





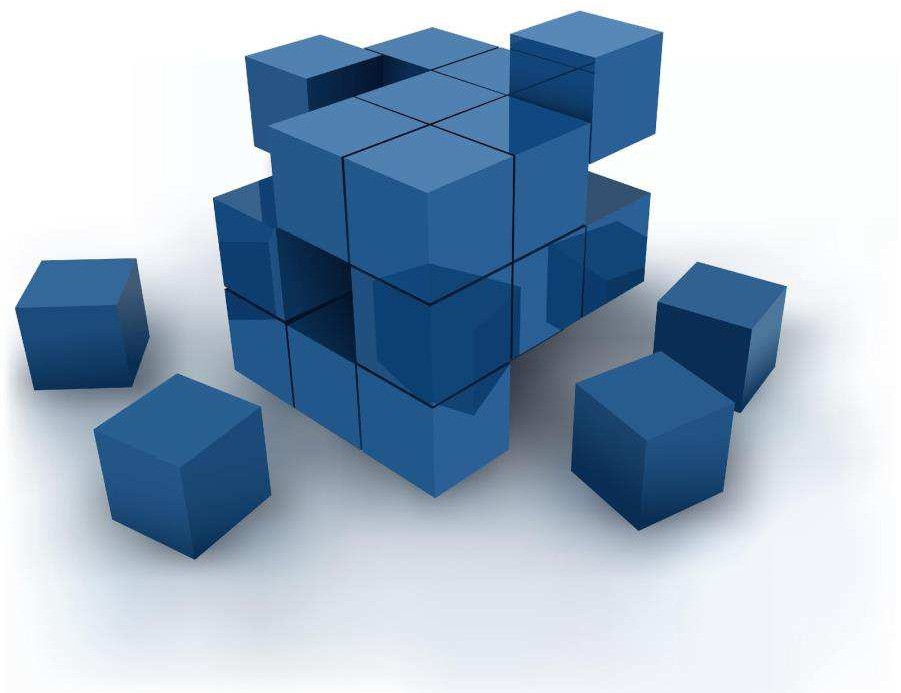
# ESTRUTURA DE DADOS



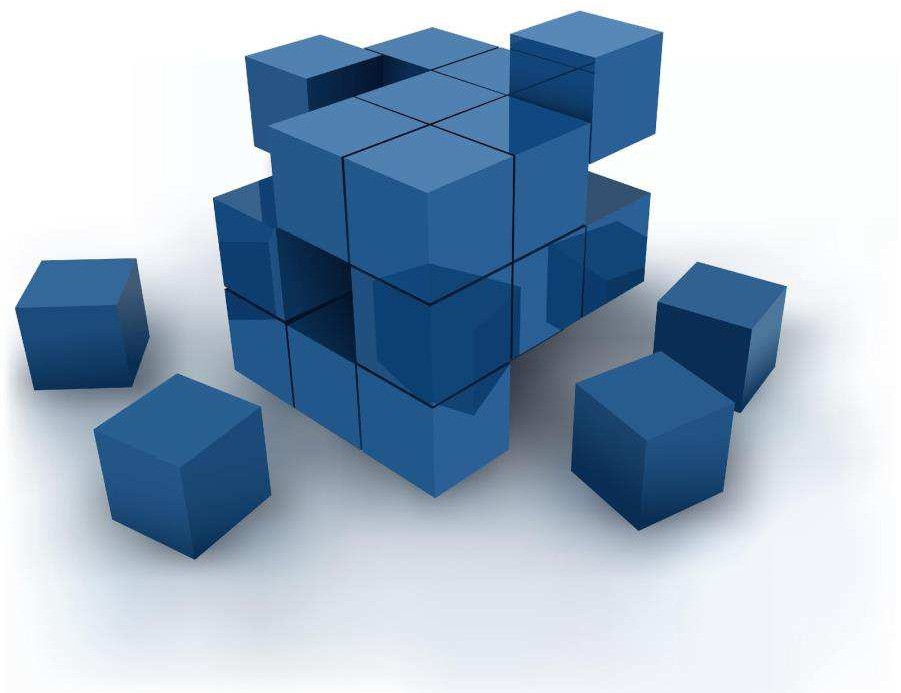


**@prof.felipeassuncao**

## O que são Estruturas de Dados?

* Temos um conjunto básico de dados primitivos (inteiro, real, caractere e lógico).
* Quando agrupamos estes dados, formamos uma **estrutura**.
* Exemplo: Este agrupamento pode resultar em vetores (matrizes unidimensionais), matrizes (com mais de uma dimensão) ou registros.



