



Mini Curso Prático a Linguagem Python

Organização Mini Curso

- **Carga horária:** 5 horas.
- **Resumo da ementa:** *Este mini curso é uma introdução prática a linguagem de programação Python.*



Leandro de Carvalho Maia



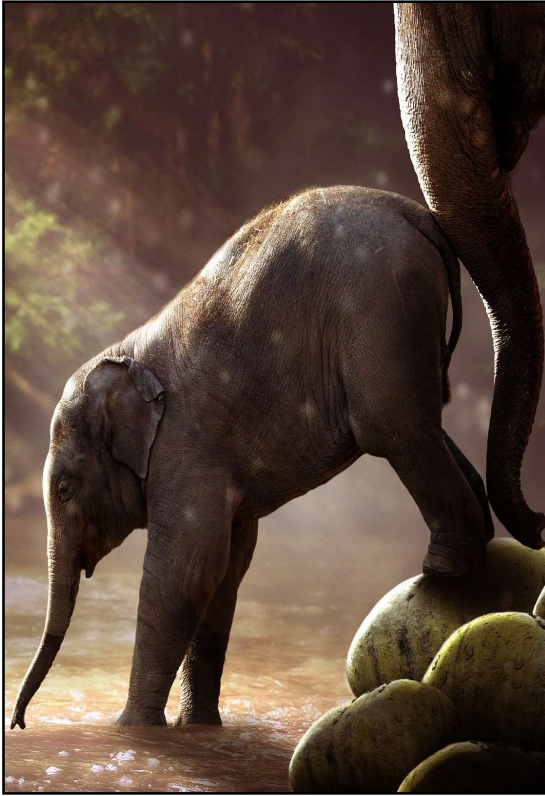
- Bacharelado em Sistemas de Informação - UNIVÁS.
- Licenciatura em Sistemas de Informação – IF SUL de Minas.
- Pós-graduação em Engenharia Biomédica e Engenharia Clínica - Inatel.
- Especialista em Sistemas – Inatel Competence Center - ICC.
- Contato:
 - leandroc@inatel.br
 - <https://about.me/leandrocarvalhomaia>



Conteúdo

- Apresentando o Python.
- Ferramentas de Desenvolvimento.
- Conceitos básicos da linguagem:
 - Modules.
 - Blocos por Indentação.
 - Tipos.
 - Comandos de controle.
 - Estruturas de dados.
 - Funções.





Metodologia

- Teoria básica.
- Apresentação das ferramentas.
- Exercícios práticos.
- Material de apoio.

Material de Apoio

Livros, Apostilas, Fóruns, Vídeos, etc.



Referências

Livro Pense Python: <https://penseallen.github.io/PensePython2e/>

Exercícios Livro Pense em Python: <https://pense-python.caravela.club/01-a-jornada-do-programa/09-exercicios.html>

Apresentando Python



Apresentando Python

O que é Python?

- Python é uma linguagem moderna, de alto nível e de propósitos gerais.
- Sua filosofia enfatiza a legibilidade do código.
- Sua sintaxe permite expressar conceitos com poucas linhas de código.
- Linguagem multiplataforma (*um mesmo programa roda em diferentes tipos de sistemas operacionais e dispositivos*).
- É um software livre (*não é necessário pagar para usá-lo ou distribuí-lo*).
- É muito organizada! Ela força o programador a deixar o código organizado.
- Orientada a objetos e procedural.



O que é Python

Python é uma linguagem de altíssimo nível (em inglês, *Very High Level Language*) orientada a objetos, de tipagem dinâmica e forte, interpretada e interativa.

O Python possui uma sintaxe clara e concisa, que favorece a legibilidade do código fonte, tornando a linguagem mais produtiva.

A linguagem inclui diversas estruturas de alto nível (listas, tuplas, dicionários, data / hora, complexos e outras) e uma vasta coleção de módulos prontos para uso, além de frameworks de terceiros que podem ser adicionados. Também possui recursos encontrados em outras linguagens modernas, tais como: geradores, introspecção, persistência, metaclasses e unidades de teste. Multiparadigma, a linguagem suporta programação modular e funcional, além da orientação a objetos. Mesmo os tipos básicos no Python são objetos.

A linguagem é interpretada através de bytecode pela máquina virtual Python, tornando o código portátil. Com isso é possível compilar aplicações em uma plataforma e rodar em outras ou executar direto do código fonte.

Python é um software de código aberto (com licença compatível com a General Public License (GPL), porém menos restritiva, permitindo que o Python seja incorporados em produtos proprietários) e a especificação da linguagem é mantida pela Python Software Foundation².

Python é muito utilizado como linguagem script em vários softwares, permitindo automatizar tarefas e adicionar novas funcionalidades, entre eles: BrOffice.org, PostgreSQL, Blender e GIMP. Também é possível integrar o Python a outras linguagens, como a Linguagem C. Em termos gerais, o Python apresenta muitas similaridades com outras linguagens dinâmicas, como Perl e Ruby.

Apresentando Python

Principais Características

- Linguagem de programação orientada a objetos de alto nível.
- Grau de abstração elevado.
- Sintaxe muito simples e intuitiva.
- Próximo à linguagem humana.
- Criada para ser tão legível quanto o inglês.
- Ideal para quem está começando a programar.
- Código aberto.



Apresentando algumas características da Linguagem Python:

- Python é uma linguagem de programação que não é necessário compilar seu código para que a máquina entenda.
- Roda em ambientes Linux, Windows, MacOS, *smartphones*, celulares, e outra infinidade de sistemas.
- Por padrão ela é uma linguagem totalmente orientada a objetos, ela permite que o programador desenvolva de forma procedural ou funcional.
- É Software Livre! Criada para ser gratuita, e sempre será!
- Possui código aberto, então você não precisa se preocupar quanto a isso, ou sobre a “estabilidade” da linguagem no mercado, já que possui uma imensa comunidade ao redor do globo.
- O Python é uma linguagem multiuso, pois permite criar desde aplicativos desktop a websites.

Apresentando Python

História

- Criada pelo holandês **Guido Van Rossum**.
- A ideia era criar uma linguagem **tão poderosa** quanto **C**, mas de mais **fácil compreensão** e mais fácil de **programar**.
- Inspirada na linguagem **ABC**.
- Em **1991** é liberada a versão 0.9.0.



História da Linguagem Python

A Linguagem Python foi concebida no fim dos anos 80. A primeira ideia de implementar o Python surgiu mais especificamente em 1982 enquanto Guido Van Rossum trabalhava no CWI (Centrum Wiskunde & Informatica, Centro de Matemática e Ciência da Computação) em Amsterdã, Holanda, no time de desenvolvimento da Linguagem **ABC**. Neste mesmo local também foi desenvolvida a linguagem Algol 68.

Posteriormente, em 1987, com o fim da linguagem ABC, Guido foi transferido para o grupo de trabalho Amoeba — um sistema operacional Microkernel liderado por Andrew Tanenbaum. Foi neste grupo que Guido percebeu a necessidade de uma linguagem para escrever programas intermediários, algo entre o C e o Shell Script.

Percebi que o desenvolvimento de utilitários para administração de sistema em C (do Amoeba) estava tomando muito tempo. Além disso, fazê-los em shell Bourne não funcionaria por diversas razões. O motivo mais importante foi que, sendo um sistema distribuído de microkernel com um design novo e radical, as operações primitivas do Amoeba eram diferiam muito (além de serem mais refinadas) das operações primitivas disponíveis no shell Bourne. Portanto, havia necessidade de uma linguagem que "preencheria o vazio entre C e o shell". Por um tempo longo, esse foi o principal objetivo do Python. - Guido Van Rossum

No ano de 1991 Guido foi transferido do grupo Amoeba para o grupo Multimídia. De acordo com o próprio Guido "ABC me deu a inspiração crucial para Python, o grupo Amoeba a motivação imediata e o grupo de multimídia fomentou seu crescimento". Ainda neste ano, no dia 20 de Fevereiro, foi lançada a primeira versão do Python, então denominada de **v0.9.0**. O anúncio foi feito no grupo de discussão (newsgroup) alt.sources.

Nesta primeira versão, o Python já contava com classes, herança, tratamento de exceções, funções, sistema de módulos (empresado da linguagem Modula-3) e os tipos de dado nativos **list**, **dict**, **str**, e etc.

Apresentando Python

Confira o Zen of Python

- O modo *pythonic*....



```
IDLE Shell 3.10.7
File Edit Shell Debug Options Window Help
Python 3.10.7 (tags/v3.10.7:6cc6b13, Sep  5 2022, 14:08:36) [MSC v.19
33 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more informati
on.
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
>>>
```

