# CADERNO PYTHON – MARCOS VALENTE

SUMÁRIO

[CADERNO PYTHON – MARCOS VALENTE 1](#_Toc121560746)

[SUMÁRIO 1](#_Toc121560747)

[O QUE É PYTHON 3](#_Toc121560748)

[CONHECIMENTOS BÁSICOS SOBRE FUNÇÕES 4](#_Toc121560749)

[PRINTING 4](#_Toc121560750)

[FORMATAÇÕES DE TEXTO 5](#_Toc121560751)

[LOCALIZAR TEXTOS EM STRINGS 8](#_Toc121560752)

[PRINTAR PARTES SELECIONADAS DA STRING (INDEXING) 9](#_Toc121560753)

[ARMAZENAR STRINGS EM VARIÁVEIS: 11](#_Toc121560754)

[DADOS E VARIÁVEIS 12](#_Toc121560755)

[ORDEM DE PREFERÊNCIA DE VARIÁVEIS 12](#_Toc121560756)

[TIPOS DE DADOS 14](#_Toc121560757)

[REGRAS PARA NOMEAÇÃO DE VARIÁVEIS 14](#_Toc121560758)

[TIPOS DE VARIÁVEIS 14](#_Toc121560759)

[VARIÁVEIS NUMÉRICAS 14](#_Toc121560760)

[VARIÁVEIS STRING 16](#_Toc121560761)

[LISTAS 19](#_Toc121560762)

[DICTIONARIES 28](#_Toc121560763)

[TUPLAS/ TUPLES 32](#_Toc121560764)

[SETS 33](#_Toc121560765)

[BOOLEANS 34](#_Toc121560766)

[OBSERVAÇÕES 35](#_Toc121560767)

[OPERAÇÕES MATEMÁTICAS SIMPLES 36](#_Toc121560768)

[PRECEDÊNCIA DE OPERAÇÕES 37](#_Toc121560769)

[SEQUÊNCIAS 37](#_Toc121560770)

[TIPOS DE SEQUÊNCIAS 37](#_Toc121560771)

[PROGRAM FLOW 38](#_Toc121560772)

[INPUT VARIÁVEIS 38](#_Toc121560773)

[IF BLOCKS/ CONDITIONS 39](#_Toc121560774)

[IF STATEMENTS 41](#_Toc121560775)

[FOR LOOPS 42](#_Toc121560776)

[WHILE LOOPS 49](#_Toc121560777)

[BREAK, CONTINUE, PASS 50](#_Toc121560778)

[OPERADORES RELEVANTES PARA PROGRAM FLOW 52](#_Toc121560779)

[ERROS 57](#_Toc121560780)

[TRY, EXCEPT, FINALLY 57](#_Toc121560781)

[FUNÇÕES 58](#_Toc121560782)

[BASIC FUNCTIONS 60](#_Toc121560783)

[FUNCTIONS WITH LOGICS 62](#_Toc121560784)

[TUPLE UNPACKING WITH FUNCTION CALL 65](#_Toc121560785)

[\*ARGS / \*\*KWARGS 65](#_Toc121560786)

[MAP & FILTER 67](#_Toc121560787)

[LAMBDA EXPRESSIONS (ANONYMOUS FUNCTIONS) 68](#_Toc121560788)

[PROGRAMAÇÃO ORIENTADA A OBJETOS (POO) 70](#_Toc121560789)

[SINTAXE BÁSICA 70](#_Toc121560790)

[BÁSICO SOBRE ATRIBUTOS 71](#_Toc121560791)

[BÁSICO SOBRE MÉTODOS 74](#_Toc121560792)

[MÉTODOS QUE AFETAM ATRIBUTOS 76](#_Toc121560793)

[INHERITANCE/ POLIMORFISMO 76](#_Toc121560794)

[MÉTODOS ESPECIAIS 79](#_Toc121560795)

[MÓDULOS E BIBLIOTECAS (MODULES AND PACKAGES) 80](#_Toc121560796)

[CHAMANDO MÓDULOS 80](#_Toc121560797)

[CRIANDO BIBLIOTECAS/PACOTES PRÓPRIOS 81](#_Toc121560798)

[\_\_name\_\_ / ‘\_\_main\_\_’ 82](#_Toc121560799)

[TESTE DE CÓDIGOS 83](#_Toc121560800)

[PYLINT 83](#_Toc121560801)

[UNITTEST 83](#_Toc121560802)

# O QUE É PYTHON

* Linguagem interpretativa orientada a objetos
* Funciona em diferentes sistemas operacionais como Windows, Mac OS X, Linux e outros
* Suporta tipos de dados como números e strings, bem como tipos mais complexos como listas e dicionários (o que pode simplificar o processamento de dados)
* Suporta diversos paradigmas de programação (procedural programming, function programming e programação orientada a objetos)

# CONHECIMENTOS BÁSICOS SOBRE FUNÇÕES

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

# PRINTING

* “Print” é uma função
* Dentro dela, entre parênteses, deve constar uma “string”

Exemplos:

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

* Repare que se produziu uma linha “vazia” quando se deu o comando “print()”

É possível que uma função apresente mais de um argumento:



Nesse caso, o terminal apresentará:



Observação: se quiser utilizar aspas duplas dentro de um texto, a função print deverá ser acompanhada de um parêntese com aspas simples. Exemplo:



* Uma forma de escapar a esse problema é inserir uma “\” antes da citação. Exemplo:





* Outra forma é utilizar três aspas duplas para delimitar a string



* + Repare que deve haver um espaço entre a última aspas dupla da citação e as três aspas que fecham a string

## FORMATAÇÕES DE TEXTO

* Utilize “\n” para quebrar linha de texto



Texto

Descrição gerada automaticamente

* + Observação: também é possível quebrar linhas utilizando-se de três aspas duplas e, no campo de código, digitando como se digitaria, por exemplo, no bloco de notas
  + Exemplo:

Texto

Descrição gerada automaticamente

Tela preta com letras brancas

Descrição gerada automaticamente

* \t para que o texto seja apresentado após o espaço de um “tab”

Texto

Descrição gerada automaticamente



* Atributo “end =”
  + Por padrão, ao final do comando print, o Python entende que existe um ‘\n’
  + Se, dentro do print, for colocado após vírgula o atributo ‘end =’, o programador poderá escolher o que acontecerá ao final
    - É possível, por exemplo, ao invés de pular uma linha, inserir um espaço utilizando-se “end = ‘ ‘” (dessa forma, o próximo print virá logo em seguida, não na linha de baixo)
    - Outro exemplo: “end = ‘.’” Insere um ponto ao final do print

Interface gráfica do usuário

Descrição gerada automaticamente com confiança baixa

ESCAPE CHARACTERS

* Para inserir o texto “cru”, sem que o Python interprete comandos dentro do texto, basta inserir “r” (de “raw text”) antes do início do texto



* + Note que, se isso não fosse feito, o Python interpretaria “\u”, “\t” e “\n” como comandos de texto
* É possível utilizar \ (o backslash é um “escape character”) para cancelar certos comandos





**Apresentar string em caixa alta**

Texto

Descrição gerada automaticamente com confiança baixa

* Observação: a string será apresentada em caixa alta, mas não será modificada permanentemente

*Transformar a variável string em caixa alta*

*Interface gráfica do usuário, Texto

Descrição gerada automaticamente*

**Apresentar string em caixa baixa**

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

**Remoção de espaços em branco no início e no fim da string**

**Texto

Descrição gerada automaticamente com confiança baixa**

**Alterar um conteúdo dentro de uma string**

**Texto

Descrição gerada automaticamente**

* ‘.replace(“*elemento\_substituível”*, “*elemento­\_substituto*”)’
* Observação: essa função (como as demais) não substitui o valor da variável
* Exemplo 2:

**Texto

Descrição gerada automaticamente**

## LOCALIZAR TEXTOS EM STRINGS

* Para localizar substrings dentro de strings, utiliza-se o comando “in”, o qual retorna True ou False

Interface gráfica do usuário, Texto

Descrição gerada automaticamente com confiança média

## PRINTAR PARTES SELECIONADAS DA STRING (INDEXING)

* De agora em diante, para efeitos didáticos, printar equivale a “escrever” em Portugol
  + Leia-se mostrar na tela

Texto

Descrição gerada automaticamente

* É possível printar **caracteres específicos** de uma string, como no exemplo a seguir:

Texto

Descrição gerada automaticamente

* + Pelo último comando, o terminal printará a letra “w”
  + Pode gerar confusão, pois essa é a 4ª e não a 3ª letra da string. Acontece que, para o Python, a contagem se inicia em zero
    - Se pedisse para ele printar o caractere zero da string, ele printaria “N”
* Negative indexing (caracteres contados de trás para frente)
  + Observação: nesse caso, assume-se que o último valor equivale a -1 (não a -0, pois esse número não existe)
  + Exemplo:

Texto

Descrição gerada automaticamenteTexto

Descrição gerada automaticamente

* Printar sequências de caracteres (slicing):



* + Repare que nesse caso, ele printa até a posição 5 (não inclui a posição 6)
  + O último caractere que você indica não está incluído no slicing (“up to but not including”)
  + Outro exemplo:



* + Caso o slicing se inicie no primeiro caractere, não é necessário indicar a primeira posição. Exemplo:

Texto

Descrição gerada automaticamente

* + Semelhantemente, se o slicing terminar no último caractere, não é necessário determinar a posição. Exemplo:

Tela de celular

Descrição gerada automaticamente com confiança média

* Slicing com negative indexes. Exemplo:

Gráfico

Descrição gerada automaticamente

* Printar slices com steps (passos)
  + Exemplo:



* + Explicação:

Calendário

Descrição gerada automaticamente

* + - O corte começa na posição zero
    - E se estende até a posição 6 (mas não a inclui, o que é padrão no Python)
    - Extrai os caracteres, nessa faixa, em passos de 2

Calendário

Descrição gerada automaticamente

* + Exemplo 2:

Texto

Descrição gerada automaticamente

* + Repare que, nesse caso, o programa mostrará somente as vírgulas
    - O valor do meio não precisa ser mostrado, pois presume-se que se trata do texto completo
* Slicing backwards

Interface gráfica do usuário, Texto

Descrição gerada automaticamente



* + Observação: se o valor limite for zero, será printado somente até a letra “b” (“a” será omitido)
  + Como o passo é negativo, é possível omitir o valor inicial, dessa forma:

Uma imagem contendo Texto

Descrição gerada automaticamente

* RESUMO:
  + *String*[posição\_inicial:posição\_final:passo]

## ARMAZENAR STRINGS EM VARIÁVEIS:

Interface gráfica do usuário, Texto

Descrição gerada automaticamente



* Como nós criamos a variável “greeting” atrelando a ela um valor de string, o Python decidiu que essa variável é do tipo string, de modo que ela só pode ser utilizada em contextos relacionados a strings
* É possível utilizar a **função “input”** para que o usuário forneça um valor para a variável

Interface gráfica do usuário

Descrição gerada automaticamente



* Nota: eu escrevi “Marcos Valente” no próprio terminal

# DADOS E VARIÁVEIS

## ORDEM DE PREFERÊNCIA DE VARIÁVEIS

Texto

Descrição gerada automaticamente

* Importante: variáveis declaradas dentro de uma função só possuem escopo dentro daquela função!

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

* Repare que, mesmo tendo a função transformado o valor de X, quando ele a variável é chamada pelo comando print, na última linha, ela utiliza o valor definido fora da função!
* É necessário utilizar o comando “return” para manipular a variável fora da função em que foi apresentada (e fora da função designar uma nova variável que coletará o resultado retornado)
* Também é possível **designar, dentro da função, a variável como global:**



* + Ao designar a variável X como global, o python permitiu que ela fosse transformada pela função

## TIPOS DE DADOS

Tabela

Descrição gerada automaticamente

## REGRAS PARA NOMEAÇÃO DE VARIÁVEIS

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

## TIPOS DE VARIÁVEIS

Existem grandes tipos de variáveis em Python:

* Numeric
* Interator
* Sequence (que também são Interators)
* Mapping
* File
* Class
* Exception

## VARIÁVEIS NUMÉRICAS

* São três os tipos de variáveis numéricas:
  + **Int** (números inteiros)
    - Não há limite para o tamanho dos valores int que o Python pode guardar (o que ocorre nas outras linguagens)
  + **Float** (números que possuem parte fracionada (equivalente aos números reais em Portugol)
  + **Complex** (números complexos)

**Formatando a apresentação das variáveis numéricas**

* Dentro de um replacement field é possível formatar a apresentação de um número
* Define-se o número de espaços que deverá ocupar da seguinte maneira: {0:2}
  + Dessa forma, o replacement field zero deverá ocupar dois espaços (dois dígitos)
* Para alinhar o valor à esquerda: {0:<3}
  + Dessa forma, o replacement field zero deverá ocupar três espaços e estar alinhado à esquerda
* Para alinhar o valor à direita: {0:>3}
* Para alinhar o valor ao centro: {0:^3}

Exemplo:



Tabela

Descrição gerada automaticamente

* Observação importante: se o número tiver mais dígitos do que lhe foi designado, o Python mostrará todos os dígitos. Isso porque **essa formatação só diz respeito ao espaço dele no print se o tamanho do número for menor do que o tamanho designado (é uma questão de alinhamento somente)**
* Para determinar o número de casas decimais (para números float), utilizar o formato o comando **round**(*variável***,** *número\_casas\_decimais*)
  + Observação: esse método irá arredondar o número
  + O Python consegue mostrar com precisão até, aproximadamente, 50 casas decimais
* Outro método para determinar o número de casas decimais é utilizar ‘:.xf’ ao lado da variável (nesse caso, ‘x’ é o número de casas decimais)
  + Exemplo: média:.2f

## VARIÁVEIS STRING

**Transformar outras variáveis em string**

* É possível outros valores em string utilizando-se do comando “str(*variável*)”

**Uma imagem contendo Esquemático

Descrição gerada automaticamente**

****

* É possível alcançar o mesmo resultado sem converter a variável para string
* Para isso, deve-se utilizar um replacement field, ou {0}, e depois indicar a formatação da variável



**Replacement field**

* Replacement fields são substituídos pelos valores que aparecem dentro de “.format()”
  + Método de chamada





* Observação: é possível repetir os replacement fields em uma string:





* É possível valer-se de replacement fields para printar o resultado de operações matemáticas
  + Pode-se apresentar qualquer expressão dentro do parênteses do .format



* É possível, e em muitos casos desejável, indicar no replacement field o que será “substituído”
  + Para isso, basta no .format() indicar variáveis:

Texto

Descrição gerada automaticamente

**f string**

* É possível utilizar uma *f string* para formatar variáveis que estejam dentro de uma string
* Para isso, basta inserir um f antes da string e colocar as variáveis não string dentro de chaves

Exemplo:

Interface gráfica do usuário

Descrição gerada automaticamente com confiança média



**Verificar se trechos estão contidos na string**

* Para verificar se, dentro da string, existe determinado trecho/slice, basta utilizar o comando “print(**“***trecho a ser conferido***”** **in** *string*). O programa printará na tela “true” se houver esse trecho contido ou “false”, se não tiver
  + Exemplo:

Texto

Descrição gerada automaticamente

**Contar número de vezes em que determinada substring está contida em uma string**

* Para isso, utiliza-se a função “.*count(‘substring’)”*

**Outras funções nativas do Python envolvendo strings:**

* .split()
  + Separa strings com base em um separador (posto dentro dos parênteses)
  + Transforma os elementos separados em lista

