Como o Python funciona?

Live de Python #218

Roteiro

1. Entendimento geral

Uma visão geral sobre a parse e a VM

2. Tokens e Parser

Transformando o .py em algo estruturado

3. ASTs e compilação

De árvores ao .pyc

4. A máquina virtual

A execução do código de fato

- Live totalmente informativa

Avisos

- Live totalmente informativa
- Objetivo de explicar de forma simples

Avisos

- Live totalmente informativa
- Objetivo de explicar de forma simples
- **Foco em pessoas não cientistas da computação**

Avisos

- Live totalmente informativa
- Objetivo de explicar de forma simples
- **Foco em pessoas não cientistas da computação**
- **Python 3.11**

Avisos

Acássio Anjos, Ademar Peixoto, Adilson Herculano, Adriana Cavalcanti, Alexandre Harano, Alexandre Lima, Alexandre Souza, Alexandre Takahashi, Alexandre Villares, Alex Lima, Alynne Ferreira, Alysson Oliveira, Ana Carneiro, Andre Azevedo, André Rafael, Aquiles Coutinho, Arnaldo Turque, Aurelio Costa, Bruno Batista, Bruno Freitas, Bruno Guizi, Bruno Oliveira, Bruno Ramos, Caio Nascimento, Carina Pereira, Carlos Alipio, Christiano Morais, Clara Battesini, Daniel Freitas, Daniel Haas, Danilo Segura, David Couto, David Kwast, Delton Porfiro, Dhyeives Rodovalho, Diego Farias, Diego Guimarães, Dilenon Delfino, Dino Aguilar, Diogo Albuquerque, Diogo Paschoal, Douglas Bastos, Douglas Braga, Douglas Zickuhr, Dutofanim Dutofanim, Eliel Lima, Elton Silva, Emerson Rafael, Erick Ritir, Érico Andrei, Eugenio Mazzini, Euripedes Borges, Everton Silva, Fabiano Tomita, Fabio Barros, Fábio Barros, Fabio Castro, Fábio Thomaz, Felipe Rodrigues, Fernanda Prado, Fernando Rozas, Flávio Meira, Flavkaz, Gabriel Barbosa, Gabriel Nascimento, Gabriel Simonetto, Geandreson Costa, Guilherme Cabrera, Guilherme Felitti, Guilherme Gall, Guilherme Ostrock, Guilherme Piccioni, Gustavo Dettenborn, Gustavo Suto, Heitor Fernandes, Henrique Junqueira, Hugo Cosme, Igor Taconi, Israel Gomes, Italo Silva, Jair Andrade, Jairo Lenfers, Janael Pinheiro, João Lugão, João Paulo, João Rodrigues, Joelson Sartori, Johnny Tardin, Jonatas Leon, Jônatas Silva, Jonathan Morais, José Gomes, Joseíto Júnior, Jose Mazolini, José Pedro, Juan Gutierrez, Juliana Machado, Júlio Gazeta, Julio Silva, Kaio Peixoto, Kaneson Alves, Leandro Miranda, Leonardo Mello, Leonardo Nazareth, Leonardo Rodrigues, Lucas Adorno, Lucas Mello, Lucas Mendes, Lucas Nascimento, Lucas Oliveira, Lucas Simon, Lucas Teixeira, Lucas Valino, Luciano Silva, Luciano Teixeira, Luiz Junior, Luiz Lima, Luiz Paula, Luiz Perciliano, Maicon Pantoja, Maiquel Leonel, Marcelino Pinheiro, Marcelo Matte, Márcio Martignoni, Marcio Moises, Marco Mello, Marcos Gomes, Marco Yamada, Maria Clara, Marina Passos, Mateus Lisboa, Matheus Cortezi, Matheus Silva, Matheus Vian, Mauricio Nunes, Mírian Batista, Murilo Andrade, Murilo Cunha, Murilo Viana, Natan Cervinski, Nathan Branco, Nicolas Teodosio, Osvaldo Neto, Patricia Minamizawa, Patrick Felipe, Paulo Braga, Paulo Tadei, Pedro Henrique, Pedro Pereira, Peterson Santos, P Muniz, Priscila Santos, Rafael Lopes, Rafael Rodrigues, Rafael Romão, Ramayana Menezes, Regis Tomkiel, Renato Veirich, Ricardo Silva, Riverfount Riverfount, Robson Maciel, Rodrigo Alves, Rodrigo Cardoso, Rodrigo Freire, Rodrigo Oliveira, Rodrigo Quiles, Rodrigo Vaccari, Rodrigo Vieira, Rogério Nogueira, Rogério Sousa, Ronaldo Silva, Ronaldo Silveira, Rui Jr, Samanta Cicilia, Thalles Rosa, Thiago Araujo, Thiago Bueno, Thiago Curvelo, Thiago Moraes, Thiago Oliveira, Thiago Salgado, Thiago Souza, Tiago Minuzzi, Tony Dias, Valcilon Silva, Valdir Tegon, Victor Wildner, Vinícius Bastos, Vitor Luz, Vladimir Lemos, Walter Reis, Wellington Abreu, Wesley Mendes, William Alves, Willian Lopes, Wilson Neto, Wilson Rocha, Xico Silvério, Yury Barros

Obrigado você

picpay.me/dunossauro

apoia.se/livedepython

pix.dunossauro@gmail.com

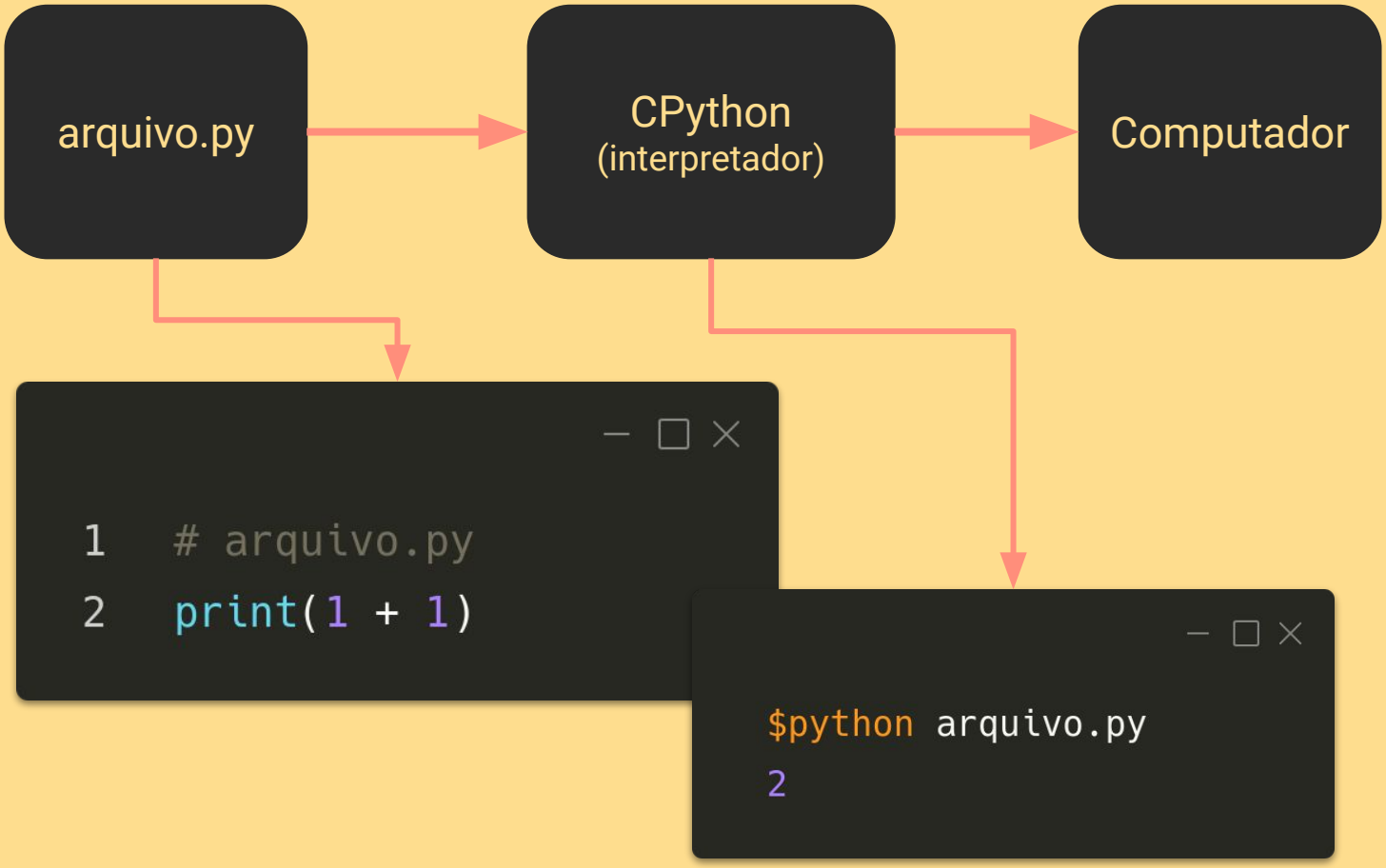
Ajude o projeto <3

Geral

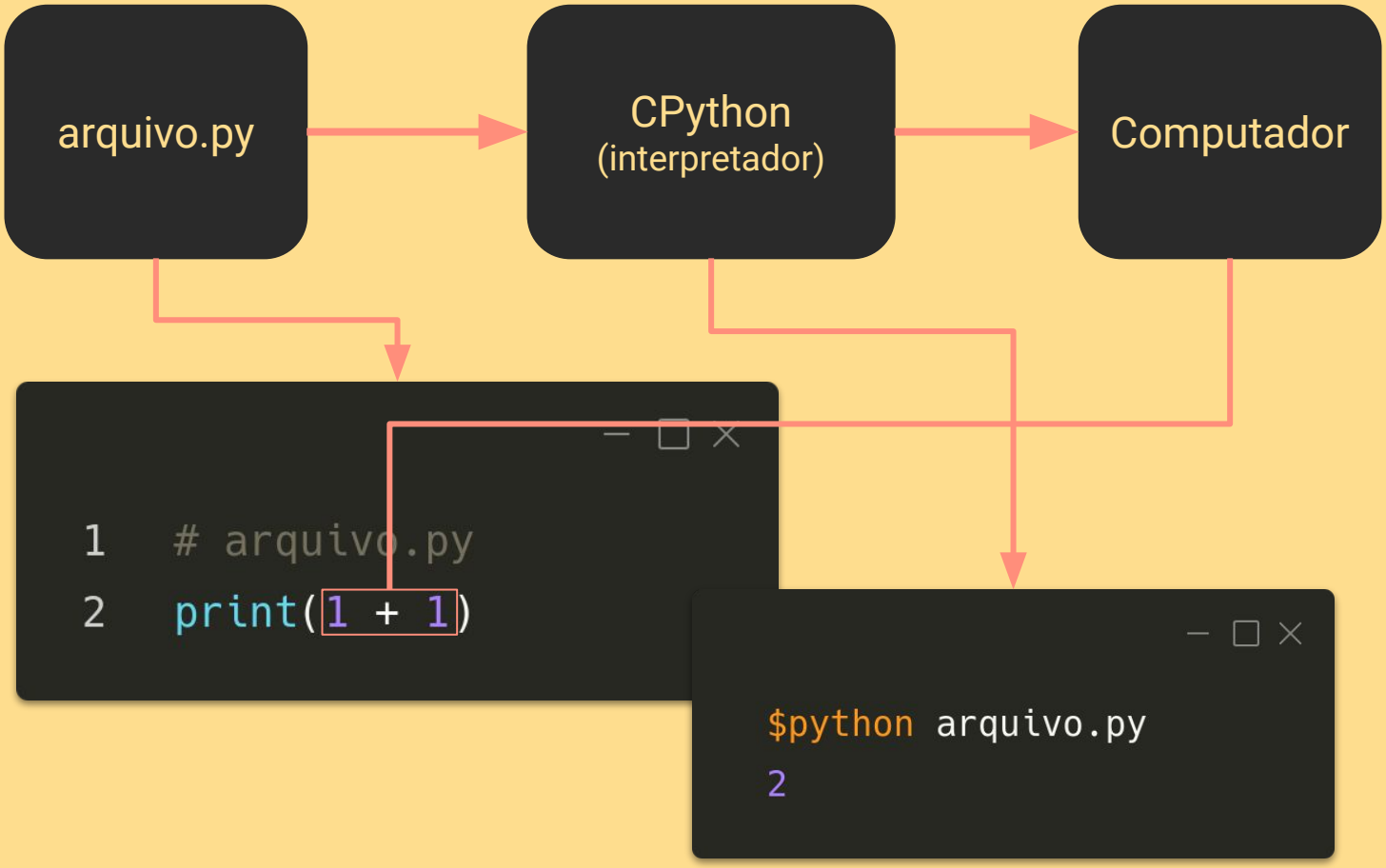
Como a mágica acontece?

Como o python executa o nosso código?

Como o python executa o nosso código?



Como o python executa o nosso código?

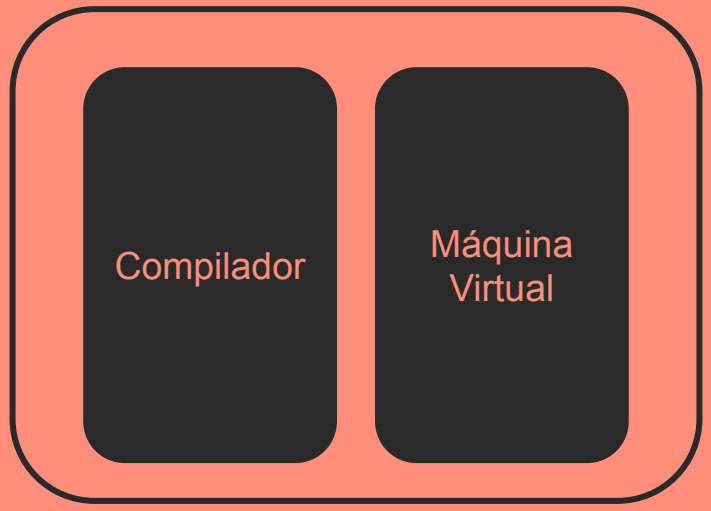


Interpretador? O que é isso?

Pense em um interpretador como alguém que interpreta textos. Por exemplo, você está lendo o texto desse slide e interpretado o que está escrito aqui. Sei que esse texto é meio longo, mas acredito que você já tenha entendido que o que o interpretador faz é interpretar o texto. Da mesma forma que você está fazendo sem perceber enquanto tenta ler tudo que escrevi nesse slide e que está colocado de forma graciosamente verborrágica para você entender que você, sim **você**, também é um interpretador de texto. Assim como o **CPython**. Embora você consiga ler uma linguagem natural e muito mais complexa do que ele.

O CPython

- Cpython é um programa que lê e executa o código da linguagem
- Escrito em C
- É um software como qualquer outro
- **Não esquecer de ir ao repositório**

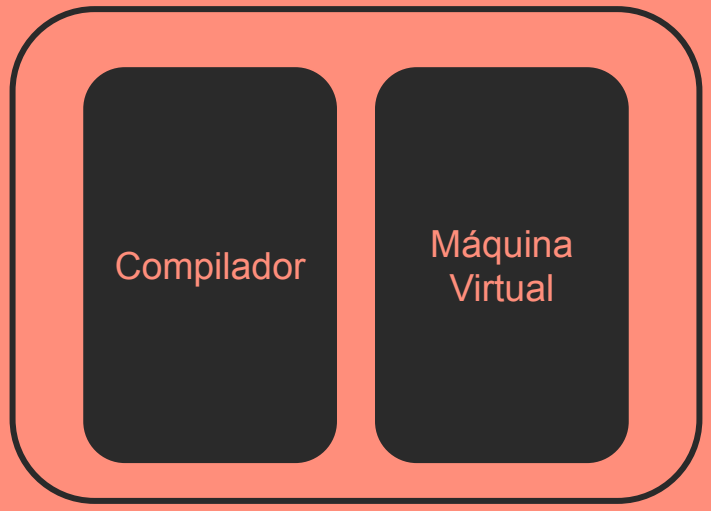


https://github.com/python/cpython

Outro interpretadores

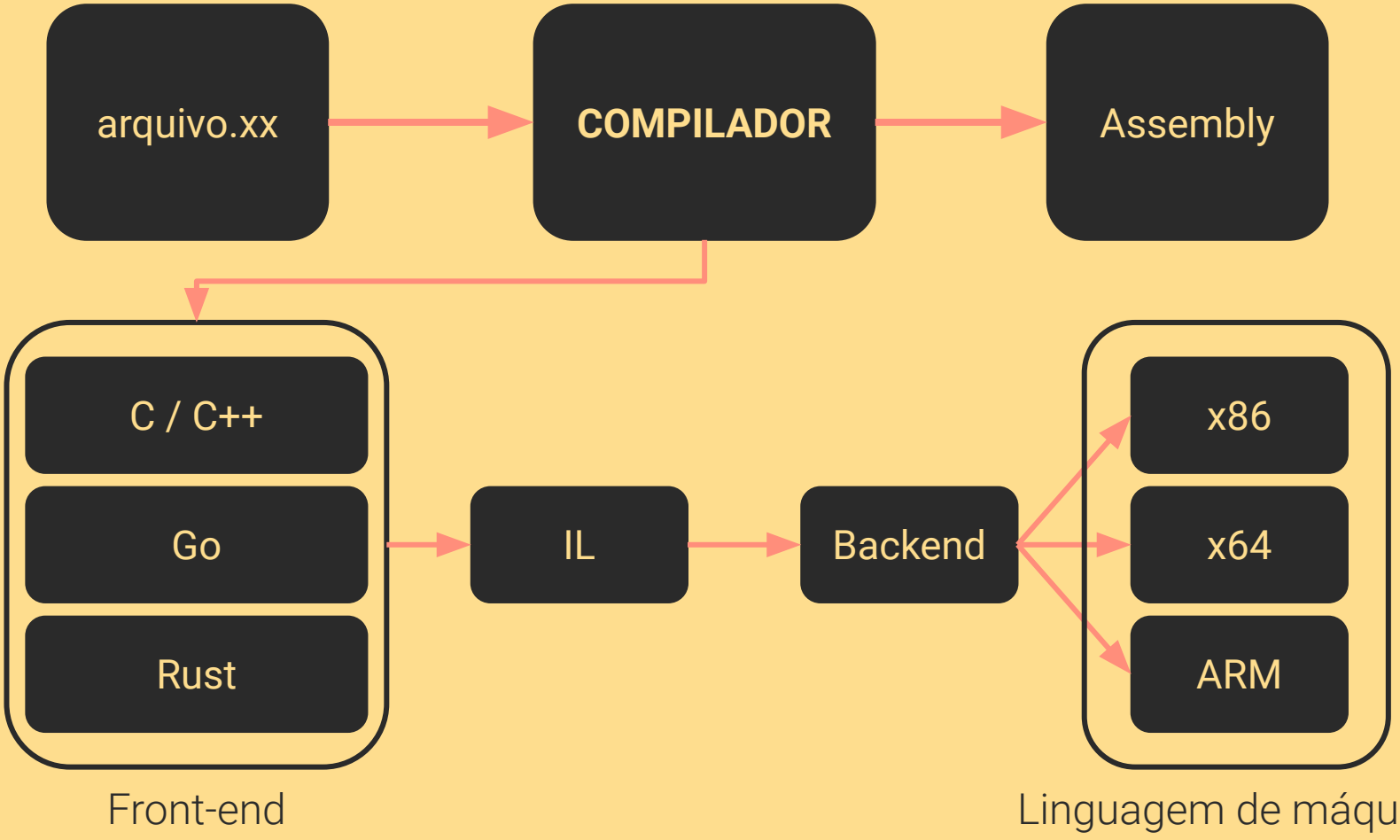
Existe uma diversidade de outros interpretadores para Python, veremos eles no final dessa live!

