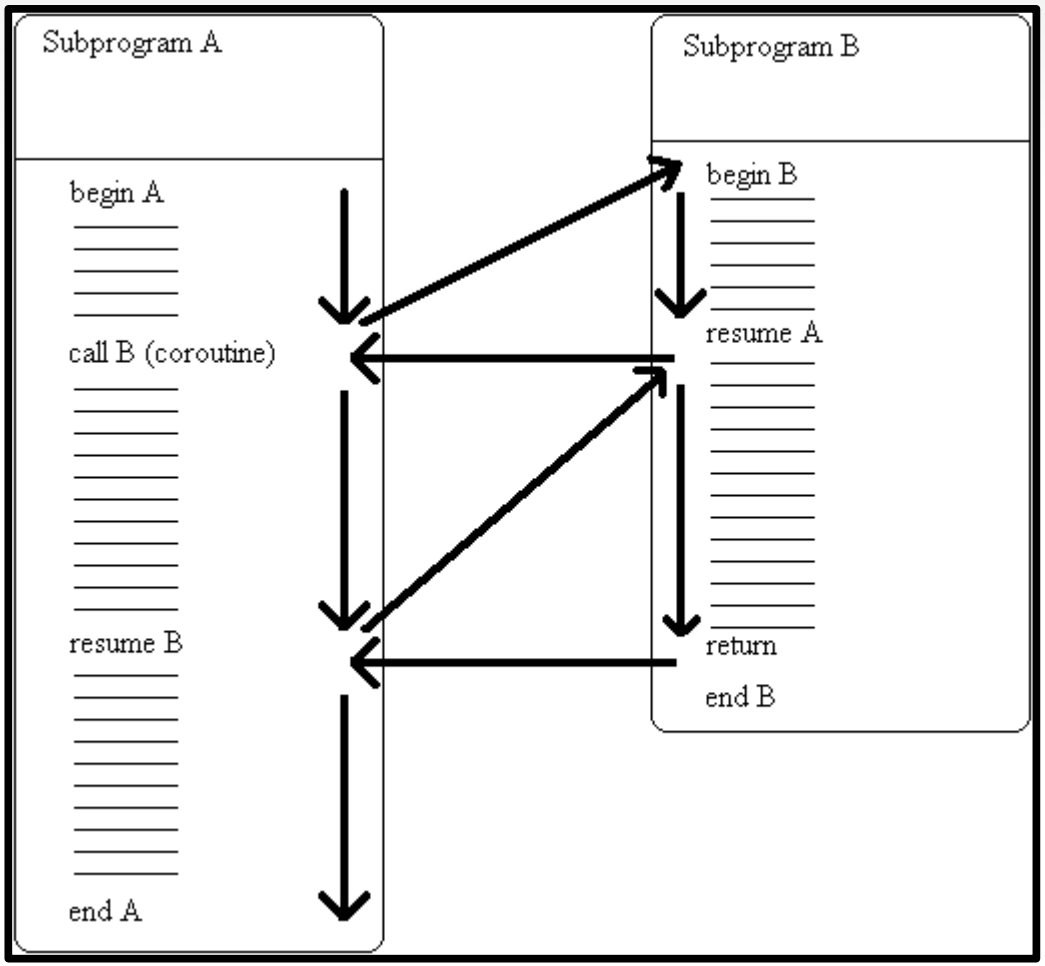
```
for i,v in <f_iter> do
    print(i,v)
end
```

- `code/iterator-0[1-3].lua`
- Estado global e encapsulado

Exemplo 2: Co-rotinas

- `code/iterator-0[4-5].lua`
- Contexto = PC, SP, pilha separada
  - estado implícito





Exemplo 2: Co-rotinas

- Controle/Pilha como “cidadão de primeira classe”
- Iteradores, Multi-Tarefa cooperativa

Comparison with subroutines [edit]

"Subroutines are special cases of ... coroutines." –Donald Knuth.[3]

Exemplo 3: Co-rotinas

- Corrida entre dois jogadores
- `code/game.lua`
- API: `player1(), player2()`
- Retorno: `'move' ou 'stand'`