# Introdução a programação

## Como funciona um programa de computador

Um computador sem programa é como um piano sem pianista: só uma caixa cara fazendo silêncio.

Computadores são gênios... mas com amnésia e sem criatividade. Eles só sabem fazer coisas super simples, tipo somar, dividir e repetir isso zilhões de vezes por segundo.

Quer saber a velocidade média da sua viagem? O computador pode ajudar! Mas ele não faz ideia do que é “velocidade”. Você precisa ensinar tudo:

* "Aqui está a distância!"
* "Aqui está o tempo!"
* "Agora, divida isso aí!"
* "Mostra o resultado bonitinho na tela!"

Pronto: isso é um programa.  
Você dá as ordens, ele obedece rapidinho.

Mas atenção: ele só entende um idioma muito específico… a linguagem de programação!

## Linguagens Naturais vs Linguagens de Programação

Uma linguagem é como um superpoder pra transformar pensamentos em algo que os outros entendem. Pode ser com palavras, gestos, emojis ou até aquela olhadinha que diz "sai daqui agora" sem som nenhum.

* Mas e os computadores? Eles também têm seus próprios idiomas! Só que são bem básicos — tipo o vocabulário de um cão treinado: "senta", "rola", "pega esse número, divide por outro e guarda o resultado."
* Esses comandos super simples formam a chamada lista de instruções (IL), que varia de computador pra computador. É como se cada tipo de máquina só entendesse seu próprio "dialeto robótico".
* Importante: essas linguagens de máquina foram inventadas por humanos. Até agora, nenhum computador criou o próprio idioma — mas vai saber o que o futuro reserva...
* Enquanto isso, as nossas línguas, como o português, nascem e mudam naturalmente. Novas gírias entram, palavras antigas saem, e o vocabulário nunca para de crescer. Essas são chamadas de linguagens naturais — orgânicas, vivas e cheias de emoção.

## O que faz uma linguagem?

* Sintaxe

Um conjunto de regras (formais ou informais, escritas ou intuitivamente) usadas para determinar se uma determinada sequência de palavras forma uma sentença válida (por exemplo, "eu sou uma píton" é uma frase sintaticamente correta, enquanto "eu uma píton estou" não é)

* Semântica

Um conjunto de regras que determina se uma determinada frase faz sentido (por exemplo, "comi uma rosquinha" faz sentido, mas "uma rosquinha me comeu" não faz)

## Linguagem da máquina vs. linguagem de alto nível

Linguagem da máquina é o que o computador fala nativamente — um monte de 0s e 1s que pra nós parece um zumbido sem sentido. É como tentar conversar com alguém usando apenas pisca-pisca de lanterna em código Morse.

Pra nós, humanos, isso é impraticável. Então criamos um meio-termo, uma ponte entre o cérebro humano e o processador do computador: as linguagens de programação de alto nível!

Essas linguagens (como Python, Java, C++) são como tradutores simpáticos. Elas usam palavras parecidas com inglês e uma estrutura mais compreensível.

Quando escrevemos um programa com essas linguagens amigáveis, estamos criando o chamado código-fonte — tipo a receita que o computador precisa seguir. E esse código vai dentro de um arquivo-fonte, como um livro de receitas digital.

Depois, o computador ainda precisa de ajuda para entender esse código — aí entram os compiladores ou interpretadores, que traduzem do "alto nível" para a linguagem de máquina.

## Compilação vs. Interpretação

Escrever um programa é como escrever um roteiro para uma peça de teatro ou um filme. Mas só escrever não basta — você precisa de atores (o computador) que entendam o que está escrito.

Primeiro, o programador escreve tudo direitinho:

* No alfabeto certo
* Usando as palavras certas
* Seguindo a gramática da linguagem
* E fazendo sentido lógico

Mas como fazer o computador "atuar" esse roteiro?

Aí entram duas formas de tradução para a linguagem que o computador realmente entende (linguagem de máquina):

Compilação

* O tradutor lê tudo antes, traduz tudo de uma vez, e só depois entrega pro computador executar.
* Exemplo: C, C++, Java (com compilação intermediária)
* Rápido na execução final
* Mas você precisa compilar tudo antes de ver o resultado
* Analogia:  
  Você escreve um livro inteiro , manda pra gráfica, e só depois vê ele impresso e pronto pra leitura.

Interpretação

O tradutor lê e traduz linha por linha, enquanto o programa está rodando.

* Exemplo: Python, JavaScript
* Mais lento pra rodar
* Mas você vê os resultados quase instantaneamente (ideal pra testes e protótipos)
* Analogia:  
   Você faz uma peça de teatro, e o narrador vai traduzindo cada fala do script enquanto os atores falam.

A maioria das linguagens foi criada pra funcionar principalmente como uma ou outra — compilada ou interpretada. Algumas até fazem os dois com jeitinhos especiais (tipo Java, que compila pra bytecode e depois interpreta com a JVM).

# Introdução a Python

Quem criou o Python?

Spoiler: não foi uma cobra, nem uma empresa gigante cheia de programadores engravatados.

