



Curso Técnico Nível Médio Subsequente

Informática Para Internet

Fundamentos de Lógica e Algoritmos

Aula 08
Estruturas Repetição e Listas

Bruno Emerson Gurgel Gomes

Instituto Federal de Educação, Ciência e Tecnologia
do Rio Grande do Norte

Natal-RN

2015

Presidência da República Federativa do Brasil

Ministério da Educação

Secretaria de Educação a Distância

Este Caderno foi elaborado em parceria entre o Instituto Federal de Educação, Ciência e Tecnologia e o Sistema Escola Técnica Aberta do Brasil – e-Tec Brasil.

Equipe de Elaboração
Cognitum

Projeto Gráfico
Eduardo Meneses e Fábio Brumana

Coordenação Institucional
COTED

Diagramação
Joaci de Paula

Professor-autor
Bruno Emerson Gurgel Gomes

Ficha catalográfica

B633c Barros, Thiago Medeiros.

Curso Técnico Nível Médio Subsequente Informática para Internet : Fundamentos de Lógica e Algoritmos - Aula 08 : Estruturas repetição e listas / Bruno Emerson Gurgel Gomes. – Natal : IFRN Editora, 2015.

25 f. : il. color.

1. Fundamentos de Lógica e Algoritmos - EaD. 2. Estruturas de repetição. 3. Estrutura de dados - Lista. 4. Linguagem Python. I. Título.

RN/IFRN/EaD

CDU 004.421

Ficha elaborada pela bibliotecária Edineide da Silva Marques, CRB 15/488

Apresentação da disciplina

Prezado(a) aluno(a), na aula anterior vimos conceitos e instruções bastante importantes para a programação. Destaco inicialmente os blocos de comandos. Um bloco é composto por um ou mais comandos que fazem parte de alguma instrução e devem ser executados em sequência. Podemos ter blocos, por exemplo, na instrução IF (se). O início de um bloco é marcado por ":" (dois pontos).

É importante que você lembre que todas as instruções que fazem parte de um bloco de devem ser indentadas. Para indentar uma instrução, você deve usar a tecla TAB do teclado ou inserir uma quantidade de espaços em branco. Isso faz com que os comandos em um bloco fiquem recuados mais à direita. Observe que se há mais de uma instrução em um bloco, cada uma deve ser inserida em uma linha diferente e no mesmo alinhamento de texto (identação) que a instrução anterior.

Na aula anterior, exploramos a decisão com IF e suas variações com ELSE (senão) e ELSIF (senão-se). É importante lembrar que o IF deve ser usado sempre que precisamos tomar uma decisão, ou simplesmente fazer um teste. Nesta aula, vamos aprender a uma nova instrução que irá economizar muito esforço de programação e permitir resolver mais problemas. Trata-se da repetição, que pode ser feita por meio dos comandos WHILE (enquanto) e FOR (para). Além disso, vamos trabalhar com listas de informações dos mais diversos tipos.

Aula 8 - Estruturas Repetição e Listas

Objetivos

Objetivo desta aula é conhecer em detalhes as estruturas de repetição e trabalhar com listas. Dessa forma, ao final da aula, o aluno deve estar apto a:

Compreender os princípios e técnicas envolvidas na repetição de instruções;

Conhecer e aplicar as estruturas WHILE e FOR para realizar a repetição;

Aprender a criar listas e usar algumas funções da linguagem *Python* para obter informações sobre uma lista e facilitar a sua manipulação no programa.

Desenvolvendo o conteúdo

Algoritmos e Programação de computadores

Na aula anterior, foi realizada uma breve introdução para motivar o uso de estruturas de repetição. Apresentamos o programa do cálculo da média das notas de uma turma de 8 alunos (Código 1). Esse é um tipo de programa ideal para aplicarmos uma repetição, pois há um padrão que “se repete” em várias instruções.



