Resumo Python – Santander Open Academy.

**Conceitos básicos da sintaxe em Python**

Indentação

No Python, a indentação (espaços ou tabulações no início de uma linha) é utilizada para delimitar blocos de código. Diferente de outras linguagens que utilizam chaves ou palavras-chave, o Python utiliza a indentação para determinar o escopo das declarações. Por exemplo:

**if condition:**

# Bloco de código se a condição for verdadeira

instrucao1

instrucao2  
**else:**  
 # Bloco de código se a condição for falsa  
instrucao3

instrucao4

**É fundamental manter uma indentação consistente em todo o código para evitar erros de sintaxe**.

Comentários

Os comentários são linhas de texto no código que são ignoradas pelo interpretador do Python. Eles são utilizados para explicar ou documentar o código. No Python, os comentários de uma única linha começam com o símbolo #, enquanto os comentários de várias linhas são delimitados por três aspas """ . Por exemplo:

# Este é um comentário de uma única linha  
  
"""  
Este é um comentário  
de várias linhas  
"""

Sensibilidade a maiúsculas e minúsculas

Python distingue entre maiúsculas e minúsculas. Portanto, variável, Variável e VARIÁVEL são consideradas variáveis diferentes.

Ponto e vírgula

Diferente de outras linguagens, o Python não requer o uso de ponto e vírgula (;) ao final de cada instrução. No entanto, se você desejar escrever várias instruções em uma única linha, pode separá-las com um ponto e vírgula. Por exemplo:

instrucao1; instrucao2; instrucao3

Uso de parênteses

Os parênteses são utilizados para agrupar expressões, definir funções e realizar chamadas a funções. Por exemplo:

resultado = (a + b) \* c

**Tipos de dados básicos**

Em Python, os tipos de dados básicos são as categorias nas quais podemos classificar os valores que utilizamos em nossos programas. Compreender os diferentes tipos de dados é fundamental para trabalhar com variáveis e realizar operações em Python. Os tipos de dados básicos incluem:

Inteiros (int)

Os números inteiros são aqueles que não têm parte decimal. Em Python, são representados simplesmente escrevendo o número sem aspas nem pontos decimais. Por exemplo:

idade = 25  
quantidade = 100

Flutuantes (float)

Os números flutuantes, também conhecidos como números de ponto flutuante, são aqueles que têm uma parte decimal. Em Python, são representados utilizando um ponto para separar a parte inteira da parte decimal. Por exemplo:

preço = 9.99  
altura = 1.75

Cadeias de texto (strings)

As cadeias de texto, ou simplesmente cadeias, são sequências de caracteres encerradas entre aspas simples ('...') ou duplas ("..."). São utilizadas para representar texto em Python. Por exemplo:

nome = "Juan"  
mensagem = '¡Hola, mundo!'

Você pode incluir caracteres especiais nas cadeias utilizando o caractere de escape \. Por exemplo, para incluir aspas dentro de uma cadeia, você pode usar \' ou \". Também pode utilizar a notação de tripla aspa ('''...''' ou """...""") para criar cadeias de várias linhas.

Booleanos

Os valores booleanos representam os valores de verdade: True (verdadeiro) e False (falso). São comumente utilizados em expressões condicionais e operações lógicas. Por exemplo:

é\_maior\_de\_idade = True  
tem\_desconto = False

Os valores booleanos em Python começam com uma letra maiúscula: True e False.

As variáveis são contêineres que nos permitem armazenar e manipular dados em nossos programas. Você pode pensar em uma variável como uma etiqueta à qual você atribui um valor específico. Em Python, não é necessário declarar o tipo de dados de uma variável com antecedência, pois o Python infere o tipo de dados automaticamente com base no valor atribuído.

Declaração e atribuição de variáveis

As variáveis são contêineres que nos permitem armazenar e manipular dados em nossos programas. Para declarar e atribuir um valor a uma variável em Python, utilizamos o operador de atribuição =. O nome da variável vai à esquerda do operador, e o valor que você deseja atribuir vai à direita. Por exemplo:

nome = "Juan"  
idade = 25  
altura = 1.75  
é estudante = True

No exemplo, declaramos e atribuímos valores às variáveis nome, idade, altura e é\_estudante. O Python infere automaticamente o tipo de dados de cada variável com base no valor atribuído.

