

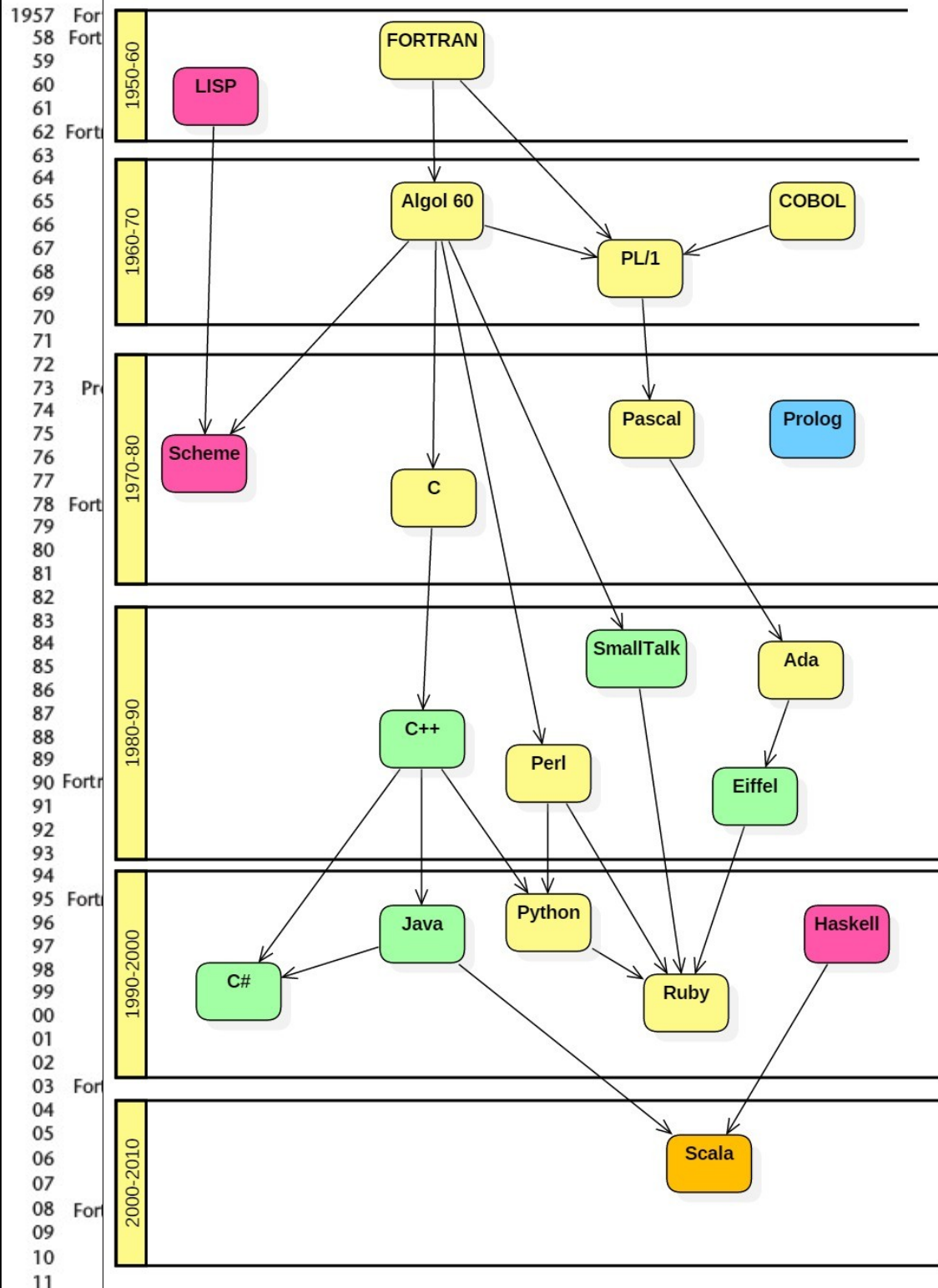
# *Estruturas de Linguagem*

**Francisco Sant'Anna  
Sala 6020-B**

**francisco@ime.uerj.br**

**<http://github.com/fsantanna-uerj/EDL>**





- Imperativa
- Funcional
- Lógica
- Orientada a Objetos
- Interpretada vs Compilada
- Dinâmica vs Estática
- Computação Científica
- Empresas e Negócios
- Inteligência Artificial
- Software Básico
- Servidores
- Internet / Web
- front/back