**Interface gráfica do usuário, Texto

Descrição gerada automaticamente**

* .isdigit()
  + Verifica se só existem dígitos dentro de uma string
  + Retorna True ou False
* .index()
  + Retorna a posição na string (index) em que a substring se encontra
  + Retornará um erro se não encontrar
* .find()
  + Retorna a posição na string (index) em que a substring se encontra
  + Diferença entre .find() e .index() é que o primeiro não retorna erro se não encontrar a substring (retornará ‘-1’)

**Interface gráfica do usuário, Texto

Descrição gerada automaticamente com confiança média**

* + Repare que ele só retorna a posição da primeira ocorrência (há mais de uma nessa frase)
* .capitalize()
  + Torna a primeira letra da string maiúscula
* **.**title()
  + Torna a primeira letra de cada palavra de uma string maiúscula

## LISTAS

Texto

Descrição gerada automaticamente

* Listas podem conter diferentes tipos de dados, como no exemplo a seguir:

Texto

Descrição gerada automaticamente

* Para saber o tamanho de uma lista, utilizar o comando **len**(*nome\_lista*)
  + Retornará quantos elementos estão contidos na lista

**Indexing e slicing em listas**

* Os mesmos princípios que se aplicam às strings se aplicam aqui (em caso de dúvida, ver parte do caderno sobre strings)

**Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente**

**Interface gráfica do usuário

Descrição gerada automaticamente com confiança média**

*.index*

* Comando utilizado para saber em que posição se encontra determinado elemento

**Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente**

* + Repare que o Python retornou que o valor 2 encontra-se na casa 1

.*count()*

* Comando utilizado para contar quantas vezes determinado elemento aparece em uma lista

**Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente**

**Concatenar listas**

**Texto

Descrição gerada automaticamente**

**Mudando valores de uma lista**

* Mudando valor individual:

**Texto

Descrição gerada automaticamente**

* Mudando elemento da lista para outro tipo
  + Nesse caso, um elemento string será transformado em um int

**Interface gráfica do usuário, Texto

Descrição gerada automaticamente**

* Adicionando novos elementos a uma lista (ao final)
  + Para isso, utiliza-se o comando ‘.append’

**Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente**

* Adicionando novos elementos a uma lista (em uma posição específica)
  + Para isso, utiliza-se o “*lista*.insert(*index*, variável)”

**Interface gráfica do usuário, Texto

Descrição gerada automaticamente**

****

* Adicionando múltiplos elementos a uma lista (adicionando uma lista a outra):

**Interface gráfica do usuário, Texto

Descrição gerada automaticamente**

* + Repare que a lista é agregada ao final da outra
* Retirar um elemento de uma lista (pelo index dele)

Tela de celular com fundo preto

Descrição gerada automaticamente

* + Repare que, dentro do () deve estar indicado qual elemento será retirado
  + Se nenhum elemento estiver indicado, o último será removido
* Retirar um elemento de uma lista (pelo valor dele)
  + Para isso, utiliza-se “*lista*.remove(*valor*)”

**Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente**

* Retirar todos os elementos de uma lista (sem excluí-la – passa a ser vazia)
  + Utilizar comando “*nome\_lista*.clear()”
  + Se quiser excluir a lista, utilizar o comando “del *nome\_lista*”

**Somar valores de uma lista**

* Para somar os valores de uma lista, utiliza-se o comando “sum(*nome\_lista*”)

Texto

Descrição gerada automaticamente

**‘Sort’ e ‘reverse’**

* Para se ordenar uma lista, utiliza-se o comando ‘.sort()’

**Texto

Descrição gerada automaticamente com confiança média**

* + **Observação**: esse comando altera a ordem da lista permanentemente
* Para tornar a lista reversa, deve-se utilizar o comando ‘.reverse()’

Texto

Descrição gerada automaticamente

* List comprehensions
  + É possível transformar uma sequência em elementos de uma lista de duas maneiras: utilizando “for loops” e “list comprehensios”
    - For loops:

Texto

Descrição gerada automaticamente

* + - * Observação: a variável de controle pode ter qualquer nome
    - List comprehensions:

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

* + - * Observação: a variável de controle pode ter qualquer nome
      * Outros exemplos:

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

* + É possível fazer operações com os valores antes de apresentá-los na lista
    - Exemplo: apresentação de números elevados ao quadrado a partir de um valor baseado em um range

Texto

Descrição gerada automaticamente

* + Transformar valores em lista se eles atenderem certo requisito
    - Exemplo: lista que apresenta valores de um range, desde que sejam pares

Texto

Descrição gerada automaticamente

**Transformar listas em string**

* Utilizar comando “ “.join(*nome\_lista*)
  + Resultado será uma **string** em que todos os valores da antiga lista serão apresentados separados por um espaço (caso queira separar por outro caractere, como traços, utilizar um “-“ antes do .join)

**Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente**

* + Observação: é possível unir os valores em uma string sem espaço ou elementos divisórios (basta utilizar “”.join(*nome\_lista*))

## DICIONÁRIOS

Texto

Descrição gerada automaticamente

Tela de celular com texto preto sobre fundo branco

Descrição gerada automaticamente com confiança média

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

**Exemplo de uso**

**Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente**

**Características**

* Dictionaries são bastante flexíveis e podem conter diferentes tipos de variáveis, inclusive listas e outros dicionários

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

**Adicionando valores a um dicionário**

* Basta atribuir uma nova chave, com seu respectivo valor, ao dicionário

**Interface gráfica do usuário

Descrição gerada automaticamente com confiança baixa**

* O mesmo método pode ser utilizado para reatribuir o valor de uma chave já existente

**Texto

Descrição gerada automaticamente com confiança média**

**Modificando chaves e valores de um dicionário**

* Para isso, utiliza-se “*nome\_dicionário*.update({*chave:* valor, *chave2:* valor2,...})”
  + Se a chave já existir, o valor é substituído
  + Se a chave não existir, é criado uma nova chave, com valor inserido no comando “.update”

**Texto

Descrição gerada automaticamente**

**Removendo itens de dicionários**

* Utilizando o método ‘*nome\_dicioário*.pop(*nome\_chave*)’

**Texto

Descrição gerada automaticamente**

* Utilizando o método ‘*nome\_dicionário*.popitem()’
  + Remove a última chave do dicionário, bem como o valor a ela vinculado

**Checar todas as chaves do dicionário**

* Utilizar o comando ‘.keys()’

**Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente**

**Checar todas os valores do dicionário**

* Utilizar o comando ‘.values()’

**Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

Descrição gerada automaticamente**

**Checar as tuplas de um dicionário**

**Texto

Descrição gerada automaticamente**

**Percorrendo variáveis do dicionário**

**Tela de celular com texto preto sobre fundo branco

Descrição gerada automaticamente**

* Repare que é necessário utilizar o método “.items()”

**Criando dicionários dentro de dicionários**

**Interface gráfica do usuário, Texto

Descrição gerada automaticamente**

**Copiar dicionários**

* Se atribuirmos um dicionário já existe a uma variável, não será criado um novo dicionário, mas sim uma referência
  + Por isso, se modificar o dicionário original, aquele vinculado à nova variável também será modificado
* Para contornar esse problema, deve-se utilizar o método ‘.copy()’

Tela de celular com texto branco sobre fundo preto

Descrição gerada automaticamente

## TUPLAS/ TUPLES

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

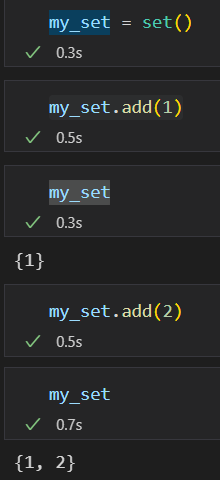
* Todos os comandos referentes a listas, com exceção daqueles que mudam valores, funcionam com tuplas