CPython

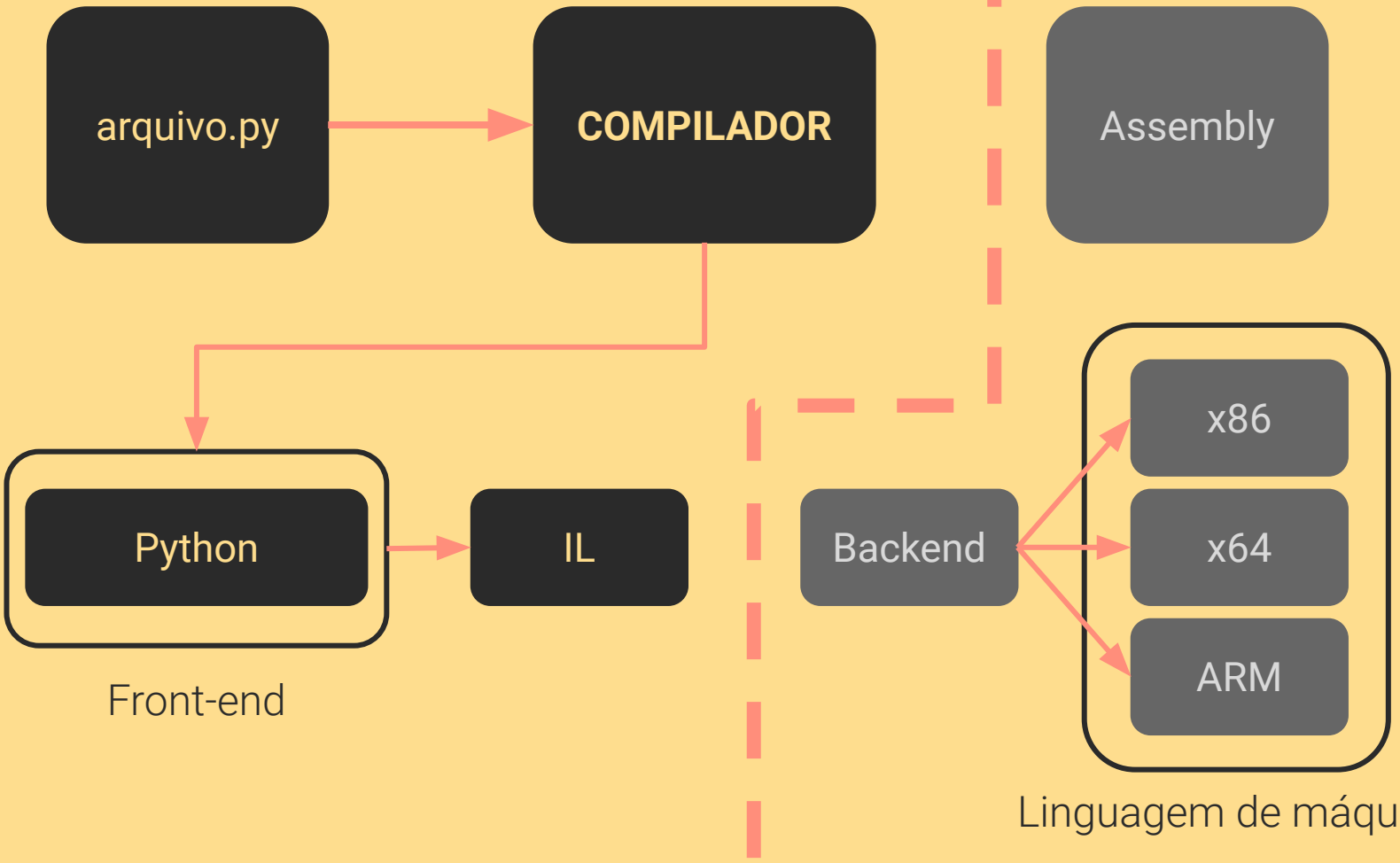


https://github.com/python/cpython

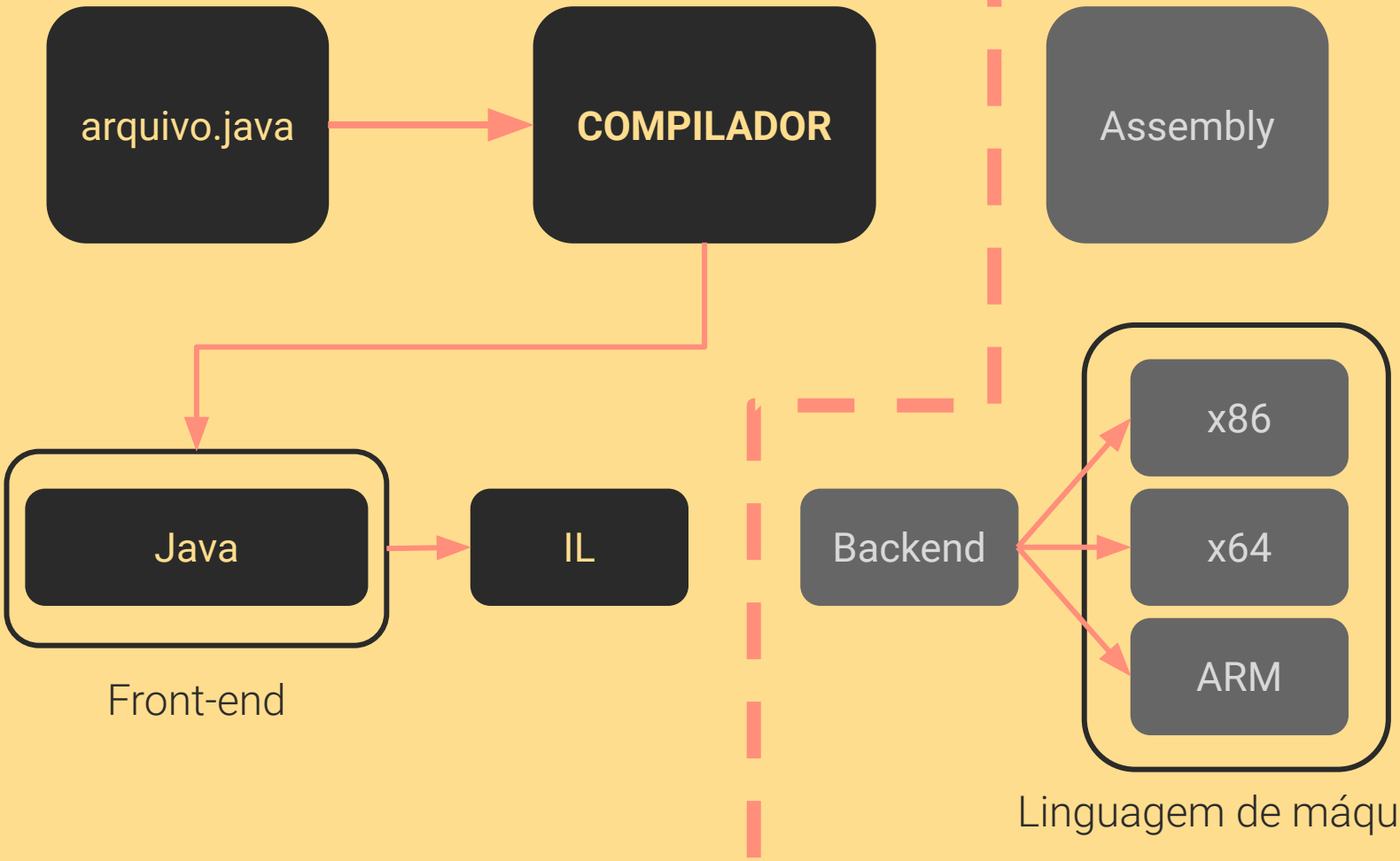
Compilador tradicional



Compilador do Python?



Compilador do Java?



IL x Linguagem de máquina

```
print('Hello world')
```

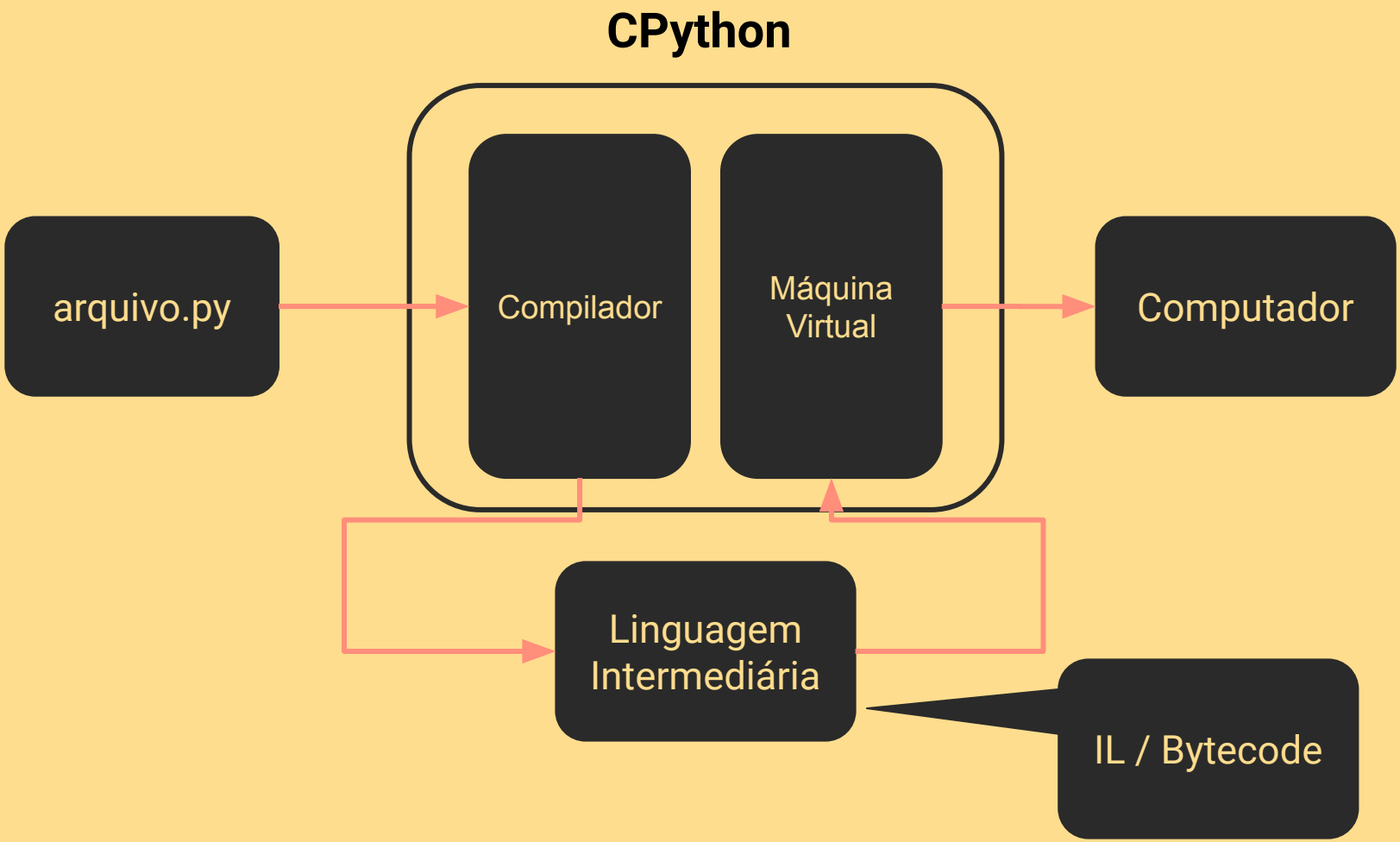
```
1   0 LOAD_NAME                0 (print)
    2 LOAD_CONST               0 ('Live de Python')
    4 CALL_FUNCTION            1
    6 POP_TOP
    8 LOAD_CONST               1 (None)
   10 RETURN_VALUE
```

Linguagem intermediaria

```
0000000 457f 464c 0102 0001 0000 0000 0000 0000
0000010 0003 003e 0001 0000 0000 0000 0000 0000
0000020 0040 0000 0000 0000 6c80 0002 0000 0000
0000030 0000 0000 0040 0038 000b 0040 0021 0020
0000040 0001 0000 0004 0000 0000 0000 0000 0000
0000050 0000 0000 0000 0000 0000 0000 0000 0000
0000060 5990 0000 0000 0000 5990 0000 0000 0000
0000070 1000 0000 0000 0000 0001 0000 0005 0000
0000080 6000 0000 0000 0000 6000 0000 0000 0000
0000090 6000 0000 0000 0000 8d65 0000 0000 0000
00000a0 8d65 0000 0000 0000 1000 0000 0000 0000
00000b0 0001 0000 0004 0000 f000 0000 0000 0000
00000c0 f000 0000 0000 0000 f000 0000 0000 0000
00000d0 2884 0000 0000 0000 2884 0000 0000 0000
00000e0 1000 0000 0000 0000 0001 0000 0006 0000
00000f0 2770 0001 0000 0000 2770 0001 0000 0000
0000100 2770 0001 0000 0000 0a68 0000 0000 0000
...
```

Python compilado com mypyc

O CPython


CPython
arquivo.py
Compilador
Máquina Virtual
Computador
Linguagem Intermediária
IL / Bytecode

Maquina virtual

Um compilador compila código para uma determinada máquina.

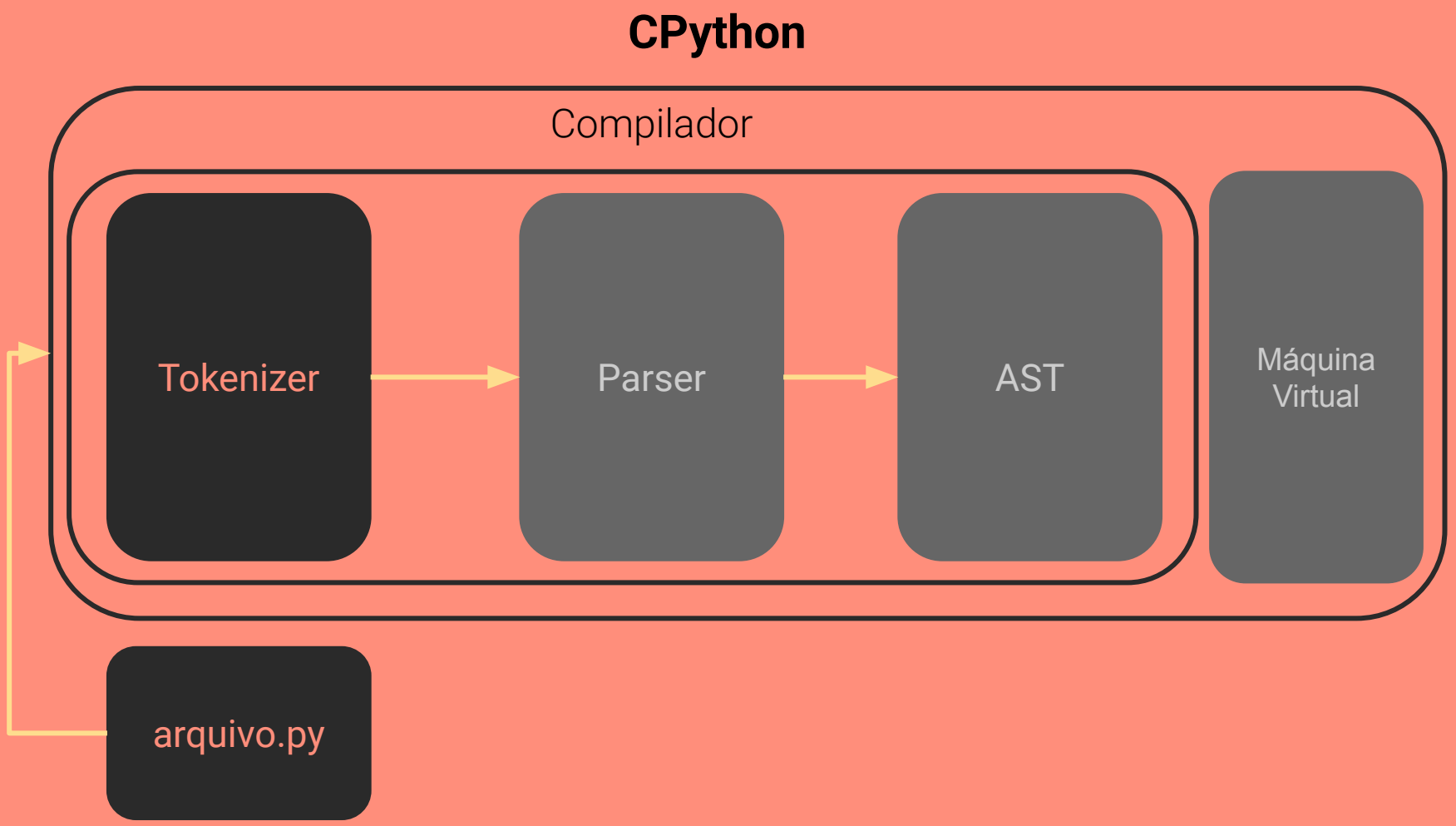
- **Compiladores tradicionais** compilam código para **máquinas reais**
- O **compilador do python** compila código para uma **máquina VIRTUAL**

