Python

Hands on

Prof. Dr. Erik Aceiro Antonio

Baseado nos materiais

Henning Schulzrinne

Guido van Rossum (Criador do Python)

https://gvanrossum.github.io

Roteiro

- **Apresentar**
  - Revisar conceitos de Python
- **Hands on**
  - Projeto HUB

gvanrossum.github.io

Guido van Rossum - Personal Home Page

"Gawky and proud of it."

Who I Am

Read my "King's Day Speech" for some inspiration.

I am the author of the Python programming language. See also my resume and my publications list, a brief bio, assorted writings, presentations and interviews (all about Python), some pictures of me, my new blog, and my old blog on Artima.com. I am @gvanrossum on Twitter.

I am retired, working on personal projects (and maybe a book). I have worked for Dropbox, Google, Elemental Security, Zope Corporation, BeOpen.com, CNRI, CWI, and SARA. (See my resume.) I created Python while at CWI.

How to Reach Me

You can send email for me to guido (at) python.org. I read everything sent there, but if you ask me a question about using Python, it's likely that I won't have time to answer it, and will instead refer you to help (at) python.org, comp.lang.python or StackOverflow. If you need to talk to me on the phone or send me something by snail mail, send me an email and I'll gladly email you instructions on how to reach me.

My Name

My name often poses difficulties for Americans.

Pronunciation: in Dutch, the "G" in Guido is a hard G, pronounced roughly like the "ch" in Scottish "loch". (Listen to the sound clip.) However, if you're American, you may also pronounce it as the Italian "Guido". I'm not too worried about the associations with mob assassins that some people have. :-)

Spelling: my last name is two words, and I'd like to keep it that way, the spelling on some of my credit cards notwithstanding. Dutch spelling rules dictate that when used in combination with my first name, "van" is not capitalized: "Guido van Rossum". But when my last name is used alone to refer to me, it is capitalized, for example: "As usual, Van Rossum was right."

Alphabetization: in America, I show up in the alphabet under "V". But in Europe, I show up under "R". And some of my friends put me under "G" in their address book...

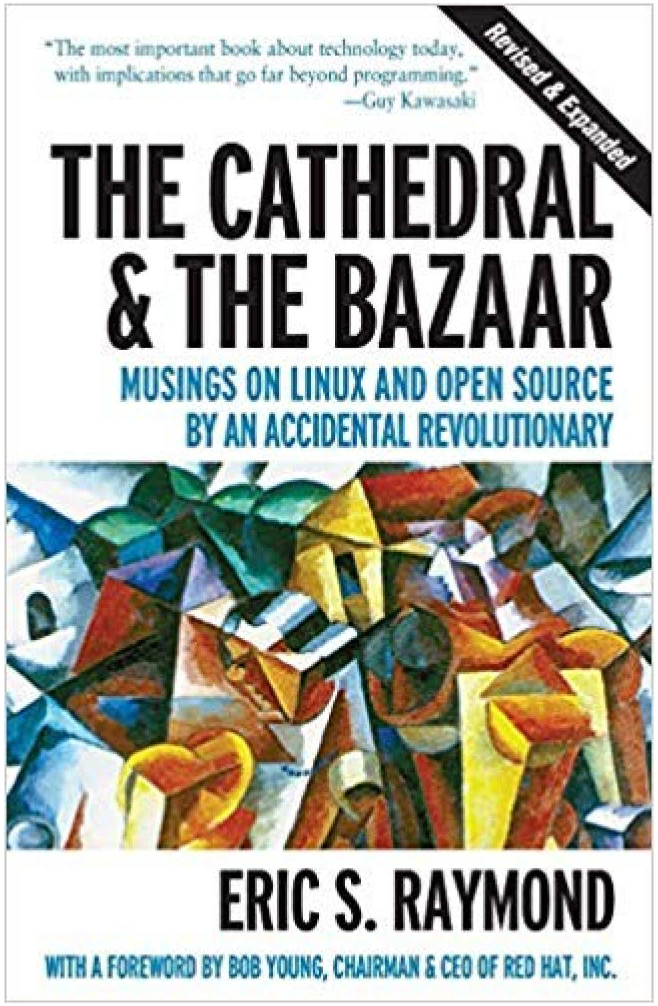
More Hyperlinks

- Here's a collection of essays relating to Python that I've written, including the foreword I wrote for Mark Lutz' book "Programming Python".

