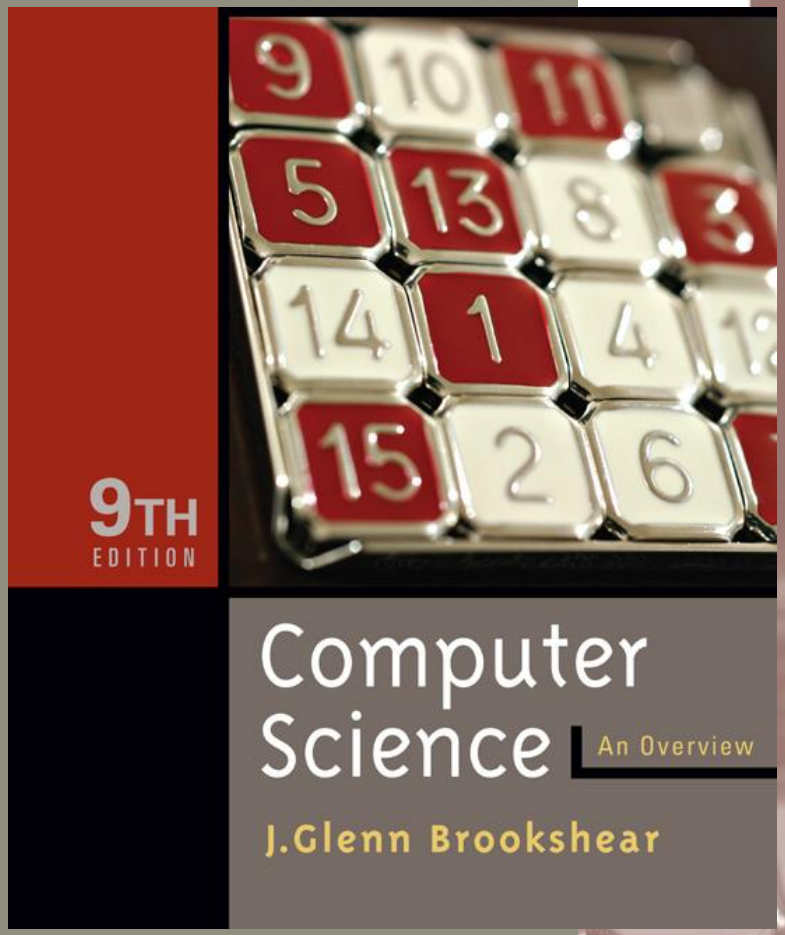
Capítulo 6

Linguagens de Programação



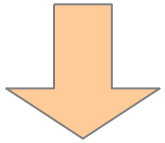
Modificados para MIM - Introdução à Informática (2425) por Pedro Ribeiro, Miguel Areias e Nuno Guimarães

Capítulo 6: Linguagens de programação

- 6.1 Perspetiva histórica
- 6.2 Conceitos de Programação Tradicionais
- 6.3 Procedimentos
- 6.4 Implementação de Linguagens
- 6.5 Programação Paralela
- 6.6 Programação Declarativa

Gerações de linguagens de programação

- Várias "gerações" de linguagens de programação:
  - Problemas resolvidos num ambiente onde o ser humano tem de se adaptar às características da máquina



  - Problemas resolvidos num ambiente onde deve ser a máquina a adaptar-se às características do ser humano

Gerações de linguagens de programação (continuação)

- **"1ª" Geração:** código máquina
- **"2ª" Geração:** linguagens ***assembly***
  - *Mnemónicas para código máquina*
- **"3ª Geração:"** Mais alto nível que 2ª (finais anos 50)
  - Ex: Fortran, Algol (passado)
    C, C++, Java, Python (presente)
- **"4ª" ou "5ª" Geração?** mais "declarativas"
  - Ex: Prolog

2ª Geração: Linguagens *Assembly*

- Um sistema de mnemónicas para representar programas
  - Nomes em mnemónicas para *op-codes*
  - Nomes para todos os registos
  - Identificadores: nomes para posições de memórias, escolhidos pelo programador

Assembly: características

- Correspondência um-para-um entre instruções máquina e instruções em assembly
  - O programador deve pensar como uma máquina
- Por inerência, dependente da máquina
- Convertido para linguagem máquina por um programa chamado **assemblador**