Uma máquina virtual é um computador hipotético, criado via software

Token e Parser

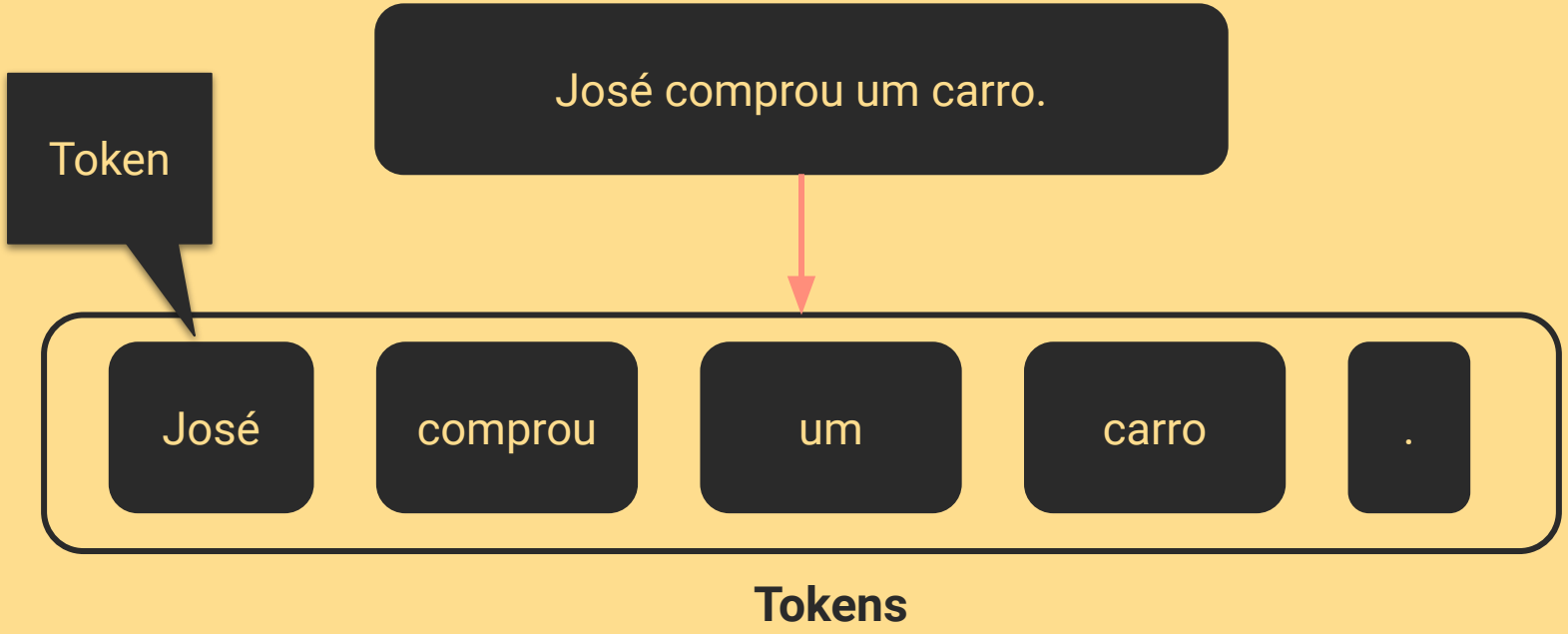
O começo do entendimento

O início do interpretador



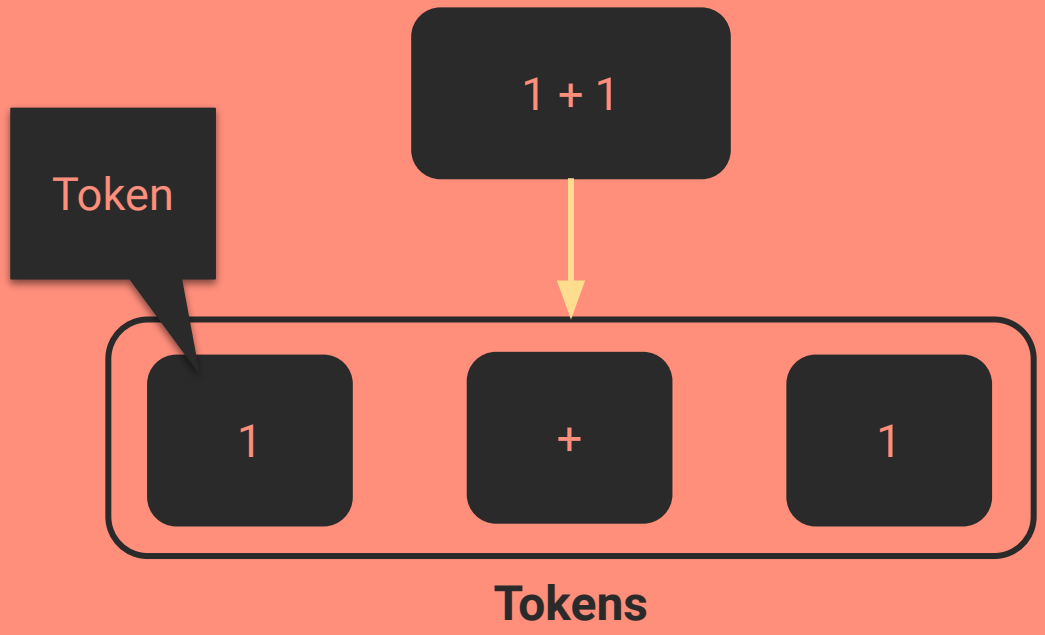
Tokenização

No primeiro momento, o python vai ler o seu código e separar em tokens.
A análise léxica é como a **análise morfológica** no português.



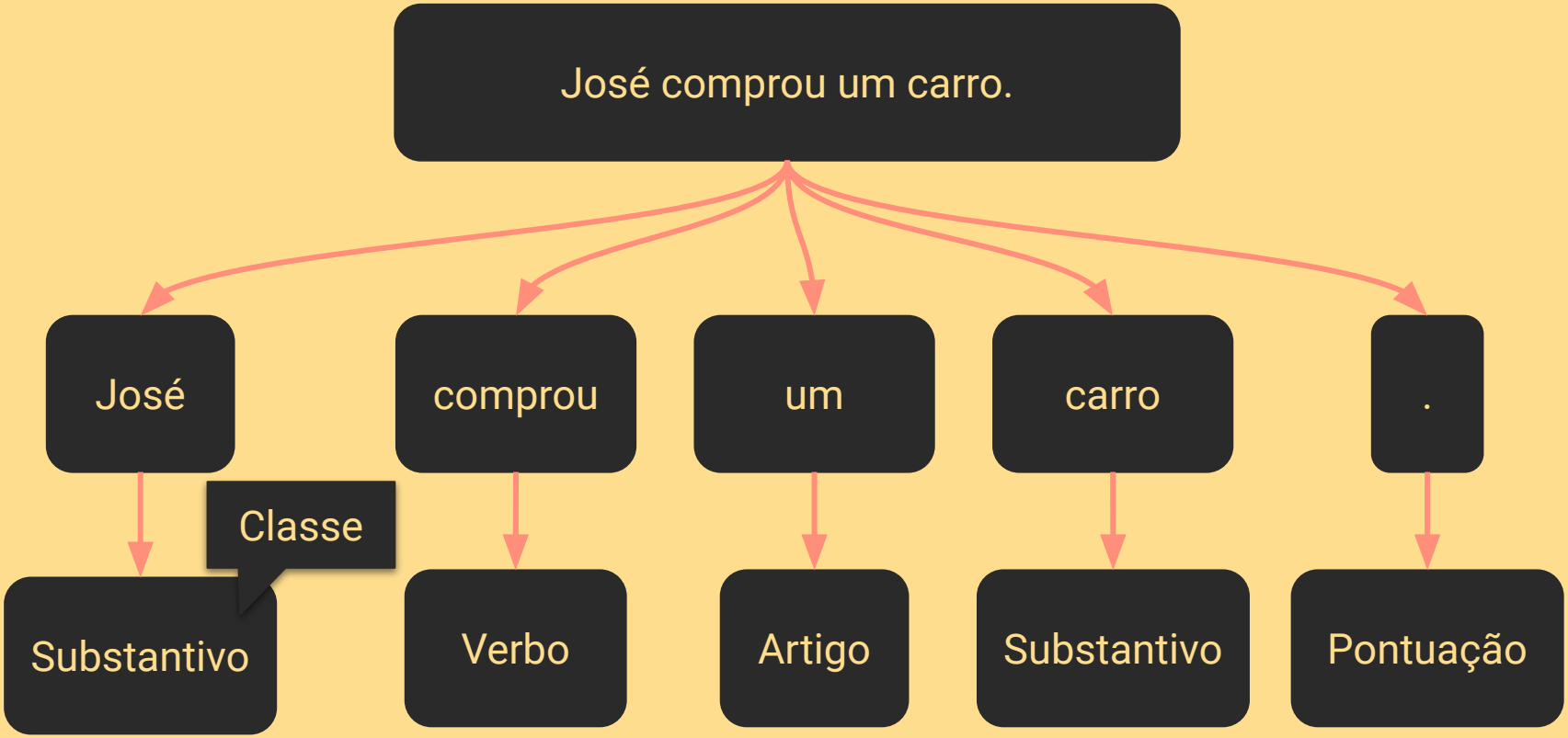
Tokens

Os tokens são as menores partes de uma linguagem. Na nossa gramática, no português são palavras. O Python tem sua própria gramática

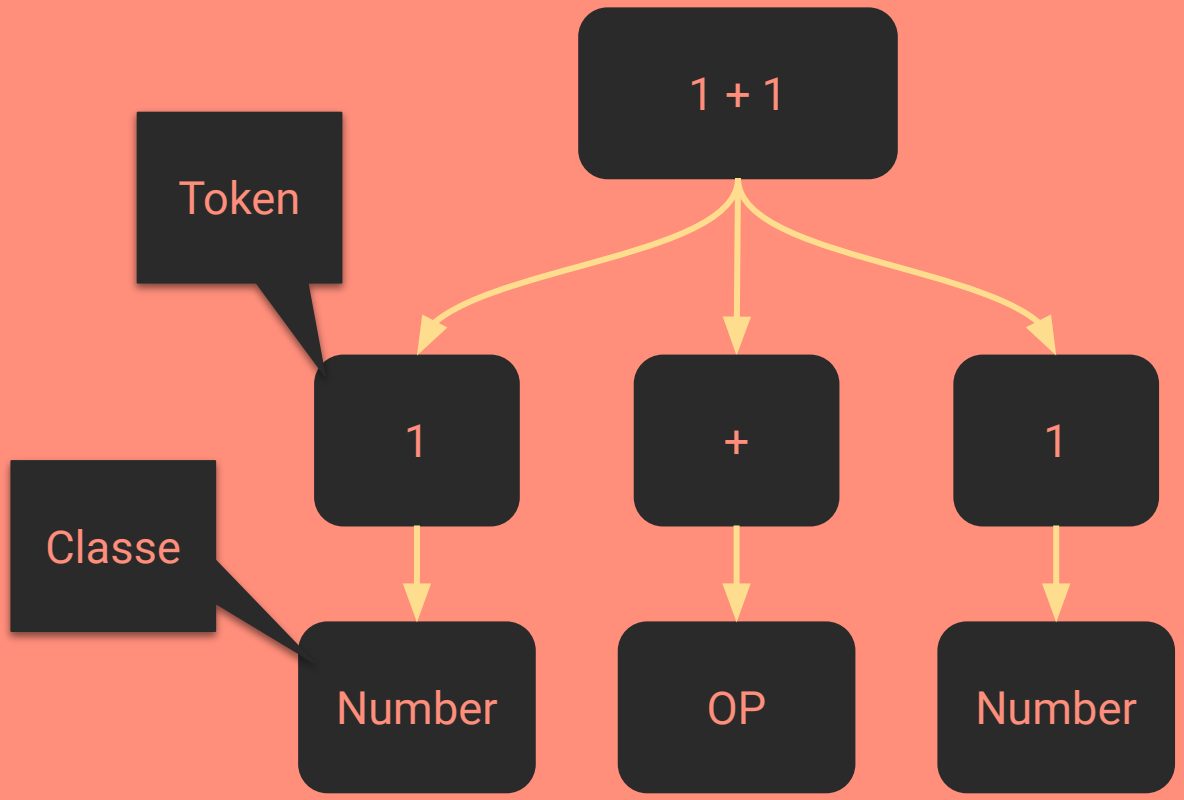


Análise léxica

Após separarmos as palavras, encontramos suas classes gramaticais.



Análise léxica



https://github.com/python/cpython/blob/main/Grammar/Tokens

Mas como o Python faz isso?

O python tem uma lista de tokens reconhecidos pela linguagem. E você pode ver o processo de tokenização.

```
  $python -m tokenize exemplo.py
#linha,posicao,      classe,         token
  0,0-0,0:           ENCODING        'utf-8'
  1,0-1,1:           NUMBER          '1'
  1,2-1,3:           OP              '+'
  1,4-1,5:           NUMBER          '1'
  1,5-1,6:           NEWLINE         '\n'
  2,0-2,0:           ENDMARKER       ''
```

https://github.com/python/cpython/blob/main/Grammar/Tokens

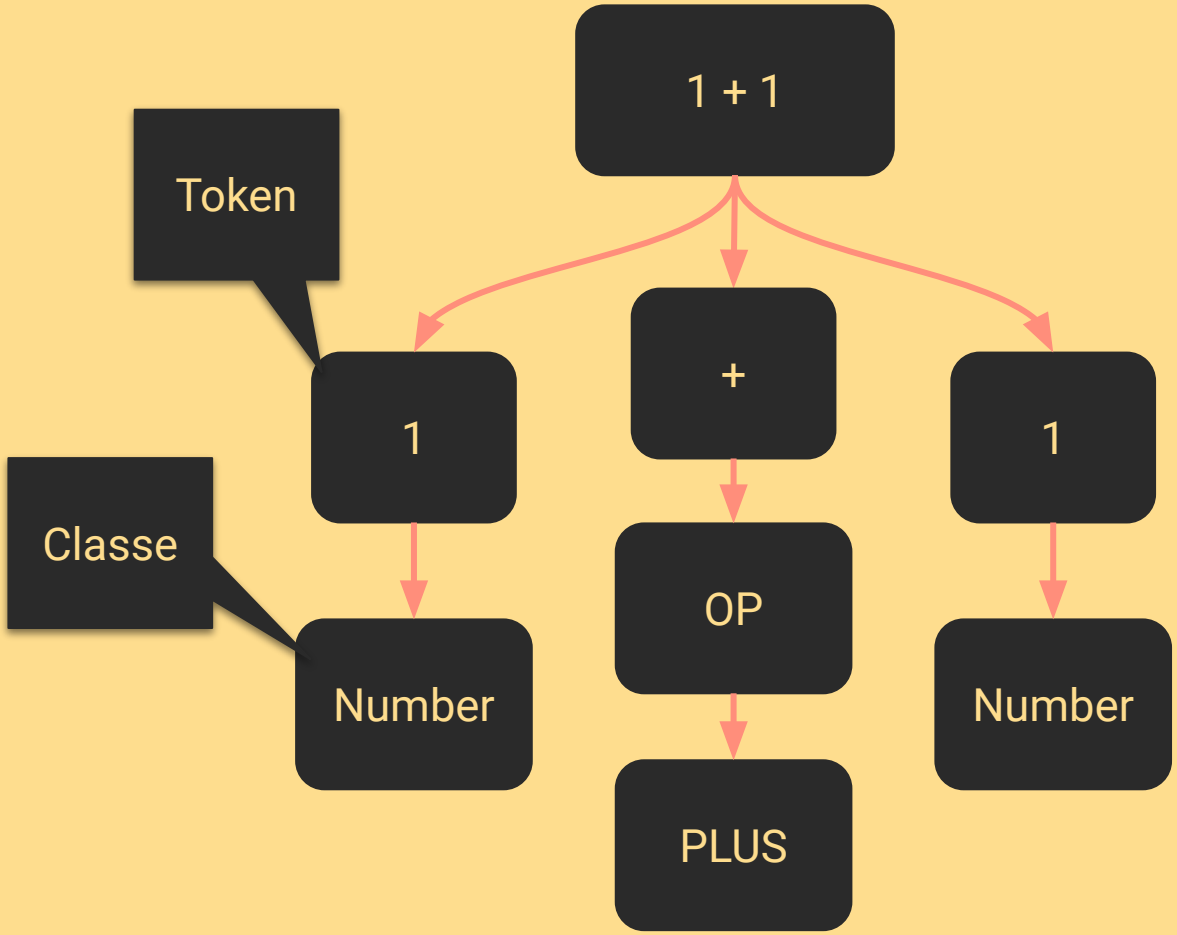
Subclasses


José comprou um carro.
José
comprou
um
carro
.
Classe
Substantivo
Verbo
Artigo
Substantivo
Pontuação
Próprio
Comum

A flag -e

```
$python -m tokenize -e exemplo.py
#linha,posicao,        classe,         token
0,0-0,0:               ENCODING        'utf-8'
1,0-1,1:               NUMBER          '1'
1,2-1,3:               PLUS            '+'
1,4-1,5:               NUMBER          '1'
1,5-1,6:               NEWLINE         '\n'
2,0-2,0:               ENDMARKER       ''
```

