Programar

Uma Introdução à Programação

António Abelha -2020

Programar

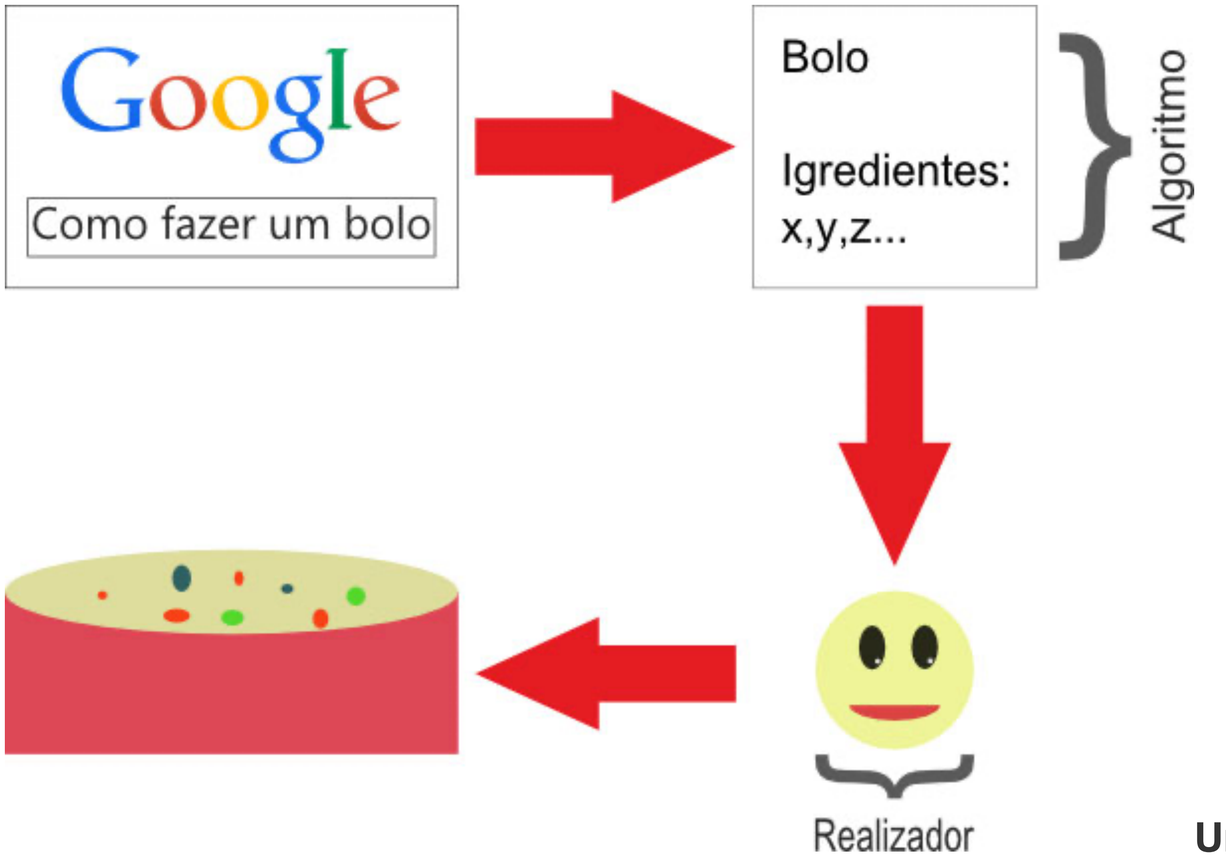
Algoritmos

Algoritmos são quaisquer sequências de instruções bem definidas e finitas.

Podem ser executados por computadores, sistemas automatizados ou mesmo humanos.

Programar

Algoritmos



Um algoritmo pode ser qualquer sequência de instruções

Programar

Fluxogramas

Os Algoritmos podem ser representados de várias formas:

- fluxogramas
- pseudocódigo

Programar

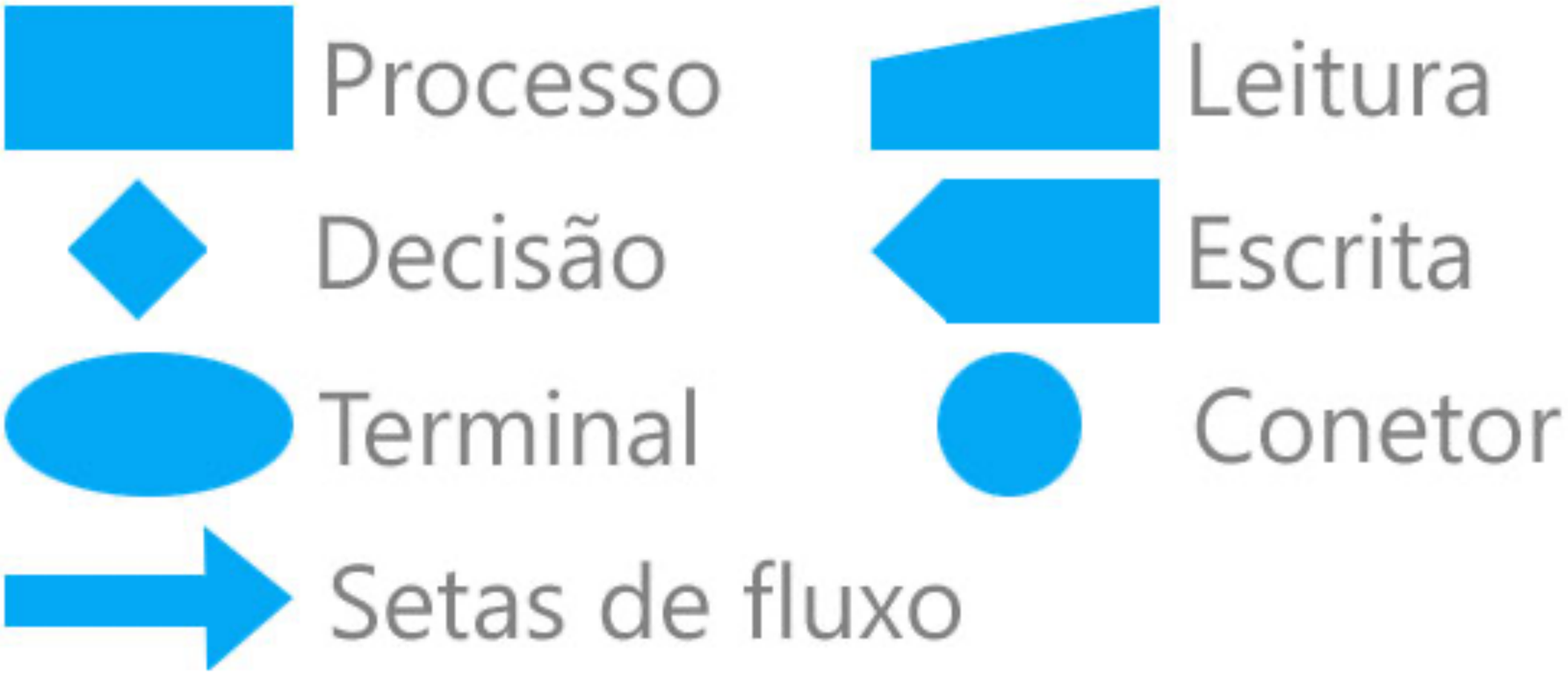
Fluxogramas

Um Fluxograma é uma forma gráfica de representar um algoritmo.

Através de símbolos, representamos o fluxo de um algoritmo.

Programar

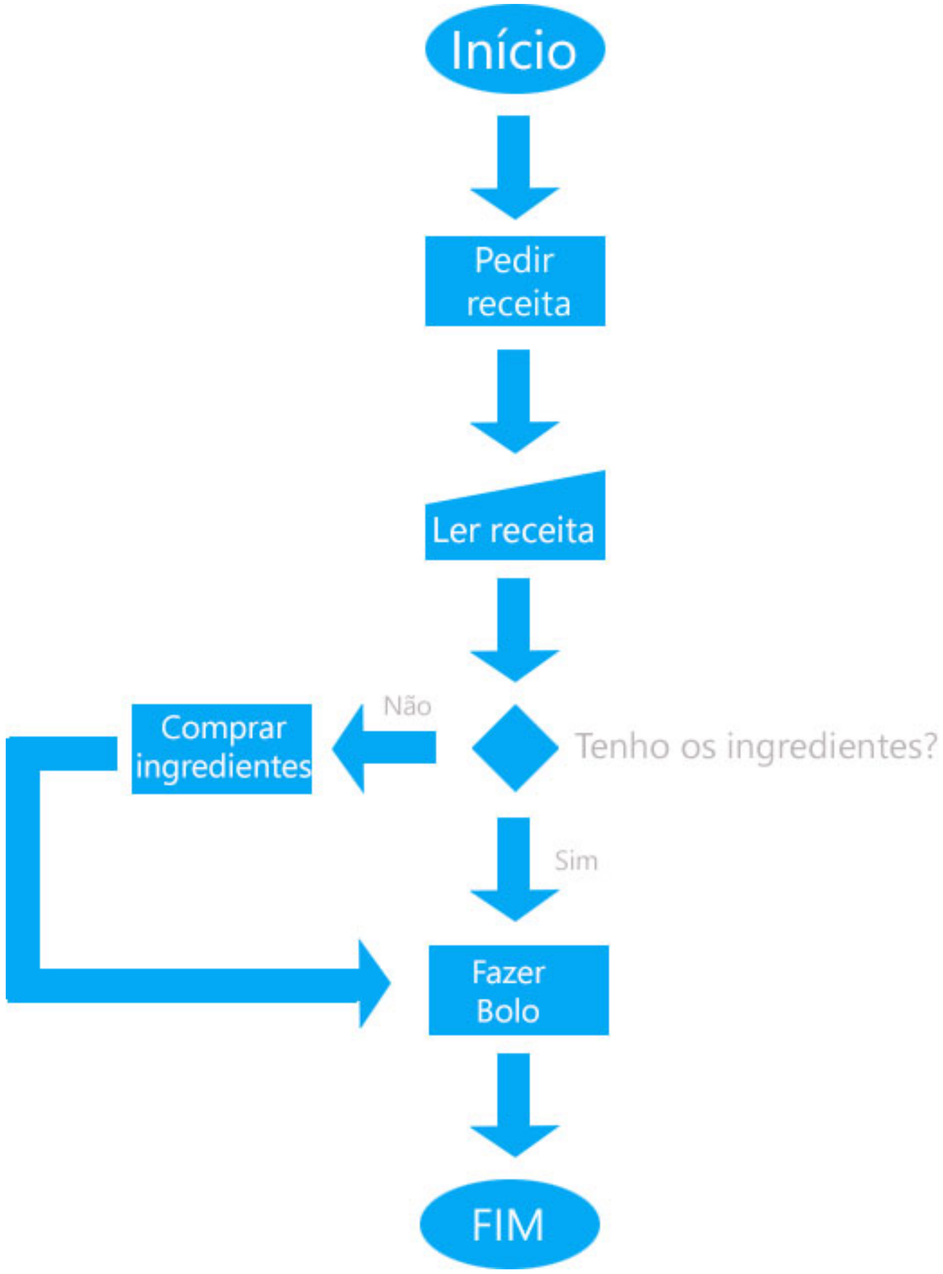
Fluxogramas



Programar

Fluxogramas

fluxograma do algoritmo anterior
a confeção do nosso bolo



Programar

Pseudocódigo

O Pseudocódigo é uma outra forma de representar algoritmos.

