

# *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

The screenshot shows a web browser displaying the GitHub repository page for 'fsantanna/EDL'. The address bar shows the URL 'https://github.com/fsantanna/EDL'. The repository page includes a search bar, navigation links for 'Pull requests', 'Issues', and 'Gist', and a sidebar with 'Unwatch', 'Star', and 'Fork' buttons. The main content area shows the repository structure with a 'Branch: master' dropdown and a 'trabalhos / trabalho-00 /' directory. The 'trabalho-00' directory contains a 'README.md' file, which is displayed in the main content area. The README file content includes the title 'Trabalho 00' and a list of instructions.

fsantanna/EDL

GitHub, Inc. [US] https://github.com/fsantanna/EDL

EDL/trabalhos/trabal... x +

GitHub, Inc. (US) https://github.com/fsantanna/EDL/tree/master/trabalhos/trabalho-00 Google

This repository Search Pull requests Issues Gist

fsantanna / EDL Unwatch 3 Star 1 Fork 0

Code Issues 0 Pull requests 0 Wiki Pulse Graphs Settings

Branch: master EDL / trabalhos / trabalho-00 / New file Upload files Find file History

fsantanna (+) edl-01, edl-02 Latest commit 62b63d9 21 minutes ago

..

README.md (+) edl-01, edl-02 21 minutes ago

README.md

## Trabalho 00

- Dar um Fork no repositório da disciplina
- Adicionar um arquivo trabalho-00/RESPOSTA.md com um texto "pessoal" qualquer formatado em Markdown.

# Trabalho 01 - Artigo

(até quarta-feira 16/03)

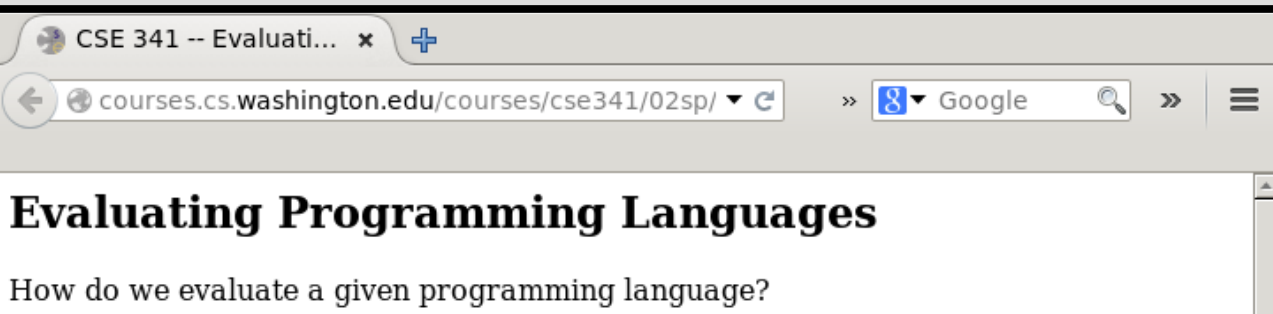
- Escolher uma linguagem com a qual você **não está familiarizado**.
  - evitar duplicatas com outros colegas
- Escrever um pequeno artigo (estilo *Wikipedia*):
  - origens e influências (linha do tempo)
  - classificação (imp/func/log/oo, est/din, usos)
  - avaliação em comparação com linguagens que 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)

```
int getRandomNumber()  
{  
    return 4; // chosen by fair dice roll.  
              // guaranteed to be random.  
}
```

# Avaliando Languages



## External Evaluation Criteria

The actual users of languages (businesses, engineers, secretaries, etc.) have certain demands on the language. To evaluate languages is to ask whether a given language meets the demands of its user community.

### *Rapid development*

Programmers are more expensive than machines, so we want to make fast progress. (We should consider both the time and money in making this evaluation.)

### *Easy maintenance*

Maintenance is expensive.

### *Reliability and safety*

When computers go down, planes crash, phone systems melt down, cash machines close. We'd like to avoid this.

### *Portability*

I'd like my program to run on many different platforms.

### *Efficiency*

The compiler should be fast. The code itself should be efficient.

### *Low training time (learnability)*

The language should be easy to learn. Training is expensive.