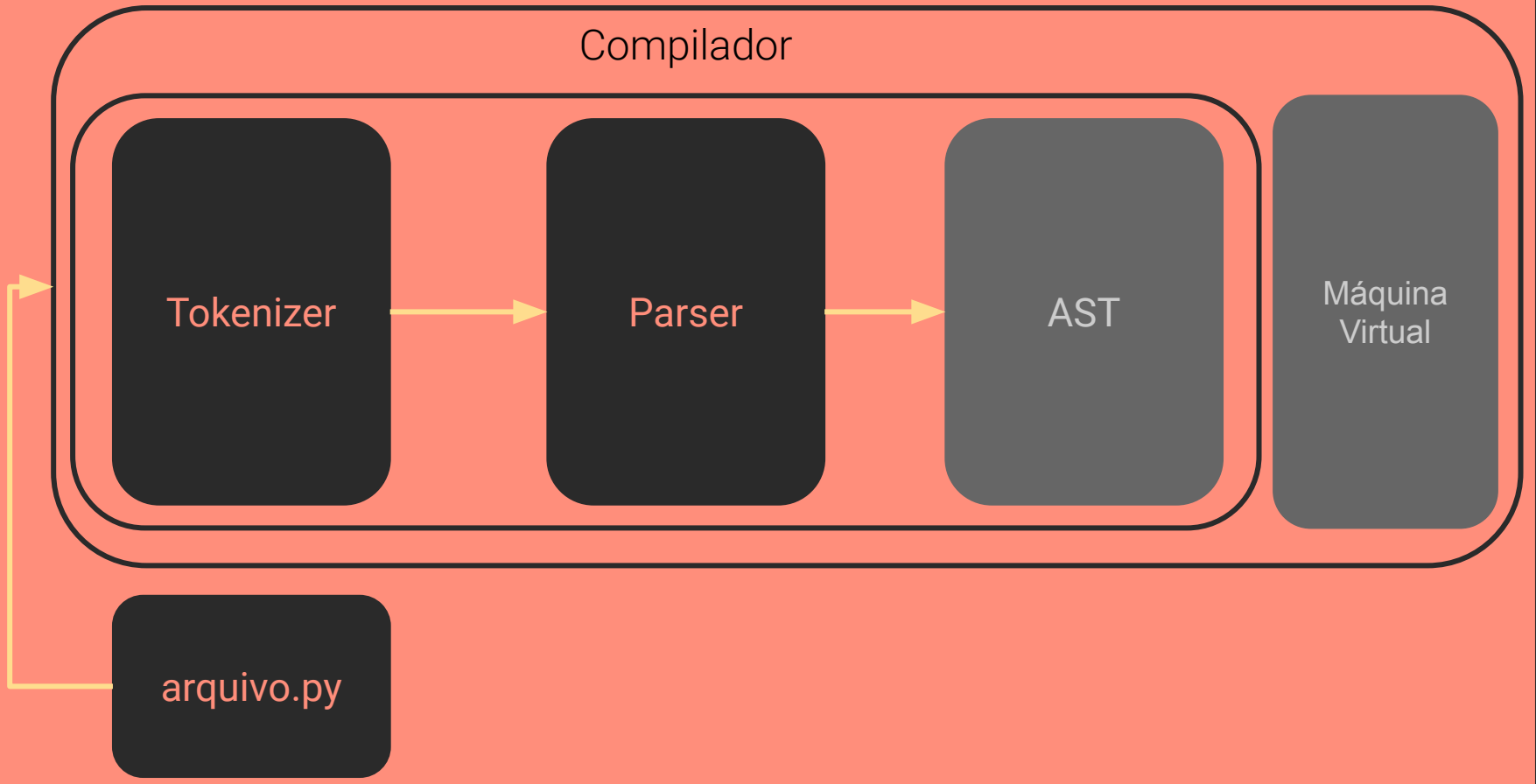
Análise léxica



https://github.com/python/cpython/blob/main/Grammar/Tokens

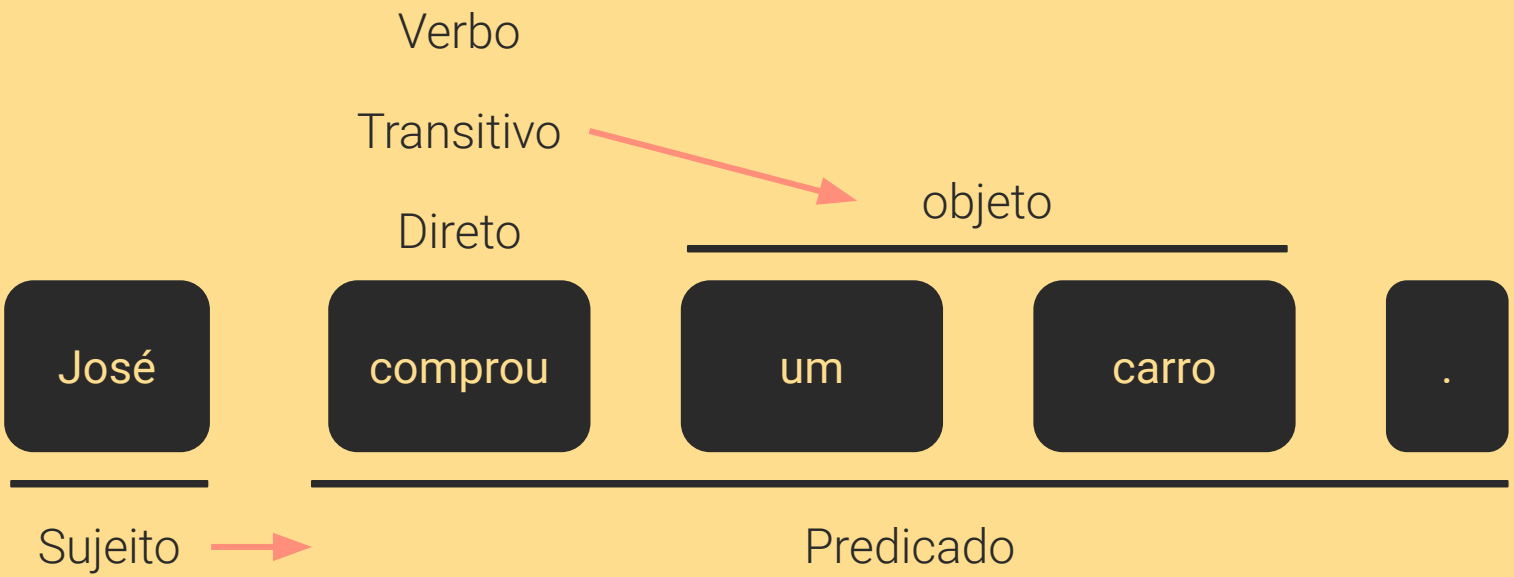
O início do interpretador

CPython



O parser

A faze de parsing é onde o Python vai analisar os tokens e dar sentido as operações. No português, por exemplo, é a análise sintática. Onde deixamos de olhar palavras independentes, mas dar sentido a frase toda.



A gramática

O que dá sentido a linguagem é sua gramática.

A ou B

```
statement: assignment | expr | if_statement
expr: expr '+' term | expr '-' term | term
term: term '*' atom | term '/' atom | atom
atom: NAME | NUMBER | '(' expr ')'
assignment: target '=' expr
target: NAME
if_statement: 'if' expr ':' statement
```

A e B

https://medium.com/@gvanrossum_83706/peg-parsers-7ed72462f97c

A gramática

O que dá sentido a linguagem é sua gramática.

```
statement: assignment | expr | if_statement
expr: expr '+' term | expr '-' term | term
term: term '*' atom | term '/' atom | atom
atom: NAME | NUMBER | '(' expr ')'
assignment: target '=' expr
target: NAME
if_statement: 'if' expr ':' statement
```

https://medium.com/@gvanrossum_83706/peg-parsers-7ed72462f97c

A gramática

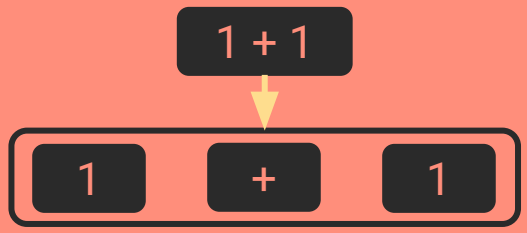
O que dá sentido a linguagem é sua gramática.

```
sum[expr_ty]:
    | a=sum '+' b=term { _PyAST_BinOp(a, Add, b, EXTRA) }
    | a=sum '-' b=term { _PyAST_BinOp(a, Sub, b, EXTRA) }
    | term
```

https://github.com/python/cpython/blob/main/Grammar/python.gram

A gramática

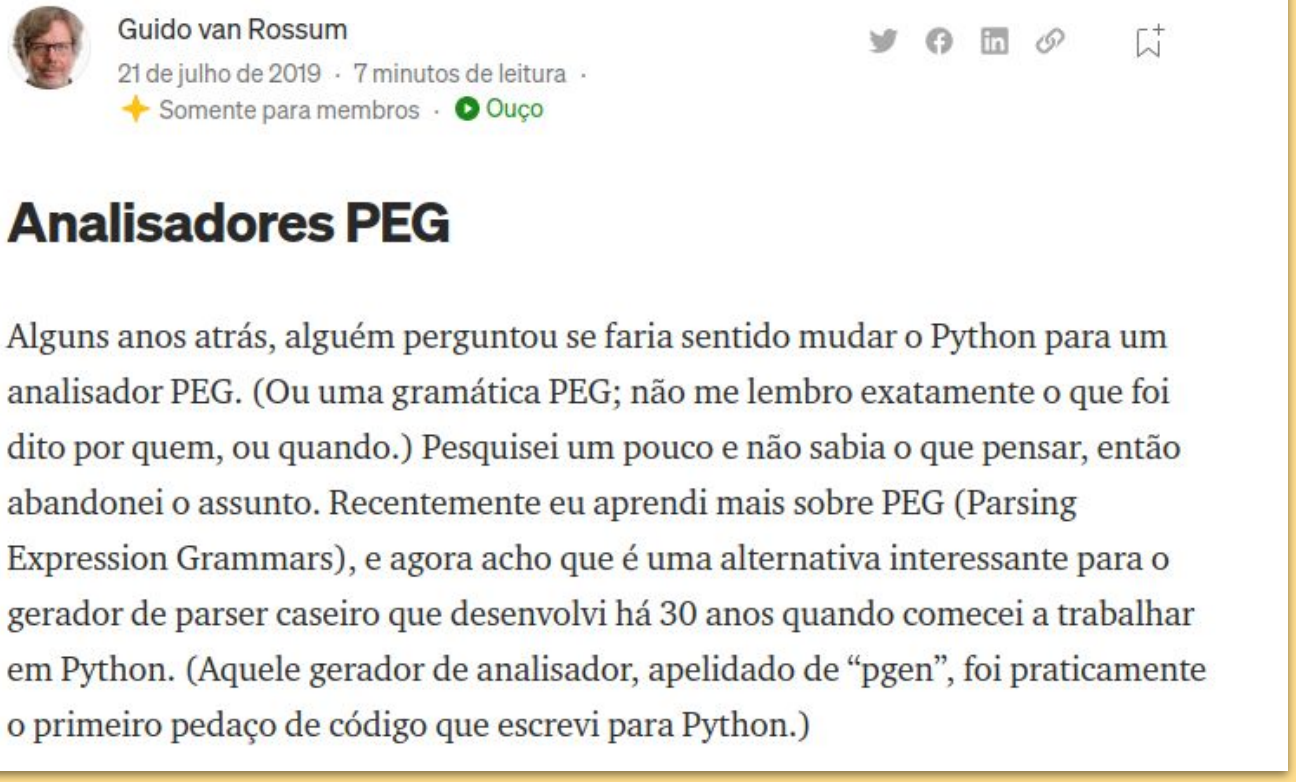
O que dá sentido a linguagem é sua gramática.




```
sum[expr_ty]:
    | a=sum '+' b=term { _PyAST_BinOp(a, Add, b, EXTRA) }
    | a=sum '-' b=term { _PyAST_BinOp(a, Sub, b, EXTRA) }
    | term
```

https://github.com/python/cpython/blob/main/Grammar/python.gram

O parser



Guido van Rossum

21 de julho de 2019 · 7 minutos de leitura ·

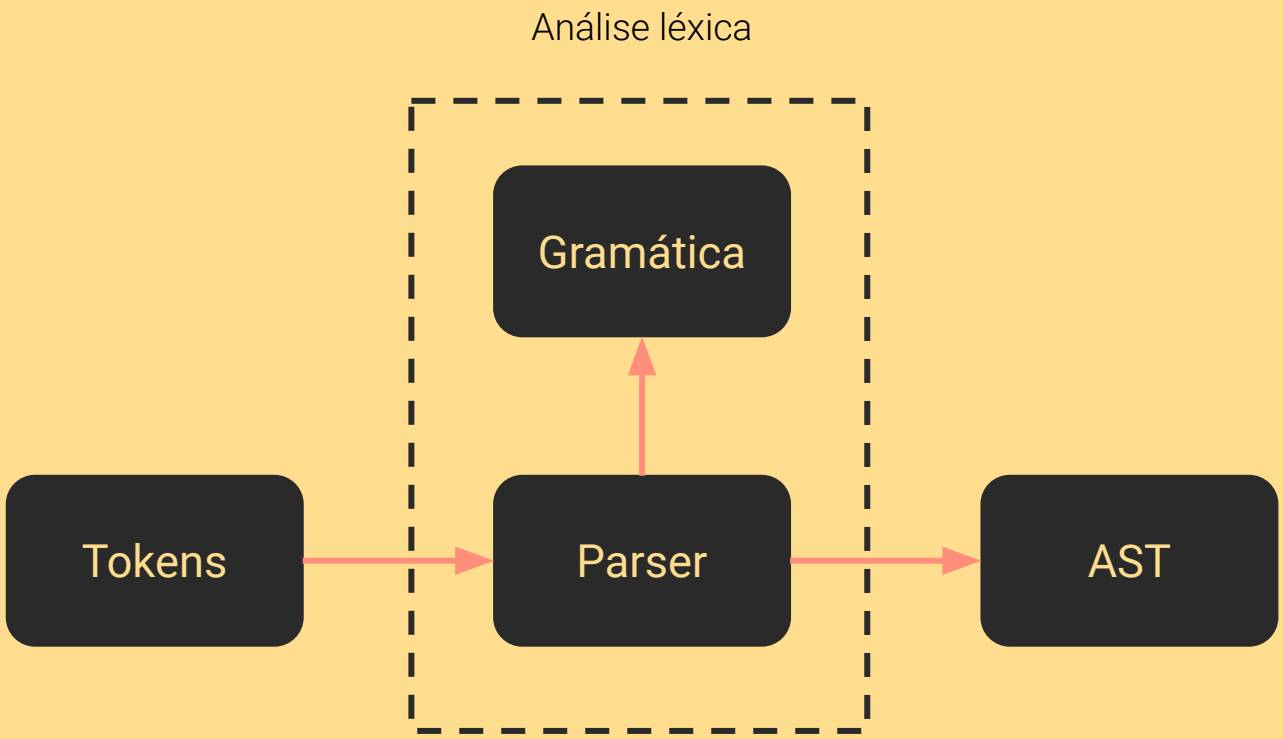
Somente para membros · Ouço

Analisadores PEG

Alguns anos atrás, alguém perguntou se faria sentido mudar o Python para um analisador PEG. (Ou uma gramática PEG; não me lembro exatamente o que foi dito por quem, ou quando.) Pesquisei um pouco e não sabia o que pensar, então abandonei o assunto. Recentemente eu aprendi mais sobre PEG (Parsing Expression Grammars), e agora acho que é uma alternativa interessante para o gerador de parser caseiro que desenvolvi há 30 anos quando comecei a trabalhar em Python. (Aquele gerador de analisador, apelidado de "pgen", foi praticamente o primeiro pedaço de código que escrevi para Python.)

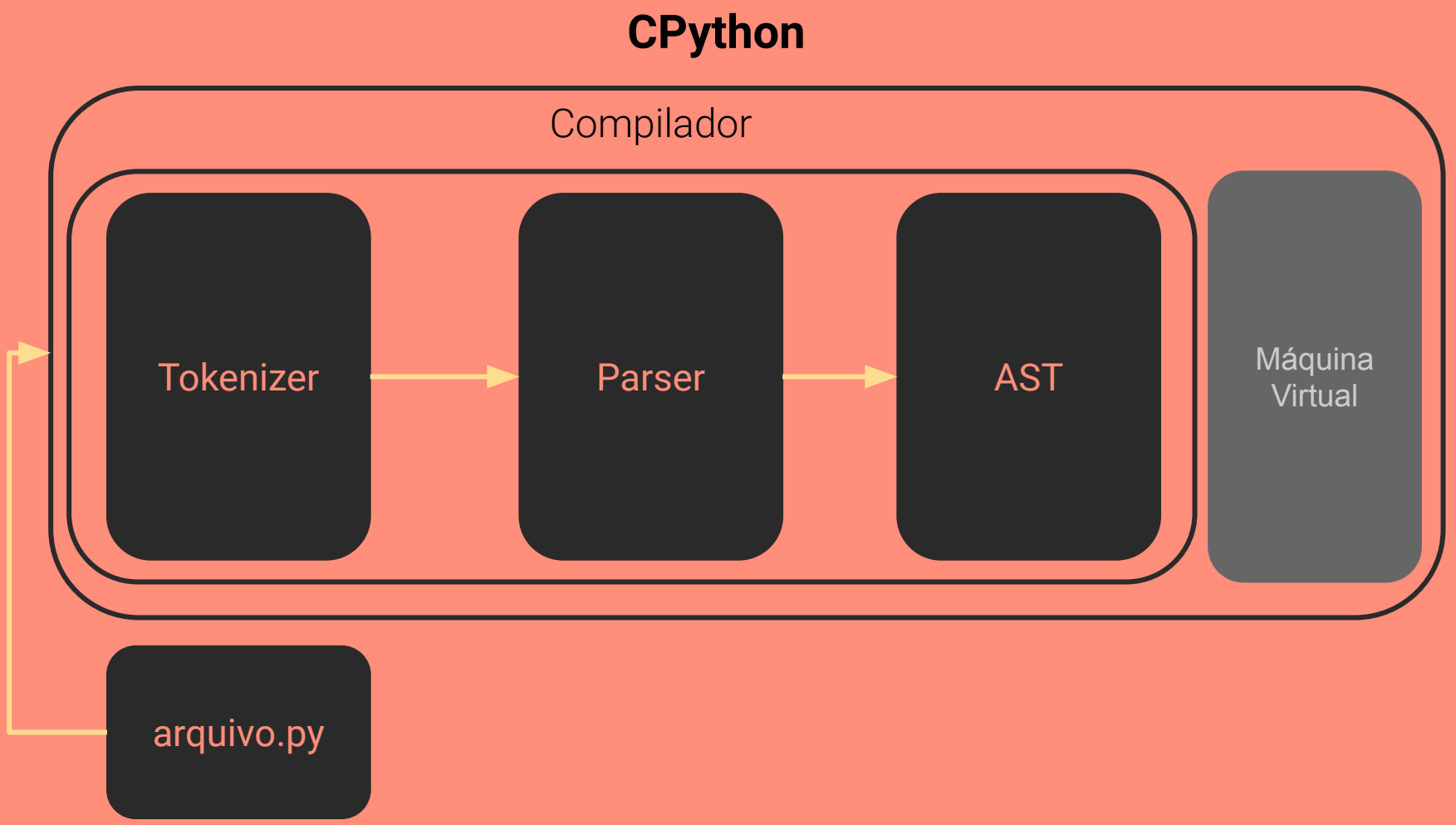
https://medium.com/@gvanrossum_83706/peg-parsers-7ed72462f97c

