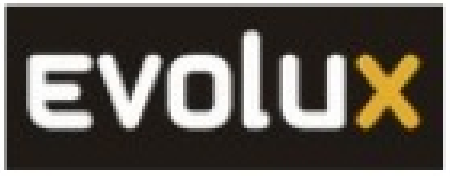
PSW

Prof. Msc. Carlos Anderson

Introdução

- Lançada em 1991 por Guido van Rossum, é uma linguagem livre (até para projetos comerciais) e hoje pode-se programar para desktops, web e mobile:
  - dinâmica (não precisa declarar tipos);
  - interpretada (usa um interpretador antes do SO);
  - robusta (código seguro);
  - multiplataforma (vários SO);
  - multi-paradigma (orientação à objetos, funcional, refletiva e imperativa) ;
  - roda em JVM e .NET Framework;

Quem usa python ?



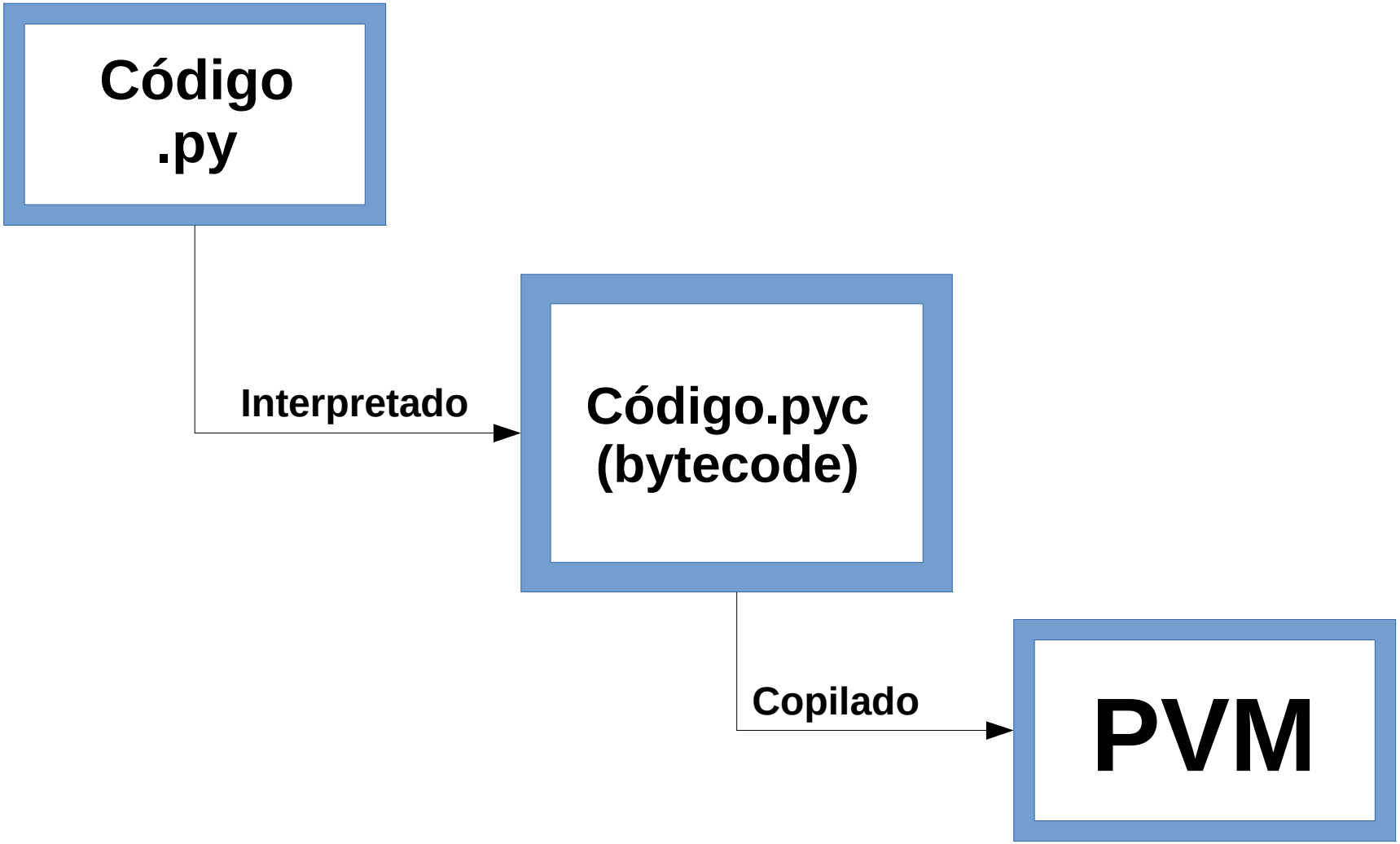
Desvantagens

- Incompatibilidade entre versões;
  - Python 2 foi o padrão da linguagem por muito tempo e receberá atualizações de segurança até 2020.
  - Python 3 introduziu algumas mudanças que quebraram a compatibilidade com a versão anterior o que criou a necessidade de se manter duas versões da linguagem. Constantemente evoluindo e recebendo novas funcionalidades, que não estarão presentes na versão anterior.

Do que preciso para iniciar?

- Interpretador Python:
  - Windows: baixar e instalar;
  - Linux e Mac OS: Provavelmente já vem instalado, mas pode ser instalado também;
  - Disponível em outras plataformas: AIX, Palm OS, AS/400, PSP, etc.

Funciona assim



Instalando o Python no Linux

- Abra o terminal e digite:
  - $ sudo apt-get update
  - $ sudo apt-get install python3

- No terminal digite:
  - $ python3

    >>> print(“Olá mundo!”)

A sintaxe

- Um comando por linha
  - Usar ; para mais de uma linha
  - Usar \ para continuar em outra linha