Figura 1: Repetição

Observe com atenção o Código 1. Nas linhas 1 a 17 é possível notar que há um padrão repetitivo. A cada linha, uma nova nota é lida e somada às notas lidas anteriormente. No caso, usamos a variável “soma” para fazer esse papel de acumulador (ou “somador”) de notas. Na linha 1 ela é iniciada com o valor 0 (zero), indicando que nenhuma nota foi lida. Na linha 2 a primeira nota é lida e salva na variável “nota1”. Esse valor é adicionado à variável soma na linha 3 ($soma = 0 + nota1$). De modo semelhante, na linha 4 é lida a segunda nota e na linha 5 é feita a inclusão dessa nota à variável soma. Nesse momento “soma” contém a soma da primeira e da segunda nota. Esse processo se repete até que a oitava nota seja adicionada. Sendo assim, é possível calcular a média da turma, que é a soma das notas lidas (o valor acumulado na variável “soma”) dividido pela quantidade de notas (linha 18).

Código 1 – Média das notas de uma turma de 8 alunos

1. soma = 0
2. nota1 = float (input (“Nota1: ”))
3. soma = soma + nota1
4. nota2 = float (input (“Nota2: ”))
5. soma = soma + nota2
6. nota3 = float (input (“Nota3: ”))
7. soma = soma + nota3
8. nota4 = float (input (“Nota4: ”))
9. soma = soma + nota4
10. nota5 = float (input (“Nota5: ”))
11. soma = soma + nota5
12. nota6 = float (input (“Nota6: ”))

13.soma = soma + nota6

14.nota7 = float(input("Nota7: "))

15.soma = soma + nota7

16.nota8 = float(input("Nota8: "))

17.soma = soma + nota8

18.média = soma / 8

19.print("Média da turma=", média)

Vamos, neste momento, reescrever este programa usando uma estrutura de repetição que irá facilitar bastante o nosso trabalho. Trata-se da estrutura WHILE (ou, em português, "enquanto").

Estrutura de repetição WHILE ("enquanto")

Permite repetir um bloco de código um certo número de vezes determinado pela avaliação de uma expressão (condição).

A forma (sintaxe) do comando WHILE em Python é a seguinte:

```
While condição;  
  
    comandos
```

Na instrução WHILE, a condição é uma expressão que deve devolver verdadeiro (*True*) ou falso (*False*). Os comandos (ou *instruções*) dentro do corpo da estrutura WHILE são executados (ou repetidos) ENQUANTO a condição for verdadeira (*True*). Quando a condição se torna falsa (*False*), os comandos não são executados e o programa continua depois da instrução WHILE.

LEMBRE-SE

Um bloco de código corresponde a uma ou mais instruções da linguagem que estão relacionadas dentro de uma mesma estrutura. O início do bloco é determinado por ":" (sinal de dois pontos) e o restante do bloco corresponde a instruções em uma mesma indentação (alinhadas por uma mesma tabulação ou o mesmo número de espaços).

O Código 2 a seguir exibe a versão do programa da média das notas de 8 (oito) alunos usando a instrução *while*.

Código 2 – Média das notas de uma turma de 8 alunos com WHILE

1. soma = 0
2. quantidade = 1
3. while quantidade <= 8:
4. nota = float(input("Nota " + str(quantidade) + ": "))
5. soma = soma + nota
6. quantidade = quantidade + 1
7. média = soma / 8
8. print("Média da turma=", média)

Você deve ter percebido algo interessante. O Código 2, que usa repetição, é muito menor que o Código 1. Pois bem, essa é uma das vantagens de se usar a estrutura de repetição para esse tipo de problema. Todo o código que tinha um padrão para ser repetido, no caso a leitura de cada uma das oito notas e a soma delas, foi inserido dentro do bloco da estrutura WHILE (repita).

