Algoritmos e Estrutura de Dados I

Aula 15

Pacotes / Numpy

Prof. Dr. Dilermando Piva Jr

1º Semestre - CDN

O que são Módulos

Ou pacotes... Ou bibliotecas...

O que são Módulos?

Funções →

Técnica altamente recomendável, que consiste em dividir um programa/algoritmo maior ou principal em partes menores tornando-o mais estruturado, organizado e refinado.

O que são Módulos?

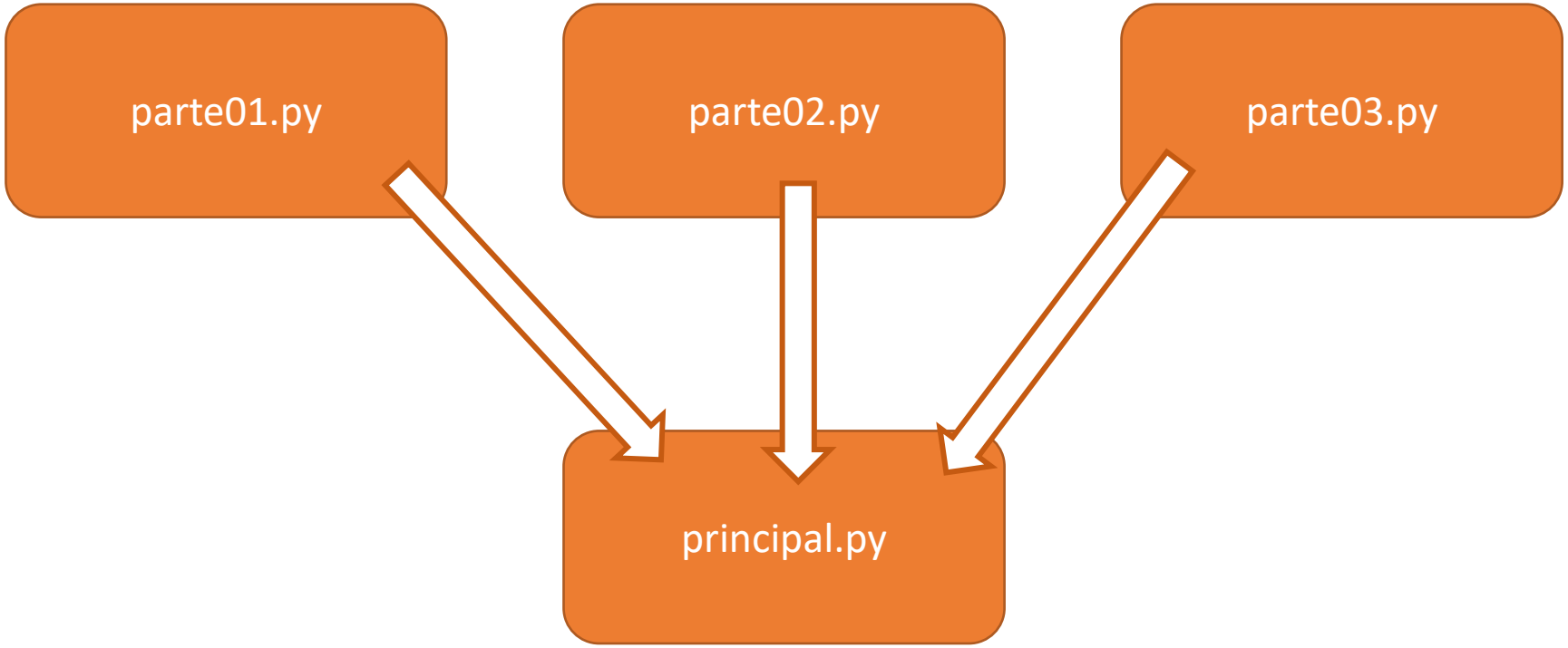
Em Python, Módulos, nada mais são do que “arquivos de scripts”

Por exemplo... Um arquivo que criamos...aula14.py pode ser um módulo que pode ser inserido e reutilizado em outro arquivo.

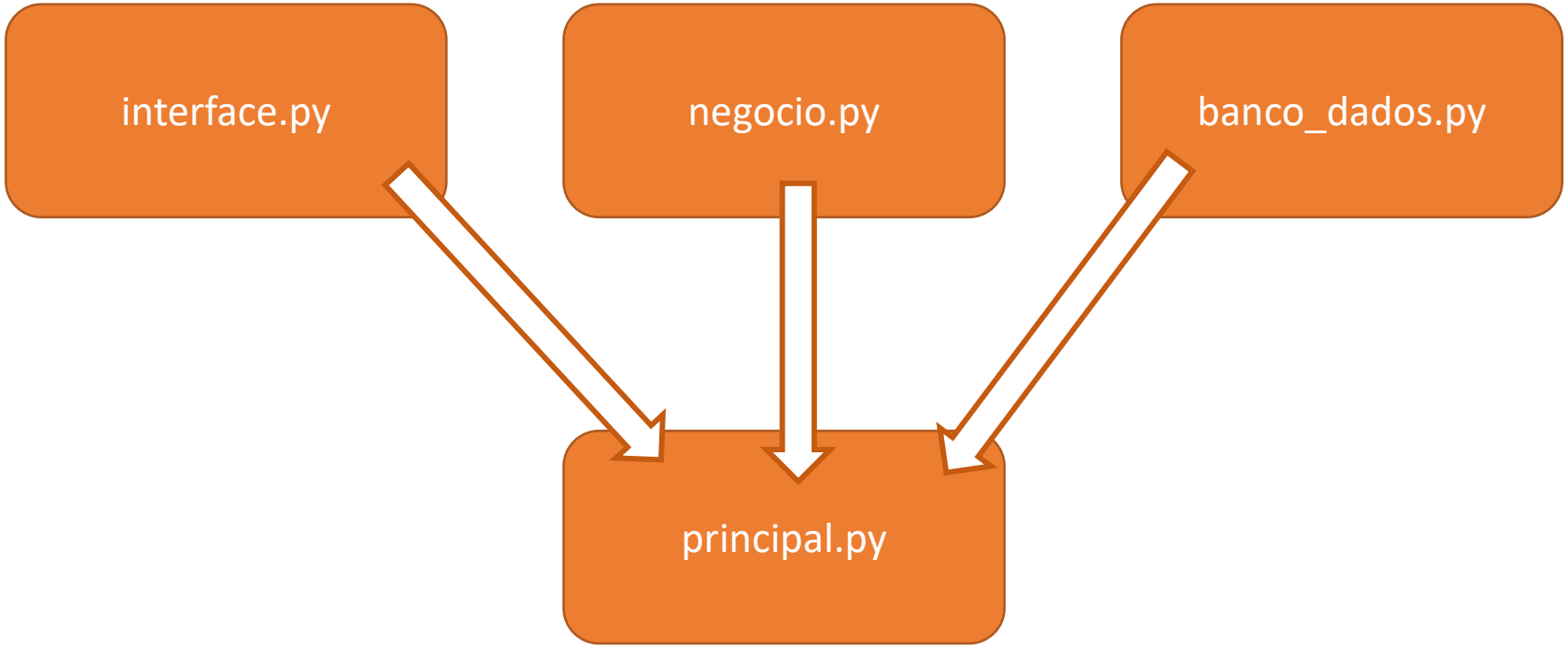
Portanto... Módulos congregam trechos de código... Principalmente funções que podem ser reutilizados!

Também utilizamos módulos para melhor organização dos scripts.

Módulos... organização



Módulos... organização



Módulos/Pacotes... tipos

- **Pacotes** (junção de módulos)
- Forma como os desenvolvedores, empresas etc. compartilham seus códigos com a comunidade

Módulos __builtins__

E como utilizamos...

Módulos Built-ins (__builtins__)

São módulos que, ao instalarmos o Python, são instalados automaticamente.

Bastando, portanto, para utilizá-los, apenas fazer sua importação!!

import <nome_do_módulo / arquivo>

Exemplo:

```
import random
```

Módulos Built-ins (__builtins__)

São módulos que, ao instalarmos o Python, são instalados automaticamente.

Bastando, portanto, para utilizá-los, apenas fazer sua importação!!

import <nome_do_módulo / arquivo>

Exemplo:

import random

Ao realizar o "import" de todo o módulo, todas as funções, atributos, classes e propriedades que estiverem dentro do módulo ficarão disponíveis.

TUDO FICARÁ ALOCADO EM MEMÓRIA, JUNTO COM SEU SCRIPT.

Módulos Built-ins (__builtins__)

São módulos que, ao instalarmos o Python, são instalados automaticamente.