The Zen of Python

As metas do projeto foram resumidas por Tim Peters (um dos programadores mais respeitados do mundo) em um texto chamado *Zen of Python*. Para visualizar as 19 filosofias digite no próprio terminal do Python o comando:

```
>> import this
The Zen of Python, by Tim Peters.
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Apresentando Python

História - A linguagem hoje

- Atualmente possui um **modelo de desenvolvimento comunitário**, aberto e gerenciado pela organização sem fins lucrativos **Python Software Foundation**.
- O padrão de implementação é o **CPython**.
- Apesar de sua notoriedade no desenvolvimento web, Python é utilizada em diversas outras áreas:
 - Aplicações desktop;
 - Computação científica;
 - Computação gráfica;
 - Desenvolvimento de games;
 - **Machine Learning e Data Science**;



Apresentando Python

Versões

- Para o desenvolvimento dos códigos em Python requer-se um interpretador Python.
- Possui versões:
 - 2.7.x - recebeu atualizações de segurança até 2020, encontra-se descontinuada;
 - **3.10.0** – em constante evolução;
- A versão 3 veio para **corrigir erros** da **versão 2**, por isso, para quem está começando a versão 3 é a versão recomendada.
- As versões a partir do Python 3.9+ não podem ser utilizadas no Windows 7 ou anteriores..



Apresentando Python

Quem Usa Python

Google

Instagram

Dropbox

YAHOO!



WOP
WASHINGTON
POST

The
New York
Times

Microsoft

INSTITUTO FEDERAL
Rio Grande do Norte

IBM



Instagram

Pinterest

NOKIA

globo
.com

CI&T

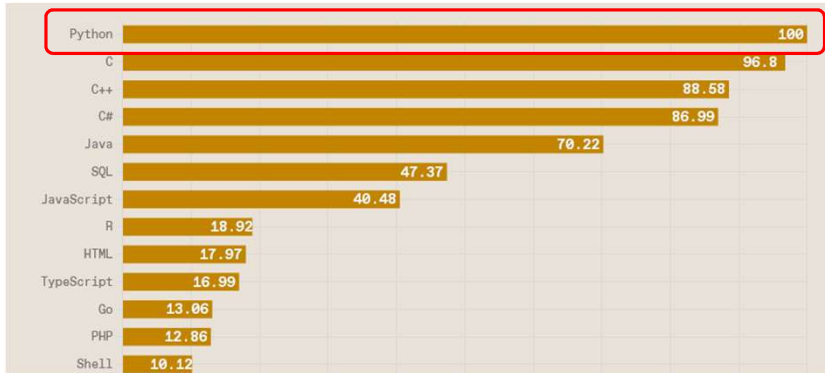
Bitbucket

BRASIL
UM PAÍS DE TODOS
GOVERNO FEDERAL

Apresentando Python

História - Atualidade

- The 2022 Top Ten Programming Languages:



Fonte: IEEE Spectrum
04/10/2022

História – Atualidade











Link de referência:

- <https://spectrum.ieee.org/top-programming-languages-2022#toggle-gdpr>

Apresentando Python

História - Atualidade

- TIOBE Index para Outubro de 2022:

Sep 2022	Sep 2021	Change	Programming Language
1	2	▲	 Python
2	1	▼	 C
3	3		 Java
4	4		 C++
5	5		 C#
6	6		 Visual Basic
7	7		 JavaScript
8	8		 Assembly language
9	10	▲	 SQL
10	9	▼	 PHP



Fonte: TIOBE
04/10/2022

História – Atualidade

Link de referência:

- <https://www.tiobe.com/tiobe-index/>

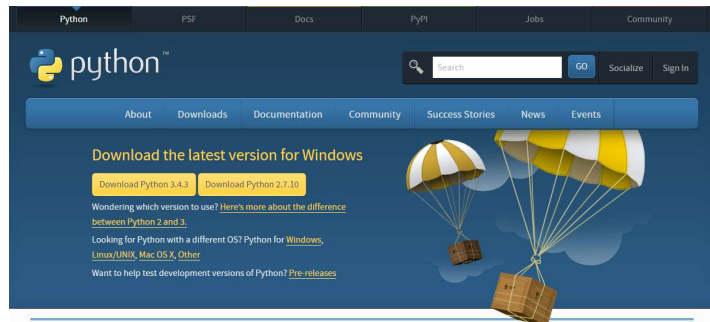
Ferramentas de Desenvolvimento



Ferramentas de Desenvolvimento

Interpretador Python

- Para sistemas operacionais Windows é necessário baixar e instalar o interpretador Python.
- Ao contrário da maioria das distribuições Linux, o Windows não vem com o Python instalado.
- Disponível em: <https://www.python.org/getit/>



Instalando o interpretador Python

O interpretador da linguagem para Windows pode ser encontrado no site do Python (<http://www.python.org/getit/>). Certifique-se de baixar a versão correta para o seu sistema operacional. Você deverá baixar a versão mais nova, a qual era a **3.7.3** quando esse tutorial foi escrito.

Observação: A maioria das versões do Linux e do OS X ainda usam a versão 2 do Python. Existem algumas pequenas diferenças entre as versões 2 e 3, a mais notável é a mudança no comando "print". Se você deseja instalar uma versão mais nova do Python no Linux ou no OS X, baixe-a no site da linguagem.

Após o download, execute a instalação NNF (Next > Next > Finish) mantenha as configurações default. Com isso, o Python foi instalado no diretório **C:\Python27**.

Configuração da variável de ambiente

Antes de dar o Python como instalado você deve ir nas **Variáveis de Ambiente**, sendo acessadas através do atalho: Tecla Super (tecla com a logo do Windows) + Pause Break, vá em Configurações Avançadas e depois em Variáveis de Ambiente.

Nas Variáveis do Sistema, encontre e edite a variável Path adicionando no final do campo Valor da variável o caminho de onde o Python foi instalado, usando o caractere ; antes de iniciar o caminho real do Python (adicionando o seguinte valor, ex: C:\Users\leandro\AppData\Local\Programs\Python\Python37-32\).

Para saber se todos esses passos realmente funcionaram, abra o console do Windows e digite: **python -V**

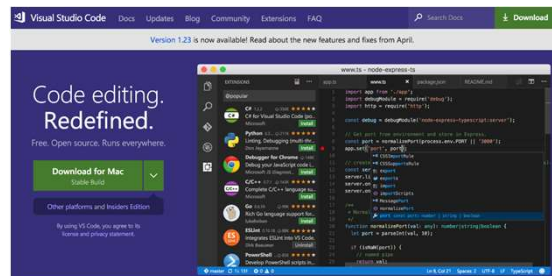
O resultado esperado deve ser:

Python 3.7.3.

Ferramentas de Desenvolvimento

Editor de Textos

- Como editor de texto utilizaremos o **Visual Studio Code**.
- Desenvolvido pela **Microsoft**.
- Projeto **Free** e **Open Source** para todas as plataformas.
- Disponibilizado de forma **gratuita**, através do link:
<https://code.visualstudio.com/>
- Para instalar basta acessar o link acima e selecionar a opção de download.



Ferramentas de Desenvolvimento

Interpretador Interativo do Python

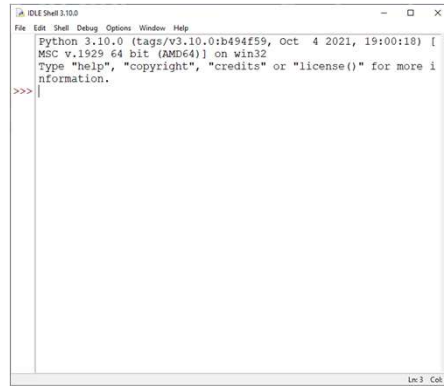
- Python 3 disponibiliza um interpretador interativo.
- Um terminal onde você pode fazer testes dinamicamente.
- Você implementa e executa seu código Python.
- Conhecido como **IDLE** - *Integrated Development Environment*.



Ferramentas de Desenvolvimento

Executando o IDLE

- Acesse o executável:
 - Iniciar -> Python 3.10 -> IDLE (Python GUI)
- Escreva seu código e aperte *Enter*:



Interpretador Interativo

O instalador do Python 3 vem com um aplicativo embutido chamado **IDLE**, que faz ambos os trabalhos (editor + interpretador). Permite escrever e editar o código Python. Ele traduz o código em forma binária e, finalmente, executa o programa Python 3.

Por conta disso é conhecido por IDLE - ***Integrated Development Environment***, ou seja, um **ambiente de desenvolvimento integrado**.

Com ele é disponibilizado um conjunto de ferramentas como:

- Editor de texto com *color code*: exibe os códigos digitados com as cores de acordo com a sintaxe Python;
- *Python Shell*: terminal para execução de códigos Python.
- *Debugger*: permite depurar o código fonte através de break points.

Para ter acesso ao IDLE acesse:

- Iniciar -> Python 3.9 > IDLE (Python GUI)

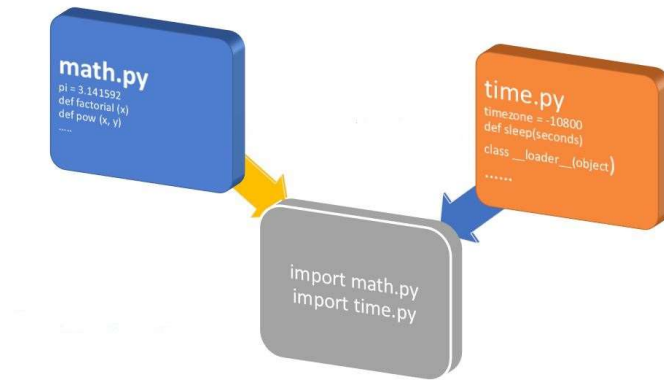
Também é possível utilizar interpretadores Python online, ou seja, utilizando-se navegadores web, independentes de sistema operacional. Confira este exemplo:

- <https://repl.it/languages/python3>

Conhecendo a Linguagem Python



Modules



Conhecendo a Linguagem Python

Modules

- Cada **arquivo** em Python é chamado de **module (ou módulo)**.
- O nome do arquivo é o módulo com o sufixo **.py** adicionado.
- Módulos são um **conjunto** de códigos como: **funções, classes, variáveis**.
- Módulos se comunicam através do comando **import**.
- É boa prática: todos os **import** no **início** do módulo.