* Listas
* Tuplas
* Conjuntos
* Dicionários



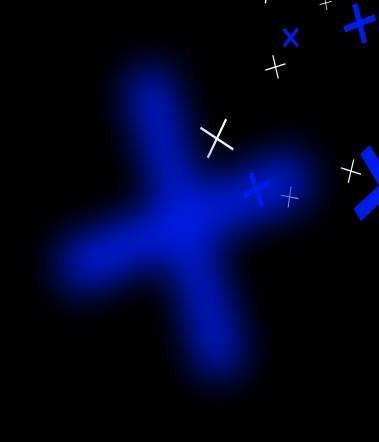
Aula 1 - Listas e Tuplas

Aula 2 - Conjuntos, Pilhas e Filas Aula 3 - Dicionários

Aula 4 - Funções e Módulos Aula 5 - Exercícios

* Jupyter Notebook





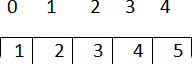
# LISTAS E TUPLAS

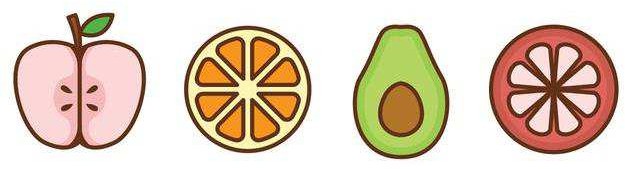












* Lista é um tipo de dados embutidos no Python usados para armazenar coleções de dados
* As listas são usadas para armazenar vários itens em uma única variável
* Os ítens de uma lista podem ser acessados como em um vetor: lista[0] e podem ter tipos de dados diferentes
* As listas são criadas usando colchetes
* Para criar uma lista em Python, a sintaxe é a seguinte:

>>> lista = [] # Criação de uma lista vazia

>>> lista = [1, 2, 3] # Criação de uma lista de inteiros

>>> lista = [1, "Olá, mundo!", 1.1] # Criação de uma lista com vários tipos diferentes

* Podemos criar listas dentro de outras listas (nested)

>>> lista = ["Olá, mundo", [1, 2, 3], ["outra\_lista"]]

|  |  |
| --- | --- |
| **METODO** | **COMANDO** |
| L1 + L2 | Concatenação |
| L \* 5 | Repetição |
| <valor> in L | Verificação de existência |
| for x in L: | Iteração |
| L.append(x) | Acrescentar itens |
| L.insert(POS, x) | Acrescentar itens na posição |
| L.index(x) | Busca de posição por valor |
| L.count(x) | Contagem de ocorrências de x |
| L.sort(x) | Ordena os elementos da lista |
| L.remove(x) | Remove o primeiro item encontrado na lista cujo valor é igual a x |
| L.pop(x) | Remove um item em uma dada posição na lista e o retorna |
| L.reverse(x) | Inverte a ordem dos elementos na lista. |
| L.copy() | Devolve uma cópia rasa da lista. Equivalente a a[:] |

* Você pode escrever códigos mais curtos e mais eficazes
* Como consequência, seu código será executado de forma mais rápida
* Para criar uma compreensão de listas em Python, a sintaxe é a seguinte

>>> [expr for item in lista]

* Dado o seguinte código:

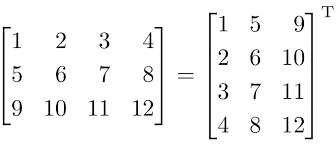
>>> for item in range(10):

>>> lista.append(x\*\*2)

* Podemos escrever uma compreensão de lista da seguinte forma:

# aplicação da potência 2 em todos os itens da lista

>>> lista = [item\*\*2 for item in range(10)]

>>> transposta = []

>>> matriz = [[1, 2, 3, 4], [4, 5, 6, 8], [9, 10, 11, 12]]

>>> for i in range(len(matriz[0])):

>>> linha\_transposta = []

>>>

>>> for linha in matriz:

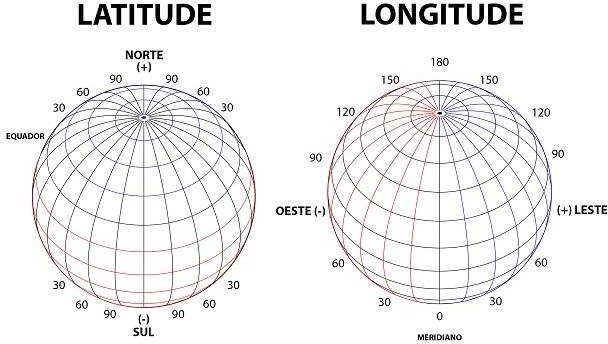
>>> linha\_transposta.append(linha[i])

>>> transposta.append(linha\_transposta)

>>> transposta

[[1, 4, 9], [2, 5, 10], [3, 6, 11], [4, 8, 12]]

>>> transposta **=** [[linha[i] **for** linha **in** matriz] **for** i **in** range(4)]

* Tuplas são tipos de dados de sequência, como listas e intervalo (range)
* Os itens de tupla são ordenados, imutáveis e permitem valores duplicados
* Consiste em uma sequência de valores separados por virgula, como exemplo:

tupla = maçã, 12345, ‘Olá mundo!’

* Os itens de tupla são indexados:
  + o primeiro item tem índice [0]
  + o segundo item tem índice [1]

>>> t = 12345, 54321, 'hello!'

>>> t[0]

12345

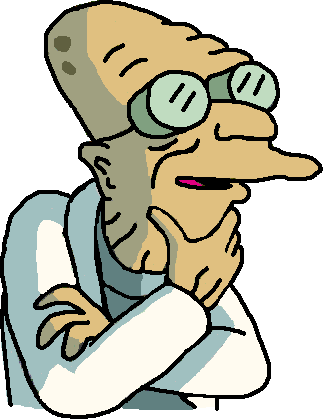
>>> t

(12345, 54321, 'hello!')

>>> # Tuples may be nested:

... u = t, (1, 2, 3, 4, 5)

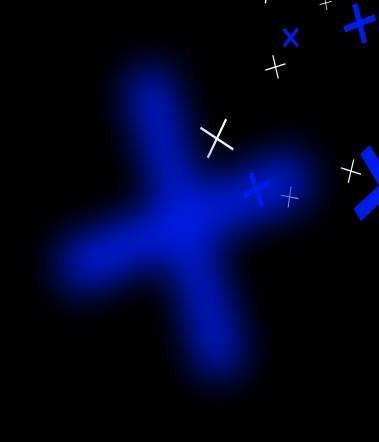
>>> u

* A estrutura de tupla é imutável, já a lista é mutável e pode crescer livremente
* Ambas tem utilizações distintas, como já abordamos:
  + a lista, por exemplo, pode se estender infinitamente
  + a tupla, em teoria, devemos conservar a sua estrutura
* Nesta prática iremos explorar a utilização listas e tuplas e trabalhar algumas operações.
* Listas
* Tuplas
* Operações





# CONJUNTO, PILHAS E FILAS





* Conjuntos são usados para armazenar vários elementos em uma única variável
* Um conjunto é uma coleção desordenada e indexada de elementos
* Elementos do conjunto não podem ser alterados
* Valores repetidos não são permitidos
* Usamos chave ou a função set( ) para declarar conjuntos

conjunto = {‘banana’, ‘maçã’, ‘laranja’, ‘uva’}

* Atenção ao criar conjuntos vazios! Use set( ) e não { }

|  |  |  |
| --- | --- | --- |
| **SÍMBOLO MATEMÁTICO** | **OPERADOR PYTHON** | **DESCRIÇÃO** |
| *e*∈S | in | elemento *e* é membro de *S* |
| A⊆B | <= | *A* é um subconjunto de *B* |
| A⊂B | < | *A* é um subconjunto próprio de *B* |
| A∪B | | | *A* união com *B* |
| A∩B | & | *A* interseção com *B* |
| A∖B | - | Diferença entre *A* e *B* |