- Bloco de comando por indentação
  - Não misture Tabs e espaços
- Comentários
  - Caracter # ou """strings multilinhas"""
  - Diferencia maiúsculas de minúsculas

A sintaxe

- Diferencia entre maiúscula e minúsculas;
- Nome deve iniciar com letra ou "_";
- Restante do nome pode conter letras, números e "_";
- Não é permitido o uso de palavras reservadas mas quando necessário costuma-se usar um "_" no fim do nome (ex. "from_").

A sintaxe

- bool()
- True, False
- 0, 0.0, [], (), {}, "", set(),
  None, ... - Falso
- ==, !=, >, >=, <, <=, is, is not,
  and, or, in e not in
- Usar "is None" para
  comparações com None
- Prefira "if valor: ..." no lugar de
  "valor == True" ou "valor != 0".

A sintaxe

- Atribuição simples (=);
- Atribuição "aumentada": +=, -=, *=, /=, //=, **=, %=, |=, &=, ^=, <<= e >>=
- Atribuição por tupla

  a, b = b, a

  (a, b) = (b, a)

```
>>> a = "a"
>>> b = "b"
>>> c, d = "cd"
>>> e, f = "e", "f"
>>> print a, b, c, d, e, f
a b c d e f
>>> e, f = f, e
>>> print a, b, c, d, e, f
a b c d f e
>>> a += b
>>> print a
ab
>>> a *= 5
>>> print a
abababababab
```

A sintaxe

- Comando de decisão;
- Executa o bloco de código em if ou elif caso a condição for verdadeira;
- Se nenhuma condição for verdadeira executa o bloco else;
- Expressão if usada entre parênteses

```
a = input("A:")
b = input("B:")

if a > b:
    print "A é maior que B"
elif b > a:
    print "B é maior que A"
else:
    print "A e B são iguais"

print ("A é maior" if a > b \
        else "A não é maior")
```

