nappacademy

Napp Academy

Orlando Saraiva Júnior

"A ship in port is safe, but that is not what ships are for. Sail out to sea and do new things."

Grace Hopper

"Um navio no porto é seguro, mas não é para isso que servem os navios. Navegue para o mar e faça coisas novas."

Grace Hopper

Sprint 7

Sprint 7
Conteúdo programático

nappacademy

- Iteradores, Iteráveis e geradores
  - Como acessar grandes arquivos
  - Funções geradoras: como funcionam ?

- Redes
  - Como funciona redes de computadores
  - Como acontece o consumo de APIs

Sprint 7
Sobre o que mais ?

nappacademy

Lidando com projetos

- Lista de exercícios baseada em projetos

Módulo OS

- Como gerenciar diretórios / arquivos
- Como manipular grandes arquivos

Iteráveis, iteradores e geradores

Iteração

nappacademy

Iteração é fundamental para o processamento de dados. Ao percorrer conjunto de dados que não cabem na memória, precisamos ter como acessar os itens de modo lazy (preguiçoso), isto é, um item de cada vez, sob demanda.

É disto que trata o padrão Iterator (Iterador). Vamos mostrar como este padrão está integrado à linguagem de modo que não precisamos implementar manualmente.

A palavra yield permite a construção de geradores que funcionam como iteradores. Python usa geradores em vários lugares.

Até a função range() devolve um objeto gerador, e não mais uma lista, como em python2. Se precisarmos de uma lista a partir de um range, precisamos ser explícitos.

Iteração

nappacademy

Toda coleção em python é iterável , e os iteradores são usados internamente para dar suporte a:

- laços for;
- construção e extensão de tipos para coleção;
- percorrer arquivo-texto linha a linha em um laço;
- list, dict e set comprehensions;
- desempacotamento de tuplas;
- desempacotamento de parâmetros com * em chamadas de função

Iteráveis, iteradores e geradores

nappacademy

A classe Sentence extrai palavras de um texto pelo índice.

Todo programador python sabe que sequencias são iteráveis. Agora vamos descobrir exatamente por quê.

sentence1.py

Iteráveis, iteradores e geradores

nappacademy

Sempre que o interpretador precisa iterar por um objeto x, ele chama iter(x) automaticamente.

A função embutida iter:

1) Verifica se o objeto implementa **__iter__** e o chama para obter o iterador.

2) Se **__iter__** não estiver implementado, mas o **__getitem__** estiver, python criará um iterador que permite acessar os itens em sequencia, começando no índice 0 (zero)

3) Se Isso Falhar, Python levantará **TypeError**, normalmente dizendo que o objeto não é iterável.

Iteráveis, iteradores e geradores

nappacademy

Por isso qualquer sequencia em Python é iterável: todas elas implementam **__getitem__**. Todas as sequencias padrões também implementam **__iter__**.

Um objeto é considerado iterável não só quando implementa o método especial __iter__, mas também quando implementa o método __getitem__, desde que este aceite chaves int.

Iteráveis, iteradores e geradores

nappacademy

Iterável é qualquer objeto a partir do qual a função embutida iter pode obter um iterador. Objetos que implementem um método __iter__ que devolva um iterador são iteráveis. Sequencias sempre são iteráveis, assim como objetos que implementem um método __getitem__ que aceite índices a partir de 0.

Python obtém iteradores a partir de iteráveis.

simula_for.py

Iteráveis, iteradores e geradores

nappacademy

A interface padrão de um iterador tem dois métodos:

__next__

Devolve o próximo item disponível, StopIteration quando não houver mais itens levantando

__iter__

Devolve self; permite que iteradores sem usados em lugares em que se espera um iterável, por exemplo, em um laço for.

Iteráveis, iteradores e geradores

nappacademy

Iterador é qualquer objeto que implemente o método **__next__**, sem argumentos, que devolva o próximo item de uma série ou levante **StopIteration** quando não houver mais itens.

Os iteradores em Python também implementam o método __iter__, portanto também são iteráveis.

Iteráveis, iteradores e geradores

nappacademy

A próxima classe Setence foi criada de acordo com o padrão de projeto Iterator clássico, seguindo o esquema do livro GoF.

Esta implementação tem como objetivo deixar clara a distinção fundamental entre um iterável e um iterador e como eles estão relacionados.

sentence2.py

Iteráveis, iteradores e geradores

nappacademy

Uma causa comum de erros na criação de iteráveis e iteradores é confundí-los.

Um **iterável** tem o método __iter__ que sempre instancia um novo iterador.

Iteradores implementam o método __next__ que devolve itens individuais e um método __iter__ que devolve self.

Um iterável jamais deve atuar como um iterador de si mesmo.

Função geradora

nappacademy

Uma implementação mais pythonica da mesma funcionalidade usa a função geradora para substituir a classe SequenceIterator.

sentence3.py

Função geradora

nappacademy

Qualquer função python que tenha a palavra reservada yield em seu corpo é uma função geradora: uma função que, quando chamada, devolve um objeto gerador.