Exemplo de Assembly

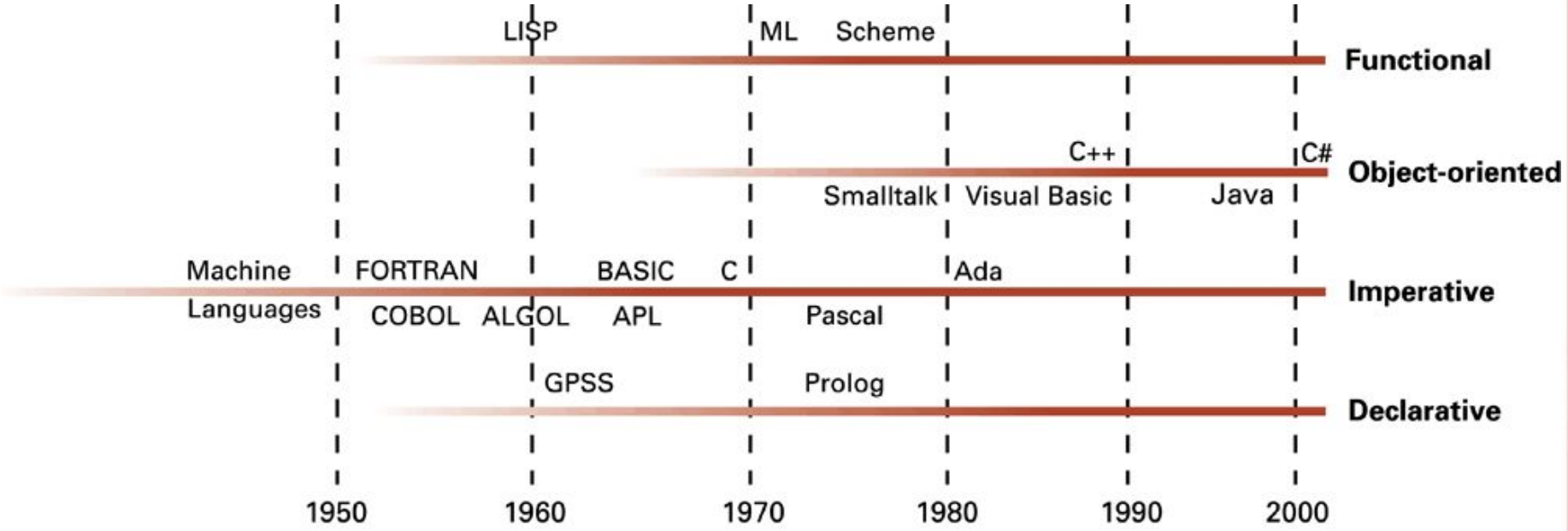
Linguagem máquina	*Assembly*
156C	LD R5, Price
166D	LD R6, ShippingCharge
5056	ADDI R0, R5, R6
30CE	ST R0, TotalCost
C000	HLT

Linguagens de 3ª geração

- Usa primitivas de mais alto nível
  - "Semelhante" ao pseudo-código
- Independente da máquina (no geral)
- Exemplos: FORTRAN, COBOL
- Cada primitiva corresponde a uma pequena sequência de instruções em linguagem máquina
- Convertida para linguagem máquina por um programa chamado **compilador**

Modificados para MIM - Introdução à Informática (2425) por Pedro Ribeiro, Miguel Areias e Nuno Guimarães

A evolução dos paradigmas de programação



Paradigmas de Programação

- Programação **imperativa**:
  - Sequência de "ordens" explícitas
  - Modificação do estado do programa
  - Controlo total do fluxo do programa
- Programação **declarativa**:
  - Descrever a lógica sem definir exatamente o fluxo do programa e o estado geral
  - Computador "decide" o fluxo

Paradigmas de Programação

- Programação **interpretada**:
  - Não precisa de compilar o programa em linguagem máquina
  - Não é executada diretamente pela máquina
  - Mais lentas no geral
  - Exemplo: Python, Javascript

Tipos de dados

- **Inteiros:** Números inteiros (ex: 3)
- **Reais:** Números reais (ex: 10.23)
- **Caracter:** Símbolos (ex: 'A')
- **Booleanos:** Verdadeiro/Falso

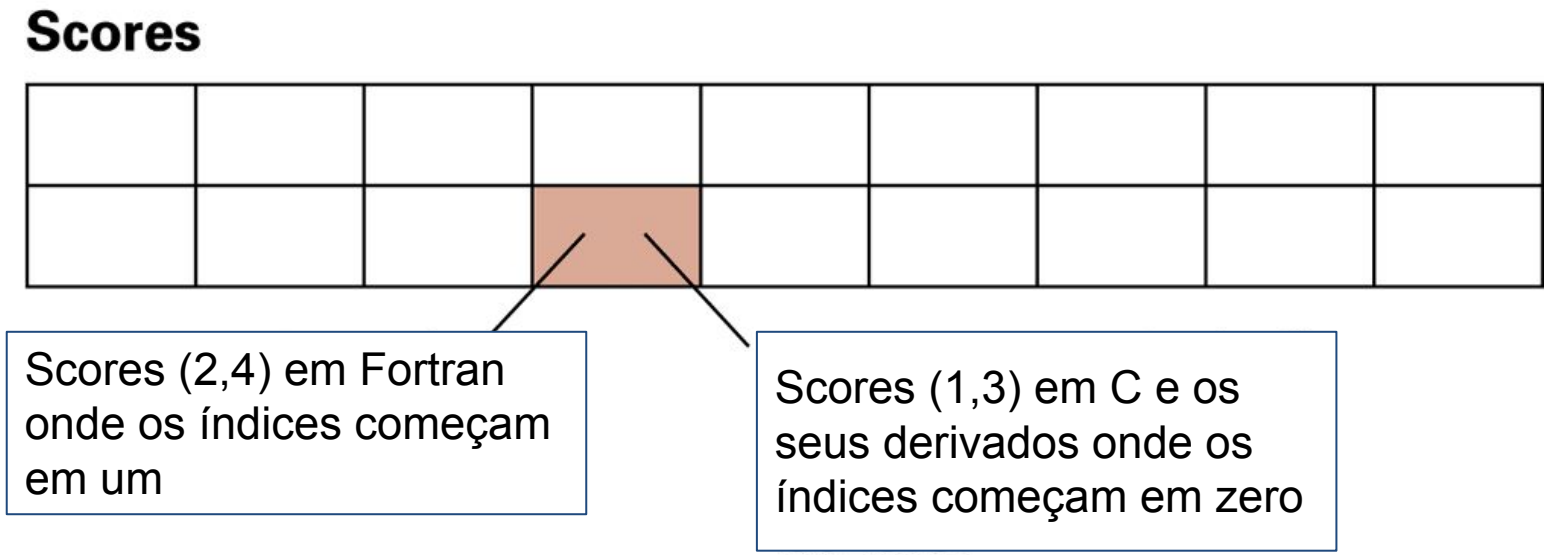
Declaração de variáveis em C, C++, C#, e Java

```
float    altura, largura;
int      preco, total, taxa;
char     simbolo;
```