Vamos passear pelo código. As linhas 1 e 2 fazem a inicialização das variáveis *soma* e *quantidade*, respectivamente. *Soma* recebe o valor zero, pois ela será um acumulador de notas. Sendo assim, o valor zero representa que, naquele momento, nenhuma nota foi adicionada à variável. Por sua vez, a *quantidade* tem o papel de estabelecer a condição do WHILE, determinando assim quantas repetições serão feitas. Ela recebe na linha 2 o valor 1 e terá que ser atualizada até o valor 8, desse modo, permitindo repetir o bloco do WHILE oito vezes, que é o que precisamos para ler e somar as notas dos oito alunos.

Quando o programa chega na linha 3, é feito o teste da expressão do WHILE para determinar se as instruções em seu corpo serão executadas. No caso, a expressão (*quantidade* <= 8) retorna verdadeiro, pois ao substituir o valor que a variável *quantidade* tem no momento, o número 1, temos como resultado a expressão $1 \leq 8$ (1 é menor ou igual que 8). Desse modo, sendo a expressão verdadeira, as instruções das linhas 4 a 6 serão executadas.

A linha 4 do programa faz a leitura da nota. Se for a primeira vez que o programa estiver executando, será a primeira nota. Perceba que no corpo do *input* temos três textos que são unidos com o operador +. Esse operador, no caso de textos, serve exatamente para juntá-los, formando assim um só texto. No caso, o objetivo foi gerar a frase "Nota 1:", "Nota 2:" e assim por diante, dependendo do valor da variável *quantidade*.

A linha 5 acumula o valor de cada nota lida na variável *soma* e a linha 6 atualiza o valor da variável *quantidade*. Perceba que essa atualização é essencial, pois sem ela o seu programa iria ficar preso no WHILE para sempre, uma vez que a expressão de teste seria sempre verdadeira ($1 \leq 8$). Você pode fazer esse teste rodando o seu programa sem a instrução da linha 6.

Nesse programa, atualizamos a variável *quantidade* de 1 em 1 (*quantidade* = *quantidade* + 1), até 8, pois queremos repetir oito vezes. Na primeira vez que essa instrução for executada, ela terá o seu valor atualizado para 2.

Nesse momento vemos a repetição acontecer. Após atualizar o valor da variável *quantidade*, que é o último comando do corpo do WHILE (perceba pela indentação), o programa volta para a linha 3, para testar novamente a expressão *quantidade* <= 8. Como *quantidade* foi atualizada para 2 (dois), então a expressão resultante $2 \leq 8$ retorna verdadeiro e as instruções das linhas 4 a 6 são executadas novamente, dessa vez para ler a segunda nota.

Esse processo de verificar a condição do WHILE, executar as instruções dentro do WHILE, atualizar a variável quantidade e voltar para o teste do WHILE se repete até que o valor de quantidade seja 9. Nesse momento, a expressão $9 \leq 8$ devolve o valor falso (*False*) e o programa sai do WHILE e vai para a próxima instrução após ele, na linha 7, que irá calcular a média. Por fim, o programa termina na linha 8, com a impressão na tela do resultado da média.

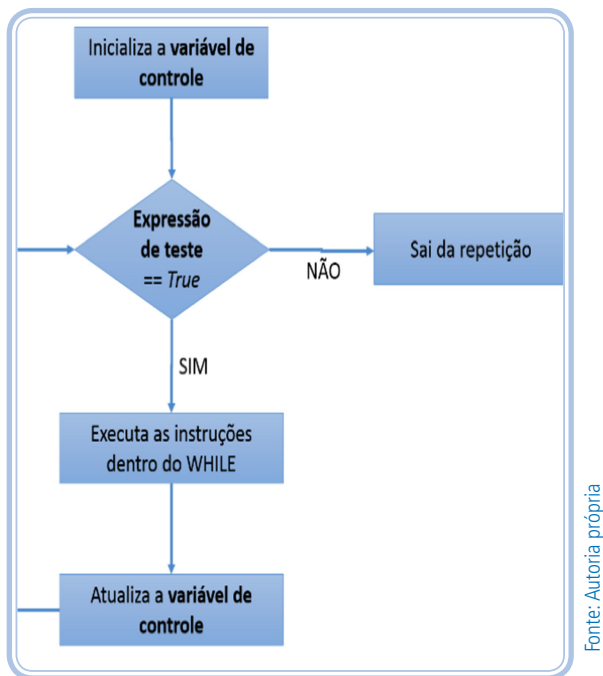
Enfim, descrevemos o nosso programa completo usando a repetição WHILE. Há alguns pontos que você precisa se concentrar para aprender a usar o WHILE em diversas situações. Primeiro, você deve sempre criar uma expressão de teste que em algum momento termine. Não há necessariamente uma receita para isso, depende do problema que você quer resolver. No nosso caso, usamos a variável quantidade para fazer esse controle da expressão e é por isso que esse tipo de variável é chamado de variável de controle da repetição. Inicializamos com o valor 1 e criamos a expressão (quantidade ≤ 8) para irmos de 1 a 8, repetindo assim 8 vezes, uma vez que quantidade é atualizada em 1 a cada repetição.