Bastando, portanto, para utilizá-los, apenas fazer sua importação!!

import <nome_do_módulo / arquivo>

Não é recomendado!!
Não é uma boa prática de programação!

Módulos Built-ins (__builtins__)

Como saber quais as funções um módulo possui?

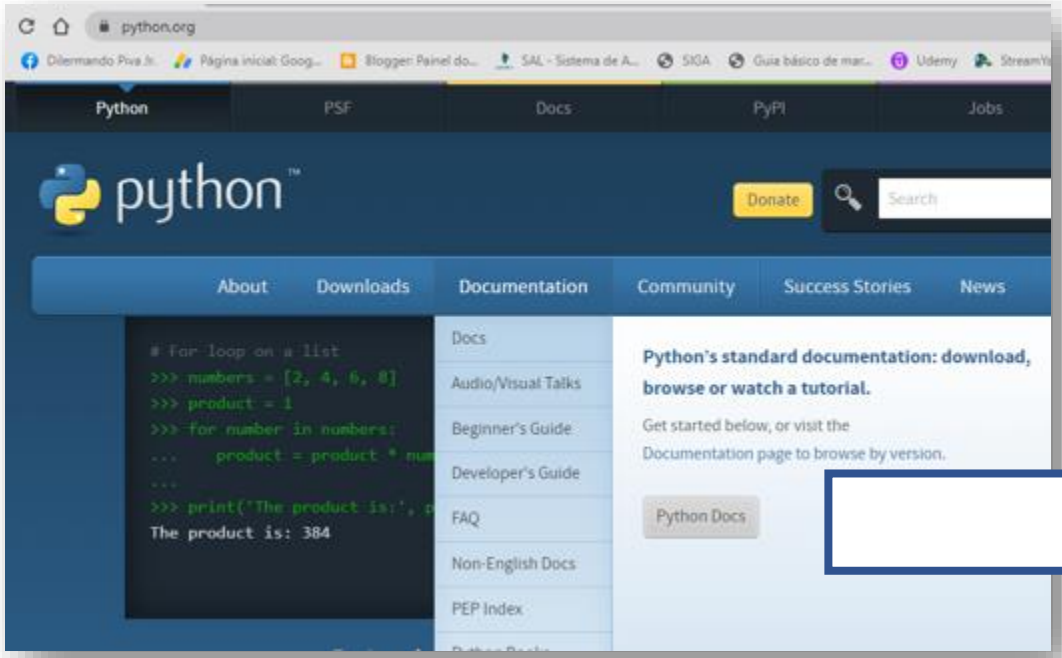
Exemplo:

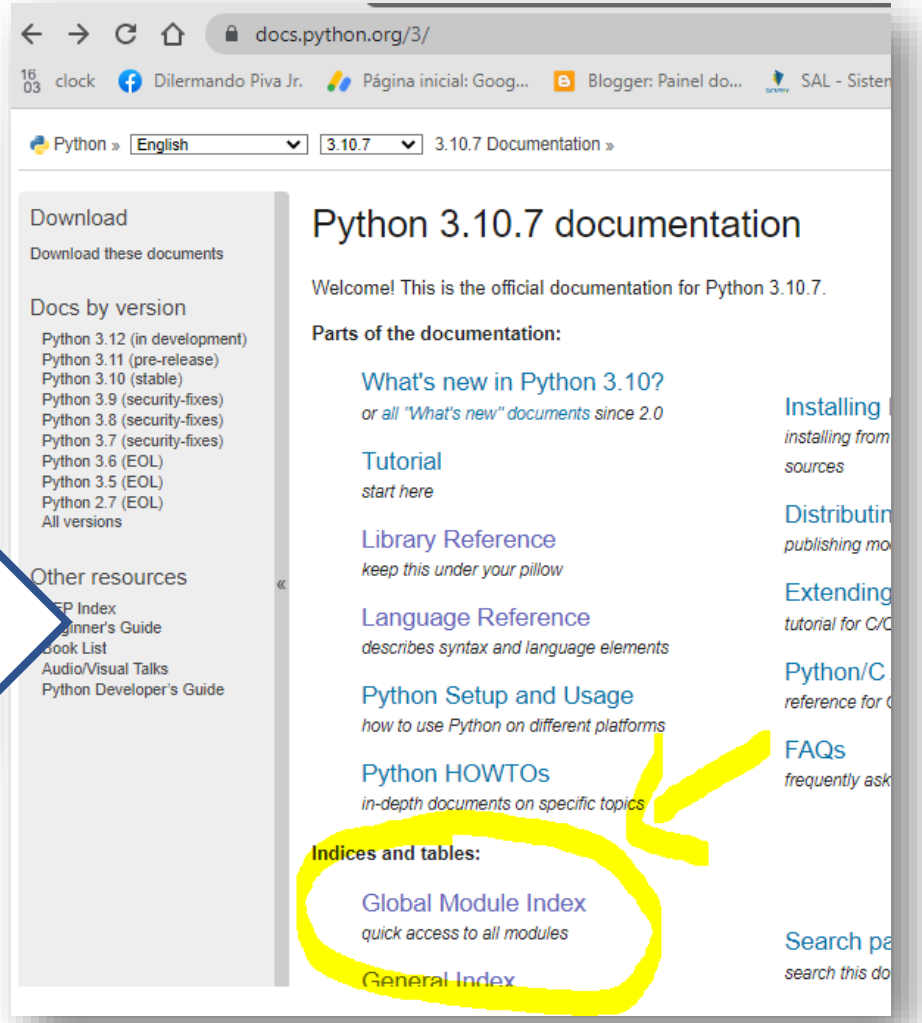
```
import random

dir(random)
help(random.random)
```

Módulos Built-ins (__builtins__)

Como saber quais os módulos __builtins__ que estão disponíveis?





Módulos Built-ins (__builtins__)

Importando uma função específica de um módulo

from <pacote> import <função>

Exemplo:

from random import random

(do módulo random, importe a função random)

from random import randint

((atenção: quando importamos uma função específica não precisamos mais mencionar o pacote para utilizá-la))

Módulos Built-ins (__builtins__)

Importando uma função específica de um módulo

from <pacote> import <função>

Exemplo:

from random import random

(do módulo random, importe a função random)

from random import randint

((aten

menci

Forma recomendada!!

Formas de importação...

Formas básicas de importação:

1) Importar todo o módulo/pacote:

```
import random
print(random.random())
```

2) Importar todo o módulo e associá-lo a um alias (apelido).

```
import random as rd
print(rd.random())
```

3) Importar todo o módulo e deixar as funções disponíveis diretamente

```
from random import *
print(random())
```

Formas de importação...

Formas básicas de importação:

4) Importação apenas uma/várias função(ões) em específica(as)

```
from random import randint, choice
print(randint())
```

5) Importação apenas uma função específica de um módulo e dar um alias (apelido) a ela

```
from random import randint as ri
print(ri(0,100))
```

Formas de importação...

Formas básicas de importação:

6) Importação de várias função(ões) de um mesmo módulo com tupla.

```
from random import (
    randint,
    choice,
    randint
)

print(randint())
```

Módulo random

E como utilizar módulos e suas funções...

Módulo random

Implementa geradores de números pseudo-aleatórios para várias distribuições.

pseudo-aleatórios POIS NÃO SÃO ALEATÓRIOS

Utiliza a hora do sistema operacional para geração de números

Módulo random (principais funções)

random() – gera um número decimal aleatório entre [0 e 1)

uniform(a,b) – gera um número decimal aleatório entre a e b

randint(a,b) – gera um número inteiro entre a e b

choice(i) – Mostra um valor aleatório entre um interável

shuffle(x [,random]) – embaralha a sequencia x

Módulo random (principais funções)

random() – gera um número decimal aleatório (pseudo-aleatório) entre [0 e 1) – o um nunca será gerado.... 0.9999.

```
import random
# importe todo o módulo random

print(random.random())
```

```
from random import random
# do módulo random importe apenas
# função random()

print(random())
```

Módulo random (principais funções)

uniform(a,b) – gera um número decimal aleatório entre [a e b) o limite superior não é incluído.

```
from random import uniform
# do módulo random importe apenas função uniform()

for i in range(1, 6):
    print(uniform(0, 100))
```

Módulo random (principais funções)

randint(a,b) – gera um número inteiro entre a e b

```
from random import randint
# do módulo random importe apenas função randint()

for i in range(1, 7):
    print(randint(1, 61))
```

Módulo random (principais funções)

choice(i) – Mostra um valor aleatório entre um interável

```
from random import choice
# do módulo random importe apenas função choice()

escolha = [‘pedra’, ‘papel’, ‘tesoura’]
print(choice(escolha))
```