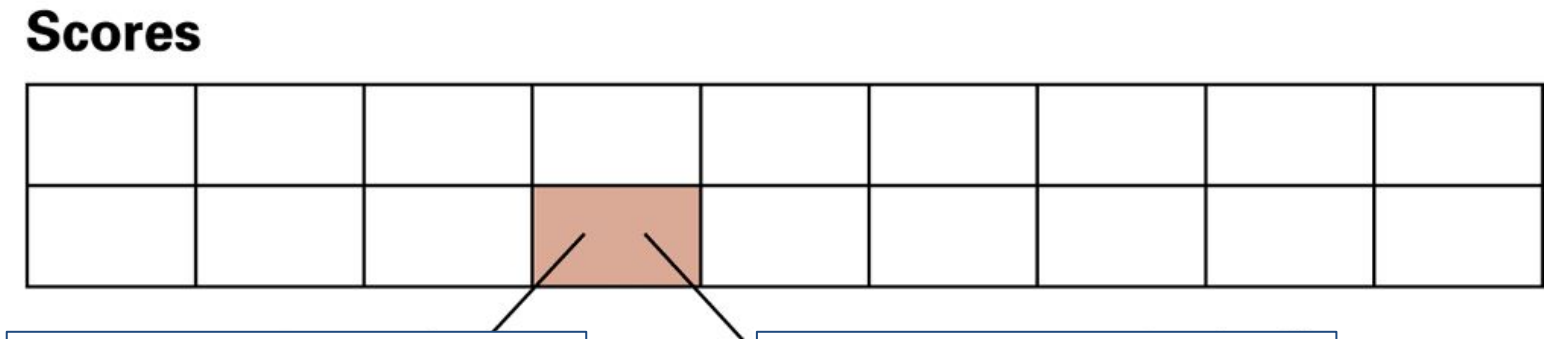
Declaração de variáveis em Python

Variáveis em Python não precisam de ser declaradas

Um vetor (*array*) bidimensional com nove elementos

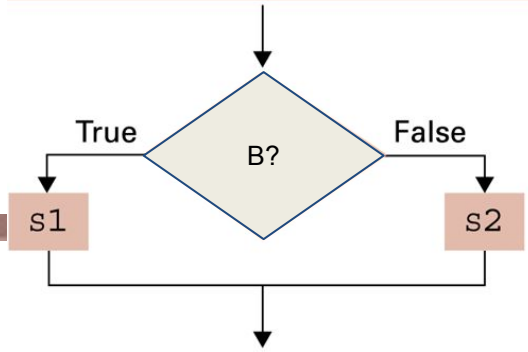
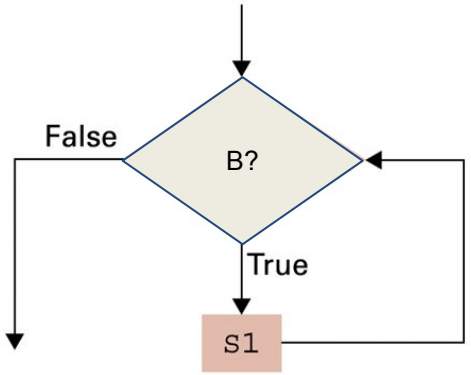


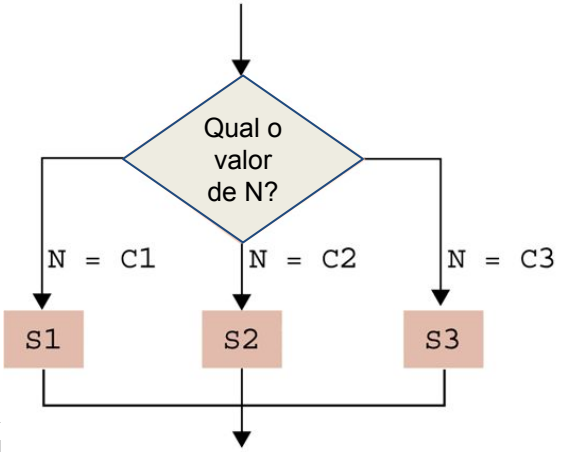
Um vetor (*array*) bidimensional com nove elementos