```
1 import pytest
2 import smtplib
3 from datetime import date
4 from datetime import timedelta
5
```

Módulos

Se você sair do interpretador do Python e entrar novamente, as definições (funções e variáveis) que você havia feito estarão perdidas. Portanto, se você quer escrever um programa um pouco mais longo, você se sairá melhor usando um editor de texto para criar e salvar o programa em um arquivo, usando depois esse arquivo como entrada para a execução do interpretador. Isso é conhecido como gerar um script. A medida que seus programas crescem, pode ser desejável dividi-los em vários arquivos para facilitar a manutenção. Você também pode querer reutilizar uma função sem copiar sua definição a cada novo programa.

Para permitir isso, Python tem uma maneira de colocar definições em um arquivo e então usá-las em um script ou em uma execução interativa no interpretador. Tal arquivo é chamado de “módulo”; definições de um módulo podem ser importadas em outros módulos ou no módulo principal (a coleção de variáveis a que você tem acesso no nível mais externo de um script executado como um programa, ou no modo calculadora).

Um módulo é um arquivo Python contendo definições e instruções. O nome do arquivo é o módulo com o sufixo **.py** adicionado. Dentro de um módulo, o nome do módulo (como uma string) está disponível na variável global **__name__**.

Conhecendo a Linguagem Python

Modules

- Os programas em Python inicialmente **operam** com um **conjunto básico** de recursos, porém, caso haja necessidade, podemos **IMPORTAR** nova bibliotecas ou módulo a fim de **adicionarmos** mais **recursos**.
- Existem duas formas de importar bibliotecas externas no Python, são elas:
- *import nova_biblioteca* -> importa uma biblioteca inteira de recursos;
- *from nova_biblioteca import parte_da_nova_biblioteca* -> importa apenas um recurso específico de uma biblioteca.

Módulos

Se você sair do interpretador do Python e entrar novamente, as definições (funções e variáveis) que você havia feito estarão perdidas. Portanto, se você quer escrever um programa um pouco mais longo, você se sairá melhor usando um editor de texto para criar e salvar o programa em um arquivo, usando depois esse arquivo como entrada para a execução do interpretador. Isso é conhecido como gerar um script. A medida que seus programas crescem, pode ser desejável dividi-los em vários arquivos para facilitar a manutenção. Você também pode querer reutilizar uma função sem copiar sua definição a cada novo programa.

Para permitir isso, Python tem uma maneira de colocar definições em um arquivo e então usá-las em um script ou em uma execução interativa no interpretador. Tal arquivo é chamado de “módulo”; definições de um módulo podem ser importadas em outros módulos ou no módulo principal (a coleção de variáveis a que você tem acesso no nível mais externo de um script executado como um programa, ou no modo calculadora).

Um módulo é um arquivo Python contendo definições e instruções. O nome do arquivo é o módulo com o sufixo .py adicionado. Dentro de um módulo, o nome do módulo (como uma string) está disponível na variável global `__name__`.

Conhecendo a Linguagem Python

Modules

- Confira um exemplo prático com as classes do módulo *datetime*:

```
1 from datetime import date
2
3 print(date.today())
4
```

Importando apenas a classe ***date***

```
1 import datetime
2
3 print(datetime.date.today())
4
5 print(datetime.datetime.now())
6
```

Importando todo o módulo ***datetime***

Módulos

Os módulos disponíveis estão armazenados na variável ***sys.modules***, para ver quais são é preciso importar o módulo *sys*:

Conhecendo a Linguagem Python

Modules – Na prática

- Aproveitando o exemplo *hello_world.py* crie no mesmo diretório outro módulo:
 - *modulos.py*
- No novo módulo importe o módulo *hello_world.py*:
 - *import hello_world*
- Execute:
 - *python modulos.py*
- Adicione uma nova linha no *hello_world.py*:
 - *curso = “Introdução ao Python”*
- Modifique o módulo *modulos.py* e execute novamente:
 - *from hello_world import curso*
 - *print(curso)*

Módulos

Os módulos disponíveis estão armazenados na variável ***sys.modules***, para ver quais são é preciso importar o módulo *sys*:

```
>>> import sys
>>> sys.modules
```

Conhecendo a Linguagem Python

Executando Programas

- Tornar um programa Python executável é igualmente simples.
- Python usa a sintaxe:

```
1 if __name__ == '__main__':  
2     #seu código
```

- Assim, determina se o arquivo está sendo executado na **linha de comando** ou sendo **importado** por outro código.

Conhecendo a Linguagem Python

Executando Programas

- Desta forma, para tornar um arquivo executável, inclua a detecção do valor `"__main__"` na variável `__name__`.

Dunder Alias: em Python é comum o uso de double underscore na nomeação de variáveis e métodos especiais. Para facilitar a pronúncia pode-se utilizar o termo *dunder*. Assim fica: *dunder main dunder* ou apenas *dunder main*.



Conhecendo a Linguagem Python

Utilizando as Bibliotecas Padrão

- Organizando-se o código por módulos, tem-se acesso a uma grande biblioteca de códigos prontos.
- Parte desta biblioteca é disponibilizada durante a instalação do Python.
- Uma parte maior ainda são oferecidos por terceiros.
- A maioria dos seus problemas podem ser resolvidos através a biblioteca padrão.
- Você pode encontrar informações detalhadas na documentação oficial.
 - <https://docs.python.org/3/library/index.html>



Biblioteca Padrão

The Python Standard Library: <https://docs.python.org/3/library/index.html>

Conhecendo a Linguagem Python

Utilizando as Bibliotecas Padrão

- Exemplo de uso do módulo math, o qual dá acesso as funções da biblioteca C para matemática de ponto flutuante:

```
>>> import math
>>> #Exponenciação
>>> math.pow(2, 3)
8.0
>>> #Logaritmo
>>> math.log(1024, 2)
10.0
>>> #Raiz quadrada
>>> math.sqrt(16)
4.0
>>> #Constante PI
>>> math.pi
3.141592653589793
>>> #Coseno
>>> math.cos(math.pi / 4.0)
0.7071067811865476
```

Conhecendo a Linguagem Python

Biblioteca Padrão

- Matemática:
 - O módulo **random** fornece ferramentas para gerar seleções aleatórias:

```
>>> import random
>>> random.choice(['maçã', 'pera', 'banana'])
'maçã'
>>> random.sample(range(100), 10)
[15, 68, 61, 95, 7, 39, 73, 43, 38, 49]
>>> random.random()
0.5557472171866676
```

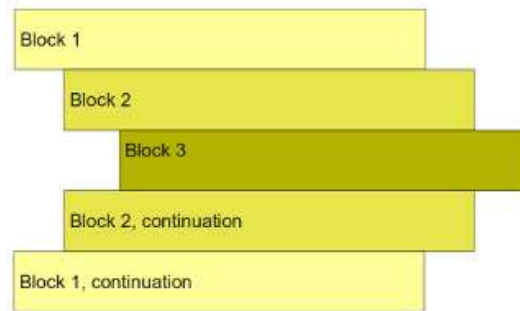

Conhecendo a Linguagem Python

Biblioteca Padrão

- Data e Hora:
 - O módulo ***datetime*** fornece classes para manipulação de datas e horas nas mais variadas formas.
 - O foco da implementação é na extração eficiente dos membros para formatação e manipulação:

```
>>> from datetime import date
>>> now = date.today()
>>> now
datetime.date(2016, 1, 29)
>>> now.strftime('%m-%d-%y')
'01-29-16'
>>> now.strftime('%d %b %Y is a %A on the %d day of %B.')
'29 Jan 2016 is a Friday on the 29 day of January.'
>>>
>>> #Operações aritméticas
>>> birthday = date(1982, 7, 2)
>>> age = now - birthday
>>> age.days
12264
```

Blocos por Indentação



Conhecendo a Linguagem Python

- Em Python, os blocos de código são delimitados pelo uso de indentação.
- Em outras linguagens utiliza-se marcadores para delimitar trechos de códigos, por exemplo:

```
1  if (true){  
2      /* bloco de código */  
3  }
```

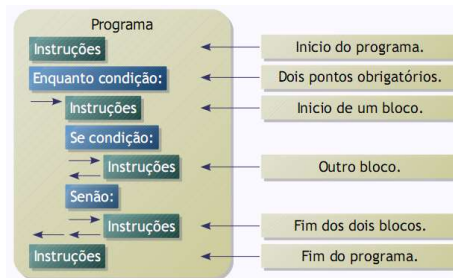
- Em Python basta você usar a indentação para demarcar os blocos:

```
1  if True:  
2      #bloco de código  
3      #(ao fim da seção indentada  
4      #finaliza a bloco)  
5  próxima instrução após o if
```

Conhecendo a Linguagem Python

Bloco por Indentação

- **Importante:** deve-se convencionar se a indentação será feita por uma tabulação ou por um número determinado de espaços.
- Python 3 não permite misturar o uso de tabulação e espaço.
- Todos que editarem um programa Python devem usar a mesma convenção.
- *PEP8* recomenda sempre o uso de 4 espaços consecutivos para indicar a indentação.



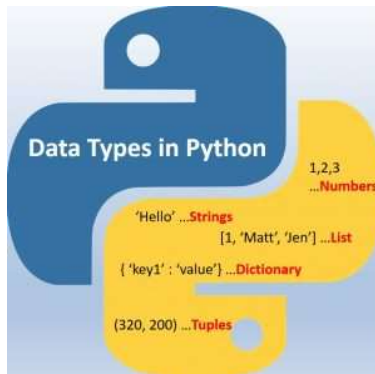
Conhecendo a Linguagem Python

Bloco por Indentação

- Confira na prática....

```
1  a = 54
2  b = 24
3  c = 21
4
5  if a > b:
6      a = b
7      if a > c:
8          a = c
9  elif a < b:
10     a = c
11  print(a)
```

Tipos



Conhecendo a Linguagem Python

- **O que é tipagem fraca?**
 - Conversão automática entre tipos.
 - Comum em linguagens de *scripting* (*JavaScript*, *Perl*, *PHP*).
 - Uma fonte de bugs difíceis de localizar e tratar.
- Exemplo executado no console do Chrome:

```
> "9" + 10
< "910"
> "9" * 10
< 90
> "9" - 10
< -1
> "9" + (-10)
< "9-10"
>
```

Conhecendo a Linguagem Python

Tipos

- **ATENÇÃO: Python não é assim!**
- Em Python a **tipagem é dinâmica**.
- Variáveis (e parâmetros) **não têm** tipos **declarados**.
- Podem ser associados a objetos de qualquer tipo em **tempo de execução**.