python 2.0

python 3.0

# Linguagens de “Baixo Nível”

- Código de Máquina

```
8B542408 83FA0077 06B80000 0000C383
FA027706 B8010000 00C353BB 01000000
C9010000 008D0419 83FA0376 078BD98B
B84AEBF1 5BC3
```

- Mapeamento 1:1 para CPU

- Máquina imperativa com espaço de endereçamento plano

- Binário vs Assembly

- Mnemônicos, Offsets, Endereços Simbólicos

- Não estamos interessados nelas

- São consequência direta da CPU

- Assembly

```
mov edx, [esp+8]
cmp edx, 0
ja @f
mov eax, 0
ret

@@:
cmp edx, 2
ja @f
mov eax, 1
ret

@@:
push ebx
mov ebx, 1
mov ecx, 1

@@:
lea eax, [ebx+ecx]
cmp edx, 3
jbe @f
mov ebx, ecx
mov ecx, eax
dec edx
jmp @b

@@:
pop ebx
ret
```

# Linguagens de “Alto Nível”

- Portabilidade
  - detalhes de arquitetura (registradores, alinhamento)
  - sintaxe uniforme
- Produtividade
  - abstrações de dados (tipos, registros, vetores, classes)
  - abstrações de controle (loops, rotinas, continuações)
  - concorrência, domínio, etc
- Performance?
  - otimizações globais
  - instruções específicas
  - complexidade

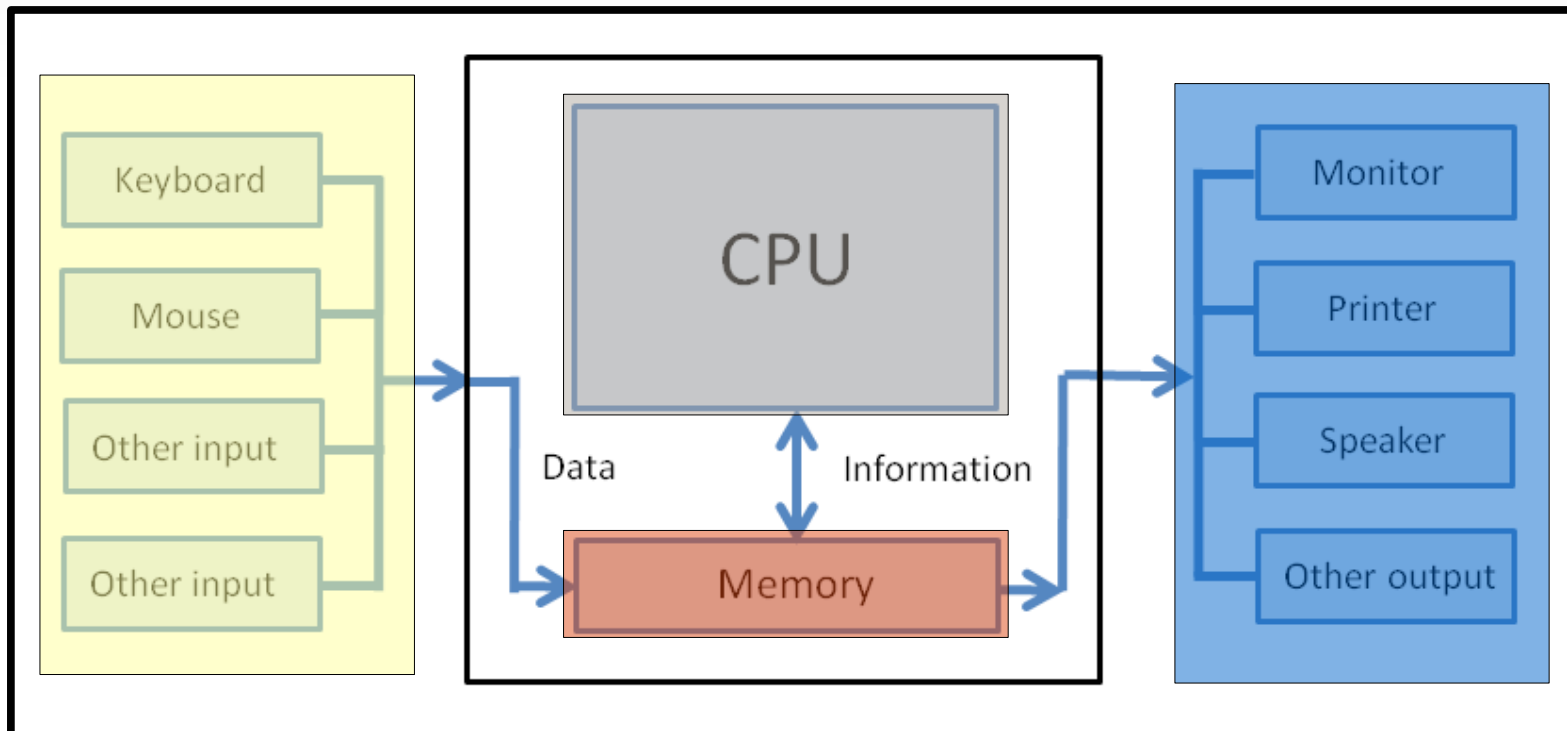
```
unsigned int (unsigned int n)
{
    if (n <= 0)
        return 0;
    else if (n <= 2)
        return 1;
    else {
        unsigned int a,b,c;
        a = 1;
        b = 1;
        while (1) {
            c = a + b;
            if (n <= 3) return c;
            a = b;
            b = c;
            n--;
        }
    }
}
```

