Estruturas de Linguagem

Francisco Sant'Anna

francisco@ime.uerj.br

http://github.com/fsantanna/EDL

- Nomes
- Binding (amarração)
- Variáveis
- Tipos

Nomes

3.2 What's in a Name?

Symbolic names are the workhorses of programming languages. They carry the burden of everything not implied by grammatical structure.

Names are used to:

1. Establish relationships between points in the program, by repeating the same spelling. Constantly inventing pithy unique names is burdensome. Misspellings and homonyms easily disrupt name-based relationships. Renaming is undecidable in the presence of reflection.

2. Implement abstractions, by delaying the binding of same-spelled names until compile-time or run-time. Much language semantics is smuggled in through arcane binding rules, for example method dispatch in OO. Delayed binding makes relationships implicit and contingent, obscuring them from the programmer.

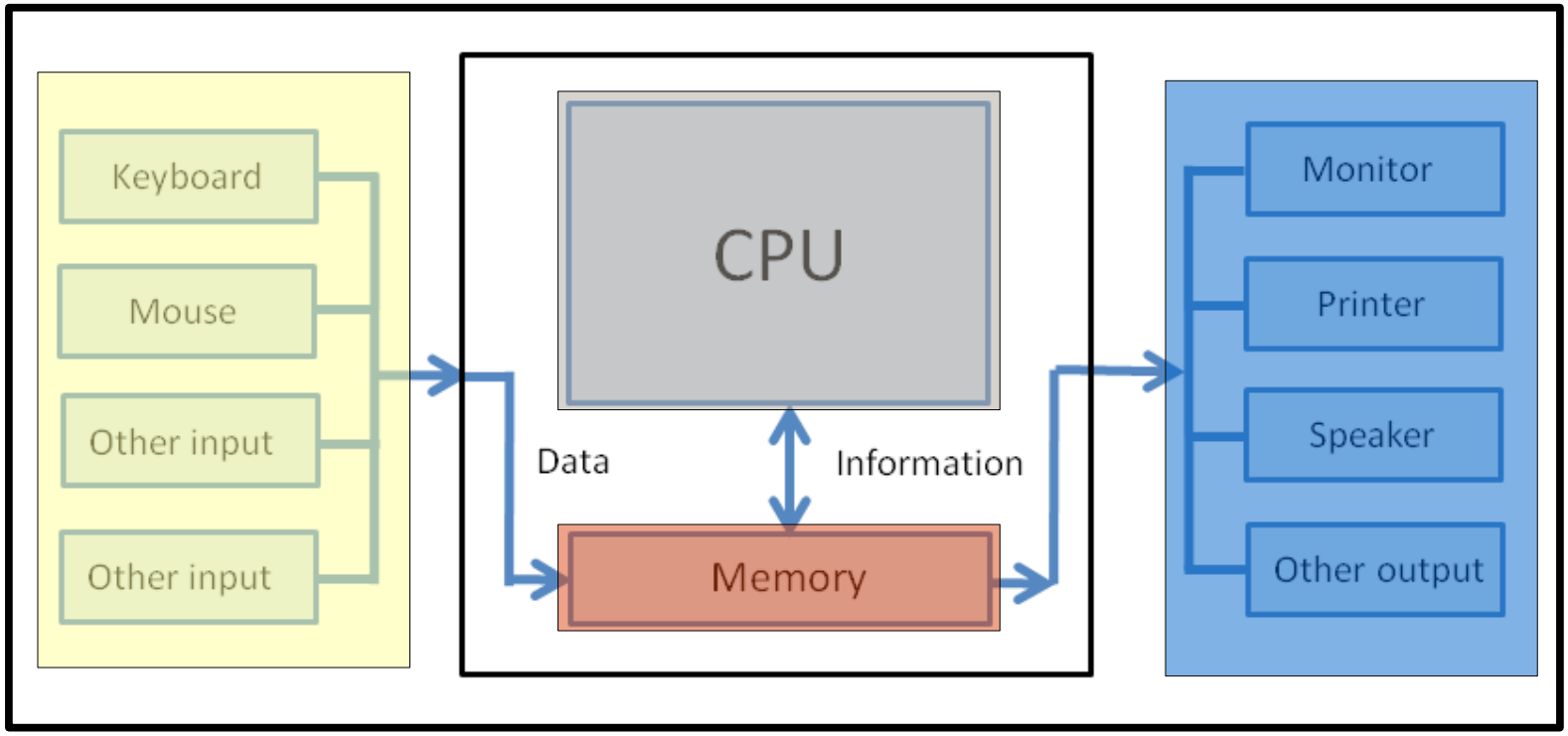
3. Serve as comments and mnemonic aids.

```
The otherUsesOfNames interfereWith this
English.Noun.Purpose.
```