Introdução

- Linguagem de script (scripting/extension)
    - origina-se ~1991
- Herança: teaching language (ABC)
    - Tcl: shell
    - perl: string (regex) processing
- Linguagem Orientada a Objetos
    - add-on (OOTcl)

"A educação em Ciência da Computação não pode tornar ninguém um especialista em programação assim como estudar pincéis e pigmentos não pode criar um pintor especialista"


"The most important book about technology today, with implications that go far beyond programming." —Guy Kawasaki
Revised & Expanded
THE CATHEDRAL & THE BAZAAR
MUSINGS ON LINUX AND OPEN SOURCE BY AN ACCIDENTAL REVOLUTIONARY
ERIC S. RAYMOND
WITH A FOREWORD BY BOB YOUNG, CHAIRMAN & CEO OF RED HAT, INC.

https://www.freecodecamp.org/news/what-programming-language-should-i-learn-first-19a33b0a467d/

http://tiny.cc/3kgpgz

http://tiny.cc/6igpgz

Python Filosofia

- Coerencia
  - Not hard to read
  - Write
  - Maintain
- Poderosa – Mineração de dados e IA
- Escopo
  - Rápido desenvolvimento
- Objetos
- Integração
  - Sistemas Hibridos

Python Features (1/2)

Lutz, *Programming Python*

no compiling or linking	rapid development cycle
no type declarations	simpler, shorter, more flexible
automatic memory management	garbage collection
high-level data types and operations	fast development
object-oriented programming	code structuring and reuse, C++
embedding and extending in C	mixed language systems
classes, modules, exceptions	"programming-in-the-large" support
dynamic loading of C modules	simplified extensions, smaller binaries
dynamic reloading of C modules	programs can be modified without stopping

Python Features (2/2)

Lutz, *Programming Python*

universal "first-class" object model	fewer restrictions and rules
run-time program construction	handles unforeseen needs, end-user coding
interactive, dynamic nature	incremental development and testing
access to interpreter information	metaprogramming, introspective objects
wide portability	cross-platform programming without ports
compilation to portable byte-code	execution speed, protecting source code
built-in interfaces to external services	system tools, GUIs, persistence, databases, etc.

Python

- Envolve elementos das seguintes linguagens
  - C++, Modula-3 (modules), ABC, Icon (slicing)
- Familiariadade com
  - Perl, Tcl, Scheme, REXX, BASIC dialects

Uso do Python

- Ferramentas de Shell Scripts
  - system admin tools
  - **command line programs (Programas de Linha de Comando)**

- **RAD - Rapid App Prototyping/Development**

- Linguagem Baseada em Módulos
- GUI
- Acesso a Bando de Dados
- Programação distribuída
- **Scripts para Internet**

Por que não usar o Python

- Existem muitas linguagens de scripts

- Não é eficiente quanto o C
  - Porque não ???

Using python

- /usr/local/bin/python
  - `#! /usr/bin/env python`
- interactive use

```
Python 1.6 (#1, Sep 24 2000, 20:40:45)  [GCC 2.95.1 19990816 (release)] on sunos5
Copyright (c) 1995-2000 Corporation for National Research Initiatives.
All Rights Reserved.
Copyright (c) 1991-1995 Stichting Mathematisch Centrum, Amsterdam.
All Rights Reserved.
>>>
```

- `python –c` *`command`* `[`*`arg`*`] ...`
- `python –i` *`script`*
  - read script first, then interactive

Python structure

- **módulos:** como Python ou C
  - extensions
  - import, top-level via from, reload
- **declaração**
  - control flow
  - Criação de objetos
  - **Indentação ao invés {}**
- **objetos**
  - TUDO É UM OBJETO
  - Automaticamente reciclado quando não é mais usado

First example

```
#!/usr/local/bin/python
# import systems module
import sys
marker = '::::::'
for name in sys.argv[1:]:
  input = open(name, 'r')
  print marker + name
  print input.read()
```