```
>>> def sum(value):  
    '''Sum the parameter value'''  
    return value + value  
  
>>> sum(7)  
14  
>>> sum('Python')  
'PythonPython'  
>>> sum([10, True, 34.5])  
[10, True, 34.5, 10, True, 34.5]  
>>>
```


Conhecendo a Linguagem Python

Tipos

- **Tipagem Dinâmica Forte.**
- Python não faz conversão automática de tipos.
- **Exceções**, por praticidade:
 - int → long → float;
 - str → unicode;

```
>>> "9" + 10
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    "9" + 10
TypeError: Can't convert 'int' object to str implicitly
>>> "9" * 10
'9999999999'
>>> "9" + (-10)
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    "9" + (-10)
TypeError: Can't convert 'int' object to str implicitly
>>> |
```

Conhecendo a Linguagem Python

Tipos - Inicialização

- Python incorpora o conceito de “**duck typing**”.
“Se anda como um pato e faz barulho como um pato, então deve ser um pato.”
- Para iniciar uma variável, basta atribuí-la **algum valor** que o interpretador irá associar a ela o **tipo** que se ajustará ao **valor** passado.

```
>>> a = 5
>>> b = 'loop'
>>> type(a)
<class 'int'>
>>> type(b)
<class 'str'>
```

Tipos – Inicialização

Em Python tudo é objeto. Isso quer dizer que um objeto do tipo *string*, por exemplo, tem seus próprios métodos.

O conceito de variável é uma associação entre um nome e um valor, mas não é necessário declarar o tipo da variável, portanto, o tipo relacionado a variável pode variar durante a execução do programa isto implica em muitos aspectos no uso da linguagem.

Este conceito é chamado em programação de “**duck typing**” (tipagem do pato) - baseado na expressão, em inglês, devida a *James Whitcomb Riley*:

Quando eu vejo uma ave que caminha como um pato, nada como um pato e grasna como um pato, eu chamo esta ave de "pato".

Conhecendo a Linguagem Python

Tipos - Variáveis

- Variáveis são criadas através da atribuição.
- Quando não existem mais referências a elas são destruídas pelo coletor de lixo (*garbage collection*).
- Para os nomes deve-se:
 - começar com letra ou sublinhado (_).
 - seguido por letras, dígitos ou sublinhados (_).
 - não conter acentuação.
 - **Case-sensitive**: maiúsculas e minúsculas são consideradas diferentes.

Conhecendo a Linguagem Python

Nomeação de Variáveis

- Para os nomes deve-se:
 - começar com letra ou sublinhado (_).
 - seguido por letras, dígitos ou sublinhados (_).
 - não conter acentuação.
 - **Case-sensitive**: maiúsculas e minúsculas são consideradas diferentes.
- Orienta-se seguir o padrão PEP8, utilizando-se *Snake Case*:
 - Nomes em minúsculo separados por *underscore*: (_).

Conhecendo a Linguagem Python

Nomeação de Variáveis – *Dunder Alias*

- Em Python é comum o uso de *double underscore* na nomeação de variáveis e métodos especiais.
- Para facilitar a pronúncia pode-se utilizar o termo dunder, ex:

– **`__name__`**

- Sendo assim, se pronuncia:
 - Dunder name.



Conhecendo a Linguagem Python

Tipos - Definição

- Existem vários tipos simples de dados pré-definidos no Python, tais como:
 - Números (inteiros, reais, complexos, ...).
 - Texto (*strings*).
 - Booleanos.
- Além disso, existem tipos que funcionam como coleções. Os principais são:
 - Lista.
 - Tupla.
 - Dicionário.
- Os tipos no Python podem ser:
 - **Mutáveis**: permitem que os conteúdos das variáveis sejam alterados.
 - **Imutáveis**: não permitem que os conteúdos das variáveis sejam alterados.

Conhecendo a Linguagem Python

Tipos - Numéricos

- Python oferece vários tipos numéricos:
 - Inteiro (**int**): $i = 1$
 - Real de ponto flutuante (**float**): $f = 3.14$
 - Complexo (**complex**): $c = 5 + 4j$
- Observações:
 - As conversões entre inteiro e longo são automáticas.
 - Reais podem ser representados em notação científica: $1.2e22$
- O tipo inteiro é de **precisão infinita**, ou seja, senão tomar cuidado pode-se gerar um número que ocupe **toda** a sua **memória**.
- As operações com inteiros retornam inteiros.

Conhecendo a Linguagem Python

Tipos - Numéricos

- Operações numéricas:
 - Soma (+).
 - Diferença (-).
 - Multiplicação (*).
 - Divisão (/).
 - Divisão inteira (//).
 - Exponenciação (**): pode ser usada para calcular a raiz, através de expoentes fracionários (exemplo: $100^{0.5}$).
 - Módulo (%): retorna o resto da divisão.

Conhecendo a Linguagem Python

Tipos - Numéricos

- O Python suporta a atribuição múltipla:

```
>>> a, b, c = 1, 3, 5
>>> a, b, c = a*2, a+b+c, a*b*c
>>> a, b, c
(2, 9, 15)
>>>
```

- Isso é útil na troca de valores entre duas variáveis:

```
>>> a, b = 2, 5
>>> print('a=%d - b=%d' % (a, b))
a=2 - b=5
>>> a, b = 2, 5
>>> print('a=%d b=%d' % (a, b))
a=2 b=5
>>> a, b = b, a #efetuando a troca dos valores entre a e b
>>> print('a=%d b=%d' % (a, b))
a=5 b=2
... |
```

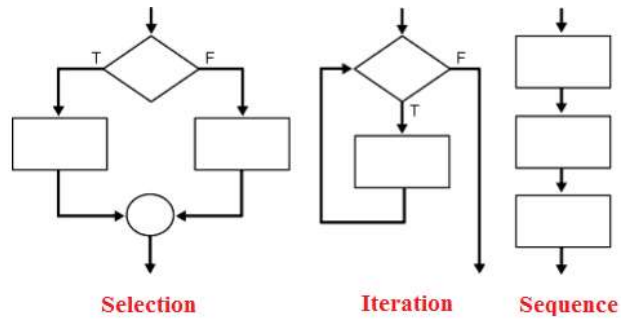
Conhecendo a Linguagem Python

Tipos - Operadores

- São símbolos que atuam sobre variáveis e valores.
- Operadores de atribuição (=).
- Operadores aritméticos (+, -, *, /, %, **, +=, -=, *=, /=, %=, **=).
- Operadores de comparação (>, <, ==, >=, <=, !=, is, in).
- Operadores lógicos (and, or, not).

Control Structures

(Estruturas de Controle)



Conhecendo a Linguagem Python

Comandos de Controle

- A estrutura de controle é um **ponto crucial** e presente em Python.
- Possui certas características um pouco diferentes das linguagens mais comuns.
- Abaixo seguem as palavras reservadas para estrutura de controle:
 - ***if | else | elif***
 - ***for***
 - ***while***
 - ****match-case -> recém chegado na versão 3.10.0***
 - ***try***
 - ***except***
 - ***with***

Conhecendo a Linguagem Python

Comandos de Controle - IF

- A estrutura **if** efetua um desvio condicional de acordo com a expressão especificada.
- Pode conter outras estruturas de **if** aninhados com **elif** e blocos **else**.
- A expressão condicional irá denominar a execução ou não do bloco de código.
- Em Python **não existe** a estrutura **else if**, esta estrutura é **elif**.



Conhecendo a Linguagem Python

Comandos de Controle - IF

- Pode haver zero ou mais seções elif, e a seção else é **opcional**.
- Uma sequência **if ... elif ... elif ...** substitui as construções **switch** ou **case** existentes em outras linguagens.
- Exemplo:

```
>>> x = eval(input('Favor digitar um inteiro: '))
Favor digitar um inteiro: 42
>>> if x < 0:
    x = 0
    print('O número informado não é positivo')
elif x == 0:
    print('O número informado é zero')
elif x == 1:
    print('Número válido com valor 1')
else:
    print('Número válido: %d' % x)

Número válido: 42
>>>
```

Conhecendo a Linguagem Python

Comandos de Controle - FOR

- O comando **for** se diferencia, um tanto, de outras linguagens como C e Java.
- Ao invés de se iterar sobre progressões aritméticas, o comando **for** de Python **itera** sobre os itens de qualquer sequência.
- Podem ser uma **lista** ou uma **string**, na ordem em que eles aparecem na sequência.

```
>>> #Calcular o tamanho de algumas strings
>>> a = ['gato', 'janela', 'contextualizar']
>>> for x in a:
    print(x, len(x))

gato 4
janela 6
contextualizar 14
>>>
```

Conhecendo a Linguagem Python

Comandos de Controle - FOR

- A função **range()**:
 - Se você precisar **iterar** sobre sequências **numéricas**, a função embutida `range()` é a **resposta**.
 - Ela gera listas contendo **progressões** aritméticas.

```
>>> for a in range(5):  
    print(a)  
  
0  
1  
2  
3  
4  
>>>
```


Conhecendo a Linguagem Python

Comandos de Controle - FOR

- Comandos **break** em laços:

- O comando **break**, como em **C**, interrompe o laço **for** mais interno.

```
>>> for x in range(1, 10):  
    if x % 2 == 0:  
        print('Número par %d' % x)  
        break  
  
Número par 2  
>>>
```

Conhecendo a Linguagem Python

Comandos de Controle - FOR

- **Comandos continue em laços:**
 - O comando **continue**, também emprestado de **C**, **avança** para a próxima iteração do laço mais interno.

```
>>> for x in range(1, 10):  
    if x % 2 == 0:  
        continue  
    print('Número ímpar %d' % x)  
  
Número ímpar 1  
Número ímpar 3  
Número ímpar 5  
Número ímpar 7  
Número ímpar 9  
>>>
```

Conhecendo a Linguagem Python

Comandos de Controle - FOR

- Cláusulas else em laços:

- Laços podem ter uma cláusula **else**, que é executada sempre que o laço se encerra por **exaustão** da lista (no caso do for).
- Mas nunca quando o laço é interrompido por um **break**.

