|  |
| --- |
|  |
| # Programação Python |
|  |  |
|  |  |
|  |  |
|  | \*\*ATALHOS\*\* |
|  |  |
|  | \* CTRL + S = Salva o script |
|  | \* CTRL + F = Pesquisa ou substitui informação dentro da consulta ou todas as consutlas abertas |
|  | \* CTRL + R = Remove ou exibe janela de consultaSELECT = Realiza uma consulta na base de dados com retorno. |
|  |  |
|  |  |
|  |  |
|  | O que é Python? |
|  | Python é uma linguagem de programação popular. Foi criado por Guido van Rossum e lançado em 1991. |
|  |  |
|  | É usado para: |
|  |  |
|  | desenvolvimento web (lado do servidor), |
|  | desenvolvimento de software, |
|  | matemática, |
|  | script do sistema. |
|  |  |
|  |  |
|  | # Começando a programar em Python |
|  |  |
|  |  |
|  | Executar sintaxe do Python |
|  | A sintaxe do Python pode ser executada escrevendo diretamente na linha de comando: |
|  |  |
|  | print("Hello, World!") |
|  | Hello, Word! |
|  |  |
|  | Recuo Python |
|  | O recuo refere-se aos espaços no início de uma linha de código. |
|  |  |
|  | Onde em outras linguagens de programação o recuo no código é apenas para legibilidade, o recuo em Python é muito importante. |
|  |  |
|  | Python usa recuo para indicar um bloco de código. |
|  |  |
|  | Exemplo |
|  | if 5 > 2: |
|  | print("Five is greater than two!") |
|  |  |
|  | if 5 > 2: |
|  | print("Five is greater than two!") |
|  |  |
|  | if 5 > 2: |
|  | print("Five is greater than two!") |
|  |  |
|  |  |
|  | Variáveis Python |
|  | Em Python, as variáveis são criadas quando você atribui um valor a ela: |
|  |  |
|  | Exemplo |
|  | Variáveis em Python: |
|  |  |
|  | x = 5 |
|  | y = "Hello, World!" |
|  |  |
|  | Comentários |
|  | O Python tem capacidade de comentários para fins de documentação no código. |
|  |  |
|  | Os comentários começam com um # e o Python renderizará o restante da linha como um comentário: |
|  |  |
|  | Exemplo |
|  | Comentários em Python: |
|  |  |
|  | #This is a comment. |
|  | print("Hello, World!") |
|  |  |
|  | Os comentários podem ser colocados no final de uma linha e o Python ignorará o resto da linha: |
|  |  |
|  | Exemplo |
|  | print("Hello, World!") #This is a comment. |
|  |  |
|  | Um comentário não precisa ser um texto que explique o código, ele também pode ser usado para impedir que o Python execute o código: |
|  |  |
|  | Exemplo |
|  | #print("Hello, World!") |
|  | print("Cheers, Mate!") |
|  |  |
|  |  |
|  | Criando variáveis |
|  | Python não tem comando para declarar uma variável. |
|  |  |
|  | Uma variável é criada no momento em que você atribui um valor a ela. |
|  |  |
|  | Exemplo |
|  | x = 5 |
|  | y = "John" |
|  | print(x) |
|  | print(y) |
|  |  |
|  |  |
|  | As variáveis não precisam ser declaradas com nenhum tipo específico e podem até mudar de tipo depois de terem sido definidas. |
|  |  |
|  | Exemplo |
|  | x = 4 # x is of type int |
|  | x = "Sally" # x is now of type str |
|  | print(x) |
|  |  |
|  | Fundição |
|  | Se você deseja especificar o tipo de dados de uma variável, isso pode ser feito com conversão. |
|  |  |
|  | Exemplo |
|  | x = str(3) # x will be '3' |
|  | y = int(3) # y will be 3 |
|  | z = float(3) # z will be 3.0 |
|  |  |
|  | Obter o tipo |
|  | Você pode obter o tipo de dados de uma variável com a type()função. |
|  |  |
|  | Exemplo |
|  | x = 5 |
|  | y = "John" |
|  | print(type(x)) |
|  | print(type(y)) |
|  |  |
|  | Citações simples ou duplas? |
|  | Variáveis de string podem ser declaradas usando aspas simples ou duplas: |
|  |  |
|  | Exemplo |
|  | x = "John" |
|  | # is the same as |
|  | x = 'John' |
|  |  |
|  | Maiúsculas e minúsculas |
|  | Os nomes de variáveis diferenciam maiúsculas de minúsculas. |
|  |  |
|  | Exemplo |
|  | Isso criará duas variáveis: |
|  |  |
|  | a = 4 |
|  | A = "Sally" |
|  | #A will not overwrite a |
|  |  |
|  | Nomes de Variáveis |
|  | Uma variável pode ter um nome curto (como x e y) ou um nome mais descritivo (age, carname, total\_volume). Regras para variáveis Python: |
|  | Um nome de variável deve começar com uma letra ou o caractere sublinhado |
|  | Um nome de variável não pode começar com um número |
|  | Um nome de variável pode conter apenas caracteres alfanuméricos e sublinhados (Az, 0-9 e \_ ) |
|  | Os nomes das variáveis diferenciam maiúsculas de minúsculas (idade, idade e IDADE são três variáveis diferentes) |
|  | Exemplo |
|  | Nomes de variáveis legais: |
|  |  |
|  | myvar = "John" |
|  | my\_var = "John" |
|  | \_my\_var = "John" |
|  | myVar = "John" |
|  | MYVAR = "John" |
|  | myvar2 = "John" |
|  |  |
|  |  |
|  | Nomes de variáveis de várias palavras |
|  | Nomes de variáveis com mais de uma palavra podem ser difíceis de ler. |
|  |  |
|  | Existem várias técnicas que você pode usar para torná-los mais legíveis: |
|  |  |
|  | Caixa de camelo |
|  | Cada palavra, exceto a primeira, começa com letra maiúscula: |
|  |  |
|  | myVariableName = "John" |
|  |  |
|  | Caso Pascal |
|  | Cada palavra começa com uma letra maiúscula: |
|  |  |
|  | MyVariableName = "John" |
|  |  |
|  | Caso da Cobra |
|  | Cada palavra é separada por um caractere de sublinhado: |
|  |  |
|  | my\_variable\_name = "John" |
|  |  |
|  | Muitos valores para múltiplas variáveis |
|  | Python permite atribuir valores a várias variáveis em uma linha: |
|  |  |
|  | Exemplo |
|  | x, y, z = "Orange", "Banana", "Cherry" |
|  | print(x) |
|  | print(y) |
|  | print(z) |
|  |  |
|  | Um valor para várias variáveis |
|  | E você pode atribuir o mesmo valor a várias variáveis em uma linha: |
|  |  |
|  | Exemplo |
|  | x = y = z = "Orange" |
|  | print(x) |
|  | print(y) |
|  | print(z) |
|  |  |
|  | Descompactar uma coleção |
|  | Se você tem uma coleção de valores em uma lista, tupla etc. Python permite que você extraia os valores em variáveis. Isso é chamado de descompactação . |
|  |  |
|  | Exemplo |
|  | Descompacte uma lista: |
|  |  |
|  | fruits = ["apple", "banana", "cherry"] |
|  | x, y, z = fruits |
|  | print(x) |
|  | print(y) |
|  | print(z) |
|  |  |
|  | Variáveis de saída |
|  | A função Python print()é frequentemente usada para gerar variáveis. |
|  |  |
|  | Exemplo |
|  | x = "Python is awesome" |
|  | print(x) |
|  |  |
|  | Na print()função, você gera várias variáveis, separadas por uma vírgula: |
|  |  |
|  | Exemplo |
|  | x = "Python" |
|  | y = "is" |
|  | z = "awesome" |
|  | print(x, y, z) |
|  | Você também pode usar o + operador para gerar várias variáveis: |
|  |  |
|  | Exemplo |
|  | x = "Python " |
|  | y = "is " |
|  | z = "awesome" |
|  | print(x + y + z) |
|  | Observe o caractere de espaço após "Python "e "is ", sem eles o resultado seria "Pythonisawesome". |
|  |  |
|  | Para números, o + caractere funciona como um operador matemático: |
|  |  |
|  | Exemplo |
|  | x = 5 |
|  | y = 10 |
|  | print(x + y) |
|  |  |
|  | Na print()função, quando você tenta combinar uma string e um número com o + operador, o Python lhe dará um erro: |
|  |  |
|  | Exemplo |
|  | x = 5 |
|  | y = "John" |
|  | print(x + y) |
|  |  |
|  | A melhor maneira de gerar várias variáveis na print()função é separá-las com vírgulas, que até suportam diferentes tipos de dados: |
|  |  |
|  | Exemplo |
|  | x = 5 |
|  | y = "John" |
|  | print(x, y) |
|  |  |
|  |  |
|  | Variáveis globais |
|  | Variáveis criadas fora de uma função (como em todos os exemplos acima) são conhecidas como variáveis globais. |
|  |  |
|  | As variáveis globais podem ser usadas por todos, tanto dentro das funções quanto fora delas. |
|  |  |
|  | Exemplo |
|  | Crie uma variável fora de uma função e use-a dentro da função |
|  |  |
|  | x = "awesome" |
|  |  |
|  | def myfunc(): |
|  | print("Python is " + x) |
|  |  |
|  | myfunc() |
|  | Se você criar uma variável com o mesmo nome dentro de uma função, essa variável será local e só poderá ser usada dentro da função. A variável global com o mesmo nome permanecerá como estava, global e com o valor original. |
|  |  |
|  | Exemplo |
|  | Crie uma variável dentro de uma função, com o mesmo nome da variável global |
|  |  |
|  | x = "awesome" |
|  |  |
|  | def myfunc(): |
|  | x = "fantastic" |
|  | print("Python is " + x) |
|  |  |
|  | myfunc() |
|  |  |
|  | print("Python is " + x) |
|  |  |
|  | A palavra-chave global |
|  |  |
|  | Normalmente, quando você cria uma variável dentro de uma função, essa variável é local e só pode ser usada dentro dessa função. |
|  |  |
|  | Para criar uma variável global dentro de uma função, você pode usar a palavra- globalchave. |
|  |  |
|  | Exemplo |
|  |  |
|  | Se você usar a palavra- globalchave, a variável pertence ao escopo global: |
|  |  |
|  | def myfunc(): |
|  | global x |
|  | x = "fantastic" |
|  |  |
|  | myfunc() |
|  |  |
|  | print("Python is " + x) |
|  |  |
|  | Além disso, use a palavra- globalchave se quiser alterar uma variável global dentro de uma função. |
|  |  |
|  | Exemplo |
|  |  |
|  | Para alterar o valor de uma variável global dentro de uma função, consulte a variável usando a globalpalavra-chave: |
|  |  |
|  | x = "awesome" |
|  |  |
|  | def myfunc(): |
|  | global x |
|  | x = "fantastic" |
|  |  |
|  | myfunc() |
|  |  |
|  | print("Python is " + x) |
|  |  |
|  |  |
|  | Tipos de dados integrados |
|  |  |
|  | Na programação, o tipo de dados é um conceito importante. |
|  |  |
|  | Variáveis podem armazenar dados de diferentes tipos, e diferentes tipos podem fazer coisas diferentes. |
|  |  |
|  | O Python tem os seguintes tipos de dados integrados por padrão, nestas categorias: |
|  |  |
|  | Tipo de texto: str |
|  | Tipos Numéricos: int, float, complex |
|  | Tipos de sequência: list, tuple, range |
|  | Tipo de mapeamento: dict |
|  | Tipos de conjunto: set,frozenset |
|  | Tipo booleano: bool |
|  | Tipos binários: bytes, bytearray, memoryview |
|  | Nenhum Tipo: NoneType |
|  |  |
|  | Obtendo o tipo de dados |
|  |  |
|  | Você pode obter o tipo de dados de qualquer objeto usando a type()função: |
|  |  |
|  | Exemplo |
|  | Imprima o tipo de dados da variável x: |
|  |  |
|  | x = 5 |
|  | print(type(x)) |
|  |  |
|  |  |
|  | Configurando o tipo de dados |
|  |  |
|  | Em Python, o tipo de dados é definido quando você atribui um valor a uma variável: |
|  |  |
|  | Example |
|  |  |
|  | x = "Hello World" str |
|  | x = 20 int |
|  | x = 20.5 float |
|  | x = 1j complex |
|  | x = ["apple", "banana", "cherry"] list |
|  | x = ("apple", "banana", "cherry") tuple |
|  | x = range(6) range |
|  | x = {"name" : "John", "age" : 36} dict |
|  | x = {"apple", "banana", "cherry"} set |
|  | x = frozenset({"apple", "banana", "cherry"}) frozenset |
|  | x = True bool |
|  | x = b"Hello" bytes |
|  | x = bytearray(5) bytearray |
|  | x = memoryview(bytes(5)) memoryview |
|  | x = None NoneType |
|  |  |
|  |  |
|  | Configurando o tipo de dados específico |
|  |  |
|  | Se você deseja especificar o tipo de dados, pode usar as seguintes funções de construtor: |
|  |  |
|  | Example Data Type Try it |
|  | x = str("Hello World") str |
|  | x = int(20) int |
|  | x = float(20.5) float |
|  | x = complex(1j) complex |
|  | x = list(("apple", "banana", "cherry")) list |
|  | x = tuple(("apple", "banana", "cherry")) tuple |
|  | x = range(6) range |
|  | x = dict(name="John", age=36) dict |
|  | x = set(("apple", "banana", "cherry")) set |