A sintaxe

- Executa o bloco de código, em loop, enquanto a condição for verdadeira;
- A cláusula 'else' pode ser usada e será executada caso o loop termine normalmente (sem break);

```
import sys, time

d = input("Despertar (HH:MM): ")

while time.strftime("%H:%M") != d:
  try:
    print("\b\b\b\b\btick"), sys.stdout.flush()
    time.sleep(0.5)

    print ("\b\b\b\b\btack"), sys.stdout.flush()
    time.sleep(0.5)

  except KeyboardInterrupt:
    break
else:
  print ("\n\nTRIM!\a\a\a")
  sys.exit(0)

print ("\n\nInterrompido!")
```

A sintaxe

- Itera sobre os elementos de um objeto (iterador, seqüência, …);
- Pode ser usado na forma: 'for key, valor in dic.items(): ...' onde ele faz atribuição por tupla;

```
import sys

for linha in sys.stdin:
    if not linha.strip(): continue
    if "FIM" in linha: break
    print "#", linha.rstrip()
else:
    print "# FIM DO ARQUIVO"
```

```
>>> a = range(10)
>>> print a
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> for i in a:
...     print i,
...
0 1 2 3 4 5 6 7 8 9
```

Tipos de dados

- Em Python, todos os tipos de dados são objetos;
- Quando fazemos algo como: int("0") estamos, na verdade instanciando um objeto do tipo int() passando "0" para seu construtor.

Listas

```
>>> ['a', 'b']
['a', 'b']
>>> ['a', 'b'] + ['c', 'd']
['a', 'b', 'c', 'd']

>>> a = ['a', 'b', 'c', 'd']
>>> a
['a', 'b', 'c', 'd']

>>> a[0] = 'X'
>>> a
['X', 'b', 'c', 'd']

>>> a += "efg"
>>> a
['X', 'b', 'c', 'd', 'e', 'f', 'g']
```

```
>>> a + "ghi"
TypeError: can only concatenate list
           (not "str") to list

>>> ['a', 'b', 'c', 'd'][0]
'a'

>>> ['a', 'b', 'c', 'd'][0:2]
['a', 'b']
```

Listas

```
>>> a = [1, 2, 3]
>>> a.append(4)
>>> a
[1, 2, 3, 4]
>>> a.extend([5, 6])
>>> a
[1, 2, 3, 4, 5, 6]
>>> a.insert(2, 2.5)
>>> a
[1, 2, 2.5, 3, 4, 5, 6]
>>> a.pop()
6
>>> a
[1, 2, 2.5, 3, 4, 5]
>>> a.reverse()
>>> a
[5, 4, 3, 2.5, 2, 1]
```

```
>>> a.sort()
>>> a
[1, 2, 2.5, 3, 4, 5]
>>> ['err', 'ok', 'err'].count('err')
2
```

Dicionários

```
>>> d = { "c1": "v1" }
>>> dic["c2"] = "v2"
>>> d
{'c1': 'v1', 'c2': 'v2'}
>>> d[1] = "chave numerica"
>>> d
{'c1': 'v1', 1: 'chave numerica',
'c2': 'v2'}
>>> tuplachave = (1, 2, 3)
>>> d[tuplachave] = "objeto chave"
>>> d
{'c1': 'v1', 1: 'chave numerica',
 'c2': 'v2',
 (1, 2, 3): 'objeto chave'}
>>> d.update({'c3':'v3', 'c4':'v4'})
>>> d
{1: 'chave numerica', 'c3': 'v3',
 'c2': 'v2', 'c1': 'v1', 'c4': 'v4',
 (1, 2, 3): 'objeto chave'}
>>> d.pop('c4')
'v4'
>>> d
{1: 'chave numerica', 'c3': 'v3',
 'c2': 'v2', 'c1': 'v1',
 (1, 2, 3): 'objeto chave'}
>>> d.items()
[(1, 'chave numerica'), ('c3', 'v3'),
 ('c2', 'v2'), ('c1', 'v1'),
 ((1, 2, 3), 'objeto chave')]
>>> d.keys()
[1, 'c3', 'c2', 'c1', (1, 2, 3)]
>>> d.values()
['chave numerica', 'v3', 'v2', 'v1',
 'objeto chave']
```