Módulo random (principais funções)

choice(i) – Mostra um valor aleatório entre um interável

```
from random import choice
# do módulo random importe apenas função choice()

escolha = [‘pedra’, ‘papel’, ‘tesoura’]
print(choice(escolha))
```

```
palavra = ‘pyPRO’
print(choice(palavra))
```

Módulo random (principais funções)

shuffle(x) – embaralha a sequência x

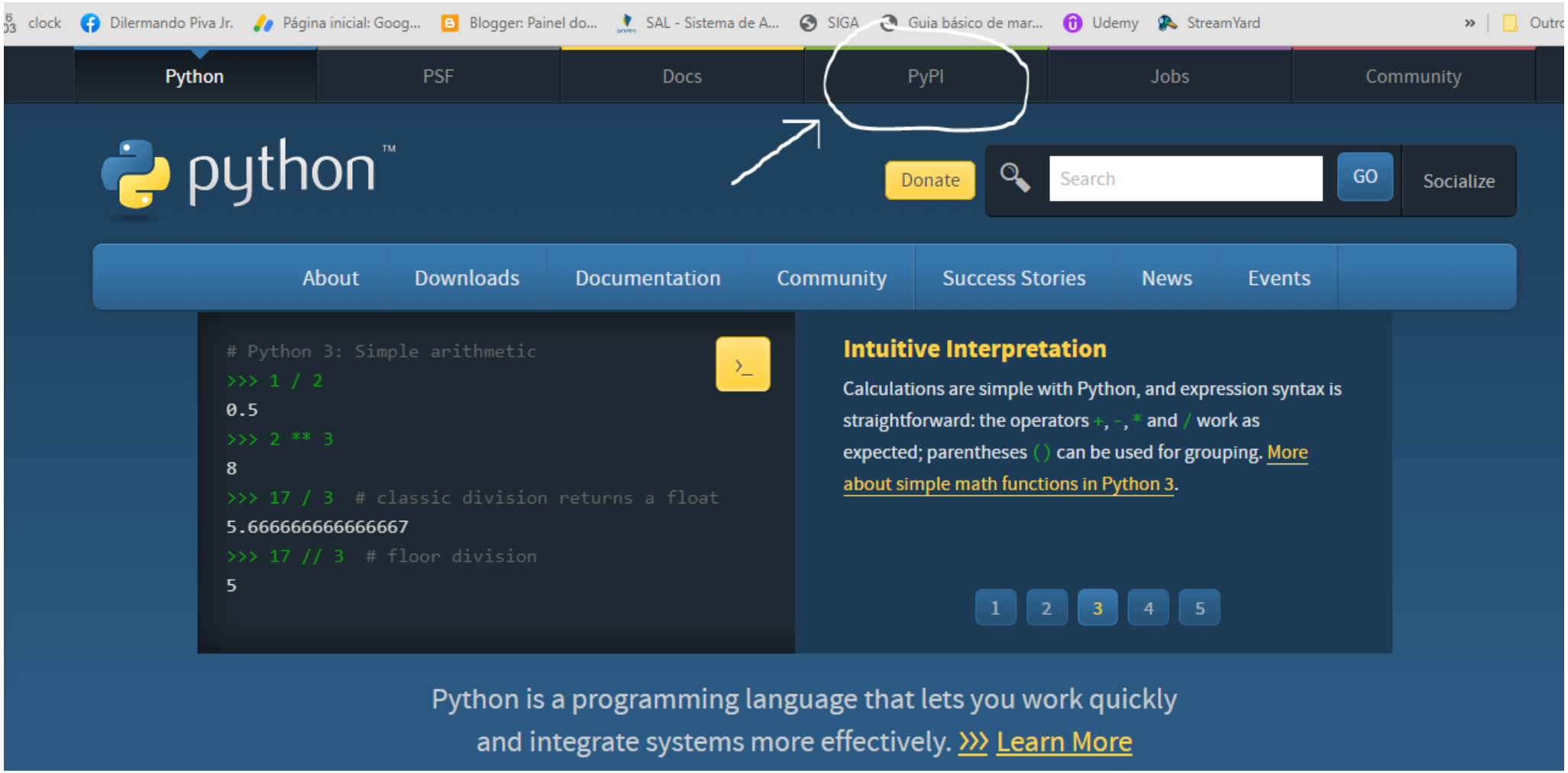
```
from random import shuffle
# do módulo random importe apenas função shuffle()

lista = [1,2,3,4,5,6,7,8,9,10]
print(lista)
shuffle(lista)
print(lista)
```

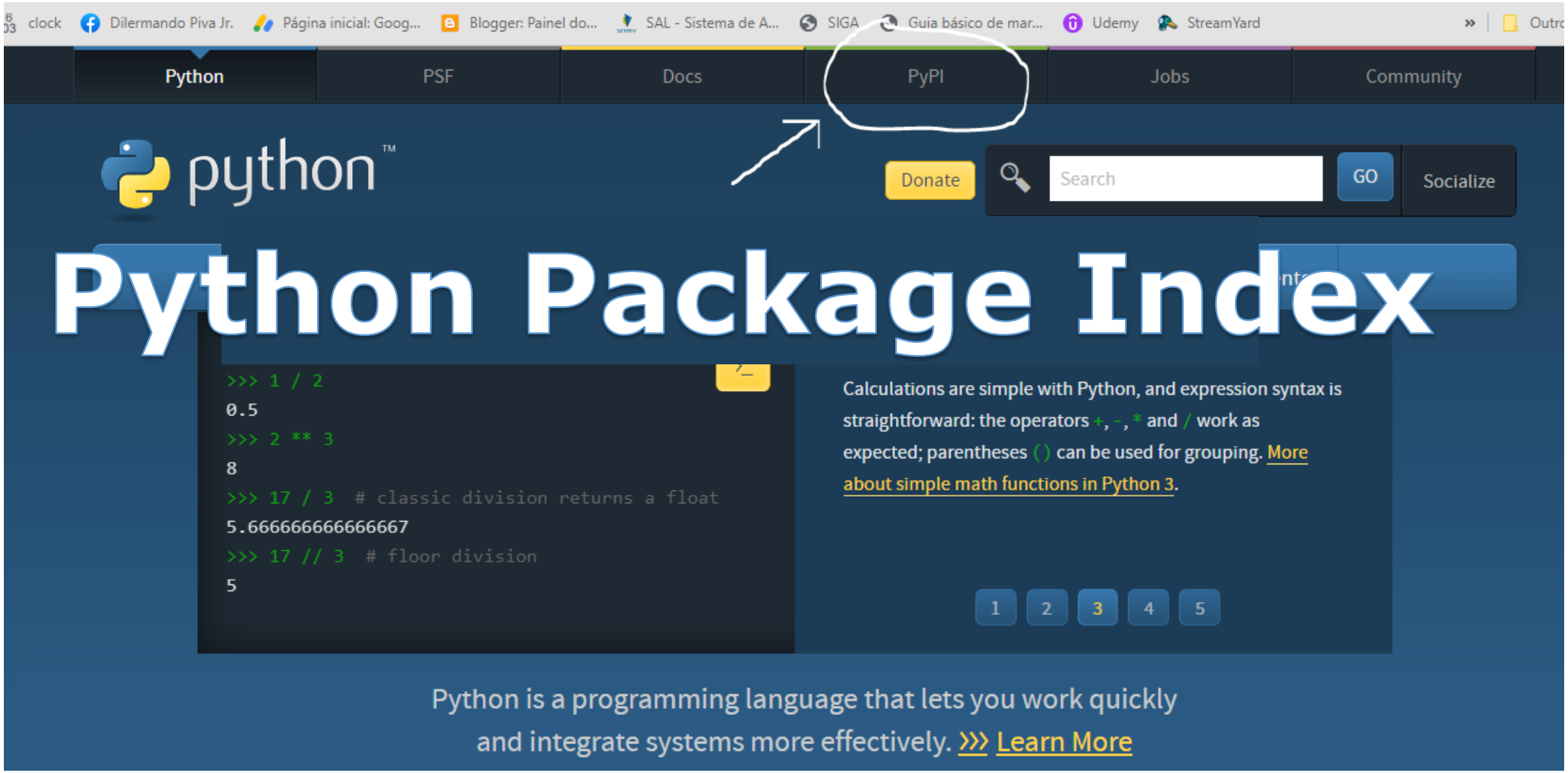
Módulos / Pacotes de Terceiros

Como utilizá-los?