Análise léxica



https://github.com/python/cpython/blob/main/Python/Python-ast.c#L2631-L2663
https://github.com/python/cpython/blob/main/Parser/Python.asdl

O início do interpretador

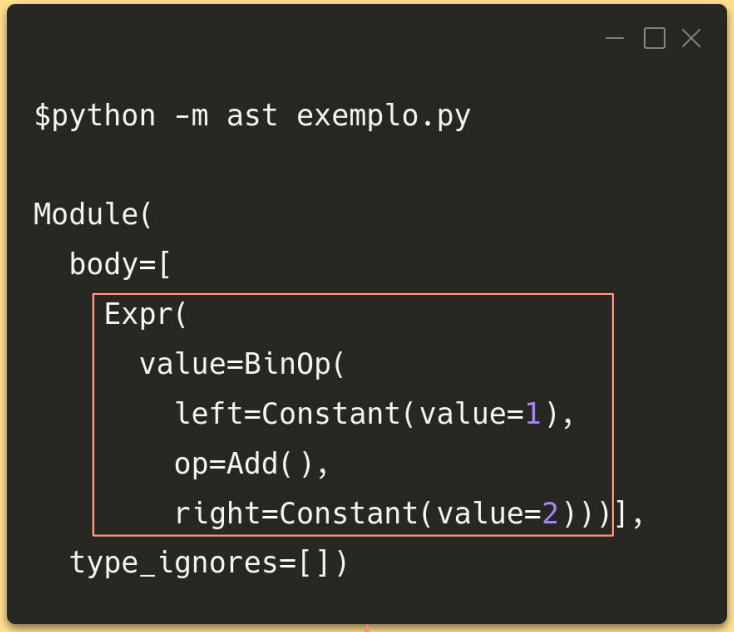


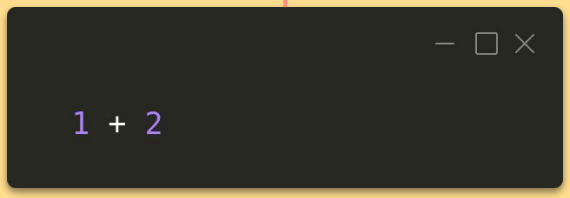
A AST

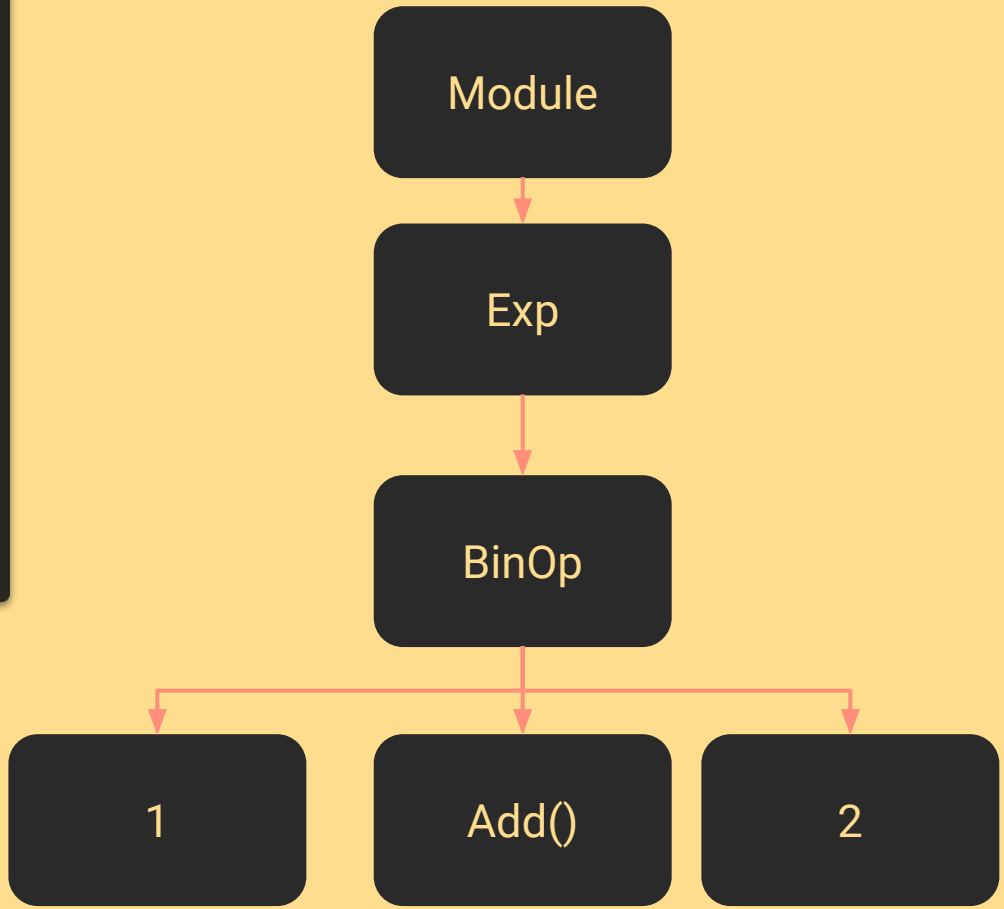
Abstract Syntax Tree, **Árvore Abstrata de Sintaxe**.

```
$python -m ast exemplo.py

Module(
  body=[
    Expr(
      value=BinOp(
        left=Constant(value=1),
        op=Add(),
        right=Constant(value=2)))],
  type_ignores=[])
```

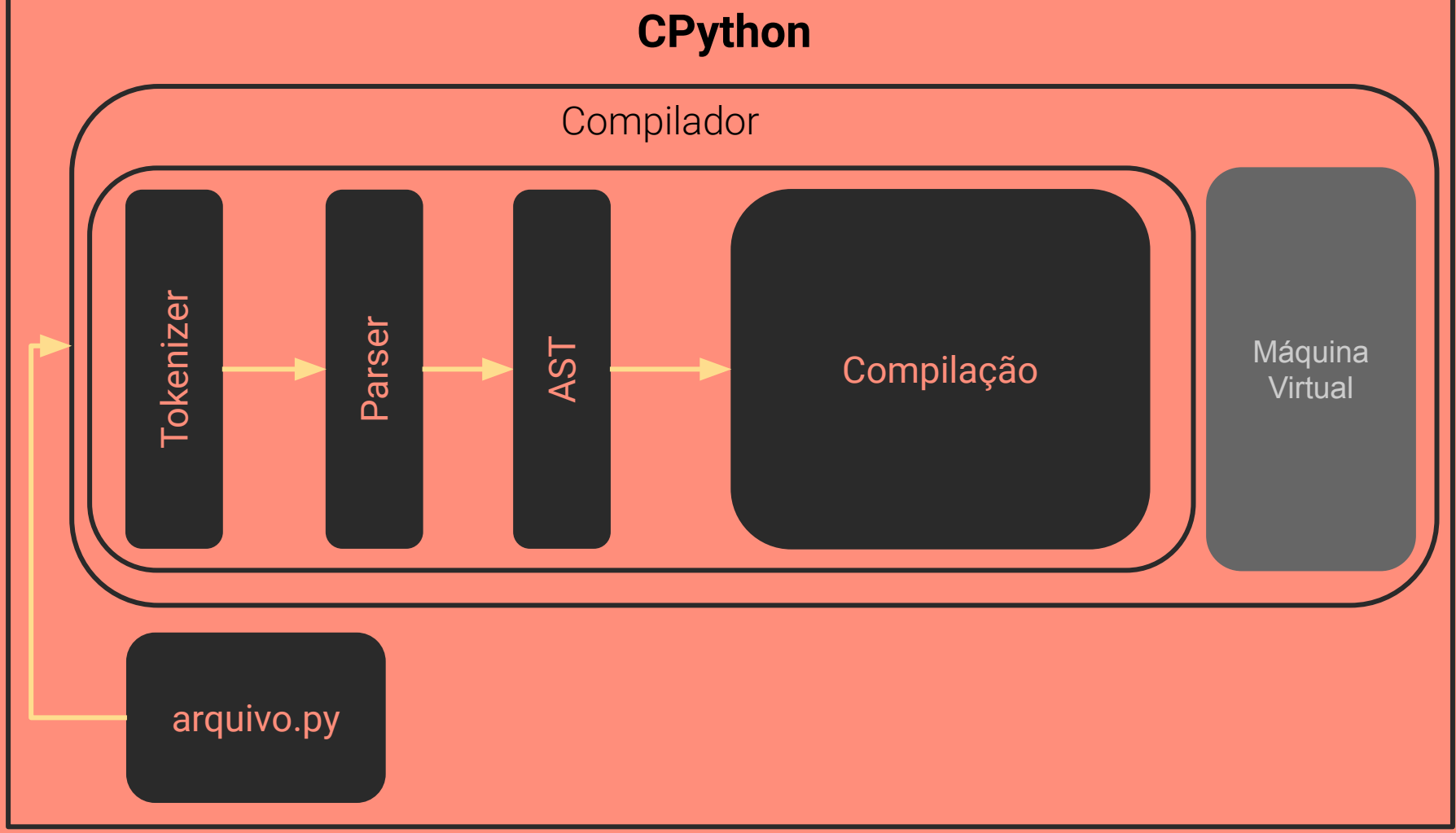
A AST


$python -m ast exemplo.py
Module(
body=[
Expr(
value=BinOp(
left=Constant(value=1),
op=Add(),
right=Constant(value=2)))],
type_ignores=[])

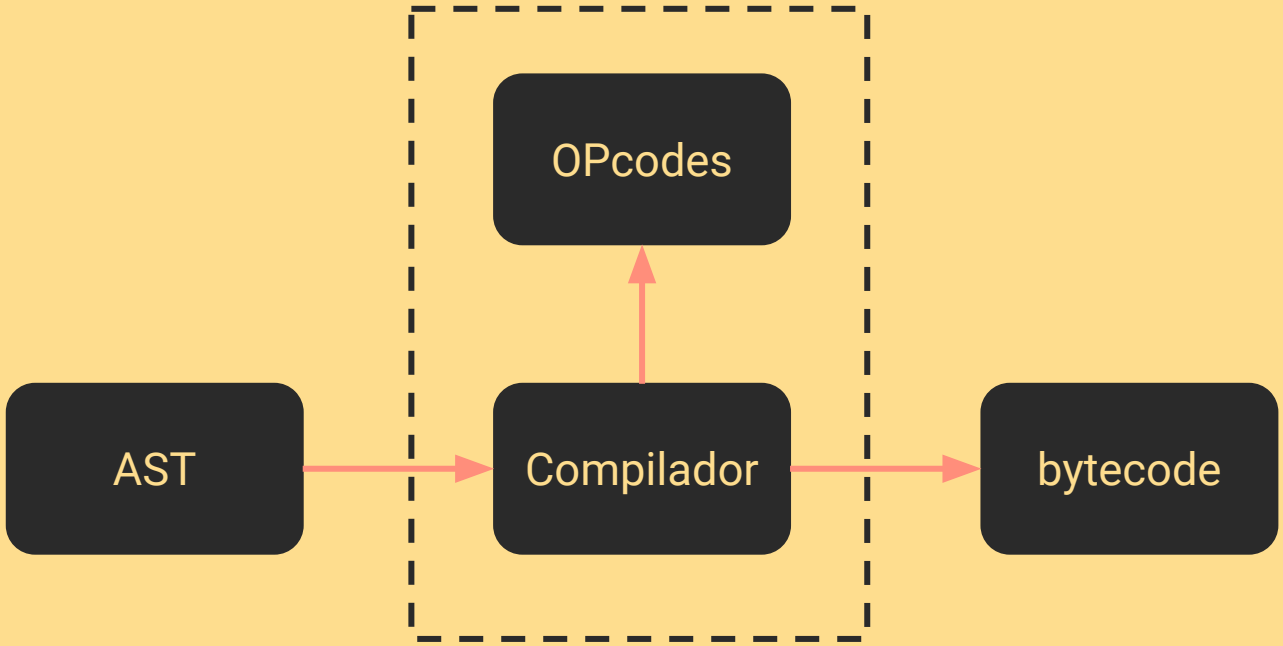

1 + 2


Module
Exp
BinOp
1
Add()
2

O início do interpretador

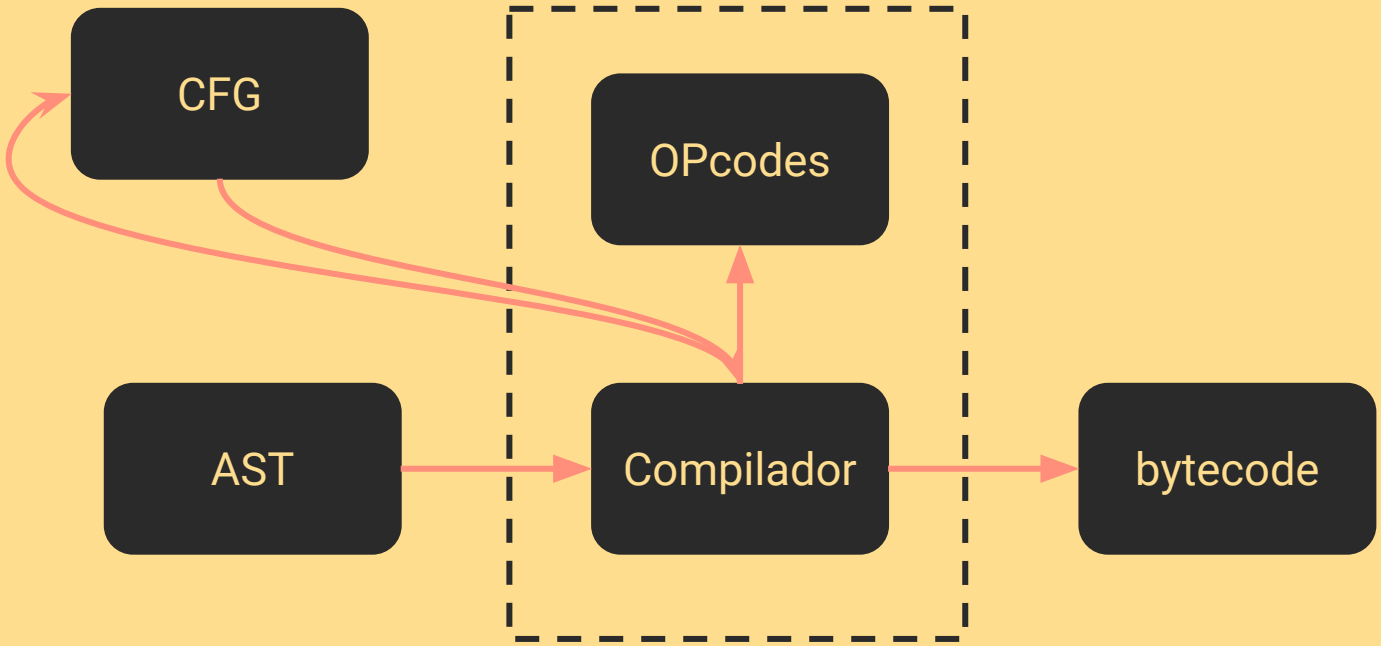


A fase de compilação



https://github.com/python/cpython/blob/main/Python/compile.c

A fase de compilação



https://github.com/python/cpython/blob/main/Python/compile.c

Tá, mas qual a cara desse bytecode?

O bytecode gerado pelo python, são os arquivos com extensão **.pyc**.

```
1 \A7
2
3 \00\00\00\00\A9j c \00\00\00\E3\00\00\00\00\00\00\00\00\00\00\00 \00\00\00\00\00\00
  \00\F3 \00\00\00\97\00d\00S\00) N\A9\00r \00\00\00\F3\00\00\00\00\FA
4 exemplo.py\FA <module>r \00\00\00 \00\00\00s
5 \00\00\00\F0    \D8\00 \80 r \00\00\00
```

Como compilar um .pyc?

```
python -m compileall exemplo.py
```

Bytecode de verdade!

```
$python -m dis exemplo.py

0           0 RESUME                   0

1           2 LOAD_CONST               0 (None)
            4 RETURN_VALUE
```

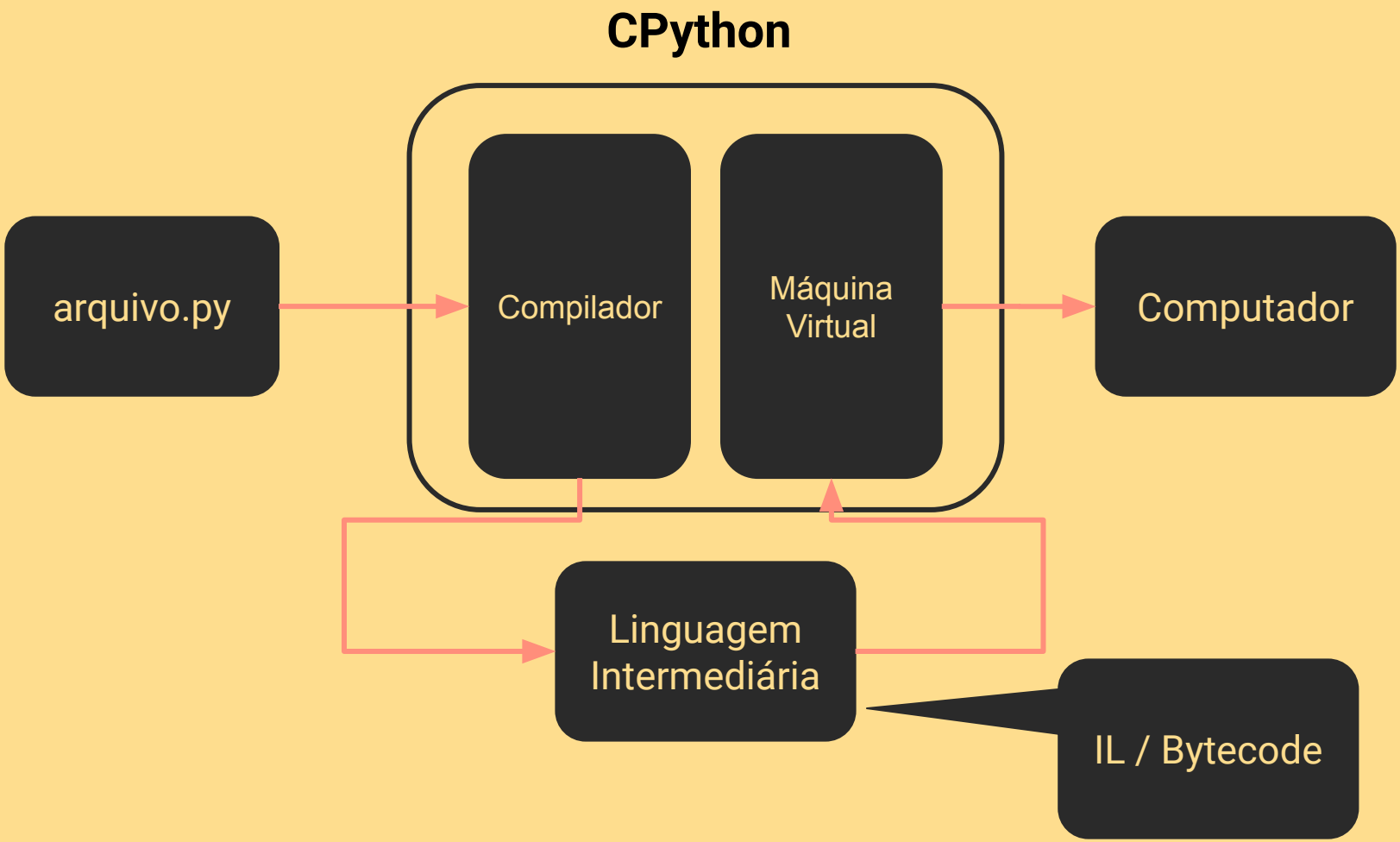
Bytecode de verdade!