|  | x = frozenset(("apple", "banana", "cherry")) frozenset |
|  | x = bool(5) bool |
|  | x = bytes(5) bytes |
|  | x = bytearray(5) bytearray |
|  | x = memoryview(bytes(5)) memoryview |
|  |  |
|  | Números Python |
|  | Existem três tipos numéricos em Python: |
|  |  |
|  | int |
|  | float |
|  | complex |
|  |  |
|  | Variáveis de tipos numéricos são criadas quando você atribui um valor a elas: |
|  |  |
|  | Exemplo |
|  | x = 1 # int |
|  | y = 2.8 # float |
|  | z = 1j # complex |
|  | Para verificar o tipo de qualquer objeto em Python, use a type()função: |
|  |  |
|  | Exemplo |
|  | print(type(x)) |
|  | print(type(y)) |
|  | print(type(z)) |
|  |  |
|  | Int |
|  | Int, ou inteiro, é um número inteiro, positivo ou negativo, sem decimais, de comprimento ilimitado. |
|  |  |
|  | Exemplo |
|  | Inteiros: |
|  |  |
|  | x = 1 |
|  | y = 35656222554887711 |
|  | z = -3255522 |
|  |  |
|  | print(type(x)) |
|  | print(type(y)) |
|  | print(type(z)) |
|  |  |
|  | Flutuador |
|  |  |
|  | Float, ou "número de ponto flutuante" é um número, positivo ou negativo, contendo um ou mais decimais. |
|  |  |
|  | Exemplo |
|  | Flutuadores: |
|  |  |
|  | x = 1.10 |
|  | y = 1.0 |
|  | z = -35.59 |
|  |  |
|  | print(type(x)) |
|  | print(type(y)) |
|  | print(type(z)) |
|  |  |
|  | Float também pode ser números científicos com um "e" para indicar a potência de 10. |
|  |  |
|  | Exemplo |
|  | Flutuadores: |
|  |  |
|  | x = 35e3 |
|  | y = 12E4 |
|  | z = -87.7e100 |
|  |  |
|  | print(type(x)) |
|  | print(type(y)) |
|  | print(type(z)) |
|  |  |
|  | Complexo |
|  | Os números complexos são escritos com um "j" como a parte imaginária: |
|  |  |
|  | Exemplo |
|  | Complexo: |
|  |  |
|  | x = 3+5j |
|  | y = 5j |
|  | z = -5j |
|  |  |
|  | print(type(x)) |
|  | print(type(y)) |
|  | print(type(z)) |
|  |  |
|  | Conversão de tipo |
|  | Você pode converter de um tipo para outro com os métodos int(), float()e :complex() |
|  |  |
|  | Exemplo |
|  | Converter de um tipo para outro: |
|  |  |
|  | x = 1 # int |
|  | y = 2.8 # float |
|  | z = 1j # complex |
|  |  |
|  | #convert from int to float: |
|  | a = float(x) |
|  |  |
|  | #convert from float to int: |
|  | b = int(y) |
|  |  |
|  | #convert from int to complex: |
|  | c = complex(x) |
|  |  |
|  | print(a) |
|  | print(b) |
|  | print(c) |
|  |  |
|  | print(type(a)) |
|  | print(type(b)) |
|  | print(type(c)) |
|  |  |
|  | Nota: Você não pode converter números complexos em outro tipo de número. |
|  |  |
|  | Número aleatório |
|  | Python não tem uma random()função para fazer um número aleatório, mas Python tem um módulo embutido chamado randomque pode ser usado para fazer números aleatórios: |
|  |  |
|  | Exemplo |
|  | Importe o módulo aleatório e exiba um número aleatório entre 1 e 9: |
|  |  |
|  | import random |
|  |  |
|  | print(random.randrange(1, 10)) |
|  |  |
|  | Especifique um tipo de variável |
|  | Pode haver momentos em que você deseja especificar um tipo para uma variável. Isso pode ser feito com fundição. Python é uma linguagem orientada a objetos e, como tal, usa classes para definir tipos de dados, incluindo seus tipos primitivos. |
|  |  |
|  | A conversão em python é, portanto, feita usando funções construtoras: |
|  |  |
|  | int() - constrói um número inteiro a partir de um literal inteiro, um literal float (removendo todos os decimais) ou um literal de string (desde que a string represente um número inteiro) |
|  | float() - constrói um número float a partir de um literal inteiro, um literal float ou um literal de string (desde que a string represente um float ou um inteiro) |
|  | str() - constrói uma string a partir de uma ampla variedade de tipos de dados, incluindo strings, literais inteiros e literais float |
|  |  |
|  | Exemplo |
|  | Inteiros: |
|  |  |
|  | x = int(1) # x will be 1 |
|  | y = int(2.8) # y will be 2 |
|  | z = int("3") # z will be 3 |
|  |  |
|  | Exemplo |
|  | Flutuadores: |
|  |  |
|  | x = float(1) # x will be 1.0 |
|  | y = float(2.8) # y will be 2.8 |
|  | z = float("3") # z will be 3.0 |
|  | w = float("4.2") # w will be 4.2 |
|  |  |
|  | Exemplo |
|  |  |
|  | Cordas: |
|  |  |
|  | x = str("s1") # x will be 's1' |
|  | y = str(2) # y will be '2' |
|  | z = str(3.0) # z will be '3.0' |
|  |  |
|  |  |
|  | Cordas |
|  |  |
|  | Strings em python são cercadas por aspas simples ou aspas duplas. |
|  |  |
|  | 'olá' é o mesmo que "olá" . |
|  |  |
|  | Você pode exibir um literal de string com a print()função: |
|  |  |
|  | Exemplo |
|  | print("Hello") |
|  | print('Hello') |
|  |  |
|  | Atribuir String a uma Variável |
|  |  |
|  | A atribuição de uma string a uma variável é feita com o nome da variável seguido por um sinal de igual e a string: |
|  |  |
|  | Exemplo |
|  | a = "Hello" |
|  | print(a) |
|  |  |
|  | Strings de várias linhas |
|  |  |
|  | Você pode atribuir uma string de várias linhas a uma variável usando três aspas: |
|  |  |
|  | Exemplo |
|  | Você pode usar três aspas duplas: |
|  |  |
|  | a = """Lorem ipsum dolor sit amet, |
|  | consectetur adipiscing elit, |
|  | sed do eiusmod tempor incididunt |
|  | ut labore et dolore magna aliqua.""" |
|  | print(a) |
|  |  |
|  | Ou três aspas simples: |
|  |  |
|  | Exemplo |
|  | a = '''Lorem ipsum dolor sit amet, |
|  | consectetur adipiscing elit, |
|  | sed do eiusmod tempor incididunt |
|  | ut labore et dolore magna aliqua.''' |
|  | print(a) |
|  |  |
|  | Strings são Arrays |
|  |  |
|  | Como muitas outras linguagens de programação populares, strings em Python são arrays de bytes que representam caracteres unicode. |
|  |  |
|  | No entanto, o Python não possui um tipo de dados de caractere, um único caractere é simplesmente uma string com um comprimento de 1. |
|  |  |
|  | Colchetes podem ser usados para acessar elementos da string. |
|  |  |
|  | Exemplo |
|  |  |
|  | Pegue o caractere na posição 1 (lembre-se que o primeiro caractere tem a posição 0): |
|  |  |
|  | a = "Hello, World!" |
|  | print(a[1]) |
|  |  |
|  | Fazendo um loop em uma string |
|  |  |
|  | Como strings são arrays, podemos fazer um loop pelos caracteres em uma string, com um forloop. |
|  |  |
|  | Exemplo |
|  |  |
|  | Percorra as letras da palavra "banana": |
|  |  |
|  | for x in "banana": |
|  | print(x) |
|  |  |
|  | Comprimento da string |
|  |  |
|  | Para obter o comprimento de uma string, use a len()função. |
|  |  |
|  | Exemplo |
|  | A len()função retorna o comprimento de uma string: |
|  |  |
|  | a = "Hello, World!" |
|  | print(len(a)) |
|  |  |
|  | String de verificação |
|  |  |
|  | Para verificar se uma determinada frase ou caractere está presente em uma string, podemos usar a palavra-chave in. |
|  |  |
|  | Exemplo |
|  | Verifique se "grátis" está presente no texto a seguir: |
|  |  |
|  | txt = "The best things in life are free!" |
|  | print("free" in txt) |
|  |  |
|  | Use-o em uma ifdeclaração: |
|  |  |
|  | Exemplo |
|  |  |
|  | Imprima apenas se "grátis" estiver presente: |
|  |  |
|  | txt = "The best things in life are free!" |
|  | if "free" in txt: |
|  | print("Yes, 'free' is present.") |
|  |  |
|  | Verifique se NÃO |
|  |  |
|  | Para verificar se uma determinada frase ou caractere NÃO está presente em uma string, podemos usar a palavra-chave not in. |
|  |  |
|  | Exemplo |
|  |  |
|  | Verifique se "caro" NÃO está presente no texto a seguir: |
|  |  |
|  | txt = "The best things in life are free!" |
|  | print("expensive" not in txt) |
|  | Use-o em uma ifdeclaração: |
|  |  |
|  | Exemplo |
|  |  |
|  | imprima somente se "caro" NÃO estiver presente: |
|  |  |
|  | txt = "The best things in life are free!" |
|  | if "expensive" not in txt: |
|  | print("No, 'expensive' is NOT present.") |
|  |  |
|  | Fatiamento |
|  |  |
|  | Você pode retornar um intervalo de caracteres usando a sintaxe de fatia. |
|  |  |
|  | Especifique o índice inicial e o índice final, separados por dois pontos, para retornar uma parte da string. |
|  |  |
|  | Exemplo |
|  | Obtenha os personagens da posição 2 para a posição 5 (não incluído): |
|  |  |
|  | b = "Hello, World!" |
|  | print(b[2:5]) |
|  |  |
|  | Nota: O primeiro caractere tem índice 0. |
|  |  |
|  | Fatia desde o início |
|  | Ao deixar de fora o índice inicial, o intervalo começará no primeiro caractere: |
|  |  |
|  | Exemplo |
|  |  |
|  | Obtenha os personagens desde o início até a posição 5 (não incluído): |
|  |  |
|  | b = "Hello, World!" |
|  | print(b[:5]) |
|  |  |
|  |  |
|  | Fatie até o fim |
|  | Ao deixar de fora o índice final , o intervalo irá para o final: |
|  |  |
|  | Exemplo |
|  |  |
|  | Pegue os personagens da posição 2 e até o final: |
|  |  |
|  | b = "Hello, World!" |
|  | print(b[2:]) |
|  |  |
|  | Indexação negativa |
|  |  |
|  | Use índices negativos para iniciar a fatia a partir do final da string: |
|  |  |
|  | Exemplo |
|  | Obtenha os personagens: |
|  |  |
|  | De: "o" em "Mundo!" (posição -5) |
|  |  |
|  | Para, mas não incluído: "d" em "Mundo!" (posição -2): |
|  |  |
|  | b = "Hello, World!" |
|  | print(b[-5:-2]) |
|  |  |
|  | Python tem um conjunto de métodos integrados que você pode usar em strings. |
|  |  |
|  | Maiúsculas |
|  |  |
|  | Exemplo |
|  | O upper()método retorna a string em maiúsculas: |
|  |  |
|  | a = "Hello, World!" |
|  | print(a.upper()) |
|  |  |
|  | Minúsculas |
|  |  |
|  | Exemplo |
|  | O lower()método retorna a string em letras minúsculas: |
|  |  |
|  | a = "Hello, World!" |
|  | print(a.lower()) |
|  |  |
|  | Remover espaço em branco |
|  |  |
|  | Espaço em branco é o espaço antes e/ou depois do texto real e, muitas vezes, você deseja remover esse espaço. |
|  |  |
|  | Exemplo |
|  |  |
|  | O strip()método remove qualquer espaço em branco do início ou do fim: |
|  |  |
|  | a = " Hello, World! " |
|  | print(a.strip()) # returns "Hello, World!" |
|  |  |
|  |  |
|  | Substituir Cadeia |
|  |  |
|  | Exemplo |
|  | O replace()método substitui uma string por outra string: |
|  |  |
|  | a = "Hello, World!" |
|  | print(a.replace("H", "J")) |
|  |  |
|  | Sequência dividida |
|  |  |
|  | O split()método retorna uma lista onde o texto entre o separador especificado se torna os itens da lista. |
|  |  |
|  | Exemplo |
|  | O split()método divide a string em substrings se encontrar instâncias do separador: |
|  |  |
|  | a = "Hello, World!" |
|  | print(a.split(",")) # returns ['Hello', ' World!'] |
|  |  |
|  | Concatenação de String |
|  |  |
|  | Para concatenar ou combinar duas strings você pode usar o operador +. |
|  |  |
|  | Exemplo |
|  | Mesclar variável acom variável b em variável c: |
|  |  |
|  | a = "Hello" |
|  | b = "World" |
|  | c = a + b |
|  | print(c) |
|  |  |
|  | Exemplo |
|  | Para adicionar um espaço entre eles, adicione um " ": |
|  |  |
|  | a = "Hello" |
|  | b = "World" |
|  | c = a + " " + b |
|  | print(c) |
|  |  |
|  | Formato de string |
|  |  |
|  | Como aprendemos no capítulo Variáveis do Python, não podemos combinar strings e números assim: |
|  |  |
|  | Exemplo |
|  | age = 36 |
|  | txt = "My name is John, I am " + age |
|  | print(txt) |
|  |  |
|  | Mas podemos combinar strings e números usando o format()método! |
|  |  |
|  | O format()método pega os argumentos passados, os formata e os coloca na string onde {}estão os espaços reservados: |
|  |  |
|  | Exemplo |