Funções e parâmetros

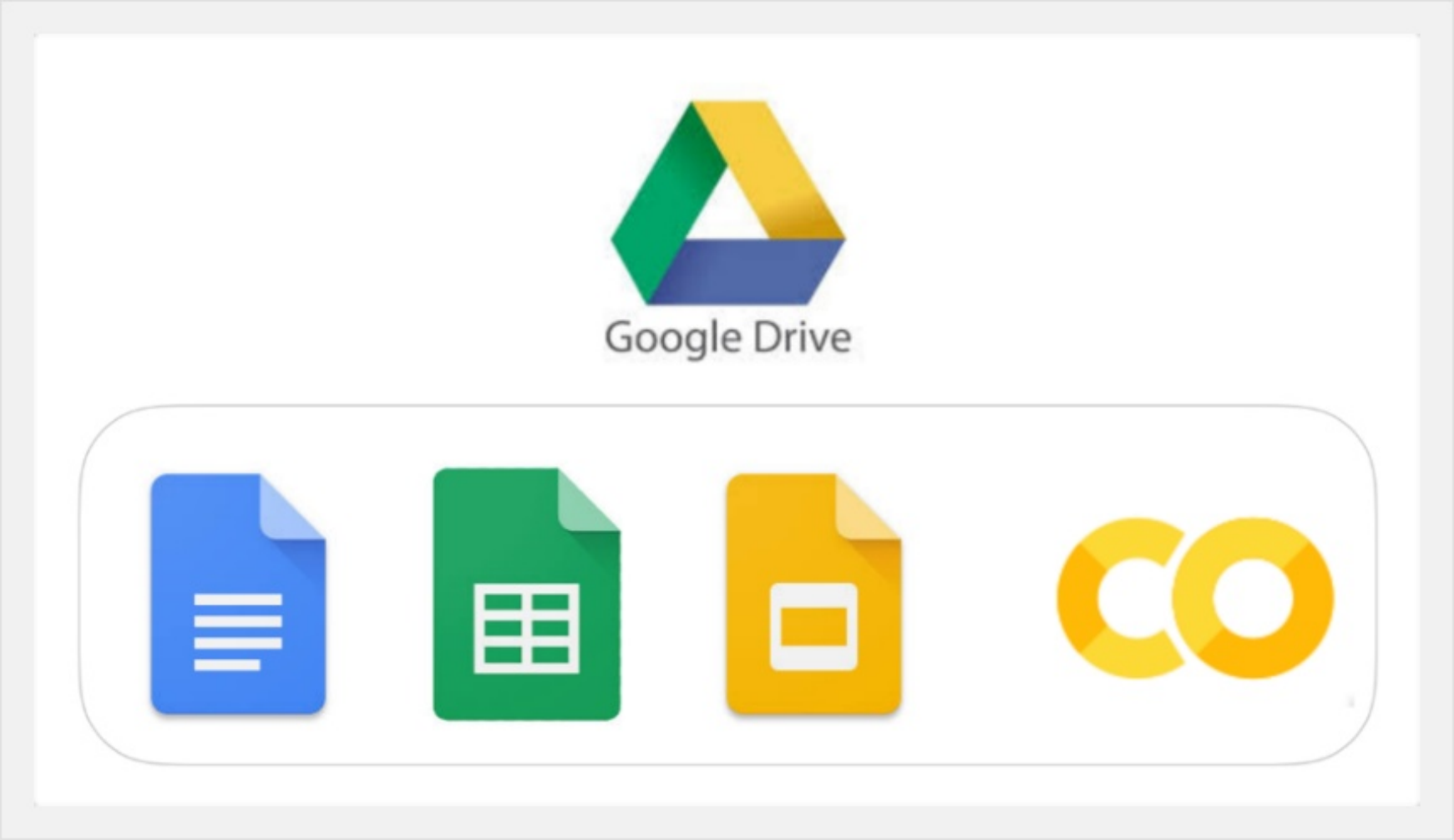
- param - nome da variável local que receberá o valor;
- param = valor - assume valor default caso não informado;
- *params - lista dos parâmetros adicionais;
- **dparams - dicionário com parâmetros adicionais passados na forma param = valor;

```
>>> def spam(x, y):
...     return x ** y
...
>>> spam(5)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: spam() takes exactly 2 arguments (1 given)

>>> def spam(x, y=2):
...     return x ** y
...
>>> spam(5)
25
>>> spam(5, 3)
125
```