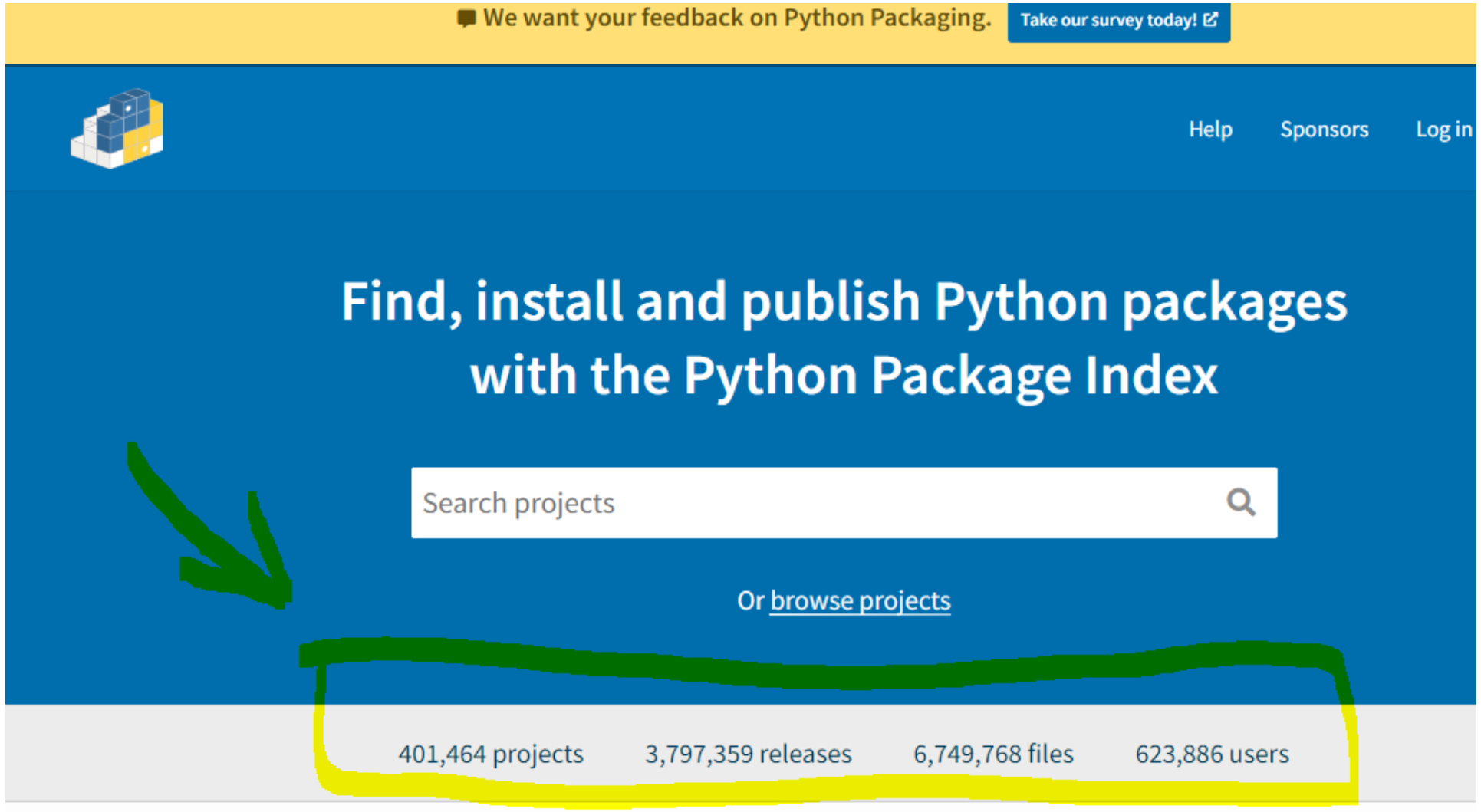
Módulo / Pacotes de Terceiros



Módulo / Pacotes de Terceiros



Módulo / Pacotes de Terceiros



Como utilizar um pacote de terceiros?

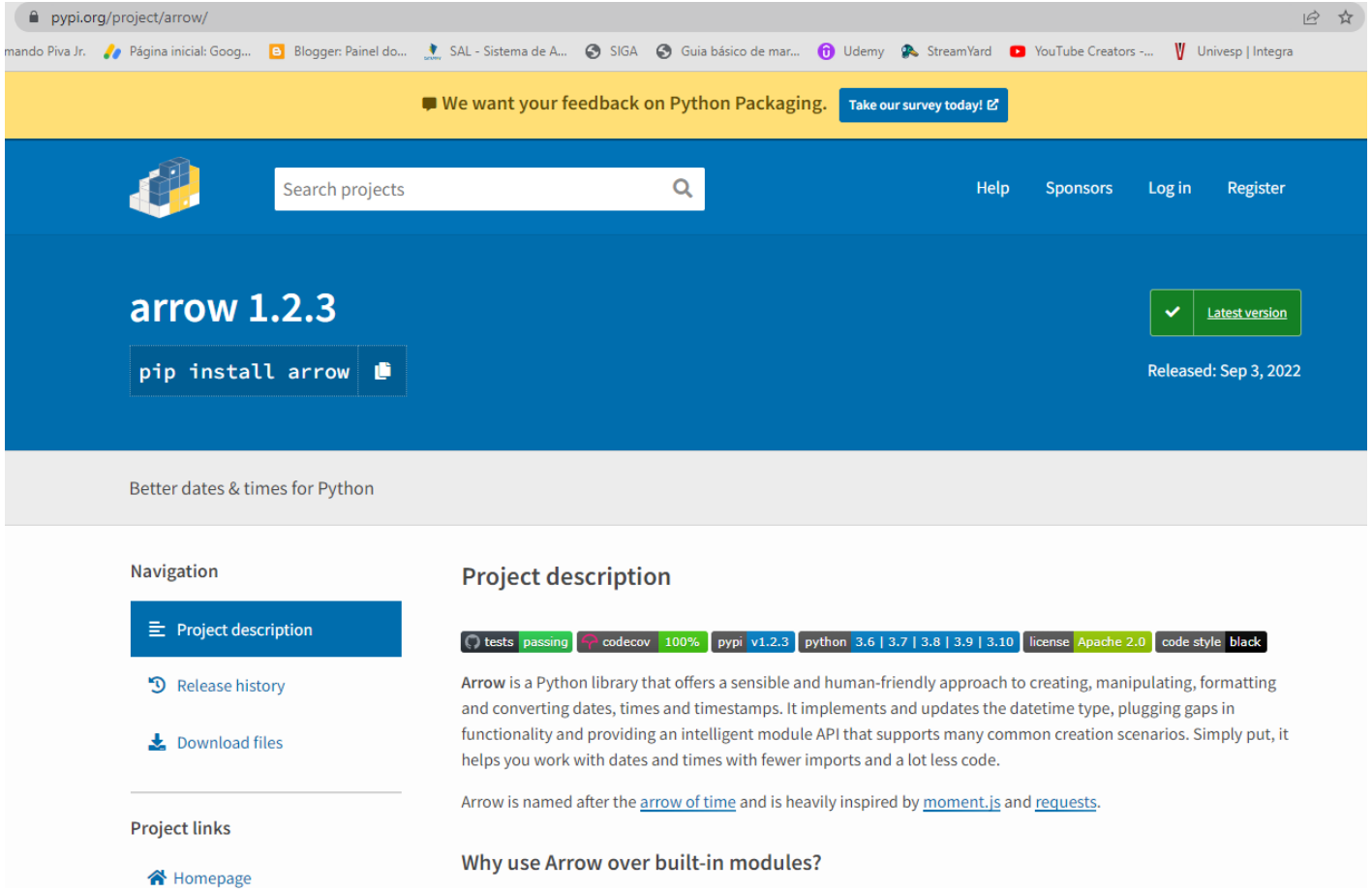
1. Escolher o pacote...
2. Instalar o pacote (utilizando PIP – gerenciador de pacotes)

   PIP = Python Installer Packages

3. Importar o pacote (ou a função em específico)
4. Utilizar

Como utilizar um pacote de terceiros?

1. Escolher o pacote...



Como utilizar um pacote de terceiros?

2. Instalar o pacote (utilizando PIP – gerenciador de pacotes)

PIP = Python Installer Packages

To install Arrow, use pip or pipenv:

```
$ pip install -U arrow
```

Prompt de Comando

```
C:\python>pip install -U arrow
Collecting arrow
  Downloading arrow-1.2.3-py3-none-any.whl (66 kB)
     |□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□| 66 kB 1.1 MB/s
Collecting python-dateutil>=2.7.0
  Downloading python_dateutil-2.8.2-py2.py3-none-any.whl (247 kB)
     |□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□| 247 kB 1.7 MB/s
Collecting six>=1.5
  Downloading six-1.16.0-py2.py3-none-any.whl (11 kB)
Installing collected packages: six, python-dateutil, arrow
Successfully installed arrow-1.2.3 python-dateutil-2.8.2 six-1.16.0
WARNING: You are using pip version 21.1.3; however, version 22.2.2 is available.
You should consider upgrading via the 'c:\python39\python.exe -m pip install --upgrade pip' command.

C:\python>
```

Como utilizar um pacote de terceiros?