|  | Use o format()método para inserir números em strings: |
|  |  |
|  | age = 36 |
|  | txt = "My name is John, and I am {}" |
|  | print(txt.format(age)) |
|  |  |
|  | O método format() recebe um número ilimitado de argumentos e são colocados nos respectivos espaços reservados: |
|  |  |
|  | Exemplo |
|  | quantity = 3 |
|  | itemno = 567 |
|  | price = 49.95 |
|  | myorder = "I want {} pieces of item {} for {} dollars." |
|  | print(myorder.format(quantity, itemno, price)) |
|  |  |
|  | Você pode usar números de índice {0}para garantir que os argumentos sejam colocados nos espaços reservados corretos: |
|  |  |
|  | Exemplo |
|  | quantity = 3 |
|  | itemno = 567 |
|  | price = 49.95 |
|  | myorder = "I want to pay {2} dollars for {0} pieces of item {1}." |
|  | print(myorder.format(quantity, itemno, price)) |
|  |  |
|  | Caractere de Fuga |
|  |  |
|  | Para inserir caracteres ilegais em uma string, use um caractere de escape. |
|  |  |
|  | Um caractere de escape é uma barra invertida \seguida pelo caractere que você deseja inserir. |
|  |  |
|  | Um exemplo de um caractere ilegal é uma aspa dupla dentro de uma string que é cercada por aspas duplas: |
|  |  |
|  | Exemplo |
|  |  |
|  | Você receberá um erro se usar aspas duplas dentro de uma string cercada por aspas duplas: |
|  |  |
|  | txt = "We are the so-called "Vikings" from the north." |
|  | Para corrigir esse problema, use o caractere de escape \": |
|  |  |
|  | Exemplo |
|  | O caractere de escape permite que você use aspas duplas quando normalmente não seria permitido: |
|  |  |
|  | txt = "We are the so-called \"Vikings\" from the north." |
|  | Caracteres de escape |
|  | Outros caracteres de escape usados em Python: |
|  |  |
|  | Code Result |
|  | \' Single Quote |
|  | \\ Backslash |
|  | \n New Line |
|  | \r Carriage Return |
|  | \t Tab |
|  | \b Backspace |
|  | \f Form Feed |
|  | \ooo Octal value |
|  | \xhh Hex value |
|  |  |
|  | Métodos de string |
|  | Python tem um conjunto de métodos integrados que você pode usar em strings. |
|  |  |
|  | Nota: Todos os métodos de string retornam novos valores. Eles não alteram a string original. |
|  |  |
|  | Method Description |
|  | capitalize() Converts the first character to upper case |
|  | casefold() Converts string into lower case |
|  | center() Returns a centered string |
|  | count() Returns the number of times a specified value occurs in a string |
|  | encode() Returns an encoded version of the string |
|  | endswith() Returns true if the string ends with the specified value |
|  | expandtabs() Sets the tab size of the string |
|  | find() Searches the string for a specified value and returns the position of where it was found |
|  | format() Formats specified values in a string |
|  | format\_map() Formats specified values in a string |
|  | index() Searches the string for a specified value and returns the position of where it was found |
|  | isalnum() Returns True if all characters in the string are alphanumeric |
|  | isalpha() Returns True if all characters in the string are in the alphabet |
|  | isdecimal() Returns True if all characters in the string are decimals |
|  | isdigit() Returns True if all characters in the string are digits |
|  | isidentifier()Returns True if the string is an identifier |
|  | islower() Returns True if all characters in the string are lower case |
|  | isnumeric() Returns True if all characters in the string are numeric |
|  | isprintable() Returns True if all characters in the string are printable |
|  | isspace() Returns True if all characters in the string are whitespaces |
|  | istitle() Returns True if the string follows the rules of a title |
|  | isupper() Returns True if all characters in the string are upper case |
|  | join() Joins the elements of an iterable to the end of the string |
|  | ljust() Returns a left justified version of the string |
|  | lower() Converts a string into lower case |
|  | lstrip() Returns a left trim version of the string |
|  | maketrans() Returns a translation table to be used in translations |
|  | partition() Returns a tuple where the string is parted into three parts |
|  | replace() Returns a string where a specified value is replaced with a specified value |
|  | rfind() Searches the string for a specified value and returns the last position of where it was found |
|  | rindex() Searches the string for a specified value and returns the last position of where it was found |
|  | rjust() Returns a right justified version of the string |
|  | rpartition() Returns a tuple where the string is parted into three parts |
|  | rsplit() Splits the string at the specified separator, and returns a list |
|  | rstrip() Returns a right trim version of the string |
|  | split() Splits the string at the specified separator, and returns a list |
|  | splitlines() Splits the string at line breaks and returns a list |
|  | startswith() Returns true if the string starts with the specified value |
|  | strip() Returns a trimmed version of the string |
|  | swapcase() Swaps cases, lower case becomes upper case and vice versa |
|  | title() Converts the first character of each word to upper case |
|  | translate() Returns a translated string |
|  | upper() Converts a string into upper case |
|  | zfill() Fills the string with a specified number of 0 values at the beginning |
|  |  |
|  |  |
|  | Booleanos representam um dos dois valores: True ou False. |
|  |  |
|  | Valores Booleanos |
|  | Na programação, muitas vezes você precisa saber se uma expressão é True ou False. |
|  |  |
|  | Você pode avaliar qualquer expressão em Python e obter uma das duas respostas, True ou False. |
|  |  |
|  | Quando você compara dois valores, a expressão é avaliada e o Python retorna a resposta booleana: |
|  |  |
|  | Exemplo |
|  | print(10 > 9) |
|  | print(10 == 9) |
|  | print(10 < 9) |
|  |  |
|  | Quando você executa uma condição em uma instrução if, o Python retorna Trueou False: |
|  |  |
|  | Exemplo |
|  | Imprima uma mensagem com base em se a condição é True ou False: |
|  |  |
|  | a = 200 |
|  | b = 33 |
|  |  |
|  | if b > a: |
|  | print("b is greater than a") |
|  | else: |
|  | print("b is not greater than a") |
|  |  |
|  | Avaliar Valores e Variáveis |
|  |  |
|  | A bool()função permite avaliar qualquer valor e dar a você True ou False em troca, |
|  |  |
|  | Exemplo |
|  | Avalie uma string e um número: |
|  |  |
|  | print(bool("Hello")) |
|  | print(bool(15)) |
|  |  |
|  | Exemplo |
|  | Avalie duas variáveis: |
|  |  |
|  | x = "Hello" |
|  | y = 15 |
|  |  |
|  | print(bool(x)) |
|  | print(bool(y)) |
|  |  |
|  | A maioria dos valores são verdadeiros |
|  |  |
|  | Quase qualquer valor é avaliado True se tiver algum tipo de conteúdo. |
|  |  |
|  | Qualquer string é True, exceto strings vazias. |
|  |  |
|  | Qualquer número é True, exceto 0. |
|  |  |
|  | Qualquer lista, tupla, conjunto e dicionário são True, exceto os vazios. |
|  |  |
|  | Exemplo |
|  | O seguinte retornará True: |
|  |  |
|  | bool("abc") |
|  | bool(123) |
|  | bool(["apple", "cherry", "banana"]) |
|  |  |
|  | Alguns valores são falsos |
|  | Na verdade, não há muitos valores avaliados para False, exceto valores vazios, como (), [], {}, "", o número 0 e o valor None. E, claro, o valor é False avaliado como False. |
|  |  |
|  | Exemplo |
|  | O seguinte retornará False: |
|  |  |
|  | bool(False) |
|  | bool(None) |
|  | bool(0) |
|  | bool("") |
|  | bool(()) |
|  | bool([]) |
|  | bool({}) |
|  |  |
|  | Mais um valor, ou objeto neste caso, é avaliado como False, e isso é se você tem um objeto que é feito de uma classe com uma \_\_len\_\_função que retorna 0 ou False: |
|  |  |
|  | Exemplo |
|  | class myclass(): |
|  | def \_\_len\_\_(self): |
|  | return 0 |
|  |  |
|  | myobj = myclass() |
|  | print(bool(myobj)) |
|  |  |
|  | Funções podem retornar um booleano |
|  | Você pode criar funções que retornam um valor booleano: |
|  |  |
|  | Exemplo |
|  | Imprima a resposta de uma função: |
|  |  |
|  | def myFunction() : |
|  | return True |
|  |  |
|  | print(myFunction()) |
|  |  |
|  | Você pode executar o código com base na resposta booleana de uma função: |
|  |  |
|  | Exemplo |
|  | Imprima "SIM!" se a função retornar True, caso contrário imprima "NO!": |
|  |  |
|  | def myFunction() : |
|  | return True |
|  |  |
|  | if myFunction(): |
|  | print("YES!") |
|  | else: |
|  | print("NO!") |
|  |  |
|  | O Python também possui muitas funções internas que retornam um valor booleano, como a isinstance() função, que pode ser usada para determinar se um objeto é de um determinado tipo de dados: |
|  |  |
|  | Exemplo |
|  | Verifique se um objeto é um inteiro ou não: |
|  |  |
|  | x = 200 |
|  | print(isinstance(x, int)) |
|  |  |
|  |  |
|  | Operadores Python |
|  |  |
|  | Os operadores são usados para realizar operações em variáveis e valores. |
|  |  |
|  | No exemplo abaixo, usamos o +operador para somar dois valores: |
|  |  |
|  | Exemplo |
|  | print(10 + 5) |
|  |  |
|  | Python divide os operadores nos seguintes grupos: |
|  |  |
|  | Operadores aritméticos |
|  | Operadores de atribuição |
|  | Operadores de comparação |
|  | Operadores lógicos |
|  | Operadores de identidade |
|  | Operadores de associação |
|  | Operadores bit a bit |
|  | Operadores aritméticos Python |
|  |  |
|  | Os operadores aritméticos são usados com valores numéricos para realizar operações matemáticas comuns: |
|  |  |
|  | Operator Name Example |
|  | + Addition x + y |
|  | - Subtraction x - y |
|  | \* Multiplication x \* y |
|  | / Division x / y |
|  | % Modulus x % y |
|  | \*\* Exponentiation x \*\* y |
|  | // Floor division x // y |
|  |  |
|  | Operadores de atribuição Python |
|  | Os operadores de atribuição são usados para atribuir valores a variáveis: |
|  |  |
|  | Operator Example Same As |
|  | = x = 5 x = 5 |
|  | += x += 3 x = x + 3 |
|  | -= x -= 3 x = x - 3 |
|  | \*= x \*= 3 x = x \* 3 |
|  | /= x /= 3 x = x / 3 |
|  | %= x %= 3 x = x % 3 |
|  | //= x //= 3 x = x // 3 |
|  | \*\*= x \*\*= 3 x = x \*\* 3 |
|  | &= x &= 3 x = x & 3 |
|  | |= x |= 3 x = x | 3 |
|  | ^= x ^= 3 x = x ^ 3 |
|  | >>= x >>= 3 x = x >> 3 |
|  | <<= x <<= 3 x = x << 3 |
|  |  |
|  |  |
|  | Operadores de comparação Python |
|  |  |
|  | Os operadores de comparação são usados para comparar dois valores: |
|  |  |
|  | Operator Name Example |
|  | == Equal x == y |
|  | != Not equal x != y |
|  | > Greater than x > y |
|  | < Less than x < y |
|  | >= Greater than or equal to x >= y |
|  | <= Less than or equal to x <= y |
|  |  |
|  | Operadores lógicos Python |
|  | Os operadores lógicos são usados para combinar instruções condicionais: |
|  |  |
|  | Operator Description Example |
|  | and Returns True if both statements are true x < 5 and x < 10 |
|  | or Returns True if one of the statements is true x < 5 or x < 4 |
|  | not Reverse the result, returns False if the result is true not(x < 5 and x < 10) |
|  |  |
|  | Operadores de identidade Python |
|  | Os operadores de identidade são usados para comparar os objetos, não se forem iguais, mas se forem |
|  | realmente o mesmo objeto, com a mesma localização de memória: |
|  |  |
|  | Operator Description Example |
|  | is Returns True if both variables are the same object x is y |