Em outras palavras, uma função geradora é uma fábrica (factory) de geradores.

Uma implementação lazy

nappacademy

A interface do Iterador foi concebida para ser lazy:

next(my_iterador_ produz um item de cada vez. O oposto ao lazy é eager (ávido).

As implementações até agora não foram lazy, pois __init__ cria uma lista de forma ávida com todas as palavras do texto, associando-a ao atributo self.word.

Isto implica em processar todo o texto, e a lista pode usar tanta memória quanto o próprio texto.

Uma implementação lazy

nappacademy

A função re.finditer é uma versão lazy de re.findall. Em vez de uma lista, ela devolve um gerador que produz instâncias re.MatchObject sob demanda.

Assim, esta versão é lazy, produzirá a próxima palavra somente quando necessário.

sentence4.py

Iteráveis, iteradores e geradores

nappacademy

A biblioteca padrão oferece muitos geradores, desde objetos-arquivo de texto puro, que oferecem iteração linha a linha, até a função os.walk, que produz nomes de arquivos enquanto percorre uma árvore de diretório, deixando as buscas recursivas no sistema de arquivos tão simples quanto um laço for.

Além disso, o módulo **itertools** possui um conjunto de iteradores.

iter.py

Performance

nappacademy

```
import sys
nums_squared_lc = [i * 2 for i in range(10000)]
sys.getsizeof(nums_squared_lc)
nums_squared_gc = (i ** 2 for i in range(10000))
print(sys.getsizeof(nums_squared_gc))
```

performance.py

Performance

nappacademy

```
import cProfile
cProfile.run('sum([i * 2 for i in range(10000)])')
cProfile.run('sum((i * 2 for i in range(10000)))')
cProfile.run('sum([i * 2 for i in range(10000000)])')
cProfile.run('sum((i * 2 for i in range(10000000)))')
```

performance.py

Acesso a grandes arquivos

nappacademy

Para acesso a grandes arquivos, geradores economizam recursos computacionais.

arquivo.py

Decoradores de função e closures

Decoradores de função e closures

nappacademy

Decoradores (*decorators*) de função nos permitem "marcar" funções no código-fonte para modificar o seu comportamento de alguma maneira.

```
@decorate
def target():
    print('Running target()')
```

É um recurso poderoso, mas dominá-lo exige compreender as *closures*.

Closure

Closures

nappacademy

Closure é uma função aninhada que tem acesso a uma *variável livre* de uma função envolvente que concluiu sua execução.

Três características de um *closure* :

- É uma função aninhada
- Tem acesso a uma *variável livre* no escopo externo
- Ele é retornado da função envolvente

  Uma variável livre é uma variável que não está vinculada ao escopo local. Para que os *closures* funcionem com variáveis imutáveis, como números e strings, temos que usar a palavra-chave *non-local*.

Funções aninhadas

nappacademy

Funções internas, também conhecidas como funções aninhadas, são funções que você define dentro de outras funções.

Em Python, esse tipo de função tem acesso direto a variáveis e nomes definidos na função envolvente.

As funções internas têm muitos usos, principalmente como fábricas de *closure* e funções de decorador.

inner.py

Regra para escopo de variáveis

nappacademy

Python não exige que você declare variáveis, mas supõe que uma variável cujo valor já tenha sido atribuído no corpo de uma função é local.

