*Introdução à programação com o Python II*

Noções básicas de programação

Variável, atribuição, expressões

Nesta seção, veremos os blocos conceituais mais básicos da programação com o Python.

Vamos pensar na seguinte expressão matemática: x + y = z

Nesta expressão, queremos representar que a soma dos números x e y resultará no número z.

Agora vamos pensar nisto em termos de programação: se quiséssemos fazer um programa que realizasse uma adição, podemos propor isso da seguinte maneira:

* Inserir um número, que desempenhará a função de x.
* Inserir outro número, que desempenhará a função de y
* Instruir o computador para efetuar a soma dos dois números.
* Faremos com que o resultado desempenhe a função de z
* Mostrar ao usuário o valor de z, ou seja, o resultado da adição dos dois números inseridos x e y.

Todo esse processo é projetado detalhadamente para a programação.

Toda vez que tivermos um símbolo ao qual damos um valor concreto (por exemplo, x = 2), estaremos fazendo uma atribuição a uma variável. O símbolo x é uma variável porque pode assumir qualquer valor. Neste caso em particular, damos a ele (através do processo de atribuição) o valor 2.

No Python, podemos trabalhar da seguinte maneira:

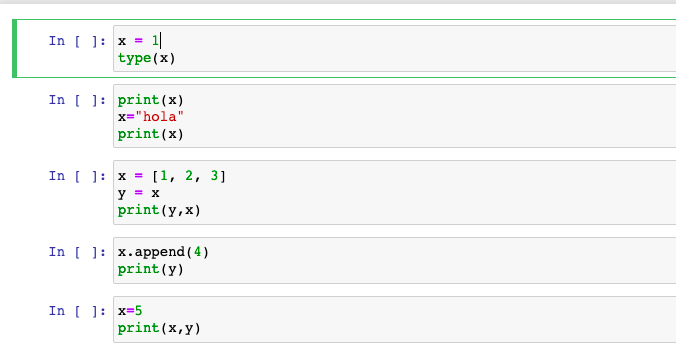
| x = 2 y = 3 z = x + y z 5 |
| --- |

A primeira linha atribui à variável x o valor 2. A segunda linha atribui à variável y o valor 3. A terceira linha soma x e y. A quarta linha mostra o resultado.

Cada parte do código "composto" que representa um valor particular (por exemplo, x + y) é uma expressão. Em princípio, podemos fazer a analogia com expressões matemáticas sem nenhum problema.

Exercícios

Quais são as saídas dos seguintes blocos de código?



Objetos e ponteiros

No Python tudo é um objeto. Um objeto, em termos de programação, é uma entidade que possui dados e, também, possui metadados (atributos ou propriedades) e funcionalidade própria (métodos). Considere o seguinte caso:

| x=1.0 x.is\_integer()  x=1.4 x.is\_integer() |
| --- |

Nesses casos, is\_integer() é uma funcionalidade de x que diz se é inteiro ou não

| print(x.real,x.imag) |
| --- |

Aqui, real e imag são propriedades de x que mostram sua parte real e sua parte imaginária.

Por outro lado, o exemplo da seção anterior mostra a forma de trabalhar com ponteiros (pointers). Ao contrário de outras linguagens onde as variáveis se comportam como contêineres de dados, as variáveis no Python são ponteiros. Isto significa que elas não contêm um valor, mas apontam para um dado que se encontra na memória. Desta forma, se atribuirmos uma variável à outra (y = x), estamos simplesmente dizendo a ambas que apontem para o mesmo dado (a lista [1, 2, 3]). Em sua forma mais específica, a operação “=” realiza a conexão entre o ponteiro e os dados. Em contrapartida, a aplicação de métodos (por exemplo, append()) altera os dados de destino.

Esta diferença é sutil e, na prática, não afeta o trabalho do Data Science. No entanto, traz melhorias substanciais na performance e é importante levá-la em conta para não cometer erros na programação.

Operações

O Python define uma série de operações básicas. Estas operações são mostradas a seguir:

As operações aritméticas são as mais básicas. Prestar atenção a algumas específicas que podem ser muito úteis.

| Aritméticas |  |
| --- | --- |
| a + b | Adição |
| a - b | Subtração |
| a \* b | Multiplicação |
| a / b | Divisão |
| a // b | Divisão inteira (resultado sem decimal) |
| a % b | Módulo (restante da divisão inteira) |
| a \*\* b | Exponenciação |
| -a | Negativo |

Além do operador de atribuição “=”, há ainda os seguintes, que são equivalentes às operações da coluna da direita, e podem nos fazer poupar muito tempo.

| Atribuições |  |
| --- | --- |
| a += b | a = a + b |
| a -= b | a = a - b |
| a \*= b | a = a \* b |
| a /= b | a = a / b |
| a //= b | a = a // b |
| a %= b | a = a % b |
| a \*\*= b | a = a \*\* b |

Os comparadores servem para comparar objetos. Eles se comportam de maneira semelhante aos símbolos matemáticos que representam.

| Comparadores |  |
| --- | --- |
| a == b | a igual a b |
| a != b | a diferente de b |
| a < b | a menor que b |
| a > b | a maior que b |
| a <= b | a menor ou igual a b |
| a >= b | a maior ou igual a b |

Os operadores de identidade vão mais fundo nos detalhes dos objetos e ponteiros. Se dois objetos são o mesmo (operador “is”), significa que apontam para o mesmo dado. Por outro lado, se quisermos saber se um dado está contido em outro, não importa em que parte, o operador "in" nos será de grande utilidade. Este último é frequentemente usado em listas de dados.

| **Identidade e associação** |  |
| --- | --- |
| a is b | a é o mesmo objeto que b |
| a is not b | a não é o mesmo objeto que b |
| a in b | a está contido em b |
| a not in b | a não está contido em b |

Estruturas de controle

Para controlar o fluxo de execução de um programa, utilizaremos estruturas de controle. Existem várias estruturas de acordo com os objetivos que perseguimos. Veremos a seguir os mais utilizados no Data Science.