Perceba que podemos muito bem inicializar a variável quantidade com o valor 0 (zero) e repetir até 7, alterando a expressão de teste para *quantidade* < 8 ou, ainda, *quantidade* ≤ 7 . Isso irá gerar os valores 0, 1, 2, 3, 4, 5, 6, 7, fazendo com que ainda tenhamos 8 repetições. Nós tomamos a decisão de inicializar quantidade com 1 apenas para permitir a impressão dos textos "Nota 1:", "Nota 2:", no comando *input* que pede a entrada das notas.

Outro ponto que você deve prestar atenção é na atualização da variável de controle, pois é ela que faz com que em algum momento o WHILE termine, quando ela se tornar falsa (*False*). Geralmente, essa expressão é a última instrução do WHILE, mas isso não precisa ser seguido à risca. Caso você esqueça de atualizar a variável de controle, a repetição e, conseqüentemente, o seu programa, nunca irão terminar.

O funcionamento da estrutura WHILE pode ser resumido a partir do diagrama da Figura 2. Na primeira etapa, você deve fornecer o valor inicial da variável de controle. Isso ocorre apenas uma vez, antes de chegarmos a etapa de teste da expressão do WHILE. Nessa etapa, por sua vez, é feito o teste para saber se a repetição deve ser ou não realizada. Caso a expressão devolva o valor verdade verdadeiro (*True*), as instruções no corpo do WHILE são realizadas. Uma dessas instruções geralmente é a atualização da variável

de controle. Após essa atualização, o teste é feito novamente. Esse processo se repete até que a expressão de teste se torne falsa e o programa vai para a próxima instrução após o WHILE.



Fonte: Autoria própria

Figura 2: Funcionamento da estrutura WHILE

Antes de você tentar resolver seus próprios problemas com o WHILE, vamos praticar um pouco mais com alguns exemplos. Suponha que você queira imprimir todos os números pares de 2 até 100, incluindo 2 e 100. Pois bem, vamos criar juntos essa repetição. Primeiro, qual seria o valor inicial da repetição? Se queremos ir de 2 até 100, então o valor inicial é 2. Suponha que a variável de controle se chama "par". Nesse caso, ela deve receber o inteiro 2, o primeiro par do intervalo ($par = 2$). O valor final é 100. Sendo assim, nossa expressão de teste deve ir até 100 ($par \leq 100$). Agora falta apenas trabalharmos com a atualização da variável "par", que deve crescer de 2 em 2, até chegar em 100 ($par = par + 2$).

Código 3 – Impressão de todos os pares de 2 a 100

1. `par = 2`
2. `while par <= 100:`
3. `print (par)`
4. `par = par + 2`

Vamos mudar um pouco o Código 3, de modo que o valor final do intervalo, que antes era 100, agora deve ser fornecido pelo usuário. Ou seja, você deve pedir a leitura desse valor usando *input*. É importante que você verifique se o valor lido está correto. Nesse caso, o valor superior deve ser maior que 2, que é o valor inicial. Para tanto, é necessário apenas acrescentar um teste antes do WHILE, conforme pode ser visto no Código 4.