Esta já se aproxima mais de uma qualquer linguagem de programação.

Programar

Pseudocódigo

Voltando ao exemplo do bolo:

```
INICIO
  LER (Receita);
    SE tenhoIngredientes == verdade ENTÃO
      FazerBolo();
    SENÃO
      ComprarIngredientes();
      FazerBolo();
  FIM SE
FIM
```

Programar

Pseudocódigo

É muito fácil de entender

Expressões como LER, SE, SENÃO, ENTÃO ....

São muito parecidas com a nossa língua materna.

Programar

Constantes e Variáveis

Quando declaramos uma variável ou constante, estamos a reservar um **endereço da memória RAM.**

Um local onde poderemos guardar um valor.

Programar

Constantes

As constantes permite-nos armazenar valores **constantes** ou seja, depois de declararmos uma constante, **não podemos alterar o seu valor**.

```
constante EMPRESA = “Universidade do Minho"

EMPRESA = "A Minha Empresa” – – Situação de erro
```

O nome das constantes é, normalmente, escrito em letras **maiúsculas**.

Uma convenção para mais fácil para um terceiro compreender o código escrito.

Programar

Variáveis

Ao contrário das constantes, podemos alterar o valor das variáveis a qualquer momento da execução de um programa.

```
temaAtual = “Constantes"
```

```
temaAtual = "Variáveis"
```

Por exemplo nas variáveis seguimos o padrão **CamelCase**,
o *lowerCamelCase* onde iniciamos a primeira palavra com letra minúscula e as restantes capitalizadas.

Programar

Regras de nomeação

Quando damos um nome a uma variável ou a uma constante:

- Não pode começar com **números**;
- Não pode ser igual a uma **palavra reservada**;
- Não pode conter **espaços**;
- Não pode conter **caracteres especiais.**

Programar

Tipos de variáveis e constantes

os “*principais*” tipos de constantes e variáveis existentes são:

- *int* de “integer number” ou número inteiro: 5, -5, 22554, -515984512;
- *float* de “floating-point numer” ou número com ponto flutuante: 3.14059 (sim, este não é o pi), 1.58596548, -52.2358;
- *boolean* ou booleano: true, false;
- *string* ou conjunto de caracteres: “A minha frase”

Programar

Paradigmas de Programação

Antes de apresentar o ***Python***, vamos aprender o que realmente é uma linguagem de programação e conhecer os paradigmas de programação mais utilizados.

Programar

Paradigmas de Programação

Uma **linguagem de programação** é uma forma padronizada para comunicar instruções a um computador, através de diversas regras semânticas e sintáticas.

Programar

Paradigmas de Programação

Todas as linguagens de programação têm várias características específicas

Paradigmas de programação.

Paradigmas são modelos ou padrões

Paradigmas de programação são formas de estruturar o código.

Podem ser adotados mais do que um paradigma (**multi-paradigma).**

Programar

Paradigma imperativo

O paradigma imperativo concentra-se num **estado** (que são as variáveis) e **ações** (comandos) que **modelam** o estado.

Pode ser comparado ao modo modo imperativo da linguagem humana visto que é criado para ordenar a realização de ações.

Exemplos (de linguagens de programação imperativas) : C, Java, C#, Pascal.

Programar

Paradigma procedimental

O paradigma procedimental permite a reutilização de código sem o copiar através da utilização de funções e procedimentos. (Vamos falar disso no contexto do Python)

Com este paradigma, podemos “reciclar” código.

A maioria das linguagens de programação são procedimentais.

Programar

Paradigma estruturado

Uma linguagem de programação estruturada é aquela em que todos os programas podem ser reduzidos a três estruturas:

sequência,

decisão e

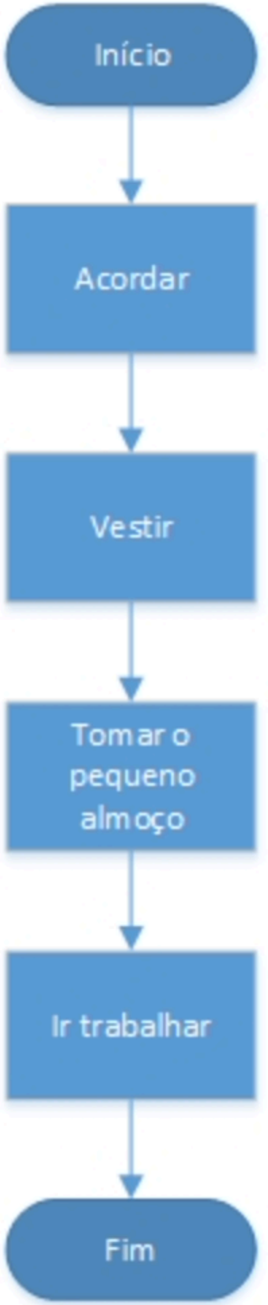
repetição (iteração).