3. Importar o pacote (ou a função em específico)

```
import arrow

from arrow import utcnow
```

Como utilizar um pacote de terceiros?

4. Utilizar

```
C:\python>python
Python 3.9.6 (tags/v3.9.6:db3ff76, Jun 28 2021, 15:
Type "help", "copyright", "credits" or "license" fo
>>> import arrow
>>> utc = arrow.utcnow()
>>> utc.year
2022
>>> utc
<Arrow [2022-09-26T18:43:38.812438+00:00]>
>>> utc.month
9
>>> utc.day
26
>>>
```

Como ver quais são os módulos de terceiros instalados?

1. Para ver os pacotes de terceiro instalados use o PIP

pip freeze

Como ver quais são os módulos de terceiros instalados e guardar essa configuração?

1. Existe uma convenção na comunidade Python... Uma vez gerado um ambiente ... Pode-se guardar a configuração (pacotes instalados) com a seguinte linha de comando:

```
pip freeze > requirements.txt
```

Como desinstalar um pacote de terceiros?

1. Para desinstalar um pacote de terceiro use o PIP

pip uninstall <nome do pacote>

```
C:\python>pip uninstall arrow
Found existing installation: arrow 1.2.3
Uninstalling arrow-1.2.3:
  Would remove:
    c:\python39\lib\site-packages\arrow-1.2.3.dist-info\*
    c:\python39\lib\site-packages\arrow\*
Proceed (y/n)? y
  Successfully uninstalled arrow-1.2.3
```

Módulos Customizados

Como construí-los?

Módulos são “scripts”

Os módulos Python nada mais são do que arquivos (.py) ou scripts.

Assim, todo e qualquer arquivo que criamos pode ser utilizado (reutilizado), usando o mesmo conceito de importação que vimos até aqui.

Módulos são “scripts”

funcao.py

```
def primo(n):
    """ Retorna True se n for primo e False caso contrário
        n tem que ser um número inteiro maior que 1 """
    ehprimo = True
    qtd_div = 0
    for i in range(1, n):
        if (n%i)==0:
            qtd_div += 1

    if qtd_div > 1:
        ehprimo = False

    return ehprimo
```

aula66.py

```
from funcao import primo

numero = int(input("Digite um numero: "))
if primo(numero):
    print(f"O numero {numero} É PRIMO")
else:
    print(f"O numero {numero} NÃO É PRIMO")
```

Imports → Sequência de busca

Quando importamos um módulo/pacote, o interpretador do Python busca recursos correspondentes na seguinte sequência:

1) Diretório base do arquivo de nível superior
2) Diretório de PYTHONPATH (se estiver configurada)
3) Diretório da biblioteca padrão (\python-dir\lib\site-packages)
4) O conteúdo de todos os arquivos .pth
   (se estiverem presentes – paths – dentro da instalação Python)

Imports → Sequência de busca

O Interpretador Python fará uma busca nos diretórios de busca pelos seguintes (tipos) arquivos:

- modulo.pyd (*.dll no Windows ou *.so no Unix)
- modulo.py
- modulo.pyc ou modulo.pyo
- modulo (pasta)
- modulo.so (em linux)
- Uma imagem na memória

Imports → VARIÁVEL PYTHONPATH

Além de fazer a modificação da variável de ambiente diretamente no SO...
Você pode fazer a modificação por código...

```
import sys
sys.path.insert(0, "c:\\dev\\tmp\\")
```

Esse código irá inserir o diretório: c:\dev\temp\

Como sendo o primeiro lugar a ser checado na busca por novos módulos.

Pacotes de módulos!

- PACOTES SÃO DIRETÓRIOS CONTENDO UMA COLEÇÃO DE MÓDULOS.

- `\<nome_pacote>`
- `\<nome_pacote>\__init__.py`
- `\<nome_pacote>\<modulo1>.py`
- `\<nome_pacote>\<modulo2>.py`
- `...`

Pacotes de módulos!

- PACOTES SÃO DIRETÓRIOS CONTENDO UMA COLEÇÃO DE MÓDULOS.

- `\<nome_pacote>`
- `\<nome_pacote>\__init__.py`
- `\<nome_pacote>\<modulo1>.py`
- `\<nome_pacote>\<modulo2>.py`
- `...`

ATENÇÃO: Nas versões 2.x do Python, existe a necessidade de um pacote (diretório) ter um arquivo __init__py. Já nas versões 3.x não existe mais essa obrigatoriedade. AINDA SE MANTEM... PARA MANTER A COMPATIBILIDADE !!!

Pacotes de módulos!

- PACOTES SÃO DIRETÓRIOS CONTENDO UMA COLEÇÃO DE MÓDULOS.

- \<nome_pacote>
- \<nome_pacote>__init__.py
- \<nome_pacote>\<modulo1>.py
- \<nome_pacote>\<modulo2>.py
- \<nome_pacote>\<nome_subpacote>__init__.py
- \<nome_pacote>\<nome_subpacote>\<modulo3>.py

Pacotes de módulos!

- EXEMPLO DE CRIAÇÃO E UTILIZAÇÃO...

- `\pypro`
- `\pypro\__init__.py`
- `\pypro\geral.py`
- `\pypro\teste.py`
- `\pypro\fundamentos\__init__.py`
- `\pypro\fundamentos\funcoes.py`

Pacotes de módulos!

- EXEMPLO DE CRIAÇÃO E UTILIZAÇÃO...

```
from pypro import geral, teste
from pypro.fundamentos import base
...
geral.funcao1()
teste.funcao2()
base.funcao3()
```

- `\pypro`
- `\pypro\__init__.py`
- `\pypro\geral.py`
- `\pypro\teste.py`
- `\pypro\fundamentos\__init__.py`
- `\pypro\fundamentos\base.py`

Under e Dunder
_X, __all__, __main__ e __name__

O que são e para que servem?

__ (Doble Under... Ou Dundder)

Dois underscore (underline)...

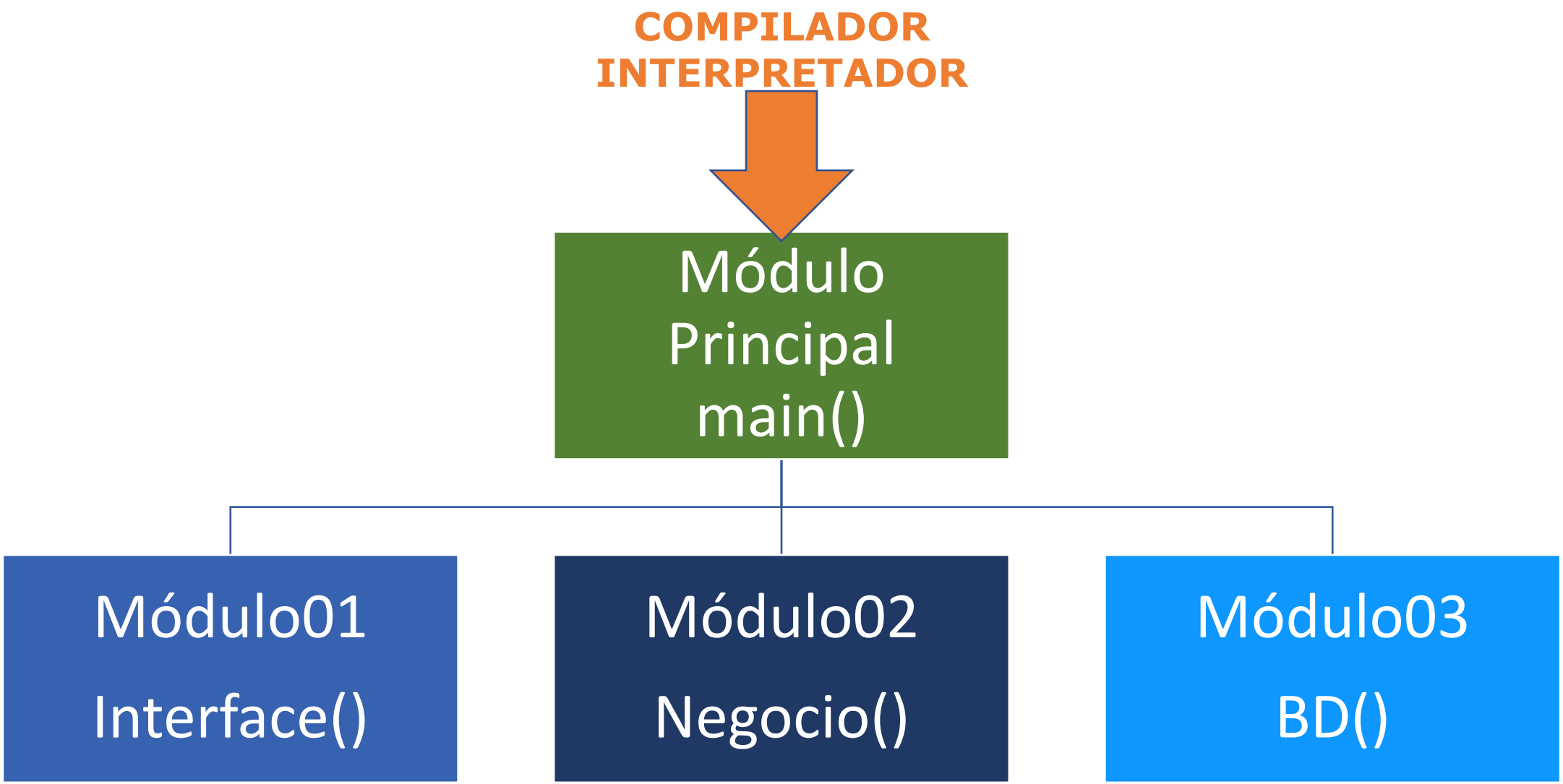
A utilização de 1 ou 2 underscore antes, depois e antes e depois de um nome de variável tem significados distintos... Podendo ser apenas uma convenção ou ser uma exigência do interpretador.