## SETS

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

* Adicionando valores a sets



* + Observação: não é possível adicionar o mesmo valor a um set

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

* Transformando listas em sets

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

* Observação: sets não seguem ordem

## BOOLEANS

Texto

Descrição gerada automaticamente

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

## OBSERVAÇÕES

Existe um comando para verificar o tipo da variável: “print(type(*nome da variável*))”

Exemplo:

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

**Observação 2:**

* Em Python, as variáveis não possuem tipos fixos
* É possível que uma variável INT se transforme em uma STR, como no exemplo a seguir:

**Texto

Descrição gerada automaticamenteTexto

Descrição gerada automaticamente**

* Nesse sentido, é mais usual em Python dizer que o VALOR possui o tipo X ou Y
  + Em C, JAVA, Portugol, entre outras, dir-se-ia que a variável é do tipo X ou Y e possui um valor

# OPERAÇÕES MATEMÁTICAS SIMPLES

a = 12

b= 3

Texto

Descrição gerada automaticamente

* Repare que, embora estejamos utilizando números inteiros, a divisão simples gera um valor do tipo float
  + Em algumas operações, misturar números inteiros e floats pode gerar erros
  + Nesses casos, é aconselhável utilizar a divisão por números inteiros (utilizando-se “//”)

**Comando para elevar um número a outro número:**

* N \*\* 2 -> “N” ao quadrado
* N \*\* 3 -> “N” ao cubo
* N \*\* 100 -> “N” elevado a 100
* Observação: Podemos também empregar o operador de exponenciação \*\* para calcular a raiz quadrada de um número, elevando o número em questão à potência de 1/2 (0.5):

## PRECEDÊNCIA DE OPERAÇÕES

Texto branco sobre fundo preto

Descrição gerada automaticamente

# SEQUÊNCIAS

Texto

Descrição gerada automaticamente

Padrão do plano de fundo

Descrição gerada automaticamente com confiança baixa

## TIPOS DE SEQUÊNCIAS

O Python possui 5 tipos de sequências:

* Sequências string
* List
* Tuple
* Range
* Bytes e Bytearray

# PROGRAM FLOW

## INPUT VARIÁVEIS

* Quando se utiliza o comando input() sem descrever a variável, o Python entende que se trata de uma string
* Para se adicionar outro tipo de variável, é necessário sinalizar isso da seguinte maneira:



* Repare que eu indiquei, na linha 2, antes do input() qual seria o tipo da variável

**Validando inserção de variáveis**

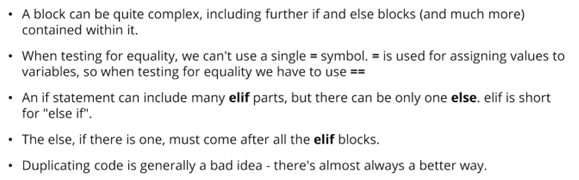
* Existem diversas formas de validar se o input corresponde ao tipo de dado que o programa espera
* Exemplo: obrigar usuário a inserir um dígito dentro de determinado intervalo:

**Texto

Descrição gerada automaticamente**

* + Observação: o método “.isdigit()” não checa intengers ou floats... ele checa strings compostas por dígitos! Por isso é importante converter a string em uma variável numérica.
  + Observação 2: esse método pode não funcionar caso o usuário escolha um valor negativo
* Outros métodos para verificar se foi inserido um número intenger e/ou float pondem ser consultados nestes links:
  + [Various Posts on This](https://www.google.com/search?q=python+check+if+input+is+number)
  + [StackOverflow Post 1](https://stackoverflow.com/questions/5424716/how-to-check-if-string-input-is-a-number)
  + [StackOverflow Post 2](https://stackoverflow.com/questions/1265665/how-can-i-check-if-a-string-represents-an-int-without-using-try-except)

## IF BLOCKS/ CONDITIONS



**Conditions de comparação**

Tabela

Descrição gerada automaticamente

* Observações:
  + Strings podem ser comparadas também (verificar se são iguais)
  + No entanto, se um número for apresentado em uma string (i.e. ‘3’), ele não poderá ser comparado a um número int ou float

**Conditions and/ or/ in**

* “and” e “or” são condições/ comandos semelhantes aos “e” e “ou” do Python (mesmas regras lógicas)
  + Se tiver dúvidas, ver caderno de Algoritmos
* Observação: sempre use parênteses quando utilizar “and” e “or” na mesma condição
  + Para não gerar confusão lógica

*and*

Tela preta com letras brancas

Descrição gerada automaticamente

* O mesmo programa poderia ser escrito de forma simplificada da seguinte maneira:

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

*or*

*Tela de celular com texto preto sobre fundo branco

Descrição gerada automaticamente com confiança média*

*In*

* Utilizada para verificar se um valor está contido em uma sequência
* O exemplo a seguir vale-se de uma string (a qual é uma sequência de caracteres):

Texto

Descrição gerada automaticamente

* Outro exemplo:

Texto

Descrição gerada automaticamente

**True/ False/ Not**

* True e False (ambos devem estar em c.A.) são valores lógicos que podem ser atribuídos a variáveis

*not*

* Torna verdadeiro condição que de outra forma seria falsa (e vice-versa)

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

Descrição gerada automaticamente

* Exemplo prático:

Texto

Descrição gerada automaticamente

## IF STATEMENTS

Texto

Descrição gerada automaticamente com confiança média

* Para que ocorra a endentação, a qual é obrigatória para que o programa funcione, é necessário que um “:” (dois pontos) siga o comando
* Em Python, “else” é o equivalente ao “senao” do Portugol

Texto

Descrição gerada automaticamente

**ELIF:**

**Texto

Descrição gerada automaticamente**

* Observação: em Python, o símbolo de igual deve ser escrito duas vezes

## FOR LOOPS

* Objetos que podem ser iterados: caracteres em strings, itens em listas, itens em tuplas, chaves em dicionários

**Sintaxe básica**

**Texto

Descrição gerada automaticamente**

**Ícone

Descrição gerada automaticamente**

* Repare que é necessário criar uma nova variável (“item\_name”) para representar os elementos dentro do elemento iterável (“my\_iterable”)

**Iterar listas**

Exemplo 1:

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Exemplo 2:

Tela de celular com fundo preto

Descrição gerada automaticamente

**Iterar intervalos (range)**

* Estrutura do comando range:

**Texto

Descrição gerada automaticamente**

* + Só é necessário indicar o valor de parada (observação: não incluirá o valor de parada)

**Texto

Descrição gerada automaticamente com confiança média**

* + - Repare que a contagem se iniciou em “0” e terminou em “9”
  + É possível indicar o valor de início e também o “passo”

**Texto

Descrição gerada automaticamente**

Texto

Descrição gerada automaticamente

* Transformando range em listas:

Texto

Descrição gerada automaticamente com confiança média

**Iterar strings**

**Texto

Descrição gerada automaticamente**

* Como visto, não é necessário atribuir à variável de controle o nome “letter” (pode ser qualquer um)
* Nesse caso, não é necessário associar a string a uma variável

**Tela de celular com texto preto sobre fundo branco

Descrição gerada automaticamente com confiança média**

**Iterar tuplas:**

* Exatamente igual às listas

**Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

Descrição gerada automaticamente**

*Tuplas dentro de listas*

* Repare como tuplas, dentro de listas, comportam-se como itens da lista:

Texto

Descrição gerada automaticamente

* É possível imprimir na tela os valores dessas tuplas, contidas em uma lista da seguinte forma:

Texto

Descrição gerada automaticamente

* + Esse método é conhecido como **Tuple unpacking**
  + Exemplo 2 de Tuple unpacking:

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

**Iterar dicionários**

* O procedimento padrão imprimirá somente as chaves

**Texto

Descrição gerada automaticamente**

* Para imprimir os itens do dicionário, é necessário chama-los utilizando “in ‘nome\_dicionário’.items():”