### *Reusability*

Writing software components once is cheaper than writing them many times.

### *Pedagogical value*

The language should support and enforce the concepts of good programming.

## 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.

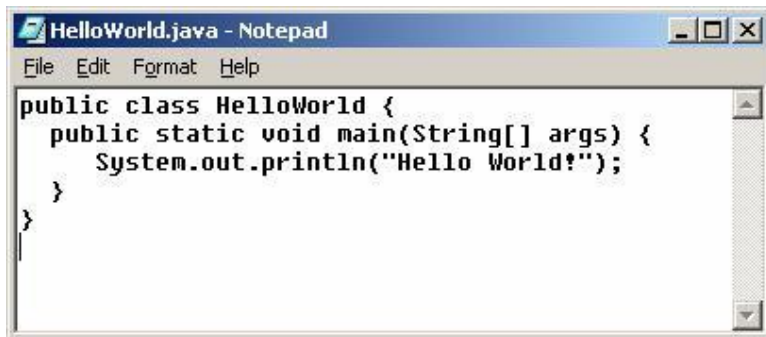
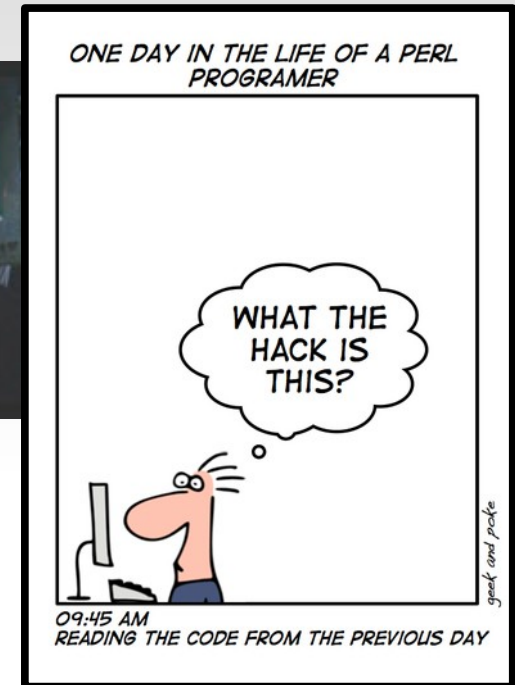
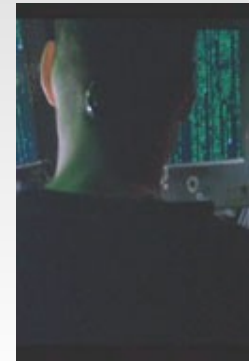
Manutenção

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$
```



# *Readability vs Writability*

