Python

# Sumário

[**Sumário 1**](#_a2enfoh2th6l)

[**Tutorial de Python 3**](#_vasdwxxo0z68)

[1. Introdução 3](#_4aq943lhxxf0)

[2. Interpretador python 3](#_jq17srnj128m)

[3. Conceitos iniciais 4](#_2itqefvgsvpk)

[3.0 Números 4](#_l5aa6b9y0owx)

[3.1 Strings 6](#_2358dsyn1py9)

[3.2 Listas 8](#_nrp7zrcmcq97)

[4. Controle de fluxo 11](#_6ci7dhyr6dmk)

[4.0 Comandos if 11](#_3hp82wxlupz1)

[4.1 Comandos for 11](#_vbjarpvdur02)

[4.2 função range() 12](#_xz27nxhmdie6)

[4.3 Break, continue e cláusula else em laços de repetição 13](#_ygscmnglozb1)

[4.4 Comando pass 14](#_lrd1tuo8n7ro)

[4.5 Instrução match 14](#_rclml6ri6lo)

[4.6 Definindo funções 17](#_s8dvsiuludz)

[4.7 Definindo funções - Parte 2 19](#_x58uyb2ibxlc)

[5. Estrutura de dados 24](#_6l0faec65e7k)

[5.0 Listas 24](#_k934vbiilem6)

[5.1 Pilhas 26](#_8z217rmd6hlk)

[5.2 Filas 26](#_gb12r7wdp985)

[5.3 List Comprehensions 27](#_l17641b95p3b)

[5.4 Instrução del 27](#_88a2yzv8ltdt)

[5.5 Tuplas e sequências 28](#_kvsyaupswucr)

[5.6 Conjuntos 29](#_5w6adfbg8if3)

[5.7 Dicionários 30](#_s2uponrd8qhg)

[5.8 Iteração 31](#_4vbiiq2db8hb)

[5.9 Condições - Parte 2 33](#_tnbe5m522ogt)

[5.10 Comparando sequências 33](#_o6qzn290tlxi)

[6. Módulos 34](#_nhqi4yd8c4ru)

[6.0 Arquivos py compilados 36](#_vwuz4kyth0us)

[6.1 Módulos padrões 37](#_13kyk5mnf0x4)

[6.2 Função dir() 38](#_c3e3f9zbnn2n)

[6.3 Pacotes 39](#_ivjffjmmp0sq)

[6.4 Importando \* de um pacote 40](#_c1pn5v55y8l8)

[7. Entrada e Saída 41](#_xufdylo8bs3s)

[7.0 Refinando a formatação de saída 41](#_78bd2slnbe2o)

[7.0.0 Strings Literais Formatadas 42](#_e0wn4yy8pbu)

[7.0.1 Método Format() 43](#_tmunpdidymf3)

[7.0.2 Formatação Manual de Strings 44](#_2cgcnnkq1fml)

[7.0.3 Interpolação de strings 45](#_9osbk972mi6p)

[7.1 Leitura e escrita de arquivos 45](#_ik3u3iwiw8f9)

[7.1.0 Métodos de objetos arquivo 47](#_kqzsjceqlm7v)

[7.1.1 Gravando dados estruturados com json 48](#_8u8d3tu4yxz1)

[8. Erros e Exceções 49](#_hmeepu85um3w)

[8.0 Erros de sintaxe 49](#_eqjfu8j5wpbf)

[8.1 Exceções 50](#_o7l2v743otsv)

[8.2 Tratamento de Exceções 50](#_kqj771glu0ev)

[8.3 Levantando exceções 53](#_tgg5k7a0h6uv)

[8.4 Encadeamento de exceções 54](#_p37vybsi4lq2)

[8.5 Exceções definidas pelo usuário 54](#_nncwpcw87vdz)

[8.6 Definindo Ações de limpeza 55](#_360e9klfxh8g)

[8.7 Ações de limpeza predefinidas 56](#_xme93xn0xne)

[8.8 Criando e tratando várias exceções não relacionadas 56](#_f06uwckkgwxe)

[8.9 Enriquecendo exceções com notas 57](#_uvezam9lr0c3)

[9. Classes 57](#_dii09rjwtw1)

[9.0 Escopo e espaços de nomes do python 58](#_gj11nty2yp22)

[9.1 Sintaxe da definição de classe 60](#_eah55ctpe3lc)

[9.2 Objeto Classe 60](#_qf5we5b86f7z)

[9.3 Objetos Instância 62](#_pz8ir4yg2nch)

[9.4 Objetos método 62](#_e3sgwo41ju09)

[9.5 Variáveis de classe e instância 62](#_xtbcb1qt7wmh)

[9.6 Observações aleatórias 64](#_3v3d6j1wrpjz)

[9.7 Herança 65](#_c66xgagqa1rz)

[9.8 Herança múltipla 66](#_ipjrjectinyd)

[9.9 Variáveis Privadas 66](#_hvoaszr2mffp)

[9.10 Curiosidades e conclusões 67](#_68owxi41019c)

[9.11 Iteradores 68](#_a9khrhvucgjn)

[9.12 Geradores 69](#_68cqcexw212)

[9.13 Expressões Geradoras 69](#_mfj3ih7per21)

**Características**:

* Estruturas de dados de alto nível.
* Programação orientada a objetos.
* Tipagem dinâmica.

**Partes**:

1. Tutorial de Python - <https://docs.python.org/pt-br/3/tutorial/index.html>.
2. Biblioteca de Python - <https://docs.python.org/pt-br/3/library/index.html#library-index>.
3. Referência de Python - <https://docs.python.org/pt-br/3/reference/index.html#reference-index>.

# **Tutorial de Python**

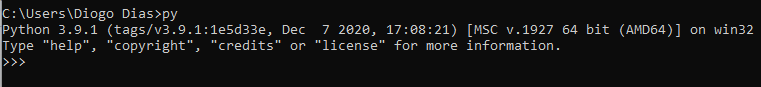
## 1. Introdução

**Benefícios da linguagem**:

* Melhor verificação de erros.
* Dicionários e arrays flexíveis.
* Permite organização em módulos e apresenta diversos módulos já disponíveis.
* definição de bloco por identação.
* sem necessidade de declaração de variáveis ou parâmetros formais.

## 2. Interpretador python

**Exemplo: inicializando interpretador python**.



**Digite py no terminal para inicializar o interpretador e ctrl+z para finalizar**.

**python -c “comando” [arg]** - inicia interpretador.

* executa instruções especificadas na posição de comando.

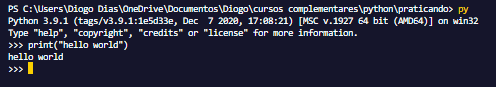
**python -m módulo [arg]** - chama módulos de python.

**Módulo sys** - armazena nome de scripts e argumentos da linha de comando que podem ser acessadas através de variável argv do módulo.

**Modo interativo - quando comandos são lidos a partir do console**.

* prompt primário representado por “>>>” (solicitação de comandos) e prompt secundário representado por “...” (continuação de comando).

**Exemplo**:



**Exemplo: linhas de continuação de comando**.



UTF-8 é a codificação padrão de python, nela caracteres de muitos idiomas podem ser usados simultaneamente em literais strings, identificadores e comentários.

**Exemplo: declarando codificação diferente da padrão**.



**Sempre deve ser adicionada na primeira linha de comando**.

**Nota: encoding é uma das codecs válidas com suporte do python**.

## 3. Conceitos iniciais

### **3.0 Números**

**Comentários em python começam com “#”** se estendendo até o final da linha.

* presente no início ou final de linha.
* não pode está entre string literal.

**Exemplo**:



**Operadores usuais → +, -, \* e /**.

**Exemplo**:



**Nota: int (números inteiros) e float (números fracionários)**.

Divisões sempre retornam números flutuantes (float), para r**etornar inteiros usar \\** e para **retornar o resto usar %**.

**Exemplo**:



**Exemplo: cálculo de potências**.



**Exemplo: atribuindo um valor a uma variável**.



**Nota: Operadores com operandos de diferentes tipos convertem inteiros para pontos flutuantes**.

**Valor da última expressão exibida (prompt)** fica armazenada em “\_”.

**Suporte de números** → int, float, **Decimal** e **Fraction**.

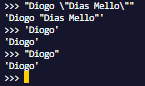
* além de suporte a números complexos.

### **3.1 Strings**

Sequências de caracteres (Strings):

* podem ser delimitadas por **aspas simples ‘ … ’ ou duplas “ … ”**. Usar “ \ ” para escapar aspas.
* permite acesso de caracteres por índice assim como arrays.

