Linguagens de Programação

Aula 2

José Jasnau Caeiro
j.caeiro@ipbeja.pt

Instituto Politécnico de Beja

2024-2025

Objetivos da Aula

- fundamentos sobre linguagens de programação;
- introdução à linguagem de programação `Python`;
- tipos elementares em `Python`;
- estruturas de controlo em `Python`;
- funções em `Python`;
- regras de estilo em `Python`;

1. Fundamentos de Linguagens de Programação
2. Python

Linguagem de Programação

Definição

*Uma **linguagem de programação** é uma **linguagem formal** projetada para a comunicação de instruções a máquinas, em particular computadores, (Wikipedia, 2013).*

Definição

*Uma **linguagem de programação** é um formalismo artificial em que **algoritmos** podem ser exprimidos.*

Definição

*Uma **linguagem formal** é um conjunto de cadeias de carateres de símbolos que podem ser restringidos por regras que lhes são específicas.*

Linguagem Formal

Definição

*O **alfabeto** duma **linguagem formal** é o conjunto de símbolos e letras a partir do qual se geram as cadeias de carateres duma linguagem.*

Definição

*Uma **linguagem formal** é frequentemente definida a partir duma gramática formal: **gramáticas regulares** ou **gramáticas independentes do contexto**.*

Domínios de Aplicação

As **linguagens de programação** têm vários domínios de aplicação:

- aplicações científicas:
  - estruturas de dados simples: tabelas e matrizes;
  - estruturas de controlo do tipo ciclo e seleções;
  - cálculos de vírgula flutuante;
  - eficiência de cálculo;
  - a linguagem de programação **FORTRAN** é uma referência e quase inultrapassada.
- aplicações na área da gestão e da banca:
  - facilidade na geração de relatórios;
  - formas precisas de representação, cálculo e armazenamento de números decimais;
  - formas precisas de representação, e armazenamento de carateres;
  - o exemplo de referência é **COBOL**.

Domínios de Aplicação

- inteligência artificial
  - caraterizada pela predominância da computação com símbolos em vez da computação numérica;
  - preferência dada ao processamento com listas ligadas em vez de tabelas;
  - capacidade de criar e executar segmentos de código em tempo de execução;
  - as linguagens de referência são **LISP** que surge em 1959 com inspiração no *lambda calculus* e com um dialeto destinado ao ensino **Scheme**, o **PROLOG** como alternativa de programação designada por **programação lógica**:

Domínios de Aplicação

- programação de sistemas
  - linguagens de programação de elevada eficiência de execução;
  - propriedades de baixo-nível que permitem a comunicação e controlo de dispositivos de *hardware*;
  - os sistemas operativos **UNIX** e o seu derivado **OSX** e **IOS**; o **Linux** e seus derivados como o **Android**; o sistema **Windows** nas suas várias versões, encontram-se todos programados na maior parte com a linguagem de programação **C**

Linguagens de Programação Dinâmicas

- conhecidas por vezes como linguagens de *scripting* apesar desta interpretação ser demasiado restritiva;
- permitem a realização rápida de aplicações e com interação fácil com sistemas;
- em geral são interpretadas;
- têm mecanismos de ligação com outras linguagens de programação;
- exemplos são **sh e bash**, **awk**, **Perl**, **Tcl/Tk**, **Perl**, **Python**, **Ruby**, **PHP**.

Outros Exemplos

As linguagens de uso geral multi-paradigmáticas são exemplos a apontar pela sua importância:

- **C++** que junta ao universo das linguagens de programação de sistemas de **C** com as capacidades da orientação por objetos;
- **Java** que é uma evolução do **C++** a funcionar sobre uma máquina virtual e com um forte impacto na programação de sistemas e na *web*;
- **C#** que permite unificar conceitos de programação de sistemas, de linguagens de programação dinâmicas e outros paradigmas;
- **Objective C** que conhece uma grande divulgação devido aos produtos da **Apple**.

Linguagens de Programação Emergentes

Exemplos

Novas linguagens de programação que conhecem alguma adoção.