Código 4 – Impressão de todos os pares de 2 a um valor superior (lim)

1. **lim = int (input ("Limite superior: "))**
2. **par = 2**
3. **if lim > 2:**
4. **while par <= lim:**
5. **print (par)**
6. **par = par + 2**
7. **else:**
8. **print ("O limite superior deve ser maior que 2.")**

Até o momento, nós usamos os operadores de comparação na expressão de teste do WHILE. Na verdade, você pode usar qualquer expressão e operadores correspondentes que devolvam verdadeiro ou falso. Por exemplo, é possível testar a igualdade (==) ou a diferença (!=) de um determinado valor. É possível deixar a variável de controle verdadeira (*True*) e em algum momento torná-la falsa (*False*). Por exemplo, o Código 5 exibe um pequeno jogo que pede para o usuário adivinhar o número que foi gerado pelo programa (entre 0 e 20). Esse número é gerado com a função *randint*.

Código 5 – Adivinhe um número (entre 0 e 20)

1. **from random import ***
2. **lido = int (input ("Digite um número entre 0 e 20: "))**

3. `gerado = randint (0, 21) #randint: gera um número inteiro entre 0 e 20`
4. `while` `gerado != lido`:
5. `if` `lido > gerado`:
6. `print` (“O número é menor que”, `lido`, “. Tente novamente!”)
7. `else`:
8. `print` (“O número é maior que”, `lido`, “. Tente novamente!”)
- 9.
10. `lido = int (input (“Digite um número entre 0 e 20: ”))`
11. `print` (“Você acertou. Parabéns!”)

Na primeira linha do programa de adivinhação de um número, temos a inclusão de uma biblioteca de funções, denominada *random*. Uma função nada mais é que um trecho de código que executa uma ação específica. Já trabalhamos com funções, como *print* e *input*. A linha 1 é necessária para acessarmos a função *randint*, que recebe dois valores, o primeiro é o valor inicial do intervalo no caso 0 e o segundo é o valor final menos 1. Ou seja, se queremos que a função gere e devolva um número qualquer entre 0 e 20, temos que colocar, como na linha 3, *randint(0, 21)*.

Na linha 2 é feita a leitura do número digitado pelo usuário do programa. Esse valor será comparado com o valor gerado pela função *randint* na linha 3. Ainda na linha 3, o texto após o sinal de “#” é um comentário, ele tem a função apenas de documentar o programa, tornando mais claro o significado de algum trecho de código para quem está lendo o programa.

O WHILE da linha 4 irá repetir enquanto o valor digitado pelo usuário for diferente do valor gerado, ou seja, enquanto o usuário não adivinhar o número gerado. Para ajudar o jogador, é inserido um teste na linha 4. Ele vai dizer se o número digitado é maior ou menor que o número lido. A última instrução do WHILE pede novamente a leitura, até que o usuário digite o mesmo número que foi gerado. Quando isso acontecer, o teste do WHILE será falso e será impressa a mensagem da linha 11 indicando que o jogador acertou o número.

Na primeira linha do programa de adivinhação de um número, temos a inclusão de uma biblioteca de funções, denominada `random`. Uma função nada mais é que um trecho de código que executa uma ação específica. Já trabalhamos com funções, como `print` e `input`. A linha 1 é necessária para acessarmos a função `randint`, que recebe dois valores, o primeiro é o valor inicial do intervalo no caso 0 e o segundo é o valor final menos 1. Ou seja, se queremos que a função gere e devolva um número qualquer entre 0 e 20, temos que colocar, como na linha 3, `randint(0, 21)`.

Na linha 2 é feita a leitura do número digitado pelo usuário do programa. Esse valor será comparado com o valor gerado pela função `randint` na linha 3. Ainda na linha 3, o texto após o sinal de “#” é um comentário, ele tem a função apenas de documentar o programa, tornando mais claro o significado de algum trecho de código para quem está lendo o programa.