Basic operations

- Atribuição (assignment):
  - `size = 40`
  - `a = b  = c = 3`
- Númeroos
  - integer, float
  - complex numbers: `1j+3, abs(z)`
- Strings
  - `'hello world', 'it\'s hot'`
  - `"bye world"`
  - continuation via \ or use """ long text """"

String operations

- concatenação **+** ou com *neighbors*
  - `word = 'Help' + x`
  - `word = 'Help' 'a'`
- Subindice de strings
  - `'Hello'[2]` → 'l'
  - slice: `'Hello'[1:2]` → 'el'
  - `word[-1]` → last character
  - `len(word)` → 5
  - **immutable**: *cannot assign to subscript*

Lists

- Listas podem ser heterogêneas
  - `a = ['spam', 'eggs', 100, 1234, 2*2]`
- Listas podem ser indexadas
  - `a[0]` → spam
  - `a[:2]` → ['spam', 'eggs']
- Listas podem ser manipuladas
  - `a[2] = a[2] + 23`
  - `a[0:2] = [1,12]`
  - `a[0:0] = []`
  - `len(a)` → 5

Basic programming

```
a,b = 0, 1
# non-zero = true
while b < 10:
  # formatted output, without \n
   print b,
  # multiple assignment
   a,b = b, a+b
```

Control flow: if

```
x = int(raw_input("Please enter #:"))
if x < 0:
  x = 0
  print 'Negative changed to zero'
elif x == 0:
  print 'Zero'
elif x == 1:
  print 'Single'
else:
  print 'More'
```

- **ATENÇÃO – NÃO TEM SWITH...CASE**

Control flow: for

```
a = ['cat', 'window', 'defenestrate']
for x in a:
  print x, len(x)
```

- ***no arithmetic progression, but***
  - `range(10)` → `[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]`

  -
    ```
    for i in range(len(a)):
     print i, a[i]
    ```
- Não modifica a sequencia interna

For em Python e C

In Python:

```
for i in range(20):
    if i%3 == 0:
        print i
        if i%5 == 0:
            print "Bingo!"
    print "---"
```

In C:

```
for (i = 0; i < 20; i++)
{
    if (i%3 == 0) {
        printf("%d\n", i);
        if (i%5 == 0) {
            printf("Bingo!\n"); }
    }
    printf("---\n");
}
```

```
0
Bingo!
---
---
---
3
---
---
---
6
---
---
---
9
---
---
---
12
---
---
---
15
Bingo!
---
---
---
18
---
---
```

Loops: break, continue, else

- `break`, `continue` como em C
- `else` **pode ser usado em loops exaustão**

```
for n in range(2,10):
  for x in range(2,n):
    if n % x == 0:
      print n, 'equals', x, '*', n/x
      break
  else:
    # loop fell through without finding a factor
    print n, 'is prime'
```

Do nothing

- Passa e não faz nada
  - *pass does nothing*
- *syntactic filler*

```
while 1:
 pass
```

Definição de Funções

```
def fib(n):
  """Print a Fibonacci series up to n."""
  a, b = 0, 1
  while b < n:
    print b,
    a, b = b, a+b

>>> fib(2000)
```

- Primeira linha é um *docstring*
- Procura variáveis locais, depois globais
- Requer atribuição de variáveis globais

Functions: default argument values

```
def ask_ok(prompt, retries=4,
  complaint='Yes or no, please!'):
  while 1:
    ok = raw_input(prompt)
    if ok in ('y', 'ye', 'yes'): return 1
    if ok in ('n', 'no'): return 0
    retries = retries - 1
    if retries < 0: raise IOError,
  'refusenik error'
    print complaint

>>> ask_ok('Really?')
```

Keyword arguments

- Último argumento como keywords

```
def parrot(voltage, state='a stiff', action='voom',
   type='Norwegian blue'):
  print "-- This parrot wouldn't", action,
  print "if you put", voltage, "Volts through it."
  print "Lovely plumage, the ", type
  print "-- It's", state, "!"

parrot(1000)
parrot(action='VOOOM', voltage=100000)
```

Lambda forms

- Funções anonimas
  - *anonymous functions*