**Texto

Descrição gerada automaticamente**

* + Repare que o programa retornou tuplas compostas pela chave e o valor correspondente
    - Para receber somente o valor, é necessário utilizar a técnica de **tuple unpacking:**

**Texto

Descrição gerada automaticamente**

* + - É possível receber os valores diretamente utilizando-se “in ‘*nome\_dicionário*’.values()”

**Texto

Descrição gerada automaticamente**

* Observação: é necessário ter em mente que os dicionários não são ordenados, razão pela qual não há garantia de que os valores serão retornados na ordem correta

## WHILE LOOPS

* While loops continuarão a executar um bloco de código enquanto alguma condição continuar verdadeira (True)
* Sintaxe padrão:

Uma imagem contendo Texto

Descrição gerada automaticamente

* É possível combinar essa estrutura com “else”, caso queira que o código execute outro padrão caso a condição não seja verdadeira (desde o início ou torne-se depois False)

Texto

Descrição gerada automaticamente com confiança média

Exemplo 1:

Texto

Descrição gerada automaticamente

## BREAK, CONTINUE, PASS

**Texto

Descrição gerada automaticamente**

* O comando **pass** permite fazer com que o programa ignore uma endentação sem comandos, sem que acuse um erro
  + Útil enquanto se escreve o programa

**Texto

Descrição gerada automaticamente**

* O comando **continue** faz com que o programa pule uma instrução

**Texto

Descrição gerada automaticamente**

* + Leia-se: se a letra for “a”, volte para o comando que deu origem ao loop
* O comando **break** encerra um loop caso alguma condição for verificada

**Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

Descrição gerada automaticamente**

* + Utilizando o comando **break** com while loops:

**Interface gráfica do usuário, Texto

Descrição gerada automaticamente**

* + - Observação: x +=1 é a forma abreviada de escrever x = x + 1

**Nesting for loops**

* Inserir um for loop dentro de outro:

Texto

Descrição gerada automaticamente

## OPERADORES RELEVANTES PARA PROGRAM FLOW

**Enumerate**

* Retorna tuplas, indicando a posição dos itens
* Exemplo:

**Texto

Descrição gerada automaticamente**

**Zip**

* Une/ pareia duas ou mais listas em tuplas
* Exemplo:

**Texto

Descrição gerada automaticamente**

* Se a listas não contiverem o mesmo número de elementos, o comando zipará até esgotar a menor lista
* Exemplo:

**Texto

Descrição gerada automaticamente com confiança média**

* As tuplas podem ser apresentadas como listas da seguinte forma:

**Texto

Descrição gerada automaticamente**

**Utilizar operador “in” para ver se determinado elemento está contido em outro**

* Em listas:

**Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

Descrição gerada automaticamente**

**Interface gráfica do usuário, Texto, Aplicativo, Word

Descrição gerada automaticamente**

* Em strings:

**Interface gráfica do usuário, Texto

Descrição gerada automaticamente**

* Em dicionários:

**Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente**

* + Repare que foi procurado um item do dicionário
  + Para procurar valores, é preciso indicar que se procuram valores:

**Interface gráfica do usuário, Aplicativo

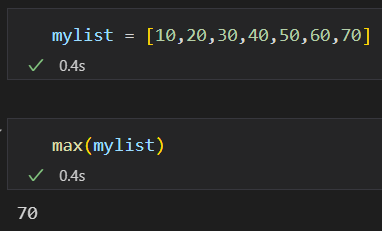
Descrição gerada automaticamente**

**Busca por valores mínimos (min)**

**Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente**

**Busca por valores máximos (max)**

****

**Random library**

* Embaralhando listas:

**Texto

Descrição gerada automaticamente**

* Importando números aleatórios (random intenger)

**Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente**

* + Repare que foi indicado o intervalo
  + Para salvar esse número, é preciso criar uma variável:

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

# ERROS

## TRY, EXCEPT, FINALLY

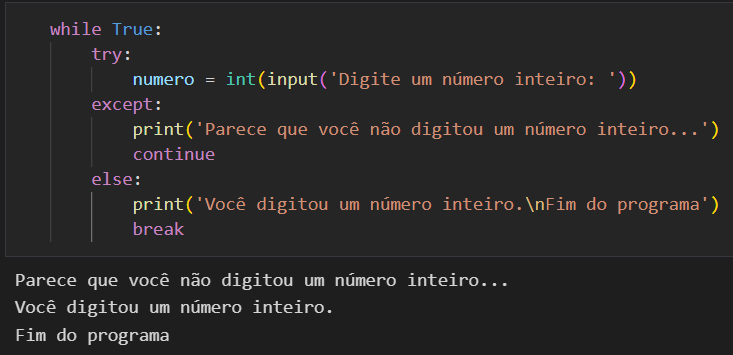
* Se houver algum erro, o programa será encerrado
* O programador pode antecipar alguns erros, utilizando mecanismos que farão com que o programa continue funcionando, mesmo que haja um erro
* Para isso, utilizam-se alguns comandos-chave:
  + **try**: esse bloco de código tentará ser executado (pode ou não ocorrer um erro)
  + **except**: bloco de código que será executado caso haja algum erro no bloco try
  + **else:** bloco de código que será executado se o código no bloco try der erro
  + **finally**: um bloco de código final que será executado, havendo ou não erro no bloco try

Texto

Descrição gerada automaticamente

**While True:**

* Utilizando o laço de repetição “While True”, é possível forçar o usuário a fornecer uma resposta correta
* Contudo, é necessário ter cautela, pois o laço “While True” sempre será verdadeiro, o que pode gerar um loop infinito...
  + Por isso, é necessário haver um comando “break” quando a intenção do programador for satisfeita (o comando break quebra o laço de repetição mais próximo)

****

**Except com erros previstos**

* É possível definir diferentes tipos de blocos except, com base no tipo de erro
  + Em outras palavras, se ocorrer um tipo específico de erro, ele executará um bloco de código específico

Texto

Descrição gerada automaticamente

# FUNÇÕES

Tela de celular com texto preto sobre fundo branco

Descrição gerada automaticamente com confiança média

Tela de celular com texto preto sobre fundo branco

Descrição gerada automaticamente com confiança média

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

* Iniciar função com “def”
* Utilizar *underlines* no nome da função (snake casing)
* Os parênteses ao final servem para passar argumentos e parâmetros pela função
* Tudo dentro da função deve estar endentado
* (‘’’) introduz e fecha comentário sobre a função (comentário pode ter várias linhas)
* Função rodará o que está dentro dela (no caso o comando “print”)
* Para chamar a função:

Ícone

Descrição gerada automaticamente

Funções que aceitam parâmetros:

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

Descrição gerada automaticamente

Tela de celular com texto preto sobre fundo branco

Descrição gerada automaticamente com confiança média

Diagrama

Descrição gerada automaticamente

## BASIC FUNCTIONS

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

* Observação: é obrigado abrir e fechar parênteses quando chamar uma função (mesmo que não tenha nenhum parâmetro dentro)

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

* O nome da variável (no caso, “name”) pode ser chamada de qualquer forma
* Se, quando chamar a função, o usuário não prover um valor para o parâmetro “name”, o python apontará um erro
  + É possível contornar esse problema gerando um valor default para a variável:

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

* + Repare que o valor atribuído é default, podendo ser mudado:

Interface gráfica do usuário

Descrição gerada automaticamente

* Criando função que retorna valor:

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

## FUNCTIONS WITH LOGICS

* Função que retorna se um número é par ou não

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

* Função que retorna TRUE se algum número de uma lista é par

Texto

Descrição gerada automaticamente

* Mesma função, mas retornará FALSE se nenhum dos números for par

Texto

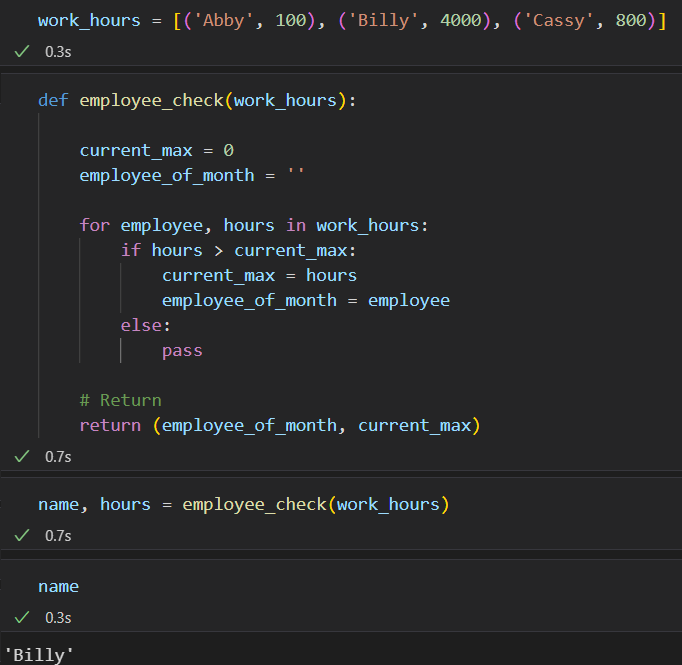
Descrição gerada automaticamente

* + Repare que o “return False” deve estar fora da endentação, pois se estivesse dentro, retornaria False no primeiro número que fosse ímpar
* Função que retorna lista com números pares de uma lista original

Texto

Descrição gerada automaticamente

## TUPLE UNPACKING WITH FUNCTION CALL



* Repare que a função retornou uma tupla
* Essa tupla pode ser decomposta quando a função for chamada (no caso, pediu-se que a separasse em “name” e “hours”)

## \*ARGS / \*\*KWARGS

* \*args = arguments
* \*\*kwargs = key-word arguments
* Utiliza-se “\*args” quando não se sabe o número de parâmetros que serão passados por uma função
  + Dessa forma, o Python criará uma tupla com todos os inputs que forem determinados pelo usuário. Essa tupla passará como argumentos para a função
  + Exemplo:

Texto, Aplicativo

Descrição gerada automaticamente

* + Observação: por convenção, utiliza-se “\*args”; no entanto, é possível utilizar qualquer nome, desde que acompanhado de um asterisco
* \*\*kwargs são utilizados para inserir dicionários, ao invés de tuplas

Texto

Descrição gerada automaticamente

* Utilizando \*args e \*\*kwargs ao mesmo tempo

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

* + Observação: como indicamos que os \*args viriam antes dos \*\*kwargs, na hora de chamar a função, devemos respeitar essa ordem

## MAP & FILTER

* O comando “map” permite que diversos valores, contidos em uma sequência, passem por uma mesma função
* Sintaxe: map(*nome\_função*, *nome\_lista*)

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

* + Observação: é necessário definir a forma de apresentação do comando map
    - Se fosse chamado somente map(square,my\_nums)), o resultado ficaria somente armazenado na memória ram, não sendo apresentado para o usuário
* Exemplo com lista composta por strings

Tela de celular com aplicativo aberto

Descrição gerada automaticamente

**Filter**

* Filtra valores que, ao passar por uma função, retornarão o valor True

**Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente**

* + Repare que, nesse caso, foram retornados os valores que são pares (eles foram filtrados a partir da função “check\_even”)

## LAMBDA EXPRESSIONS (ANONYMOUS FUNCTIONS)

* Função que se pretende usar somente uma vez
  + Ocupa menos espaço na memória do que criar uma função
  + Geralmente utilizam-se expressões lambda para rodar os **comandos map e filter** uma única vez
* Convertendo uma função em uma lambda expression:

Função:

Tela preta com letras brancas

Descrição gerada automaticamente

Expressão lambda:

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

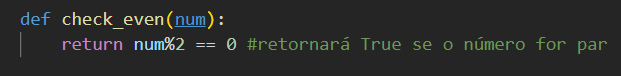
Utilização:

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

* Exemplo 2:

Função:



Expressão lambda:

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Utilização:

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

# PROGRAMAÇÃO ORIENTADA A OBJETOS (POO)

* Programação orientada a objetos permite criar objetos que possuam métodos e atributos próprios
* Importante para organização de códigos (códigos repetíveis, flexíveis e organizados)
  + Funções por si só não são o suficiente quando o código é muito grande
* Objetos possuem atributos ligados a eles (atributos são características de um objeto)

## SINTAXE BÁSICA

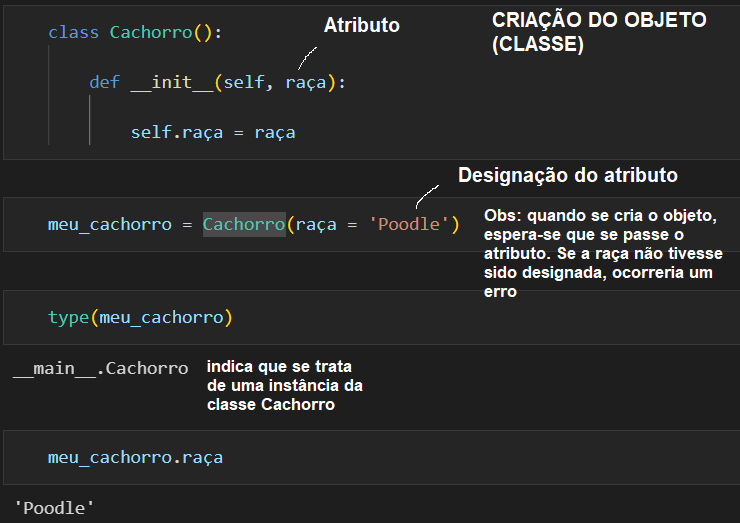
**Texto

Descrição gerada automaticamente**

* Objetos são definidos utilizando-se a palavra “class”
  + Nome do objeto/classe deve estar grafado com “cammel cases”
* Dentro de objetos, existem métodos (os quais se parecem com funções)
  + Método ‘\_\_init\_\_’ (dois underlines) permite criar uma instância do objeto
    - Espera-se que se passem parâmetros quando se cria uma instância do objeto
    - Quando um parâmetro é passado, ele é designado a um atributo da função (dessa forma, o python sabe que, quando se referir a “self.param2”, estará se referindo ao atributo “param2” que está conectado a essa instância do objeto
  + Depois, é possível haver outros métodos, que se parecerão com funções
    - Deve passar “self”, para que o Python saiba que não se trata de uma função, mas de um objeto

## BÁSICO SOBRE ATRIBUTOS

**Criação de uma classe e designação de um atributo:**

****

* “\_\_init\_\_” é o método que constrói uma classe
  + Será chamado automaticamente quando se cria uma instância da classe (em outras palavras, rodará sempre que se criar uma instância da classe)
* self representa a instância do próprio objeto
  + Deve ser declarada explicitamente
* Quando se criar uma instância da classe (que o Python entende convencionalmente por “self”), ele pedirá que seja atribuído um valor para o atributo “raça”
  + self.raça receberá o valor de raça
    - por isso, quando se chamou “meu\_cachorro.raça”, o Python retornou “Poodle”
  + Em outras palavras, o atributo, quando designado, será atribuído a “self.*nome\_do\_atributo*”
    - Lembrando que “self” representa o nome da instância

**Adicionando outros atributos a uma classe:**

**Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente**

* Repare que nem todo atributo será uma string (pode ser qualquer tipo de variável)
* Na hora de criar o objeto, todos os atributos devem ser designados, senão ocorrerá um erro
  + É possível atribuir um valor padrão para um atributo, quando se determinar o médoto “\_\_init\_\_”
  + Para isso, basta, ao lado do método, utilizar “=*valor\_default*”