Você também pode atribuir o mesmo valor a várias variáveis em uma única linha usando o operador de atribuição múltipla:

a = b = c = 10

Neste caso, as variáveis a, b e c terão o valor 10.

Regras para nomear variáveis

Ao nomear variáveis em Python, é importante seguir algumas regras para manter um código legível e evitar erros:

Os nomes das variáveis só podem conter letras (a-z, A-Z), números (0-9) e sublinhados (\_). Não podem começar com um número.

O Python diferencia maiúsculas de minúsculas, então nome e Nome são variáveis diferentes.

Não se pode usar palavras-chave reservadas do Python como nomes de variáveis (por exemplo, if, else, for, while, etc.).

Recomenda-se usar nomes descritivos para as variáveis, que indiquem claramente seu propósito: nome, idade, total\_vendas, etc.

Seguindo essas regras, alguns exemplos de nomes de variáveis válidos são:

idade  
nome\_completo  
total\_vendas  
\_contador

E alguns exemplos de nomes de variáveis inválidos são:

1idade # Começa com um número  
nome-completo # Usa um hífen em vez de um sublinhado  
if # Palavra-chave reservada do Pyths operadores são símbolos especiais que nos permitem realizar operações em variáveis e valores. Python fornece diferentes tipos de operadores para realizar operações aritméticas, comparações e operações lógicas.

**Aritméticos**

Os operadores aritméticos são utilizados para realizar operações matemáticas básicas. Os principais operadores aritméticos em Python são:

* Soma (+): soma dois valores.
* Subtração (-): subtrai o segundo valor do primeiro.
* Multiplicação (\*): multiplica dois valores.
* Divisão (/): divide o primeiro valor pelo segundo e devolve um resultado de tipo flutuante.
* Divisão inteira (//): divide o primeiro valor pelo segundo e devolve um resultado de tipo inteiro (a parte decimal é descartada).
* Módulo (%): devolve o resto da divisão entre o primeiro valor e o segundo.
* Exponenciação (\*\*): eleva o primeiro valor à potência do segundo.

Exemplos:

a = 10  
b = 3  
  
  
soma = a + b   # 13  
subtracao = a - b    # 7  
multiplicacao = a \* b    # 30  
divisao = a / b   # 3.333333333  
divisao\_inteira = a // b   # 3  
modulo = a % b   # 1  
exponenciacao = a \*\* b   # 1000

**De comparação**

Os operadores de comparação são utilizados para comparar dois valores e devolvem um valor booleano (True ou False) segundo o resultado da comparação. Os operadores de comparação em Python são:

* Igual a (==): devolve True se ambos os valores são iguais.
* Diferente de (!=): devolve True se os valores são diferentes.
* Maior que (>): devolve True se o primeiro valor é maior que o segundo.
* Menor que (<): devolve True se o primeiro valor é menor que o segundo.
* Maior ou igual que (>=): devolve True se o primeiro valor é maior ou igual que o segundo.
* Menor ou igual que (<=): devolve True se o primeiro valor é menor ou igual que o segundo.

Exemplos:

a = 10  
b = 3  
  
  
igual = a == b   # False  
diferente = a != b   # True  
maior que = a > b   # True  
menor que = a < b   # False  
maior ou igual = a >= b   # True  
menor ou igual = a <= b   # False

**Lógicos**

Os operadores lógicos são utilizados para combinar expressões condicionais e avaliar múltiplas condições. Os operadores lógicos em Python são:

* AND (and): devolve True se ambas as condições são verdadeiras.
* OR (or): devolve True se ao menos uma das condições é verdadeira.
* NOT (not): inverte o valor de uma condição, devolve True se a condição é falsa e False se a condição é verdadeira.

Exemplo:

a = 10  
b = 3  
  
  
resultado\_and = (a > 5) and (b < 5)   # True  
resultado\_or = (a > 15) or (b < 5)   # True  
resultado\_not = not (a > 5)   # False

Você pode utilizar esses operadores para realizar cálculos, tomar decisões baseadas em comparações e combinar condições lógicas em seus programas.

Os loops nos permitem repetir um bloco de código várias vezes. Em Python, os loops mais comuns são for e while.