```
def make_incrementor(n):
  return lambda x: x + n

f = make_incrementor(42)
f(0)
f(1)
```

Métodos de List

- `append(`*`x`*`)`
- `extend(`*`L`*`)`
  - append all items in list (like Tcl lappend)
- `insert(`*`i`*`,`*`x`*`)`
- `remove(`*`x`*`)`
- `pop([i]), pop()`
  - create stack (FIFO), or queue (LIFO) → pop(0)
- `index(`*`x`*`)`
  - return the index for value *x*

List methods

- `count(x)`
  - how many times x appears in list
- `sort()`
  - sort items in place
- `reverse()`
  - reverse list

Programação Funcional

- `filter(`*`function, sequence`*`)`

```
def f(x): return x%2 != 0 and x%3
filter(f, range(2,25))
```

- `map(`*`function, sequence)`*
  - call function for each item
  - return list of return values
- `reduce(`*`function, sequence`*`)`
  - return a single value
  - call binary function on the first two items
  - then on the result and next item
  - iterate

List comprehensions (2.0)

- Create lists without `map()`, `filter()`, `lambda`
- = expression followed by for clause + zero or more for or of clauses

```
>>> vec = [2,4,6]
>>> [3*x for x in vec]
[6, 12, 18]
>>> [{x: x**2} for x in vec}
[{2: 4}, {4: 16}, {6: 36}]
```

List comprehensions

- Produto Vetorial *cross products*:

```
>>> vec1 = [2,4,6]
>>> vec2 = [4,3,-9]
>>> [x*y for x in vec1 for y in vec2]
[8,6,-18, 16,12,-36, 24,18,-54]
>>> [x+y for x in vec1 and y in vec2]
[6,5,-7,8,7,-5,10,9,-3]
>>> [vec1[i]*vec2[i] for i in
  range(len(vec1))]
[8,12,-54]
```

List comprehensions

- can also use `if`:

```
>>> [3*x for x in vec if x > 3]
[12, 18]
>>> [3*x for x in vec if x < 2]
[]
```

del – removindo itens

- remove by index, not value
- remove slices from list (rather than by assigning an empty list)

```
>>> a = [-1,1,66.6,333,333,1234.5]
>>> del a[0]
>>> a
[1,66.6,333,333,1234.5]
>>> del a[2:4]
>>> a
[1,66.6,1234.5]
```

Tuples/Sequences

- lists, strings, **tuples**: examples of *sequence* type
- tuple = values separated by commas

```
>>> t = 123, 543, 'bar'
>>> t[0]
123
>>> t
(123, 543, 'bar')
```

Tuples

- Tuples podem ser aninhadas

```
>>> u = t, (1,2)
>>> u
((123, 542, 'bar'), (1,2))
```

- Tipos de estrutudas:
  - (x,y) coordinates
  - database records
- como strings, immutable → *can't assign to individual items*

Tuples

- Empty tuples: ()

```
>>> empty = ()
>>> len(empty)
0
```

- one item → trailing comma

```
>>> singleton = 'foo',
```

Tuples

- Abrir elementos
  - unpacking → distribute elements across variables

```
>>> t = 123, 543, 'bar'
>>> x, y, z = t
>>> x
123
```

- **packing** always creates tuple
- **unpacking** works for any sequence

Dicionários
Dictionaries

- Como Tcl/awk
  - indexed by keys
  - keys are any immutable type: e.g., tuples
  - but not lists (mutable!)
- **uses 'key: value' notation**

```
>>> tel = {'hgs' : 7042, 'lennox': 7018}
>>> tel['cs'] = 7000
>>> tel
```

Dictionaries

- Não tem ordem particular
- Remoção de elementos com del

```
>>> del tel['foo']
```

- **keys()** method → unsorted list of keys

```
>>> tel.keys()
['cs', 'lennox', 'hgs']
```

- use **has_key()** checar existência

```
>>> tel.has_key('foo')
0
```

Condições

- can check for sequence membership with is and is not:

```
>>> if (4 in vec):
...    print '4 is'
```

- chained comparisons: a less than b AND b equals c:

```
a < b == c
```

- and and or are short-circuit operators:
  - evaluated from left to right
  - stop evaluation as soon as outcome clear

Condições

- Can assign comparison to variable:

```
>>> s1,s2,s3='', 'foo', 'bar'
>>> non_null = s1 or s2 or s3
>>> non_null
foo
```

- Unlike C, no assignment within expression

Modules

- Coleçoes de funções, classes...
- Definição pode ser importada
- [nome_do_modulo] + .py
- e.g., create module `fibo.py`

def fib(n): # write Fib. series up to n

…

def fib2(n): # return Fib. series up to n

Modules

- Importe o modulo:

```
import fibo
```

- Use modules via "name space":

```
>>> fibo.fib(1000)
>>> fibo.__name__
'fibo'
```

- can give it a local name:

```
>>> fib = fibo.fib
>>> fib(500)
```

Modules

- Funções + Definições globais
- Executado apenas se o módulo for importado
- Módulos podem ter campos privados
- Evite nomes de classes globais
- Acessível como *module.globalname*
- Pode ser importado:

```
>>> from fibo import fib, fib2
>>> fib(500)
```

Pode ser importado todas definições

```
>>> from fibo import *
```

Module search path

- Busca no diretório local
- Lista de diretórios em PYTHONPATH environment variable
- Uso da instalação padrão, e.g., .:/usr/local/lib/python
- uses sys.path

```
>>> import sys
>>> sys.path
['', 'C:\\PROGRA~1\\Python2.2', 'C:\\Program Files\\
   Python2.2\\DLLs', 'C:\\Program Files\\Python2.2\\lib', 'C:\\
   Program Files\\Python2.2\\lib\\lib-tk', 'C:\\Program Files\\
   Python2.2', 'C:\\Program Files\\Python2.2\\lib\\site-
   packages']
```

Standard modules

- system-dependent list
- always sys module

```
>>> import sys
>>> sys.p1
'>>> '
>>> sys.p2
'... '
>>> sys.path.append('/some/directory')
```

Module listing

- use `dir()` for each module

```
>>> dir(fibo)
['___name___', 'fib', 'fib2']
>>> dir(sys)
['__displayhook__', '__doc__', '__excepthook__', '__name__', '__stderr__', '__st
din__', '__stdout__', '_getframe', 'argv', 'builtin_module_names', 'byteorder',
'copyright', 'displayhook', 'dllhandle', 'exc_info', 'exc_type', 'excepthook', '
exec_prefix', 'executable', 'exit', 'getdefaultencoding', 'getrecursionlimit', '
getrefcount', 'hexversion', 'last_type', 'last_value', 'maxint', 'maxunicode', '
modules', 'path', 'platform', 'prefix', 'ps1', 'ps2', 'setcheckinterval', 'setpr
ofile', 'setrecursionlimit', 'settrace', 'stderr', 'stdin', 'stdout', 'version',
 'version_info', 'warnoptions', 'winver']
```

Classes

- Mistura de C++ e Modula-3
- Permite herânça multipla
- Classes derivadas podem sobrescrever qualquer método de sua classe base
- Métodos podem chamar métodos da classe base com o mesmo nome
- Objetos tem campos privados
- C++ terms:
  - Todas as classe são publicas
  - *all member functions are virtual*
  - *no constructors or destructors (not needed)*

Classes

- Classes são objetos
- *built-in types* **cannot be used as base classes by user**
- Operadores artiméticos podem ser sobrescritos para serem redefinidos
  - Como C++
  - Diferente de Java

Definição de Classe

```
class ClassName:
   <statement-1>
   ...
   <statement-N>
```

- must be executed
- can be executed conditionally (see Tcl)
- creates new namespace

Class objects

- `obj.name` references (plus module!):

```
class MyClass:
  "A simple example class"
  i = 123
  def f(self):
    return 'hello world'
>>> MyClass.i
123
```

- `MyClass.f` is method object

Class objects

- class *instantiation*:

```
>>> x = MyClass()
>>> x.f()
'hello world'
```

- Cria novas instancias
  - note `x = MyClass` vs. `x = MyClass()`
- `__init__()` *special method for initialization of object*

```
def __init__(self,realpart,imagpart):
  self.r = realpart
  self.i = imagpart
```

Instancias

- Referências de atributos
- data attributes (C++/Java data members)
  - created dynamically