Mostrar uma função no ipython

```
$python -m dis exemplo.py

0           0 RESUME                   0

1           2 LOAD_CONST               0 (None)
            4 RETURN_VALUE
```

https://docs.python.org/3/library/dis.html#python-bytecode-instructions

O CPython


CPython
arquivo.py
Compilador
Máquina Virtual
Computador
Linguagem Intermediária
IL / Bytecode

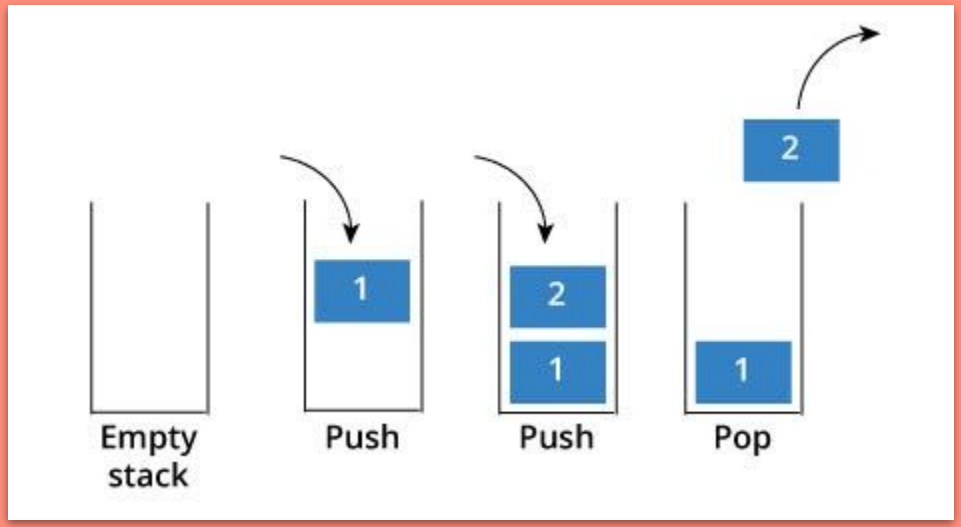
A máquina virtual (VM)

A VM do Python é uma máquina virtual no formato de Stack Machine. Um processador implementado em código. Stack, em português, quer dizer Pilha.

Uma pilha

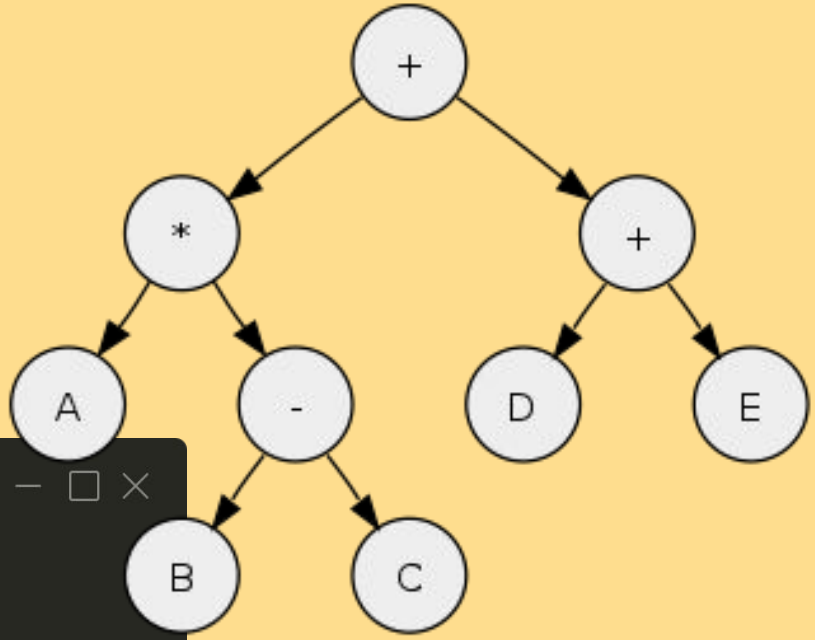
Basicamente, uma pilha só tem duas operações.

- PUSH: Colocar no topo
- POP: Remove do topo



https://medium.com/swlh/stacks-and-queues-simplified-ef0f838fc534

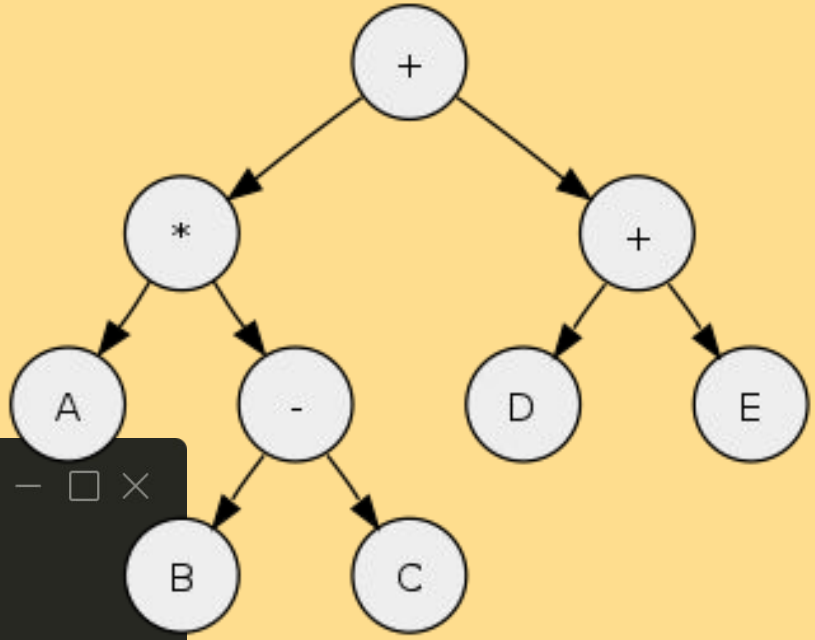
Stack Machine



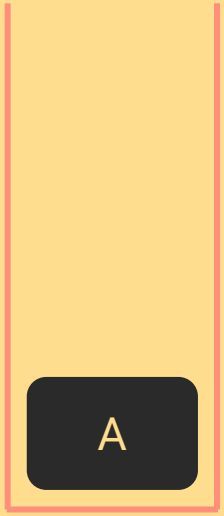

```
push A          #           A
push B          #     B     A
push C          # C   B     A
subtract        #     B-C   A
multiply        #           A*(B-C)
push D          #     D     A*(B-C)
push E          # E   D     A*(B-C)
add             #     D+E   A*(B-C)
add             #           A*(B-C)+(D+E)
```

https://en.wikipedia.org/wiki/Stack_machine

Stack Machine

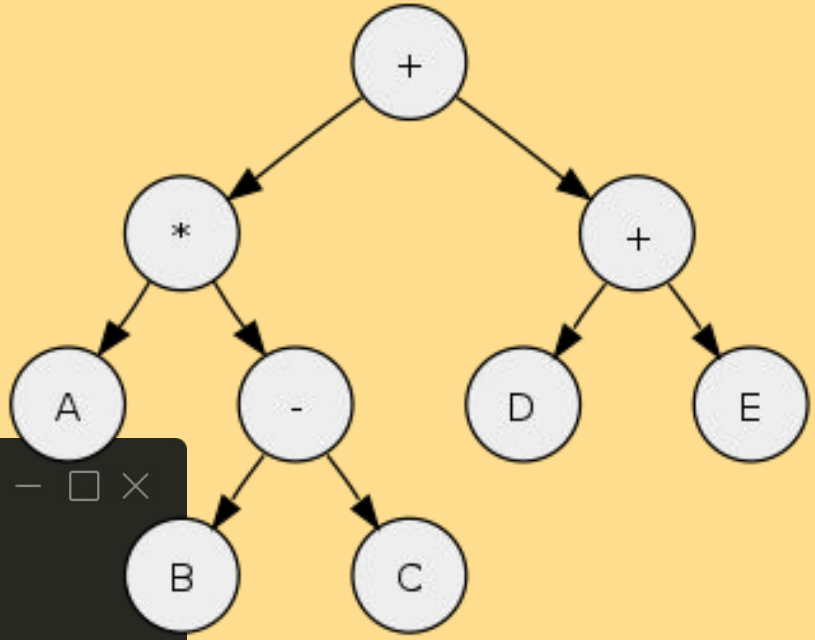



```
push A          #           A
push B          #     B     A
push C          # C   B     A
subtract        #     B-C   A
multiply        #           A*(B-C)
push D          #     D     A*(B-C)
push E          # E   D     A*(B-C)
add             #     D+E   A*(B-C)
add             #           A*(B-C)+(D+E)
```

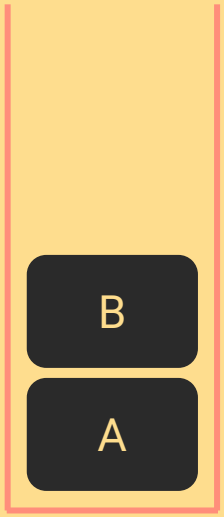


https://en.wikipedia.org/wiki/Stack_machine

Stack Machine

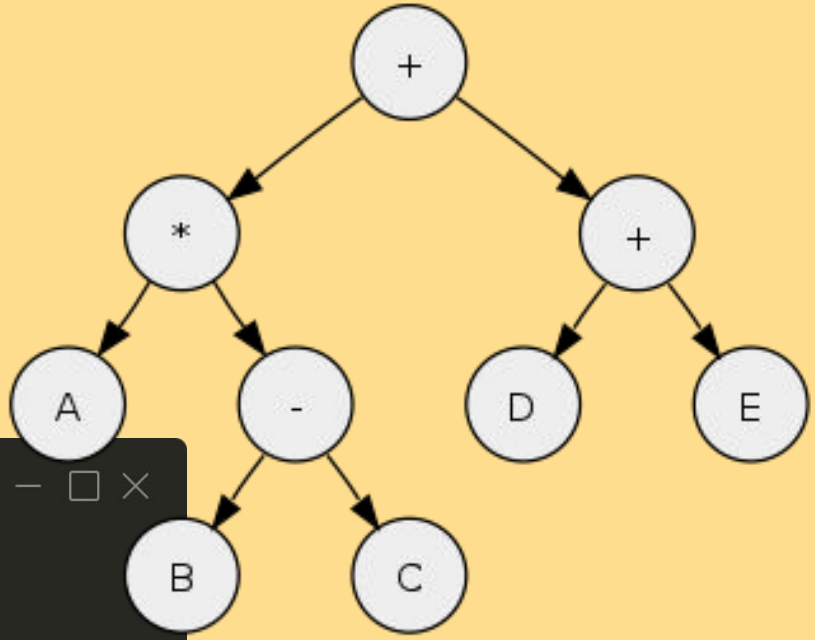



```
push A          #           A
push B          #     B     A
push C          # C   B     A
subtract        #     B-C   A
multiply        #           A*(B-C)
push D          #     D     A*(B-C)
push E          # E   D     A*(B-C)
add             #     D+E   A*(B-C)
add             #           A*(B-C)+(D+E)
```

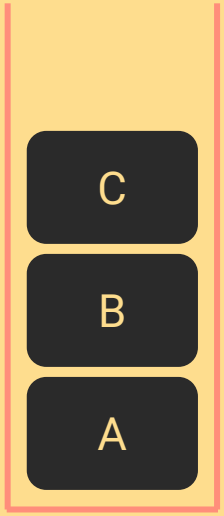


https://en.wikipedia.org/wiki/Stack_machine

Stack Machine

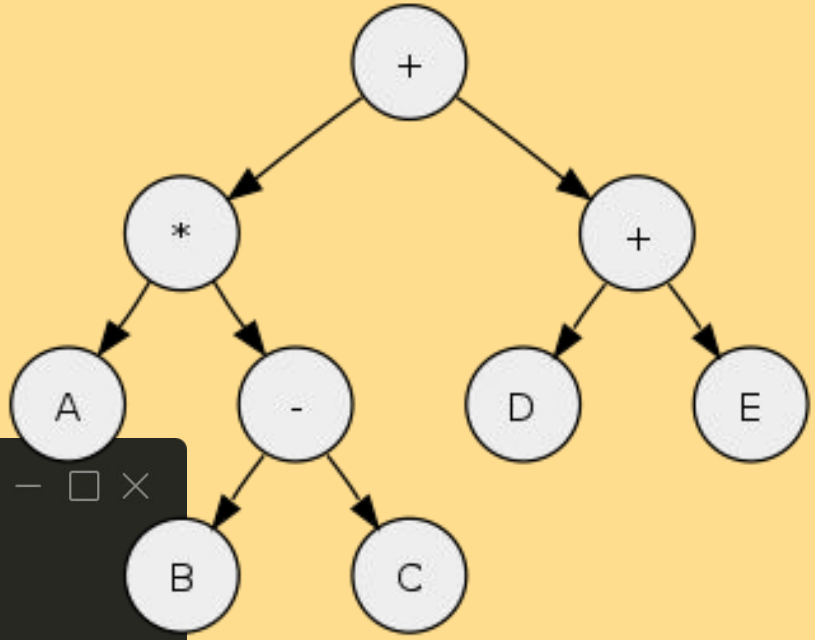



```
push A          #           A
push B          #     B     A
push C          # C   B     A
subtract        #     B-C   A
multiply        #           A*(B-C)
push D          #     D     A*(B-C)
push E          # E   D     A*(B-C)
add             #     D+E   A*(B-C)
add             #           A*(B-C)+(D+E)
```

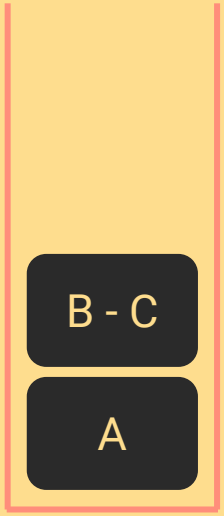


https://en.wikipedia.org/wiki/Stack_machine

Stack Machine

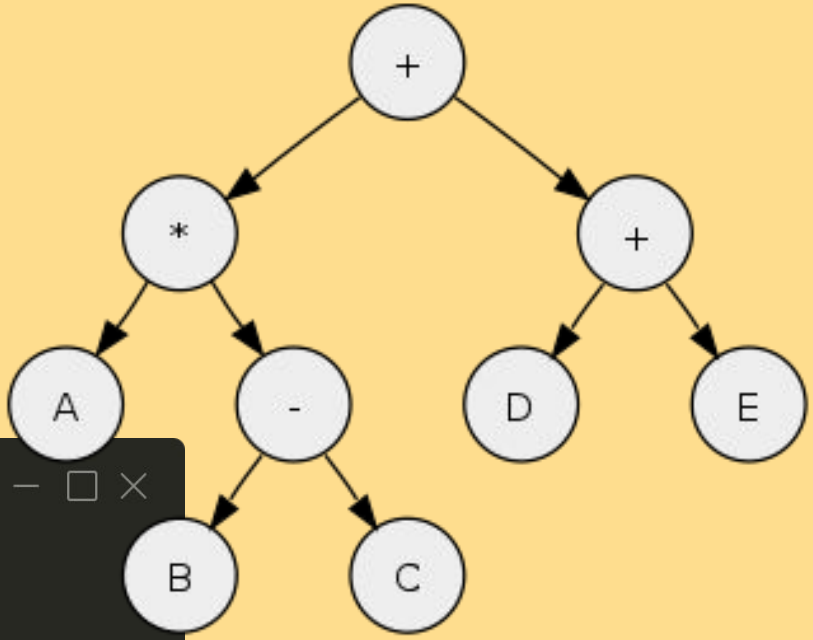



```
push A          #           A
push B          #     B     A
push C          # C   B     A
subtract        #     B-C   A
multiply        #           A*(B-C)
push D          #     D     A*(B-C)
push E          # E   D     A*(B-C)
add             #     D+E   A*(B-C)
add             #           A*(B-C)+(D+E)
```



https://en.wikipedia.org/wiki/Stack_machine

Stack Machine

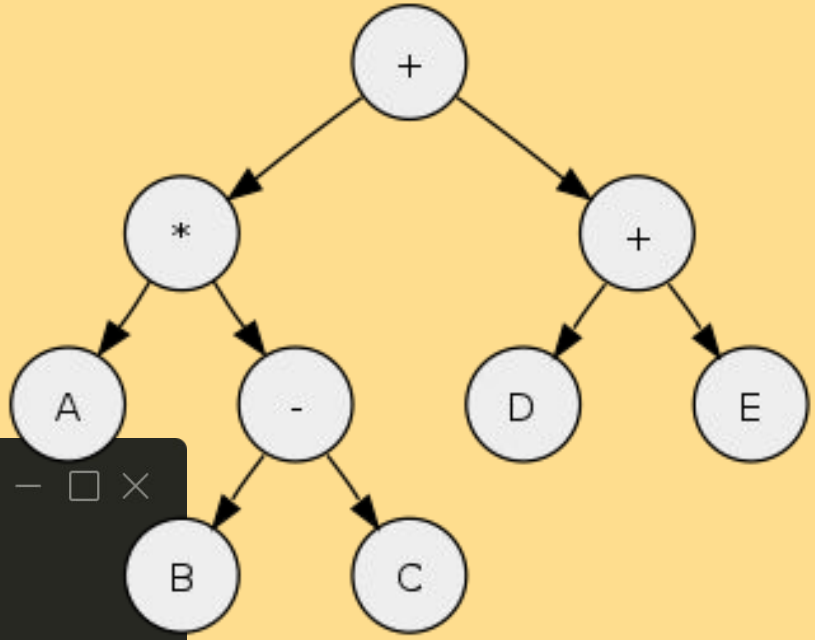



```
push A          #           A
push B          #     B     A
push C          # C   B     A
subtract        #     B-C   A
multiply        #           A*(B-C)
push D          #     D     A*(B-C)
push E          # E   D     A*(B-C)
add             #     D+E   A*(B-C)
add             #           A*(B-C)+(D+E)
```