```
>>> for n in range(2, 10):
    for x in range(2, n):
        if n % x == 0:
            print('%d = %d * %d' % (n, x, n/x))
            break
        else:
            print('%d é um número primo' % n)

2 é um número primo
3 é um número primo
4 = 2 * 2
5 é um número primo
6 = 2 * 3
7 é um número primo
8 = 2 * 4
9 = 3 * 3
>>>
```

Estrutura for

A estrutura de laço *for* segue a mesma ideia do *for* do *bash*, com a adição da sentença *else*. No laço *for*, a variável do laço alterna seu conteúdo com os valores da lista passada e caso nenhum *break* seja encontrado, até que o último elemento da lista seja processado, os comandos da sentença *else* serão executados. Sua sintaxe segue a forma abaixo:

```
for <variável> in <lista ou tupla de valores>:
    <comandos>
    ...
    break
    ...
    continue
else:
    <comandos>
```

O *break*, *continue* e o *else* são opcionais. O *break* e o *continue* podem aparecer em qualquer nível de indentação, dentro do laço *for*. Ao alcançar um *break*, o laço irá terminar imediatamente, enquanto que um *continue* irá iniciar, imediatamente, a próxima interação do laço *for*.

Abaixo seguem alguns exemplos de laços *for*, executados no interpretador *Python*:

```
>>> semana = ['dom', 'seg', 'ter', 'qua', 'qui', 'sex', 'sab']
>>> for s in semana:
...     print s,
...
dom seg ter qua qui sex sab
>>> for d in range(30):
...     if (d+1) % 7 == 0:
...         print '%4s' % (d+1)
...     else:
...         print '%4s' % (d+1),
...
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30
```

Conhecendo a Linguagem Python

Comandos de Controle - WHILE

- A estrutura de controle **while** será executada enquanto a expressão condicional **não for satisfeita**.
- É empregado quando se deseja realizar um **teste** a cada **interação** do laço.

```
while <condição>:  
    <comandos>  
    ...  
    break  
    ...  
    continue  
else:  
    <comandos>
```

Comando WHILE

O while é um comando que manda um bloco de código ser executado enquanto uma condição **não for satisfeita**. Assim, permite que sejam criados loops de execução, assim como temos em jogos e aplicativos com interfaces gráficas.

O while é um comando muito útil, mas pode ser perigoso, pois se não tratarmos corretamente o critério de parada, o laço pode não ter fim, e o programa não faz o que deveria.

Conhecendo a Linguagem Python

Comandos de Controle - WHILE

- O **while** também suporta as sentenças **break**, **continue** e **else**, como no **for**.

```
>>> a = 10
>>> while a > 0:
    if a % 2 == 0:
        a -= 1
        continue
    print('Número ímpar: %d' % a)
    a -= 1

Número ímpar: 9
Número ímpar: 7
Número ímpar: 5
Número ímpar: 3
Número ímpar: 1
```

Conhecendo a Linguagem Python

Comandos de Controle – MATCH-CASE

- Python 3.10 implementou o *switch-case* – ou algo do tipo!
- Nas outras linguagens como C ou Java, um switch-case verifica o valor de uma variável e caso localize um correspondente executa um novo bloco de código.
- Python implementou o mesmo princípio, porém mais poderoso, com mais recursos.



Conhecendo a Linguagem Python

Comandos de Controle – MATCH-CASE

- Confira um exemplo:

```
for thing in [1,2,3,4]:  
    match thing:  
        case 1:  
            print("thing is 1")  
        case 2:  
            print("thing is 2")  
        case 3:  
            print("thing is 3")  
        case _:  
            print("thing is not 1, 2 or 3")
```

Conhecendo a Linguagem Python

Comandos de Controle – MATCH-CASE

- A estrutura *match-case* é similar ao *switch-case* da linguagem C.
- No entanto, diferente de C, quando um código de um *case* é executado, automaticamente a execução é direcionada para o fim da estrutura *match*.
- Se o *match* não é correspondido, ou seja, o valor da variável não é encontrada o *case* com `_` é executado.
- Veja o resultado do código anterior:

```
thing is 1  
thing is 2  
thing is 3  
thing is not 1, 2 or 3
```


Conhecendo a Linguagem Python

Comandos de Controle – MATCH-CASE

- Analise mais um exemplo, agora listas dentro de uma lista:

```
for thing in [[1,2],[9,10],[1,2,3],[1],[0,0,0,0,0]]:
    match thing:
        case [x]:
            print(f"single value: {x}")
        case [x,y]:
            print(f"two values: {x} and {y}")
        case [x,y,z]:
            print(f"three values: {x}, {y} and {z}")
        case _:
            print("too many values")
```

Conhecendo a Linguagem Python

Comandos de Controle – MATCH-CASE

- A estrutura *match-case* é capaz de encontrar além de valores, também padrões.
- Veja o resultado:

```
two values: 1 and 2  
two values: 9 and 10  
three values: 1, 2 and 3  
single value: 1  
too many values
```

- Mas surge a pergunta: é capaz de encontrar padrão e valor?
- Com certeza!

Conhecendo a Linguagem Python

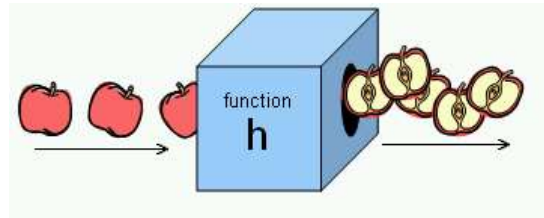
Comandos de Controle – MATCH-CASE

- Analise mais um exemplo, agora listas dentro de uma lista:

```
for thing in [[1,2],[9,10],[1,2,3],[1],[0,0,0,0,0]]:
    match thing:
        case [x]:
            print(f"single value: {x}")
        case [1,y]:
            print(f"two values: 1 and {y}")
        case [x,10]:
            print(f"two values: {x} and 10")
        case [x,y]:
            print(f"two values: {x} and {y}")
        case [x,y,z]:
            print(f"three values: {x}, {y} and {z}")
        case _:
            print("too many values")
```

```
two values: 1 and 2
two values: 9 and 10
three values: 1, 2 and 3
single value: 1
too many values
```

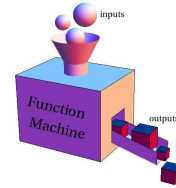
Funções



Conhecendo a Linguagem Python

Funções

- Permite pensar num problema em diversos níveis.
- Dividir para conquistar:
 - Um problema é dividido em diversos subproblemas;
 - As soluções dos subproblemas são combinadas numa solução do problema maior.
- Programas são divididos em subprogramas:
 - Cada subprograma é invocado por meio de um identificador e uma lista de entradas.
 - Os resultados computados por um subprograma pode ser combinado com os de outros subprogramas.



Conhecendo a Linguagem Python

Funções

- Em **Python** subprogramas tem nome de **funções**.
- São blocos de código com **nome** e/ou **argumentos**.
- Usadas para realizar **tarefas**.
- Formato geral:

```
1 def nome_funcao(arg, arg, arg, ...): #infinitos argumentos
2     #bloco de código
3
```

Conhecendo a Linguagem Python

Funções

- A palavra reservada ***def*** inicia a definição de uma função.
- Deve ser seguida do nome da função e da lista de **parâmetros**.
- Os comandos da função começam na linha **seguinte**.
- Seus comando devem ser **indentados**.
- **Opcionalmente**, a primeira linha da função pode ser utilizada para **documentar** a função:
 - Se presente, essa *string* chama-se ***docstring***.

Conhecendo a Linguagem Python

Funções

- Tipicamente computam um ou mais valores.
- Para indicar o valor a ser devolvido como resultado usa-se o comando:
 - `return expressão`
- Onde expressão é **opcional** ou o **valor** a ser retornado.
- Ao encontrar o comando `return`, a função **finaliza imediatamente**, retornando ao ponto que foi **chamada**.
- Se nenhum valor de retorno é especificado, o valor de retorno é ***None***.

Função em Python

Você sabe por que é importante usar funções num programa?

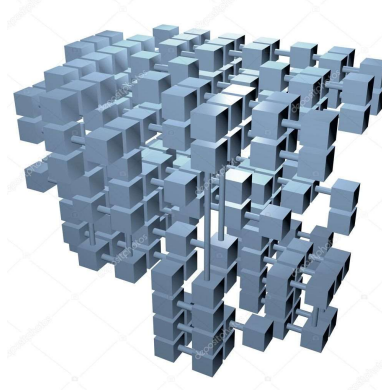
Imagine um trecho de programa que tenha que ser executado várias vezes durante a execução. O que é melhor: reescrever várias vezes a mesma coisa e criar um código imenso ou criar uma estrutura que execute este trecho sem necessidade de repeti-lo?

Com certeza, a segunda opção deixa o código menor, mais legível e mais organizado. Esta subdivisão de um programa mais complexo em partes menores é chamada modularização.

Inicialmente, devemos saber que a palavra reservada **def** é quem indica que será definida uma função. Em seguida, damos o nome da função e especificamos os nomes dos parâmetros de entrada.

Estes parâmetros podem assumir várias formas. Porém nunca são usados para retornar dados para o programa principal ou para a função “chamante”.

Estruturas de Dados



Conhecendo a Linguagem Python

Estruturas de Dados

- Forma de se organizar dados de maneira a facilitar seu acesso.
- Algumas formas são clássicas:
 - Listas;
 - Arrays (vetores e matrizes);
 - Tuplas (registros);
 - Árvores;
- Em Python existem dois tipos:
 - **Sequências:** são objetos ordenados e finitos;
 - **Dicionários:** um conjunto de elementos de mapeamentos indexados por chaves;
- Existem as sequências imutáveis: ***str*** e ***tuple***.
- E as mutáveis: ***dict*** e ***list***.

Conhecendo a Linguagem Python

Estruturas de Dados - Strings

- Uma *string* em Python é um conjunto linear de elementos imutáveis, mas de tamanho variável.
- Uma *string* em *Python* pode ser definida entre aspas simples ou duplas:

```
>>> msg1 = "Isto é uma mensagem"
>>> msg2 = 'Mas "frequentemente" me perguntam ...'
>>> print(msg1)
Isto é uma mensagem
>>> print(msg2)
Mas "frequentemente" me perguntam ...
>>>
```

- Assim como as listas/tuplas, os elementos de uma *string* podem ser acessadas por índices ou *slices*:

```
>>> msg1[0]
'I'
>>> print(msg2[5:19])
frequentemente
>>>
```

Conhecendo a Linguagem Python

Estruturas de Dados - Strings

- Uma *string* pode conter várias linhas, usando o caractere de quebra de linha “\”.
- E ainda pode possuir caracteres de controle como mudança de linha, “\n”, tabulações, “\t”, entre outros.

>>> Falar de triple quotes: <https://www.askpython.com/python/string/difference-between-single-and-double-quotes-in-python>

```
>>> msg = "\tFalar é completamente fácil, \
quando se tem a palavra em mente \
\nDifícil é se expressar por gestos e atitude."
>>> print(msg)
    Falar é completamente fácil, quando se tem a palavra em mente
Difícil é se expressar por gestos e atitude.
>>>
```

Conhecendo a Linguagem Python

Estruturas de Dados - Strings

- *String* possui muitos métodos promissores.
- Permitem várias operações:
 - Substituir;
 - Capitalizar;
 - Converter maiúsculo e minúsculo;
 - ... entre outras;

```
>>> a = 'carlos drummond de andrade'
>>> a.capitalize()
'Carlos drummond de andrade'
>>> a.title()
'Carlos Drummond De Andrade'
>>> a.upper()
'CARLOS DRUMMOND DE ANDRADE'
>>> a.lower()
'carlos drummond de andrade'
```

Métodos de String

A tabela a seguir apresenta os métodos utilizados para editar uma *string*.

Método	Descrição
<code>.capitalize()</code>	retorna o conteúdo da <i>string</i> com apenas a primeira letra capitalizada
<code>.center(width[, fillchar])</code>	centraliza a <i>string</i> , dentro da largura <i>width</i> , preenchendo com o caracter <i>fillchar</i>
<code>.expandtabs([tabsize])</code>	substitui os caracteres de tabulação por espaços. Se <i>tabsize</i> for omitido, será usado 8 espaços por padrão
<code>S.join(seq)</code>	retorna uma <i>string</i> com a concatenação das <i>strings</i> em sequência (lista/tupla de <i>strings</i>). A <i>string</i> <i>S</i> será usada como separador.
<code>.ljust(width[, fillchar])</code>	retorna uma <i>string</i> justificada a esquerda, com a largura <i>width</i>
<code>.lower()</code>	retorna uma <i>string</i> em caixa baixa
<code>.lstrip([chars])</code>	o mesmo que <i>strip</i> , abaixo, pela esquerda
<code>.replace(old, new[, n])</code>	substitui a n-ésima ocorrência de <i>old</i> , na <i>string</i> , por <i>new</i>
<code>.rjust(width[, fillchar])</code>	retorna uma <i>string</i> justificada a direita, com a largura <i>width</i>
<code>.rstrip([chars])</code>	o mesmo que <i>strip</i> , abaixo, pela direita
<code>.strip([chars])</code>	remove espaços (ou <i>chars</i>) do início e final da <i>string</i>
<code>.swapcase()</code>	substitui caixa alta por caixa baixa e vice-versa
<code>.title()</code>	coloca em caixa alta apenas a primeira letras de todas as palavras de uma <i>string</i>
<code>.translate(table [, delchars])</code>	retorna uma <i>string</i> com os caracteres substituídos, usando <i>table</i> para superpor a tabela ascii
<code>.upper()</code>	retorna uma <i>string</i> em caixa alta
<code>.zfill(width)</code>	retorna uma <i>string</i> com zeros à esquerda, para preencher até a largura <i>width</i>

Conhecendo a Linguagem Python

Estruturas de Dados - Strings

- Permite também as operações de adição, multiplicação, operadores de comparação, entre outros.
- Verifique abaixo alguns destes comandos:

```
>>> a = 'carlos '  
>>> b = 'drummond'  
>>> c = ' de andrade'  
>>> a+b+c  
'carlos drummond de andrade'  
>>> d = a+b+c  
>>> d  
'carlos drummond de andrade'  
>>> 'andrade' in d  
True  
>>> a > b  
False  
>>> a.__len__()  
7  
>>> len(a+b+c)  
26  
>>>
```

Métodos de String

Métodos de saída booleana no tipo *string*:

Método	Descrição
<code>.startswith(prefix[, s[, e]])</code>	retorna <i>True</i> se a <i>string</i> iniciar com a <i>string prefix</i> . <i>start</i> e <i>end</i> delimitam o intervalo de busca
<code>.endswith(suffix[, s[, e]])</code>	retorna <i>True</i> se a <i>string</i> terminar com a <i>string suffix</i> . <i>start</i> e <i>end</i> delimitam o intervalo de busca
<code>.isalnum()</code>	retorna <i>True</i> se o conteúdo da <i>string</i> for alfanumérico
<code>.isalpha()</code>	retorna <i>True</i> se o conteúdo da <i>string</i> for alfabético
<code>.isdigit()</code>	retorna <i>True</i> se o conteúdo da <i>string</i> for numérico
<code>.islower()</code>	retorna <i>True</i> se o conteúdo da <i>string</i> for caixa baixa
<code>.isspace()</code>	retorna <i>True</i> se o conteúdo da <i>string</i> for espaços vazios
<code>.istitle()</code>	retorna <i>True</i> se o conteúdo da <i>string</i> for um título, veja <code>.title()</code>
<code>.isupper()</code>	retorna <i>True</i> se o conteúdo da <i>string</i> for caixa alta

Métodos de pesquisa e divisão em lista da *string*

Método	Descrição
<code>.count(sub[, s[, e]])</code>	conta o número de ocorrências da <i>string</i> sub, no intervalo <i>start-end</i>
<code>.find(sub [, s[, e]])</code>	retorna o índice da ocorrência de <i>string</i> sub
<code>.index(sub [, s[, e]])</code>	o mesmo que <code>find</code> , acima, com a diferença de gerar o erro <i>ValueError</i> , caso a pesquisa falhe
<code>.partition(sep)</code>	usando <i>sep</i> como separador, retorna uma tupla da <i>string</i> , com a forma (head, sep, tail)
<code>.rfind(sub [, s[, e]])</code>	o mesmo que <code>find</code> , acima, com a pesquisa sendo feita da direita para a esquerda
<code>.rindex(sub [, s[, e]])</code>	o mesmo que <code>index</code> , acima, mas a busca é feita da direita para a esquerda
<code>.rpartition(sep)</code>	mesmo que <code>partition</code> , acima, pela direita
<code>.rsplit([sep [, maxsplit]])</code>	o mesmo que <code>split</code> , abaixo, pela direita
<code>.split([sep [, maxsplit]])</code>	separa uma <i>string</i> em uma lista, usando <i>sep</i> como separador, até <i>maxsplit</i> elementos
<code>.splitlines([keepends])</code>	separa uma <i>string</i> em uma lista, usando quebras de linha como separador. Se <i>keepends</i> for verdadeiro, as quebras de linhas serão incluídas

Conhecendo a Linguagem Python

String

- Considerando String como uma lista, implemente um script Python para gerar o seguinte resultado:

P
y
t
h
o
n



Conhecendo a Linguagem Python

Estrutura de Dados - Tuplas

- As *tuplas*, assim como as listas, são **sequências**.
- No entanto **não** são **mutáveis**.
- É possível realizar *slices*, gerando uma **nova** tupla.
- Podem conter elementos de **diferentes tipos** dentro dela.
- As tuplas são representados por **()**.
- Não é possível adicionar, remover ou atualizar elementos de uma Tupla.

```
>>> t = (1, 2, 3, 4, 5)
>>> t[3]
4
>>> t[2] = 4
Traceback (most recent call last):
  File "<pyshell#80>", line 1, in <module>
    t[2] = 4
TypeError: 'tuple' object does not support item assignment
>>>
```


Conhecendo a Linguagem Python

Estrutura de Dados - Tuplas

- **E por que usar tuplas?**
 - Mais rápidas que as listas.
 - Usadas na formatação das strings.
 - Para “listas” de elementos **constantes** é a melhor opção.
 - Usadas com parâmetros de funções de parâmetros variáveis.
- Há mais de uma forma para se criar:

```
>>> minhaTupla = tuple()
>>> type(minhaTupla)
<class 'tuple'>
>>> minhaTupla = (1, 2, 3)
>>> type(minhaTupla)
<class 'tuple'>
>>> minhaTupla = 1, 2, 3
>>> type(minhaTupla)
<class 'tuple'>
```

Conhecendo a Linguagem Python

Estrutura de Dados - Tuplas

- Você não pode modificar, mas pode trocar:

```
>>> vogais = ('a', 'e', 'i', 'o')
>>> id(vogais)
51599856
>>> vogais = vogais + ('u',)
>>> vogais
('a', 'e', 'i', 'o', 'u')
>>> id(vogais)
47944304
```

- Observe que ao “mudar” o conteúdo da tuple *vogais*, mudamos também o seu *id*.

Conhecendo a Linguagem Python

Estrutura de Dados - Tuplas

- Para criar uma *tupla* com um único elemento, temos que incluir uma vírgula **final**:

```
>>> t1 = ('a',)
>>> type(t1)
<class 'tuple'>
>>>
```

- Sem a vírgula, Python entende ('a') como uma *string* entre parênteses:

```
>>> t2 = ('a')
>>> type(t2)
<class 'str'>
>>>
```

Conhecendo a Linguagem Python

Estrutura de Dados - Dicionários

- Outra estrutura de dados muito **útil** em Python é o *dicionário*, cujo tipo é ***dict***.
- Diferente de sequências que são indexadas por inteiros, dicionários são indexados por chaves (**keys**).
- As *keys* podem ser de **qualquer** tipo **imutável** (como *strings* e inteiros).
- Tuplas também podem ser chaves se não contiverem dados mutáveis.
 - Ou seja, apenas strings, inteiros ou outras tuplas.
- Dicionários são um conjunto **não ordenado** de pares **chave-valor**.

Dicionários em Python

Outra estrutura de dados muito útil embutida em Python é o *dicionário*, cujo tipo é *dict*. Dicionários são também chamados de “memória associativa” ou “vetor associativo” em outras linguagens. Diferente de sequências que são indexadas por inteiros, dicionários são indexados por chaves (*keys*), que podem ser de qualquer tipo imutável (como strings e inteiros). Tuplas também podem ser chaves se contiverem apenas strings, inteiros ou outras tuplas. Se a tupla contiver, direta ou indiretamente, qualquer valor mutável, não poderá ser chave. Listas não podem ser usadas como chaves porque podem ser modificadas *in place* pela atribuição em índices ou fatias, e por métodos como `append()` e `extend()`.

Um bom modelo mental é imaginar um dicionário como um conjunto não ordenado de pares chave-valor, onde as chaves são únicas em uma dada instância do dicionário. Dicionários são delimitados por chaves: {}, e contém uma lista de pares *chave:valor* separada por vírgulas. Dessa forma também será exibido o conteúdo de um dicionário no console do Python. O dicionário vazio é {}.

As principais operações em um dicionário são armazenar e recuperar valores a partir de chaves. Também é possível remover um *parchave:valor* com o comando `del`. Se você armazenar um valor utilizando uma chave já presente, o antigo valor será substituído pelo novo. Se tentar recuperar um valor usando uma chave inexistente, será gerado um erro.

O método `keys()` do dicionário devolve a lista de todas as chaves presentes no dicionário, em ordem arbitrária (se desejar ordená-las basta aplicar o a função [sorted\(\)](#) à lista devolvida). Para verificar a existência de uma chave, use o operador [in](#).

Conhecendo a Linguagem Python

Estrutura de Dados - Dicionários

- Dicionários são delimitados por **chaves**: **{ }**.
- Contém uma lista de pares **chave:valor** separada por vírgulas.
- Principais operações: **armazenar** e **recuperar** valores a partir de chaves.
- Valores são armazenados e recuperados por **colchetes**: **[]**.

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127
>>> print(tel)
{'guido': 4127, 'sape': 4139, 'jack': 4098}
>>> tel['jack']
4098
>>>
```

Conhecendo a Linguagem Python

Estrutura de Dados - Dicionários

- Também é possível remover um par *chave:valor* com o comando **del**.

```
>>> del tel['sape']
>>> print(tel)
{'guido': 4127, 'jack': 4098}
>>>
```

- Ao atribuir valor a uma chave **existente**, remove-se o valor antigo.

```
>>> print(tel)
{'guido': 4127, 'jack': 4098}
>>>
>>> print(tel)
{'guido': 4127, 'jack': 4098}
>>> tel['guido'] = 9999
>>> print(tel)
{'guido': 9999, 'jack': 4098}
>>>
```

Conhecendo a Linguagem Python

Estrutura de Dados - Dicionários

- Se buscar uma chave inexistente é gerado um erro.

```
>>> tel
{'guido': 9999, 'jack': 4098}
>>> tel['jone']
Traceback (most recent call last):
  File "<pyshell#87>", line 1, in <module>
    tel['jone']
  KeyError: 'jone'
>>>
```

- O método **keys()** do dicionário devolve a lista de todas as chaves presentes no dicionário.

```
>>> tel.keys()
dict_keys(['guido', 'jack'])
>>> .
```

Conhecendo a Linguagem Python

Estrutura de Dados - Dicionários

- O construtor ***dict()*** produz dicionários diretamente a partir de uma lista de chaves-valores.
- Para isso devem ser armazenadas como duplas (*tuplas de 2 elementos*).

```
>>> dict([('sape', 4923), ('guido', 4127), ('jack', 4567)])  
{'guido': 4127, 'sape': 4923, 'jack': 4567}  
>>>
```

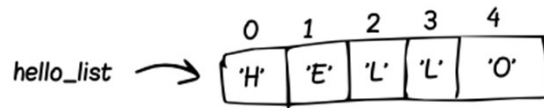
- Com o operador **in** podemos verificar se um elemento é membro de um dicionário:

```
>>> print(tel)  
{'guido': 4127, 'sape': 4923, 'jack': 4567}  
>>> 'sape' in tel  
True  
>>> 'jone' in tel  
False
```


Conhecendo a Linguagem Python

Estrutura de Dados - Listas

- Forma de organização através da **enumeração** dos dados.
- Teoricamente possui tamanho **infinito**.
- Acesso eficiente aos seus elementos em **ordem sequencial**.
- Cada valor é identificado por um **índice**.
- Os índices **iniciam** em **0**.



Listas em Python

Uma **lista** é um conjunto ordenado de valores, onde cada valor é identificado por um índice. Os valores que compõem uma lista são chamados **elementos**. Listas são similares a strings, que são conjuntos ordenados de caracteres, com a diferença que os elementos de uma lista podem possuir qualquer tipo. Listas e strings XXX e outras coisas que se comportam como conjuntos ordenados XXX são chamados **seqüências**.

Existem várias maneiras de criar uma nova lista; a mais simples é envolver os elementos em colchetes ([e]).

A sintaxe para acessar os elementos de uma lista é a mesma que a sintaxe para acessar os caracteres de uma string XXX o operador colchete ([]). A expressão dentro dos colchetes especifica o índice. Lembre-se que os índices iniciam em 0.

A função **len** devolve o comprimento de uma lista. É uma boa ideia utilizar este valor como o limite superior de um laço ao invés de uma constante. Desta forma, se o tamanho da lista mudar, você não precisará ir através de todo o programa modificando todos os laços; eles funcionarão corretamente para qualquer tamanho de lista.

Conhecendo a Linguagem Python

Estrutura de Dados - Listas

- Em Python é definida por *list*.
- Em Python a representação de lista é feita por `[]`.
- Listas são sequências **mutáveis**.
- Uma *list* é uma **sequência**, assim como as *strings*.

```
>>> [10, 20, 30, 40]
[10, 20, 30, 40]
>>> ['spam', 'bungee', 'swallow']
['spam', 'bungee', 'swallow']
>>>
```

Conhecendo a Linguagem Python

Estrutura de Dados - Listas

- Uma *list* em Python é **diferente** das tradicionais listas.
- São listas mais **generalizadas**.
- Podem conter elementos de tipos diferentes.
- Uma lista dentro de outra lista é dita estar **aninhada**.

```
>>> ['alô', 2.0, 5, [10, 20]]  
['alô', 2.0, 5, [10, 20]]  
>>>
```

Conhecendo a Linguagem Python

Estrutura de Dados - Listas

- As listas podem ser criadas de **duas formas**:

```
>>> #Usando o construtor
>>> alunos_curso = list()
>>> type(alunos_curso)
<class 'list'>
>>> #Mais comum usando []
>>> alunos_curso = []
>>> type(alunos_curso)
<class 'list'>
>>> .
```

- Acima, exemplo de listas **vazias**:

```
>>> len(alunos_curso)
0
>>>
```

- Assim como para strings, o comando **len()** retorna o tamanho da list.

Conhecendo a Linguagem Python

Estrutura de Dados - Listas

- As listas podem ser criadas definindo direto o seu conteúdo.

```
>>> num_par = [0, 2, 4, 6, 8]
>>> print(num_par)
[0, 2, 4, 6, 8]
>>> socios = ['Fabio', 'Silvio', 'Nino', 'Pedro']
>>> print(socios)
['Fabio', 'Silvio', 'Nino', 'Pedro']
>>> len(socios)
4
>>>
```

Conhecendo a Linguagem Python

Estrutura de Dados - Listas

- Acesso aos valores:
 - Utiliza-se o mesmo operador de string, o `[]`.
 - Os **slices** também funcionam aqui.

```
>>> alunos = [['Fabio', 20], ['Pedro', 21], ['Tiago', None]]
>>> alunos[1]
['Pedro', 21]
>>> alunos[1:3]
[['Pedro', 21], ['Tiago', None]]
>>> nome, idade = alunos[2]
>>> print(nome)
Tiago
>>> print(idade)
None
>>>
```

Conhecendo a Linguagem Python

Estrutura de Dados - Listas

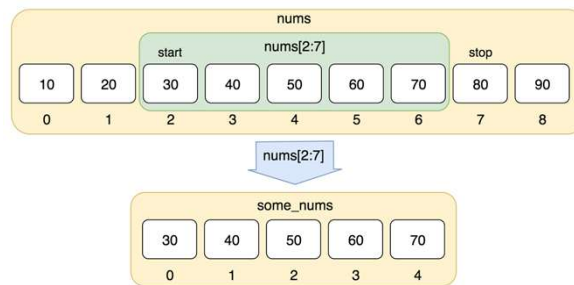
- A sintaxe completa para *slice*:
 - ***start: stop: step***
 - *start*: refere-se ao índice do elemento que é usado como o início de nossa fatia.
 - *stop*: refere-se ao índice do elemento que devemos parar antes de terminar nossa fatia.
 - *step*: permite que você pegue cada enésimo elemento dentro de um intervalo *start: stop*:

Conhecendo a Linguagem Python

Estrutura de Dados - Listas

- Confira outro exemplo:

```
>>> nums = [10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> some_nums = nums[2:7]
>>> some_nums
[30, 40, 50, 60, 70]
>>>
```



Conhecendo a Linguagem Python

Estrutura de Dados - Listas

- Pegando os primeiros elementos da lista:
 - A notação de *slice* permite que você ignore qualquer elemento da sintaxe completa. Se pularmos o número inicial, ele começará a partir do índice 0:

```
>>> nums = [10, 20, 30, 40, 50, 60, 70, 80, 90]
>>> nums[:5]
[10, 20, 30, 40, 50]
>>>
```

- Portanto, `nums[:5]` é equivalente a `nums[0:5]`.