Programar

Sequência

Nesta estrutura as tarefas são executadas de forma sequencial:

```
Acordar;
Vestir;
Tomar o pequeno-almoço;
Ir trabalhar;
```



Programar

Sequência

Em muitas linguagens de programação, os comandos/ações terminam com ponto e vírgula pois estas permitem que os comandos sejam colocados em linha da seguinte forma:

```
Acordar; Vestir; Tomar o pequeno-almoço; Ir trabalhar;
```

A utilização do ponto e vírgula é obrigatória quando existem várias instruções numa única linha.
Linguagens de programação como Java, C# e C obrigam ao uso do ponto e vírgula em todas as instruções independentemente se estão em linha ou não.

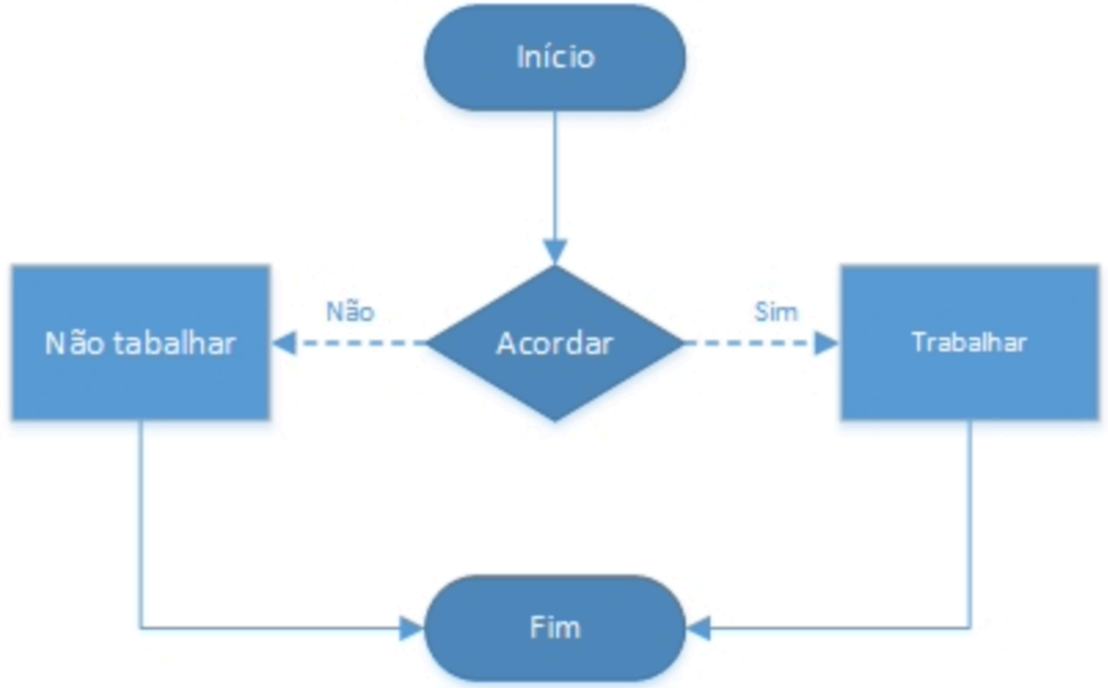
No Python vamos ver que não temos de nos preocupar com o “;”

Programar

Decisão

Neste tipo de estrutura, um determinado trecho de código é executado ou não dependendo do resultado de um teste lógico.

```
if "Acordar" then
  "Trabalhar"
else
  "Não trabalhar"
endif
```



Programar

Decisão

O pseudocódigo acima já não está em português e já se assemelha ao que irá visualizar nas linguagens de programação. Sempre que aparecerem termos em inglês no código relacionado com a sintaxe.

- **if** → se
- **then** → então
- **else** → caso contrário
- **endif (end if)** → fim do if

Programar

Decisão

Dói-me a cabeça. Se doer muito pouco, vou trabalhar. Se doer pouco, tomo um comprimido e vou trabalhar. Se doer muito, vou ao médico e falto ao trabalho.