```
x.counter = 1
while x.counter < 10:
  x.counter = x.counter * 2
print x.counter
del x.counter
```

Métodos

- Chamado imediatamente:

```
x.f()
```

- *can be referenced*:

```
xf = x.f
while 1:
    print xf()
```

- object is passed as first argument of function → 'self'
  - **x.f()** é igual **MyClass.f(x)**

Another example

- bag.py

```
class Bag:
  def __init__(self):
    self.data = []
  def add(self, x):
    self.data.append(x)
  def addtwice(self,x):
    self.add(x)
    self.add(x)
```

Another example, cont'd.

- invoke:

```
>>> from bag import *
>>> l = Bag()
>>> l.add('first')
>>> l.add('second')
>>> l.data
['first', 'second']
```

Herançca
Inheritance

```
class DerivedClassName(BaseClassName)
   <statement-1>
   ...
   <statement-N>
```

- search class attribute, descending chain of base classes
- may override methods in the base class
- call directly via `BaseClassName.method`

Herança Multipla
Multiple inheritance

```
class DerivedClass(Base1,Base2,Base3):
    <statement>
```

- depth-first, left-to-right
- problem: class derived from two classes with a common base class

Campos Privados

- Não tem real suporte
- `__var` is replaced by `_classname_var`
- Previne de modificações acidentais, mas não é confiável

~ C structs

- Parecido com Structs do C:

```
class Employee:
  pass

john = Employee()
john.name = 'John Doe'
john.dept = 'CS'
john.salary = 1000
```

Exceptions

- syntax (parsing) errors

```
while 1 print 'Hello World'
File "<stdin>", line 1
  while 1 print 'Hello World'
                    ^
SyntaxError: invalid syntax
```

- exceptions
  - run-time errors
  - e.g., `ZeroDivisionError`, `NameError`, `TypeError`

Capturando...
Handling exceptions

```
while 1:
  try:
    x = int(raw_input("Please enter a number: "))
    break
  except ValueError:
    print "Not a valid number"
```

- Primeiro, executa `try` clause
  - if no exception, skip `except` clause
  - If exception, skip rest of `try` clause and use except clause
  - if no matching exception, attempt outer `try` statement

Handling exceptions

- try.py

```
import sys
for arg in sys.argv[1:]:
 try:
        f = open(arg, 'r')
 except IOError:
        print 'cannot open', arg
 else:
        print arg, 'lines:', len(f.readlines())
        f.close
```

- e.g., as `python try.py *.py`

Requests via HTTP

```
pip install requests

import requests
r =requests.get('https://xkcd.com/1906/')
r.status_code
r.headers
r.text
```

Requests com Payloads HTTP

```
pip install requests


import requests
ploads = {'things':2,'total':25}
r = requests.get('https://httpbin.org/get',params=ploads)
print(r.text)
print(r.url)
```

Post com Payloads HTTP

```
pip install requests


import requests

pload = {'username':'Olivia','password':'123'}

r = requests.post('https://httpbin.org/post',data =
  pload)

print(r.text)
```

Post com Payloads HTTP

```
pip install requests


import requests
pload = {'username':'olivia','password':'123'}
r = requests.post('https://httpbin.org/post',data =
  pload)