# Linguagem de Programação

- De quem pra quem?
  - tradutor
- Um programa que *reconhece* e executa programas
  - (compilador ou interpretador da linguagem)
- *Sintaxe (forma)* e Semântica (significado)
  - (a linguagem)
- Abstração sobre o computador

# Linguagem como Abstração

```
frase = input()
print("----")
for i in range(1,5):
    print(i, frase)
```



# *Sintaxe vs Semântica*

- *Forma, Símbolos vs Significado, Execução*
- Exemplo: Como é o comando **while** de C?
  - *Sintaxe:*
    - `While ::= while ( Expression ) Statement`
    - Formal, BNF
  - Semântica:
    - Informal, RTFM!

## 3.3.4 – Control Structures

The control structures **if**, **while**, and **repeat** have the usual meaning and familiar syntax:

```
stat ::= while exp do block end
stat ::= repeat block until exp
stat ::= if exp then block {elseif exp then block} [else block] end
```

Lua also has a **for** statement, in two flavors (see §3.3.5).

The condition expression of a control structure can return any value. Both **false** and **nil** are considered false. All values different from **nil** and **false** are considered true (in particular, the number 0 and the empty string are also true).



# Sintaxe vs Semântica

Are semantics and s... x +

← stackoverflow.com/questions/209979/are-semantics-anc ↻ g Google 🔍 ☆ 📄 ⬇ 🏠 ☰


▲  
65  
▼  
✓

Syntax is the grammar. It describes the way to construct a correct sentence. For example, *this water is triangular* is syntactically correct.