**Texto

Descrição gerada automaticamente**

* + Nesses casos, esse atributo não precisará ter seu valor determinado quando o objeto for formado (embora isso seja possível – leia-se: caso se queira atribuir outro valor que não o default)

**Atributos que não precisam ser declarados (serão calculados)**

* Alguns atributos não precisam ser declarados, podendo ser calculados, com base em outros que por sua vez serão declarados
* Exemplo:

**Texto

Descrição gerada automaticamente**

**Designar atributos a todas as instâncias (padrões, atribuídos a todas as instâncias)**

* Atributos pertencentes à classe e, consequentemente, a todas as instâncias, devem ser declarados antes do método “\_\_init\_\_”

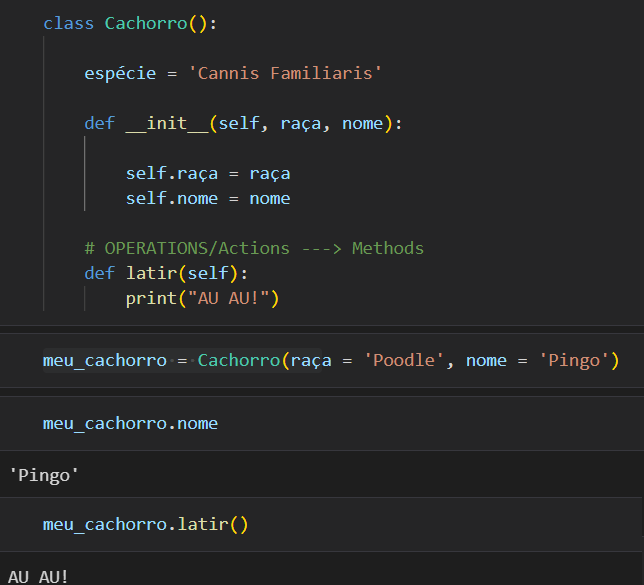
**Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente**

* Nesse caso, ao invés de utilizar o self.espécie para chamar esse atributo, também seria possível chamar Cachorro.espécie (*NomeDaClasse.atributo\_padrão)*

## BÁSICO SOBRE MÉTODOS

* Métodos são, em síntese, funções definidas dentro de uma classe, utilizadas para realizar operações que em alguns momentos utilizam atributos dos objetos criados
* Uma importante diferença entre atributos e métodos é a maneira com que são chamados
  + Métodos devem ser chamados junto de “()”

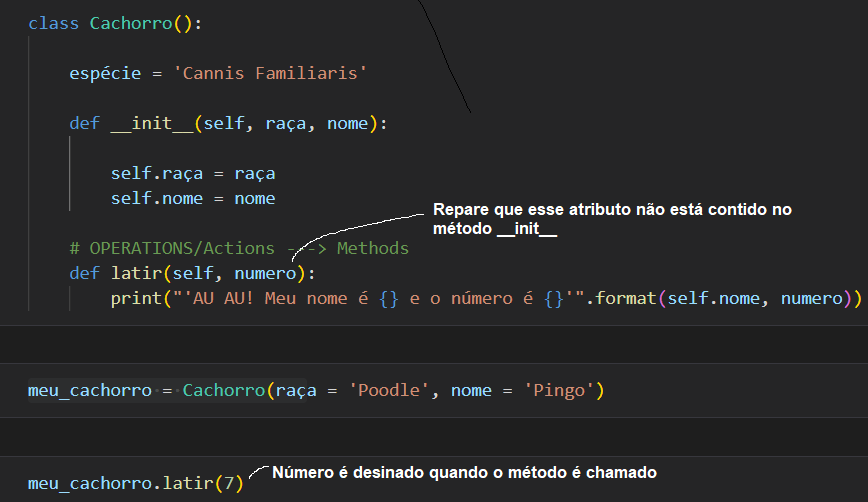


* Perceba que o método deve estar ligado ao “self”, para poder ser conectado ao objeto (Cachorro)
* Geralmente, métodos utilizarão informações dos atributos designados aos objetos

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

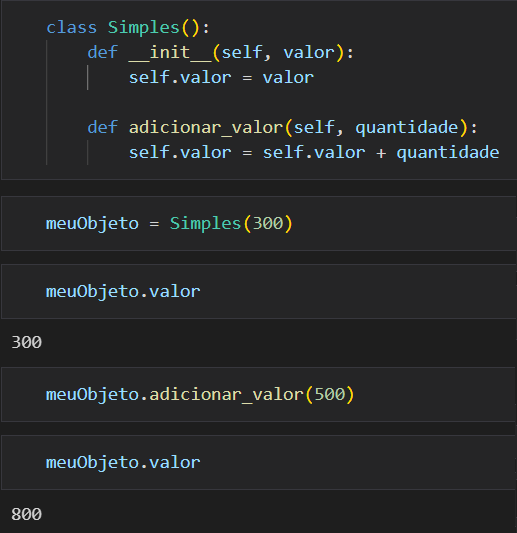
* Métodos podem utilizar argumentos externos



* Output: 'AU AU! Meu nome é Pingo e o número é 7'

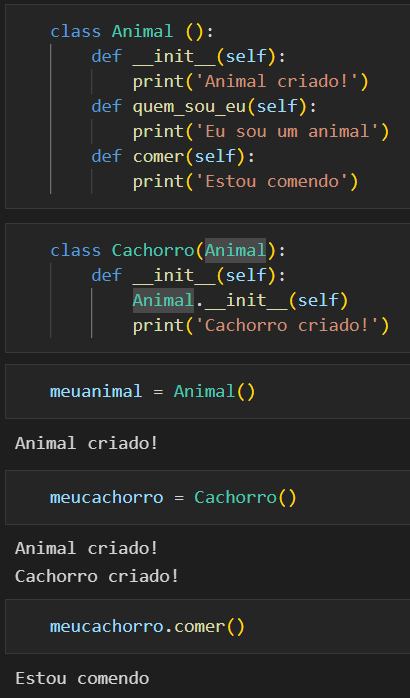
## MÉTODOS QUE AFETAM ATRIBUTOS

* É possível chamar métodos que afetam os atributos de um objeto



## INHERITANCE/ POLIMORFISMO

* **Inheritance** trata-se da capacidade de formar novas classes a partir de classes que já foram definidas
  + Utiliza códigos que já foram escritos
  + Reduz complexidade do programa
* Para que uma classe herde os métodos de outra, ao se criar a nova classe, deve-se passar o nome da outra como parâmetro (forma-se uma “classe derivada”)
  + Feito isso, os métodos da nova classe poderão utilizar-se dos métodos da outra
  + Além disso, os outros métodos da classe Animal estarão disponíveis para a classe Cachorro



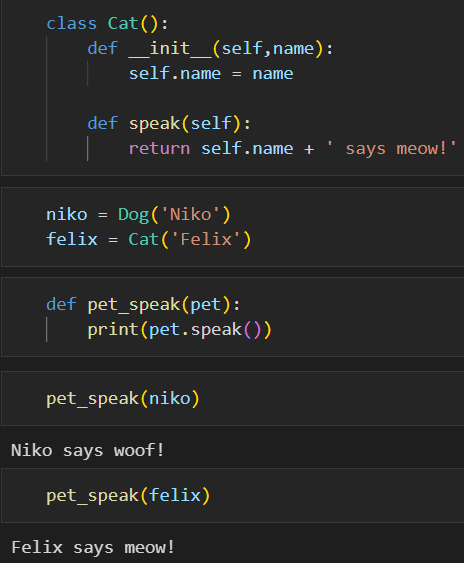
* Para que modificar um atributo da classe herdeira, basta criar um novo atributo, na nova classe (nesse caso, Cachorro), com o mesmo nome

Texto

Descrição gerada automaticamente

**Polimorfismo**

* Diz respeito à maneira com que os métodos de diferentes objetos e classes podem compartilhar o mesmo nome