|  | is not Returns True if both variables are not the same object x is not y |
|  |  |
|  | Operadores de associação Python |
|  | Os operadores de associação são usados para testar se uma sequência é apresentada em um objeto: |
|  |  |
|  | Operator Description Example |
|  | in Returns True if a sequence with the specified value is present in the object x in y |
|  | not in Returns True if a sequence with the specified value is not present in the object x not in y |
|  |  |
|  | Operadores Bitwise Python |
|  | Operadores bit a bit são usados para comparar números (binários): |
|  |  |
|  | Operator Name Description |
|  | & AND Sets each bit to 1 if both bits are 1 |
|  | | OR Sets each bit to 1 if one of two bits is 1 |
|  | ^ XOR Sets each bit to 1 if only one of two bits is 1 |
|  | ~ NOT Inverts all the bits |
|  | << Zero fill left shift Shift left by pushing zeros in from the right and let the leftmost bits fall off |
|  | >> Signed right shift Shift right by pushing copies of the leftmost bit in from the left, |
|  | and let the rightmost bits fall off |
|  |  |
|  |  |
|  | Lista |
|  |  |
|  | As listas são usadas para armazenar vários itens em uma única variável. |
|  |  |
|  | As listas são um dos 4 tipos de dados internos do Python usados para armazenar coleções de dados, |
|  | os outros 3 são Tuple , Set e Dictionary , todos com qualidades e usos diferentes. |
|  |  |
|  | As listas são criadas usando colchetes: |
|  |  |
|  | Exemplo |
|  | Crie uma lista: |
|  |  |
|  | thislist = ["apple", "banana", "cherry"] |
|  | print(thislist) |
|  |  |
|  | lista de itens |
|  | Os itens da lista são ordenados, alteráveis e permitem valores duplicados. |
|  |  |
|  | Os itens da lista são indexados, o primeiro item possui índice [0], o segundo item possui índice [1]etc. |
|  |  |
|  | Encomendado |
|  |  |
|  | Quando dizemos que as listas estão ordenadas, significa que os itens têm uma ordem definida, |
|  | e essa ordem não será alterada. |
|  |  |
|  | Se você adicionar novos itens a uma lista, os novos itens serão colocados no final da lista. |
|  |  |
|  | Nota: Existem alguns métodos de lista que irão alterar a ordem, mas em geral: a ordem dos itens não será alterada. |
|  |  |
|  | Mutável |
|  |  |
|  | A lista é mutável, o que significa que podemos alterar, adicionar e remover itens em uma |
|  | lista após ela ter sido criada. |
|  |  |
|  | Permitir duplicatas |
|  |  |
|  | Como as listas são indexadas, as listas podem ter itens com o mesmo valor: |
|  |  |
|  | Exemplo |
|  |  |
|  | As listas permitem valores duplicados: |
|  |  |
|  | thislist = ["apple", "banana", "cherry", "apple", "cherry"] |
|  | print(thislist) |
|  |  |
|  |  |
|  | Comprimento da lista |
|  |  |
|  | Para determinar quantos itens uma lista possui, use a len()função: |
|  |  |
|  | Exemplo |
|  | Imprima o número de itens na lista: |
|  |  |
|  | thislist = ["apple", "banana", "cherry"] |
|  | print(len(thislist)) |
|  |  |
|  | Listar itens - Tipos de dados |
|  | Os itens da lista podem ser de qualquer tipo de dados: |
|  |  |
|  | Exemplo |
|  |  |
|  | Tipos de dados String, int e boolean: |
|  |  |
|  | list1 = ["apple", "banana", "cherry"] |
|  | list2 = [1, 5, 7, 9, 3] |
|  | list3 = [True, False, False] |
|  |  |
|  | Uma lista pode conter diferentes tipos de dados: |
|  |  |
|  | Exemplo |
|  | Uma lista com strings, números inteiros e valores booleanos: |
|  |  |
|  | list1 = ["abc", 34, True, 40, "male"] |
|  | tipo() |
|  |  |
|  | Da perspectiva do Python, as listas são definidas como objetos com o tipo de dados 'list': |
|  |  |
|  | <class 'list'> |
|  | Exemplo |
|  | Qual é o tipo de dados de uma lista? |
|  |  |
|  | mylist = ["apple", "banana", "cherry"] |
|  | print(type(mylist)) |
|  |  |
|  | O construtor list() |
|  | Também é possível usar o construtor list() ao criar uma nova lista. |
|  |  |
|  | Exemplo |
|  | Usando o list()construtor para fazer uma lista: |
|  |  |
|  | thislist = list(("apple", "banana", "cherry")) # note the double round-brackets |
|  | print(thislist) |
|  |  |
|  | Coleções Python (matrizes) |
|  | Existem quatro tipos de dados de coleção na linguagem de programação Python: |
|  |  |
|  | Lista é uma coleção que é ordenada e mutável. Permite membros duplicados. |
|  | Tupla é uma coleção ordenada e imutável. Permite membros duplicados. |
|  | Set é uma coleção não ordenada, imutável\* e não indexada. Nenhum membro duplicado. |
|  | Dicionário é uma coleção ordenada\*\* e mutável. Nenhum membro duplicado. |
|  |  |
|  | \*Os itens do conjunto são imutáveis, mas você pode remover e/ou adicionar itens sempre que quiser. |
|  |  |
|  | \*\*A partir da versão 3.7 do Python, os dicionários são ordenados . No Python 3.6 e anteriores, os |
|  | dicionários não são ordenados . |
|  |  |
|  | Ao escolher um tipo de coleção, é útil entender as propriedades desse tipo. Escolher o tipo certo |
|  | para um determinado conjunto de dados pode significar retenção de significado e pode significar um |
|  | aumento na eficiência ou segurança. |
|  |  |
|  |  |
|  | Acessar itens |
|  |  |
|  | Os itens da lista são indexados e você pode acessá-los consultando o número do índice: |
|  |  |
|  | Exemplo |
|  | Imprima o segundo item da lista: |
|  |  |
|  | thislist = ["apple", "banana", "cherry"] |
|  | print(thislist[1]) |
|  |  |
|  | Nota: O primeiro item tem índice 0. |
|  |  |
|  | Indexação negativa |
|  |  |
|  | Indexação negativa significa começar do fim |
|  |  |
|  | -1 refere-se ao último item, -2 refere-se ao penúltimo item etc. |
|  |  |
|  | Exemplo |
|  | Imprima o último item da lista: |
|  |  |
|  | thislist = ["apple", "banana", "cherry"] |
|  | print(thislist[-1]) |
|  |  |
|  | Faixa de índices |
|  |  |
|  | Você pode especificar um intervalo de índices especificando onde começar e onde terminar o intervalo. |
|  |  |
|  | Ao especificar um intervalo, o valor de retorno será uma nova lista com os itens especificados. |
|  |  |
|  | Exemplo |
|  | Retorne o terceiro, quarto e quinto item: |
|  |  |
|  | thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"] |
|  | print(thislist[2:5]) |
|  |  |
|  | Nota: A pesquisa começará no índice 2 (incluído) e terminará no índice 5 (não incluído). |
|  |  |
|  | Lembre-se que o primeiro item tem índice 0. |
|  |  |
|  | Ao deixar de fora o valor inicial, o intervalo começará no primeiro item: |
|  |  |
|  | Exemplo |
|  | Este exemplo retorna os itens do início, mas NÃO incluindo, "kiwi": |
|  |  |
|  | thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"] |
|  | print(thislist[:4]) |
|  |  |
|  | Ao deixar de fora o valor final, o intervalo irá para o final da lista: |
|  |  |
|  | Exemplo |
|  | Este exemplo retorna os itens de "cherry" até o final: |
|  |  |
|  | thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"] |
|  | print(thislist[2:]) |
|  |  |
|  | Faixa de índices negativos |
|  |  |
|  | Especifique índices negativos se desejar iniciar a pesquisa a partir do final da lista: |
|  |  |
|  | Exemplo |
|  | Este exemplo retorna os itens de "laranja" (-4) para, mas NÃO incluindo "manga" (-1): |
|  |  |
|  | thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"] |
|  | print(thislist[-4:-1]) |
|  |  |
|  | Verifique se o item existe |
|  |  |
|  | Para determinar se um item especificado está presente em uma lista, use a inpalavra-chave: |
|  |  |
|  | Exemplo |
|  | Verifique se "apple" está presente na lista: |
|  |  |
|  | thislist = ["apple", "banana", "cherry"] |
|  | if "apple" in thislist: |
|  | print("Yes, 'apple' is in the fruits list") |
|  |  |
|  |  |
|  | Alterar valor do item |
|  |  |
|  | Para alterar o valor de um item específico, consulte o número do índice: |
|  |  |
|  | Exemplo |
|  | Altere o segundo item: |
|  |  |
|  | thislist = ["apple", "banana", "cherry"] |
|  | thislist[1] = "blackcurrant" |
|  | print(thislist) |
|  |  |
|  | Alterar um intervalo de valores de itens |
|  |  |
|  | Para alterar o valor dos itens dentro de um intervalo específico, defina uma lista |
|  | com os novos valores e consulte o intervalo de números de índice onde deseja inserir os novos valores: |
|  |  |
|  | Exemplo |
|  | Altere os valores "banana" e "cherry" pelos valores "blackcurrant" e "watermelon": |
|  |  |
|  | thislist = ["apple", "banana", "cherry", "orange", "kiwi", "mango"] |
|  | thislist[1:3] = ["blackcurrant", "watermelon"] |
|  | print(thislist) |
|  |  |
|  | Se você inserir mais itens do que substituir, os novos itens serão inseridos onde você |
|  | especificou e os itens restantes serão movidos de acordo: |
|  |  |
|  | Exemplo |
|  |  |
|  | Altere o segundo valor substituindo-o por dois novos valores: |
|  |  |
|  | thislist = ["apple", "banana", "cherry"] |
|  | thislist[1:2] = ["blackcurrant", "watermelon"] |
|  | print(thislist) |
|  |  |
|  | Nota: O comprimento da lista mudará quando o número de itens inseridos não corresponder |
|  | ao número de itens substituídos. |
|  |  |
|  | Se você inserir menos itens do que substituir, os novos itens serão inseridos onde você |
|  | especificou e os itens restantes serão movidos de acordo: |
|  |  |
|  | Exemplo |
|  | Altere o segundo e o terceiro valor substituindo-o por um valor: |
|  |  |
|  | thislist = ["apple", "banana", "cherry"] |
|  | thislist[1:3] = ["watermelon"] |
|  | print(thislist) |
|  |  |
|  |  |
|  | Inserir itens |
|  | Para inserir um novo item de lista, sem substituir nenhum dos valores existentes, podemos usar o insert()método. |
|  |  |
|  | O insert()método insere um item no índice especificado: |
|  |  |
|  | Exemplo |
|  | Insira "melancia" como terceiro item: |
|  |  |
|  | thislist = ["apple", "banana", "cherry"] |
|  | thislist.insert(2, "watermelon") |
|  | print(thislist) |
|  |  |
|  |  |
|  | Anexar itens |
|  |  |
|  | Para adicionar um item ao final da lista, use o método append() : |
|  |  |
|  | Exemplo |
|  | Usando o append()método para anexar um item: |
|  |  |
|  | thislist = ["apple", "banana", "cherry"] |
|  | thislist.append("orange") |
|  | print(thislist) |
|  |  |
|  | Inserir itens |
|  | Para inserir um item de lista em um índice especificado, use o insert()método. |
|  |  |
|  | O insert()método insere um item no índice especificado: |
|  |  |
|  | Exemplo |
|  | Insira um item como a segunda posição: |
|  |  |
|  | thislist = ["apple", "banana", "cherry"] |
|  | thislist.insert(1, "orange") |
|  | print(thislist) |
|  |  |
|  | Nota: Como resultado dos exemplos acima, as listas agora conterão 4 itens. |
|  |  |
|  | Estender lista |
|  | Para anexar elementos de outra lista à lista atual, use o extend()método. |
|  |  |
|  | Exemplo |
|  | Adicione os elementos de tropicala thislist: |
|  |  |
|  | thislist = ["apple", "banana", "cherry"] |
|  | tropical = ["mango", "pineapple", "papaya"] |
|  | thislist.extend(tropical) |
|  | print(thislist) |
|  |  |
|  | Os elementos serão adicionados ao final da lista. |
|  |  |
|  | Adicionar qualquer iterável |
|  | O extend()método não precisa anexar listas , você pode adicionar qualquer objeto |
|  | iterável (tuplas, conjuntos, dicionários etc.). |
|  |  |
|  | Exemplo |
|  | Adicionar elementos de uma tupla a uma lista: |
|  |  |
|  | thislist = ["apple", "banana", "cherry"] |
|  | thistuple = ("kiwi", "orange") |
|  | thislist.extend(thistuple) |