Scala uma linguagem de programação funcional, https://scala-lang.org/

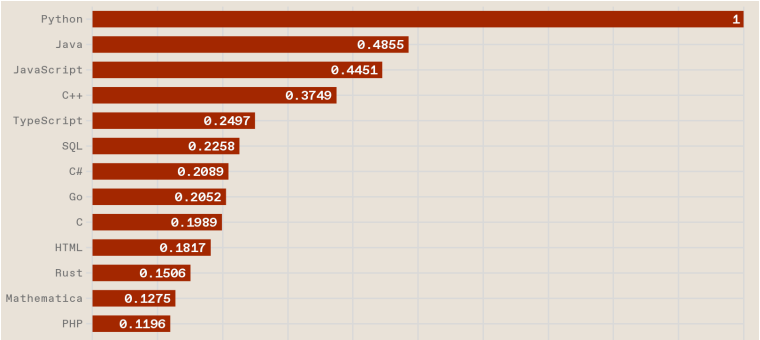
Rust uma linguagem de programação de sistemas multi-paradigma, https://www.rust-lang.org/en-US/

Kotlin uma alternativa a Java para a programação em Android, https://kotlinlang.org/

Julia uma alternativa a Matlab para a programação científica, https://julialang.org/

Clojure uma versão de Lisp para a JVM, https://clojure.org/

Índice Spectrum IEEE



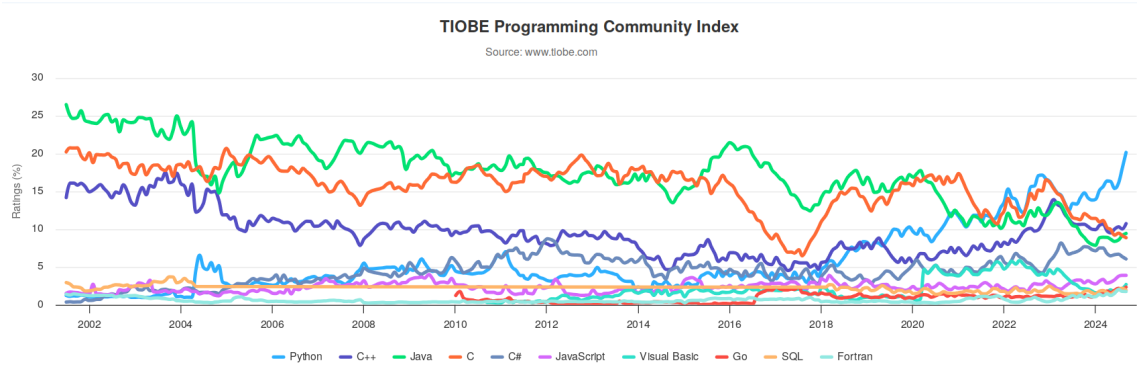
https://spectrum.ieee.org/the-top-programming-languages-2024

Índice TIOBE

Sep 2024	Sep 2023	Change	Programming Language	Ratings	Change
1	**1**		**Python**	**20.17%**	**+6.01%**
2	3	↑	C++	10.75%	+0.09%
3	4	↑	Java	9.45%	-0.04%
4	2	↓	C	8.89%	-2.38%
5	5		C#	6.08%	-1.22%
6	6		JavaScript	3.92%	+0.62%
7	7		Visual Basic	2.70%	+0.48%
8	12	⇈	Go	2.35%	+1.16%
9	10	↑	SQL	1.94%	+0.50%
10	11	↑	Fortran	1.78%	+0.49%
11	15	⇈	Delphi/Object Pascal	1.77%	+0.75%
12	13	↑	MATLAB	1.47%	+0.28%
13	8	⇊	PHP	1.46%	-0.09%
14	17	↑	Rust	1.32%	+0.35%

https://www.tiobe.com/tiobe-index/

Gráfico TIOBE



https://www.tiobe.com/tiobe-index/

PYPL PopularitY of Programming Language index

Worldwide, Sept 2024 :

Rank	Change	Language	Share	1-year trend
1		Python	29.66 %	+1.6 %
2		Java	15.64 %	-0.2 %
3		JavaScript	8.3 %	-1.0 %
4		C#	6.64 %	-0.1 %
5		C/C++	6.46 %	-0.2 %
6	↑	R	4.66 %	+0.2 %
7	↓	PHP	4.35 %	-0.5 %
8		TypeScript	2.96 %	-0.0 %
9		Swift	2.69 %	+0.0 %
10	↑	Rust	2.65 %	+0.6 %
11	↓	Objective-C	2.45 %	+0.2 %
12		Go	2.08 %	+0.2 %
13		Kotlin	1.95 %	+0.2 %
14		Matlab	1.45 %	-0.1 %
15		Ruby	0.99 %	-0.1 %

http://pypl.github.io/PYPL.html

Síntaxe

Definição

A síntaxe duma linguagem de programação é o conjunto de regras que define as combinações dos símbolos do que se considera ser um documento corretamente estruturado ou fragmento nessa linguagem.