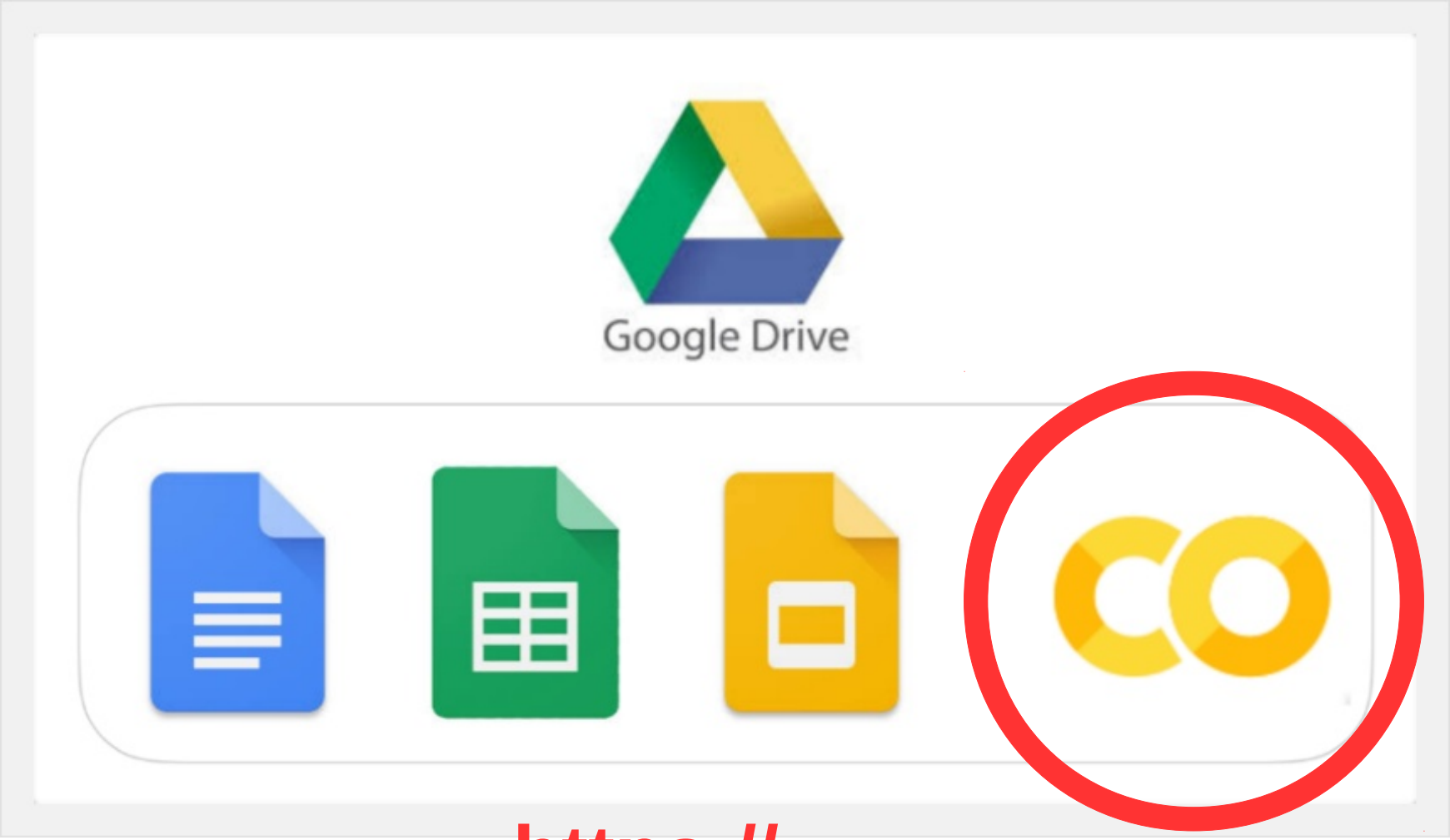
```
>>> def spam(x, y=2, *args, **kw):
...     print x, y, args, kw
...
>>> spam(1)
1 2 () {}
>>> spam(1, 3)
1 3 () {}
>>> spam(1, 2, 3, 4, a1=5, a2=6)
1 2 (3, 4) {'a1': 5, 'a2': 6}

>>> def spam(x, *a, **k, y):
  File "<stdin>", line 1
    def spam(x, *a, **k, y):
                         ^
SyntaxError: invalid syntax
```

```
>>> def spam(x, y=2, z):
...     pass
...
  File "<stdin>", line 1
SyntaxError: non-default argument follows default argument

>>> def spam(x, y=2, *a, **kw):
...     print x, y, a, kw
...

>>> spam(1, z=3)
1 2 () {'z': 3}
>>> spam(1, z=3, 2)
  File "<stdin>", line 1
SyntaxError: no-keyword arg after keyword arg
```