**Exemplo**:



**Exemplo: usando aspas duplas dentro de aspas simples**.



**O inverso também serve**.

**Barra invertida ( \ )** também pode ser usada com caracteres especiais como \n ou \t.

**Exemplo**:

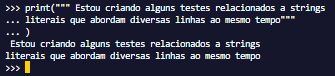


**função print produzindo saída mais legível**.

**OBS: Adicionar “ r ” (STRINGS RAW) antes de primeira aspas para que caracteres precedidos de \ não sejam considerados caracteres especiais**.

* Não pode terminar com números ímpares.

**Exemplo: adicionando strings literais de várias linhas**.



**Exemplo: concatenando e repetindo strings**.



**Utilizando operador “ \* ” e “ + ”, o primeiro repete as strings e o segundo concatena**.

**Nota: outra forma de concatenar as strings é colocando uma ao lado da outra entre aspas**.

Exemplo:

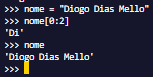


**Só funciona com strings literais**.

**Fatiamento de strings** - permite a obtenção de substrings.

* **O primeiro índice** representa por padrão é zero (onde começa) e o **segundo índice** é o tamanho da string fatiada (onde terminar).

**Exemplo**:

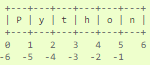


**Pega valores do índice 0 até o índice 2**.



**Pega valores do índice 1 até 4**.

**Ideia de fatiamento**:



**Exemplo: colocando limite acima do comprimento**.



**o resultado será a string completa**.

**OBS: strings em python são imutáveis, ou seja, não se pode atribuir novos valores a posições indexadas**.

* Uma opção seria criar uma nova string.

**função embutida len()** → devolve o comprimento de uma string.

**Exemplo**:

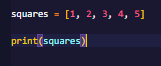


### **3.2 Listas**

**Lista de valores (itens) separados por vírgula**, entre colchetes **“ [ ] ”**.

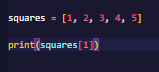
* estrutura de dados composta (agrupa valores).

**Exemplo**:

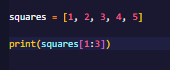


* podem ser **indexadas e fatiadas**.

**Exemplo: acessando listas de forma indexada e fatiados**.

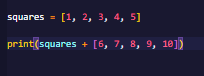


**Indexação**.



**fatiamento - devolve uma nova lista contendo os elementos solicitados**.

**Exemplo: concatenando listas**.



**OBS: listas são mutáveis, ou seja, todos os elementos podem ser modificados individualmente**.

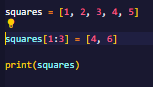
**método append()** - adiciona novos itens ao final da lista.

**Exemplo**:

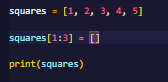


**atribuição de fatias** - pode alterar o tamanho da lista ou remover todos os itens dela.

**Exemplo**:



**Os valores atribuídos estão entre o índice 1 e 3**.



**valores removidos estão entre o índice 1 e 3**.

**Nota: função embutida len() também se aplica a listas**.

**aninhamento de lista** → listas que contém outras listas.

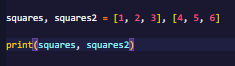
**Exemplo**:



**Atribuições múltiplas** - variáveis recebem simultaneamente os novos valores.

* Expressões avaliadas da esquerda pra direita.

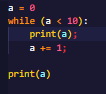
**Exemplo**:



**laço de REPETIÇÃO WHILE** - executa condição enquanto se permanecer verdadeira.

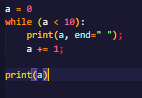
* qualquer valor inteiro que não seja 0 é **considerado verdadeiro**.

**Exemplo**:



**Nota: utilize end para evitar nova linha após saída ou finalizar a saída com uma string diferente**.

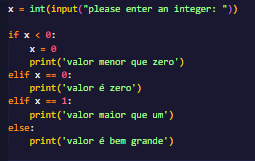
**Exemplo**:



## 4. Controle de fluxo

### **4.0 Comandos if**

**Exemplo: colocando um condicional**.

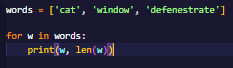


**elif** - abreviação para “else if” que evita indentação excessiva.

### **4.1 Comandos for**

**Itera sobre os itens de qualquer sequência (lista ou string)**, na ordem que aparecem na sequência.

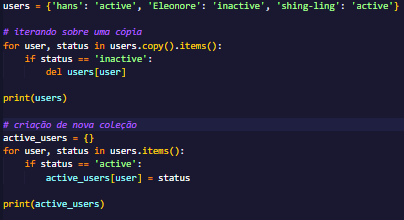
**Exemplo**:



**parâmetro w percorre cada valor de words**.

**OBS: Alteração de valores não é recomendado, então iterar sobre uma cópia ou uma nova coleção é mais recomendado**.

**Exemplo**:

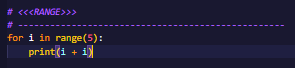


**os dois parâmetros após o laço de repetição for estão recebendo valores do dicionário por meio de atribuição multipla**.

### **4.2 função range()**

Itera sobre sequências numéricas gerando progressões aritmética.

**Exemplo**:



**Exemplo: iniciando com outro valor**.



**Exemplo: pulando valores**.



**aqui os valores estão pulando de 2 em 2**.

**Exemplo: iterando sobre os índices de uma sequência**.



**outro modo**,



**aqui o parâmetro i está iterando sobre o índice e valor de array**.

**OBS: range() retorna um objeto itens sucessivos de uma sequência**.

**Nota: objeto iterável é aquele que retorna sucessivamente seus elementos um de cada vez**.

### **4.3 Break, continue e cláusula else em laços de repetição**

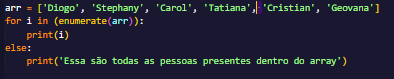
**Break** - instrução encerra loops como for ou while mais internamente.

**Else** - pode ser incluídas em loops como for e while.

* **while** - else executado depois que a condição se tornar falsa.
* **for** - else executado depois que loop atingir sua iteração final.

**OBS: se houver um break encerrando um loop, a cláusula else não será executada**.

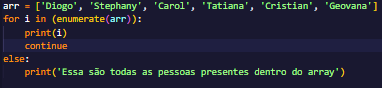
**Exemplo**:



**Else**.



**break**.

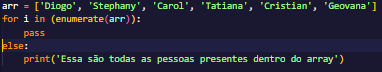


**continue - continua até a próxima iteração do laço**.

### **4.4 Comando pass**

Usada quando a sintaxe exige um comando, mas a semântica do programa não requer nenhuma ação.

**Exemplo**:



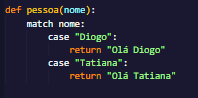
**Pode ser utilizado como um substituto temporário para uma função ou bloco de código condicional** quando se ainda está trabalhando em um novo.

### **4.5 Instrução match**

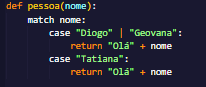
Pega uma expressão e **compara seu valor com padrões sucessivos fornecidos como um ou mais blocos de case**.

* permite extração de componentes (**elementos de sequência ou atributo de objetos**) de valor em variáveis.

**Exemplo**:

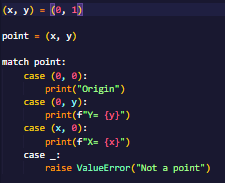


**Exemplo: combinando literais**.

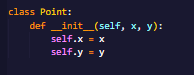


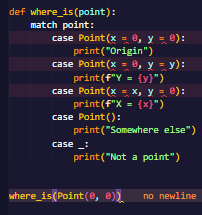
padrões podem se parecer com **atribuições de desempacotamento e podem ser usados para vincular variáveis**.

**Exemplo**:



**Exemplo: estruturando dados com classe**.



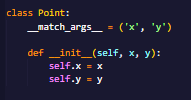


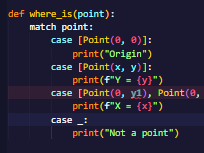
**case 1 - apresenta dois literais, case 2 e 3 - combinam literal e variável que vincula um valor do assunto point, case 4 - captura dois valores (semelhante à atribuição de desempacotamento)**.

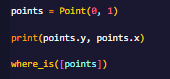
**OBS: apenas nomes autônomos são atribuídos por uma instrução de correspondência**.

**\_\_match\_args\_\_**: define posição específica para atributos em padrões.

**Exemplo: aninhando padrões arbitrariamente**.

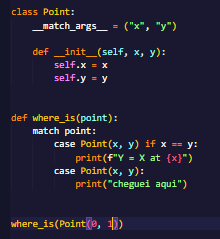
****





**Cláusula if no match** - cria condição que caso seja falsa, match continua até o próximo bloco de case.

**Exemplo**:



**case 1 só será executado se x for igual a y**.

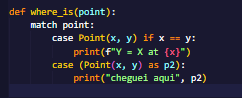
**Nota: padrão de sequência têm suporte ao desempacotamento estendido → [ x, y, \*rest ] / ( x, y, \*rest )**.

* nome após “ \* ” pode ser “ \_ ”.