Estrutura FOR

A estrutura FOR repete um comando uma quantidade fixa de vezes. Vejamos alguns exemplos:

| for i in range(1,10):  print(i) for i in [1,4,6,2]:  print(i) |
| --- |

Estrutura WHILE

A estrutura WHILE repete uma sequência de comandos "desde que" uma condição seja cumprida. Quando a condição não é mais cumprida, termina a repetição.

| i = 1 while i < 10:  print(i)  i += 1 |
| --- |

Estrutura condicional (IF)

A estrutura condicional funciona com opções de código. Se uma condição for cumprida, uma sequência de comando é executada. Caso contrário, outra sequência de comando é executada. Mais de duas opções podem ser operadas com o comando elif. Vejamos um exemplo.

| x = 1 if x < 10:  print(x, "es menor a 10") elif x > 10:  print(x, "es mayor a 10") else:  print(x, "es 10") |
| --- |

Exercícios

* O fatorial x! é definido como o produto de todos os números de 1 a x. Digitar o código para calcular o fatorial de 5.
* J. B. Büttner, um professor de uma escola alemã, puniu todas as crianças dando à elas a tarefa de somar os primeiros 100 números naturais para mantê-las entretidas e quietas por um longo tempo. Carl Friedrich Gauss obteve a resposta quase imediatamente, encontrando uma fórmula para a soma de uma série de números naturais.   
  Não estamos pedindo aqui para chegar à fórmula. O que pedimos neste exercício é para fazer um programa que o calcule sem utilizar a fórmula.

Exercício 1:

| fac = 1 for i in range(1,6):  # print(fac)  fac \*= i fac |
| --- |

Exercício 2:

| soma = 0 for i in range(1,101):  print(i, soma)  suma += i soma |
| --- |

Funções, argumentos e retorno

Uma forma de organizar os comandos que são utilizados de forma repetitiva é definir funções. Uma função é um agrupamento de instruções que têm um nome definido. As funções podem ter argumentos, ou seja, entradas de dados. Por fim, as funções têm um valor de retorno, ou seja, a saída obtida ao executar da função. Vejamos um exemplo:

| print(x) print(x,y) |
| --- |

As funções são escritas com um nome e os argumentos entre parênteses. Se não houver argumentos, também são escritos os parênteses vazios "()".

Além das muitas funções oferecidas pelo Python, podemos incorporar funções próprias. Vejamos o seguinte:

| def suma(x,y):  z = x + y  return z |
| --- |

então vamos inserir:

| soma(2,3) |
| --- |

Exercícios

* Escrever uma função para calcular o fatorial de um número qualquer.
* Escrever uma função para calcular a soma de uma série começando em um número qualquer e terminando em outro número que deve ser maior do que o primeiro.

Tipos de dados

O Python processa dois grupos de tipos de dados: simples e estruturados.

Dados simples

Os dados simples são os seguintes:

| Tipos de dados |  |  |
| --- | --- | --- |
| int | x = 1 | Inteiros |
| float | x = 1.0 | Ponto flutuante (decimais) |
| complex | x = 1 + 2j | Complexos (parte real e imaginária) |
| bool | x = True | Booleanos ou lógicos: verdadeiro / falso |
| str | x = 'abc' | Texto |
| NoneType | x = None | Tipo especial para indicar valores nulos |

Podemos saber o tipo de um dado x com a função type(x).

Dados estruturados

Os tipos de dados estruturados são agrupamentos de tipos de dados simples. Os tipos de dados predeterminados no Python são mostrados a seguir:

| Estruturados | |  |
| --- | --- | --- |
| list | [1, 2, 3] | Lista requisitada |
| tuple | (1, 2, 3) | Lista ordenada imutável |
| dict | {'a':1, 'b':2, 'c':3} | Dicionário: conjunto de pares chave: valor |
| set | {1, 2, 3} | Conjunto, na forma de um conjunto matemático |

Dizemos que a estrutura list é *mutável* porque permite que seus elementos sofram modificações depois que são definidos. Ou seja, sempre poderemos mudar da forma que desejarmos os elementos que compõem a lista. Por outro lado, as *estruturas imutáveis* como as tuplas não admitem esta reatribuição de elementos em tempo de execução do programa.

Outro exemplo de uma estrutura mutável é a *estrutura de dict, de modo que sempre* poderemos modificar os valores de um dict. Entretanto, deve ser destacado que não é possível modificar suas chaves.

As formas de organizar os tipos de dados estruturados têm aplicações diferentes de acordo com a forma como cada um deles é destinado. Para começar com no Data Science, usaremos o tipo de dado list.