print(r.json())
```

Post com Payloads HTTP

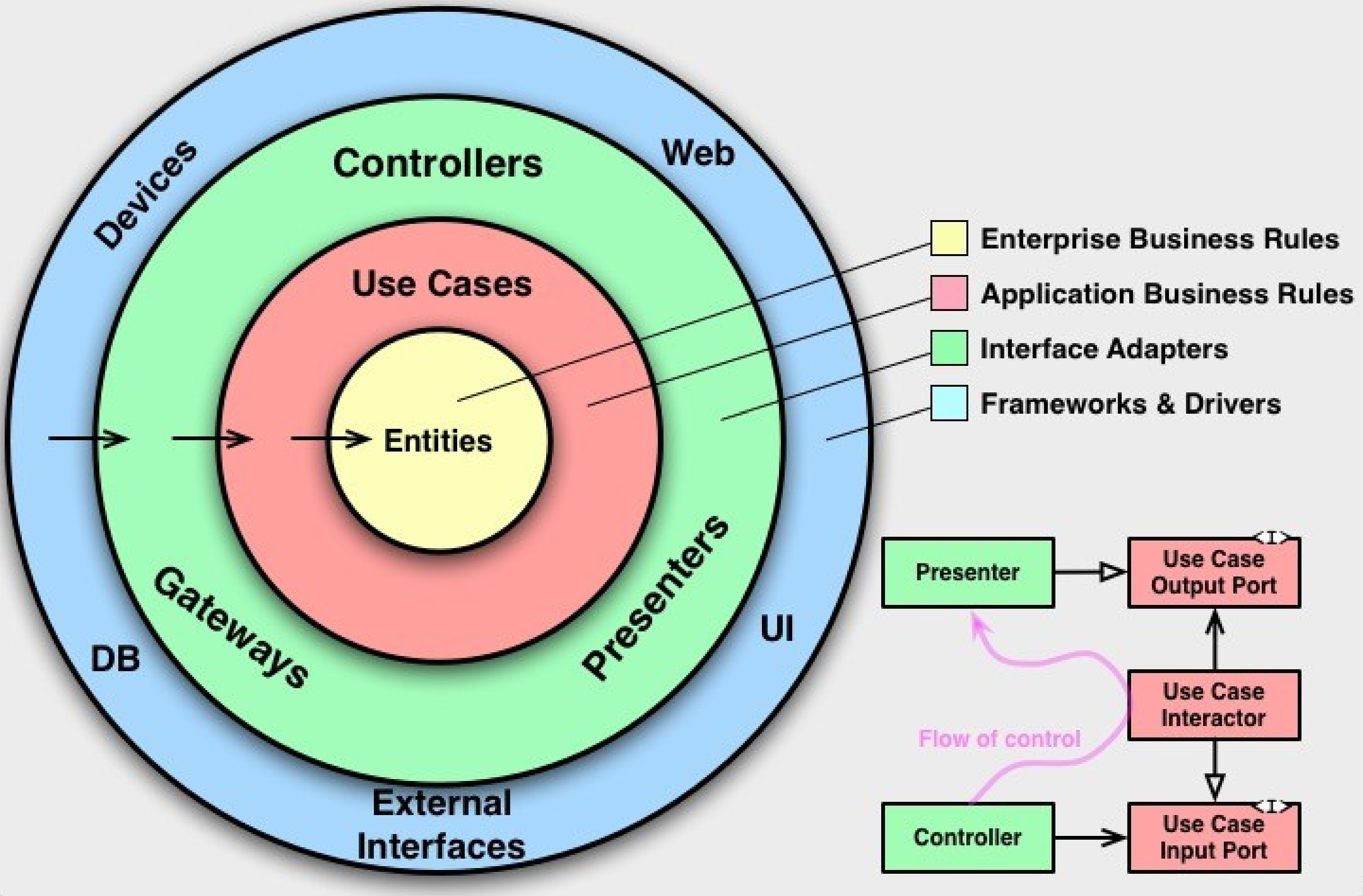
```
pip install requests