**Padrões de mapeamento** → { chave: valor } captura os valores chave de um dicionário.

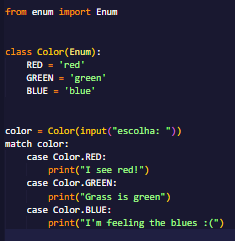
**subpadrões** - podem ser capturados usando a palavra reservada “ as ”.

**Exemplo**:



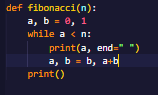
**OBS: singletons True, False e None são comparados por identidade**.

**Exemplo: usando constantes nomeadas como padrões**.



### **4.6 Definindo funções**

**exemplo: definindo uma função**.



**função definida por def irá rodar uma laço de repetição while enquanto “a” for menor que “n”**.

**def** - inicia a definição de função.

* deve ser seguida do nome da função.
* lista de parâmetros devem ficar entre parênteses.

**Nota: a primeira linha pode ser uma literal string com propósito de documentar a função (docstring)**.

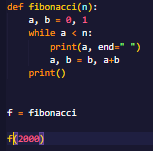
Execução de função cria **nova tabela de símbolos para variáveis locais da função**.

* As atribuições de variáveis ficam armazenadas na tabela.
* **Quando uma função chama a si mesma ou outra**, uma nova tabela é criada.

**referências de variáveis (busca)**: tabela de símbolos locais **→** tabela de símbolos de funções delimitadoras ou circundantes **→** tabela de símbolos globais **→** tabela de nomes.

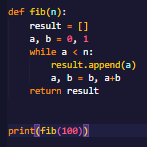
**argumentos passados por valor** → referência ao objeto, não o valor do objeto.

**Exemplo: acessando a função por outro nome**.



**OBS: Retorno None (nome embutido) - ocorre quando funções não utilizam return**.

**Exemplo: retornando uma lista de números de Fibonacci**.

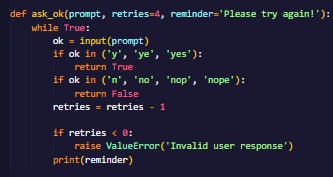


**Extra**:

* **return** - finaliza a execução e retorna um valor da função.
  + se não houver expressão como argumento retorna None.
* **método** - função que pertence ao objeto e é chamada pela utilização de pontos.
  + **append()** - usado em objetos do tipo lista para adicionar um novo elemento ao final da lista.

### **4.7 Definindo funções - Parte 2**

**Exemplo: Argumentos com valor padrão**.

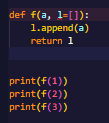


**nessa função pode ser retornado apenas valores obrigatórios ou todos os argumentos**.

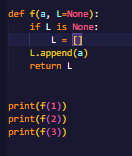
* **in** - verifica se uma sequência contém ou não um determinado valor.
* **valores padrões são avaliados no momento da definição da função** e no escopo onde a mesma foi definida.

**OBS: valores padrões são avaliados apenas uma vez**.

**Exemplo: acumulando valor padrão**.

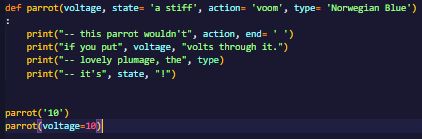


**outro modo para evitar acúmulo de valores padrões**.



**valor padrão sempre será none para evitar acúmulo de valor**.

**Exemplo argumentos nomeados**.



**forma chave=valor**.

* argumentos nomeados devem vir depois de argumentos posicionais.

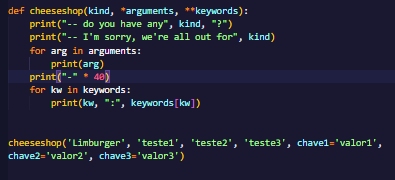
**Exemplo**:



**Parâmetros formais**:

* **“ \*\*nome ”** - recebe um dicionário contendo todos os argumentos nomeados.
* **“ \*nome ”** - recebe uma tupla contendo os elementos posicionais, além de lista de parâmetros formais.

**Exemplo**:



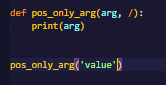
**O primeiro argumento está recebendo um valor de argumento posicional, o segundo parâmetro recebe uma trupla de valores posicionais e o terceiro parâmetro recebo argumentos nomeados**.

**símbolos “ / ” e “ \* ”** - indicam o tipo de parâmetro pela qual os argumentos poderão ser passados para as funções: **somente posicional**, **posicional ou nomeado** e **somente nomeado**.

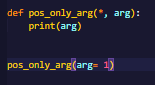
* definição: **def func( pos1, pos2, /, pos\_or\_kw, \*, kwd1, kwd2 )**.

**OBS: se os parâmetros forem somente posicional, a ordem importa e os mesmo não podem ser passados por nome**.

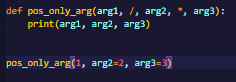
**Exemplo: somente posicional**.



**Exemplo: nomeado**.

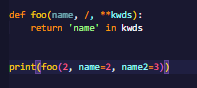


**Exemplo: combinado**.



**Definição de função com potencial de colisão** - possui argumento posicional como chave de um argumento nomeado.

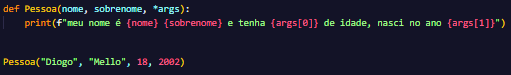
**Exemplo**:



**Nota: use somente posicional para evitar quebras na mudança de API**.

**argumentos arbitrários** - chamado com número arbitrário de argumentos.

* argumentos empacotados em tuplas.

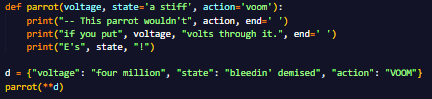


**todos os valores passados depois de \*args são parâmetros nomeados que podem ser usados como chave-valor**.

**Exemplo: desempacotando elementos**.



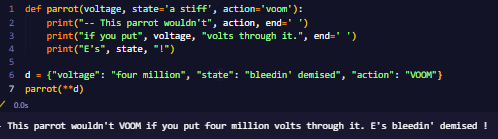
**obtém todos os valores de presentes em args como uma lista**.



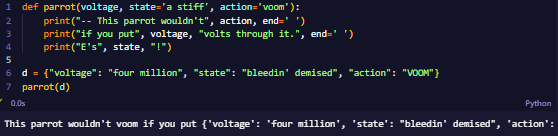
**retorna valores de um dicionário**.

**OBS: se não utilizar “ \* ” ou “ \*\* ” como feito nos exemplos acima, será passado apenas uma lista e um dicionário por completo para funções**.

**Exemplo**:



**antes**.

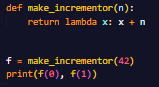


**depois**.

**lambda** - define funções anônimas.

* restritos é uma única expressão.

**Exemplo**:



**valor passado a função maker\_incrementor representa o parâmetro n e valor passado a f representa o valor de lambda**.

**regras para docStrings**:

* **linha 1**: resumo conciso do propósito do objeto sem se referir ao nome ou tipo do objeto.
* **linha 2**: em branco.
* **demais linhas**: descrever convenções de chamada ao objeto, seus efeitos colaterais.

**regras de escritas PEP8**:

* usar indentação com 4 sem tabulação.
* não criar linhas extensas.
* deixar linhas em brancos para separar funções, classes e blocos de código.
* escrever strings de documentação.

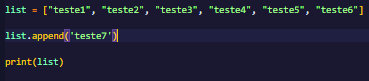
## 5. Estrutura de dados

### **5.0 Listas**

**Métodos mais utilizados**:

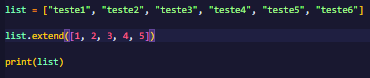
* **append(x)** - adiciona um item ao fim da lista.

**Exemplo**:



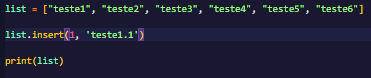
* **extend(iterável)** - adiciona no fim todos os elementos do argumento itarável passado como parâmetro.

**Exemplo**:



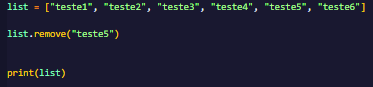
* **insert(índice, elemento)** - insere um item em uma dada posição.

**Exemplo**:



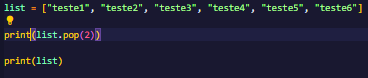
* **remove(x)** - remove item cujo valor é igual a x.

**Exemplo**:



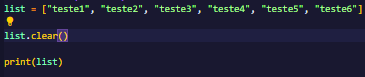
* **pop([i)** - remove item em uma dada posição da lista e o retorna.
  + se nenhum índice for específico, remove o último elemento.

**Exemplo**:



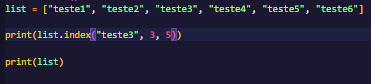
* **clear()** - remove todos os itens de uma lista.

**Exemplo**:



* **index(x, [, start[, end] ])** - devolve o índice base-zero do primeiro item cujo valor é igual a x, levantando ValueError se este valor não existe.
  + start e end - interpretados como notações de fatiamento.