Semantics relates to the meaning. *this water is triangular* does not mean anything, though the grammar is ok.

share improve this answer

answered Oct 16 '08 at 19:57

 **Christian Lescuyer**  
11.3k ● 2 ● 33 ● 37

# *Sintaxe vs Semântica*

- *Sintaxe* diferente, Semântica igual

```
chico@note:~$ lua
Lua 5.1.5 Copyright (C) 1994-2012 Lua.org, PUC-Rio
> list = { 1, 2, 3 }
> print(#list)
3
>

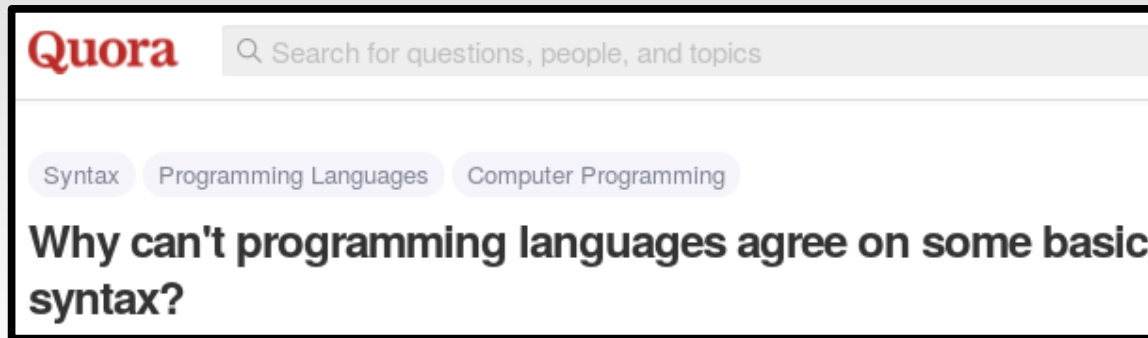
chico@note:~$ python3
Python 3.4.3 (default, Oct 14 2015, 20:28:29)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> list = [ 1, 2, 3 ]
>>> print(len(list))
3
>>>
```

- *Sintaxe* igual, Semântica diferente

```
chico@note:~$ python2
Python 2.7.6 (default, Mar 22 2014, 22:59:56)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> 1/2
0
>>>

chico@note:~$
chico@note:~$
chico@note:~$ python3
Python 3.4.3 (default, Oct 14 2015, 20:28:29)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 1/2
0.5
>>>
```

# *Sintaxe* vs Semântica



- Decisões de design:
  - indentação obrigatória em Python
  - comentários aninhados
- Semântica influencia a *Sintaxe*
  - S-expressions de LISP
  - Lambdas em linguagens funcionais

# *Sintaxe vs Semântica*

- O curso aborda, principalmente, semântica de linguagens.

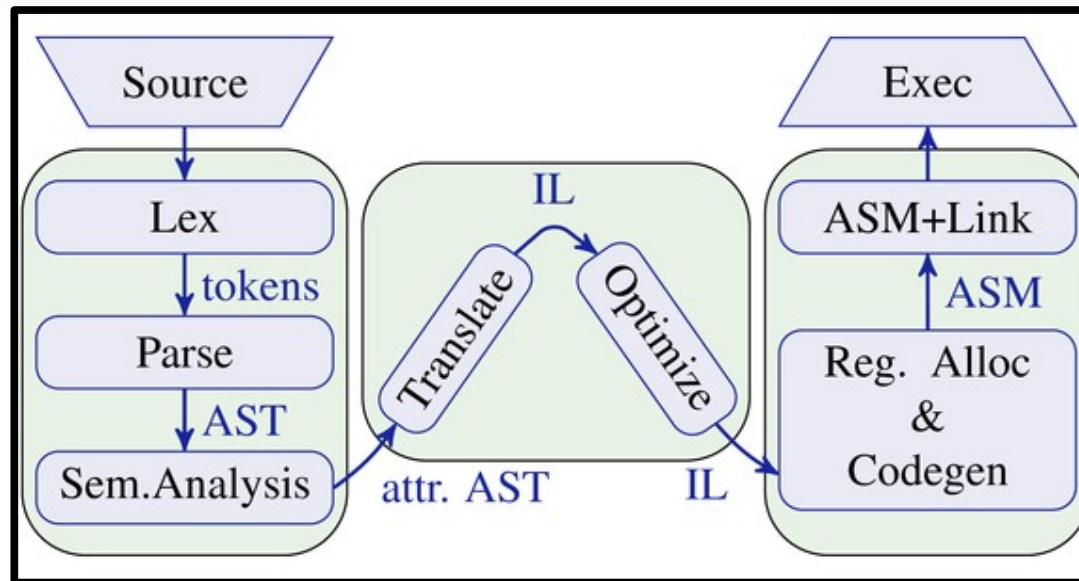
Wadler's Law states that:

In any language design, the total time spent discussing a feature in this list is proportional to two raised to the power of its position.