Iremos ver isso em detalhes quando vermos
ORIENTAÇÃO A OBJETOS.

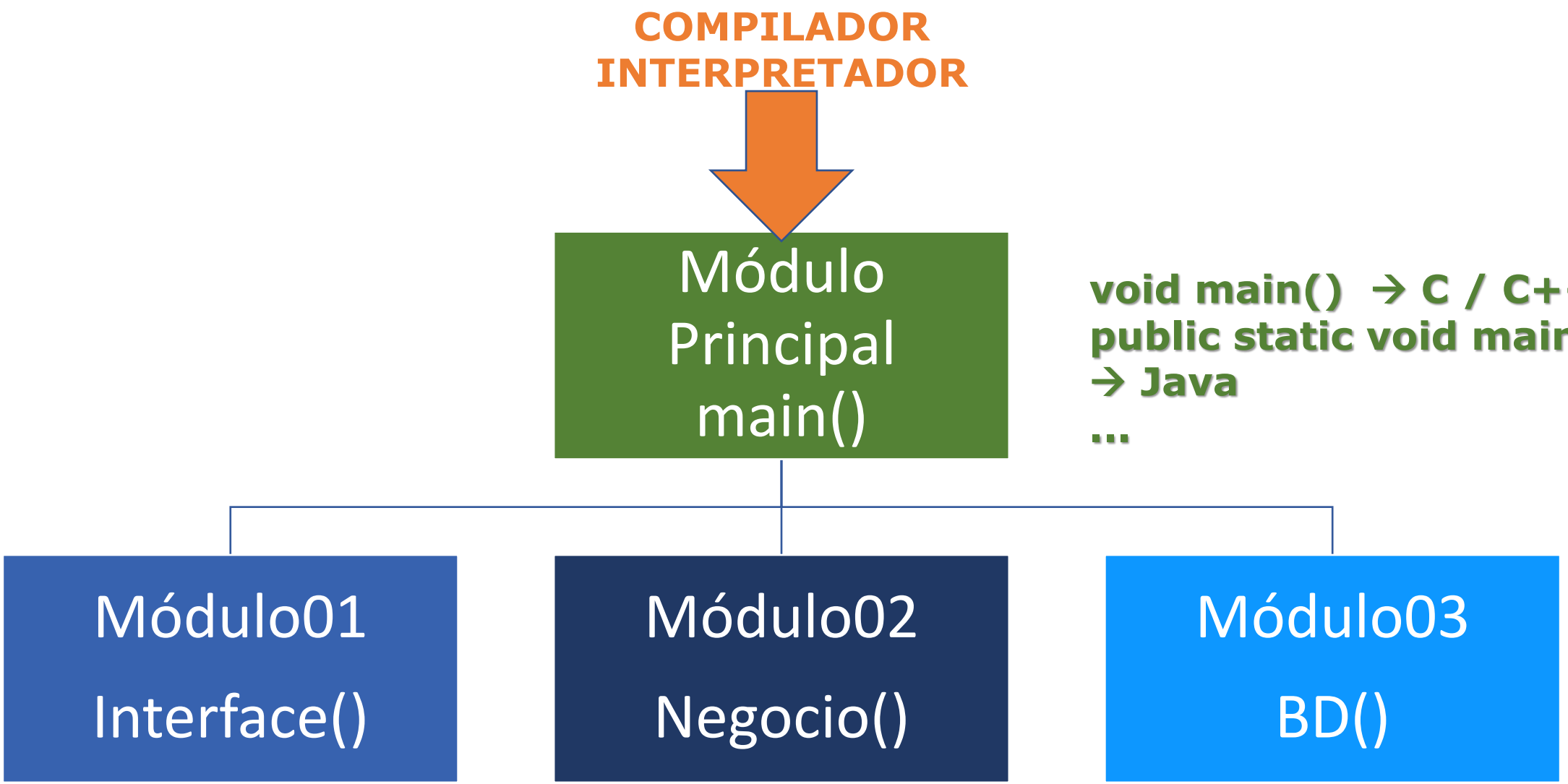
Variáveis (Atributos)
_X, __all__, __main__ e __name__

O que são e para que servem?

Na maioria das linguagens...

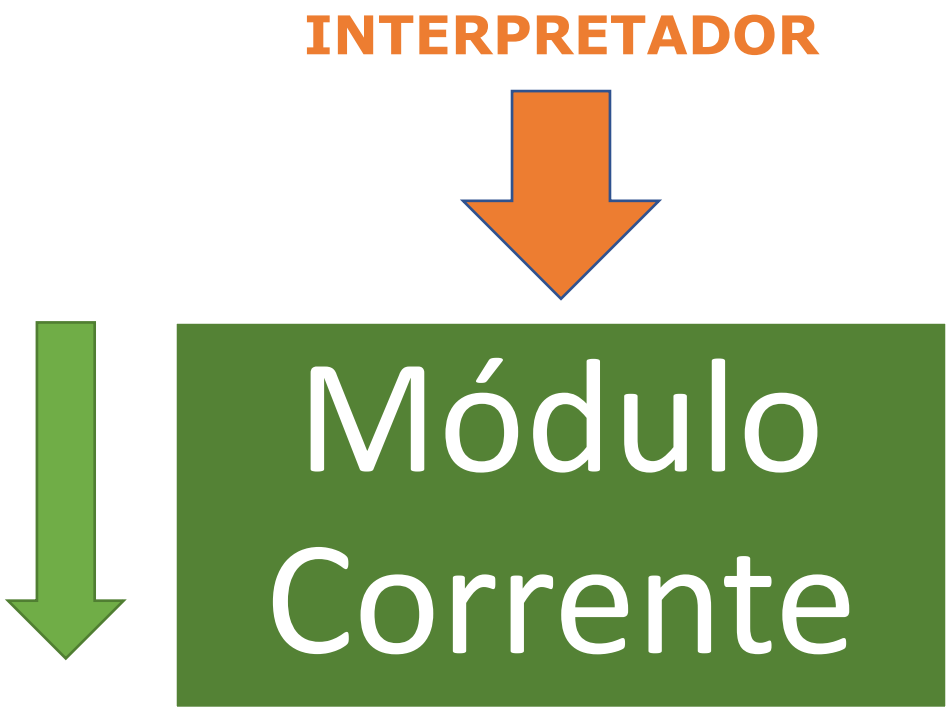


Na maioria das linguagens...