**Exemplo**:



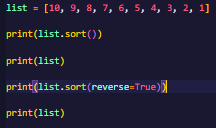
* **count(x)** - devolve o número de vezes em que x aparece na lista.

**Exemplo**:



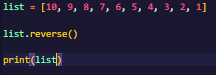
* **sort(\*, key=None, reverse=False)** - ordena os itens na lista (argumentos podem ser usados para personalizar a ordenação).

**Exemplo**:



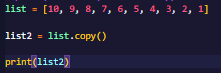
* **reverse()** - inverte a ordem dos elementos na lista.

**Exemplo**:



* **copy()** - devolve cópia rasa da lista.

**Exemplo**:



**Nota: retorno None (padrão) - princípio de design para estruturas de dados**.

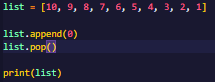
* nem todos os tipos de dados podem ser comparados ou ordenados, **ex: [ none, ‘hello’ , 10]**.

### **5.1 Pilhas**

**Funcionamento de pilhas** - item adicionado por último é o primeiro a ser recuperado.

* **append()** - adiciona ao topo.
* **pop()** - recupera item do topo da pilha.

**Exemplo**:

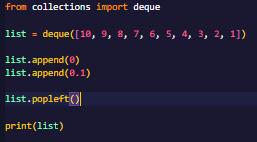


### **5.2 Filas**

**Funcionamento de listas** - primeiro item adicionado é o primeiro a ser recuperado.

* **NÃO RECOMENDADO USAR - inserts ou pops no início podem ser mais lentos**.
* **class collections.deque** - permite utilização de métodos de forma mais eficiente.

**Exemplo**:



### **5.3 List Comprehensions**

Criação de novas listas onde cada elemento é **resultado de uma operação aplicada a cada elemento de outra sequência ou iterável**, assim como a criação de subsequência de elementos que satisfaçam uma certa condição.

**Exemplo**:



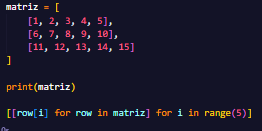
**cria uma lista de truplas onde cada uma terá o valor de x diferente de y**.



**cria uma lista normal de valores que são maiores que 20**.

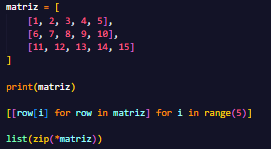
comprehensions podem conter expressões complexas e funções aninhadas.

**Exemplo**:



**a comprehension mais interna lê as linhas e a mais externa lê as colunas da matriz**.

**outro modo utilizando a função zip**,

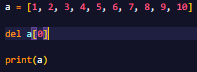


### **5.4 Instrução del**

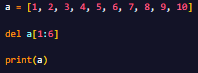
**remove um item de uma lista por meio do índice** no lugar do valor.

* usada para remover fatias de uma lista ou limpar a lista inteira.

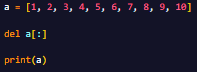
**Exemplo**:



**deleta apenas o valor 1**.

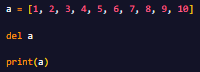


**fatia a lista do índice 1 ao 6**.



**deleta todos os valores da lista**.

**Exemplo: excluindo totalmente uma variável**.

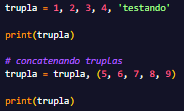


**gera um erro NameError**.

### **5.5 Tuplas e sequências**

Sequência de valores separados por vírgula.

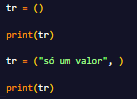
**Exemplo: definindo uma trupla**.



* tuplas podem ser facilmente aninhadas quando estiverem entre parênteses.
* **na criação de tuplas podem ou não ser envolvidas por parênteses**.
* elementos podem ser acessados via desempacotamento ou índice.

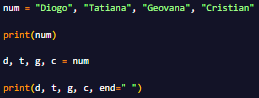
**OBS: truplas são imutáveis ao contrário das listas**.

**Exemplo: criando truplas com 0 ou 1 valor**.



**Se houver apenas um valor e a vírgula não for colocada, será interpretada como string**.

**Exemplo: sequência de desempacotamento**.



**variáveis estão pegando valores da tupla**.

### **5.6 Conjuntos**

**set** - tipo de dados para conjuntos.

* **conjunto** - coleção desordenada de elementos, sem elementos repetidos.
* usado para **verificação de existência de objetos e a eliminação de itens duplicados**.

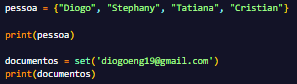
**Operações matemáticas suportadas**:

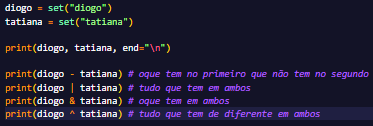
* **união**.
* **interseção**.
* **diferença**.
* **diferença simétrica**.

**“ { } ” ou função set()** - usados para criar conjuntos.

* **para criar um conjunto vazio**, usar set() ao invés de { }. A segunda pode criar um dicionário vazio.

**Exemplo**:





**Nota: da mesma forma que as listas a compreensão de conjuntos também são suportados**.

### **5.7 Dicionários**

**dict** - estrutura de dados embutida chamada de dicionário (**memória associativa / vetor associativo**).

* **indexados por chaves (keys)** - pode ser string / int.
* conjunto não ordenado de pares chave:valor (**chaves são únicas em dada instância**).

**Nota: tuplas também podem ser chaves se contiverem apenas strings, inteiros ou outras tuplas**.

* se contiver valor mutável, não poderá ser chave.

**OBS: lista não podem ser usados como chaves, pois podem ser facilmente mutáveis**.

**funções de um dicionário** - armazenar e recuperar valores a partir de chaves.

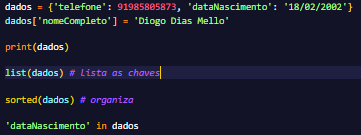
* valores podem ser modificados se adicionando um novo valor a uma chave existente.
* uma chave pode ser delatada utilizando **del**.

**list(d)** - devolve a lista de todas as chaves presentes no dicionário, na ordem de inserção.

**sorted(d)** - ordena uma lista.

**Operador in** - verifica a existência de uma chave.

**Exemplo**:



**Exemplo: utilizando o construtor dict**.





**Nota: comprehensions também podem ser usadas em dicionários**.

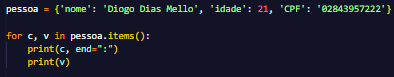
**Exemplo**:



### **5.8 Iteração**

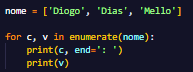
**método items()** - obtém chave e valor simultaneamente.

**Exemplo: iterando sobre um dicionário**.



**função enumerate()** - obtém posição e o valor (**itera sobre sequências**).

**Exemplo**:



**zip()** - percorre duas ou mais sequências ao mesmo tempo com entradas pareadas.

**Exemplo**:



**percorrendo sobre dois valores**.

**reversed()** - percorre uma sequência em ordem inversa por meio de uma entrada com a ordem original.

**Exemplo**:



**função sorted()** - percorrer sequência de maneira ordenada, retornando uma lista ordenada.

* mantém sequência original inalterada.

**Exemplo**:



**OBS: set() elimina elementos duplicados em uma sequência**.

**Nota: sempre que desejar alterar elementos de uma lista, é mais viável criar uma nova lista para receber os novos valores**.

### **5.9 Condições - Parte 2**

**Condições usadas em while e if podem conter qualquer operador**, não apenas comparação.

* **in e not in** - realiza teste de inclusão determinando a presença de um valor em um conteiner.
* **is e is not** - compara os tipos, exemplo: compara se dois objetos são realmente os mesmo.

**OBS: operadores de comparação tem menos prioridades que operadores numéricos**.

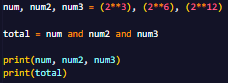
**Nota: comparações podem ser encadeadas**.

* encadeamento pode ser realizado com **and** e **or**.
* resultado de uma **comparação pode ter valor negado com not**.
* **possui menor prioridade**.

**operadores de curto-circuito** - argumentos avaliados da esquerda para direita, e avaliação encerrada quando resultado é determinado.

* valores genéricos - avaliação retorna último valor avaliado.

**Exemplo**:



### **5.10 Comparando sequências**

Objetos de sequência podem ser comparados com objetos de sequência, **desde que o tipo das sequências seja o mesmo**.

* Os itens serão comparados da esquerda para direita, **se encontrar valores diferentes o resultado será determinado**.
* **Comparação em strings**: utiliza codificação Unicode.

**Exemplo**:



**compara valores de tuplas**.

**Nota: comparar objetos diferentes funciona desde que os objetos possuam os métodos de comparação apropriado**.

## 6. Módulos

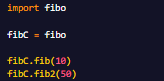
**Nota: uma boa prática é dividir um programa em arquivos menores para facilitar a manutenção**.

* criar um script separado para uma função que fosse utilizada para vários programas.

**módulo** - coloca definições em um arquivo e então utiliza em um script ou em uma execução interativa do interpretador.