Vamos então conhecer o



Vamos então conhecer o



https://
colab.research.google.com/

E classes?

Você REALMENTE sabe o que é desenvolvimento web?

Definição

- O desenvolvimento web se trata da criação de um software, seja ele um sistema, um app, um portal ou um site, que está conectado a uma rede, podendo ser internet ou intranet (uma rede interna);

Fonte: https://www.intelectua.com.br/blog/o-que-e-desenvolvimento-web

Quais as vantagens?

- Maior mobilidade (facilidade de acesso);
- Menos preocupações e investimentos;
- Integração entre sistemas;
- Não preciso instalar nada no cliente;
- Gerenciamento de atualizações centralizadas;
- Escalabilidade no processamento;
- Dentre outros...

Porém, como funciona uma aplicação web de fato?

- Os responsáveis por fazer com que uma aplicação web funcione são os servidores web, as solicitações realizadas pelo usuário, o protocolo HTTP, os métodos HTTP, outros tipos de protocolo e a resposta do HTTP.

Porém, como funciona uma aplicação web de fato?

- Servidor web;
- Solicitações dos usuários;
- Protocolo HTTP;
- Métodos HTTP – POST e GET;
- Outros tipos de protocolo – FTP, SMTP, POP e IMAP, etc;
- Resposta do HTTP – Erro 404.

A atuação profissional

- Desenvolvedor front-end:
  - interfaces de um projeto web;
- Desenvolvedor back-end:
  - RN, segurança, banco de dados e integração dos serviços web;
- Desenvolvedor full-stack:
  - tanto do front-end quanto do back-end.

Frameworks web - Python

- Django;
- Flask;
- Tornado;
- Dentre outras...

Vamos trabalhar com...

Quem usa Django?

Sobre o Django

- Django é um framework web de alto nível escrito em Python que estimula o desenvolvimento rápido e limpo;
- Construído por desenvolvedores experientes, ele cuida de grande parte incômoda do desenvolvimento web;
- É gratuito e de código aberto.

Fonte: https://www.djangoproject.com/

Por que Django?