import requests
pload = {'username':'olivia','password':'123'}
r = requests.post('https://httpbin.org/post',data =
  pload)
r_dictionary= r.json()
print(r_dictionary['form'])
```

Hands on

Projeto HUB

The Clean Architecture



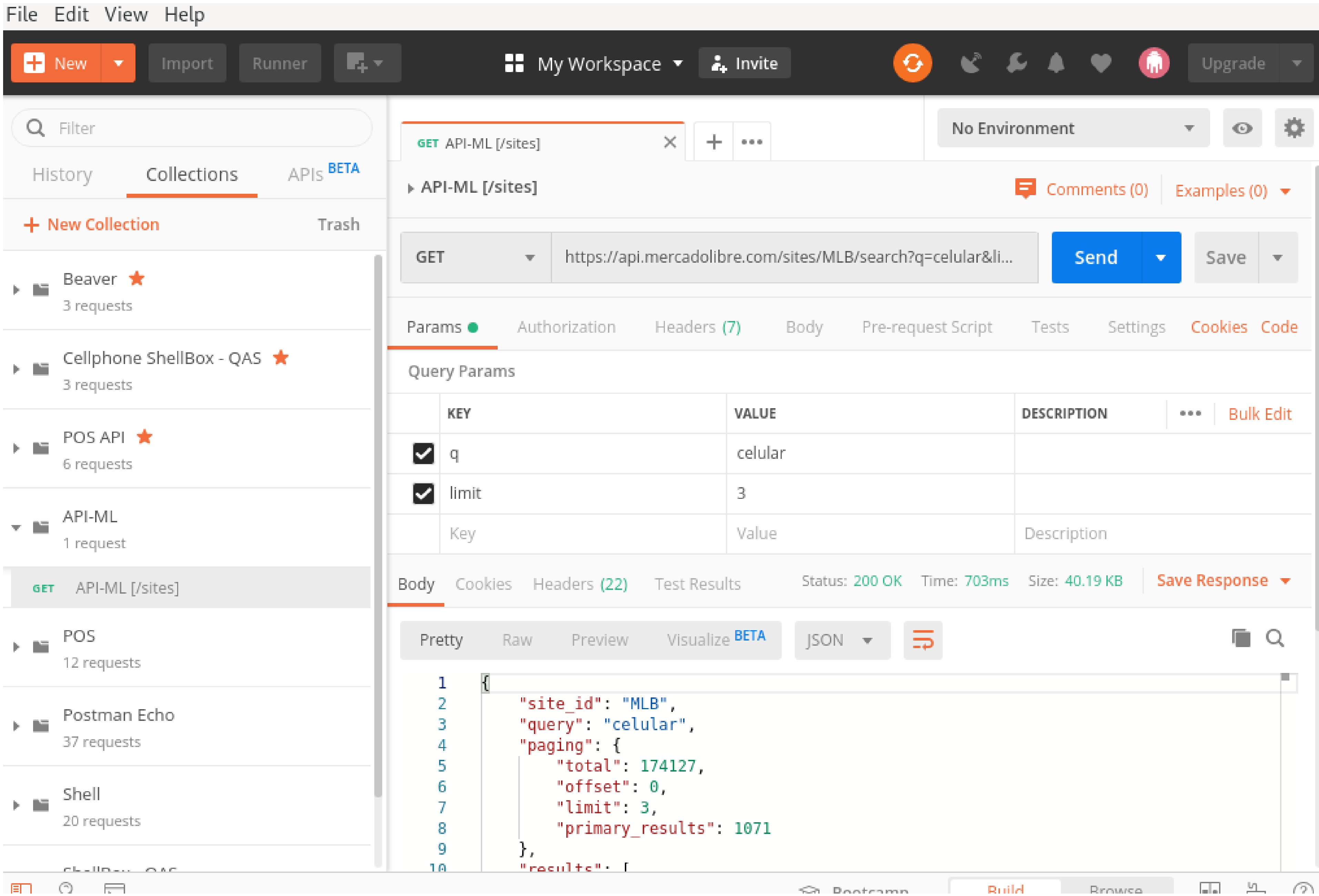
Projeto HUB (POO)

Passos

- Criar arquitetura clean code
- Criar módulos necessários
- Criar classe para leitura da API do ML
  - Usar APIs e Postman (sugestão)
- Criar classe para leitura de parâmetros via CMD
- Preparar estrutura do projeto features

USE

→ https://developers.mercadolivre.com.br/pt_br/api-docs-pt-br



Beaver ★ — 3 requests
Cellphone ShellBox - QAS ★ — 3 requests
POS API ★ — 6 requests
API-ML — 1 request
GET API-ML [/sites]
POS — 12 requests
Postman Echo — 37 requests
Shell — 20 requests

GET API-ML [/sites]

API-ML [/sites]

GET https://api.mercadolibre.com/sites/MLB/search?q=celular&li...

Query Params

	KEY	VALUE	DESCRIPTION
✓	q	celular	
✓	limit	3	
	Key	Value	Description

Status: 200 OK Time: 703ms Size: 40.19 KB

```
{
    "site_id": "MLB",
    "query": "celular",
    "paging": {
        "total": 174127,
        "offset": 0,
        "limit": 3,
        "primary_results": 1071
    },
    "results": [
```