* **módulo principal** - coleção de variáveis que dá acesso a um script executado como um programa.
* arquivo contendo definições e instruções python.
* **\_\_name\_\_** → possui o nome do módulo.

**Exemplo: importando módulo**.



**adiciona o nome do módulo que permite acessar as funções**.



**retornando o nome do módulo**.

Módulos podem conter **instruções executáveis, definições de funções e classes**.

**nomes privados** - espaço de nome global para todas as funções definidas no módulo.

* variáveis globais do módulo não irão conflitar com variáveis globais do usuário.

**Nota: se um módulo for importado em outro, suas definições serão adicionadas ao espaço de nomes global do módulo**.

**from** - importa definições de módulo diretamente para o espaço de nomes do módulo importador.

**Exemplo**:



**Exemplo: importando todos os nomes definidos em um módulo**.



**importa todas as declarações de nomes, exceto aqueles que começam com sublinhado “ \_ ”**.

* **prática não aprovada**.

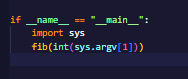
**Exemplo: dando um apelido ao nome**.



****

**importlib.reload(moduleName)** - realiza um teste de módulo interativamente.

**Exemplo: executando como arquivo principal**.



**código que analisa a linha de comando só roda se módulo for o principal**.

**Caminhos de busca de módulos**:

1. **sys.builtin\_module\_name** - lista módulos embutidos
2. **sys.path** - lista de diretórios.
   * **locais**: diretório atual, variável de ambiente PYTHONPATH (lista de nomes de diretórios) e o padrão.

**OBS: diretórios que contém links simbólicos não são adicionados ao caminho de busca de módulos**.

**Erros após inicialização de programas**:

* **diretório de script de execução é colocado no início de caminhos**, ficando a frente da biblioteca padrão.
* módulos de script serão carregados no lugar de módulos da biblioteca padrão.

### **6.0 Arquivos py compilados**

**\_\_pycache\_\_** - guarda versões compiladas de cada módulo.

* versão: formato do arquivo compilado.

**Conversões de módulos** - permite coexistência de módulos compilados de diferentes releases e versão de python.

* realização de verificação de data onde ocorreu a modificação do arquivo fonte mediante a versão compilada.
* **módulos compilados independente da plataforma**.

**Circunstância onde não ocorre verificação**:

1. módulos carregados diretamente da linha de comando - **recompila e não armazena**.
2. existência de módulo fonte - **sem verificações**.

**Nota: para corrigir o problema acima o módulo deve está no diretório de fonte, sem que haja um módulo fonte**.

**Informações extras**:

* **compileall** - cria arquivos pyc para todos os módulos de um diretório.

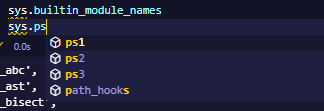
### **6.1 Módulos padrões**

Alguns dos **módulos estão embutidos no interpretador** para possibilitar acesso a operações que não são parte do núcleo da linguagem.

* garantem eficiência e acesso às chamadas de sistema.

**sys.ps1 / ps2** - definem as strings utilizadas como prompt primário e secundário.

**Exemplo**:



**argumentos que irão definir as entradas, códigos em execução e saídas**.

* somente definidas se interpretador estiver em modo interativo.

**sys.path** - lista de strings que determina os caminhos de busca de módulos conhecidos pelo interpretador.

* normalmente inicializada para um caminho padrão (**determinado por PYTHONPATH**).

**Exemplo: modificando o caminho padrão**.

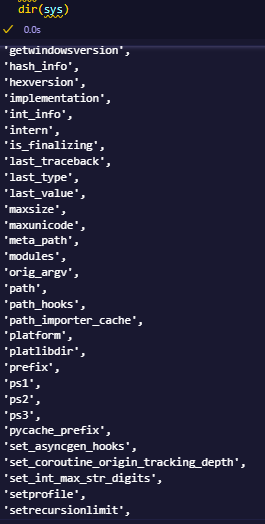


### **6.2 Função dir()**

Usada para descobrir quais nomes são definidos por um módulo.

* **devolve lista ordenada de strings**.

**Exemplo**:

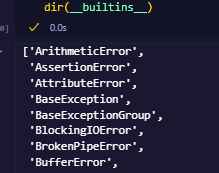


**retorna os nomes definidos pelo módulo sys**.

**Nota: sem argumentos dir() lista os nomes atualmente definidos**.

**OBS: dir() não lista variáveis e funções embutidas, somente o módulo builtins disponibiliza elas**.

**Exemplo**:



### **6.3 Pacotes**

**Forma de estruturação do namespace do módulo python** usando nomes de módulos pontilhados.

* **exemplo**: A,B, **significa que um módulo A possui um submódulo B**.

**Nota: \_\_init\_\_ - necessário para tratamento de diretórios contendo o arquivo como pacote**.

* pode ser utilizado para inicialização de pacotes.

**Exemplo: importação de submódulos**.

 - **importanto o submódulo echo**.



 - **importa uma variável a partir de uma submódulo**.

* variáveis ou funções também podem ser importadas do submódulo.

**from pacote Import item -** sempre teste se um item importado está definido no pacote, senão assume que é um módulo e tenta carregá-lo.

**import item.subitem.subsubitem** - todos devem ser um pacote com exceção do último.

* neste caso o último nunca poderá ser uma classe, função ou variável contida em um módulo.

### **6.4 Importando \* de um pacote**

Realiza vasculhamento do sistema de arquivos para encontrar todos os módulos presentes no pacote para importação.

**Nota: se arquivo \_\_init\_\_.py definir lista chamada \_\_all\_\_, está indicará o nome dos módulos a serem importados**.

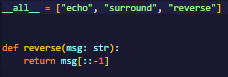
Exemplo:



**ao realizar uma importação completa, somente estes submódulos serão importados**.

**Módulos podem ficar ocultos por nomes definidos localmente**, ou seja, ao realizar a importação somente os dois módulos dos módulos citados acima seriam importados.

**Exemplo**:



**submódulo reverse está sendo obscurecido pela função**.

**OBS: se all não estiver definido, o código apenas irá garantir que o pacote foi importado e depois importará quaisquer nomes definidos no pacote**.

**Nota: uma boa prática de programação é a importação “from pacote import submodulo\_especifico”**.

**Importação relativa** - utilizam pontos iniciais para indicar os pacotes atuais e pai envolvidos na importação relativa.

**Exemplo**:



**imports relativos são baseados no nome do módulo atual**.

**\_\_path\_\_** - lista que contém o nome do diretório onde está o arquivo \_\_init\_\_.py do pacote.

## 7. Entrada e Saída

### **7.0 Refinando a formatação de saída**

**Write()** - método de objetos de arquivos.

**Exemplo**:



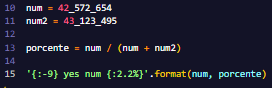
**Strings literais formatadas** - normalmente escritas com f antes da abertura das aspas simples ou duplas, este tipo de string aceita expressões python entre { … }.

**Exemplo**:



**str.format** - têm a mesma idéia da forma utilizada no exemplo acima, mas inclui diretivas de formatação detalhadas.

**Exemplo**:

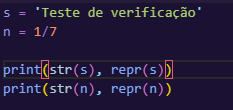


**Funções repr() e str()** - podem ser usadas para exibir variáveis com propósito de depuração.

* **str()** - retorna representação de valores que sejam legíveis para pessoas.
* **repr()** - retorna representação de valores que o interpretador python consegue ler.

**Nota: números e estruturas podem apresentar as mesmas representações, agora quando falamos de strings, podemos obter dois formatos distintos**.

**Exemplo**:



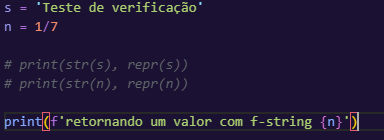
**Nota: o módulo string fornece uma classe Tamplate que oferece outra maneira de substituir valores em strings, por meio de espaços reservados $x**.

* valores substituídos por valores de um dicionário.
* menor controle de formatação.

#### **7.0.0 Strings Literais Formatadas**

Também conhecidas como f-strings, permite que se inclua um valor de expressão python dentro de uma string.

**Exemplo**:

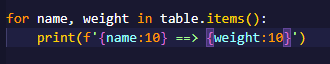


**Exemplo: colocando especificador de formato**.



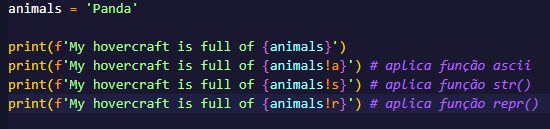
**a sintaxe é uma expressão seguida de :.qtdCasasDecimais**.