Conhecendo a Linguagem Python

Estrutura de Dados - Listas

- Mais exemplos com slice em Python confira:
 - <https://railsware.com/blog/python-for-machine-learning-indexing-and-slicing-for-lists-tuples-strings-and-other-sequential-types/>

Conhecendo a Linguagem Python

Estrutura de Dados - Listas

- Elementos das listas podem ser modificadas:

```
>>> passaros = ['canario', 'pica-pau', 'papagaio', 'foca']
>>> #ops... foca não é ave
>>> passaros[3]
'foca'
>>> passaros[3] = 'joão-de-barro'
>>> print(passaros)
['canario', 'pica-pau', 'papagaio', 'joão-de-barro']
>>> len(passaros)
4
>>>
```

- E também removidos (comando **del()** ou método **remove()**):

```
>>> del passaros[1]
>>> print(passaros)
['canario', 'papagaio', 'joão-de-barro']
>>> #Também pode-se remover pelo valor
>>> passaros.remove('canario')
>>> print(passaros)
['papagaio', 'joão-de-barro']
>>>
```

Conhecendo a Linguagem Python

Estrutura de Dados - Listas

- O operador `+` concatena listas:

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> print(c)
[1, 2, 3, 4, 5, 6]
>>>
```

- Com o operador `in` podemos verificar se um elemento é membro de uma lista:

```
>>> cavaleiros = ['guerra', 'fome', 'peste', 'morte']
>>> 'peste' in cavaleiros
True
>>> 'depravação' in cavaleiros
False
>>>
```

Conhecendo a Linguagem Python

Estrutura de Dados - Listas

- Métodos úteis de list():
 - list.append(x): adiciona um item para o fim da lista;
 - list.insert(i, x): insere um item em uma determinada posição;
 - list.pop(index): remove o item na posição determinada na lista, e o retorna. Caso não seja passado a posição é retirado o último;
 - list.count(x): retorna o número de vezes que x aparece na lista;
 - list.reverse(): inverte os elementos da lista, no própria instância;
- Com os métodos append e pop permite-se trabalhar com filas: FIFO - *First In, First Out*.

Conhecendo a Linguagem Python

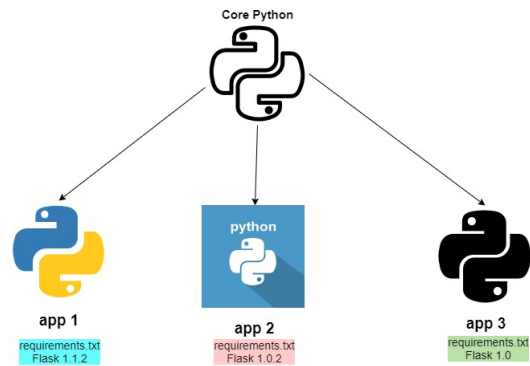
Estrutura de Dados - Listas

- Métodos na prática:

```
>>> familia = ['Leandro', 'Eloise', 'Ellen']
>>> familia.append('Unida')
>>> familia
['Leandro', 'Eloise', 'Ellen', 'Unida']
>>> last_value = familia.pop()
>>> last_value
'Unida'
>>> familia
['Leandro', 'Eloise', 'Ellen']
>>> familia.insert(0, 'Todos')
>>> familia
['Todos', 'Leandro', 'Eloise', 'Ellen']
>>> first_value = familia.pop(0)
>>> first_value
'Todos'
>>> familia
['Leandro', 'Eloise', 'Ellen']
>>> familia.count('Eloise')
1
>>> familia.reverse()
>>> familia
['Ellen', 'Eloise', 'Leandro']
...

```

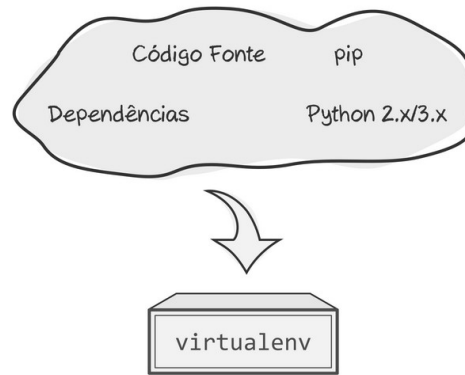
Python Virtual Environment



Ambiente Virtualizado Python

- Quando desenvolvemos em Python, é comum ficarmos maravilhados com o poder de suas bibliotecas.
- Tudo se torna mais fácil ainda quando podemos baixá-las com um simples comando, por exemplo:
 - `pip install <pacote>`
- Com o ambiente virtual é possível se desenvolver em Python isoladamente, sem conflitos entre versões de bibliotecas.
- Outro ponto, é poder desenvolver um projeto em Python 2 e outro em Python 3, no mesmo sistema operacional.

Como Funciona?



Como o Virtualenv Funciona:

O funcionamento do **virtualenv** é realmente simples. Ele basicamente cria uma cópia de todos os diretórios necessários para que um programa Python seja executado, isto inclui:

- As bibliotecas comuns do Python (*standard library*);
- O gerenciador de pacotes pip;
- O próprio binário do Python (Python 2.x/3.x);
- As dependências que estiverem no diretório site-packages;
- Seu código fonte descrevendo sua aplicação.

Assim, ao instalar uma nova dependência dentro do ambiente criado pelo **virtualenv**, ele será colocado no diretório **site-packages** relativo à esse ambiente, e não mais globalmente.

Instalação

- Para a instalação do *virtualenv* vamos precisar do pip.
- Agora com o pip, precisamos executar apenas um comando para instalar o virtualenv:
 - `sudo pip install virtualenv`
- **Pronto! virtualenv** instalado e funcionando! Agora vamos começar a utilizá-lo!

Inicializando um VirtualEnv

- Vamos criar um *virtualenv* chamado **cursoenv**.
- Execute o seguinte comando:
 - `virtualenv -p python cursoenv`
- Esse comando cria um novo ambiente de desenvolvimento totalmente isolado!
- **cursoenv** é o nome da sua *virtualenv*. Pode-se usar qualquer outro nome, mas permaneça em minúsculo e sem espaços entre os nomes.
- Após a criação da *virtualenv* devemos ativá-la com o comando:
 - `cursoenv\Scripts\activate.bat`

Alterar Terminal do Visual Studio Code

No Windows a *VirtualEnv* permite ser utilizada no prompt CMD: `C:/Windows/System32/cmd.exe`.

Para utilizarmos este *prompt* integrado ao Visual Studio Code, devemos alterar a configuração de *Powershell* para o *CMD*, siga os seguintes passos:

1. No Visual Studio Code acesse: File > Preferences > Settings.
2. No canto superior direito clique no ícone { } (*Open JSON settings*).
3. Na tela da direita (*User Settings*) inclua a seguinte configuração personalizada para o seu Visual Studio Code:

"terminal.integrated.shell.windows": "C:\\Windows\\System32\\cmd.exe"

VirtualEnv

- Para desativar utilize o comando:
 - `cursoenv\Scripts\deactivate.bat`
- Agora com a *Virtualenv* iniciada, podemos instalar qualquer biblioteca Python usando o comando `pip3`, confira
 - `pip install ipdb`
- Listar bibliotecas instaladas:
 - `pip freeze`
- Salvar bibliotecas instaladas:
 - `pip freeze > requirements.txt`
- Instalar bibliotecas salvas para um novo *VirtualEnv*:
 - `pip install -r requirements.txt`

Em ambientes MacOS e Linux

Ativar virtualenv:

- `source cursoenv/bin/activate`

Desativar virtualenv:

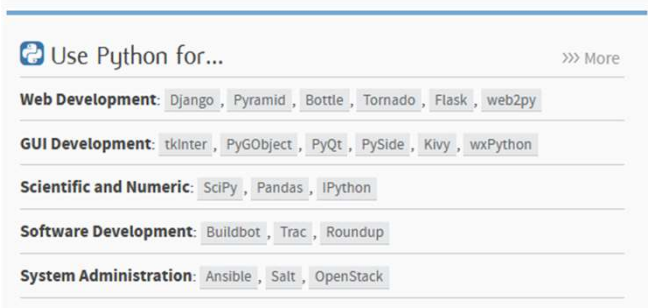
- `source cursoenv/bin/deactivate`



Python Mão na Massa!!!

django

- Continue os estudos.
- Faça algo prático.
- Pesquise e teste frameworks.



Referência:

Applications for Python: <https://www.python.org/about/apps/>

Referências

- <http://www.python.org/getit/>
- <https://atom.io/>
- <http://pycursos.com/python3-para-desenvolvedores/>
- <http://learnpythonthehardway.org/book/ex44.html>
- <http://www.ibm.com/developerworks/br/library/l-python3-2/>
- <http://wiki.python.org.br/ListaDeExercicios>
- <http://www.cbsi.net.br/2015/08/profissionais-com-conhecimentos-em-python-ganham-mais.html>
- <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>
- http://spectrum.ieee.org/computing/software/the-2015-top-ten-programming-languages/?utm_source=techalert&utm_medium=email&utm_campaign=072315