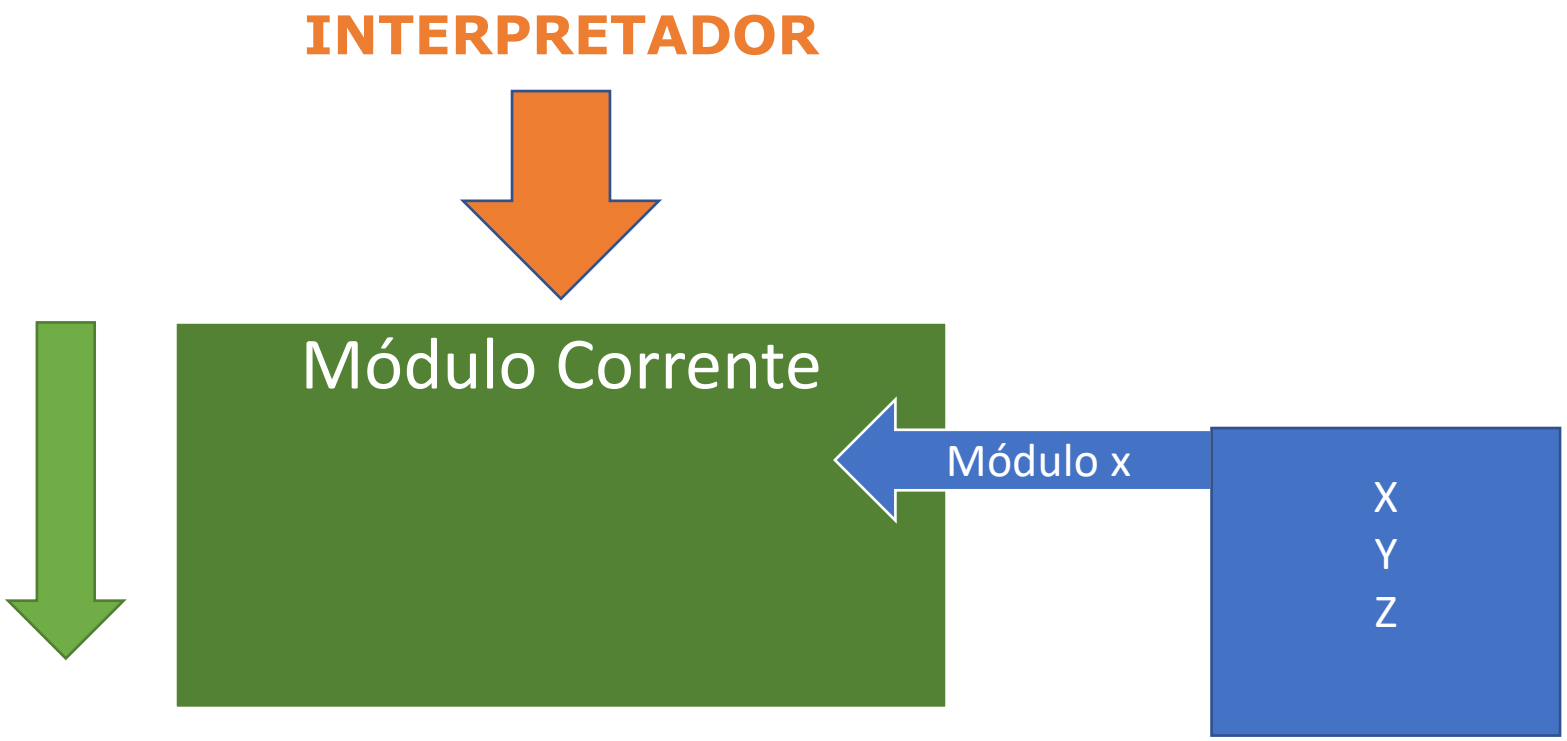


void main() → C / C++
public static void main() → Java
...

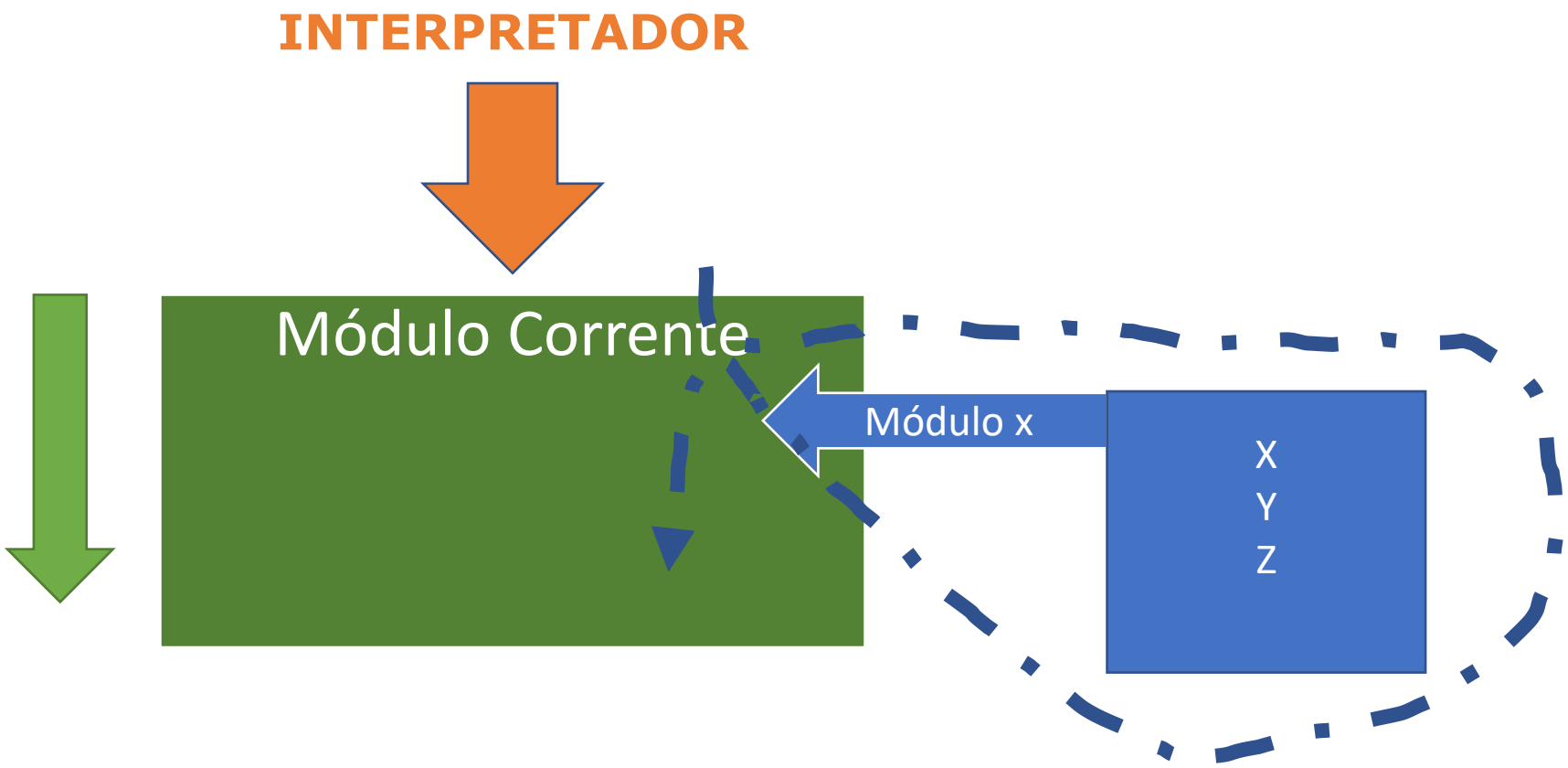
Em Python...



Em Python...



Em Python...



Exemplo de execução dos módulos importados...

MODULO1.PY

```
x = "pyPRO - Seja um Profissional Python"
print(x)
```

PROGRAMA.PY

```
import modulo1
print("-------------------------")
y = "pyPRO - http://www.pypro.com.br"
print(y)
print("-------------------------")
print(x)
```

Ocultação de dados em Módulos

Quando importamos um módulo / pacote, também importamos todas as variáveis, funções, etc.

Toda a gama de “espaços de nomes” desses módulos.

Existem algumas técnicas que minimizam essa “importação” e possibilidade de alteração de nomes dos módulos.

Ocultação de dados em Módulos
_X

A primeira técnica é utilizar um _ (underscore / underline) na frente de cada nome utilizado no módulo.

Isso evita que ele seja copiado se o módulo for importado com "`from ...`"

Ocultação de dados em Módulos __all__

A segunda técnica faz o contrário que _X se propõe. Ao invés de não permitir a cópia, ao atribuirmos valores a esta variável (__all__), estamos dizendo o que deve ser exportado (apenas o que deve ser).

Por exemplo: "`__all__ = [*Error, *encode*, *decode*]`"

Estamos dizendo que quando importado com "`from ...`" apenas essas variáveis (nomes) devem ser importados.

Variáveis... __main__ e __name__

A utilização conjunta dessas duas variáveis permite a IMPORTAÇÃO DE UM ARQUIVO COMO MÓDULO E A SUA EXECUÇÃO COMO UM PROGRAMA INDEPENDENTE...

Vamos ver isso mais detalhadamente.

Variável: __name__

Cada módulo (arquivo .py) possui um atributo interno chamado __name__ que o Interpretador Python configura automaticamente.

Se o arquivo que está sendo executado for de nível superior, o valor configurado para __name__ é igual a __main__.