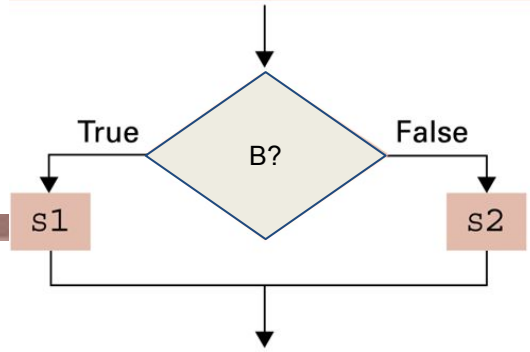
Igual em Python

Estruturas de controle e suas representações em *C*, C++, C#, e Java

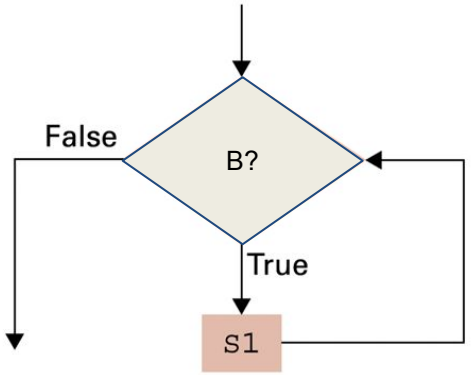
Estruturas de Controlo	C, C#, C++ e Java
	`if (B) S1` `else S2;`
	`while (B)` `  S1;`

	`switch (N)`
	`{ case C1: S1; break;`
	`  case C2: S2; break;`
	`  case C3: S3; break;`
	`};`

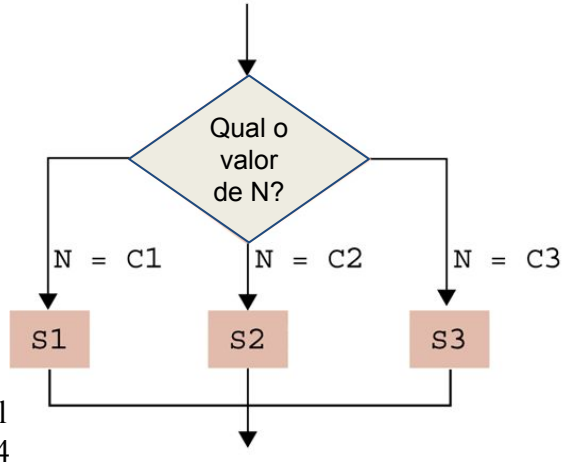
Estruturas de controle e suas representações em *Python*




```
if (b):
    S1
else:
    S2
```