Observe como o interpretador trata *b*

escopo.py

Closures

nappacademy

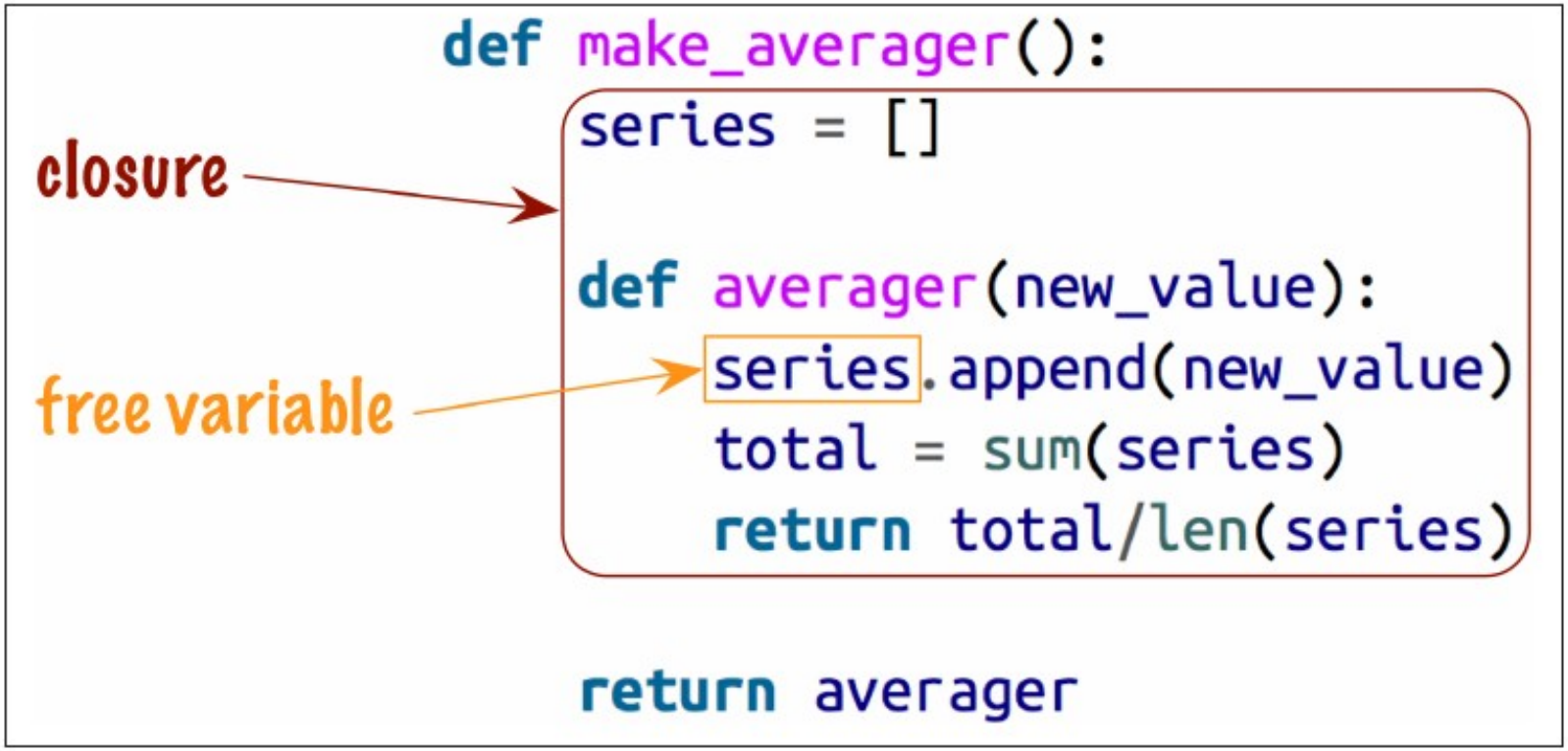
Closure é uma função com um escopo estendido, que engloba variáveis não globais referenciadas no corpo da função que não estão definidas ali.

Ela precisa acessar variáveis não globais definidas for a de seu corpo.

avg.py

Closures

nappacademy



Closures

nappacademy

Vamos inspecionar a função criada por make_averanger

```
>>> avg
<function make_averager.<locals>.averager at 0x7f1eb321a8b0>
>>> avg.__closure__
(<cell at 0x7f1eb3262f70: list object at 0x7f1eb322da40>,)
>>> avg.__code__.co_varnames
('new_value', 'total')
>>> avg.__code__.co_freevars
('series',)
>>> avg.__closure__[0].cell_contents
[10, 11, 12]
>>>
```

avg.py

Closures

nappacademy

Nossa implementação de make_averager não é eficiente.

A próxima versão calcula a soma com sum sempre que averager é chamada.

Mas em que ponto falha ?

avg2.py

Nonlocal

Estamos fazendo uma atribuição a ***count*** no corpo de ***averager***, e isso faz dela uma variável *local*. O mesmo ocorre com a variável ***total***.

Para contornar este problema, a declaração *nonlocal* foi introduzida no python 3.

avg3.py

Pesquisar
- Documentação
- PEP3104

Decorador

Decorador

nappacademy

Um decorador é um invocável (*callable*) que aceita outra função como argumento (função decorada). O decorador pode realizar algum processamento com a função decorada e devolvê-la ou substituí-la por outra função ou objeto invocável.

decorador1.py

Quando os decoradores são executados

nappacademy

Uma característica fundamental dos decoradores é que eles são executados imediatamente após a função decorada ser definida. Normalmente isso ocorre em *tempo de importação* (isto é, quando um módulo é carregado pelo interpretador python)

decorador2.py

Os decoradores de uma função são executados assim que o módulo é importado, porém as funções decoradas executam somente quando são explicitamente chamadas.

Tempo de importação (*import time*) é diferente de tempo de execução (*runtime*)

decorador3.py

Decoradores da biblioteca-padrão

nappacademy

Python tem três funções embutidas criada para decorar métodos:

- property
- classmethod
- staticmethod.

Decoradores parametrizados

nappacademy

Ao fazer um parse de um decorador, python pega a função decorada e passa-a como primeiro argumento para a função decoradora. Como fazer um decorador aceitar outros argumentos ?

Crie uma fábrica (*factory*) de decoradores que aceite esses argumentos e devolva um decorador que, por sua vez, será aplicado à função a ser decorada.

decorador4.py

Sobre padrões de projetos

nappacademy

O **Decorator** é um padrão de projeto estrutural que permite que você acople novos comportamentos para objetos ao colocá-los dentro de invólucros de objetos que contém os comportamentos.