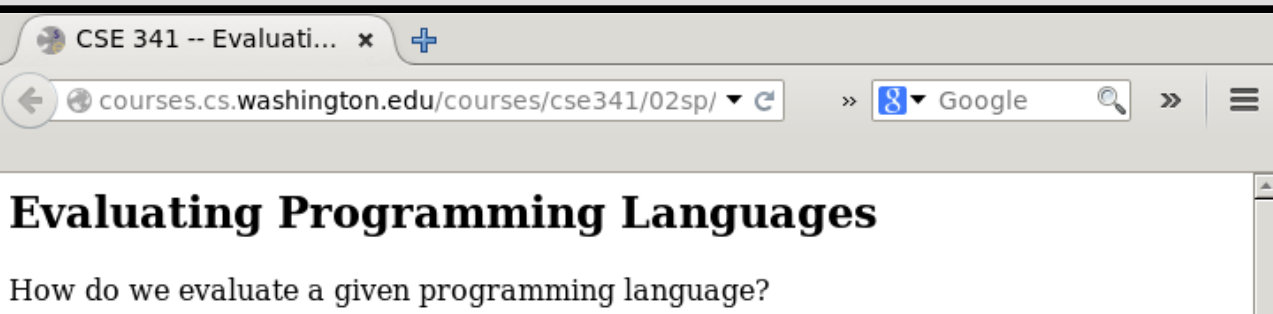
0. Semantics
1. Syntax
2. Lexical syntax
3. Lexical syntax of comments

# Compiladores

- **Não** é um curso de compiladores.
  - Implementação vs Design



# 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.

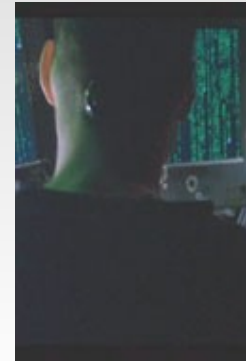
Manutenção

Prototipação

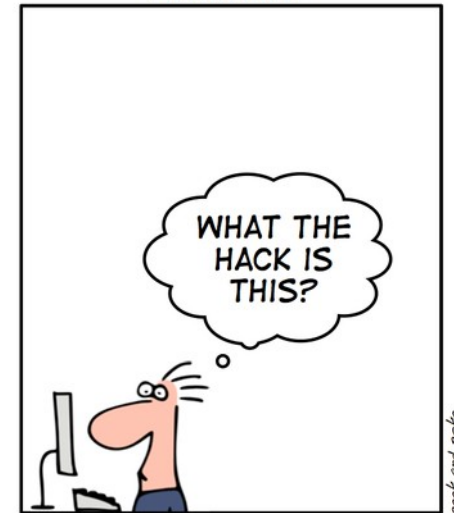
# Readability vs Writability

```
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$
```



ONE DAY IN THE LIFE OF A PERL PROGRAMMER



09:45 AM  
READING THE CODE FROM THE PREVIOUS DAY

geek and poke

```
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
```

```
int timeOut = 1;
```

```
<...>
```

```
timeOut = 0;
```

```
// Java
```

```
boolean timeOut = true;
```

```
<...>
```

```
timeOut = false;
```



# Tarefa 1

# 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.

- Exemplo Python vs C
- Python é estritamente mais expressiva que C se as duas condições forem verdadeiras:
  - Qualquer programa em C pode ser reescrito em Python mantendo a **estrutura essencial do programa** intacta.
  - Alguns programas em Python precisam ser violentamente reestruturados para serem escritos em C.

# Poder de Expressividade

- Exemplos em Lua
  - Closures: `counter/*`
  - Co-rotinas: `iter/*`