- o aspeto superficial de uma linguagem de programação é conhecido como síntaxe;
- a maioria das linguagens de programação são puramente textuais, usando sequências de texto incluindo palavras, números e pontuação;
- outras são mais gráficas e usam relações visuais entre símbolos para a especificação de um programa.

Síntaxe

É comum encontrar linguagens de programação representadas com a notação **BNF**

Backus-Naur Form por exemplo a própria representação **BNF**:

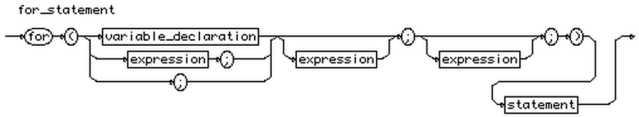
```
<syntax>         ::= <rule> | <rule> <syntax>
<rule>           ::= <opt-whitespace> "<" <rule-name> ">" <opt-whitespace> "::=" <opt-whitespace> <expression> <line-end>
<opt-whitespace> ::= " " <opt-whitespace> | ""
<expression>     ::= <list> | <list> "|" <expression>
<line-end>       ::= <opt-whitespace> <EOL> | <line-end> <line-end>
<list>           ::= <term> | <term> <opt-whitespace> <list>
<term>           ::= <literal> | "<" <rule-name> ">"
<literal>        ::= '"' <text> '"' | "'" <text> "'"
```

Síntaxe

Outro exemplo, o ciclo **for** em **Java BNF**:

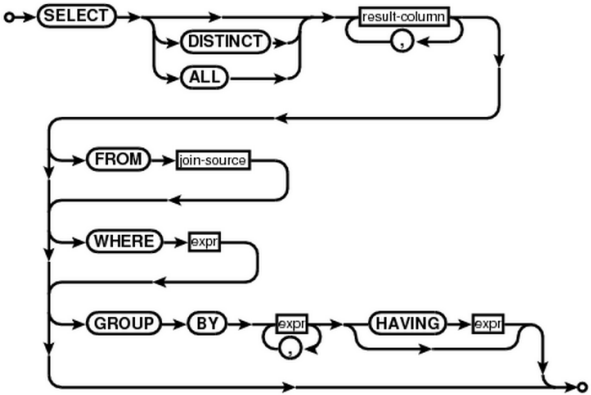
for_statement

```
for_statement
      ::=
      "for" "(" ( variable_declaration | ( expression ";" ) | ";" )
      [ expression ] ";"
      [ expression ] ";"
      ")" statement
```



http://cui.unige.ch/isi/bnf/JAVA/for_statement.html

Síntaxe



Ou, o **SELECT** em **SQLite BNF**:
http://www.sqlite.org/lang_select.html

Semântica

Definição

*A **semântica** é o campo de conhecimento relacionado com o estudo matemático rigoroso do significado de linguagens de programação. Fá-lo por análise do significado de cadeias de carateres sintaticamente corretas definidas por uma linguagem de programação, exibindo a computação envolvida. A **semântica** descreve os procedimentos que um computador segue quando executa um programa numa linguagem específica. (Wikipedia, 2013)*

Semântica

Java

Em **Java**, uma expressão **if** é executada:

- avaliando a expressão de **teste**;
- se é **verdadeira** executa a expressão após o **teste**;
- se é **falsa** salta a expressão após o teste.

C

Em **C**, uma expressão **if** é executada:

- avaliando a expressão de **teste**;
- se é > 0 executa a expressão após o **teste**;
- se é $<= 0$ salta a expressão após o teste.

Sistema de Tipos

Definição

*Um **sistema de tipos** é uma coleção de regras que atribuem uma propriedade designada por **tipo** a várias construções de um programa*

- *variáveis;*
- *expressões;*
- *funções;*
- *módulos;*

O tipo representa uma descrição dos possíveis valores que podem ser assumidos pela construção.

O objetivo dos **tipos** é a redução de erros de programação. Os tipos das construções podem ser verificados em tempo de compilação ou tempo de execução.

Paradigmas de Programação

As linguagens de programação suportam paradigmas de programação. Há 4 paradigmas principais:

imperativo a computação é descrita em termos de expressões que modificam o **estado** de um programa;

funcional a computação é tratada como uma avaliação sucessiva de funções matemáticas em que se evita o **estado** e dados **mutáveis**;

orientado por objetos em que se representam conceitos como **objetos** que possuem atributos e procedimentos associados conhecidos por **métodos**;

lógico é baseado em lógica de primeira ordem, e é puramente declarativo.