|  | print(thislist) |
|  |  |
|  | Remover item especificado |
|  | O remove()método remove o item especificado. |
|  |  |
|  | Exemplo |
|  | Remova a "banana": |
|  |  |
|  | thislist = ["apple", "banana", "cherry"] |
|  | thislist.remove("banana") |
|  | print(thislist) |
|  |  |
|  | Remover índice especificado |
|  | O pop()método remove o índice especificado. |
|  |  |
|  | Exemplo |
|  | Remova o segundo item: |
|  |  |
|  | thislist = ["apple", "banana", "cherry"] |
|  | thislist.pop(1) |
|  | print(thislist) |
|  |  |
|  | Se você não especificar o índice, o pop()método removerá o último item. |
|  |  |
|  | Exemplo |
|  | Remova o último item: |
|  |  |
|  | thislist = ["apple", "banana", "cherry"] |
|  | thislist.pop() |
|  | print(thislist) |
|  |  |
|  | A palavra- delchave também remove o índice especificado: |
|  |  |
|  | Exemplo |
|  | Remova o primeiro item: |
|  |  |
|  | thislist = ["apple", "banana", "cherry"] |
|  | del thislist[0] |
|  | print(thislist) |
|  |  |
|  | A delpalavra-chave também pode excluir a lista completamente. |
|  |  |
|  | Exemplo |
|  | Apague a lista inteira: |
|  |  |
|  | thislist = ["apple", "banana", "cherry"] |
|  | del thislist |
|  |  |
|  | Limpar a lista |
|  | O clear()método esvazia a lista. |
|  |  |
|  | A lista ainda permanece, mas não tem conteúdo. |
|  |  |
|  | Exemplo |
|  | Limpe o conteúdo da lista: |
|  |  |
|  | thislist = ["apple", "banana", "cherry"] |
|  | thislist.clear() |
|  | print(thislist) |
|  |  |
|  | Percorrer uma lista |
|  |  |
|  | Você pode percorrer os itens da lista usando um for loop: |
|  |  |
|  | Exemplo |
|  | Imprima todos os itens da lista, um por um: |
|  |  |
|  | thislist = ["apple", "banana", "cherry"] |
|  | for x in thislist: |
|  | print(x) |
|  |  |
|  | Percorrer os números de índice |
|  | Você também pode percorrer os itens da lista consultando seu número de índice. |
|  |  |
|  | Use as funções range()e len()para criar um iterável adequado. |
|  |  |
|  | Exemplo |
|  | Imprima todos os itens consultando seu número de índice: |
|  |  |
|  | thislist = ["apple", "banana", "cherry"] |
|  | for i in range(len(thislist)): |
|  | print(thislist[i]) |
|  |  |
|  | O iterável criado no exemplo acima é [0, 1, 2]. |
|  |  |
|  | Usando um loop while |
|  | Você pode percorrer os itens da lista usando um whileloop. |
|  |  |
|  | Use a len()função para determinar o comprimento da lista, então comece em 0 e faça um |
|  | loop pelos itens da lista consultando seus índices. |
|  |  |
|  | Lembre-se de aumentar o índice em 1 após cada iteração. |
|  |  |
|  | Exemplo |
|  | Imprima todos os itens, usando um whileloop para percorrer todos os números de índice |
|  |  |
|  | thislist = ["apple", "banana", "cherry"] |
|  | i = 0 |
|  | while i < len(thislist): |
|  | print(thislist[i]) |
|  | i = i + 1 |
|  |  |
|  | Loop usando compreensão de lista |
|  | O List Compreension oferece a sintaxe mais curta para percorrer listas: |
|  |  |
|  | Exemplo |
|  | Um forloop de mão curto que imprimirá todos os itens em uma lista: |
|  |  |
|  | thislist = ["apple", "banana", "cherry"] |
|  | [print(x) for x in thislist] |
|  |  |
|  |  |
|  | Compreensão da lista |
|  | A compreensão de lista oferece uma sintaxe mais curta quando você deseja criar uma nova lista |
|  | com base nos valores de uma lista existente. |
|  |  |
|  | Exemplo: |
|  |  |
|  | Com base em uma lista de frutas, você deseja uma nova lista, contendo apenas as frutas com a letra "a" no nome. |
|  |  |
|  | Sem compreensão de lista, você terá que escrever uma fordeclaração com um teste condicional dentro: |
|  |  |
|  | Exemplo |
|  | fruits = ["apple", "banana", "cherry", "kiwi", "mango"] |
|  | newlist = [] |
|  |  |
|  | for x in fruits: |
|  | if "a" in x: |
|  | newlist.append(x) |
|  |  |
|  | print(newlist) |
|  |  |
|  | Com compreensão de lista, você pode fazer tudo isso com apenas uma linha de código: |
|  |  |
|  | Exemplo |
|  | fruits = ["apple", "banana", "cherry", "kiwi", "mango"] |
|  |  |
|  | newlist = [x for x in fruits if "a" in x] |
|  |  |
|  | print(newlist) |
|  |  |
|  |  |
|  | A Sintaxe |
|  | newlist = [expression for item in iterable if condition == True] |
|  | O valor de retorno é uma nova lista, deixando a lista antiga inalterada. |
|  |  |
|  | Doença |
|  | A condição é como um filtro que aceita apenas os itens que avaliam como True. |
|  |  |
|  | Exemplo |
|  | Aceite apenas itens que não sejam "maçã": |
|  |  |
|  | newlist = [x for x in fruits if x != "apple"] |
|  | A condição if x != "apple" retornará True para todos os elementos exceto "maçã", |
|  | fazendo com que a nova lista contenha todas as frutas, exceto "maçã". |
|  |  |
|  | A condição é opcional e pode ser omitida: |
|  |  |
|  | Exemplo |
|  | Sem if declaração: |
|  |  |
|  | newlist = [x for x in fruits] |
|  |  |
|  | Iterável |
|  | O iterável pode ser qualquer objeto iterável, como uma lista, tupla, conjunto etc. |
|  |  |
|  | Exemplo |
|  | Você pode usar a range()função para criar um iterável: |
|  |  |
|  | newlist = [x for x in range(10)] |
|  | Mesmo exemplo, mas com uma condição: |
|  |  |
|  | Exemplo |
|  | Aceite apenas números inferiores a 5: |
|  |  |
|  | newlist = [x for x in range(10) if x < 5] |
|  |  |
|  | Expressão |
|  | A expressão é o item atual na iteração, mas também é o resultado, que você pode manipular |
|  | antes que termine como um item de lista na nova lista: |
|  |  |
|  | Exemplo |
|  | Defina os valores na nova lista para maiúsculas: |
|  |  |
|  | newlist = [x.upper() for x in fruits] |
|  |  |
|  | Você pode definir o resultado para o que quiser: |
|  |  |
|  | Exemplo |
|  | Defina todos os valores na nova lista para 'hello': |
|  |  |
|  | newlist = ['hello' for x in fruits] |
|  |  |
|  | A expressão também pode conter condições, não como um filtro, mas como forma de manipular o resultado: |
|  |  |
|  | Exemplo |
|  | Retorne "laranja" em vez de "banana": |
|  |  |
|  | newlist = [x if x != "banana" else "orange" for x in fruits] |
|  |  |
|  |  |
|  | Classificar lista alfanumérica |
|  |  |
|  | Objetos de lista possuem um sort()método que ordenará a lista de forma alfanumérica, em ordem crescente, por padrão: |
|  |  |
|  | Exemplo |
|  | Classifique a lista em ordem alfabética: |
|  |  |
|  | thislist = ["orange", "mango", "kiwi", "pineapple", "banana"] |
|  | thislist.sort() |
|  | print(thislist) |
|  |  |
|  | Exemplo |
|  | Classifique a lista numericamente: |
|  |  |
|  | thislist = [100, 50, 65, 82, 23] |
|  | thislist.sort() |
|  | print(thislist) |
|  |  |
|  | Classificação decrescente |
|  | Para classificar de forma decrescente, use o argumento de palavra-chave reverse = True: |
|  |  |
|  | Exemplo |
|  | Ordene a lista decrescente: |
|  |  |
|  | thislist = ["orange", "mango", "kiwi", "pineapple", "banana"] |
|  | thislist.sort(reverse = True) |
|  | print(thislist) |
|  |  |
|  | Exemplo |
|  | Ordene a lista decrescente: |
|  |  |
|  | thislist = [100, 50, 65, 82, 23] |
|  | thislist.sort(reverse = True) |
|  | print(thislist) |
|  |  |
|  | Personalizar a função de classificação |
|  |  |
|  | Você também pode personalizar sua própria função usando o argumento de palavra-chave .key = function |
|  |  |
|  | A função retornará um número que será usado para ordenar a lista (o número mais baixo primeiro): |
|  |  |
|  | Exemplo |
|  | Classifique a lista com base em quão próximo o número está de 50: |
|  |  |
|  | def myfunc(n): |
|  | return abs(n - 50) |
|  |  |
|  | thislist = [100, 50, 65, 82, 23] |
|  | thislist.sort(key = myfunc) |
|  | print(thislist) |
|  |  |
|  | Classificação sem distinção entre maiúsculas e minúsculas |
|  |  |
|  | Por padrão, o sort()método diferencia maiúsculas de minúsculas, resultando em todas as letras |
|  | maiúsculas classificadas antes das minúsculas: |
|  |  |
|  | Exemplo |
|  |  |
|  | A classificação com distinção entre maiúsculas e minúsculas pode fornecer um resultado inesperado: |
|  |  |
|  | thislist = ["banana", "Orange", "Kiwi", "cherry"] |
|  | thislist.sort() |
|  | print(thislist) |
|  |  |
|  | Felizmente, podemos usar funções internas como funções-chave ao classificar uma lista. |
|  |  |
|  | Portanto, se você deseja uma função de classificação que não diferencia maiúsculas de minúsculas, use |
|  | str.lower como uma função de chave: |
|  |  |
|  | Exemplo |
|  |  |
|  | Execute uma classificação que não diferencia maiúsculas de minúsculas da lista: |
|  |  |
|  | thislist = ["banana", "Orange", "Kiwi", "cherry"] |
|  | thislist.sort(key = str.lower) |
|  | print(thislist) |
|  |  |
|  | Ordem reversa |
|  |  |
|  | E se você quiser inverter a ordem de uma lista, independentemente do alfabeto? |
|  |  |
|  | O reverse()método inverte a ordem de classificação atual dos elementos. |
|  |  |
|  | Exemplo |
|  | Inverta a ordem dos itens da lista: |
|  |  |
|  | thislist = ["banana", "Orange", "Kiwi", "cherry"] |
|  | thislist.reverse() |
|  | print(thislist) |
|  |  |
|  | Copiar uma lista |
|  | Você não pode copiar uma lista simplesmente digitando list2 = list1, porque: list2será apenas uma |
|  | referência a list1, e as alterações feitas list1automaticamente também serão feitas em list2. |
|  |  |
|  | Existem maneiras de fazer uma cópia, uma delas é usar o método List interno copy(). |
|  |  |
|  | Exemplo |
|  | Faça uma cópia de uma lista com o copy()método: |
|  |  |
|  | thislist = ["apple", "banana", "cherry"] |
|  | mylist = thislist.copy() |
|  | print(mylist) |
|  |  |
|  | Outra maneira de fazer uma cópia é usar o método interno list(). |
|  |  |
|  | Exemplo |
|  | Faça uma cópia de uma lista com o list()método: |
|  |  |
|  | thislist = ["apple", "banana", "cherry"] |
|  | mylist = list(thislist) |
|  | print(mylist) |
|  |  |
|  | Junte-se a duas listas |
|  | Existem várias maneiras de unir ou concatenar duas ou mais listas em Python. |
|  |  |
|  | Uma das maneiras mais fáceis é usando o + operador. |
|  |  |
|  | Exemplo |
|  | Junte-se a duas listas: |
|  |  |
|  | list1 = ["a", "b", "c"] |
|  | list2 = [1, 2, 3] |
|  |  |
|  | list3 = list1 + list2 |
|  | print(list3) |
|  | Outra maneira de juntar duas listas é anexando todos os itens da lista2 na lista1, um por um: |
|  |  |
|  | Exemplo |
|  | Anexar list2 em list1: |
|  |  |
|  | list1 = ["a", "b" , "c"] |
|  | list2 = [1, 2, 3] |
|  |  |
|  | for x in list2: |
|  | list1.append(x) |
|  |  |
|  | print(list1) |
|  | Ou você pode usar o extend() método, cuja finalidade é adicionar elementos de uma lista para outra lista: |
|  |  |
|  | Exemplo |
|  | Use o extend()método para adicionar list2 no final de list1: |
|  |  |
|  | list1 = ["a", "b" , "c"] |
|  | list2 = [1, 2, 3] |
|  |  |
|  | list1.extend(list2) |
|  | print(list1) |
|  |  |
|  | Listar métodos |
|  | Python tem um conjunto de métodos integrados que você pode usar em listas. |
|  |  |
|  | Method Description |
|  | append() Adds an element at the end of the list |
|  | clear() Removes all the elements from the list |
|  | copy() Returns a copy of the list |
|  | count() Returns the number of elements with the specified value |
|  | extend() Add the elements of a list (or any iterable), to the end of the current list |
|  | index() Returns the index of the first element with the specified value |