Os decoradores de função em python se enquadram na descrição geral de Decorator do livro GoF: “Confere responsabilidades adicionais a um objeto dinamicamente. Os decoradores oferecem uma alternativa à criação de subclasses para estender funcionalidades”

Em nível de implementação, os decoradores em python não lembram o padrão de projeto clássico Decorator.

Gerenciadores de contexto

Gerenciadores de contexto

nappacademy

Digamos, por exemplo, que você tenha certa funcionalidade com pré-condições e pós-condições que precisam ser atendidas ao executar essa funcionalidade.

Neste cenário, gerenciadores de contexto são interessantes.

spoiler.py

Gerenciadores de contexto

nappacademy

- Normalmente, um Context Manager consiste em dois métodos mágicos: __*enter*__ e __*exit*__.
- A instrução *with* chama o *Context Manager*.
- A instrução with chama o método __*enter*__ e o valor retornado será atribuído à variável rotulada após a palavra-chave *as*.
- O método __enter__ não precisa retornar um valor.
- Depois que o método __*enter*__ foi chamado, o que quer que esteja dentro do bloco *with* será executado, então o python irá chamar o método __*exit*__

Gerenciadores de contexto

nappacademy

@contextmanager é um decorador que permite criar um gerenciador de contexto partir de uma função geradora simples, em vez de criar uma classe e implementar o protocolo. **(exemplo 2)**

Já ContextDecorator é uma classe-base para definir gerenciadores de contexto baseados em classes que também podem ser usados como decoradores de função, executando toda a função em um contexto gerenciado. **(exemplo 3)**

Pesquisar
- Documentação
  contextlib

context_manager1.py
context_manager2.py
context_manager3.py

Python:
Sistema Operacional e Redes

Módulo OS

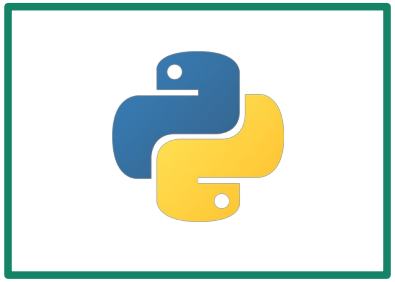
Processo

Um processo é um programa em execução acompanhado dos valores atuais do contador de programas, dos registradores e das variáveis.

É uma abstração para um programa em execução.

Processo
(script em execução)

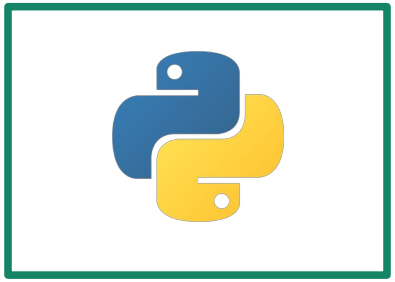
nappacademy



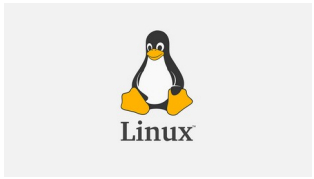
Seu script

Processo no Sistema Operacional

nappacademy

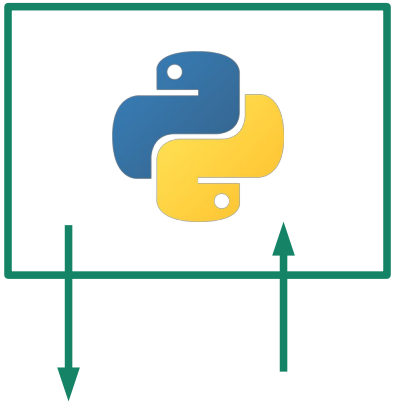


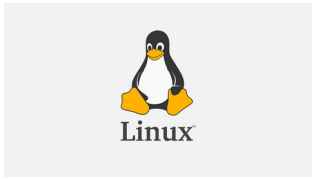
Seu script



O script e o sistema operacional

nappacademy





Módulo OS

nappacademy

Este módulo fornece uma maneira portátil de usar a funcionalidade dependente do sistema operacional.

O design de todos os módulos internos dependentes do sistema operacional do Python é tal que, desde que a mesma funcionalidade esteja disponível, ele usa a mesma interface.

Por exemplo, a função os.stat(path) retorna informações estatísticas sobre o caminho no mesmo formato (que por acaso se originou com a interface POSIX).

modulo_os.py

Socket

Socket

nappacademy

Socket é um fluxo de comunicação entre processos através de uma rede de computadores.

Para reduzir a complexidade do projeto lógico em redes, a maioria delas é organizada como uma pilha de **camadas**, colocada uma sobre as outras.

O número de camadas, o nome, o conteúdo e função de cada camada é o que difere uma rede de outra.