```
// C
```

```
int timeOut = 1;
```

```
<...>
```

```
timeOut = 0;
```

```
// Java
```

```
boolean timeOut = true;
```

```
<...>
```

```
timeOut = false;
```



# Poder de Expressividade



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

18

- 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?)
- Closure como “cidadão de primeira classe”
  - Atribuição, passagem, retorno, **criação dinâmica**
- Com o que se parecem `c1` e `c2`?

# Exemplo 2: Iteradores

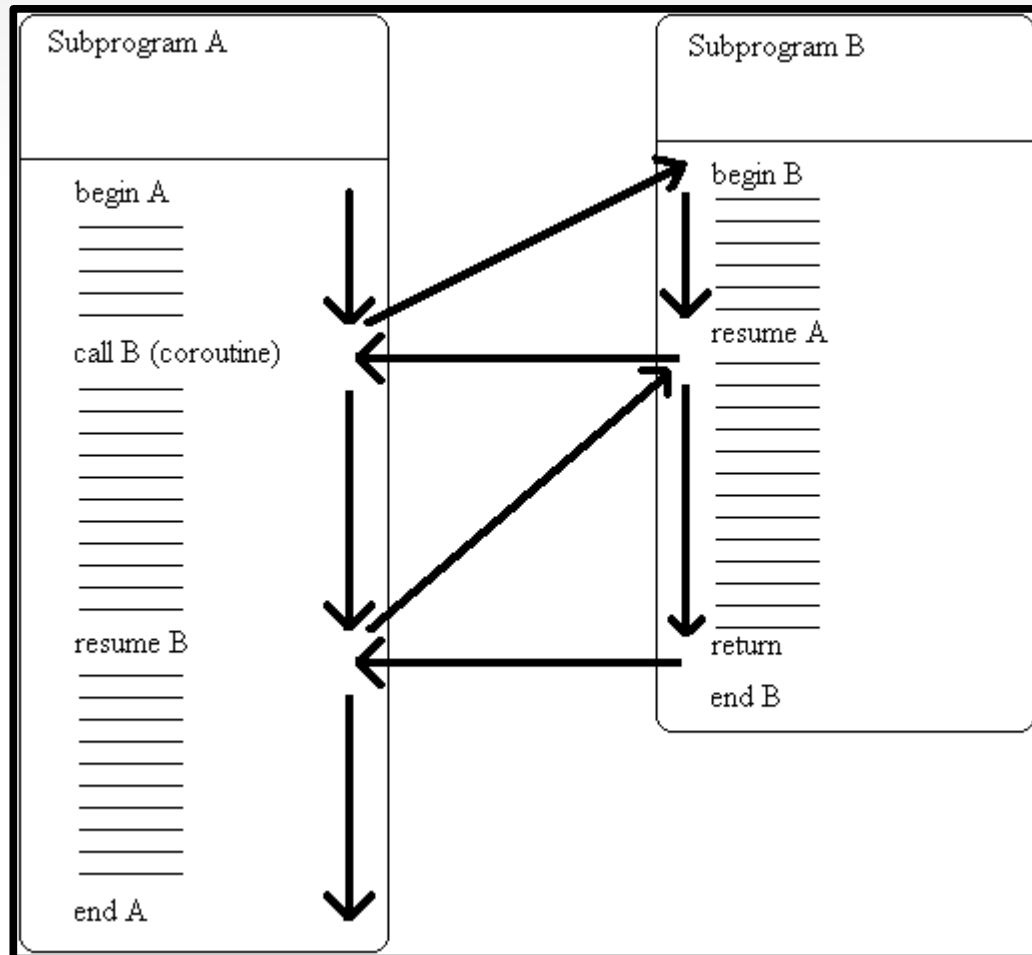
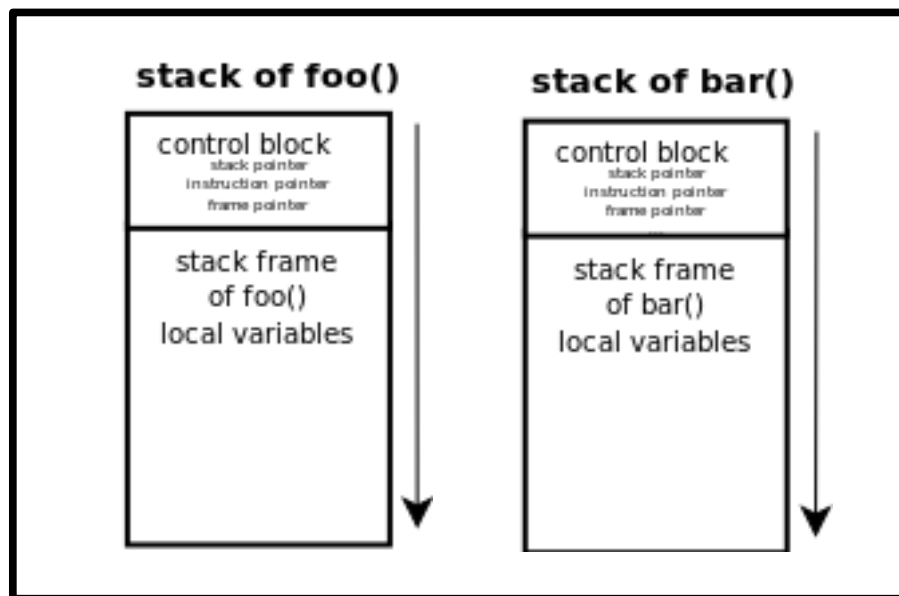
```
for i=1, 10 do  
    local v = i*i  
    print(i,v)  
end
```

```
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](#).<sup>[3]</sup>

# Exemplo 3: Co-rotinas

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