Advanced Institute for Artificial Intelligence

# Programação em Python

https://advancedinstitute.ai

# Background

## Referências Fontes das Imagens


- Hackernoon Post on Stativ vs Dynamic Typing
- Understanding Python Bytecode
- The Difference Between Compiled and Interpreted Languages
- Learning Python: Powerful Object-Oriented Programming (Book)
- Fluent Python (Book)

## Motivação

- Por quê utilizar Python?
  - **Qualidade do Software**:
    - projetado para ser legível e, portanto, reutilizável e sustentável;
    - Fácil de entender, mesmo que você não tenha escrito
    - Suporte a mecanismos de reutilização de software avançados, como orientação a objetos (OO) e programação funcional;
  - **Produtividade**:
    - Normalmente um terço a um quinto do tamanho do código C ++ ou Java equivalente;
    - Menos digitado, menos para depurar e menos para manter;

## Motivação

- Por quê utilizar Python?
  - **Portabilidade**:
    - A maioria dos programas Python é executada *out-of-the-box* em todas as principais arquiteturas/sistemas operacionais;
  - **Disponibilidade de Bibliotecas**:
    - Grande coleção de funcionalidades *built-in – Standard Library*
  - **Integração de Componentes**:
    - Pode invocar bibliotecas C e C++, pode ser chamado de programas C e C++, pode ser integrado a componentes Java e .NET
    - Interação via chamada remota de funções como SOAP, XML-RPC e CORBA;

## Histórico

- **Computadores eletrônicos** surgiram meados da **década de 1940**;
  - Programas: longa sequência de zeros e uns (**linguagem de máquina**)
- **Linguagem de Montagem** (1948):
  - Assembly language, comandos curtos com adição de mnemônicos: `ADD R1 R2`
    - Programas simples contendo centenas desses comandos;
- **Linguagens de Alto Nível**: FORTRAN (1957) - *formula translator*
  - **Traduzir fórmulas da matemáticas** para a linguagem de máquina do computador;
    - **Linguagem Compilada**;
  - LISP (1958): inicialmente utilizada para IA;
    - **Linguagem Interpretada**;

## Histórico

- **Orientação a objetos**: 1967 com o **Simula**
- Aperfeiçoada até chegar no **Smalltalk-80**;
- Seguido de **C**, **C++**, **Objetive C** e hoje dia a gente tem uma série de opções, **Java**, **Python**, **Ruby**;

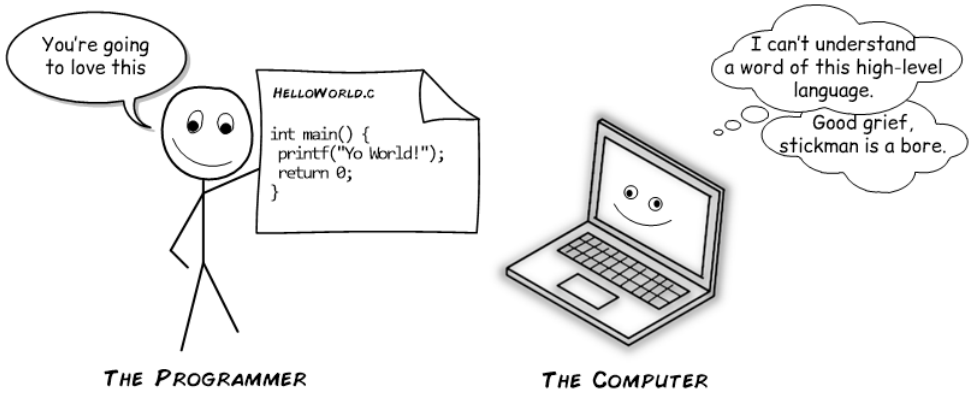
## Interpretadores vs Compiladores

- Os computadores não executam seu código diretamente;
- É necessária uma etapa de tradução;
- Converte código de alto nível em código de máquina;
- Transformação de código;

**Importante**

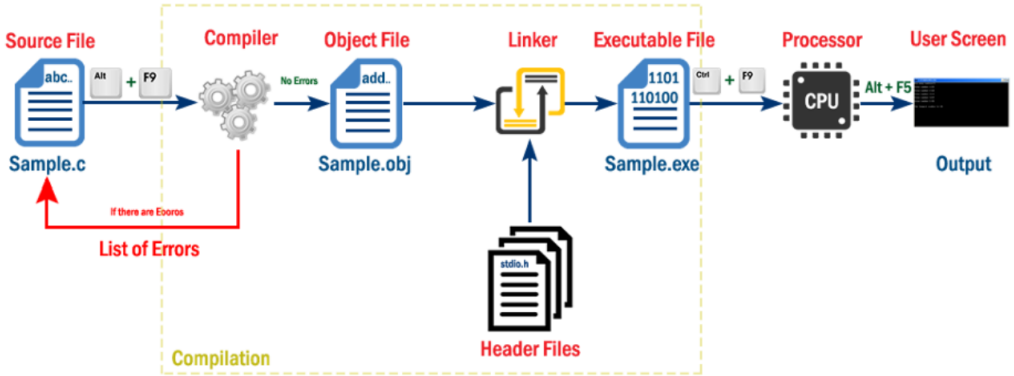
Como traduzir seu código de alto nível para baixo nível?