Johnathan Edwards

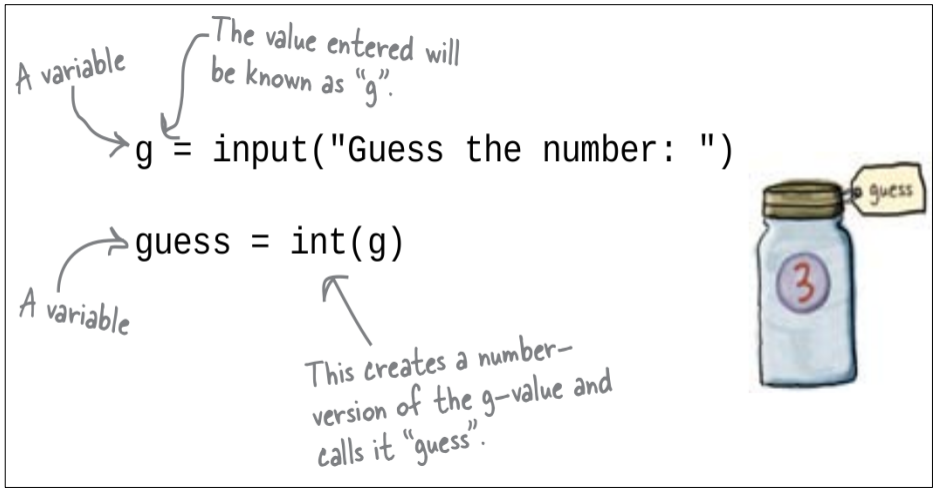
Linguagem como Abstração

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



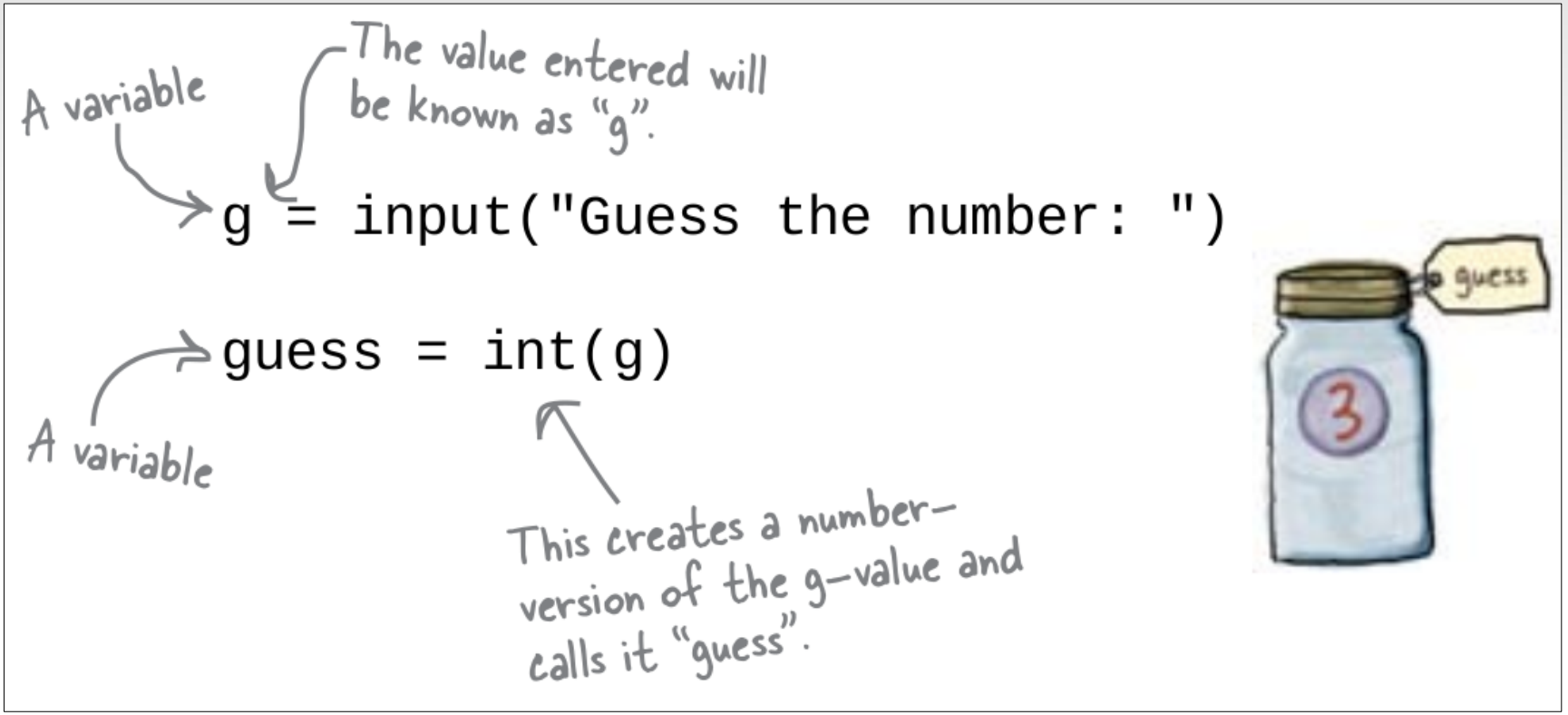
Variáveis

- Uma “etiqueta” (ou nome) que representa uma região de memória
- Uma abstração da memória do computador
  - endereço
  - valor
  - tipo
  - escopo
  - tempo de vida



Créditos: “Head First Programming”

Variáveis



Créditos: "Head First Programming" (Python 3)

Sintaxe - Forma

- string de caracteres
  - `i, minha_variavel`
  - `10i, $i, variável, if`
- Palavras reservadas?
- “Case sensitive”?
- Caracteres especiais?

> Names in most programming languages have the same form: a letter followed by a string consisting of letters, digits, and underscore characters (_).

Sintaxe - Forma

Instance variable: self vs @

120

43

Here is some code:

```
class Person
  def initialize(age)
    @age = age
  end

  def age
    @age
  end

  def age_difference_with(other_person)
    (self.age - other_person.age).abs
  end

  protected :age
end
```

What I want to know is the difference between using `@age` and `self.age` in `age_difference_with` method.

Sintaxe - Forma

```
@numeros = (0,1,2);
$numeros = @numeros;
print "$numeros: @numeros\n";
```

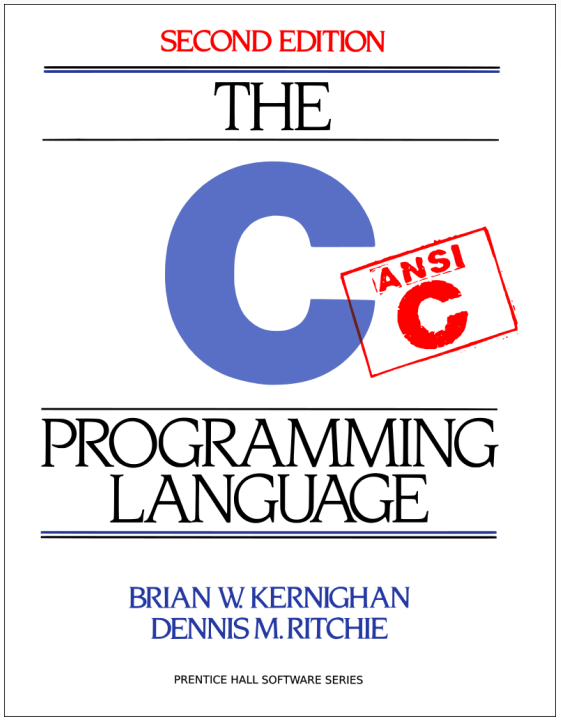
```
$ perl numeros.pl
3: 0 1 2
$
```

Binding (Amarração)

- Associação entre “entidade” e “atributo”
  - *binding time*
    - *language design time*
    - *language implementation time*
    - *preprocess time*
    - *compile time*
    - *link time*
    - *load time*
    - *run time*

Binding (Amarração)

- *language design time*
  - especificação da linguagem



Binding (Amarração)