**For**

O loop for é utilizado para iterar sobre uma sequência (como uma lista, uma tupla ou uma string) ou qualquer objeto iterável. A sintaxe básica é a seguinte:

for variável in sequência:  
  
    # Bloco de código a repetir  
    instruções

Exemplo:

frutas = ["maçã", "banana", "laranja"]  
  
  
for fruta in frutas:  
    print(fruta)

Neste exemplo, o loop for itera sobre a lista frutas. Em cada iteração, a variável fruta assume o valor de um elemento da lista, e o bloco de código dentro do loop é executado. Neste caso, cada fruta é impressa em uma linha separada.

**While**

O loop while é utilizado para repetir um bloco de código enquanto uma condição for verdadeira. A sintaxe básica é a seguinte:

while condição:  
  
    # Bloco de código a repetir  
    instruções

Exemplo:

contador = 0  
  
  
while contador < 5:  
  
    print(contador)  
    contador += 1

Neste exemplo, o loop while é executado enquanto a variável contador for menor que 5. Em cada iteração, o valor de contador é impresso e depois incrementado em 1 pela instrução contador += 1. O loop será interrompido quando contador atingir o valor de 5.

É importante ter cuidado ao usar o loop while, pois, se a condição nunca se tornar falsa, o loop será executado indefinidamente, o que é conhecido como um loop infinito.

**Controle de loops**

Python fornece algumas instruções especiais para controlar o fluxo de execução dentro dos loops:

* Break

A instrução break é utilizada para sair prematuramente de um loop, independentemente da condição. Quando um break é encontrado, o loop é interrompido e o fluxo de execução continua com a próxima instrução fora do loop.

contador = 0  
  
  
while True:  
  
    print(contador)  
    contador += 1  
  
  
    if contador == 5:  
        break

Neste exemplo, o loop while é executado indefinidamente devido à condição True. No entanto, dentro do loop é utilizada uma estrutura condicional if para verificar se contador é igual a 5. Quando essa condição é satisfeita, a instrução break é executada, fazendo com que o loop seja interrompido e o fluxo de execução continue com a próxima instrução fora do loop.

* Continue

A instrução continue é utilizada para pular o restante do bloco de código dentro de um loop e passar para a próxima iteração.

Exemplo:

for i in range(10):  
  
    if i % 2 == 0:  
        continue  
    print(i)

Neste exemplo, o loop for itera sobre os números de 0 a 9 utilizando a função range(). Dentro do loop, verifica-se se o número é divisível por 2 utilizando o operador de módulo %. Se o número for divisível por 2 (ou seja, se for par), a instrução continue é executada, fazendo com que o restante do bloco de código seja pulado e passando para a próxima iteração do loop. Como resultado, apenas os números ímpares serão impressos.

* Pass

A instrução pass é uma operação nula que não faz nada. É utilizada como um marcador de posição quando uma instrução é sintaticamente necessária, mas nenhuma ação é desejada.

Exemplo:

for i in range(5):  
    pass

Neste exemplo, o loop for itera sobre os números de 0 a 4, mas nenhuma ação é realizada dentro do loop devido à instrução pass. Isso pode ser útil quando se está desenvolvendo um programa e se deseja reservar um bloco de código para implementá-lo mais tarde.

Estruturas de dados

As estruturas de dados nos permitem organizar e armazenar dados de maneira eficiente em nossos programas. Python fornece várias estruturas de dados integradas, como listas, tuplas, dicionários e conjuntos, cada uma com suas próprias características e usos.

**Listas**

Uma lista é uma estrutura de dados mutável e ordenada que permite armazenar uma coleção de elementos. Os elementos de uma lista podem ser de diferentes tipos de dados e são encerrados entre colchetes [], separados por vírgulas.