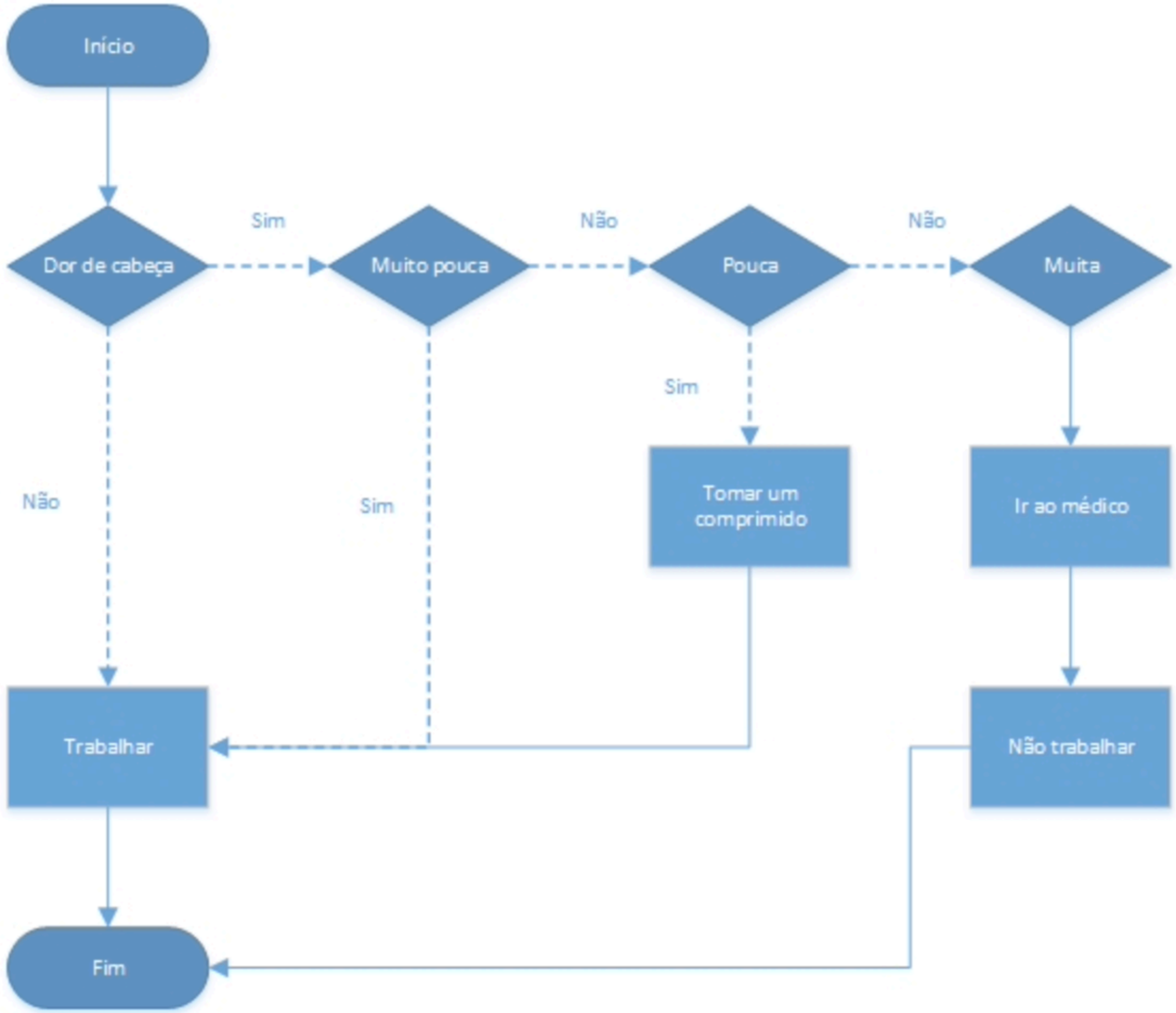
```
case "Dor de cabeça"
  when "muito pouco" then "trabalhar"
  when "pouco" then "tomar comprimido"; "trabalhar"
  when "muito" then "ir ao médico"; "não trabalhar"
```

Programar

Decisão

Este programa também escrito através de comandos **if/else** da seguinte forma:

```
if "Dor de cabeça"
  if "muito pouco" then
    "trabalhar";
  elseif "pouco" then
    "tomar comprimido";
    "trabalhar";
   else if "muito" then
    "ir ao médico";
    "não trabalhar";
   endif
endif
```



Programar

Decisão

- **case** → caso
- **when** → quando
- **else if** → caso contrário se

Programar

Iteração

Neste tipo de estrutura, também conhecido como repetição, um trecho de código será repetido um número finito de vezes dependendo do resultado de um teste lógico.

“não saio de casa enquanto não estiver vestido”

```
do{
  "não sair de casa";
} while ( "não estou vestido" )
```

pode ler da seguinte forma: fazer x enquanto y.

- **do** → fazer

Programar

Iteração

Enquanto estiver a Dormir, não me vou Vestir.

```
while ( "Durmo" )
  "Não me visto”;
```