>>> A = {0, 1, 3, 5, 7, 9}

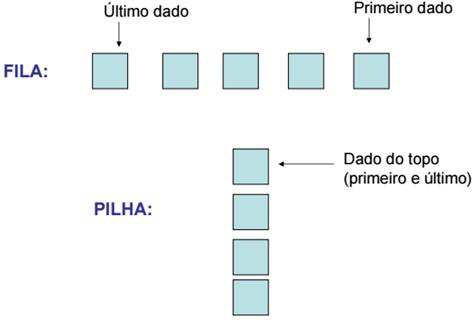
>>> B = {0, 2, 4, 6, 8}

>>> C = A.union(B) #ou de forma mais concisa C = A | B

>>> print(C)

{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

**FIFO**



**LIFO**

Podemos declarar uma pilha utilizando a seguinte sintaxe:

>>> pilha = [1, 2, 3]

>>> pilha.append(4)

>>> pilha.append(5)

>>> pilha

[1, 2, 3, 4, 5]

>>> pilha.pop() 5

>>> pilha.pop()

4

Podemos declarar uma fila utilizando a seguinte sintaxe:

>>> from collections import deque

>>> fila = deque(["Eric", "John", "Michael"])

>>> fila.append("Terry") # Terry arrives

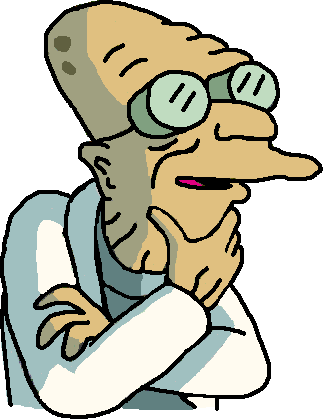
>>> fila.append("Graham") # Graham arrives

>>> fila.popleft() # The first to arrive now leaves 'Eric'

>>> fila.popleft() # The second to arrive now leaves 'John'

>>> fila # Remaining queue in order of arrival

fila(['Michael', 'Terry', 'Graham'])

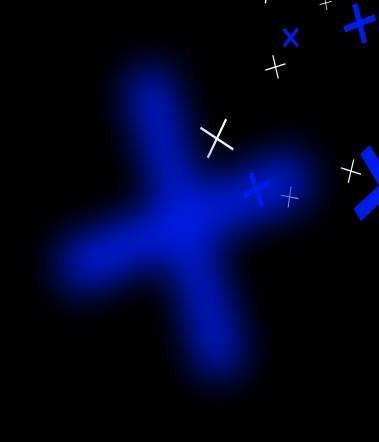
* Nesta prática iremos explorar como declarar conjuntos, pilhas, filas e executar operações em cada uma dessas estruturas



* Conjuntos, Pilhas, Filas
* Operações







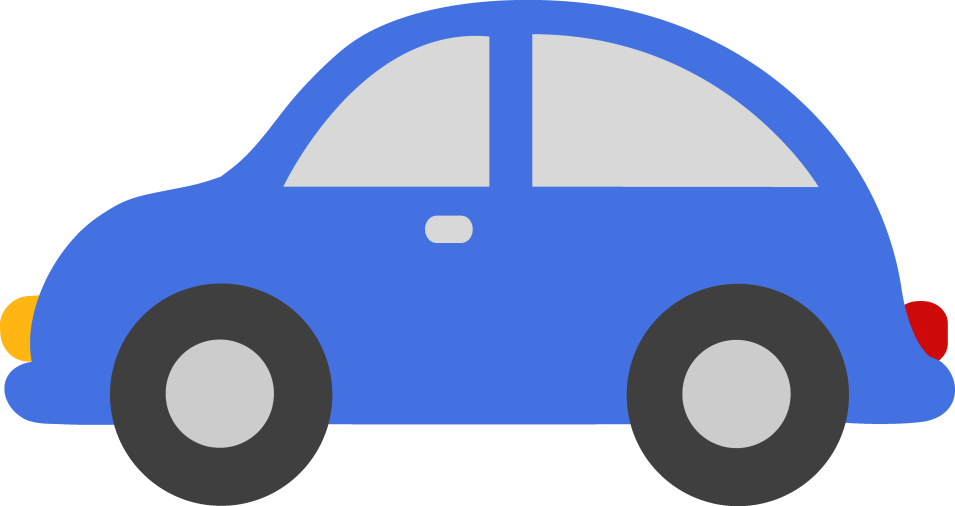
# DICIONÁRIOS







* Como criar um dicionário sobre carros?