|  | insert() Adds an element at the specified position |
|  | pop() Removes the element at the specified position |
|  | remove() Removes the item with the specified value |
|  | reverse() Reverses the order of the list |
|  | sort() Sorts the list |
|  |  |
|  | Tuplas Python |
|  |  |
|  | mytuple = ("apple", "banana", "cherry") |
|  |  |
|  | Tupla |
|  | Tuplas são usadas para armazenar vários itens em uma única variável. |
|  |  |
|  | Tuple é um dos 4 tipos de dados internos do Python usados para armazenar |
|  | coleções de dados, os outros 3 são List , Set e Dictionary , todos com qualidades e usos diferentes. |
|  |  |
|  | Uma tupla é uma coleção ordenada e imutável . |
|  |  |
|  | Tuplas são escritas com colchetes. |
|  |  |
|  | Exemplo |
|  | Crie uma Tupla: |
|  |  |
|  | thistuple = ("apple", "banana", "cherry") |
|  | print(thistuple) |
|  |  |
|  | Itens de Tupla |
|  | Os itens de tupla são ordenados, imutáveis e permitem valores duplicados. |
|  |  |
|  | Os itens de tupla são indexados, o primeiro item possui índice [0], o segundo item possui índice [1]etc. |
|  |  |
|  | Encomendado |
|  | Quando dizemos que as tuplas estão ordenadas, significa que os itens têm uma ordem definida, e essa ordem |
|  | não será alterada. |
|  |  |
|  | Imutável |
|  | As tuplas são imutáveis, o que significa que não podemos alterar, adicionar ou remover itens após a criação da tupla. |
|  |  |
|  | Permitir duplicatas |
|  | Como as tuplas são indexadas, elas podem ter itens com o mesmo valor: |
|  |  |
|  | Exemplo |
|  | Tuplas permitem valores duplicados: |
|  |  |
|  | thistuple = ("apple", "banana", "cherry", "apple", "cherry") |
|  | print(thistuple) |
|  |  |
|  |  |
|  | Comprimento da Tupla |
|  | Para determinar quantos itens uma tupla possui, use a len()função: |
|  |  |
|  | Exemplo |
|  | Imprima o número de itens na tupla: |
|  |  |
|  | thistuple = ("apple", "banana", "cherry") |
|  | print(len(thistuple)) |
|  |  |
|  | Criar Tupla Com Um Item |
|  | Para criar uma tupla com apenas um item, você deve adicionar uma vírgula após o item, caso contrário, |
|  | o Python não o reconhecerá como uma tupla. |
|  |  |
|  | Exemplo |
|  | Uma tupla de item, lembre-se da vírgula: |
|  |  |
|  | thistuple = ("apple",) |
|  | print(type(thistuple)) |
|  |  |
|  | #NOT a tuple |
|  |  |
|  | thistuple = ("apple") |
|  | print(type(thistuple)) |
|  |  |
|  | Itens de Tupla - Tipos de Dados |
|  | Os itens de tupla podem ser de qualquer tipo de dados: |
|  |  |
|  | Exemplo |
|  | Tipos de dados String, int e boolean: |
|  |  |
|  | tuple1 = ("apple", "banana", "cherry") |
|  | tuple2 = (1, 5, 7, 9, 3) |
|  | tuple3 = (True, False, False) |
|  |  |
|  | Uma tupla pode conter diferentes tipos de dados: |
|  |  |
|  | Exemplo |
|  | Uma tupla com strings, inteiros e valores booleanos: |
|  |  |
|  | tuple1 = ("abc", 34, True, 40, "male") |
|  | tipo() |
|  |  |
|  | Da perspectiva do Python, as tuplas são definidas como objetos com o tipo de dados 'tupla': |
|  |  |
|  | <class 'tuple'> |
|  | Exemplo |
|  | Qual é o tipo de dados de uma tupla? |
|  |  |
|  | mytuple = ("apple", "banana", "cherry") |
|  | print(type(mytuple)) |
|  | O construtor tupla() |
|  |  |
|  | Também é possível usar o construtor tuple() para fazer uma tupla. |
|  |  |
|  | Exemplo |
|  | Usando o método tuple() para fazer uma tupla: |
|  |  |
|  | thistuple = tuple(("apple", "banana", "cherry")) # note the double round-brackets |
|  | print(thistuple) |
|  |  |
|  | Coleções Python (matrizes) |
|  | Existem quatro tipos de dados de coleção na linguagem de programação Python: |
|  |  |
|  | Lista é uma coleção que é ordenada e mutável. Permite membros duplicados. |
|  | Tupla é uma coleção ordenada e imutável. Permite membros duplicados. |
|  | Set é uma coleção não ordenada, imutável\* e não indexada. Nenhum membro duplicado. |
|  | Dicionário é uma coleção ordenada\*\* e mutável. Nenhum membro duplicado. |
|  |  |
|  | \*Os itens do conjunto são imutáveis, mas você pode remover e/ou adicionar itens sempre que quiser. |
|  |  |
|  | \*\*A partir da versão 3.7 do Python, os dicionários são ordenados . No Python 3.6 e anteriores, |
|  | os dicionários não são ordenados . |
|  |  |
|  | Ao escolher um tipo de coleção, é útil entender as propriedades desse tipo. Escolher o tipo certo para |
|  | um determinado conjunto de dados pode significar retenção de significado e pode significar um aumento na |
|  | eficiência ou segurança. |
|  |  |
|  | Acessar itens de tupla |
|  |  |
|  | Você pode acessar os itens da tupla consultando o número do índice, entre colchetes: |
|  |  |
|  | Exemplo |
|  | Imprima o segundo item na tupla: |
|  |  |
|  | thistuple = ("apple", "banana", "cherry") |
|  | print(thistuple[1]) |
|  |  |
|  | Nota: O primeiro item tem índice 0. |
|  |  |
|  | Indexação negativa |
|  | Indexação negativa significa começar do fim. |
|  |  |
|  | -1refere-se ao último item, -2refere-se ao penúltimo item etc. |
|  |  |
|  | Exemplo |
|  | Imprima o último item da tupla: |
|  |  |
|  | thistuple = ("apple", "banana", "cherry") |
|  | print(thistuple[-1]) |
|  |  |
|  | Faixa de índices |
|  | Você pode especificar um intervalo de índices especificando onde começar e onde terminar o intervalo. |
|  |  |
|  | Ao especificar um intervalo, o valor de retorno será uma nova tupla com os itens especificados. |
|  |  |
|  | Exemplo |
|  | Retorne o terceiro, quarto e quinto item: |
|  |  |
|  | thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango") |
|  | print(thistuple[2:5]) |
|  |  |
|  | Nota: A pesquisa começará no índice 2 (incluído) e terminará no índice 5 (não incluído). |
|  |  |
|  | Lembre-se que o primeiro item tem índice 0. |
|  |  |
|  | Ao deixar de fora o valor inicial, o intervalo começará no primeiro item: |
|  |  |
|  | Exemplo |
|  | Este exemplo retorna os itens do início para, mas NÃO inclui, "kiwi": |
|  |  |
|  | thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango") |
|  | print(thistuple[:4]) |
|  |  |
|  | Ao deixar de fora o valor final, o intervalo irá para o final da lista: |
|  |  |
|  | Exemplo |
|  | Este exemplo retorna os itens de "cherry" e até o final: |
|  |  |
|  | thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango") |
|  | print(thistuple[2:]) |
|  |  |
|  | As tuplas são imutáveis, o que significa que você não pode alterar, adicionar ou remover itens depois |
|  | que a tupla for criada. |
|  |  |
|  | Mas existem algumas soluções alternativas. |
|  |  |
|  | Valores mudança Tuple |
|  | Depois que uma tupla é criada, você não pode alterar seus valores. Tuplas são imutáveis , ou imutáveis , |
|  | como também é chamado. |
|  |  |
|  | Mas há uma solução alternativa. Você pode converter a tupla em uma lista, alterar a lista e convertê-la |
|  | novamente em uma tupla. |
|  |  |
|  | Exemplo |
|  | Converta a tupla em uma lista para poder alterá-la: |
|  |  |
|  | x = ("apple", "banana", "cherry") |
|  | y = list(x) |
|  | y[1] = "kiwi" |
|  | x = tuple(y) |
|  |  |
|  | print(x) |
|  |  |
|  | Adicionar itens |
|  | Como as tuplas são imutáveis, elas não possuem um append()método embutido, mas existem outras maneiras de |
|  | adicionar itens a uma tupla. |
|  |  |
|  | 1. Converter em uma lista : Assim como a solução alternativa para alterar uma tupla, você pode convertê-la |
|  | em uma lista, adicionar seu(s) item(ns) e convertê-la novamente em uma tupla. |
|  |  |
|  | Exemplo |
|  | Converter a tupla em uma lista, adicione "laranja", e convertê-lo de volta em uma tupla: |
|  |  |
|  | thistuple = ("apple", "banana", "cherry") |
|  | y = list(thistuple) |
|  | y.append("orange") |
|  | thistuple = tuple(y) |
|  |  |
|  | 2. Adicionar tupla a uma tupla . Você tem permissão para adicionar tuplas a tuplas, portanto, se você quiser |
|  | adicionar um item (ou muitos), crie uma nova tupla com o(s) item(ns) e adicione-a à tupla existente: |
|  |  |
|  | Exemplo |
|  | Crie uma nova tupla com o valor "laranja" e adicione essa tupla: |
|  |  |
|  | thistuple = ("apple", "banana", "cherry") |
|  | y = ("orange",) |
|  | thistuple += y |
|  |  |
|  | print(thistuple) |
|  |  |
|  | Nota: Ao criar uma tupla com apenas um item, lembre-se de incluir uma vírgula após o item, caso contrário ele |
|  | não será identificado como uma tupla. |
|  |  |
|  |  |
|  | Remover itens |
|  | Nota: Você não pode remover itens em uma tupla. |
|  |  |
|  | As tuplas não podem ser alteradas , portanto, você não pode remover itens dela, mas pode usar a mesma solução |
|  | alternativa que usamos para alterar e adicionar itens de tupla: |
|  |  |
|  | Exemplo |
|  | Converter a tupla em uma lista, remova "maçã", e convertê-lo de volta em uma tupla: |
|  |  |
|  | thistuple = ("apple", "banana", "cherry") |
|  | y = list(thistuple) |
|  | y.remove("apple") |
|  | thistuple = tuple(y) |
|  |  |
|  | Ou você pode excluir a tupla completamente: |
|  |  |
|  | Exemplo |
|  | A delpalavra-chave pode excluir a tupla completamente: |
|  |  |
|  | thistuple = ("apple", "banana", "cherry") |
|  | del thistuple |
|  | print(thistuple) #this will raise an error because the tuple no longer exists |
|  |  |
|  | Descompactando uma Tupla |
|  |  |
|  | Quando criamos uma tupla, normalmente atribuímos valores a ela. Isso é chamado de "empacotar" uma tupla: |
|  |  |
|  | Exemplo |
|  | Empacotando uma tupla: |
|  |  |
|  | fruits = ("apple", "banana", "cherry") |
|  | Mas, em Python, também podemos extrair os valores de volta para as variáveis. Isso é chamado de "descompactar": |
|  |  |
|  | Exemplo |
|  | Descompactando uma tupla: |
|  |  |
|  | fruits = ("apple", "banana", "cherry") |
|  |  |
|  | (green, yellow, red) = fruits |
|  |  |
|  | print(green) |
|  | print(yellow) |
|  | print(red) |
|  |  |
|  | Nota: O número de variáveis deve corresponder ao número de valores na tupla, caso contrário, você deve usar um |
|  | asterisco para coletar os valores restantes como uma lista. |
|  |  |
|  |  |
|  | Usando asterisco\* |
|  | Se o número de variáveis for menor que o número de valores, você pode adicionar um \* ao nome da variável e os |
|  | valores serão atribuídos à variável como uma lista: |
|  |  |
|  | Exemplo |
|  | Atribua o restante dos valores como uma lista chamada "red": |
|  |  |
|  | fruits = ("apple", "banana", "cherry", "strawberry", "raspberry") |
|  |  |
|  | (green, yellow, \*red) = fruits |
|  |  |
|  | print(green) |
|  | print(yellow) |
|  | print(red) |
|  |  |
|  | Se o asterisco for adicionado a outro nome de variável que não o último, o Python atribuirá valores à variável |
|  | até que o número de valores restantes corresponda ao número de variáveis restantes. |
|  |  |
|  | Exemplo |
|  | Adicione uma lista de valores a variável "tropic": |
|  |  |
|  | fruits = ("apple", "mango", "papaya", "pineapple", "cherry") |
|  |  |
|  | (green, \*tropic, red) = fruits |
|  |  |
|  | print(green) |
|  | print(tropic) |
|  | print(red) |
|  |  |
|  |  |
|  | Percorrer uma Tupla |
|  |  |
|  | Você pode percorrer os itens da tupla usando um forloop. |
|  |  |
|  | Exemplo |
|  | Iterar pelos itens e imprimir os valores: |
|  |  |
|  | thistuple = ("apple", "banana", "cherry") |
|  | for x in thistuple: |
|  | print(x) |
|  |  |
|  | Percorrer os números de índice |
|  | Você também pode percorrer os itens da tupla consultando seu número de índice. |