* A linguagem Python nasceu da cabeça brilhante de um cara só:  
   Guido van Rossum, nascido em 1956 em Haarlem, na Holanda 🇳🇱.
* Enquanto a maioria das linguagens é feita por times gigantes dentro de empresas corporativas, Python começou como um projeto pessoal!
* Durante o fim de ano de 1989, Guido pensou:

“Vou passar o recesso criando uma linguagem de programação só minha."

* E assim nasceu Python — leve, clara, poderosa, e (por incrível que pareça) divertida de usar!

Claro, o Guido não fez tudo sozinho pra sempre. Ao longo do tempo, milhares de desenvolvedores do mundo inteiro ajudaram a tornar o Python o que é hoje — de especialistas a curiosos, todos contribuíram para essa comunidade incrível.

## Objetivos do Python eram:

* Ser fácil e intuitivo, mas poderoso
* Ser código aberto, colaborativo
* Ser tão claro quanto inglês simples
* Ser útil pro dia a dia, com desenvolvimento rápido

Hoje, mais de 30 anos depois, todos esses objetivos foram alcançados — e com folga!

Python está sempre no topo dos rankings de linguagens mais usadas, como:

* PYPL (Popularidade de Linguagens de Programação)
* TIOBE Index

Python cresceu. Saiu do quarto de um programador para virar uma superestrela da programação mundial — usada em:

* Ciência de dados
* Inteligência artificial
* Desenvolvimento web
* Automação
* E até jogos

## Motivos Para Amar Python

1. Fácil de aprender
2. Ideal pra quem tá começando. Pouca dor de cabeça, mais resultados.
3. Fácil de ensinar
4. Professores gastam mais tempo ensinando lógica de programação, não decifrando gírias do código.
5. Fácil de escrever código novo
6. Menos linhas, mais funcionalidade. Você programa mais em menos tempo.
7. Fácil de ler e entender
8. Fácil de instalar e usar
9. Grátis, leve e roda em Windows, Mac, Linux… até em um Raspberry Pi!

Além disso...

1. Tem bibliotecas pra tudo: web, IA, jogos, ciência, automação...
2. É usado por Google, Netflix, NASA, Spotify e muito mais!
3. A comunidade é gigante e acolhedora: você nunca programa sozinho.
4. E sim… Python é cool. Você se sente poderoso usando

## Rivais do Python

1. Perl: amado por admins e veteranos, mas perdeu terreno.
2. Python fica no meio: nem tão caótico quanto Perl, nem tão filosófico quanto Ruby.
3. Ruby: inovador, mas seu brilho ficou mais restrito à web.

A verdade?  
Cada linguagem tem seu charme e seu uso ideal. Mas Python acabou ganhando o coração da galera por ser prático, legível e versátil.

## Por que não usar Python?

Python é incrível, mas... ele tem seus limites. Aqui estão os principais territórios onde ele ainda não reina:

1. Programação de Baixo Nível (“Próxima ao Metal”)

* Ex: Drivers, sistemas operacionais, manipulação direta de hardware
* Python é interpretado e mais lento do que linguagens como C ou C++.
* Ele não te dá acesso direto à memória e recursos de hardware como essas linguagens fazem.
* Melhores alternativas: C, C++, Rust

2. Aplicativos para Dispositivos Móveis

* Ex: Apps nativos para Android e iOS
* O suporte para apps móveis ainda é limitado e menos eficiente.
* A maioria das lojas e sistemas preferem linguagens como Kotlin, Swift ou Java.
* Existem tentativas com Python? Sim! Com frameworks como Kivy, BeeWare, e PySide, mas ainda não são os queridinhos da indústria.
* Melhores alternativas: Kotlin (Android), Swift (iOS), Flutter (Dart)

## Python 2 vs Python 3: A Batalha das Cobras

Python 2

* Versão mais antiga, tipo um veterano de guerra.
* Parou de evoluir — não ganha novos recursos, só correções de bugs e segurança.
* Código Python 2 não roda direto no Python 3.
* Migrar do 2 para o 3 pode ser um parto: tem que revisar o código todo, corrigir incompatibilidades, e ainda rola o risco de novos bugs surgirem.

Python 3

* A versão moderna, atual e que está ganhando upgrades e novidades.
* Tem sintaxe mais limpa, novos recursos, e é o futuro da linguagem.
* Não é compatível com Python 2, o que gerou confusão e atrasou a migração de muitos projetos.

## Implementações de Python: A Família Grande!

CPython — o “Python padrão”, o queridinho oficial!

* Criado pelo próprio Guido van Rossum em C.
* É a implementação de referência, ou seja, é o modelo que as outras tentam seguir.
* Roda na maioria das máquinas do mundo, porque quase todo lugar aceita código C.

PyPy

* Um primo turbo que usa JIT (Just-In-Time Compilation) para acelerar o Python.
* É mais rápido para muitos tipos de programa

Jython

* Python que fala diretamente com o Java!
* Perfeito para integrar com aplicativos Java.

IronPython

* O primo que trabalha com .NET Framework da Microsoft.
* Ideal para quem quer misturar Python com tecnologias Microsoft.

MicroPython

* Python para o mundo dos microcontroladores e IoT (Internet das Coisas).
* Leve e rápido para gadgets pequenininhos.