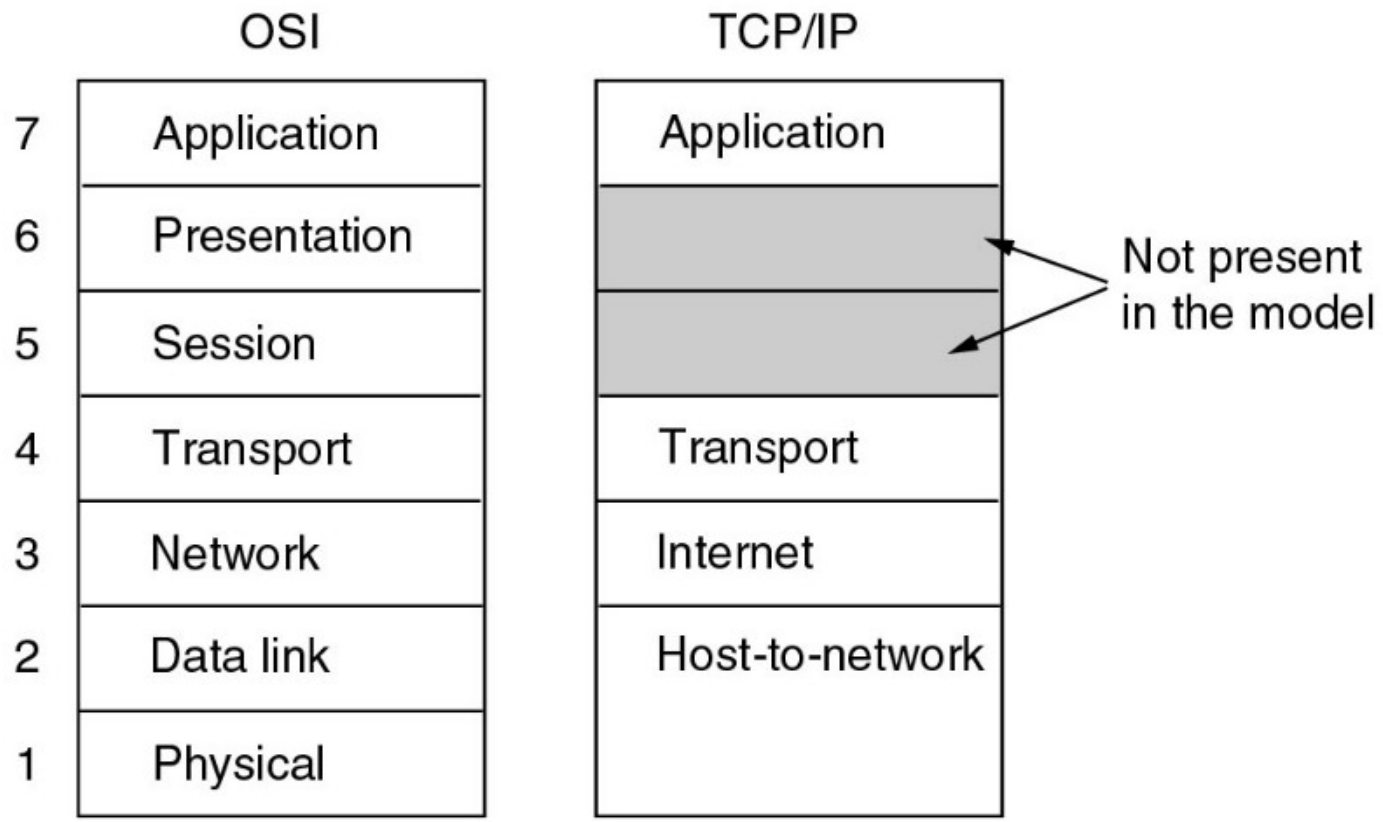
Em cada camada há **protocolos** específicos, que precisam estar presentes no sistemas finais cliente e servidor.

Pesquisar

- Documentação
  Módulo socket

Socket

nappacademy



O modelo de referência TCP/IP.