Paradigmas de Programação

Paradigm	Description	Main characteristics	Related paradigm(s)	Critics	Examples
Imperative	Computation as statements that *directly* change a program state (datafields)	Direct assignments, common data structures, global variables		Edsger W. Dijkstra, Michael A. Jackson	C, C++, Java, PHP, Python
Structured	A style of imperative programming with more logical program structure	Structograms, indentation, either no, or limited use of, goto statements	Imperative		C, C++, Java
Procedural	Derived from structured programming, based on the concept of modular programming or the *procedure call*	Local variables, sequence, selection, iteration, and modularization	Structured, imperative		C, C++, Lisp, PHP, Python
Functional	Treats computation as the evaluation of mathematical functions avoiding state and mutable data	Lambda calculus, compositionality, formula, recursion, referential transparency, no side effects			Erlang, Haskell, Lisp, Clojure, Scala, F#
Event-driven including time driven	Program flow is determined mainly by events, such as mouse clicks or interrupts including timer	Main loop, event handlers, asynchronous processes	Procedural, dataflow		ActionScript
Object-oriented	Treats datafields as *objects* manipulated through pre-defined methods only	Objects, methods, message passing, information hiding, data abstraction, encapsulation, polymorphism, inheritance, serialization-marshalling		See here and[1][2]	C++, C#, Java, PHP, Python, Ruby, Scala
Declarative	Defines computation logic without defining its detailed control flow	4GLs, spreadsheets, report program generators			SQL, regular expressions, CSS
Automata-based programming	Treats programs as a model of a finite state machine or any other formal automata	State enumeration, control variable, state changes, isomorphism, state transition table	Imperative, event-driven		
Paradigm	**Description**	**Main characteristics**	**Related paradigm(s)**	**Critics?**	**Examples**

Paradigmas de Programação

Paradigm	Related paradigm(s)	Critics	Examples
Imperative		Edsger W. Dijkstra, Michael A. Jackson	C, C++, Java, PHP, Python
Structured	Imperative		C, C++, Java
Procedural	Structured, imperative		C, C++, Lisp, PHP, Python
Functional			Erlang, Haskell, Lisp, Clojure, Scala, F#
Event-driven including time driven	Procedural, dataflow		ActionScript
Object-oriented		See here and[1][2]	C++, C#, Java, PHP, Python, Ruby, Scala
Declarative			SQL, regular expressions, CSS
Automata-based programming	Imperative, event-driven		
Paradigm	**Related paradigm(s)**	**Critics?**	**Examples**

Introdução ao Python

- linguagem de programação de uso geral de nível elevado;
- enfatiza a clareza na leitura do código;
- multiparadigmática: suporta programação imperativa; funcional e orientada por objetos;
- sistema de tipos dinâmico, com forte verificação;
- linguagem de programação dinâmica;
- livre e com código fonte aberto e baseada em desenvolvimento pela comunidade;
- criada em 1991 e mantida pela organização sem fins lucrativos *Python Software Foundation*;
- local na Internet da versão principal http://www.python.org/.

Palavras Reservadas

and	as	assert	break	class
continue	def	del	elif	else
except	exec (func. in 3.x)	False (3.x)	finally	for
from	global	if	import	in
is	lambda	None (3.x)	nonlocal (3.x)	not
or	print (func. in 3.x)	pass	raise	return
True (3.x)	try	while	with	yield

Indentação

Usa espaços em branco para indentar e os blocos são exprimidos pela indentação *off-side rule*:

```
1  def␣par(a):
2  ␣␣␣␣if␣a␣%␣2␣==␣0:
3  ␣␣␣␣␣␣␣␣#␣se␣o␣resto␣da␣divisão␣por␣2␣for␣0␣é␣par
4  ␣␣␣␣␣␣␣␣print("É␣par!")
5  ␣␣␣␣␣␣␣␣return␣True
6  ␣␣␣␣else:
7  ␣␣␣␣␣␣␣␣#␣caso␣contrário␣é␣ímpar
8  ␣␣␣␣␣␣␣␣print("É␣ímpar!")
9  ␣␣␣␣␣␣␣␣return␣False
```

Sistema de Tipos

- inteiros `int`;
- vírgula flutuante `float`, tipicamente com 53 bits de precisão;
- números complexos `complex`;
- aritmética de precisão arbitrária com a biblioteca `mpmath` https://code.google.com/p/mpmath/;
- números decimais com o módulo `decimal`;
- cadeias de carateres imutáveis;
- listas e túplos;
- dicionários;
- conjuntos.

