Certified Tech Developer
The Ultimate Degree

Infraestrutura I

Estruturas de controle

Como vimos antes, os algoritmos requerem duas estruturas de controle importantes: iteração e seleção. Ambos estão disponíveis em Python em várias formas. Quem programar pode escolher a instrução que é mais útil para a circunstância dada.

Para iteração, o Python fornece uma instrução **while** padrão e uma instrução **for** muito poderosa. A instrução **while** repete um corpo de código enquanto uma condição for verdadeira. Por exemplo:

```
>>> contador = 1
>>> while contador <= 5:
...     print("Olá, mundo")
...     contador = contador + 1

Olá, mundo
Olá, mundo
Olá, mundo
Olá, mundo
Olá, mundo
```

Este código imprime a frase "Olá, mundo" cinco vezes. A condição na instrução **while** é avaliada no início de cada iteração. Se a condição for **True**, o corpo da instrução será executado. É fácil ver a estrutura de uma instrução while do Python por causa do padrão de recuo obrigatório que a linguagem impõe.

A instrução **while** é uma estrutura iterativa de propósito geral que usaremos em vários algoritmos diferentes. Em muitos casos, uma condição composta controlará a iteração.

Um trecho como o seguinte:

```
while contador <= 10 and not hecho:
...
```

Isso faria com que o corpo da instrução fosse executado apenas no caso em que ambas as partes da condição fossem atendidas. O valor da variável **contador** deve ser menor ou igual a 10 e o valor da variável **hecho** tinha que ser **False** (**not False** é **True**) de modo que **True and True** dá como resultado **True**.

Embora esse tipo de estrutura seja muito útil em uma ampla variedade de situações, outra estrutura iterativa, a instrução **for**, pode ser usada em conjunto com muitas das coleções do Python. A instrução **for** pode ser usada para iterar sobre os membros de uma coleção, desde que a coleção seja uma sequência. Por exemplo:

```
>>> for item in [1,3,6,2,5]:
...    print(item)
...
1
3
6
2
5
```

Atribua cada valor sucessivo na lista [1,3,6,2,5] à variável do **item**. Em seguida, o corpo da iteração é executado. Isso funciona para qualquer coleção que seja uma sequência (listas, tuplas e strings).

Um uso comum da instrução **for** é implementar uma iteração definida em um intervalo de valores. A seguinte instrução:

```
>>> for item in range(5):
...   print(item**2)
...
0
1
4
9
16
>>>
```

Ele executará a função **print** cinco vezes. A função **range** retornará um objeto range representando a sequência "0,1,2,3,4" e cada valor será atribuído à variável **item**. Este valor é então elevado ao quadrado e impresso na tela.
As instruções de seleção permitem que os programadores façam perguntas e, com base no resultado, executem ações diferentes. A maioria das linguagens de programação fornece duas versões dessa estrutura útil: **ifelse** e **if**. Um exemplo simples de uma seleção binária usando a instrução **ifelse** é a seguinte:

```
if n<0:
  print("Desculpe, o valor é negativo")
else:
  print(math.sqrt(n))
```

Neste exemplo, o objeto referenciado por **n** é verificado para ver se é menor que zero. Em caso afirmativo, uma mensagem é impressa indicando que é negativo. Caso contrário, a instrução executa a cláusula **else** e calcula a raiz quadrada.

As estruturas de seleção, como qualquer outra estrutura de controle, podem ser aninhadas para que o resultado de uma pergunta ajude a decidir se deve ser feita a próxima. Por exemplo, suponha que **pontuacao** seja uma variável que contém uma referência a uma pontuação de teste de ciência da computação.

```
if pontuacao>= 90:
  print('A')
else:
  if pontuacao>= 80:
        print('B')
  else:
        if pontuacao>= 70:
        print('C')
        else:
        if pontuacao>= 60:
        print('D')
        else:
        print('F')
```

Este fragmento classificará um valor denominado **pontuacao** imprimindo a qualificação qualitativa obtida. Se a pontuação for maior ou igual a 90, a instrução imprimirá **"A"**. Se não for (**else**), a próxima pergunta é feita. Se a pontuação for maior ou igual a 80, então deve estar entre 80 e 89, pois a resposta da primeira questão foi falsa. Neste caso, a letra **"B"** será impressa. Você pode ver que o padrão de recuo do Python ajuda a entender a associação entre **if** y **else** sem a necessidade de elementos de sintaxe adicionais. Uma sintaxe alternativa para esse tipo de seleção aninhada usa a palavra-chave **elif**. O **else** e **if** abaixo são combinados para eliminar a necessidade de níveis de aninhamento adicionais. Observe que o último **else** ainda é necessário para fornecer o caso padrão, caso todas as outras condições falhem.

```
if pontuacao>= 90:
  print('A')
elif pontuacao>= 80:
  print('B')
elif pontuacao>= 70:
  print('C')
elif pontuacao>= 60:
  print('D')
else:
  print('F')
```

Python também tem uma estrutura de seleção unidirecional, a instrução **if**. Com esta declaração, se a condição for verdadeira, uma ação é executada. Caso a condição seja falsa, o processamento simplesmente continua com a instrução que segue o **if.** Por exemplo, o trecho a seguir verificará primeiro se o valor de uma variável **n** é negativo. Se for, então é modificado pela função de valor absoluto. Seja qual for o caso, a próxima ação é calcular a raiz quadrada.

```
if n<0:
  n = abs(n)
print(math.sqrt(n))
```

Voltando às listas, existe um método alternativo de criar uma lista que usa estruturas de iteração e seleção, conhecido como **compreensão de lista.** Uma compreensão de lista permite que você crie facilmente uma lista com base em alguns critérios de processamento ou seleção. Por exemplo, se quiséssemos criar uma lista dos primeiros 10 quadrados perfeitos, poderíamos usar uma instrução **for**:

```
>>> listaQuadrados=[]
>>> for x in range(1,11):
        listaQuadrados.append(x*x)
>>> listaQuadrados
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
>>>
```

Usando a compreensão da lista, podemos fazer a mesma coisa em uma única etapa:

```
>>> listaQuadrados=[x*x for x in range(1,11)]
>>> listaQuadrados
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
>>>
```

A variável **x** recebe os valores de 1 a 10 especificados pela estrutura **for**. O valor de **x*x** é calculado e adicionado à lista que está sendo construída. A sintaxe geral para uma compreensão de lista também permite adicionar critérios de seleção para que apenas determinados itens sejam adicionados. Por exemplo:

```
>>> listaQuadrados=[x*x for x in range(1,11) if x%2 != 0]
>>> listaQuadrados
[1, 9, 25, 49, 81]
>>>
```

Essa compreensão de lista construiu uma lista contendo apenas os quadrados de números ímpares no intervalo de 1 a 10. Qualquer sequência que suporte iteração pode ser usada em uma compreensão de lista para construir uma nova lista.

```
>>>[letra.upper() for letra in 'estruturas' if letra not in 'aeiou']
['S', 'T', 'R', 'C', 'T', 'R', 'S']
>>>
```