A. S. Tanenbaum, Computer Networks, 4th ed., Prentice Hall PTR, 2003.

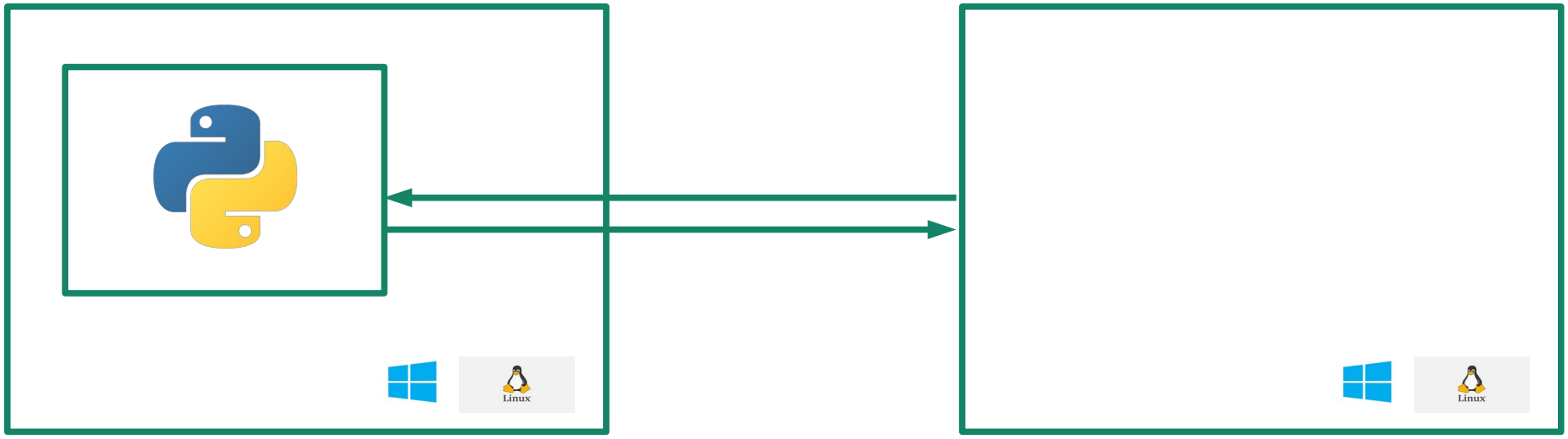
Módulo Socket

nappacademy

Primitiva	Significado
LISTEN	Espera bloqueada por uma conexão de entrada
CONNECT	Estabelece uma conexão com um par que está à espera
RECEIVE	Espera bloqueada por uma mensagem de entrada
SEND	Envia uma mensagem ao par
DISCONNECT	Encerra uma conexão

Cliente

Servidor



Socket: Serviços e padronizações

Um **serviço** é especificado formalmente por um conjunto de **primitivas** (operações) disponíveis para que um processo do usuário acesse o serviço. Essas primitivas informam ao serviço que ele deve executar alguma ação ou relatar uma ação executada por uma entidade par.

Os *Request for Comments* (**RFC**) são documentos usados pela comunidade online. Atualmente, os RFCs são gerenciados pela IETF (*Internet Engineering Task Force*).

Protocolo FTP

nappacademy

O protocolo de transferência de arquivos, conforme definido na RFC 959 e na RFC 1123 , é uma das mais antigas e amplamente utilizadas protocolos na Internet.

O protocolo FTP usa duas conexões TCP paralelas para transferir um arquivo: uma **conexão de controle (21/TCP)** e uma **conexão de dados (20/TCP)**.

Protocolo SSH

nappacademy

O protocolo Secure Shell (SSH) é um protocolo para login remoto seguro e outros serviços de rede segura em uma rede insegura.

Pesquisar

- http://www.paramiko.org/

Protocolo HTTP

Protocolo HTTP

nappacademy

O protocolo de transferência de hipertexto (HTTP) é um protocolo da camada de aplicação. O HTTP tem sido usado pela World-Wide Web global desde 1990. A primeira versão do HTTP, referido como HTTP/0.9, era um protocolo simples para transferência de dados brutos.

Pesquisar
- RFC 2616

Protocolo HTTP
Status Code

nappacademy

1xx Informativo

2xx Sucesso

3xx Redirecionamento

4xx Erro do cliente

5xx Erro do servidor

Pesquisar
- RFC 2616

Protocolo HTTPs

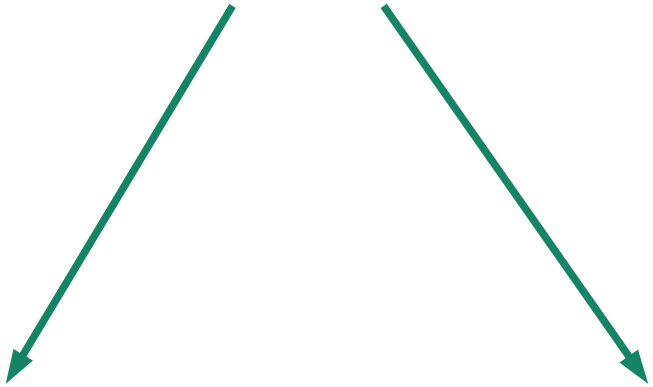
O protocolo HTTP [RFC2616] foi originalmente usado abertamente na Internet. No entanto, o aumento do uso de HTTP para aplicativos confidenciais medidas de segurança necessárias.

O protocolo SSL e seu sucessor TLS [RFC2246] foram projetados para fornecer esta segurança.

Pesquisar
- RFC 2818

Protocolo HTTP(s)

Protocolo HTTP(s)



Consumo de APIs

Web Scrapers

Consumo de APIs

Application Programming Interface

nappacademy

Conjunto de rotinas documentadas e disponibilizadas para que outras aplicações possam consumir suas funcionalidades.

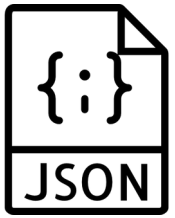
Uma API não precisa ser necessariamente um serviço web.