- *language design time*
- *language implementation time*



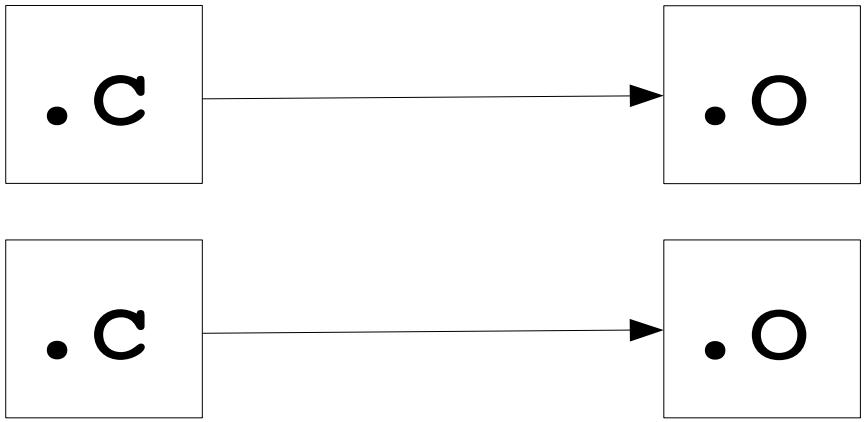

Binding (Amarração)

- *language design time*
- *language implementation time*
- *preprocess time*

```
#include ...
#define ...
#ifdef ...
#endif
```

Binding (Amarração)

- *language design time*
- *language implementation time*
- *preprocess time*
- *compile time*



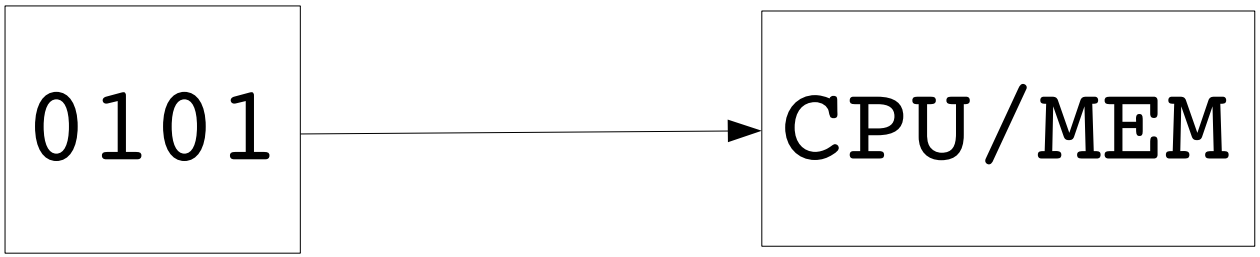
Binding (Amarração)

- *language design time*
- *language implementation time*
- *preprocess time*
- *compile time*
- *link time*

```
$ gcc -lpthread ...
```

Binding (Amarração)

- *language design time*
- *language implementation time*
- *preprocess time*
- *compile time*
- *link time*
- *load time*



Binding (Amarração)

- *language design time*
- *language implementation time*
- *preprocess time*
- *compile time*
- *link time*
- *load time*
- *run time*

```
$ ./a.out
```

Binding - Exemplo

```
#include <stdio.h>
#include <math.h>
#define PI 3.14
static int v = 10;
int f (void);
int main (void) {
    uint8_t x = sin(PI) + v + f();
    return x;
}
```

- Valor de **PI**
- Endereço de **v**
- Tamanho de **int**
- Implementação de **f**
- Tipo de retorno de **f**
- Tamanho de **uint8_t**
- Endereço de **x**
- Semântica de “**+**”
- Implementação de **sin**
- Escopo

Binding - Estático vs Dinâmico

- Estático
  - binding ocorre antes da execução (e não é alterado durante a execução)
- Dinâmico
  - binding ocorre durante a execução

Lua: Binding Times

- *language design time*
- *language implementation time*
- *preprocess time*
- *compile time*
- *link time*
- *load time*
- *run time*

Lua: Binding Times

- *language design time*
- ~~*language implementation time*~~
- ~~*preprocess time*~~

```
$ cat lua-compile.lua
print(1/3)
$ lua5.3 lua-compile.lua
0.33333333333333
$ luac5.3 -l lua-compile.lua

main <lua-compile.lua:0,0> (4 instructions at 0x211ab20)
0+ params, 2 slots, 1 upvalue, 0 locals, 2 constants, 0 functions
        1       [1]     GETTABUP        0 0 -1  ; _ENV "print"
        2       [1]     LOADK           1 -2    ; 0.33333333333333
        3       [1]     CALL            0 2 1
        4       [1]     RETURN          0 1
$
```