- O Django é atualmente o principal framework para desenvolvimento de aplicações web em Python.
- Baseado no padrão MTV (Model-Template-View), uma variação do MVC, esse framework oferece recursos para a implementação das necessidades mais comuns nesse contexto.

MVT – Model, View e Template

- Como é MVC mesmo?
  - **View** – Fala Controller ! O usuário acabou de pedir para acessar o Facebook ! Pega os dados de login dele ai.
  - **Controller** – Blz. Já te mando a resposta. Ai model, meu parceiro, toma esses dados de login e verifica se ele loga.
  - **Model** – Os dados são válidos. Mandando a resposta de login.
  - **Controller** – Blz. View, o usuário informou os dados corretos. Vou te mandar os dados dele e você carrega a página de perfil.
  - **View** – Vlw. Mostrando ao usuário…

Fonte: https://github.com/fga-gpp-mds/A-Disciplina/wiki/Padr%C3%B5es-Arquiteturais---MVC-X-Arquitetura-do-Django

MVT – Model, View e Template

- Model:
  - As Models do MVC e do MVT são equivalentes em responsabilidades;
  - Cada classe da modelo se compara a uma tabela do BD, e as instâncias destas classes, representam os registros destas tabelas.

Fonte: https://github.com/fga-gpp-mds/A-Disciplina/wiki/Padr%C3%B5es-Arquiteturais---MVC-X-Arquitetura-do-Django

MVT – Model, View e Template

- View:
  - Responsável pela implementação das regras de apresentação e negócio do nosso sistema;
  - É nela onde iremos nos comunicar com a Model e a Template, cadastrando e tratando as informações recebidas. Retornando para o usuário uma resposta, como HTMLs, XML, ou erros encontrados.

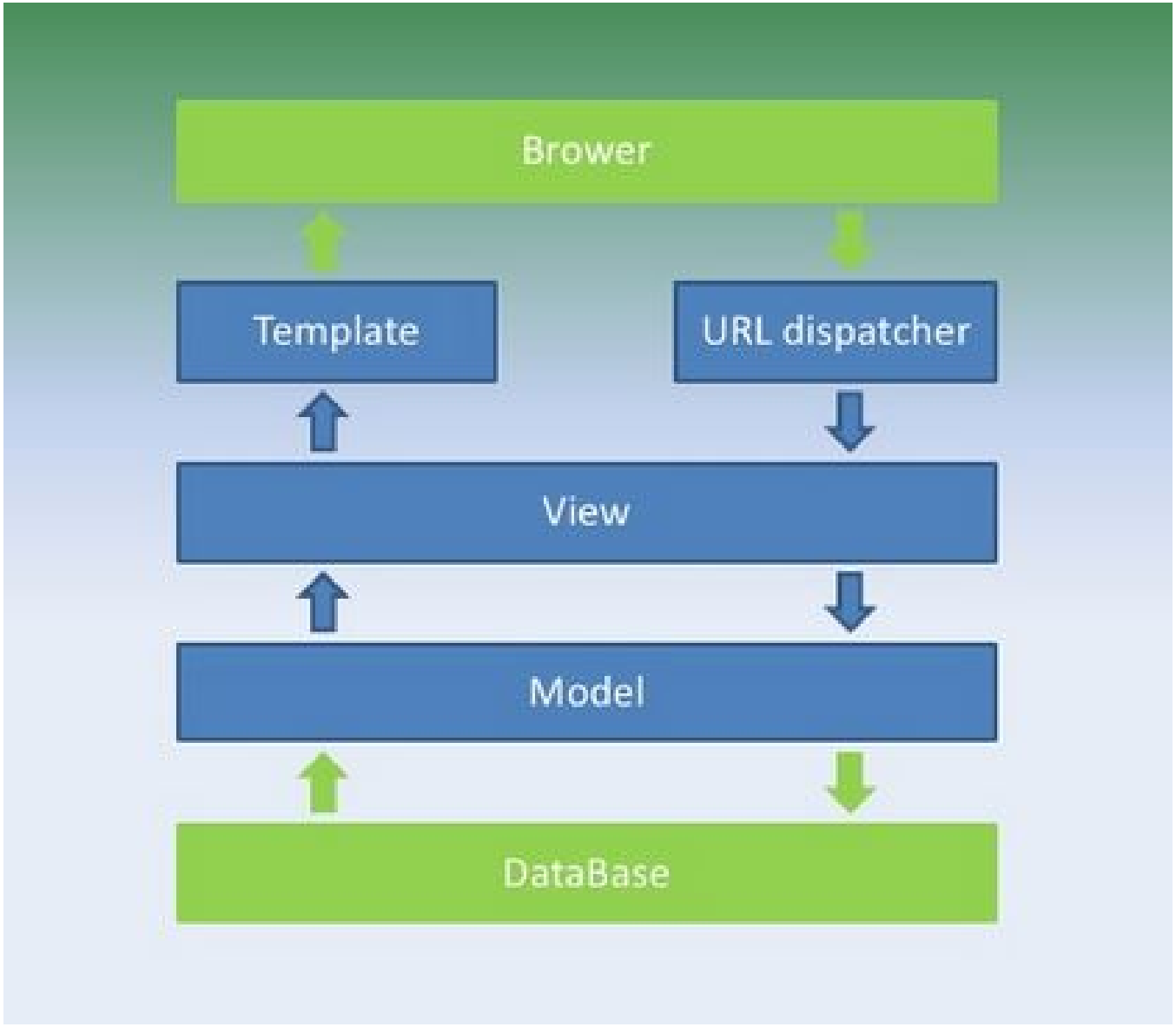
Fonte: https://github.com/fga-gpp-mds/A-Disciplina/wiki/Padr%C3%B5es-Arquiteturais---MVC-X-Arquitetura-do-Django