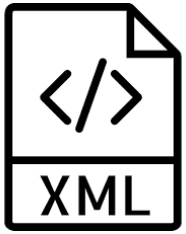
REST Representational State Transfer

nappacademy

É um design de arquitetura de software que define a implementação de um serviço web.

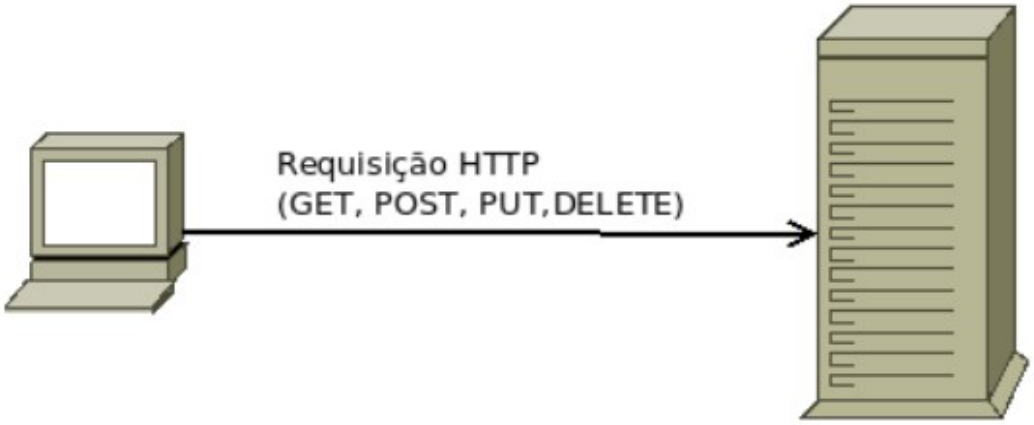
Podem trabalhar com formatos XML, JSON ou outros.





Consumo de APIs

nappacademy



Consumo de APIs

nappacademy

Verbos

nappacademy

GET – Solicita a representação de um recurso

POST – Solicita a criação de um recurso

DELETE – Solicita a exclusão de um recurso

PUT – Solicita a atualização de um recurso

Pesquisar
- RFC 3986

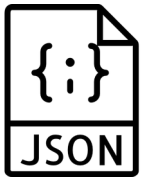
JSON

nappacademy

JavaScript Object Notation

Formatação que permite a troca de mensagens entre sistemas

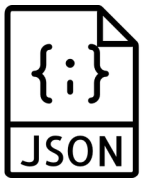
Usa-se o formato chave-valor e listas



JSON

nappacademy

```
dicionario = {}
dicionario['empresa'] = 'Napp Solutions'
dicionario['produtos'] = ['O2O','Connector']
import json
json_object = json.dumps(dicionario, indent = 4)
```



Web Scrapers

Web Scrapers

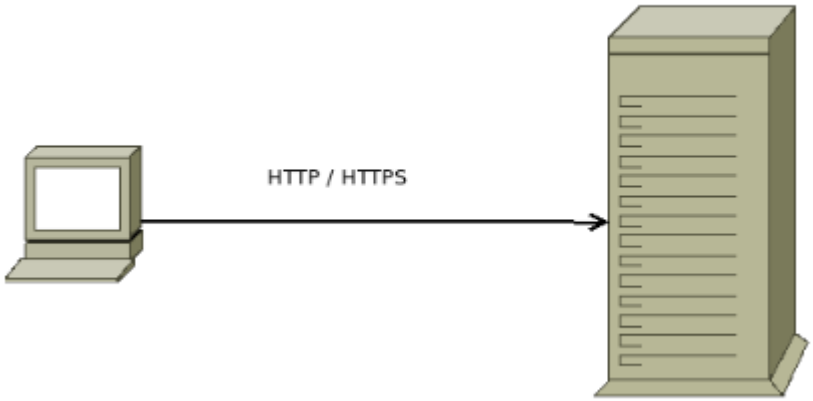
nappacademy

Web Scrapers (raspadores) são pequenos programas que pegam uma URL ou domínio e extraem todos os dados daquele local para onde você quiser, em um formato de sua preferência.

Um raspador da web geralmente é criado para atingir um determinado site ou sites e para coletar informações específicas sobre esses sites.