Ou

Lavo os dentes 20 vezes.

```
for ( i = 0; i < 20; i++ )
 "Lavar os dentes"
```

Programar

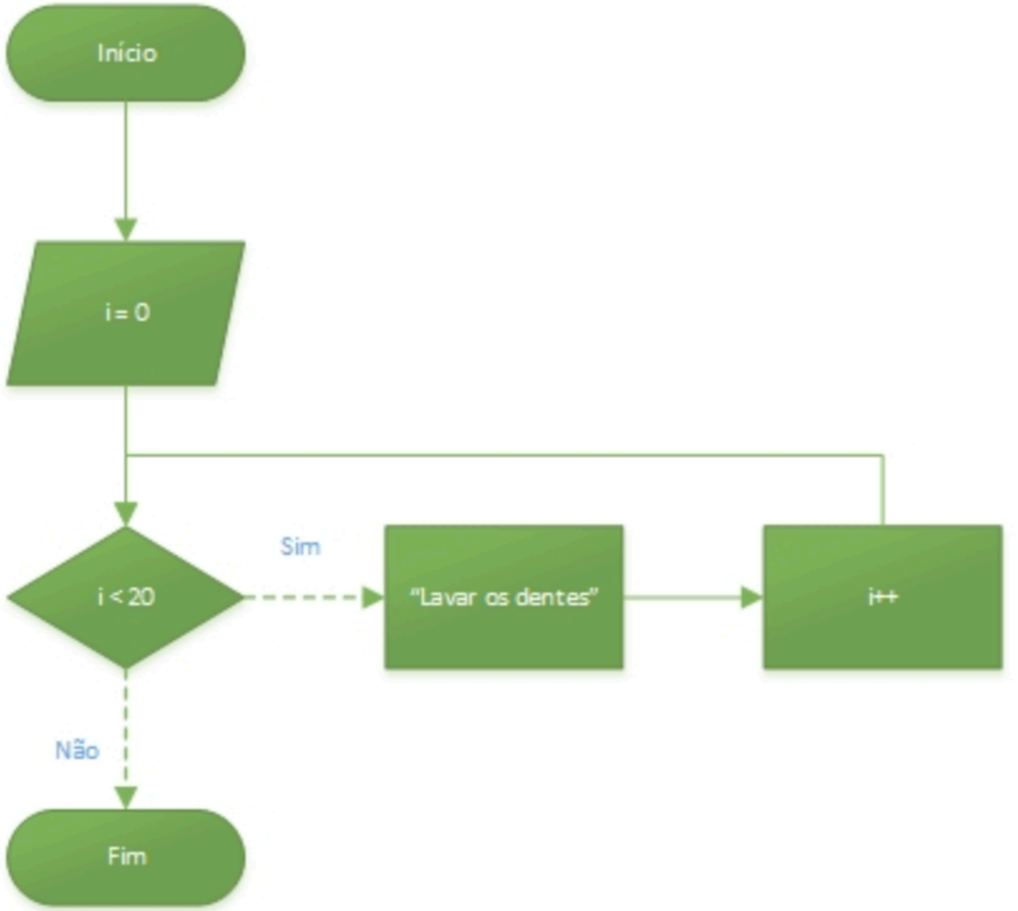
Iteração

- **for** → para/enquanto

Para cada dente, lavo-o muito bem.

```
for each "dente" in "boca"
  "Lavar muito bem"
```

- **each** → cada
- **in** → em



Programar

Paradigma Declarativo

O Paradigma Declarativo contrasta com o Imperativo pois é capaz de expressar a lógica sem descrever como o fluxo de comandos funciona,

Apenas diz ao computador **o que** fazer e não **como** fazer.

Um excelente **exemplo** de uma linguagem que utiliza este paradigma é **Prolog,** muito utilizado na área de inteligência artificial.

Programar

Paradigma Funcional

O Paradigma Funcional engloba todas as linguagens de programação que utilizam funções matemáticas.

Estas linguagens de programação são muito utilizadas no campo da matemática.

Exemplos: Matlab, Wolfram Language/Mathmatica/M, B.

Programar

Paradigma Orientado a Objetos

A Programação Orientada a Objetos permite a criação de **objetos** com base em **classes**. Estes objetos são instâncias dessas classes e possuem todos os atributos e funções presentes nas classes em questão.

Este paradigma é muito extenso e tem muita informação que mais à frente irá ser abordada. Atualmente, existem muitas linguagens que utilizam este paradigma.

Exemplos: Java, C++, C#, PHP.

Programar

Python

O Python é uma linguagem interpretada e orientada a objetos criada por um holandês chamado Guido van Rossum na CWI (Centro de Matemática e Informática)

Programar

Python

Python é uma linguagem de programação multiplataforma que permite desenvolver aplicações para jogos, desktops, web e dispositivos móveis.

Além disso, ela pode se comunicar com outras aplicações que foram desenvolvidas em outras linguagens como C, C++, Java e C#.

Programar

Conceito de Variável em Python

O **conceito de variável em Python** é representado sempre por um objeto (tudo é objeto), e toda variável é uma referência.

Na maioria das linguagens de programação, quando iniciamos uma variável e atribuímos um valor a ela, essas carregam valores que são alocados em memória, e quando alteramos os seus valores, estamos alterando o valor na memória também.

No Python as variáveis armazenam endereços de memória e não os valores.

Programar

Conceito de Variável em Python

```
1 x = [1,2,3]
2 y = x
3 x.append(4)
4 print(y)
```

Programar

Tipos de Dados em Python

```
a = float(22/5)
b = int(4.5)
c = int(3.9)
d = int(0xff563)
e = float(int(3.9))
f = int(float(3.9))
g = int(float(3))
h = round(3.9)
i = round(3)
j = int(round(3.9))
print(a,b,c,d,e,f,g,h,i,j)
```

Programar

Tipos de Dados em Python

```
1 | a = 42 #decimal
2 | b = 010 #octal
3 | c = 0xA #hexadecimal
```

```
1 | a = 'Isso é uma String com aspas Simples'
2 | b = "Isso é uma String com aspas Duplas"
3 | c = """Isso é Uma String com aspas Triplas"""
4 | print(a + "\n" + b + "\n" + c)
```

Programar

Tipos de Dados em Python

```
1 #coding: utf-8
2 meu_nome = "Sintaxes Python"
3 meu_nick = 'Devmedia'
4 print ("Nome: %s, Nick: %s" % (meu_nome.upper(), meu_nick))
5 print ("Meu nome comeca com a letra ", meu_nome[0])
6 print ("Meu nome comeca com a letra ", meu_nome[0].lower())
7 print ("Meu primeiro nome é ", meu_nome[0:7])
```

Programar

Tipos de Dados em Python

Formato do Símbolo	Conversão
%c	Caractere
%s	String convertida pelo método str() tem prioridade
%i	Sinal decimal inteiro
%d	Sinal decimal inteiro
%u	Decimal inteiro sem negativos (unsigned)
%o	Octal inteiro
%x	Hexadecimal Inteiro (letras em caixa baixa)
%X	Hexadecimal Inteiro (Letras em Caixa ALTA)
%e	Notação exponencial (Para letras minúsculas ‘e’)
%E	Notação exponencial (Para letras MAIÚSCULAS ‘E’)
%f	Ponto flutuante (Números reais)
%g	O mais curto de ‘f%’ e ‘%e’
%G	O mais curto de ‘f%’ e ‘%E’

Programar

Operadores

Aritméticos	Comparação	Lógicos
+	==	and
-	!=	or
*	>	not
/ ou // (parte inteira)	<	
%	>=	
+= -= *= /=	<=	
**	in in not	
is		

