

A linguagem de programação Python

Carlos Tavares

September 16, 2024

Linguagens de Programação

História das Linguagens de Programação

Uma linguagem de programação é essencialmente uma forma de descrever programas de uma forma que eles possam ser utilizados num computador.

Ano	Linguagem / Evento
1950s	Assembly, Fortran, Lisp
1960s	COBOL, BASIC, ALGOL
1970s	C, Pascal
1980s	C++, Perl
1990s	Python, Java, JavaScript
2000s	C#, Ruby, Go
2010s	Swift, Rust, Kotlin

Table 1: História das Linguagens de Programação

Linguagens de programação i

Qualquer linguagem de programação que tenha as seguintes construções é uma linguagem universal:

- **Intruções** com um conjunto completo de instruções aritméticas;
- **Variáveis**;
- **Aceitar** sequências de instruções
- **Operações** do tipo **Se ... então... senão**;
- **Ciclos**, operações do tipo **repita até**.

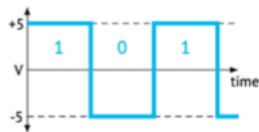
Existem milhares de linguagens de programação com diferentes funcionalidades e paradigmas.

Como é que um computador é universal? Qual é a linguagem "primordial"?

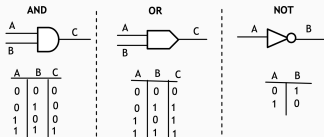
Bit

Representamos um bit com voltagem

$$\{0,1\} \equiv$$



Portas Lógicas

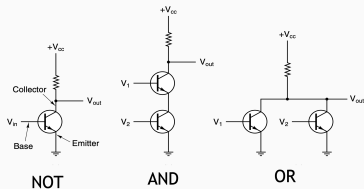


Este conjunto de portas é universal!

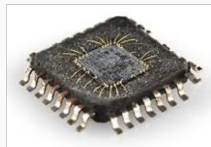
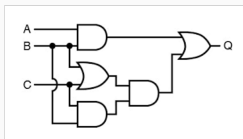
Como implementar portas lógicas utilizando apenas voltagens? É necessário algo como um transístor (provoca interrupção de corrente mediante uma certa voltagem de input)



Implementação de portas lógicas utilizando transístores



Com milhões de portas lógicas e transístores é possível ter um processador; algo que é capaz de interpretar instruções:



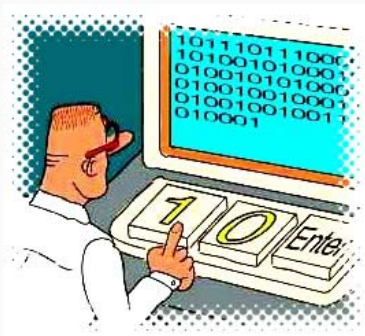
E com um processador mais memória (algo capaz de memorizar resultados) temos um computador universal se:

- O processador for capaz de executar somas, subtrações, multiplicações e divisões;
- For capaz de aceder à memória, tanto para dados como programas;
- **For capaz de realizar saltos nas instruções baseados em dados.**

Exemplo de uma instrução em código máquina:

001000 00001 00010 0000000101011110 := R1 = R2 + R3

Ultimate programmer:



Para resolver questões de portabilidade e abstração foi introduzido o Assembly nos anos 50.

Instrução	Descrição	Operação
ADD R1, R2, R3	Soma o conteúdo de R2 e R3, armazena em R1	$R1 = R2 + R3$
MOV R13, R14	Move o valor de R14 para R13	$R13 = R14$
JNZ R1, LABEL	Desvia a execução do programa para a instrução localizada no rótulo LABEL se R1 for o	LABEL Início de um bloco de código ou instrução

Table 2: Conjunto de Instruções de Assembly

Exemplo: **ADD R1, R2, 50; MOV R1, R2;**

Portabilidade

Programa » Assemblador » Código máquina

Existe imenso software importante e influente que foi exclusivamente desenvolvido em Assembly!

No entanto, o Assembly apresenta ainda problemas de portabilidade e abstração. O Assembly é uma linguagem baixo nível

Solução foi inventada a linguagem C

Example:

```
#include <stdio.h>
```

```
int main() {  
    printf ("Hello_world");  
}
```

C é uma linguagem alto-nível, com uma sintaxe simples e compacta que permite a estruturação dos programas.