Se o arquivo que está sendo executado for um módulo importado, __name__ terá o nome deste módulo.

Usando: __name__ e __main__

Criar o módulo: modulo.py

```
def funcao():
    print("pypro - seja um profissional python")

if __name__ == '__main__':
    funcao()
```

```
C:\python>python modulo.py
pypro - seja um profissional Python!

C:\python>
```

```
C:\python>python
Python 3.9.6 (tags/v3.9.6:db3ff76, Jun 28 2021, 15:26:21) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import modulo
>>> modulo.funcao()
pypro - seja um profissional Python!
>>>
```

Usando: __name__ e __main__

Criar o módulo: modulo2.py

```
def funcao():
    print("pypro - seja um profissional python")

funcao()
```

```
C:\python>python
Python 3.9.6 (tags/v3.9.6:db3ff76, Jun 28 2021, 15:26:21) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import modulo2
pypro - seja um profissional Python!
>>>
```

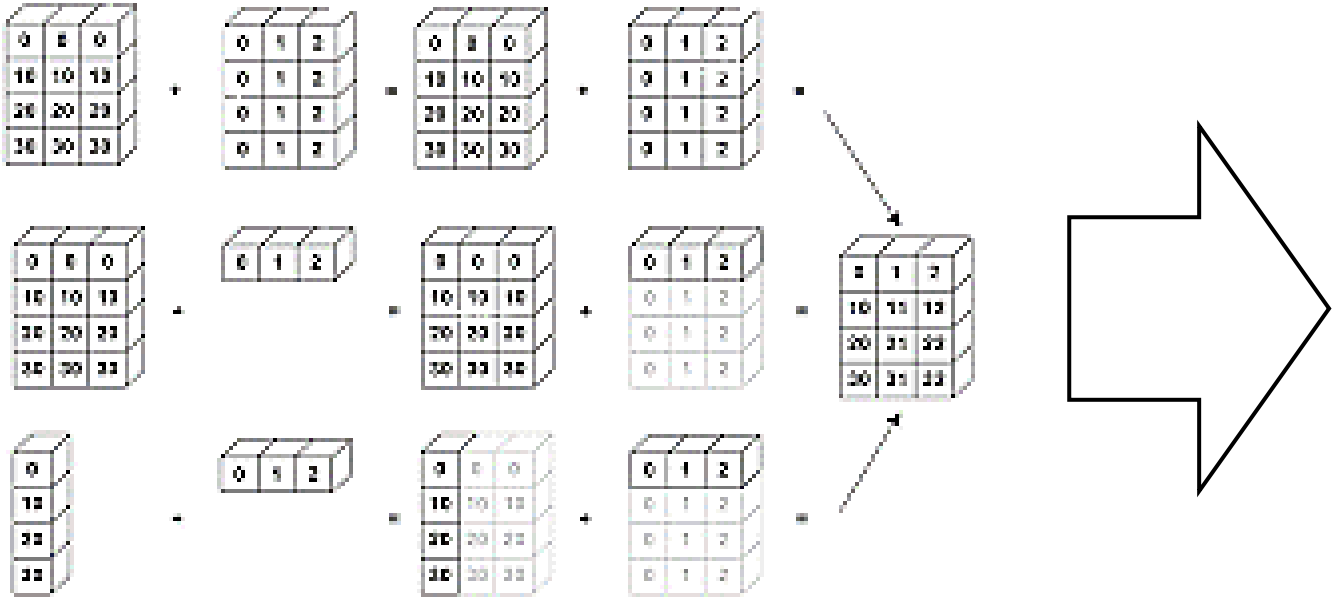
Noções Matemáticas c/ Numpy

NumPy – Numerical Python

- É um pacote fundamental para computação matemática em Python.
- Fornece suporte para arrays e matrizes, além de uma série de funções matemáticas para operações com esses objetos.
- É o pacote mais utilizado para Análise de Dados, Ciência de Dados, Machine Learning e Inteligência Artificial.

NumPy – Numerical Python

- Operações com matrizes (Arrays)



Tudo que fazemos com:
- Ciência de Dados
- Machine Learning
- Inteligência Artificial

→ MODELO MAIS AVANÇADO

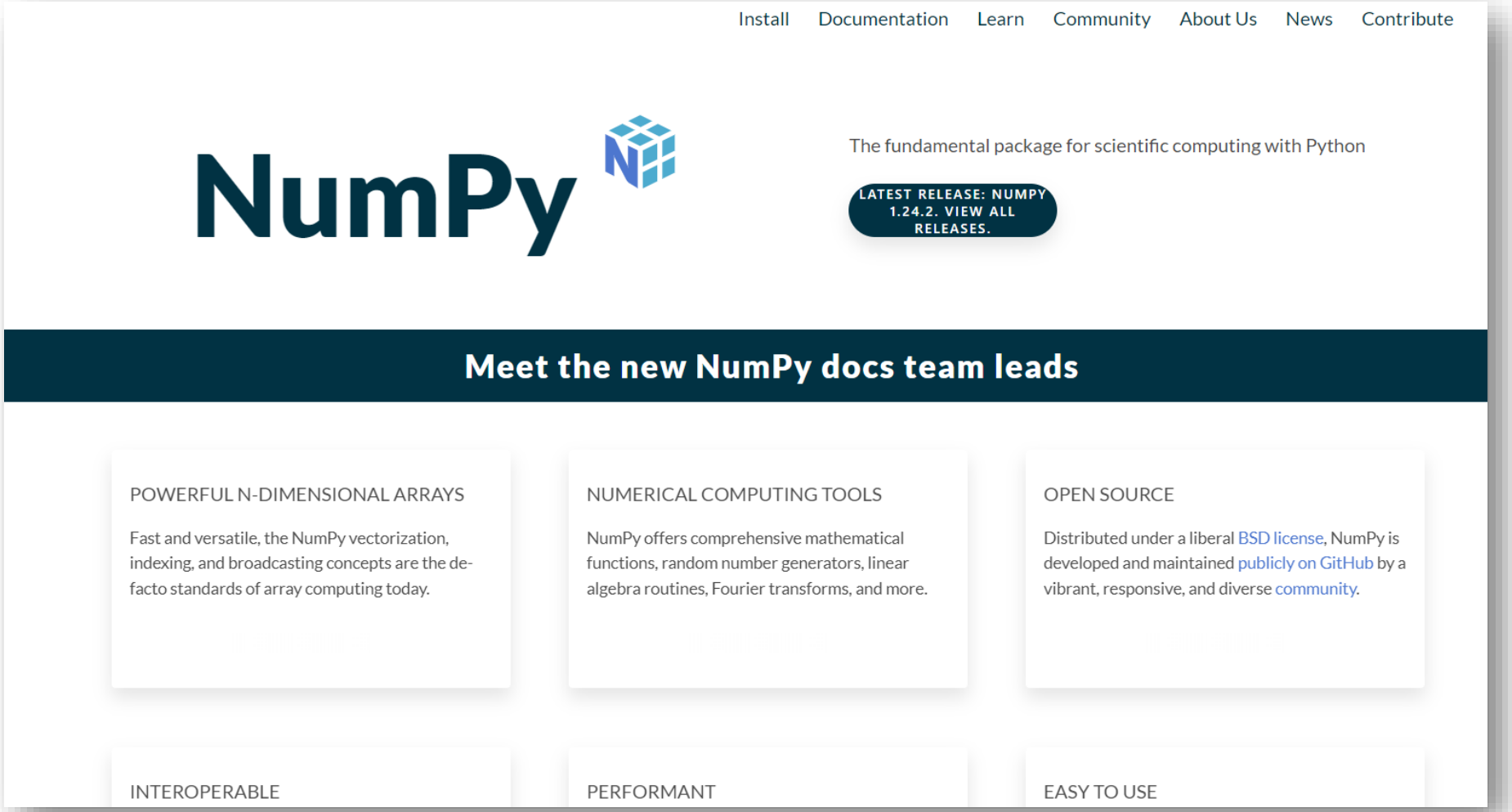
TUDO SE RESUME A OPERAÇÕES COM MATRIZES

NumPy – Numerical Python

- Operações de álgebra linear
- Operações para manipulação de Arrays Multidimensionais
- Ferramentas de leitura de datasets baseados em arrays
- Série de ferramentas de análise de dados

NumPy é mais eficiente para armazenar e manipular dados numéricos do que qualquer estrutura built-in em Python

NumPy – Numerical Python



numpy.org

NumPy – Numerical Python

ANACONDA

NumPy

```
import numpy  as np
```

NumPy – Numerical Python

<u>O que é um Array?</u>

Conjunto de valores, todos do mesmo tipo, indexados por uma tupla de valores não-negativos