O `WHILE` da linha 4 irá repetir enquanto o valor digitado pelo usuário for diferente do valor gerado, ou seja, enquanto o usuário não adivinhar o número gerado. Para ajudar o jogador, é inserido um teste na linha 4. Ele vai dizer se o número digitado é maior ou menor que o número lido. A última instrução do `WHILE` pede novamente a leitura, até que o usuário digite o mesmo número que foi gerado. Quando isso acontecer, o teste do `WHILE` será falso e será impressa a mensagem da linha 11 indicando que o jogador acertou o número.



ATIVIDADE

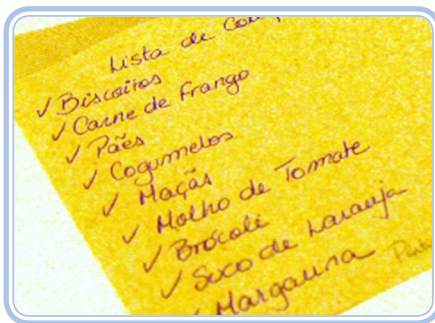
1. Faça um programa, usando repetição, que imprima os números de 1 até 30.
2. Modifique o programa da questão 1 e imprima os números de 1 até um valor maior que 1. Esse valor deve ser fornecido pelo usuário (use a instrução `input` para ler o valor).
3. Faça um programa que leia os pontos dos últimos 10 jogos de um time de futebol e imprima o total de pontos do time nesses 10 jogos. Ou seja, você deve imprimir a soma de todos os pontos lidos. Os pontos podem ser 3 (três), em caso de vitória, 1 (um) em caso de empate e 0 (zero) no caso de derrota.

Listas

Após apresentar a estrutura de repetição WHILE, vamos iniciar o estudo de uma nova estrutura de dados, a lista. Não se preocupe, que o estudo anterior também irá se aplicar aqui, pois precisamos das estruturas de repetição para percorrer as listas.

Trabalhar com listas nos permite resolver vários problemas. Você pode criar listas de cada um dos tipos básicos e até mesmo de outras listas. Mas, o que vem a ser uma lista? Ora tenho certeza que você já criou uma lista alguma vez na vida. Podemos citar diversos exemplos, como uma lista de compras no supermercado (Figura 3), a lista dos seus filmes favoritos, a lista das matérias que você precisa estudar mais.

Em *Python*, a lista é uma estrutura muito flexível e bastante simples de criar e usar.



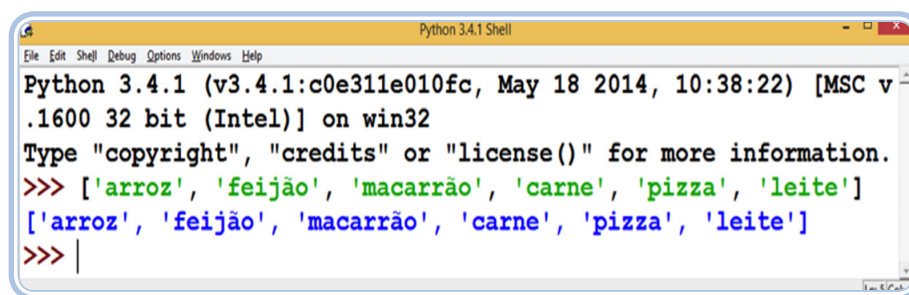
Fonte: <http://goo.gl/1fhp0>

Figura 3: Lista de compras

Lista

Uma lista corresponde a zero ou mais valores (elementos) definidos entre colchetes []. Usa-se a vírgula (,) para separar os elementos no caso da lista ser composta por mais de um elemento. A lista é dita vazia caso ela não possua nenhum elemento.