**Exemplo: colocando número mínimo de caracteres de largura**.



**importante passar um inteiro após os : para determinar a largura**.

**Exemplo: aplicando outros modificadores disponíveis**.



**O especificador =** pode ser usado para expandir uma expressão para o texto da expressão.

**Exemplo**:



#### **7.0.1 Método Format()**

**Exemplo: aplicação do método**.



**chaves e conteúdos são substituídos por objetos passados para o método str.format**.

* número de chaves referencia a posição do objeto.

Argumentos nomeados terão seus valores referenciados usando o nome do argumento.

**Exemplo**:



**Exemplo: passando argumentos posicionais**.



**Se a string de formatação for grande**, faça uma referência para seus valores a serem formatados por nome, em vez de posição.

* passe um dicionário usando colchetes para acessar as chaves.

**Exemplo**:



**Exemplo: passando como argumento nomeado**.

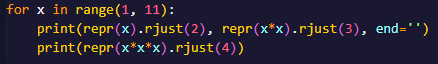


#### **7.0.2 Formatação Manual de Strings**

**Método rjust()** - justifica uma string à direita, num campo de tamanho definido, acrescentando espaços à esquerda.

* **métodos similares**: ljus() - justifica à esquerda; e center() - centraliza.
* apenas retorna uma nova string.

**Exemplo**:



**OBS: o método acima não funciona para strings muito longas**.

**Método zfill()** - preenche string numérica com zeros à esquerda.

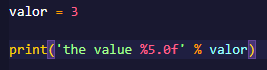
* sabe lidar com sinais positivos e negativos.

**Exemplo**:



#### **7.0.3 Interpolação de strings**

**Exemplo**:



**Sintaxe: ‘String’ % valores**.

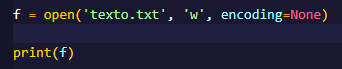
* cada instância de %, é substituída por zero ou mais elementos de valores.

### **7.1 Leitura e escrita de arquivos**

**open()** - retorna um objeto de arquivos.

* utilizado com dois argumentos posicionais e um argumento.
* **sintaxe: open(filename, mode, enconding=none)**.

**Exemplo**:



**O primeiro argumento é o nome do arquivo e o segundo é o modo como o arquivo será usado**.

* **modos de uso**:
  + r - reading.
  + r+ - abre o arquivo para leitura e escrita.
  + w - writing.
  + a - addition (qualquer escrita será adicionada ao final do arquivo).

**OBS: se a codificação não for especificado em encoding, o padrão irá depender da plataforma**.

* Na maioria dos casos, o UTF-8 é o mais usado.

**Exemplo: abrindo o arquivo em modo binário**.



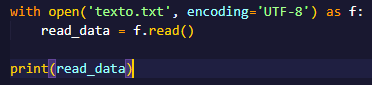
**Não pode especificar o encoding quando abrir o arquivo neste modo**.

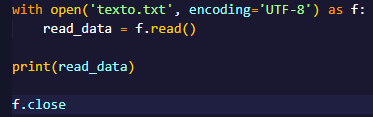
**OBS: a modificação de terminadores de linha da plataforma para apenas \n, e retornar para terminadores de linha, pode corromper os dados binários**.

**palavra-chave with** - usada para lidar com arquivos.

* O arquivo é fechado corretamente após o término de sua utilização, mesmo que uma exceção seja levantada.

**Exemplo**:





**f.close - fecha o arquivo e imediatamente libera qualquer recurso do sistema usado por ele**.

* depois de fechado, o arquivo não poderá ser mais aberto.

#### **7.1.0 Métodos de objetos arquivo**

**read(tamanho)** - lê um punhado de dados devolvendo-os como uma string ou bytes.

* tamanho: argumento numérico opcional, se omitido ou negativo, todo conteúdo do arquivo é lido e devolvido.

**método readline()** - lê uma única linha do arquivo.

* \n é mantido ao final da string e omitido na última linha do arquivo.
* elimina ambiguidade do valor retornado.

**Exemplo**:



**Exemplo: interando sobre um arquivo**.



**list(arquivo)** - ler todos as linhas de um arquivo em uma lista.

* uma segunda opção é arquivo.readlines().

**arquivo.write(string)** - escreve o conteúdo de string para o arquivo, retornando o número de caracteres escritos.

**Exemplo**:

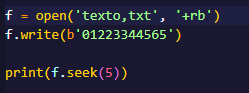


**tell()** - retorna inteiro dando a posição atual do objeto arquivo, no arquivo representado, como número de bytes desde o início do arquivo.

**seek(offset, de\_onde)** - muda a posição.

* nova posição computada pela soma do deslocamento offset a um ponto de referência especificado pelo argumento de\_onde.

**Exemplo**:



**OBS: arquivos de texto abertos sem um b, somente seeks relativos ao início do arquivo serão permitidos**.

* exceção: indicado ao final do arquivo com seek(0, 2).

**Nota: único valor válido para offset são aqueles retornados por chamada à tell()**.

#### **7.1.1 Gravando dados estruturados com json**

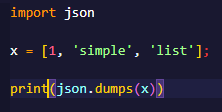
**JSON (Javascript Object Notation)** - salva tipos de dados mais complexos, como listas e dicionários aninhados.

**Serialização** - obtém hierarquias de dados em python e converte em representações de strings

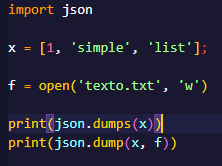
**Desserialização** - reconstrói os dados estruturados da representação string.

String que representa o objeto pode ser armazenada em uma string ou estrutura de dados, enviada por uma conexão de rede.

**Exemplo**:



**retorna uma representação JSON**.



**serializa o objeto para arquivo de texto**.

**json.load()** - decodifica objeto novamente.

**Exemplo**:



**Nota: utilize enconding=”utf-8” quando abrir um arquivo JSON como um arquivo de texto tanto para leitura quanto para escrita**.

## 8. Erros e Exceções

### **8.0 Erros de sintaxe**

**Conhecidos como erros de parse**, este erro apresenta uma linha inválida e aponta para o ponto da linha em que o erro foi detectado.

**Exemplo**:



**erro causado pelo símbolo que precede a seta**.

### **8.1 Exceções**

Erros detectados durante a execução são chamados de exceções e não são necessariamente fatais.

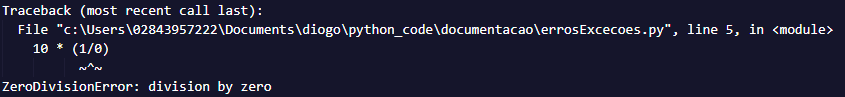
**OBS: caso uma exceção não seja tratada pelo programa, podem acabar resultando em mensagens de erro**.

**Identificadores embutidos** - considerados exceções padrões dentro de python.

**Stack traceback** - mensagem de erro apresenta contexto onde ocorreu a exceção.

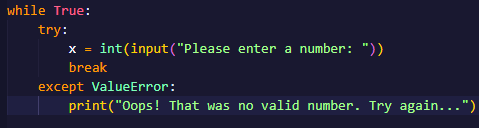
* parte anterior da mensagem de erro.

**Exemplo**:



### **8.2 Tratamento de Exceções**

**Exemplo**:



**requisita um número até que valor seja um inteiro**.

**Instrução try, funcionalidade**:

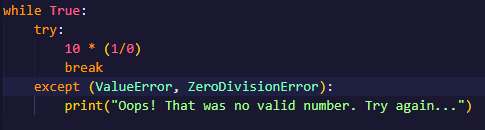
1. A cláusula try é executada até que ocorra uma exceção.
2. se ocorrer uma exceção e ela for prevista em uma except, esta cláusula será executada.
3. depois a execução continua após o bloco try/except.

**OBS: se exceção não corresponder a exceção listada dentro de except, então ela pode ser entrega ao try mais externo**.

* se não existir tratador mais externo, o programa termina e um erro é lançado.

**Nota: expressão try pode mais de um except para múltiplos tratadores, mas no máximo um único tratador será executado**.

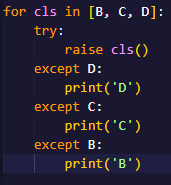
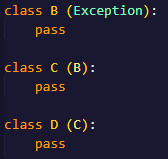
**Exemplo: colocando múltiplas exceções dentro de uma except**.



**exceções devem ser passadas dentro de uma tupla**.

**Nota: uma classe dentro uma cláusula except é compatível com uma exceção, se ela for da mesma classe ou de uma classe base desta**.

**Exemplo**:

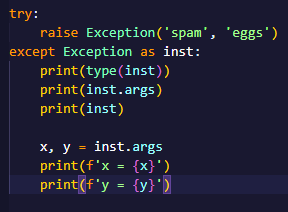


**OBS: uma exceção pode estar associada a valores chamados de argumentos da exceção**.

Uma variável vinculada à instância de exceção, normalmente possui um args que armazena os argumentos.