|  |  |
|  | Use as funções range()e len()para criar um iterável adequado. |
|  |  |
|  | Exemplo |
|  | Imprima todos os itens consultando seu número de índice: |
|  |  |
|  | thistuple = ("apple", "banana", "cherry") |
|  | for i in range(len(thistuple)): |
|  | print(thistuple[i]) |
|  |  |
|  |  |
|  | Usando um loop while |
|  | Você pode percorrer os itens da lista usando um whileloop. |
|  |  |
|  | Use a len()função para determinar o comprimento da tupla, então comece em 0 e faça um loop |
|  | pelos itens da tupla consultando seus índices. |
|  |  |
|  | Lembre-se de aumentar o índice em 1 após cada iteração. |
|  |  |
|  | Exemplo |
|  | Imprima todos os itens, usando um whileloop para percorrer todos os números de índice: |
|  |  |
|  | thistuple = ("apple", "banana", "cherry") |
|  | i = 0 |
|  | while i < len(thistuple): |
|  | print(thistuple[i]) |
|  | i = i + 1 |
|  |  |
|  | Junte-se a duas tuplas |
|  |  |
|  | Para juntar duas ou mais tuplas você pode usar o + operador: |
|  |  |
|  | Exemplo |
|  | Junte duas tuplas: |
|  |  |
|  | tuple1 = ("a", "b" , "c") |
|  | tuple2 = (1, 2, 3) |
|  |  |
|  | tuple3 = tuple1 + tuple2 |
|  | print(tuple3) |
|  |  |
|  | Multiplicar Tuplas |
|  | Se você quiser multiplicar o conteúdo de uma tupla um determinado número de vezes, você pode usar o \* operador: |
|  |  |
|  | Exemplo |
|  | Multiplique a tupla de frutas por 2: |
|  |  |
|  | fruits = ("apple", "banana", "cherry") |
|  | mytuple = fruits \* 2 |
|  |  |
|  | print(mytuple) |
|  |  |
|  | Métodos de Tupla |
|  | Python tem dois métodos integrados que você pode usar em tuplas. |
|  |  |
|  | Method Description |
|  | count() Returns the number of times a specified value occurs in a tuple |
|  | index() Searches the tuple for a specified value and returns the position of where it was found |
|  |  |
|  | Conjuntos Python |
|  |  |
|  | myset = {"apple", "banana", "cherry"} |
|  |  |
|  | Definir |
|  | Os conjuntos são usados para armazenar vários itens em uma única variável. |
|  |  |
|  | Set é um dos 4 tipos de dados internos do Python usados para armazenar coleções de dados, os outros 3 são List , |
|  | Tuple e Dictionary , todos com qualidades e usos diferentes. |
|  |  |
|  | Um conjunto é uma coleção não ordenada , imutável\* e não indexada . |
|  |  |
|  | \* Nota: Os itens definidos não podem ser alterados, mas você pode remover itens e adicionar novos itens. |
|  |  |
|  | Os conjuntos são escritos com colchetes. |
|  |  |
|  | Exemplo |
|  | Crie um conjunto: |
|  |  |
|  | thisset = {"apple", "banana", "cherry"} |
|  | print(thisset) |
|  | Observação: os conjuntos não são ordenados, portanto, você não pode ter certeza em qual ordem os itens aparecerão. |
|  |  |
|  | Definir itens |
|  | Os itens do conjunto não são ordenados, não podem ser alterados e não permitem valores duplicados. |
|  |  |
|  | Não ordenado |
|  | Não ordenado significa que os itens em um conjunto não têm uma ordem definida. |
|  |  |
|  | Os itens do conjunto podem aparecer em uma ordem diferente toda vez que você os usa e não podem ser referenciados |
|  | por índice ou chave. |
|  |  |
|  | Imutável |
|  | Os itens do conjunto são imutáveis, o que significa que não podemos alterar os itens após a criação do conjunto. |
|  |  |
|  | Depois que um conjunto é criado, você não pode alterar seus itens, mas pode remover itens e adicionar novos itens. |
|  |  |
|  | Duplicatas não permitidas |
|  | Os conjuntos não podem ter dois itens com o mesmo valor. |
|  |  |
|  | Exemplo |
|  | Valores duplicados serão ignorados: |
|  |  |
|  | thisset = {"apple", "banana", "cherry", "apple"} |
|  |  |
|  | print(thisset) |
|  |  |
|  |  |
|  | Obter o comprimento de um conjunto |
|  | Para determinar quantos itens um conjunto possui, use a len() função. |
|  |  |
|  | Exemplo |
|  | Obtenha o número de itens em um conjunto: |
|  |  |
|  | thisset = {"apple", "banana", "cherry"} |
|  |  |
|  | print(len(thisset)) |
|  |  |
|  | Definir itens - Tipos de dados |
|  | Os itens do conjunto podem ser de qualquer tipo de dados: |
|  |  |
|  | Exemplo |
|  | Tipos de dados String, int e boolean: |
|  |  |
|  | set1 = {"apple", "banana", "cherry"} |
|  | set2 = {1, 5, 7, 9, 3} |
|  | set3 = {True, False, False} |
|  | Um conjunto pode conter diferentes tipos de dados: |
|  |  |
|  | Exemplo |
|  | Um conjunto com strings, inteiros e valores booleanos: |
|  |  |
|  | set1 = {"abc", 34, True, 40, "male"} |
|  | tipo() |
|  | Da perspectiva do Python, os conjuntos são definidos como objetos com o tipo de dados 'set': |
|  |  |
|  | <class 'set'> |
|  | Exemplo |
|  | Qual é o tipo de dado de um conjunto? |
|  |  |
|  | myset = {"apple", "banana", "cherry"} |
|  | print(type(myset)) |
|  | O construtor set() |
|  | Também é possível usar o construtor set() para fazer um conjunto. |
|  |  |
|  | Exemplo |
|  | Usando o construtor set() para fazer um conjunto: |
|  |  |
|  | thisset = set(("apple", "banana", "cherry")) # note the double round-brackets |
|  | print(thisset) |
|  | Coleções Python (matrizes) |
|  | Existem quatro tipos de dados de coleção na linguagem de programação Python: |
|  |  |
|  | Lista é uma coleção que é ordenada e mutável. Permite membros duplicados. |
|  | Tupla é uma coleção ordenada e imutável. Permite membros duplicados. |
|  | Set é uma coleção não ordenada, imutável\* e não indexada. Nenhum membro duplicado. |
|  | Dicionário é uma coleção ordenada\*\* e mutável. Nenhum membro duplicado. |
|  | \*Os itens do conjunto não podem ser alterados, mas você pode remover itens e adicionar novos itens. |
|  |  |
|  | \*\*A partir da versão 3.7 do Python, os dicionários são ordenados . No Python 3.6 e anteriores, os |
|  | dicionários não são ordenados . |
|  |  |
|  | Ao escolher um tipo de coleção, é útil entender as propriedades desse tipo. Escolher o tipo certo |
|  | para um determinado conjunto de dados pode significar retenção de significado e pode significar um |
|  | aumento na eficiência ou segurança. |
|  |  |
|  |  |
|  | Acessar itens |
|  |  |
|  | Você não pode acessar itens em um conjunto fazendo referência a um índice ou a uma chave. |
|  |  |
|  | Mas você pode percorrer os itens do conjunto usando um for loop ou perguntar se um valor |
|  | especificado está presente em um conjunto, usando a palavra- inchave. |
|  |  |
|  | Exemplo |
|  | Percorra o conjunto e imprima os valores: |
|  |  |
|  | thisset = {"apple", "banana", "cherry"} |
|  |  |
|  | for x in thisset: |
|  | print(x) |
|  |  |
|  | Exemplo |
|  | Verifique se "banana" está presente no conjunto: |
|  |  |
|  | thisset = {"apple", "banana", "cherry"} |
|  |  |
|  | print("banana" in thisset) |
|  |  |
|  | Adicionar itens |
|  |  |
|  | Depois que um conjunto é criado, você não pode alterar seus itens, mas pode adicionar novos itens. |
|  |  |
|  | Para adicionar um item a um conjunto, use o add() método. |
|  |  |
|  | Exemplo |
|  | Adicione um item a um conjunto, usando o add() método: |
|  |  |
|  | thisset = {"apple", "banana", "cherry"} |
|  |  |
|  | thisset.add("orange") |
|  |  |
|  | print(thisset) |
|  |  |
|  | Adicionar conjuntos |
|  | Para adicionar itens de outro conjunto ao conjunto atual, use o update() método. |
|  |  |
|  | Exemplo |
|  | Adicione elementos de tropicalinto thisset: |
|  |  |
|  | thisset = {"apple", "banana", "cherry"} |
|  | tropical = {"pineapple", "mango", "papaya"} |
|  |  |
|  | thisset.update(tropical) |
|  |  |
|  | print(thisset) |
|  |  |
|  | Adicionar qualquer iterável |
|  | O objeto no update()método não precisa ser um conjunto, pode ser qualquer objeto iterável (tuplas, listas, dicionários etc.). |
|  |  |
|  | Exemplo |
|  | Adicione elementos de uma lista a at set: |
|  |  |
|  | thisset = {"apple", "banana", "cherry"} |
|  | mylist = ["kiwi", "orange"] |
|  |  |
|  | thisset.update(mylist) |
|  |  |
|  | print(thisset) |
|  |  |
|  |  |
|  | Remover item |
|  | Para remover um item em um conjunto, use o método remove(), ou .discard() |
|  |  |
|  | Exemplo |
|  | Remova "banana" usando o remove() método: |
|  |  |
|  | thisset = {"apple", "banana", "cherry"} |
|  |  |
|  | thisset.remove("banana") |
|  |  |
|  | print(thisset) |
|  |  |
|  | Nota: Se o item a ser removido não existir, remove()será gerado um erro. |
|  |  |
|  | Exemplo |
|  | Remova "banana" usando o discard() método: |
|  |  |
|  | thisset = {"apple", "banana", "cherry"} |
|  |  |
|  | thisset.discard("banana") |
|  |  |
|  | print(thisset) |
|  |  |
|  | Nota: Se o item a ser removido não existir, NÃOdiscard() irá gerar um erro. |
|  |  |
|  | Você também pode usar o pop()método para remover um item, mas esse método removerá |
|  | o último item. Lembre-se de que os conjuntos não são ordenados, então você não saberá qual item será removido. |
|  |  |
|  | O valor de retorno do pop()método é o item removido. |
|  |  |
|  | Exemplo |
|  | Remova o último item usando o pop() método: |
|  |  |
|  | thisset = {"apple", "banana", "cherry"} |
|  |  |
|  | x = thisset.pop() |
|  |  |
|  | print(x) |
|  |  |
|  | print(thisset) |
|  |  |
|  | Nota: Os conjuntos não são ordenados , portanto, ao usar o pop()método, você não sabe qual item será removido. |
|  |  |
|  | Exemplo |
|  | O clear() método esvazia o conjunto: |
|  |  |
|  | thisset = {"apple", "banana", "cherry"} |
|  |  |
|  | thisset.clear() |
|  |  |
|  | print(thisset) |
|  |  |
|  | Exemplo |
|  | A del palavra-chave excluirá o conjunto completamente: |
|  |  |
|  | thisset = {"apple", "banana", "cherry"} |
|  |  |
|  | del thisset |
|  |  |
|  | print(thisset) |
|  |  |
|  | Itens de loop |
|  |  |
|  | Você pode percorrer os itens definidos usando um for loop: |
|  |  |
|  | Exemplo |
|  | Percorra o conjunto e imprima os valores: |
|  |  |
|  | thisset = {"apple", "banana", "cherry"} |
|  |  |
|  | for x in thisset: |
|  | print(x) |
|  |  |
|  |  |
|  | Junte dois conjuntos |
|  |  |
|  | Existem várias maneiras de unir dois ou mais conjuntos em Python. |
|  |  |
|  | Você pode usar o union()método que retorna um novo conjunto contendo todos os itens de ambos |
|  | os conjuntos ou o update()método que insere todos os itens de um conjunto em outro: |
|  |  |
|  | Exemplo |
|  | O union()método retorna um novo conjunto com todos os itens de ambos os conjuntos: |
|  |  |
|  | set1 = {"a", "b" , "c"} |
|  | set2 = {1, 2, 3} |
|  |  |
|  | set3 = set1.union(set2) |
|  | print(set3) |
|  | Exemplo |
|  | O update()método insere os itens em set2 em set1: |
|  |  |
|  | set1 = {"a", "b" , "c"} |
|  | set2 = {1, 2, 3} |
|  |  |
|  | set1.update(set2) |
|  | print(set1) |
|  | Nota: Ambos union()e update() excluirão quaisquer itens duplicados. |
|  |  |
|  | PROPAGANDA |
|  |  |
|  | Mantenha APENAS as duplicatas |
|  | O intersection\_update()método manterá apenas os itens presentes em ambos os conjuntos. |
|  |  |
|  | Exemplo |
|  | Mantenha os itens que existem em ambos set x, e set y: |
|  |  |
|  | x = {"apple", "banana", "cherry"} |
|  | y = {"google", "microsoft", "apple"} |
|  |  |
|  | x.intersection\_update(y) |
|  |  |
|  | print(x) |
|  | O intersection()método retornará um novo conjunto, que contém apenas os itens presentes em ambos os conjuntos. |
|  |  |
|  | Exemplo |
|  | Retorna um set que contém os itens que existem em set xe set y: |