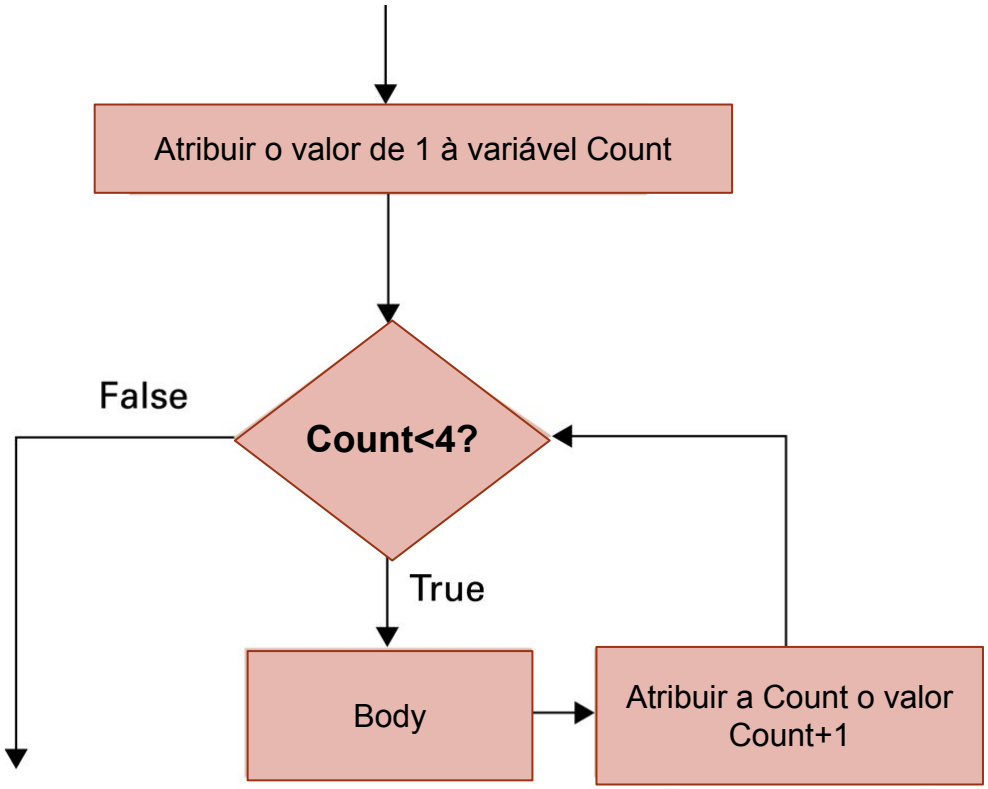



```
while (b):
    S1
```



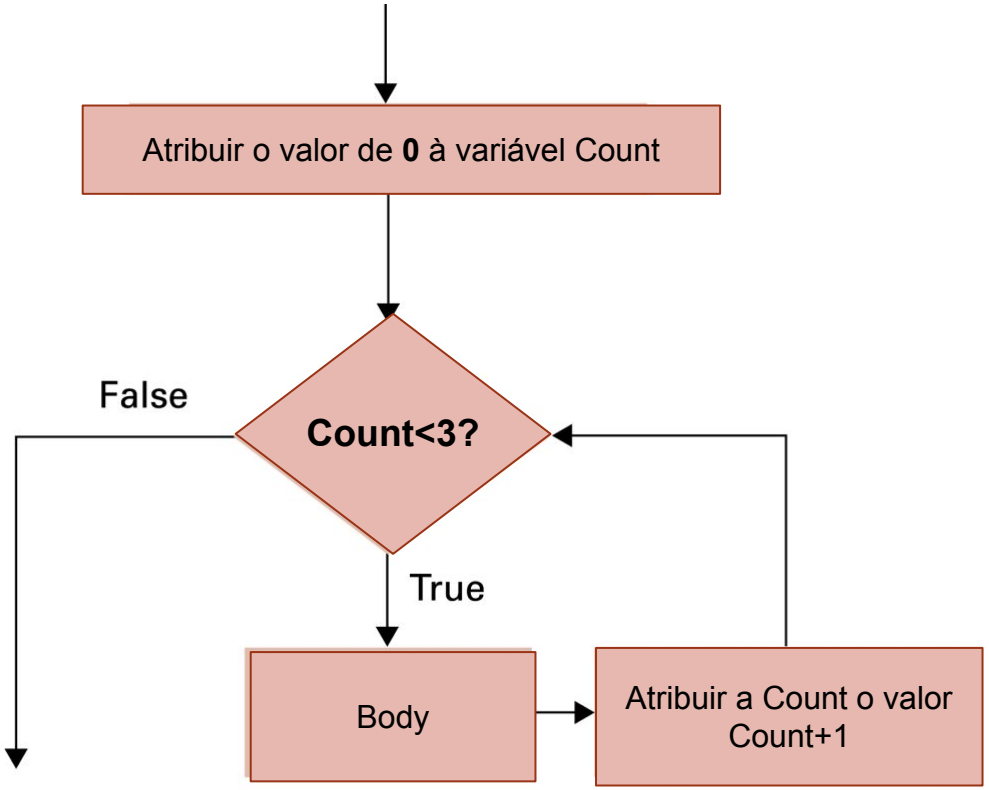
Não existe switch em Python (tem que ser construído com ifs elses)

A estrutura de *loop* e suas representações em *C*, C++, C#, e Java




```
for (int Count = 1; Count<4; Count++)
    body ;
```

A estrutura de *loop* e suas representações em *Python*




```
for i in range(0,3):
    body
```

Constantes e comentários

- Por vezes não se quer mudar nunca o valor de uma variável.
- Declara-se por isso uma **constante**. Em Python são normalmente declaradas com o nome da variável em maiúsculas. Em C existe uma palavra reservada para as mesmas
  - const int PI = 3.14159265 (em C)
  - Ex: PI = 3.14159265 (em Python)
- Podem adicionar-se informações destinadas a explicar o que está a ser feito. São chamados de **comentários**:
  - Ex: # Isto é um comentário em Python
  - /* Isto é um comentário em C e em Java*/

O procedimento ProjectPopulation escrito na linguagem de programação C

começar o cabeçalho com o termo "*void*" permite especificar que o bloco de código é um procedimento e não uma função.

a lista de parâmetros . O C, assim como muitas outras linguagens de programação, requer que o tipo de cada parâmetro seja especificado

```
void   ProjectPopulation   (float GrowthRate)
{ int Year;
  Population[0] = 100.0;
  for (Year = 0; Year =< 10; Year++)
  Population[Year+1] = Population[Year] + (Population[Year] * GrowthRate);
}
```

declaração de uma variável local chamada de *Year*

Estas instruções descrevem como a população vai ser computada e guardada num array global de nome *Population*

O procedimento ProjectPopulation escrito na linguagem de programação Python

Forma de o Python saber que estamos a falar de uma função ou procedimento

Lista formal de parâmetros.
No Python não precisamos de declarar o tipo de variável

```
def PrintWelcomeMessage (list_names):

    for name in list_names:
        print("Bem-vindo " + name)
```

Chamada do procedimento e output das mensagens

```
In [530]: PrintWelcomeMessage(["Ana","Paulo"])
Bem-vindo Ana
Bem-vindo Paulo
```

Itera sobre os diferentes nomes e mostra a mensagem "Bem-vindo x"

A função “CylinderVolume” escrita em C

The function header begins with the type of the data that will be returned.

```
float CylinderVolume (float Radius, float Height)
{ float Volume;
  Volume = 3.14 * Radius * Radius * Height;
  return Volume;
}
```

Declare a local variable named Volume.