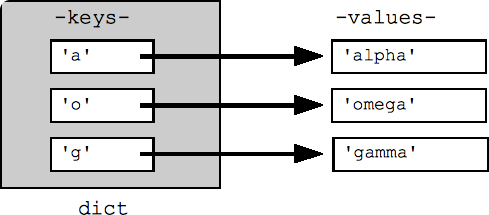
* Como criar dicionários sobre funcionários?



* Dicionários são usados para armazenar valores de dados em pares:

**chave: valor**

* Um dicionário é uma coleção não ordenada, mutável e não permite duplicatas.



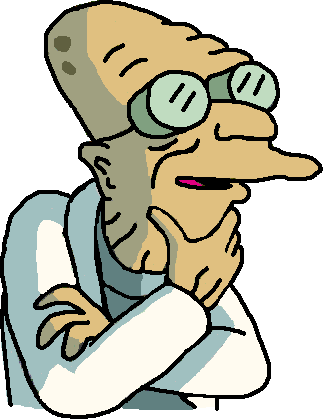
* Os dicionários são escritos com chaves e suportam vários tipos de dados, com a sintaxe:

>>> func = {"matricula":123, "nome": "José", "idade": 20, "salario": 9200.45}

>>> vazio = {} # criação de um dicionário vazio

>>> print(type(func))

<class ‘dict’>

* Nesta prática iremos explorar a utilização do dicionário e executar algumas operações
* Dicionários
* Operações

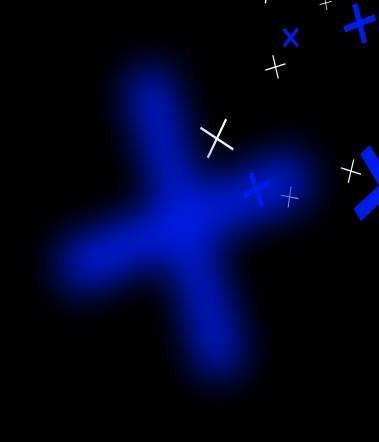




# FUNÇÕES E



**MÓDULOS**





* Ao longo dos nossos estudos conhecemos diversas funções como: len(), int(), float(), print(), type() etc.
* No contexto da programação, uma função é uma sequência instruções nomeadas que executa uma operação de computação.
* Ao definir uma função, você especifica o nome e a sequência de instruções. Depois, pode “chamar” a função pelo nome.

Podemos declarar uma função da seguinte forma:

>>> def nome(argumentos):

>>> instruções

>>> return alguma\_coisa

>>> type(50)

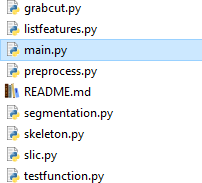
<class ‘int’>

>>> def imprime(mensagem):

>>> print(mensagem)

>>> mensagem = “Adoro aprender Python”

>>> imprime(mensagem) Adoro aprender Python

* Um módulo é um arquivo que contém uma coleção de funções relacionadas
* Vários módulos podem se comunicar através do comando:

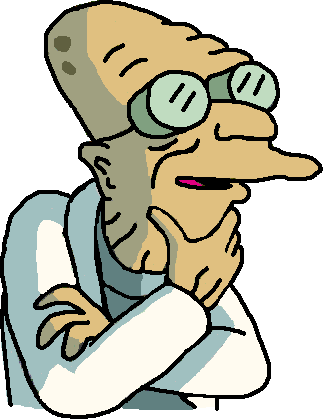
import nome\_módulo

* Como podemos utilizar módulos?

>>> import notafiscal

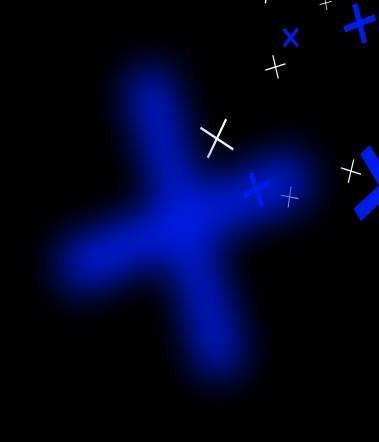
>>> import notafiscal \*

notafiscal.gerarpdf()

* Nesta prática iremos explorar a utilização das funções e módulos
* O que são funções
* O que são módulos
* Operações







**EXERCICIOS**