## Interpretadores vs Compiladores

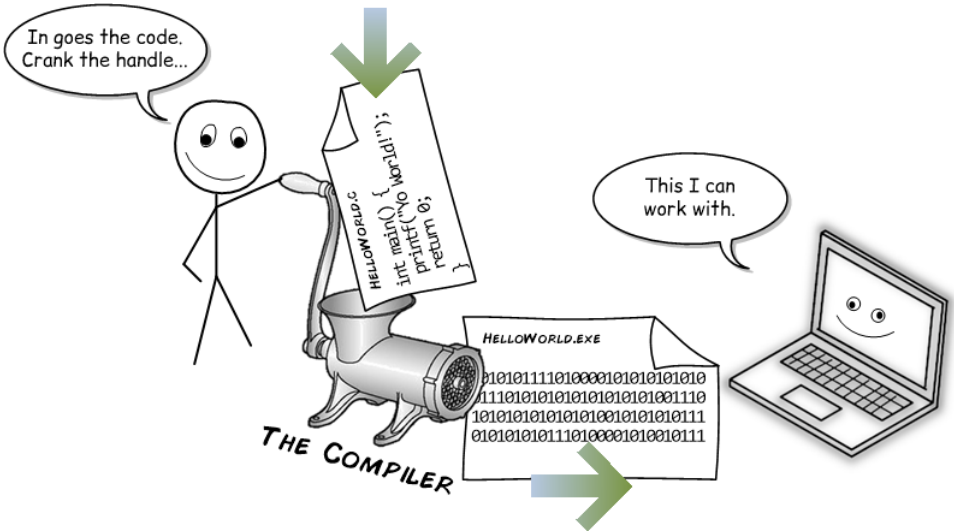


## Processo de Compilação

- A compilação traduz seu código-fonte em instruções de máquina:



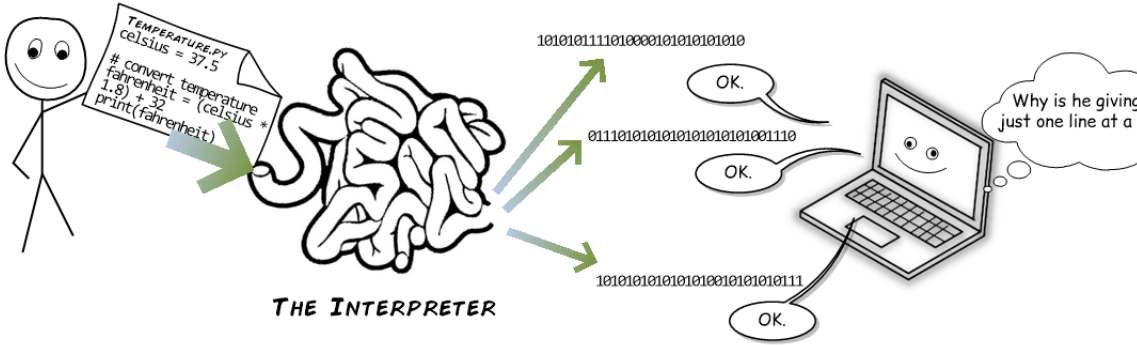
# Linguagens Compiladas



## O Interpretador

- Leitura de **uma linha por vez** do código fonte, seguida da interpretação daquela linha;
- **Traduzir** cada linha para linguagem de máquina do computador
- Transformar para o código binário e executar usando as instruções do processador do hardware do computador;
- **Similar ao trabalho do compilador, porém com porções menores** (e.g., linha a linha);

# Linguagens Interpretadas



# Vantagens e Desvantagens

## Linguagens Interpretadas

- Prós: *Write once, Run anywhere*; facilidade de depuração (ligação entre código fonte e execução), menor tempo para testar (prototipação rápida)
- Cons: desempenho (otimização da execução é difícil), necessidade do interpretador para execução e **desempenho**

## Linguagens Compiladas

- Prós: desempenho, distribuição do executável e **desempenho**
- Cons: dependência de plataforma/S.O., processo de depuração mais custoso, prototipação lenta

## Python

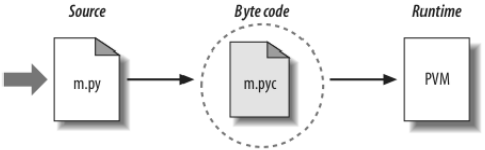
- Compilação antes de executar o programa;
- Geração de *bytecode*
  - Algo parecido com linguagem de montagem *assembly*
  - **Execução rápida** por ser próximo da linguagem de máquina
  - Utilização de Máquina virtual: **Python Virtual Machine** (PVM)

```
1  LOAD_FAST        0 (y)
2  LOAD_CONST       1 (1)
3  BINARY_ADD
4  STORE_FAST       1 (x)
```