Compute the volume of the cylinder.

Terminate the function and return the value of the variable Volume.

A função “CylinderVolume” escrita em C

```
def CylinderVolume (radius, height):
    volume=3.14*radius*radius*height
    return volume
```

Retorna o valor calculado

calcula o valor do cilindro

```
In [531]: nomes=PrintWelcomeMessage(["Ana","Paulo"])
Bem-vindo Ana
Bem-vindo Paulo

In [532]: nomes
```

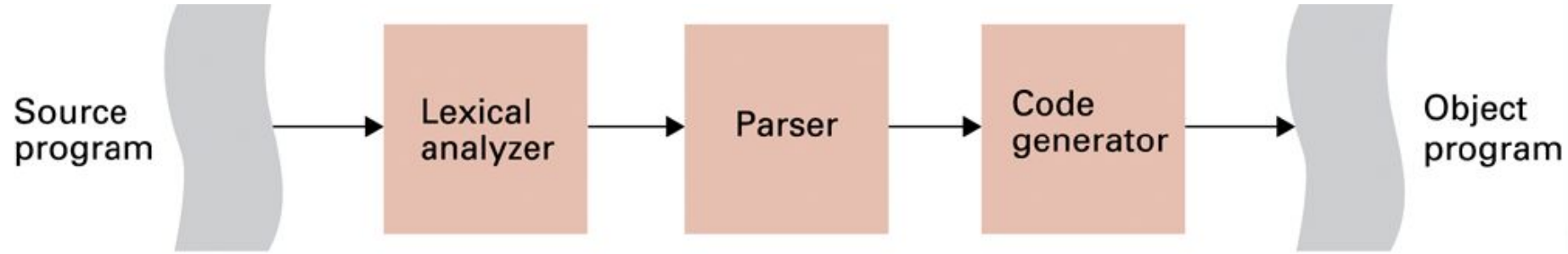
Nomes não contêm o resultado

```
In [536]: v=CylinderVolume(2,3)

In [537]: v
Out[537]: 37.68
```

v contêm o resultado retornado da função

O processo de tradução



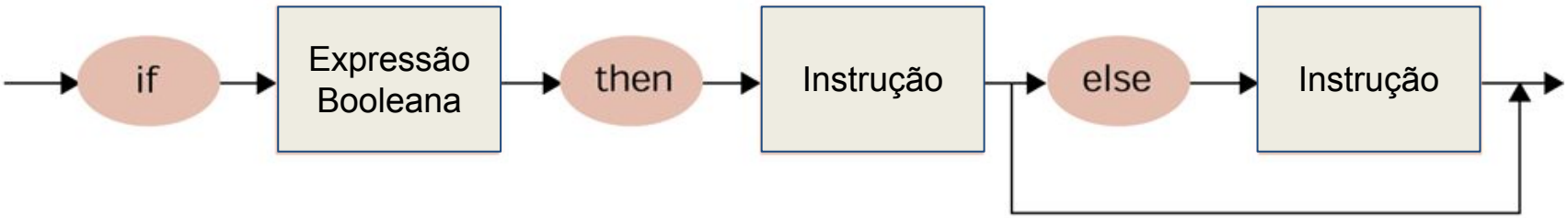
Análise Léxica

- Reconhecer as “entidades”/símbolos (tokens) do programa original.
  - Ex: 123 é “cento e vinte e três” e não “um” “dois” “três”
  - Ex: “if” é a palavra reservada da instrução condicional e não uma sequência qualquer de caracteres