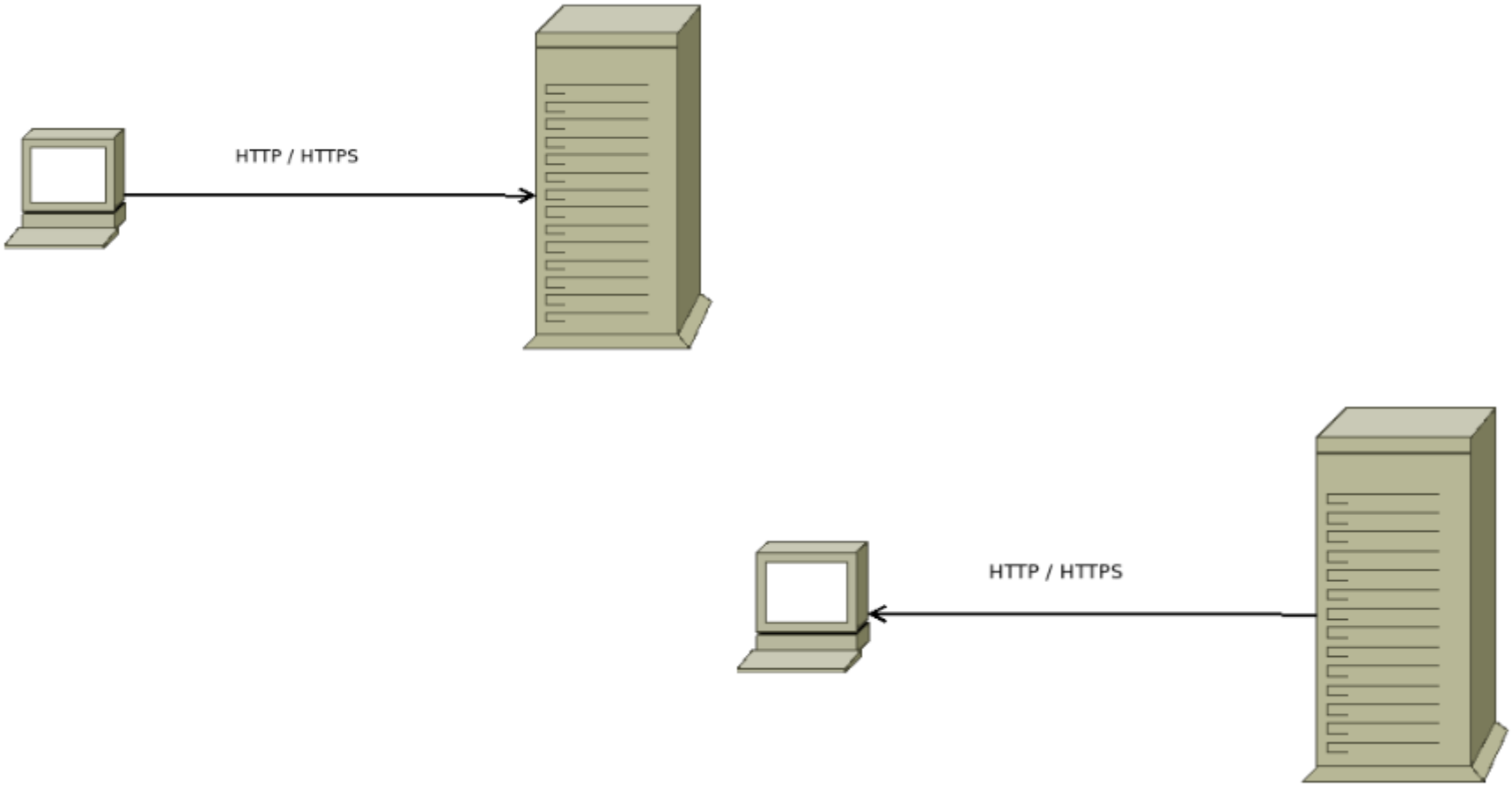
Web Scrapers

nappacademy



Web Scrapers

nappacademy



Bibliotecas Python para Web Scrapers

nappacademy

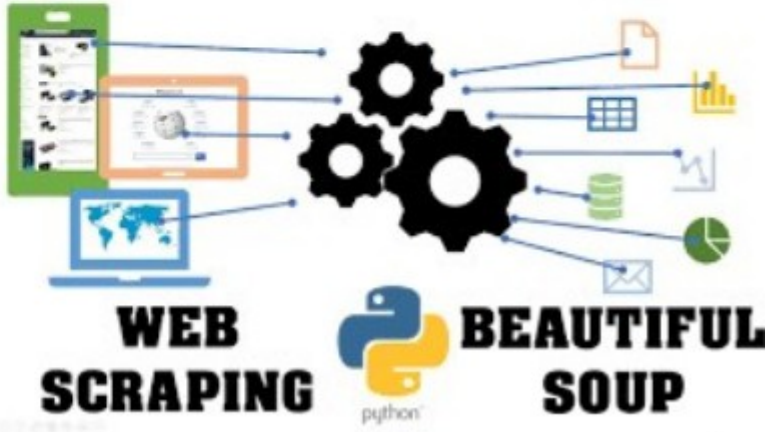
Scrapy

urllib3

MechanicalSoup
A Python library for automating website interaction.

Selenium Python

Requests



nappacademy

Napp Academy

Orlando Saraiva Júnior

Práticas

nappacademy

Como são usados iteradores, iteráveis e geradores na Napp ?

Como os programas desenvolvidos na Napp interagem com o sistema operacional ?

Quais as bibliotecas para consumo de APIs e web crawlers são as mais populares na Napp ?