* Criação e acesso

Para criar uma lista, simplesmente encerre os elementos entre colchetes:

frutas = ["maçã", "banana", "laranja"]

Para acessar os elementos de uma lista, utilize o índice do elemento entre colchetes. Os índices começam a partir de 0.

print(frutas[0])  # Imprime "maçã"  
print(frutas[1])  # Imprime "banana"  
print(frutas[2])  # Imprime "laranja"

Você também pode acessar os elementos a partir do final da lista utilizando índices negativos. O índice -1 representa o último elemento, -2 representa o penúltimo, e assim por diante.

print(frutas[-1])  # Imprime "laranja"  
print(frutas[-2])  # Imprime "banana"  
print(frutas[-3])  # Imprime "maçã"

* Métodos de listas

As listas em Python têm vários métodos incorporados que nos permitem manipular e modificar os elementos da lista. Alguns métodos comuns são:

* append(elemento): adiciona um elemento ao final da lista.
* insert(indice, elemento): insere um elemento em uma posição específica da lista.
* remove(elemento): remove a primeira ocorrência de um elemento na lista.
* pop(indice): remove e retorna o elemento em uma posição específica da lista.
* sort(): ordena os elementos da lista em ordem ascendente.
* reverse(): inverte a ordem dos elementos na lista.

Exemplo:

frutas = ["maçã", "banana", "laranja"]  
  
  
frutas.append("pera")  
print(frutas)  # Imprime ["maçã", "banana", "laranja", "pera"]  
  
  
frutas.insert(1, "uva")  
print(frutas)  # Imprime ["maçã", "uva", "banana", "laranja", "pera"]  
  
  
frutas.remove("banana")  
print(frutas)  # Imprime ["maçã", "uva", "laranja", "pera"]  
  
  
fruta\_removida = frutas.pop(2)  
print(frutas)  # Imprime ["maçã", "uva", "pera"]  
print(fruta\_removida)  # Imprime "laranja"  
  
  
frutas.sort()  
print(frutas)  # Imprime ["maçã", "pera", "uva"]  
  
  
frutas.reverse()  
print(frutas)  # Imprime ["uva", "pera", "maçã"]

* Listas de compreensão

As listas de compreensão são uma forma concisa de criar novas listas baseadas em uma sequência existente. Permitem filtrar e transformar os elementos de uma lista em uma única linha de código.

nova\_lista = [expressão for elemento in sequência if condição]

Exemplo:

números = [1, 2, 3, 4, 5]  
quadrados = [x \*\* 2 for x in números if x % 2 == 0]  
print(quadrados)  # Imprime [4, 16]

Neste exemplo, é criada uma nova lista chamada quadrados, que contém os quadrados dos números pares da lista números. A expressão x \*\* 2 eleva cada elemento ao quadrado, e a condição if x % 2 == 0 filtra apenas os números pares.

**Tuplas**

Uma tupla é uma estrutura de dados imutável e ordenada que permite armazenar uma coleção de elementos. Os elementos de uma tupla são encerrados entre parênteses (), separados por vírgulas.

* Criação e acesso

Para criar uma tupla, encerre os elementos entre parênteses:

ponto = (3, 4)

Para acessar os elementos de uma tupla, utilize o índice do elemento entre colchetes, similar às listas:

print(ponto[0])  # Imprime 3  
  
print(ponto[1])  # Imprime 4

Ao contrário das listas, as tuplas são imutáveis, o que significa que não podem ser modificadas uma vez criadas. Não se pode adicionar, eliminar ou alterar elementos em uma tupla existente.

As tuplas são úteis quando você precisa armazenar uma coleção de elementos que não devem ser modificados, como coordenadas ou dados de configuração.

* Métodos de tuplas

Embora as tuplas sejam imutáveis, Python fornece vários métodos úteis para trabalhar com elas:

* **count(elemento):** devolve o número de vezes que um elemento aparece na tupla.
* **index(elemento):** devolve o índice da primeira aparição de um elemento na tupla. Opcionalmente, pode-se especificar o início e fim da busca.
* **len(tupla):** embora não seja um método de tupla propriamente dito, esta função incorporada devolve o comprimento da tupla.

minha\_tupla = (1, 2, 3, 2, 4, 2)  
  
  
print (minha\_tupla.index(2))   # Saída: 1  
  
print (minha\_tupla.index(2, 2))   #Saída: 3  
  
print (minha\_tupla.index(2, 2, 4))   #Saída: 3

Um dicionário é uma estrutura de dados mutável e não ordenada que permite armazenar pares de chave-valor. Cada elemento em um dicionário consiste em uma chave única e seu valor correspondente. Os dicionários são delimitados por chaves {}, e os pares chave-valor são separados por vírgulas.