**\_\_str\_\_()** - pode exibir todos os argumentos sem acessar explicitamente .args.

**Exemplo**:



**inst obtém todos os argumentos de Exception e fica instanciado na exceção**.

**BaseException** - classe base comum de todas as exceções.

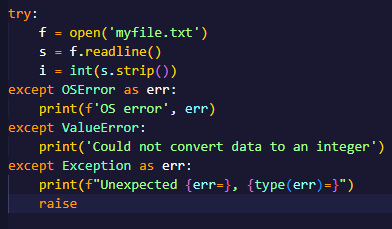
* **subclasses**:
  + Exception - possui todas as exceções não fatais.

**SystemExit e KeyboardInterrupt**, indicam que o programa deve terminar.

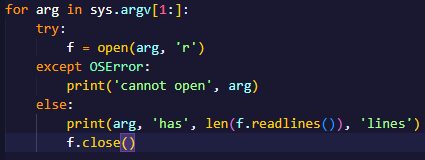
* A primeira é levantada por sys.exit.
* a segunda é usada quando o usuário deseja interromper o programa.

**Nota: Exception pode ser usado como coringa, que irá capturar de tudo**.

**Exemplo**:



**Exemplo: inserindo cláusula else**.



**normalmente essa forma é usada quando nenhuma exceção for ativada**.

* evita tratamento de exceções indevidamente.

**OBS: os manipuladores não só tratam exceções que ocorrem imediatamente na cláusula try, mas também aquelas que ocorrem dentro de funções na cláusula**.

### **8.3 Levantando exceções**

**Instrução raise** - força uma ocorrência de um determinado tipo de exceção.

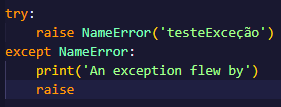
**Exemplo**:



* nameError é um argumento de raise que deve ser uma instância de exceção ou uma classe de exceção.

**Nota: se uma classe de exceção for passada, ela será implicitamente instanciada invocando seu construtor sem argumentos**.

**Exemplo: determinando se exceção foi levantada ou não**.

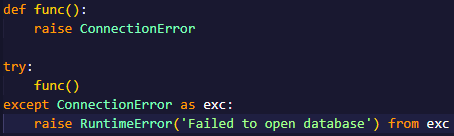


**O último raise levanta novamente a exceção lançada**.

### **8.4 Encadeamento de exceções**

**from** - cláusula opcional de raise, permite indicar quando uma exceção é consequência de outra.

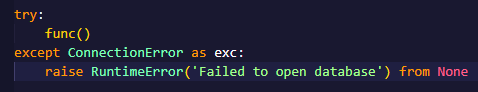
**Exemplo**:



**útil para a transformação de exceções**.

**from none** - desabilita encadeamento automático de exceções.

**Exemplo**:



### **8.5 Exceções definidas pelo usuário**

Programas podem definir novos tipos de exceções, através da criação de uma nova classe.

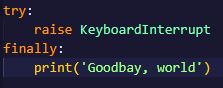
**OBS: novas exceções devem ser definidas com nomes terminados em Error**.

### **8.6 Definindo Ações de limpeza**

**Finally** - cláusula opcional de try que tem a finalidade de permitir a implementação de ações de limpeza.

* executadas independente da ocorrência de exceções.

**Exemplo**:

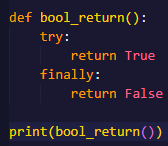


**a cláusula opcional é executada como última tarefa antes da conclusão da instrução try**.

**Casos complexos na existência de finally**:

1. uma exceção pode ser levantada novamente após a execução da cláusula finally.
2. se existir break, continue ou return, as exceções não serão levantadas.
3. finally sempre irá executar antes de break, continue ou return.
4. se finally possuir return, o valor retornado será de finally.

**Exemplo**:

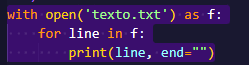


**Nota: a utilização de finally é indicada para liberação de recursos externos, independente do uso do recurso ter sido bem sucedido ou não**.

### **8.7 Ações de limpeza predefinidas**

Objetos podem definir ações de limpeza padrões para serem executados quando o objeto não é mais necessário.

**Exemplo**:

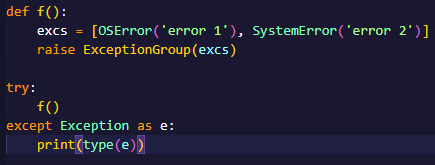


**with permite que objetos sejam utilizados com a certeza de que sempre serão prontamente e corretamente finalizados**.

### **8.8 Criando e tratando várias exceções não relacionadas**

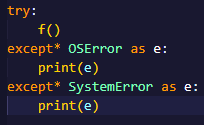
**ExceptionGroup** - envolve uma lista de instâncias de exceções para que elas possam ser levantadas juntas.

**Exemplo**:



**except\*** - usada para manipular seletivamente apenas as exceções no grupo que corresponde a um determinado tipo.

**Exemplo**:



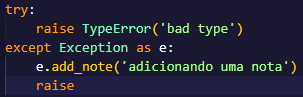
**OBS: grupo de exceções devem ser instâncias, não tipos**.

### **8.9 Enriquecendo exceções com notas**

**add note(note)** - aceita uma string e adiciona à lista de notas de exceção.

* adição de informação após a captura de exceções.

**Exemplo**:



## 9. Classes

**Forma de organizar dados e funcionalidades juntos**.

Cria uma nova classe, gera um novo tipo de objeto, permitindo que novas instâncias desse tipo sejam produzidas.

* Instâncias de classes possuem métodos para modificar seu estado.

**Nota: as classes são criadas em tempo de execução e podem ser alteradas após a sua criação**.

**Membros da classe**, incluindo os seus dados, são públicos. E todas as suas **funções membro** são virtuais.

### **9.0 Escopo e espaços de nomes do python**

**Espaço de nomes** - mapeamento que associa nomes a objetos.

* implementadas como dicionários em python.

**Tipos de espaços de nome**:

1. nomes pré-definidos.
2. nomes globais em um módulo.
3. nomes locais na invocação de função.

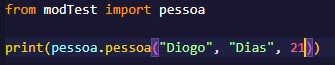
**OBS: não existe relação entre nomes em espaços de nomes distintos**.

Referências para nomes em módulos são atributos, por exemplo: **moduloNome.atributoNome**.

**Tipos de atributos**:

1. somente leitura.
2. leitura e escrita - possibilita a atribuição de novos valores ao atributo.

**Exemplo**:



**OBS: atributos que só aceitam escrita podem ser apagados com del**.

**nomes embutidos** são criados quando o interpretador inicializa e nunca serão removidos.

**nomes globais** de um módulo são criados quando a definição do módulo é lida.

* duram até a terminação do interpretador.

**\_\_main\_\_** - possui comando executados pela invocação do interpretador.

* script principal.
* comandos possuem seu próprio espaço de nomes.

**nomes locais de uma função** é criado quando a função é invocada, e apagado quando a função retorna ou levanta uma exceção não tratada.

* cada invocação recursiva tem seu próprio espaço de nomes.

**escopo** - região textual de um programa python onde um espaço de nomes é diretamente acessível.

* usados dinamicamente.

**nonlocal** - altera variáveis declaradas fora do escopo mais interno.

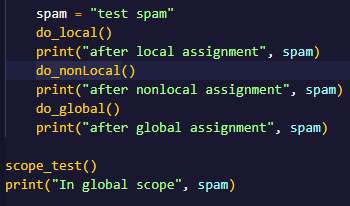
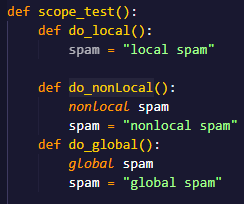
* indica que variáveis particulares estão em um escopo mais interno e devem ser recuperadas lá.

**O escopo global de uma função** definida em um módulo é o espaço de nomes deste módulo, sem se importar de onde ou por qual apelido a função é invocada.

**Nota: se nenhuma instrução global ou nonlocal estiver em vigor, as atribuições de nomes sempre entram no escopo mais interno**.

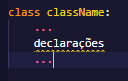
**Instrução global** - indica quais variáveis residem no escopo global ao invés do local.

**Exemplo**:



### **9.1 Sintaxe da definição de classe**

**Exemplo: Forma básica**.



**OBS: as definições de classes assim como de funções, devem ser executadas antes que tenham qualquer efeito**.

**Nota: Definições dentro de classes geralmente são de funções**.

* funções = métodos.
* forma peculiar de lista de argumentos.

Ao criar uma nova classe, um novo espaço de nome é criado e usado como escopo local.

* variáveis locais irão ocorrer dentro do escopo da classe.
* funções são vinculadas aos nomes nesse escopo.

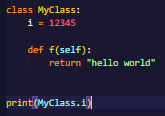
### **9.2 Objeto Classe**

**Tipos de operação**:

1. referências a atributos.
2. instanciação.