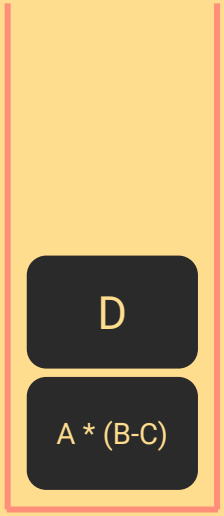
A * (B-C)

https://en.wikipedia.org/wiki/Stack_machine

Stack Machine

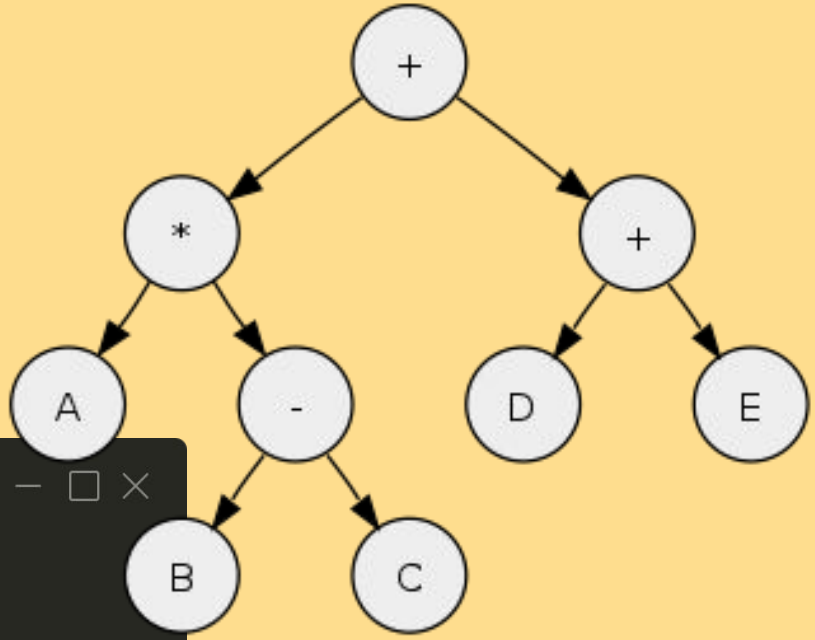



```
push A          #           A
push B          #     B     A
push C          # C   B     A
subtract        #     B-C   A
multiply        #           A*(B-C)
push D          #     D     A*(B-C)
push E          # E   D     A*(B-C)
add             #     D+E   A*(B-C)
add             #           A*(B-C)+(D+E)
```

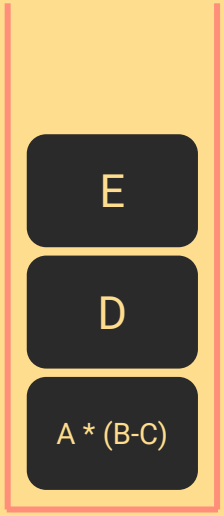


https://en.wikipedia.org/wiki/Stack_machine

Stack Machine

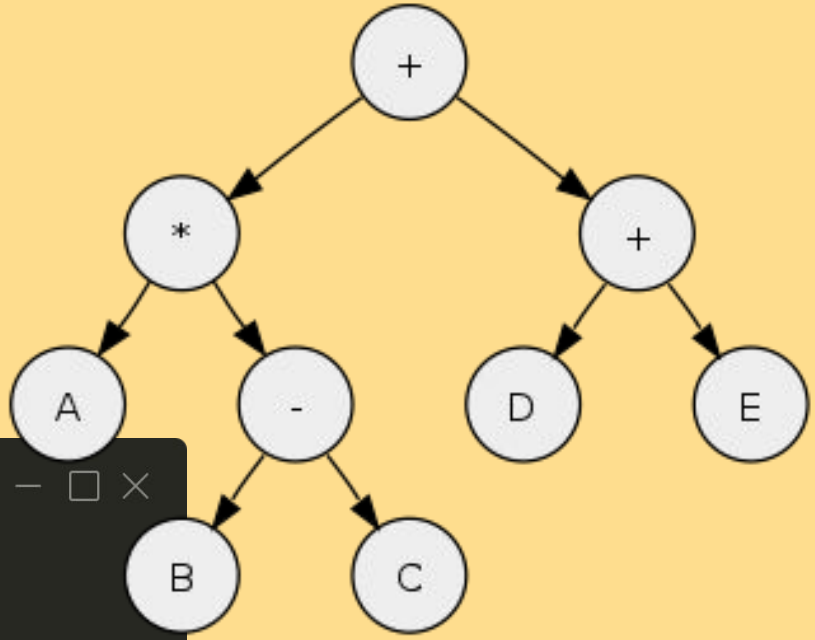



```
push A          #           A
push B          #     B     A
push C          # C   B     A
subtract        #     B-C   A
multiply        #           A*(B-C)
push D          #     D     A*(B-C)
push E          # E   D     A*(B-C)
add             #     D+E   A*(B-C)
add             #           A*(B-C)+(D+E)
```

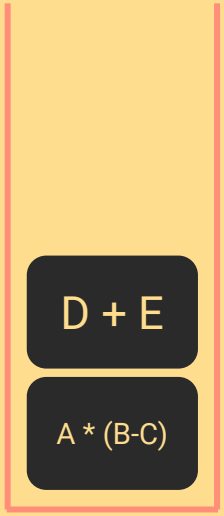


https://en.wikipedia.org/wiki/Stack_machine

Stack Machine

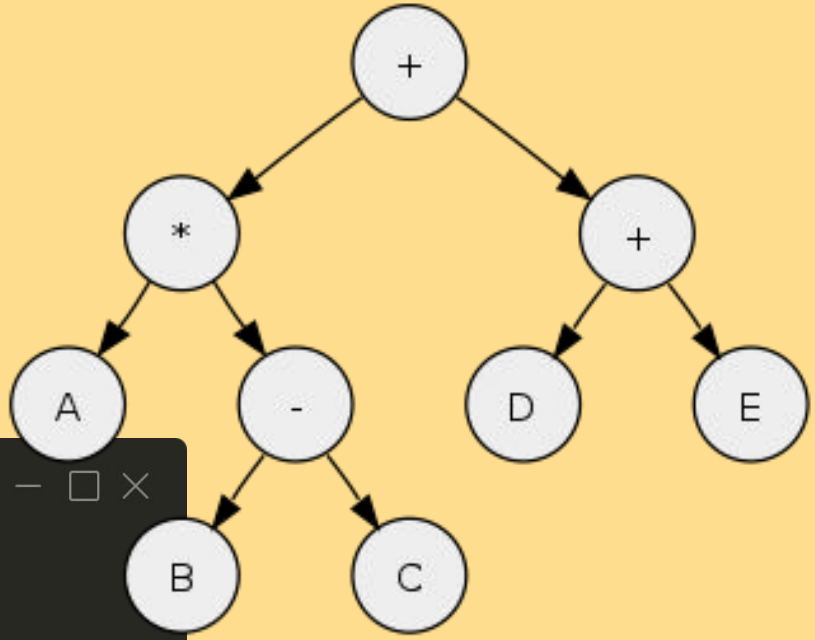



```
push A          #           A
push B          #     B     A
push C          # C   B     A
subtract        #     B-C   A
multiply        #           A*(B-C)
push D          #     D     A*(B-C)
push E          # E   D     A*(B-C)
add             #     D+E   A*(B-C)
add             #           A*(B-C)+(D+E)
```



https://en.wikipedia.org/wiki/Stack_machine

Stack Machine




```
push A          #           A
push B          #     B     A
push C          # C   B     A
subtract        #     B-C   A
multiply        #           A*(B-C)
push D          #     D     A*(B-C)
push E          # E   D     A*(B-C)
add             #     D+E   A*(B-C)
add             #           A*(B-C)+(D+E)
```

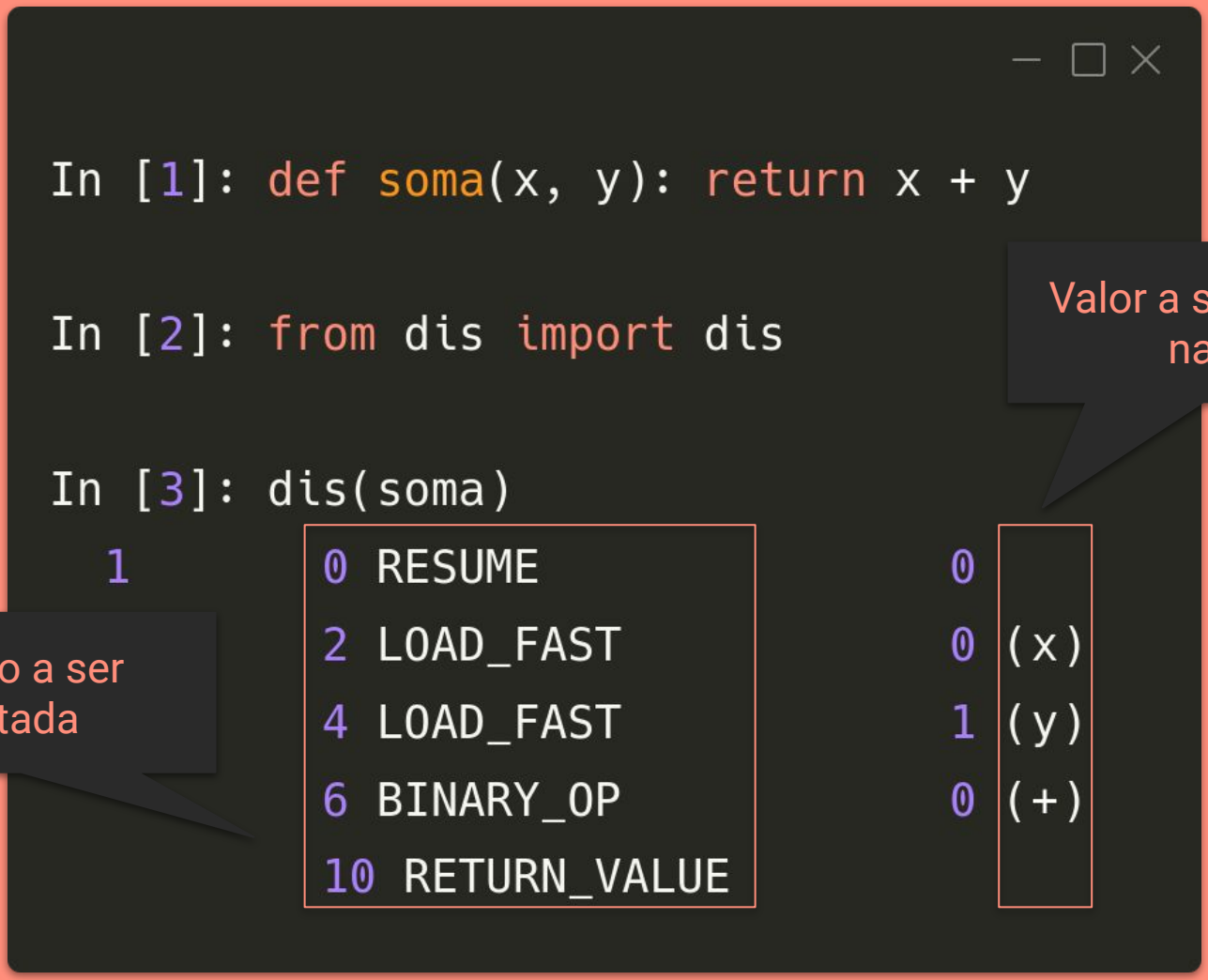
A * (B-C) + (D + E)

https://en.wikipedia.org/wiki/Stack_machine

A máquina do Python

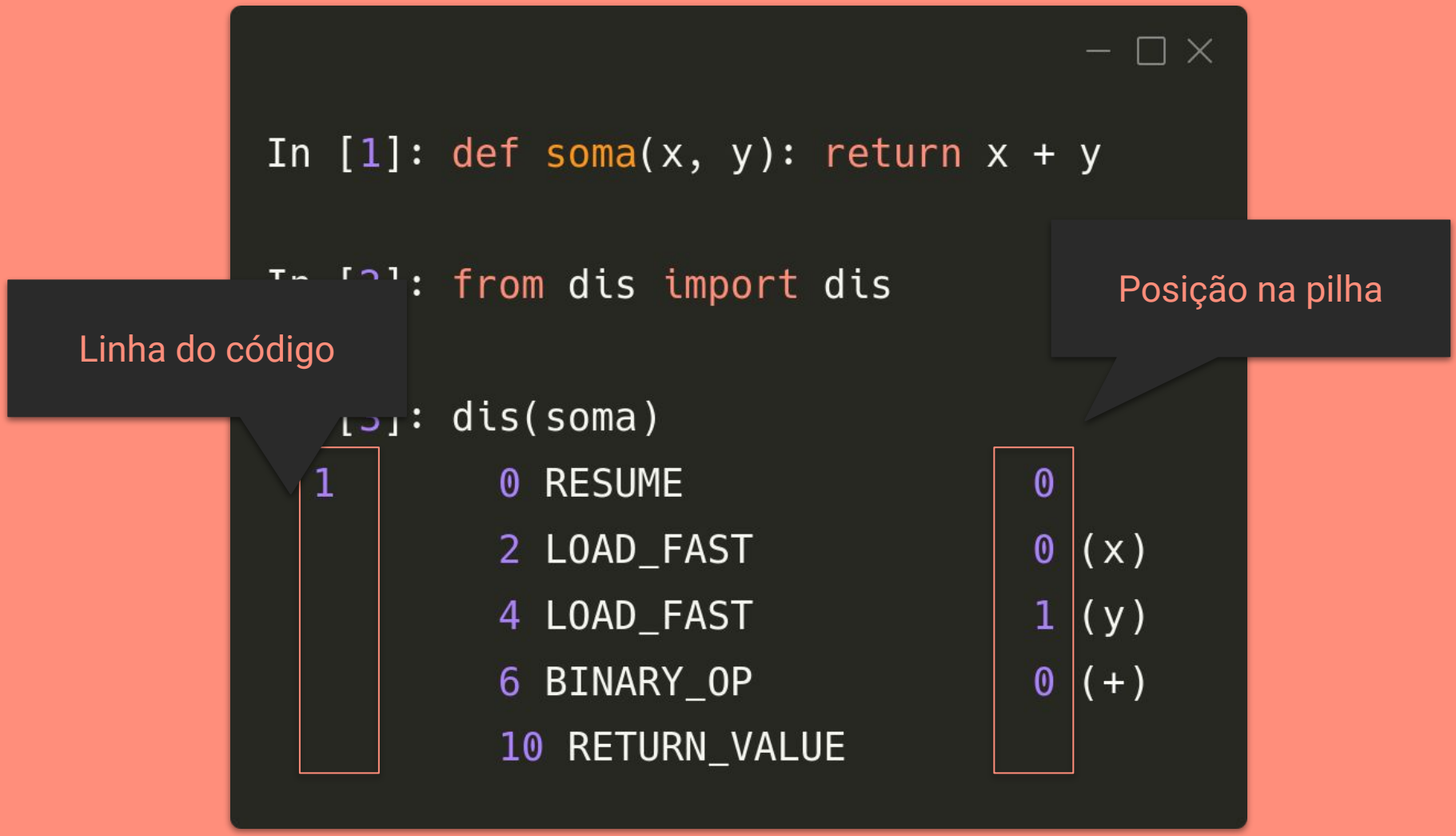
```
In [1]: def soma(x, y): return x + y

In [2]: from dis import dis

In [3]: dis(soma)
  1           0 RESUME                   0
              2 LOAD_FAST                0 (x)
              4 LOAD_FAST                1 (y)
              6 BINARY_OP                0 (+)
             10 RETURN_VALUE
```

A máquina do Python



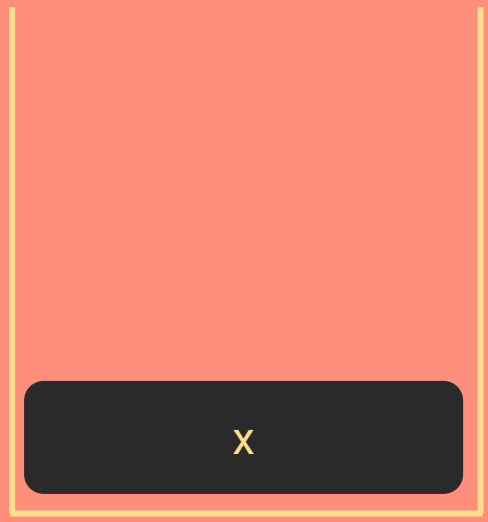
https://docs.python.org/3.11/library/dis.html#python-bytecode-instructions