* Criação e acesso

Para criar um dicionário, utilize chaves e separe as chaves e valores com dois pontos.

pessoa = {"nome": "João", "idade": 25, "cidade": "Madri"}

Para acessar os valores de um dicionário, utilize a chave correspondente entre colchetes:

print(pessoa["nome"])  # Imprime "João"  
print(pessoa["idade"])    # Imprime 25  
print(pessoa["cidade"])  # Imprime "Madri"

Você também pode utilizar o método get() para obter o valor de uma chave. Se a chave não existir, retorna um valor padrão (por padrão, None).

* Métodos de dicionários

Os dicionários em Python têm vários métodos incorporados para manipular e acessar os elementos. Alguns métodos comuns são:

* **keys():** retorna uma visualização de todas as chaves do dicionário.
* **values():** retorna uma visualização de todos os valores do dicionário.
* **items():**retorna uma visualização de todos os pares chave-valor do dicionário.
* **update(outro\_dicionario):**atualiza o dicionário com os pares chave-valor de outro dicionário.

Exemplo:

pessoa = {"nome": "João", "idade": 25, "cidade": "Madri"}  
  
  
print(pessoa.keys())    # Imprime dict\_keys(["nome", "idade", "cidade"])  
print(pessoa.values())  # Imprime dict\_values(["João", 25, "Madri"])  
print(pessoa.items())   # Imprime dict\_items([("nome", "João"), ("idade", 25), ("cidade", "Madri")])  
  
  
pessoa.update({"profissao": "Engenheiro"})  
print(pessoa)  # Imprime {"nome": "João", "idade": 25, "cidade": "Madri", "profissao": "Engenheiro"}

Conjuntos (set)

1

2

3

4

5

6

7

8

Um conjunto é uma estrutura de dados mutável e não ordenada que permite armazenar uma coleção de elementos únicos. Os conjuntos são delimitados por chaves {} ou são criados utilizando a função set().

* Criação e operações básicas

Para criar um conjunto, utilize chaves ou a função set():

frutas = {"maçã", "banana", "laranja"}  
numeros = set([1, 2, 3, 4, 5])

Os conjuntos suportam operações matemáticas de conjuntos, como a união (|), a interseção (&), a diferença (-) e a diferença simétrica (^).

conjunto1 = {1, 2, 3}  
conjunto2 = {3, 4, 5}  
  
  
uniao = conjunto1 | conjunto2  
print(uniao)  # Imprime {1, 2, 3, 4, 5}  
  
  
intersecao = conjunto1 & conjunto2  
print(intersecao)  # Imprime {3}  
  
  
diferenca = conjunto1 - conjunto2  
print(diferenca)  # Imprime {1, 2}  
  
  
diferenca\_simetrica = conjunto1 ^ conjunto2  
print(diferenca\_simetrica)  # Imprime {1, 2, 4, 5}

* Métodos de conjuntos

Os conjuntos em Python têm vários métodos incorporados para manipular e acessar os elementos. Alguns métodos comuns são:

* add(elemento): adiciona um elemento ao conjunto.
* remove(elemento): remove um elemento do conjunto. Se o elemento não existir, gera um erro.
* discard(elemento): remove um elemento do conjunto se estiver presente. Se o elemento não existir, não faz nada.
* clear(): remove todos os elementos do conjunto.

Exemplo:

frutas = {"maçã", "banana", "laranja"}  
  
  
frutas.add("pera")  
print(frutas)  # Imprime {"maçã", "banana", "laranja", "pera"}  
  
  
frutas.remove("banana")  
print(frutas)  # Imprime {"maçã", "laranja", "pera"}  
  
  
frutas.discard("uva")  
print(frutas)  # Imprime {"maçã", "laranja", "pera"}  
  
  
frutas.clear()  
print(frutas)  # Imprime set()

As estruturas de dados em Python nos oferecem grande flexibilidade e potência para armazenar e manipular dados em nossos programas. As listas são úteis para coleções ordenadas e mutáveis, as tuplas para coleções ordenadas e imutáveis, os dicionários para armazenar pares de chave valor e os conjuntos para coleções não ordenadas de elementos únicos.