## Python Virtual Machine - PVM

- Parte do interpretador Python;
  - Inspirado na JVM do Java;
- Execução de arquivos `.pyc`
  - Salvo no diretório `.__pycache__`
  - Evita a necessidade de recompilar o programa para execução em outras arquiteturas;
- O mesmo programa rodar tanto num notebook, quanto num desktop, quanto num celular, quanto num relógio;
- Possui sua própria especificação de arquitetura:
  - Máquina ideal, que tem uma certa linguagem, que na verdade é o *bytecode*

## Python Virtual Machine - PVM




- Várias implementações:
  - **CPython**: Implementação referência completa da linguagem;
  - **Jython** e **IronPython**: Geração de bytecode para JVM e framework .NET;
  - **PyPy**: Implementação completa do python otimizada com **J**ust **I**n **T**ime Compiler - JIT;

## Checagem de Tipos

- Quando que tipos são checados:
  - **Estático**: Tipos verificados antes do tempo de execução
    - Adição de um passo extra ao processo de compilação para checagem de tipos;
    - Implícitos vs Explícitos;
    - Mudanças de tipos não são permitidas;
    - Melhor performance em tempo de execução (sem necessidade de fazer checagem de tipos)

## Checagem de Tipos

- **Dinâmico**: Tipos verificados em tempo real, durante a execução
  - ***Duck typing***: "if it walks like a duck and it quacks like a duck, then it must be a duck"
  - O tipo ou a classe de um objeto é menos importante do que os métodos que ele define;
- ***Type Hints***: Introduzido com Python 3.9

## Checagem de Tipos

- Python faz checagem dinâmica de tipos;
  - Compatibilidade de tipos só é verificada quando a **sentença é executada**

```
>>> if False:
...     1 + "two"  # This line never runs, so no TypeError is raised
... else:
...     1 + 2
...
3

>>> 1 + "two"  # Now this is type checked, and a TypeError is raised
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

## Checagem de Tipos

- ***Type Hints***: Introduzido com Python 3.9

```
1  def greeting(message: str = "World") -> str:
2      return f"Hello, {message}"
3
4  greeting(1)
```

Checando os *hints*

```
1  > mypy file.py
2  file.py:4: error: Argument 1 to "greeting" has incompatible type "int";
       expected "str"
3  Found 1 error in 1 file (checked 1 source file)
```

# Python

## Módulos e arquivos

- Todo arquivo de código-fonte Python cujo nome termina em uma extensão .py é um módulo;
- Nenhum código ou sintaxe especial é necessário para tornar um arquivo um módulo;
- Outros arquivos podem acessar os itens que um módulo define, importando esse módulo;
- Ideia central por trás da arquitetura do programa em Python;
- Importação funciona uma única vez por execução (operação custosa);
- Variável de ambiente `$PYTHONPATH` descreve o lugar de busca pelo módulo;

## Módulos e arquivos

Arquivo `script1.py`

```
1 msg = "Hello World"
2 print(f"{msg}")
```

Executando o `import`:

```
1 >>> import script1
2 Hello World
3 >>> import script1
4 (None)
```

## Módulos e arquivos

Forçando o *reload* do módulo:

```
1 >>> from importlib import reload
2 >>> reload(script1)
3 Hello World
4 <module 'script1' from '/private/tmp/script1.py'>
```

## Módulos e arquivos

Acesso aos atributos definidos no módulo:

```
1  >>> import script1
2  Hello World
3  >>> print(f"Mensagem no módulo: {script1.msg}")
4  Mensagem no módulo: Hello World
5
6  >>> from script1 import msg
7  Hello World
8  >>> print(f"Mensagem no módulo: {msg}")
9  Mensagem no módulo: Hello World
```

# Python

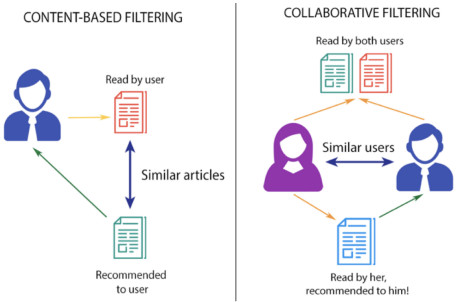
## Exemplo: Sistemas de Recomendação

## Referências Fontes das Imagens


- Comprehensive Guide on Item Based Collaborative Filtering
- Build a Recommendation Engine With Collaborative Filtering
- Recommendation Systems from Scratch in Python

## Introdução

- Utilizar preferência de usuários para recomendar itens
- **Filtros baseados em conteúdo**: Único Usuário
- **Filtros Colaborativos**: Utilizar preferências de outros;



## Introdução

- **Medida de Similaridade**
- Filtro colaborativo **baseado no usuário**:
  - Agrupamento de usuários *similares*;
  - Predição do valor de classificação do item não revisado pelo usuário baseado nos usuários *similares;*
  - Algoritmo que consome muito tempo:
    - Preciso calcular a similaridade com todos os usuários;

## Introdução

- Filtro colaborativo **baseado nos itens**:
  - Agrupamento de itens *similares*;
  - Predição do *score* de classificação do item não revisado pelo usuário baseado nos scores dados a itens *similares;*
  - Algoritmo que consome muito tempo:
    - Preciso calcular a similaridade entre todos os itens;

## Dúvidas?