A máquina do Python



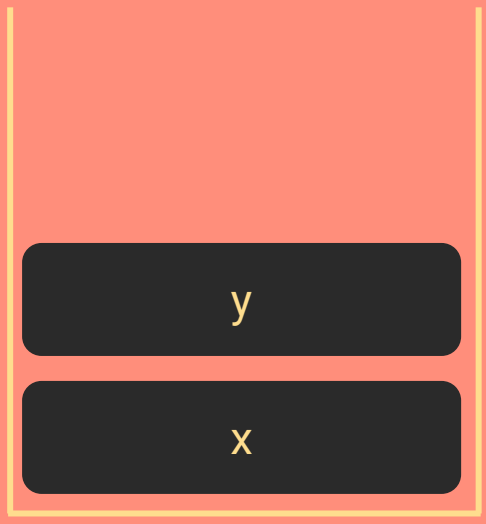
A máquina do Python

```
In [1]: def soma(x, y): return x + y

In [2]: from dis import dis

In [3]: dis(soma)
  1           0 RESUME                   0
              2 LOAD_FAST                0 (x)
              4 LOAD_FAST                1 (y)
              6 BINARY_OP                0 (+)
             10 RETURN_VALUE
```



A máquina do Python

```
In [1]: def soma(x, y): return x + y

In [2]: from dis import dis

In [3]: dis(soma)
  1           0 RESUME                   0
              2 LOAD_FAST                0 (x)
              4 LOAD_FAST                1 (y)
              6 BINARY_OP                0 (+)
             10 RETURN_VALUE
```



A máquina do Python

```
In [1]: def soma(x, y): return x + y

In [2]: from dis import dis

In [3]: dis(soma)
  1           0 RESUME                   0
              2 LOAD_FAST                0 (x)
              4 LOAD_FAST                1 (y)
              6 BINARY_OP                0 (+)
             10 RETURN_VALUE
```

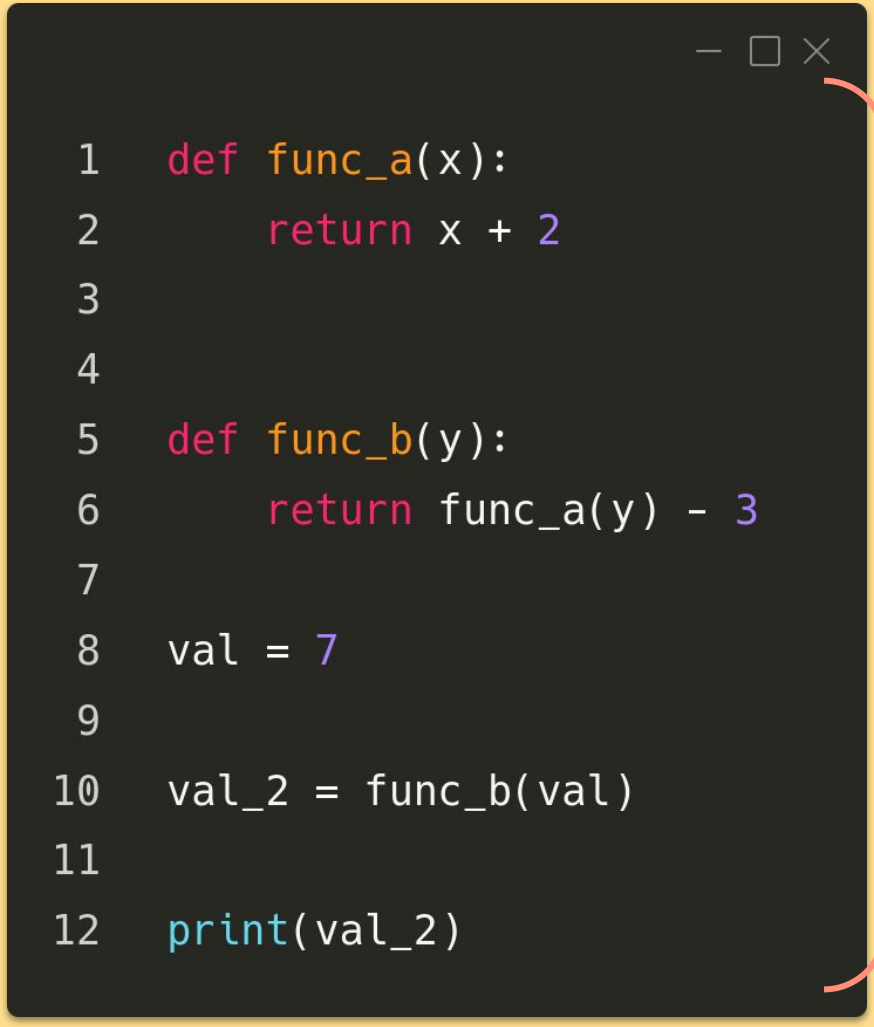
https://github.com/python/cpython/blob/main/Python/ceval.c

A máquina do Python

```
In [1]: def soma(x, y): return x + y

In [2]: from dis import dis

In [3]: dis(soma)
  1           0 RESUME                   0
              2 LOAD_FAST                0 (x)
              4 LOAD_FAST                1 (y)
              6 BINARY_OP                0 (+)
             10 RETURN_VALUE
```

?

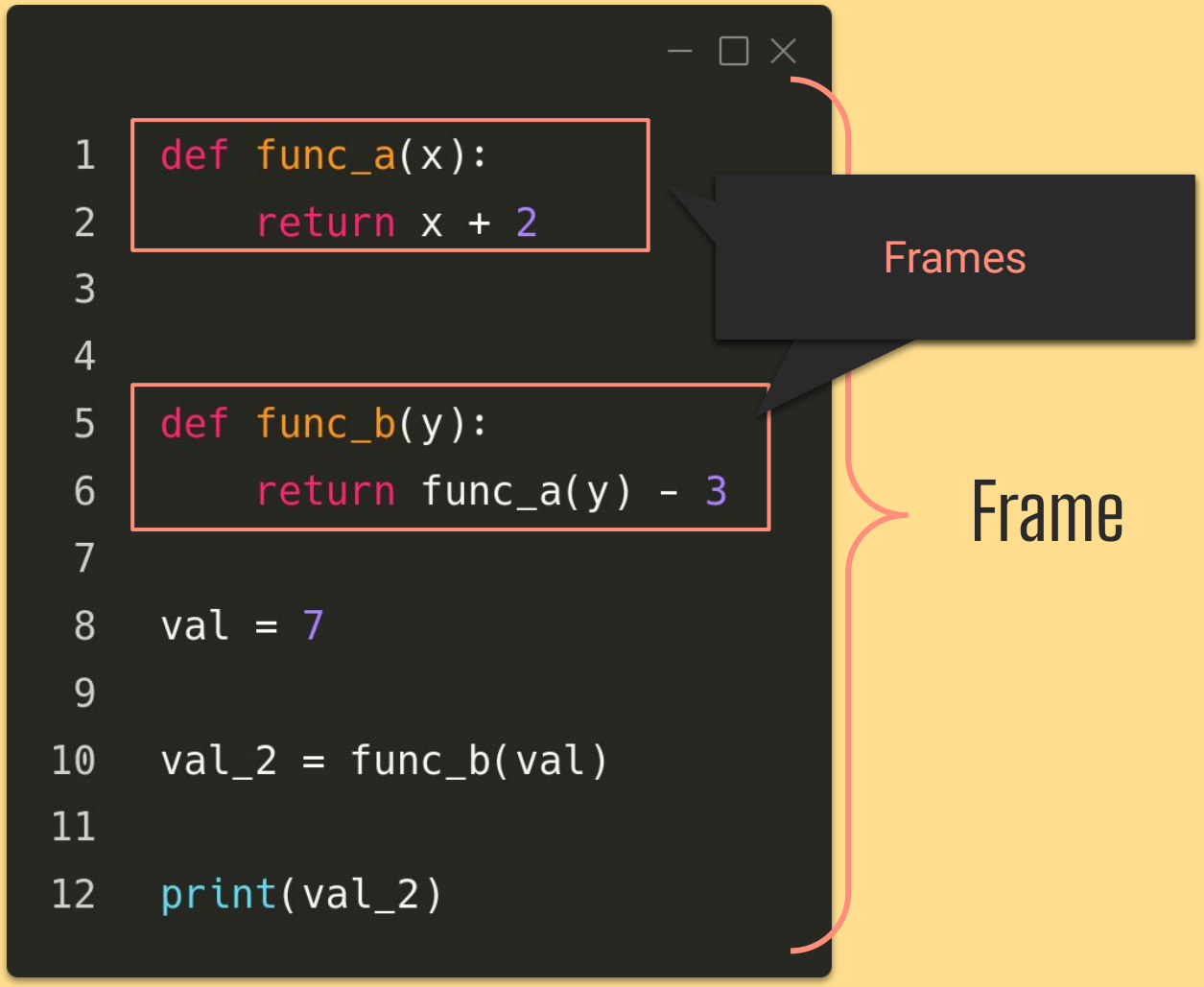
Frames



```
1   def func_a(x):
2       return x + 2
3
4
5   def func_b(y):
6       return func_a(y) - 3
7
8   val = 7
9
10  val_2 = func_b(val)
11
12  print(val_2)
```

Frame

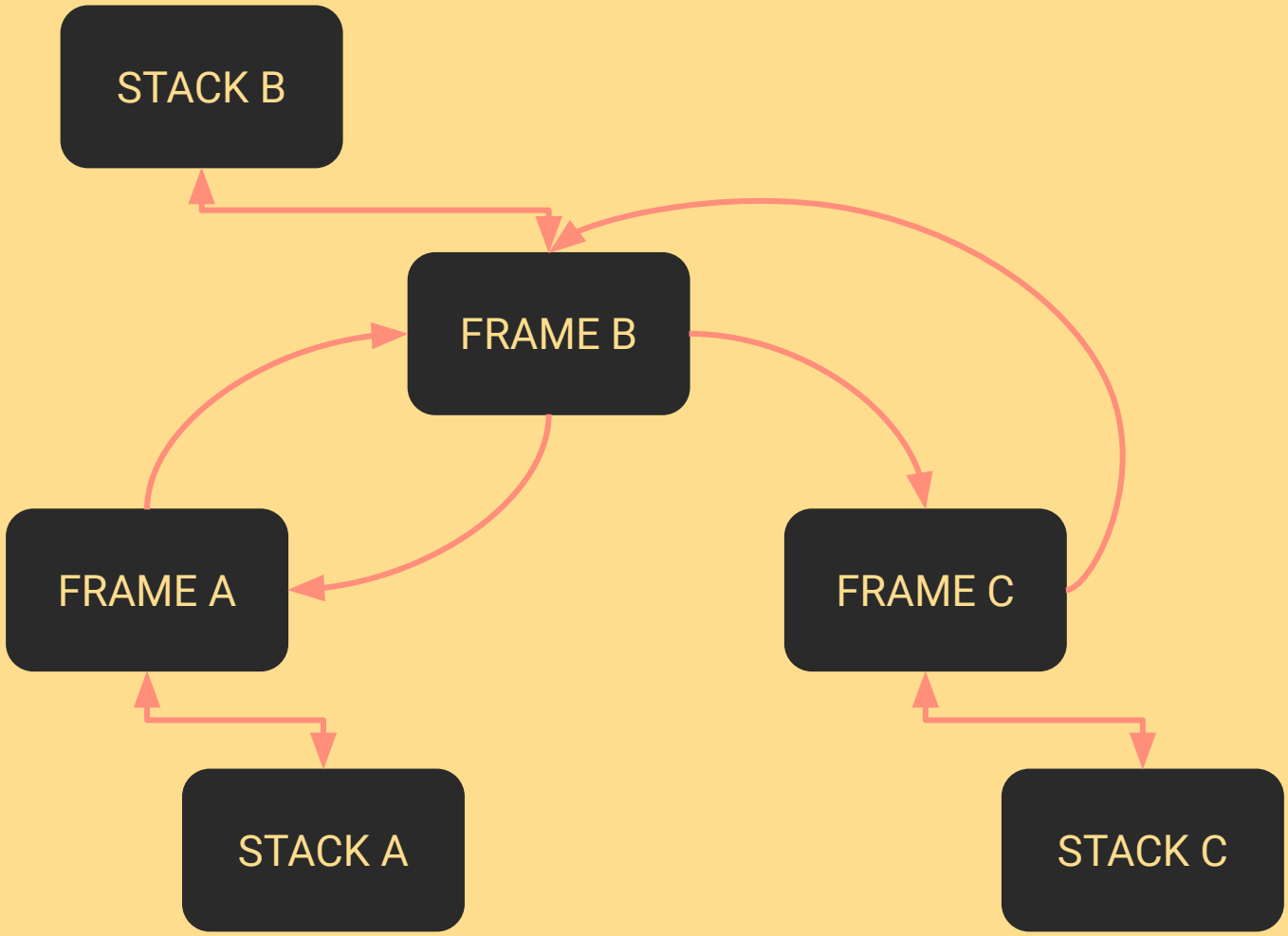
Frames


1 def func_a(x):
2 return x + 2
3
4
5 def func_b(y):
6 return func_a(y) - 3
7
8 val = 7
9
10 val_2 = func_b(val)
11
12 print(val_2)
Frames
Frame

Bytecode

```
python -m dis exemplo2.py
  0           0 RESUME                   0

  1           2 LOAD_CONST               0 (<code object func_a at 0x7f84b8ceaf50, file "exemplo2.py", line 1>)
              4 MAKE_FUNCTION            0
              6 STORE_NAME               0 (func_a)

  5           8 LOAD_CONST               1 (<code object func_b at 0x7f84b8cc7ad0, file "exemplo2.py", line 5>)
             10 MAKE_FUNCTION            0
             12 STORE_NAME               1 (func_b)

  8          14 LOAD_CONST               2 (7)
             16 STORE_NAME               2 (val)

 10          18 PUSH_NULL
             20 LOAD_NAME                1 (func_b)
             22 LOAD_NAME                2 (val)
             24 PRECALL                  1
             28 CALL                     1
             38 STORE_NAME               3 (val_2)

 12          40 PUSH_NULL
             42 LOAD_NAME                4 (print)
             44 LOAD_NAME                3 (val_2)
             46 PRECALL                  1
             50 CALL                     1
             60 POP_TOP
             62 LOAD_CONST               3 (None)
             64 RETURN_VALUE

Disassembly of <code object func_a at 0x7f84b8ceaf50, file "exemplo2.py", line 1>:
  1           0 RESUME                   0

  2           2 LOAD_FAST                0 (x)
              4 LOAD_CONST               1 (2)
              6 BINARY_OP                0 (+)
             10 RETURN_VALUE

Disassembly of <code object func_b at 0x7f84b8cc7ad0, file "exemplo2.py", line 5>:
  5           0 RESUME                   0

  6           2 LOAD_GLOBAL              1 (NULL + func_a)
             14 LOAD_FAST                0 (y)
             16 PRECALL                  1
             20 CALL                     1
             30 LOAD_CONST               1 (3)
             32 BINARY_OP               10 (-)
             36 RETURN_VALUE
```

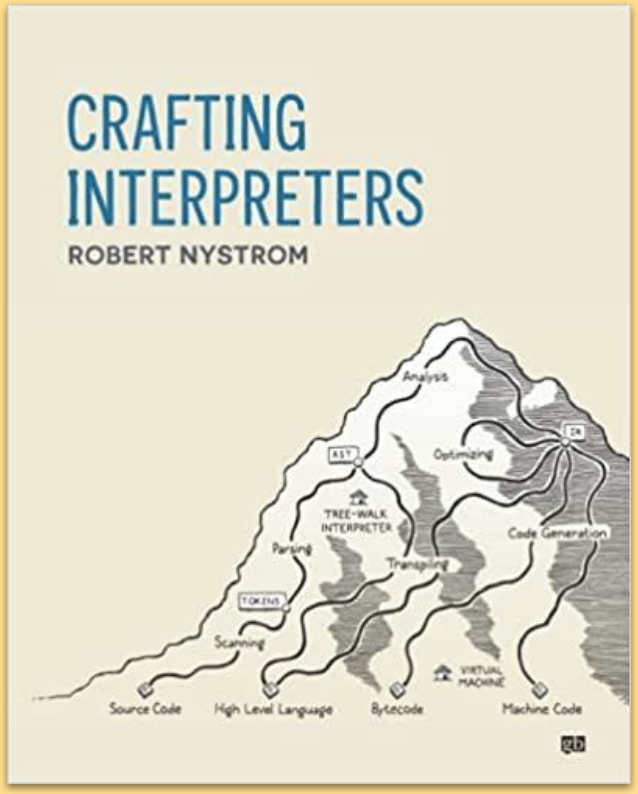
Frames

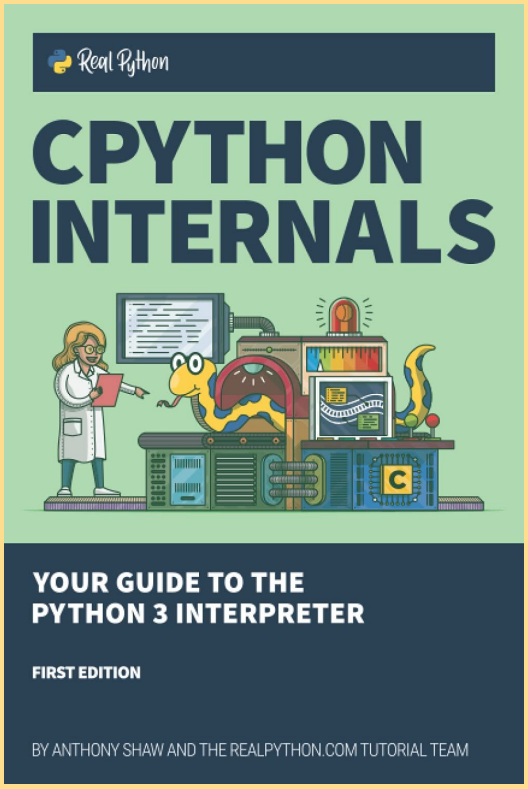


Perguntas?

UFA

Para saber mais





Coisas que me ajudaram a fazer essa live

- Victor Skvortsov - **Python behing the scenes**: https://tenthousandmeters.com/tag/python-behind-the-scenes/
- Palestra James Bennett (Pycon 2019): https://youtu.be/cSSpnq362Bk
- Palestra Sebastiaan Zeef (PyCon 2022): https://youtu.be/HYKGZunmF50
- Palestra Guido sobre o PEG (North Bay Python 2019): https://youtu.be/QppWTvh7_sI

picpay.me/dunossauro

apoia.se/livedepython

pix.dunossauro@gmail.com

Ajude o projeto <3