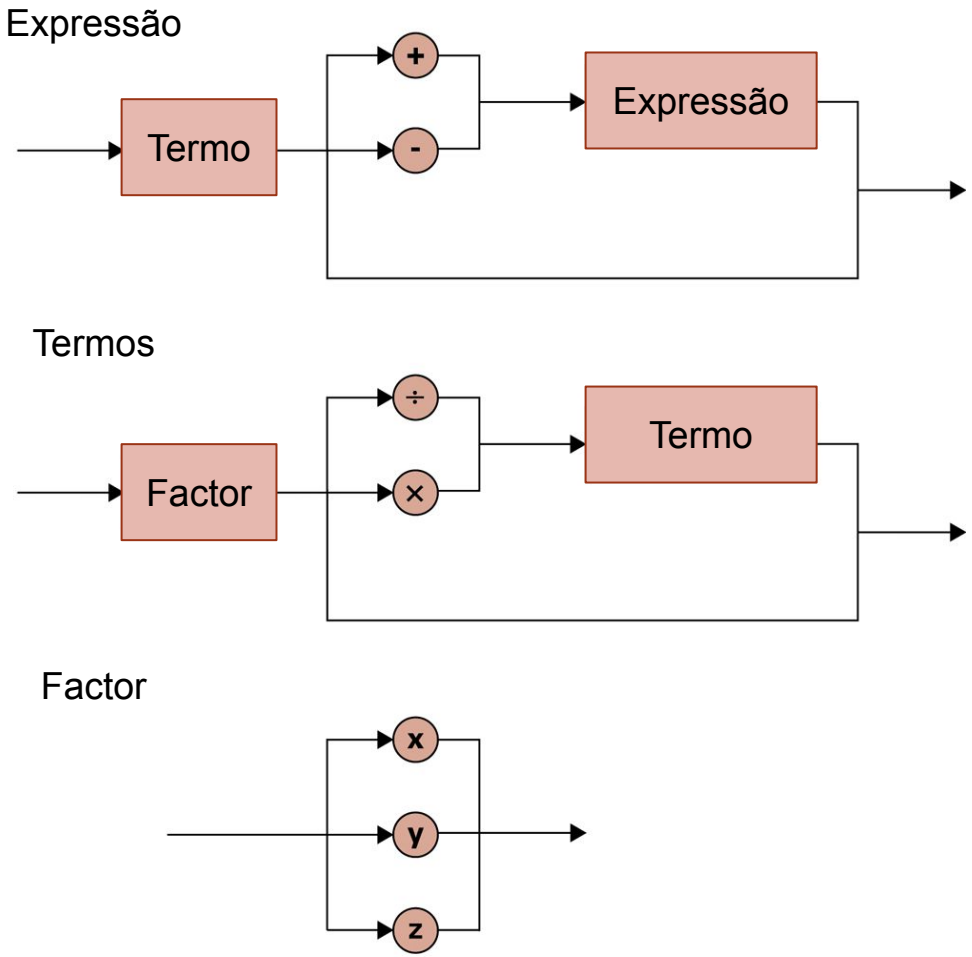
Análise Sintática

- Agrupar os tokens em “frases”/instruções com sentido.
- Conjunto de regras que definem sintaxe: **gramática**.

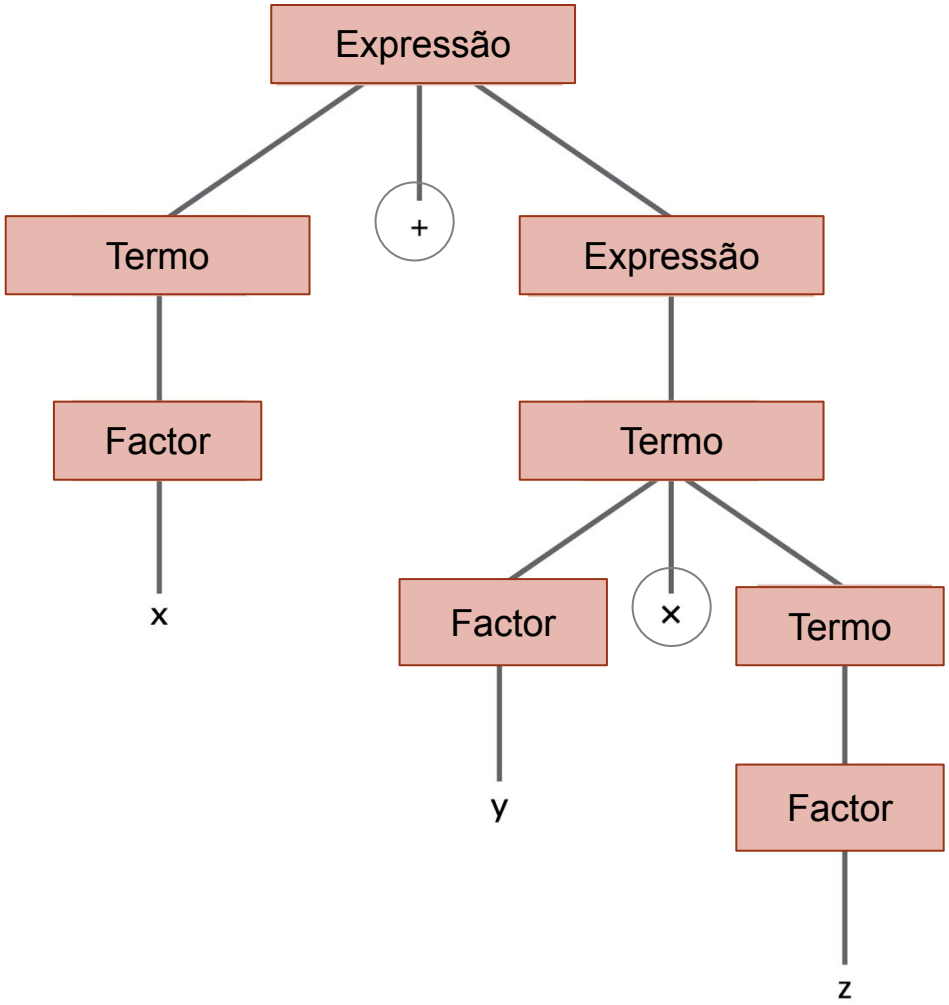
Um diagrama de sintaxe do *if-then-else*