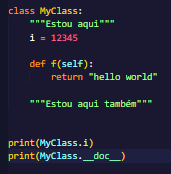
**Nota: nomes de atributos válidos, são todos os nomes presentes dentro do espaço de nomes da classe**.

**Exemplo: realizando referências a atributos válidos**.

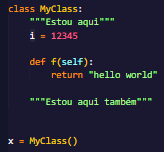


**\_\_doc\_\_** - atributo válido da classe que retorna um docstring associado à classe.

**Exemplo**:



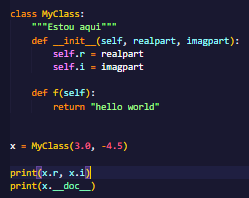
**Exemplo: instânciando uma classe**.



**\_\_init\_\_** - cria um objeto com um estado inicial predeterminado.

* processo de instanciação irá invocar automaticamente init sobre a instância recém criada.

**Exemplo**:



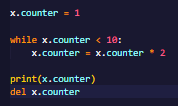
**instância criada deve ser inicializada com os valores de realpart e imagpart**.

### **9.3 Objetos Instância**

**Maneiras de validação de atributos**:

* **dados** - variáveis de instância em smaltalk e a membros de dados em C++.
  + não necessitam de declarações.
* **métodos** - função que pertence ao objeto instância.

**Exemplo**:

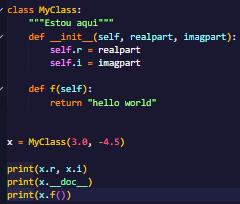


**Nota: instancia.m acessa um objeto método, e a class.m acessa um objeto função**.

* considerando que m representa uma função.

### **9.4 Objetos método**

**Exemplo: método de referenciado imediatamente**.



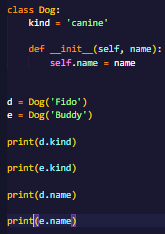
**self - passa o objeto da instância como primeiro argumento da função**.

### **9.5 Variáveis de classe e instância**

**Variáveis de instância** - indicam dados que são únicos a cada instância individual.

**Variáveis de classe** - atributos e métodos que são comuns a todas as instâncias de uma classe.

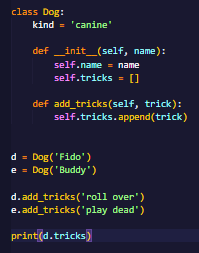
**Exemplo**:



**como kind foi definido dentro da classe, este será igual para todas as instâncias da classe**.

**OBS: dados compartilhados podem causar efeitos inesperados quando envolvem objetos mutáveis**.

**Exemplo**:



**a variável foi feita para ser apenas da instância, caso contrário os dados seriam compartilhados para todas as instâncias indevidamente**.

### **9.6 Observações aleatórias**

**OBS: se um mesmo nome de atributo ocorre tanto na instância quanto na classe, a busca por atributo prioriza a instância**.

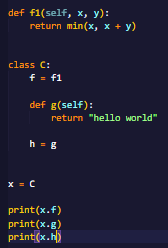
**Nota: atributos de dados podem ser referenciados por métodos da própria instância, bem como qualquer outro usuário**.

* classes não servem para implementar tipos puramente abstratos de dados.

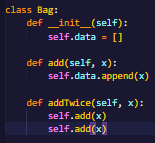
**self** - representa o primeiro argumento de um método.

* não é obrigatório, mas é uma convenção que deve ser seguida para aprimoramento do código.

**Exemplo: atribuindo um objeto função a uma variável local da classe**.



**Exemplo: invocando métodos por meio do atributo de método do argumento self**.



**nessa classe é possível observar que um método dentro da classe Bag está sendo invocado por outro método**.

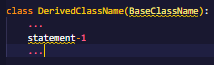
**Nota: métodos podem referenciar nomes globais da mesma forma que funções comuns**.

* o escopo global associado a um método é o módulo contendo a sua definição na classe.

**object.\_\_class\_\_** - possui as classes e seus valores.

### **9.7 Herança**

**Exemplo: sintaxe de uma classe derivada**.



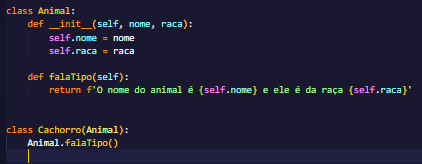
**caso a classe esteja definida em outro módulo**.



**Precedência de busca**: classe derivada > classe base.

**Classes derivadas podem sobrescrever métodos das suas classes bases**, ou seja, um método da classe base pode acabar invocando um método sobreposto por uma classe derivada.

**Exemplo**:

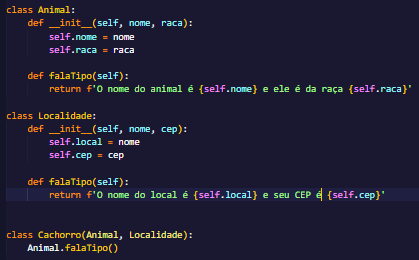


**Funções embutidas que trabalham com herança**:

1. **isinstance()** - verifica o tipo de uma instância.
   1. isinstance(obs, int) - somente será true se obj.\_\_class\_\_ for int ou classe derivada de int.
2. **issubclass()** - verifica a herança entre as classes.
   1. issubclass(bool, int) - true se bool é uma subclasse de int.

### **9.8 Herança múltipla**

**Exemplo: sintaxe de uma classe com múltiplas bases**.



Ordem de resolução de métodos muda dinamicamente para suportar chamadas cooperativas para super().

**Ordenação dinâmica** - importante para que todos os casos de herança múltipla exibem um ou mais relacionamentos de diamante.

* assim as classes não são acessadas mais de uma vez.

### **9.9 Variáveis Privadas**

**OBS: variáveis privadas não existem em python**.

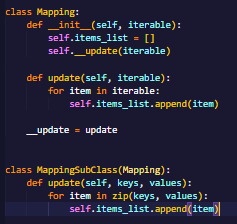
* isso pode ser substituído por um nome prefixado com um sublinhado, exemplo: \_spam.

**Todo identificador no formato \_\_spam** (Nota: pode ter dois sublinhados no início e um no final) é textualmente substituido por \_classname\_\_spam.

* classname representa a classe atual.

**Desfiguração de nomes** - usada para que subclasses possam sobrescrever métodos sem quebra invocações de métodos dentro de outra classe.

**Exemplo**:



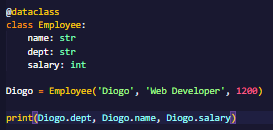
**Mesmo que a subclasse inserisse um identificador de mesmo nome, o exemplo iria funcionar visto que ele seria substituido por \_classPai\_\_itentificador e \_classFilho\_\_identificador**.

### **9.10 Curiosidades e conclusões**

**dataclasses** - agrupa itens de dados.

* semelhante ao record de Pascal ou ao struct de C.

**Exemplo**:



**trecho de código espera um tipo de dado abstrato em particular**.

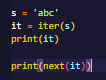
**Nota: métodos de instância também pode ter atributos, como por exemplo: m.\_\_self\_\_ (objeto instância com o método) e m.\_\_func\_\_ (objeto função correspondente ao método)**.

### **9.11 Iteradores**

**Funcionamento do for**:

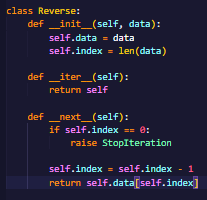
1. for chama iter() no objeto contêiner > função retorna método iterador.
2. define o método next() > acessa elementos no contêiner, um de cada vez.
3. sem elementos next() > levanta uma exceção StopIteration.
4. fim do for.

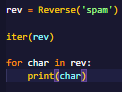
**Exemplo: chamando o método next()**.



**a cada nova chamada de next um elemento é acessado**.

**Exemplo: definindo um método \_\_iter\_\_() que retorna um objeto que tenha um método \_\_next\_\_()**.





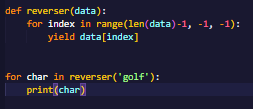
### **9.12 Geradores**

**Ferramentas usadas na criação de iteradores**, eles são escritos como funções normais mas usam a instrução yield quando precisam retornar dados.

**Nota: neste modo, cada vez que next() é chamado, o gerador volta ao ponto onde parou**.

* valores de dados e instrução executada são lembrados.

**Exemplo**:



**neste modo, os métodos \_\_iter\_\_() e \_\_next\_\_() são criados automaticamente, assim como StopIteration**.

Variáveis locais e o estado da execução são preservados automaticamente entre as chamadas.

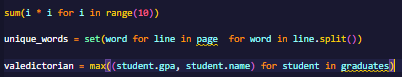
* forma mais fácil de implementação.

### **9.13 Expressões Geradoras**

Projetadas para situações em que o gerador é usado imediatamente pela função que o engloba.

* mais compactas do que definições completas de geradores.
* utilizam menas memória.

**Exemplo**:



**formato: método(expressão lacoRepetição)**.