* Repare que a função chama o método ‘.speak’, mas não determina a classe de origem
  + A classe é definida pelo argumento passado pela função

## MÉTODOS ESPECIAIS

* Também chamados de Magic Methods ou Dunder Methods
* Possibilitam utilizar funções internas do Python (i.e. “len()”, “print()”, “str()”) com objetos definidos pelo usuário
* Para utilizar essas funções internas, deve-se criar, na classe, um método especial que faça referência à função desejada
  + Essa função deve ser definida entre dois underlines: “def \_\_*função\_interna*\_\_():”
  + Resultado dessa função deve sempre ser retornado (“return”)

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

# MÓDULOS E BIBLIOTECAS (MODULES AND PACKAGES)

* Módulos são scripts .py que são chamados por outro script .py
* Bibliotecas são coleções de módulos
  + Para instalar bibliotecas/pacotes externos, basta digitar no CMD/Terminal “pip install *nome\_pacote*”

## CHAMANDO MÓDULOS

* Para que um script python chame outro (em outras palavras, para que um programa no python utilize um módulo), é necessário que ambos os scripts estejam na mesma pasta
* Deve-se utilizar “**from** *nome\_modulo* **import** *nome\_função*”
  + Observação: o nome do módulo corresponde ao nome do arquivo, mas sem o “.py”
* Feito isso, é possível chamar a função do módulo no programa em que foi importado

**Exemplo**

* Módulo:

**Texto

Descrição gerada automaticamente**

* Programa

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

## CRIANDO BIBLIOTECAS/PACOTES PRÓPRIOS

* Para que o python saiba que uma coleção de módulos constitui uma biblioteca, é necessário haver um script chamado “\_\_init\_\_.py” em uma pasta contendo esses scripts
  + Não é necessário escrever nada nesse arquivo
  + Basta ele estar lá para o Python entender que se trata de uma
  + Subpastas também devem possuir o arquivo “\_\_init\_\_.py”
* Para importar de uma biblioteca, deve-se chamá-la com “**from** *nome\_biblioteca*/*pacote* **import** *nome\_módulo*”
* Para importar de um subpackage (uma pasta que, dentro de uma biblioteca, constitui teoricamente outra biblioteca), deve-se chamá-la com “from *nome\_biblioteca*/*pacote*.***nome\_subpasta*** import*nome\_módulo*”
* Para chamar uma função contida em um módulo, deve-se utilizar, no corpo do código, “*nome\_módulo*.**nome\_função**()”

Texto

Descrição gerada automaticamente

## \_\_name\_\_ / ‘\_\_main\_\_’

* Como módulos podem ser chamados, muitas vezes é de interesse do programador distinguir o que ele deverá fazer quando estiver rodando por conta própria ou quando for chamado
* Nesse sentido, o Python possui uma condição que permite verificar se o programa está sendo rodado diretamente ou se está sendo chamado a partir de outro módulo
* “‘if \_\_name\_\_ == ‘\_\_main\_\_’:” é a condição que verifica isso

Texto

Descrição gerada automaticamente

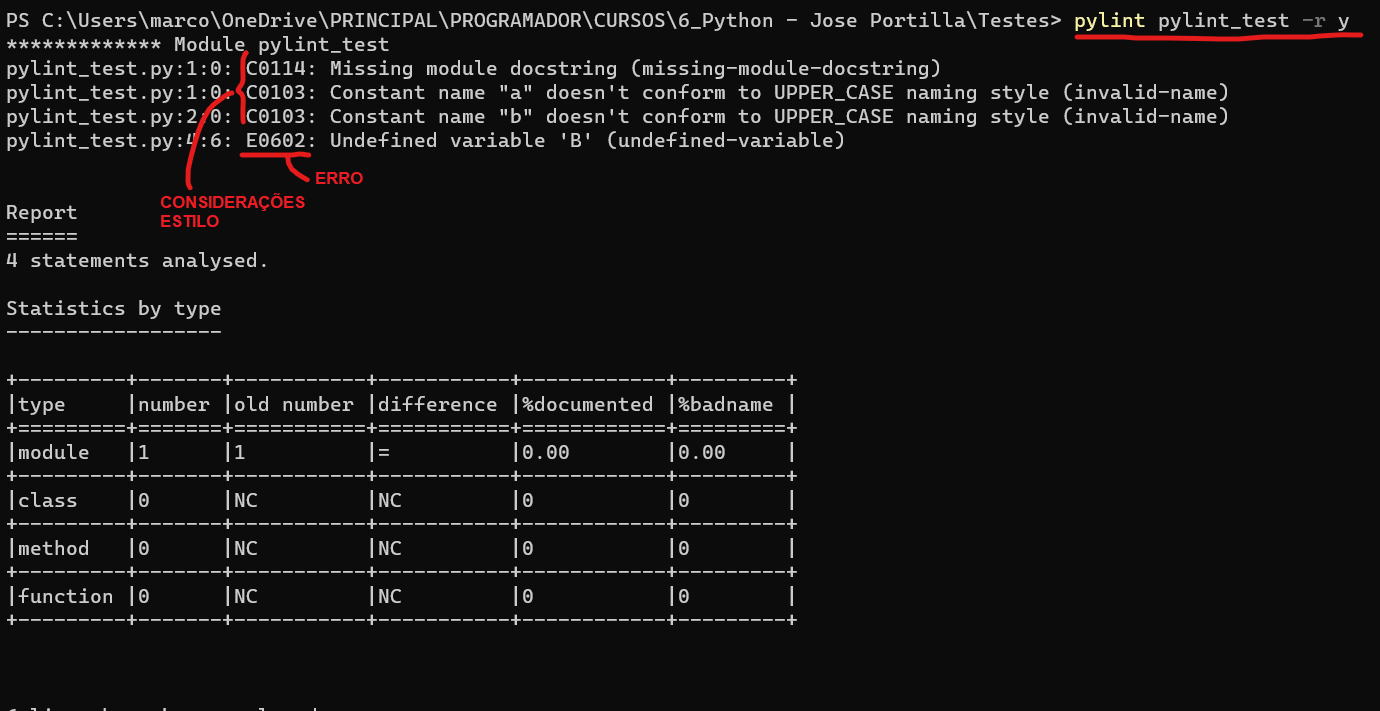
* Observação: essa condição geralmente é utilizada como uma divisória, para melhor organizar o código, entre as funções e a execução do programa

# TESTE DE CÓDIGOS

* É possível rodar arquivos de teste para ver se o código funciona como deveria
* Existe bibliotecas que ajudam nesses testes:
  + Pylint
    - Avalia seu código e reporta possíveis erros (inclusive de estilo)
  + Unittest
    - (Inerente ao Python) Permite testar seus programas e checa se eles devolvem os outputs desejados

## PYLINT

* Para chamar o pylint, é necessário utilizar o terminal
  + Deve-se estar na mesma pasta, depois chamar “pylint *nome\_arquivo*.py -r y”



* Ainda apresenta outras estatísticas interessantes, como número de classes, comentários, métodos, funções, etc... além de detalhar erros de estilo

## UNITTEST

* Necessário criar um script de teste, contendo funções e métodos de teste
* Exemplo
  + Script que será testado (cap.py):

Texto

Descrição gerada automaticamente

* + Script de teste

Texto

Descrição gerada automaticamente

* + Repare que no método de teste (nesse caso, ‘test\_one\_word(self)’), deve-se passar uma variável que será testada; determinar o resultado esperado; e verificar se o resultado esperado será igual ao resultado alcançado
    - “text=’python’” – passa variável de teste
    - result = cap.cap\_text(text) – passa a variável de texte pela função a ser avaliada
    - self.assertEqual(result, ‘Python’) – compara o resultado alcançado ao resultado esperado
  + Observação: é possível criar diversos métodos para verificar uma ou mais funções
  + Depois, deve-se rodar esse programa para verificar se houve ou não erro