MVT – Model, View e Template

- Template:
  - É a camada que retorna a visão para o usuário do programa.
  - Essa camada é composta por HTML, CSS, Javascript e etc. Geralmente linguagens focadas na apresentação do site para o usuário.

Fonte: https://github.com/fga-gpp-mds/A-Disciplina/wiki/Padr%C3%B5es-Arquiteturais---MVC-X-Arquitetura-do-Django

Como ele funciona?



Como ele funciona?

URL dispatcher

- Chega requisição para o servidor web, Django que tenta descobrir do que ela se trata.
- Essa parte é feita pelo ***urlresolver*** do Django:
  - Note que o endereço de um site se chama URL - Uniform Resource Locator, em português Localizador de Recursos Uniforme, dessa forma o nome ***urlresolver***, ou resolvedor de urls, faz sentido;
- Isso não é muito complexo - ele pega uma lista de padrões e tenta corresponder com a URL;
- Identificado o que é pra fazer, passa a solicitação para a chamada view.

Chega de falar e vumbora fazer!

Entendo a estrutura criada

O arquivo **mysite/** exterior é apenas um contêiner para o seu projeto. Seu nome não importa para Django;

manage.py: Um utilitário de linha de comando que permite a você interagir com esse projeto Django de várias maneiras;

O arquivo **mysite/** interior é o pacote Python para o seu projeto. Seu nome é o nome do pacote Python que você vai precisar usar para importar coisas do seu interior;

mysite/__init__.py: Um arquivo vazio que diz ao Python que este diretório deve ser considerado um pacote Python;

mysite/settings.py: Configurações para este projeto Django;

mysite/urls.py: As declarações de URLs para este projeto Django; um “índice” de seu site movido a Django;

mysite/wsgi.py: Um ponto de integração para servidores WEB compatíveis com WSGI usado para servir seu projeto.

- Qual é a diferença entre um projeto e um app?
  - Um app é uma aplicação que faz alguma coisa;
  - Um projeto é uma coleção de configurações e apps.
    - Um projeto pode conter múltiplos apps;
    - Um app pode estar em múltiplos projetos;