Portability

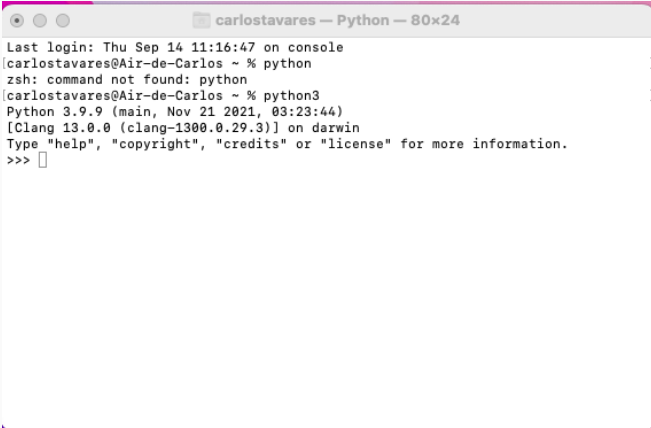
Programa » compilador de C » Fichiero executável » Máquina

Foram sendo inventadas outras linguagens cada vez mais poderosas e mais longe do código máquina

Uma delas foi o Python que é uma linguagem interpretada, alto-nível com grande grau de abstração que permite programação direcionado ao objecto, programação funcional, etc.

Programa » Interpretador » Máquina

O Python é uma linguagem interpretada!



```
Last login: Thu Sep 14 11:16:47 on console
carlostavares@Air-de-Carlos ~ % python
zsh: command not found: python
carlostavares@Air-de-Carlos ~ % python3
Python 3.9.9 (main, Nov 21 2021, 03:23:44)
[Clang 13.0.0 (clang-1300.0.29.3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

Python - Sintaxe e expressões aritméticas

A sintaxe de uma linguagem é a sua gramática, i.e. o conjunto de regras que garantem que a expressão está bem formada.

Comentários

Comentários em python começam com #

» **# This is a comment**

Palavras reservadas

Existe um conjunto grande de palavras reservadas no python que não devem ser utilizadas para outro fim do que aquele a que estão destinadas

<https://docs.python.org/2.5/ref/keywords.html>

Texto (string)

Texto (string) é um tipo de dados muito importante no Python. Pode ser definido entre "" ou ". Por exemplo:

```
>> "isto é texto" ou >> 'isto tambem é texto'
```


Expressões e números

Expressões aritméticas i

O Python permite expressões aritméticas, envolvendo números e chamada a funções

<expression> ::= <term> | <expression> <add_op> <term>

<term> ::= <factor> | <term> <mul_op> <factor>

<factor> ::= <number> | <function_call> | "(" <expression> ")"

<number> ::= <number>

<add_op> ::= "+" | "-"

<mul_op> ::= "*" | "/" | "//" | "%"

Há três tipos de números no Python

- Inteiros;
- Vírgula flutuante;
- Complexos;

Inteiros - O conjunto \mathbb{Z}

`<integer> ::= <decimal> | <binary> | <octal> | <hexadecimal>`

`<decimal> ::= [0-9][0-9_]*`

`<binary> ::= "ob" [0-1][0-1_]*`

`<octal> ::= "oo" [0-7][0-7_]*`

`<hexadecimal> ::= "ox" [0-9a-fA-F][0-9a-fA-F_]*`

Números inteiros ii

Operações:

Adição (+)	$5 + 3$	Soma dois inteiros. O resultado é a soma dos dois números. Exemplo: $5 + 3$ resulta em 8.
Subtração (−)	$5 - 3$	Subtrai o segundo número do primeiro. Exemplo: $5 - 3$ resulta em 2.
Multiplicação (*)	$5 * 3$	Multiplica dois inteiros. Exemplo: $5 * 3$ resulta em 15.
Divisão (/)	$5 / 3$	Divide o primeiro número pelo segundo. Resulta em um número de ponto flutuante. Exemplo: $5 / 3$ resulta em 1.6667.
Divisão Inteira (//)	$5 // 3$	Realiza a divisão inteira, retornando o quociente sem a parte decimal (arredonda para baixo). Exemplo: $5 // 3$ resulta em 1.
Módulo (Resto) (%)	$5 \% 3$	Retorna o resto da divisão do primeiro número pelo segundo. Exemplo: $5 \% 3$ resulta em 2.
Exponenciação (**)	$5 ** 3$	Eleva o primeiro número à potência do segundo. Exemplo: $5 ** 3$ resulta em 125.

Exemplo de divisão inteira

» `5//2`

2

» `5 % 2`

1

O Python suporta números inteiros arbitrariamente grandes de forma implícita

Números inteiros iv

O Python permite gerir números de outras bases que não a base 10. Podem ser representados por prefixos *ob* (base binária), *oo* (base octal), *ox* (base hexadecimal). O que significa tudo isto?

Um número na base 10 (base "natural"), pode ser entendido como:

$$N_{10} = \sum_{i=0}^{D-1} d_i \times 10^i \text{ with } d_i \in \{0, \dots, 9\} \quad (1)$$

Por exemplo:

$$985 = 5 \times 10^0 + 8 \times 10^1 + 9 \times 10^2$$

Podemos interpretar números nas outras bases da mesma maneira!

Base 2:

$$N_{10} = \sum_{i=0}^{D-1} d_i \times 2^i \text{ with } d_i \in \{0, 1\}$$

Por exemplo, o número 1001 em binário corresponde a 9 em decimal:

$$1 * 2^0 + 0 * 2^1 + 0 * 2^2 + 1 * 2^3 = 9_{10}$$

Base 8

$$N_{10} = \sum_{i=0}^{D-1} d_i \times 8^i \text{ with } d_i \in \{0, \dots, 7\}$$

Números inteiros vi

Por exemplo, o número 375 em octal corresponde a 288 em decimal

$$5 * 8^0 + 7 * 8^1 + 3 * 8^2 = 40 + 56 + 192 = 288_{10}$$

Base 16:

$$N_{10} = \sum_{x=0}^{D-1} d_i \times 16^x \text{ with } d_i \in \{0, \dots, 9, A, B, C, D, E, F\}$$

Por exemplo o número 0xAA corresponde a 170 em decimal

$$A * 16^0 + A * 16^1 = 10 * 1 + 10 * 16 = 170_{10}$$

Números inteiros vii

O Python faz operações entre números de diferentes bases de forma transparente e apresenta o resultado em decimal

É possível traduzir um número em decima para outras bases:
hex (x) (decimal)

oct (x) (octal)

bin (x) (binário)

O resultado destas funções é **texto**!

Porém, podemos traduzir texto para decimal com `int (x, base)`

Floating point numbers i

Um número em vírgula flutuante é um número com parte decimal e.g. 13.45. Pode-se representar números em notação científica

O nome deve-se à sua representação interna em vírgula flutuante de acordo com a norma IEEE754.

As operações funcionam como nos números inteiros.

Não há números arbitrariamente grandes nem pequenos de vírgula flutuante, há limite superior e inferior: (max: $1.7976931348623157 \times 10^{308}$, min: 5.0×10^{-324})

Números complexos i

Um número complexo é um número com parte real e imaginária, representado por $a + bj$.

Exemplo:

» $3 + 4j$

As operações funcionam como é esperado para números complexos

» $(2 + 3j)(4 + 5j)$

$(-7 + 22j)$

» $(2 + 3j) + (4 + 5j)$

$(6 + 8j)$

Partes reais e imaginárias podem ser obtidas com

» $z.\text{real}$ # Real part

» `z.imag` # Imaginary part

Pode-se converter entre números de diferentes tipos

int (x) onde x é um texto ou um número de vírgula flutuante

» **int** ("34567")

» **int** (12.3) # Only the integer part is returned: 12

float (x) onde x é um inteiro ou um texto

» **float** (12)

» **float** ("12.23")

» **float** ("2.04E23")

complex (x) onde x é um inteiro ou **complex (x, y)**

Exemplo:

» **complex (12)** # 12.0 + 0.0j

» **complex (12, 13)** # 12.0 + 13.0j

» **complex ("12+13j")**

str (x) onde x é um número de qualquer tipo

» **str (12)** # "12"

Retorna o absoluto de um número **abs (x)**

» `abs (-5)` = 5

Arredonda um número a um certo número de casas decimais

round (number, digits)

» `round (2.5678, 2)` # Result: 2.57

Eleva um numero x a uma potência y módulo z **pow (x, y, [z])**

» `pow (3, 4, 5)` # will yield 1

Booleanos e operações

Operadores de comparação

Os Booleanos são tipo de valores que só têm dois valores possíveis: verdadeiro (True) e falso (False). Têm muitas aplicações particularmente em instruções de controlo e ciclos e são originados por operadores de comparação e operadores lógicos

Operadores de comparação: (`==`, `!=`, `<`, `>`, `<>`, `<=`, `>=`)

Operador de igualdade: **expression == expression**

» `5 == 5`

Operador "menor que": **expression < expression**

» `5 < 6`

Greater "maior que": **expression > expression**

» `6 > 5`

Operadores lógicos

O Python tem também conectivos lógicos, da lógica Booleana:

Operador **not**: **not** expression

» not 5 == 10

False origina **True**

True origina **False**

Operador **and**: expression **and** expression

» $5+7 == 12$ **and** $5+6 = 10$

False and **False** origina **False**

True and **False** origina **False**

False and **True** origina **False**

True and **True** origina **True**

Operador **or**: expression **or** expression

» `5+2 == 7 or 6+1 == 9`

False or **False** origina **False**

True or **False** origina **True**

False or **True** origina **True**

True or **True** origina **True**

Variáveis, sequências e modo de "scripting"

O que é uma **variável**?

A **variável** é uma denotação para uma zona de memória, i.e. um lugar onde se encontram armazenados dados.

Criar uma variável em Python é fácil, não é necessário nenhum tipo de declaração prévia. A variável é criada na primeira instrução de atribuição