Programar

input e print

```
print("-----------------------------------")

x1 = float(input("abcissa do ponto A “))

print("O antecessor de",n,"é",n-1)

print(f”A abcissa do porto A={x1:.2f}”)
```

Programar

Operadores

```
1 import sys
2
3 nome = input("Digite seu nome: ")
4 idade = input("Digite sua idade: ")
5 print("Digite seu sexo: ")
6 sexo = sys.stdin.readline()
7
8 print("Nome:"+nome + "\n" + "Sexo: %s Idade: %s" %(sexo,idade))
```

Programar

IF, ELIF, ELSE no Python

```
1  # coding:utf-8
2  dedos = int(input("Você tem quantos anos? "))
3
4  if dedos == 18:
5    print("Você tem 18 anos")
6  elif dedos > 18:
7    print("Você tem mais de 18 anos")
8  else:
9    print("Você é menor de idade”)
```

Python

Python List

António Abelha

Python List

Uma lista é uma **estrutura de dados** composta por itens organizados de forma linear.

Podemos aceder a cada item a partir de um índice,

Índice que representa sua posição na coleção, e

Índice que se inicia em zero.

Python List

Em **Python**, uma **lista é representada como uma sequência de objetos** separados por vírgula e dentro de parênteses retos “[ ]”.

Uma lista vazia, por exemplo, pode ser representada por “[ ]” sem nenhum conteúdo.

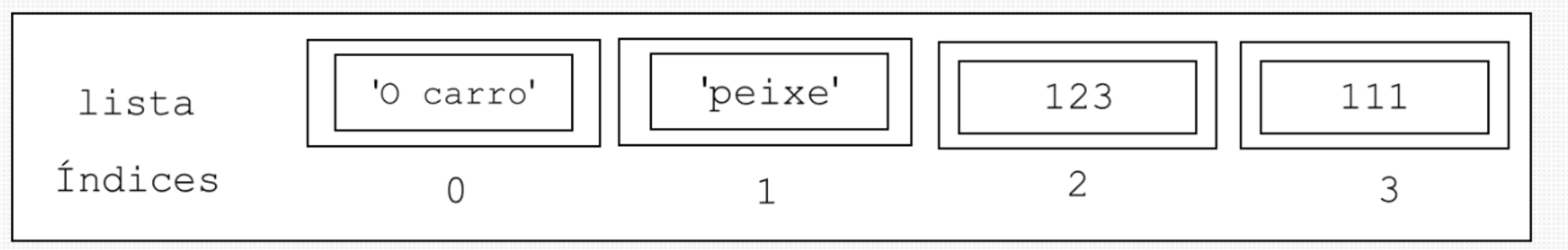
As possibilidades de declaração e atribuição de valores a uma lista são várias

Python List (criar uma lista)

```
>>> lista = []
>>> lista
[]
>>> lista = ['0 carro','peixe',123,111]
>>> lista
['0 carro', 'peixe', 123, 111]
>>> nova_lista = ['pedra',lista]
>>> nova_lista
['pedra', ['0 carro', 'peixe', 123, 111]]
```

Python List (Operadores)

Em **Python**, dispomos de vários métodos e operadores para manipular listas listas.
O primeiro e mais básico é o operador de acesso a cada item a partir dos índices.



Python List (Operadores)

Em **Python** as listas podem possuir diferentes tipos de elementos na sua composição.

```
>>>
>>> aluno = ['Mario', 19, 1.79]
>>> print(type(aluno))
<type 'list'>
>>> print(aluno)
['Mario', 19, 1.79]
>>>
```

Python List (Operadores)

```
>>> lista[0]
'0 carro'
>>> lista[2]
123
>>> nova_lista[0]
'pedra'
>>> nova_lista[1]
['0 carro azul', 'peixe', 123, 111]
>>> nova_lista[1][2]
123
```

Python List (Len)

O comprimento de uma lista, ou o número de itens que a compõem, pode ser obtido a partir da função *len().*

```
>>> len(nova_lista)
2
```

Python List (Concatenação e multiplicação)

```
>>> lista = ['0 carro','peixe',123,111]
>>> nova_lista = ['pedra',lista]
>>> lista+nova_lista
 ['0 carro', 'peixe', 123, 111, 'pedra', ['0 carro', 'peixe', 123, 111]]
>>> lista*3
 ['0 carro', 'peixe', 123, 111, '0 carro', 'peixe', 123, 111, '0 carro', 'peixe', 123, 111]
 >>>
```

Python List (Operador in)

```
>>> 'peixe' in lista
True
>>> 'gato' in lista
False
```

```
nomes = ['Pedro', 'João', 'Leticia']
for n in nomes:
    print(n)
```

Python List (Valores mínimos, máximos e soma)

```
>>> numeros = [14.55, 67, 89.88, 10, 21.5]
>>> min(numeros)
10
>>> max(numeros)
89.88
>>> sum(numeros)
202.93
```

Python List (Métodos)

append()

```
>>> livros = ['Java','SqlServer', 'Delphi','Python']
>>> livros.append('Android')
>>> livros
['Java', 'SqlServer', 'Delphi', 'Python', 'Android']
```

Python List (Métodos)

insert()

```
>>> livros = ['Java', 'SqlServer', 'Delphi', 'Python', 'Android']
>>> livros.insert(0,'Oracle')
>>> livros
['Oracle', 'Java', 'SqlServer', 'Delphi', 'Python', 'Android']
```

Python List (Métodos)

pop()

```
>>> livros
['Java', 'SqlServer', 'Delphi', 'Python', 'Android']
>>> livros.pop()
'Android'
>>> livros.pop(1)
'SqlServer'
>>> livros
['Java', 'Python', 'Delphi']
```