Vamos começar com um exemplo simples, a lista de compras no mercado. Pois bem, vamos supor que a lista é composta pelos seguintes produtos: arroz, feijão, macarrão, carne, pizza e leite. Essa é uma lista de *strings* (textos) constante, pois já sabemos de antemão quais são os seus elementos. A Figura 3 exibe a lista criada no interpretador. Perceba que o interpretador devolve como resultado a própria lista. Nesse caso, como não salvamos a lista em nenhum lugar, não podemos fazer nada com ela.



```
Python 3.4.1 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 10:38:22) [MSC v
.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ['arroz', 'feijão', 'macarrão', 'carne', 'pizza', 'leite']
['arroz', 'feijão', 'macarrão', 'carne', 'pizza', 'leite']
>>> |
```

Fonte: Autoria própria.

Figura 4: Lista de compras criada no interpretador Idle

Vamos agora salvar a lista em uma variável para manipulá-la usando algumas propriedades. No Código 6, salvamos a lista de compras na variável `compras`. Uma primeira propriedade sobre listas é que os seus elementos ocupam uma posição determinada. O primeiro elemento está na posição 0, o segundo na posição 1, e o último na posição *tamanho da lista* - 1. Assim, se você quiser obter o terceiro elemento da lista de compras, você deve fazer `compras[2]` (linha 2). Observe que você sempre deve seguir esse mesmo padrão (nome da lista, abre colchetes, posição, fecha colchetes) quando quiser obter um elemento da lista.

Código 6 – Lista de compras salva em uma variável

1. `compras = ['arroz', 'feijão', 'macarrão', 'carne', 'pizza', 'leite']`
2. `print (compras[2])` #irá devolver o valor 'macarrão'
3. `tamanho = len(compras)`
4. `último = tamanho - 1`
5. `print(compras[último])`

Caso você queira obter o tamanho da lista você deve usar a função "`len`", passando para ela o nome da lista, como na linha 3 do Código 6 (`len(compras)`). Nós usamos essa informação para obter o último elemento da lista, que está na posição tamanho da lista - 1 (linhas 4 e 5).

Da mesma forma que podemos acessar os elementos de uma lista pela sua posição, é possível também modificar esses elementos. Se você quiser trocar o item "macarrão" por "iogurte", você deve fazer `compras[2] = 'iogurte'`. Agora, se você der um `print` na lista (`print(compras)`) verá que na posição 3 (`compras[2]`) o item "macarrão" foi substituído por "iogurte".

Preste atenção ao fato de que você só pode acessar um elemento que esteja na lista, ou seja, entre a posição 0 (primeiro elemento) e a posição *tamanho - 1* (último elemento). Se você tentar acessar um elemento fora dos limites da lista, o programa irá terminar com um erro.

Se você quiser adicionar um elemento em uma lista já criada anteriormente, você pode usar a palavra *append*. O *append* adiciona o novo elemento ao final da lista. Caso queira remover definitivamente um elemento de uma determinada posição, você deve usar a palavra *del*, na forma como é exemplificada no código 7.

Código 7 – Lista de compras – adicionando e removendo elementos

1. `compras = ['arroz', 'feijão', 'macarrão', 'carne', 'pizza', 'leite']`
2. `compras.append('sabão')` #adiciona o elemento 'sabão'
3. `print (compras)`
4. `del (compras [1])` #remove o elemento feijão
5. `print (compras)`

No Código 7, ao adicionar o elemento “sabão” na linha 3 usando *append* a lista será composta da seguinte forma: ['arroz', 'feijão', 'macarrão', 'carne', 'pizza', 'leite', 'sabão']. Por sua vez, na linha 4 removemos o elemento da posição 1, ou seja, o “feijão”, tendo como resultado a lista: ['arroz', 'macarrão', 'carne', 'pizza', 'leite', 'sabão'].

Com o uso do *append*, é possível construirmos uma lista a partir da lista vazia. Por exemplo, vamos gerar a lista dos números ímpares de 3 até 15. Para isso, vamos precisar também de uma estrutura de repetição, conforme pode ser visto no Código 8.