Tipos numéricos

```
1  import decimal
2  import mpmath
3  a = 99      # int
4  print(float('inf')) # maximo valor numérico
5  u = 99.76   # float
6  v = 2 + 2j # complex
7  z = u + v   # soma de um complex com float
8  print(z, z.real, z.imag) # resultado no ecrã
9  # um decimal
10 b = decimal.Decimal(4.42347529760276257652678257696)
11 w = mpmath.mpf(4.23331) # precisão arbitrária
```

Tipos Sequenciais

```
1  s1 = "The 'Portuguese' Inquisition"  # uma string
2  s2 = "tRoiKa"                        # mais uma string
3  s3 = s1 + ' ' + s2                   # concatenação
4  l1 = [1, 2, "lista", [3, 4, 5.33]]   # uma lista
5  a = l1[0] + l1[1] + l1[3][1]         # soma de valores
6  t1 = (1, 2, "tuplo")                 # os túplos são
7  print(t1[1])                         # imutáveis
```

Estruturas de Controlo - Seleções

```
1  if 1 > 0:
2      print("OK")
3  if 1.2 == 33:
4      print("NO")
5  else:
6      print("OK")
```

```
7  expr1 = True or False
8  expr2 = True and True
9  if expr1:
10     if expr2:
11         print("OK")
12     else:
13         print("NOT")
14 else:
15     print("NOT")
```

Estruturas de Controlo - Ciclo for

```
1  for i in range(0, 10): # vai de 0 ate 9 inclusivé
2      print(i * i)
3  l1 = range(20, 30)      # lista de números
4  print()
5  soma = 0
6  for x in l1:
7      soma += x               # soma dos números de l1
8  for k in range(20, 0, -1): # 20 downto 1 inclusivé
9      if k > 10:
10         print(k)
11     else:
12         print(2 * k)
```

Estruturas de Controlo - Ciclo while

```
1  l1 = [1, 2, 3, 4, 5, 6, 7, 8, 9]
2  l2 = []
3  k = len(l1)
4  while k: # 0 é considerado como sendo False
5      if k % 2:
6          l2.append( k )
7      else:
8          print(k, " par")
9      k -= 1
10 print(l2) # lista de ímpares
```

Estruturas de Controlo - Ciclo `while`

```
a = 0
k = 20
while True:
    a += 1
    if a > k:
        break
    if a % 2:
        continue
    print("valor de a", a)
    pass
```

```
x = 1
while x < 10:
    print(x)
    x += 1
else:
    print("Finito")
```

Funções

```
1  def fsoma(a, b): # declaração da função
2      """
3      fsoma - soma de 2 valores
4      a - um inteiro
5      b - outro inteiro
6      """
7      if isinstance(a, int) and isinstance(b, int):
8          soma = a + b
9      else:
10         print("a ou b não são inteiros e a soma será nula")
11         soma = 0
12     return soma
13 y1 = fsoma( 2, 3 ), y2 = fsoma( 4.3, "League of Legends" )
```

Funções

```
1  def frec(l1, x, n):
2      '''
3      frec - devolve uma lista com n elementos x
4      x    - o elemento a repetir
5      n    - numero de repeticoes
6      '''
7      if not n:
8          return l1
9      else:
10         return [x] + frec(l1, x, n-1)
11 l2 = frec([], 'Bu ', 5)
12 print(l2)
```

Funções

```
1  def func1(a, b, c=True):
2      '''
3      func1 - função com parâmetros por defeito
4      '''
5      if c:
6          return a + b
7      else:
8          return a - b
9  print(func1(2, 3))
10 print(func1(4, 5, False))
```

Regras de Estilo

- É importante ter em mente que o código deve ser de leitura fácil e escrito de modo consistente.
- No local da Internet `http://www.python.org/dev/peps/pep-0008/`, encontra-se um conjunto de convenções que ajudam na escrita de código consistente.
- A consistência não deve conduzir à ilegibilidade...

Leitura

Os alunos devem:

- realizar o tutorial de Python https://docs.python.org/3/tutorial/index.html;
- ler a documentação entregue pelo docente com *granum salis*;
- e ler os livros que entenderem... Há centenas!
- ChatGPT, Youtube!!!, Open CourseWare, EdX, *etc.* ...