Expressões algébricas simples



Árvore de *parsing* da *string* x + y x z baseada na sintaxe anterior

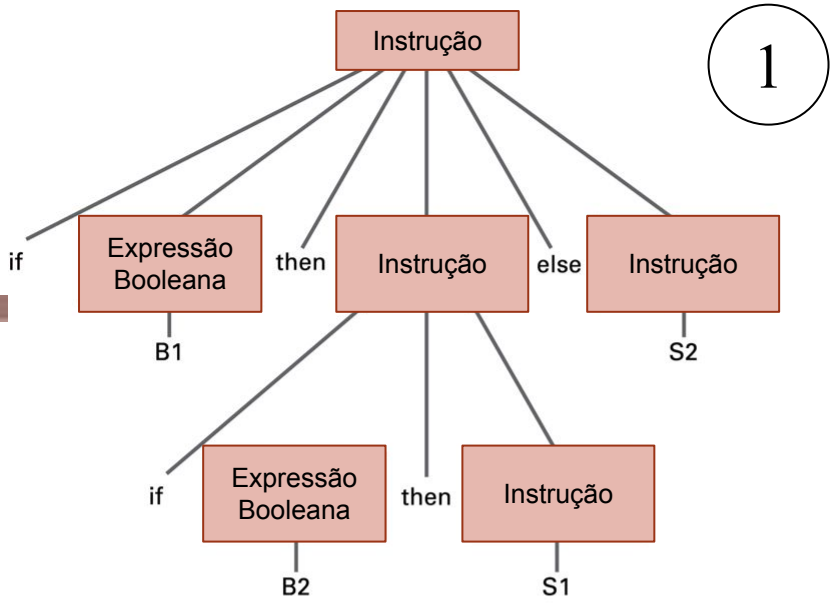


Duas árvores de *parsing* para a expressão

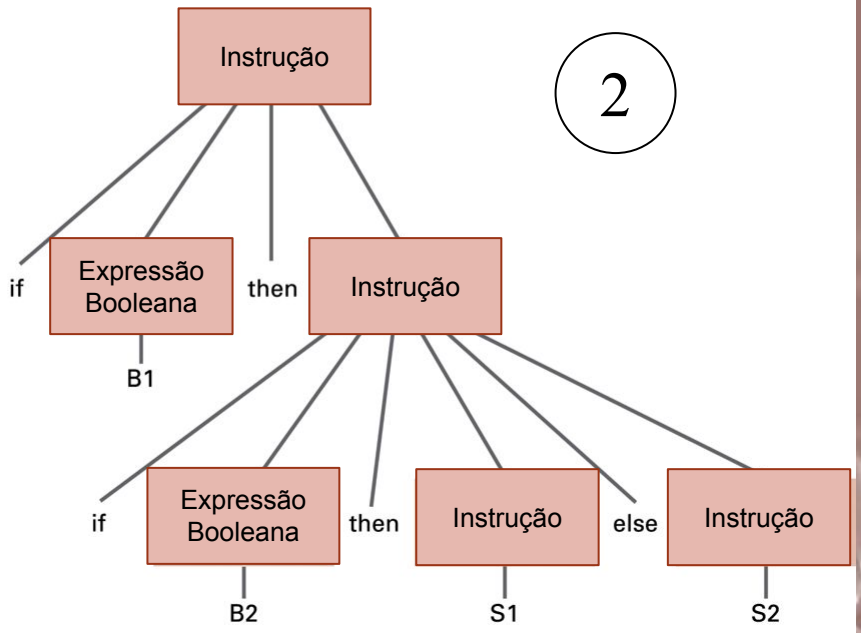
if B1 then if B2 then S1 else S2

if B1 then (if B2 then S1) else S2 (1)

if B1 then (if B2 then S1 else S2) (2)



1



2

Programação declarativa (em particular programação lógica)

- **Lógica formal** dá-nos um "algoritmo" geral para resolução de problemas
- **Resolução:** combinar dois ou mais factos (ou "frases") para produzir um novo facto (ou "frase") equivalente
  - **Resolvente:** um novo facto obtido por resolução
- **Cláusula:** facto cujos componentes elementares são ligados por operadores `OR` (ex: "P → Q" é equivalente a "Q OR ¬P")
- **Unificação:** atribuir um valor a uma variável de modo a que duas "frases" fiquem compatíveis

Programação declarativa (Prolog)

- **Facto:** *nomePredicado*(*argumentos*).
  - Ex: `progenitor(carlos, ana).`
    `progenitor(madalena, ana).`
  - `homem(carlos). mulher(ana). mulher(madalena).`
- **Regra:** *conclusão* :- *premissa*.
  - :- significa "se"
  - Ex: `filha(X, Y) :- progenitor(Y, X).`
    `pai(X, Y) :- homem(X), progenitor(X, Y).`
  - `mae(X, Y) :- mulher(X), progenitor(X, Y).`