```
» x = 2**34
```

Esta variável pode ser utilizada ao longo de todo o programa

```
» y = x + 1
```

Como podemos utilizar variáveis em Python? Através de instruções de atribuição

» `variable = expression`

É possível fazer várias atribuições numa só linha

» `variable_1, variable_2 = expression_1, expression_2`

As variáveis só podem ser utilizadas depois de criadas

» (...)

» variable = variable + 1 # This will not work!

» (...)

» variable = 1

» variable = variable + 1 # This will work!

Sequências de instruções

Uma sequência de instruções é um conjunto de instruções separadas por carácter de **fim de linha (Enter)**:

- » instruction_1
- » instruction_2
- » (...)
- » instruction_n

A noção de sequência, do ponto de vista, implica também que possa haver passagem de informação entre instruções, o que no Python é feito através de variáveis.

Modo de "scripting" i

Até agora foi utilizado o modo interactivo em que o python espera por instruções. No modo de podemos passar-lhe o conjunto de instruções que queremos que sejam executadas num ficheiro e o Python termina depois desse conjunto.

Python Program to find the area of triangle

```
a = 5
```

```
b = 6
```

```
c = 7
```

```
# calculate the semi-perimeter
```

```
s = (a + b + c) / 2
```

```
# calculate the area
```

```
area = (s*(s-a)*(s-b)*(s-c)) ** 0.5
```

```
print('The area of the triangle is', area)
```

É possível executar o python utilizando o ficheiro como argumento

» **python** program.py

Input e Output básico

Um programa regnal geral precisa de "comunicar" com o mundo exterior:

- Outros computadores na rede ou na internet;
- Sensores ou outros dispositivos externos;
- **Utilizadores;**

Os interfaces mais comuns com o utilizador são o teclado e o écran.

No Python a leitura é feita através da função `input()`, que é capaz de ler uma linha inteira de texto até o Enter ser pressionado. É possível mostrar uma mensagem ao utilizador:

```
x = input("Message to the user: ")
```

É da responsabilidade do utilizador processar o texto, de modo a torná-lo útil.

Transformá-lo em inteiro:

```
» x = int(input()) # 12 or 1234
```

Transformá-lo em vírgula flutuante:

```
» x = float(input()) # 12.2
```

Transformá-lo em decimal

» `x = complex(input ())`

(...)

Booleanos são deixado como exercício.

Se o texto for por exemplo uma linha com vários inteiros separados por espaços já temos de utilizar funções mais avançadas de processamento de texto que serão exploradas em capítulos mais à frente.

» `x = input.split ()`

A escrita de texto para o ecrã é feito através da função print

» `print (x)`

Também é possível escrever várias strings, ou números, separados por vírgula para o ecrã

» `print ("Hello ", " world")`

Também é possível escrever vários textos, com um separador pré-definido:

» `print ("Hello", "is it me you looking for?", sep ="—")`

Questions?