|  |  |
|  | x = {"apple", "banana", "cherry"} |
|  | y = {"google", "microsoft", "apple"} |
|  |  |
|  | z = x.intersection(y) |
|  |  |
|  | print(z) |
|  | Mantenha tudo, mas não as duplicatas |
|  | O symmetric\_difference\_update()método manterá apenas os elementos que NÃO estão presentes em ambos os conjuntos. |
|  |  |
|  | Exemplo |
|  | Guarde os itens que não estão presentes em ambos os conjuntos: |
|  |  |
|  | x = {"apple", "banana", "cherry"} |
|  | y = {"google", "microsoft", "apple"} |
|  |  |
|  | x.symmetric\_difference\_update(y) |
|  |  |
|  | print(x) |
|  | O symmetric\_difference()método retornará um novo conjunto, que contém apenas os elementos que NÃO estão |
|  | presentes em ambos os conjuntos. |
|  |  |
|  | Exemplo |
|  | Retorna um conjunto que contém todos os itens de ambos os conjuntos, exceto os itens presentes em ambos: |
|  |  |
|  | x = {"apple", "banana", "cherry"} |
|  | y = {"google", "microsoft", "apple"} |
|  |  |
|  | z = x.symmetric\_difference(y) |
|  |  |
|  | print(z) |
|  |  |
|  | Definir métodos |
|  | Python tem um conjunto de métodos integrados que você pode usar em conjuntos. |
|  |  |
|  | Method Description |
|  | add() Adds an element to the set |
|  | clear() Removes all the elements from the set |
|  | copy() Returns a copy of the set |
|  | difference() Returns a set containing the difference between two or more sets |
|  | difference\_update() Removes the items in this set that are also included in another, specified set |
|  | discard() Remove the specified item |
|  | intersection() Returns a set, that is the intersection of two other sets |
|  | intersection\_update() Removes the items in this set that are not present in other, specified set(s) |
|  | isdisjoint() Returns whether two sets have a intersection or not |
|  | issubset() Returns whether another set contains this set or not |
|  | issuperset() Returns whether this set contains another set or not |
|  | pop() Removes an element from the set |
|  | remove() Removes the specified element |
|  | symmetric\_difference() Returns a set with the symmetric differences of two sets |
|  | symmetric\_difference\_update() inserts the symmetric differences from this set and another |
|  | union() Return a set containing the union of sets |
|  | update() Update the set with the union of this set and others |
|  |  |
|  |  |
|  | Dicionários Python |
|  |  |
|  | thisdict = { |
|  | "brand": "Ford", |
|  | "model": "Mustang", |
|  | "year": 1964 |
|  | } |
|  |  |
|  | Dicionário |
|  |  |
|  | Os dicionários são usados para armazenar valores de dados em pares chave:valor. |
|  |  |
|  | Um dicionário é uma coleção ordenada\*, mutável e que não permite duplicatas. |
|  |  |
|  | A partir da versão 3.7 do Python, os dicionários são ordenados . No Python 3.6 e anteriores, |
|  | os dicionários não são ordenados . |
|  |  |
|  | Os dicionários são escritos com colchetes e possuem chaves e valores: |
|  |  |
|  | Exemplo |
|  | Crie e imprima um dicionário: |
|  |  |
|  | thisdict = { |
|  | "brand": "Ford", |
|  | "model": "Mustang", |
|  | "year": 1964 |
|  | } |
|  | print(thisdict) |
|  |  |
|  | Itens do dicionário |
|  | Os itens do dicionário são ordenados, alteráveis e não permitem duplicatas. |
|  |  |
|  | Os itens do dicionário são apresentados em pares chave:valor e podem ser referenciados usando o nome da chave. |
|  |  |
|  | Exemplo |
|  | Imprima o valor "marca" do dicionário: |
|  |  |
|  | thisdict = { |
|  | "brand": "Ford", |
|  | "model": "Mustang", |
|  | "year": 1964 |
|  | } |
|  | print(thisdict["brand"]) |
|  |  |
|  | Ordenado ou Não Ordenado? |
|  |  |
|  | A partir da versão 3.7 do Python, os dicionários são ordenados . No Python 3.6 e anteriores, os dicionários não |
|  | são ordenados . |
|  |  |
|  | Quando dizemos que os dicionários estão ordenados, significa que os itens têm uma ordem definida, e essa ordem não |
|  | será alterada. |
|  |  |
|  | Não ordenado significa que os itens não têm uma ordem definida, você não pode fazer referência a um item usando um |
|  | índice. |
|  |  |
|  | Mutável |
|  | Os dicionários são mutáveis, o que significa que podemos alterar, adicionar ou remover itens após a criação do |
|  | dicionário. |
|  |  |
|  | Duplicatas não permitidas |
|  | Os dicionários não podem ter dois itens com a mesma chave: |
|  |  |
|  | Exemplo |
|  | Valores duplicados substituirão os valores existentes: |
|  |  |
|  | thisdict = { |
|  | "brand": "Ford", |
|  | "model": "Mustang", |
|  | "year": 1964, |
|  | "year": 2020 |
|  | } |
|  | print(thisdict) |
|  |  |
|  |  |
|  | Tamanho do dicionário |
|  | Para determinar quantos itens um dicionário possui, use a len()função: |
|  |  |
|  | Exemplo |
|  | Imprima o número de itens no dicionário: |
|  |  |
|  | print(len(thisdict)) |
|  |  |
|  | Itens do Dicionário - Tipos de Dados |
|  | Os valores nos itens do dicionário podem ser de qualquer tipo de dados: |
|  |  |
|  | Exemplo |
|  | Tipos de dados String, int, boolean e list: |
|  |  |
|  | thisdict = { |
|  | "brand": "Ford", |
|  | "electric": False, |
|  | "year": 1964, |
|  | "colors": ["red", "white", "blue"] |
|  | } |
|  | tipo() |
|  |  |
|  | Da perspectiva do Python, os dicionários são definidos como objetos com o tipo de dados 'dict': |
|  |  |
|  | <class 'dict'> |
|  |  |
|  | Exemplo |
|  | Imprima o tipo de dados de um dicionário: |
|  |  |
|  | thisdict = { |
|  | "brand": "Ford", |
|  | "model": "Mustang", |
|  | "year": 1964 |
|  | } |
|  | print(type(thisdict)) |
|  |  |
|  | Coleções Python (matrizes) |
|  |  |
|  | Existem quatro tipos de dados de coleção na linguagem de programação Python: |
|  |  |
|  | Lista é uma coleção que é ordenada e mutável. Permite membros duplicados. |
|  | Tupla é uma coleção ordenada e imutável. Permite membros duplicados. |
|  | Set é uma coleção não ordenada, imutável\* e não indexada. Nenhum membro duplicado. |
|  | Dicionário é uma coleção ordenada\*\* e mutável. Nenhum membro duplicado. |
|  |  |
|  | \*Os itens do conjunto são imutáveis, mas você pode remover e/ou adicionar itens sempre que quiser. |
|  |  |
|  | \*\*A partir da versão 3.7 do Python, os dicionários são ordenados . No Python 3.6 e anteriores, os |
|  | dicionários não são ordenados . |
|  |  |
|  | Ao escolher um tipo de coleção, é útil entender as propriedades desse tipo. Escolher o tipo certo |
|  | para um determinado conjunto de dados pode significar retenção de significado e pode significar um |
|  | aumento na eficiência ou segurança. |
|  |  |
|  |  |
|  | Acessando itens |
|  |  |
|  | Você pode acessar os itens de um dicionário consultando o nome da chave, entre colchetes: |
|  |  |
|  | Exemplo |
|  | Obtenha o valor da chave "model": |
|  |  |
|  | thisdict = { |
|  | "brand": "Ford", |
|  | "model": "Mustang", |
|  | "year": 1964 |
|  | } |
|  | x = thisdict["model"] |
|  |  |
|  | Existe também um método chamado get()que lhe dará o mesmo resultado: |
|  |  |
|  | Exemplo |
|  | Obtenha o valor da chave "model": |
|  |  |
|  | x = thisdict.get("model") |
|  |  |
|  | Obter chaves |
|  | O keys()método retornará uma lista de todas as chaves do dicionário. |
|  |  |
|  | Exemplo |
|  | Obtenha uma lista das chaves: |
|  |  |
|  | x = thisdict.keys() |
|  |  |
|  | A lista de chaves é uma visualização do dicionário, o que significa que quaisquer |
|  | alterações feitas no dicionário serão refletidas na lista de chaves. |
|  |  |
|  | Exemplo |
|  | Adicione um novo item ao dicionário original e veja se a lista de chaves também é atualizada: |
|  |  |
|  | car = { |
|  | "brand": "Ford", |
|  | "model": "Mustang", |
|  | "year": 1964 |
|  | } |
|  |  |
|  | x = car.keys() |
|  |  |
|  | print(x) #before the change |
|  |  |
|  | car["color"] = "white" |
|  |  |
|  | print(x) #after the change |
|  |  |
|  | Obter valores |
|  | O values()método retornará uma lista de todos os valores do dicionário. |
|  |  |
|  | Exemplo |
|  | Obtenha uma lista dos valores: |
|  |  |
|  | x = thisdict.values() |
|  | A lista de valores é uma visualização do dicionário, o que significa que quaisquer |
|  | alterações feitas no dicionário serão refletidas na lista de valores. |
|  |  |
|  | Exemplo |
|  | Faça uma alteração no dicionário original e veja se a lista de valores também é atualizada: |
|  |  |
|  | car = { |
|  | "brand": "Ford", |
|  | "model": "Mustang", |
|  | "year": 1964 |
|  | } |
|  |  |
|  | x = car.values() |
|  |  |
|  | print(x) #before the change |
|  |  |
|  | car["year"] = 2020 |
|  |  |
|  | print(x) #after the change |
|  |  |
|  | Exemplo |
|  | Adicione um novo item ao dicionário original e veja se a lista de valores também é atualizada: |
|  |  |
|  | car = { |
|  | "brand": "Ford", |
|  | "model": "Mustang", |
|  | "year": 1964 |
|  | } |
|  |  |
|  | x = car.values() |
|  |  |
|  | print(x) #before the change |
|  |  |
|  | car["color"] = "red" |
|  |  |
|  | print(x) #after the change |
|  |  |
|  | Obter itens |
|  | O items()método retornará cada item em um dicionário, como tuplas em uma lista. |
|  |  |
|  | Exemplo |
|  | Obter uma lista dos pares chave:valor |
|  |  |
|  | x = thisdict.items() |
|  |  |
|  | A lista retornada é uma visualização dos itens do dicionário, o que significa que quaisquer alterações |
|  | feitas no dicionário serão refletidas na lista de itens. |
|  |  |
|  | Exemplo |
|  | Faça uma alteração no dicionário original e veja se a lista de itens também é atualizada: |
|  |  |
|  | car = { |
|  | "brand": "Ford", |
|  | "model": "Mustang", |
|  | "year": 1964 |
|  | } |
|  |  |
|  | x = car.items() |
|  |  |
|  | print(x) #before the change |
|  |  |
|  | car["year"] = 2020 |
|  |  |
|  | print(x) #after the change |
|  |  |
|  | Exemplo |
|  | Adicione um novo item ao dicionário original e veja se a lista de itens também é atualizada: |
|  |  |
|  | car = { |
|  | "brand": "Ford", |
|  | "model": "Mustang", |
|  | "year": 1964 |
|  | } |
|  |  |
|  | x = car.items() |
|  |  |
|  | print(x) #before the change |
|  |  |
|  | car["color"] = "red" |
|  |  |
|  | print(x) #after the change |
|  |  |
|  | Verifique se a chave existe |
|  | Para determinar se uma chave especificada está presente em um dicionário, use a inpalavra-chave: |
|  |  |
|  | Exemplo |
|  | Verifique se "modelo" está presente no dicionário: |
|  |  |
|  | thisdict = { |
|  | "brand": "Ford", |
|  | "model": "Mustang", |
|  | "year": 1964 |
|  | } |
|  | if "model" in thisdict: |
|  | print("Yes, 'model' is one of the keys in the thisdict dictionary") |
|  |  |
|  |  |
|  | Mudar valores |
|  |  |
|  | Você pode alterar o valor de um item específico consultando seu nome de chave: |
|  |  |
|  | Exemplo |
|  | Altere o "ano" para 2018: |
|  |  |
|  | thisdict = { |
|  | "brand": "Ford", |
|  | "model": "Mustang", |
|  | "year": 1964 |
|  | } |
|  | thisdict["year"] = 2018 |
|  |  |
|  | Atualizar dicionário |
|  | O update()método atualizará o dicionário com os itens do argumento fornecido. |
|  |  |
|  | O argumento deve ser um dicionário ou um objeto iterável com pares chave:valor. |
|  |  |
|  | Exemplo |
|  | Atualize o "ano" do carro usando o update() método: |
|  |  |
|  | thisdict = { |
|  | "brand": "Ford", |
|  | "model": "Mustang", |
|  | "year": 1964 |
|  | } |
|  | thisdict.update({"year": 2020}) |
|  |  |