Python List (Métodos)

remove()

```
>>> livros
['Oracle', 'Java', 'SqlServer', 'Delphi', 'Python', 'Android', 'Oracle']
>>> livros.remove('Oracle')
>>> livros
['Java', 'SqlServer', 'Delphi', 'Python', 'Android', 'Oracle']
>>> livros.remove('Oracle')
>>> livros
['Java', 'SqlServer', 'Delphi', 'Python', 'Android']
>>> livros.remove('Oracle')
Traceback (most recent call last):
 File "<pyshell#34>", line 1, in <module>
   livros.remove('Oracle')
ValueError: list.remove(x): x not in list
```

Python List (Métodos - ordenação)

sort(); reverse()

```
>>> livros
['Java', 'SqlServer', 'Delphi', 'Python', 'Android']
>>> livros.reverse()
>>> livros
['Android', 'Python', 'Delphi', 'SqlServer', 'Java']
>>> livros.sort()
>>> livros
['Android', 'Delphi', 'Java', 'Python', 'SqlServer']
```

Python List (Métodos)

count()

```
>>> livros = ['Oracle', 'Java', 'SqlServer', 'Delphi', 'Python', 'Android', 'Oracle']
>>> livros.count('Python')
1
>>> livros.count('Oracle')
2
>>>
```

Python List (Exemplos)

```
>>> programadores = ['Vitor', 'Julio', 'Samuel', 'Carlos', 'Luisa']
>>> print(type(programadores))
<type 'list'>
>>> type(programadores)
<type 'list'>
>>> print(len(programadores))
5
>>> print(programadores[4])
Luisa
>>>
```

Python List (Exemplos)

```
>>> print(programadores)
['Vitor', 'Julio', 'Samuel', 'Carlos', 'Luisa']
>>> programadores[1] = 'Carolina'
>>> print(programadores)
['Vitor', 'Carolina', 'Samuel', 'Carlos', 'Luisa']
>>>
```

Python

Python Matrices

António Abelha

Python List (Exemplos)

Podemos usar listas para armazenar e manipular matrizes.

$$\begin{pmatrix} 2 & -3 & 4 \\ 0 & 7 & 5 \end{pmatrix}$$

Pode ser representada na lista

[[2,-3,4] , [0, 7,5]]

Python List

MATRIZ [linha][coluna]

MATRIZ [0] [0] = 2
MATRIZ [0] [1] = -3
MATRIZ [0] [2] = 4
MATRIZ [1] [0] = 0
MATRIZ [1] [1] = 7
MATRIZ [1] [2] = 5

Python List

Vamos escrever uma função que construa uma matriz 4x3 com valores iguais a zero e que retorne a matriz construída.

```
# Funcao para construir uma matriz 4x3 de 0

def constroiMatriz1():
        matriz = []
        for i in range(4):
                linha = []
                for j in range(3):
                        list.append(linha,0)
                matriz = matriz + [linha]
        return matriz

def constroiMatriz2():
        matriz = []
        for i in range(4):
                list.append(matriz,[0]*3)
        return matriz
~
```

```
execfile('./aula.py')
```

Python List

Vamos escrever uma função para multiplicar os elementos da diagonal principal de uma matriz por um valor k.

```
# Funcao para multiplicar os elementos da uma matriz K
# list, int -> list

def multidiagonal(matriz,k):
        for i in range(len(matriz)):
                matriz[i][i] *=k
return matriz
~
~
```

A função deve receber a matriz e k, e devolver a matriz resultante.

```
execfile('./aula2.py')
```

Python List

Escreva uma função que dadas duas matrizes A e B de mesmo tamanho, devolva a soma das duas matrizes.

```
# Funcao para somar duas matrizes C = A+B
# list, list -> list

def somaMatrizes(A,B):
        C=[]
        for i in range(len(A)):
                linha = [ ]
                for j in range(len(A[0])):
                        list.append(linha,A[i][j] + B[i][j])
                list.append(C,linha)
        return C
```

```
execfile('./aula3.py')
```

Python List

Escreva uma função que dada uma matriz retorne a linha de maior soma e a respectiva soma

```
# Funcao para apresentar a linha de maior soma e soma
# list -> list, int

def maiorLinha1(matriz):
        somas = [ ]
        for i in range(len(matriz)):
                soma = 0
                for j in range(len(matriz[0])):
                        soma += matriz[i][j]
                list.append(somas,soma)
        maior = max(somas)
        pos = list.index(somas,maior)
        return matriz[pos],maior

def maiorLinha2(matriz):
        somas = [ ]
        for i in range(len(matriz)):
                soma = sum(matriz[i])
                list.append(somas,soma)
        maior = max(somas)
        pos = list.index(somas,maior)
        return matriz[pos],maior
```

```
execfile('./aula4.py')
```

Python List

Escreva uma função para calcular o produto de duas matrizes M1 e M2.

A função deve verificar se as matrizes têm de tamanhos compatíveis para a multiplicação.

Devolver matriz resultado ou mensagem de erro.

-

1 * 1 + 2 * 4 + 3 * 7 | 1 *2 + 2*5 + 3*8

4 * 1 + 5 * 4 + 6 * 7 | 4 *2 + 5*5 + 6*8

30	36
66	81
102	126

As colunas de M na linha 1 * as linhas de N na coluna 1.
Depois, as mesmas colunas de M * as linhas de N na coluna 2

As colunas de M na linha 2 * as linhas de N na coluna 1.
Depois, as mesmas colunas de M * as linhas de N na coluna 2...