Código 8 – Gera uma lista de números ímpares de 3 a 15

1. `ímpares = []`
2. `n = 3`

3. **while** $n \leq 15$:

4. `ímpares.append(n)`

5. `n = n + 2`

6. **print** (ímpares)

Observe que na linha 1 do Código 8 foi criada uma lista vazia, denominada de *ímpares*. A variável “n” da linha 2 recebe o primeiro número ímpar. Ela serve de variável de controle do WHILE, ao mesmo tempo que é usada para construir a lista. Na linha 3 temos o teste do WHILE, que irá parar apenas quando “n” atingir o valor 15. Na linha 4, o *append* é usado para inserir cada elemento, um por vez em cada repetição. A linha 5 atualiza o valor de “n” para o próximo número ímpar. Por fim, a linha 6, que está fora do WHILE, realiza a impressão da lista gerada.



ATIVIDADE

1. Crie uma lista com as notas de 5 alunos, como por exemplo: [80.0, 76.0, 56.8, 68.0, 92.5].
2. Usando *append*, adicione mais 3 notas à sua lista.
3. Usando *del*, remova a primeira e a terceira notas.
4. Usando repetição (WHILE), percorra toda a lista e aumente o valor de cada nota em 5.0 (cinco) pontos.

Como você pode perceber, o assunto de lista é bastante vasto. Ainda há diversos recursos, operadores e funções que podemos usar. Você pode se aprofundar mais no assunto usando a bibliografia sugerida ao final desta aula. Por hora, vamos ver mais alguns recursos interessantes. Um desses recursos são os operadores **in** e **not in** que, em português, seriam traduzidos por dentro (*in*) e fora, ou, literalmente, “não dentro” (*not in*). Ou seja, usamos “in” se queremos saber se algum elemento está na lista e “not in” se queremos perguntar se determinado elemento não pertence à lista.

Código 9 – Procura por um nome em uma lista de nomes

```
1. pessoas = ['Maria', 'João', 'Pedro', 'Letícia', 'Bruno', 'Francisco']  
  
2. nome = input ("Digite um nome: ")  
  
3. x = 0  
  
4. encontrado = False  
  
5. while x <= len (pessoas) - 1:  
  
6.     if pessoas [x] == nome:  
  
7.         encontrado = True  
  
8.         x = x + 1  
  
9. if encontrado:  
  
10.    print (nome, "está na lista")  
  
11. else:  
  
12.    print (nome, "não está na lista")
```

O Código 9 é bastante interessante. Ele realiza a busca de um nome em uma lista de nomes. O nome a ser buscado deve ser fornecido pelo usuário do programa na linha 2. A variável "x" na linha 3 é usada para representar as posições da lista, de modo que possamos percorrer cada uma das posições. Ela também é usada para controlar as repetições do WHILE, indo de 0 até o tamanho da lista menos 1. A variável "encontrado" da linha 4 representa se o nome foi ou não encontrado. Ela foi inicializada com falso (*False*) representando que o nome não foi encontrado. No corpo do WHILE temos um teste (linha 6) para verificar se o nome da posição atual (x) da lista é igual ao nome que estamos procurando. Caso, em algum momento, seja encontrado um nome igual, a variável "encontrado" recebe o valor verdade verdadeiro (*True*). Na linha 8, o valor de "x" é atualizado para que se possa obter o próximo nome na lista. Por fim, o IF da linha 9 irá imprimir se o nome foi ou não encontrado com base no valor da variável encontrado.

O programa do Código 9 pode ficar bem mais curto se usarmos o recurso do operador "in". Você pode ver como isso é possível no Código 10.

Código 10 – Procura por um nome em uma lista de nomes (versão com "in")

1. `peessoas = ['Maria', 'João', 'Pedro', 'Letícia', 'Bruno', 'Francisco']`
2. `nome = input ("Digite um nome: ")`
3. `if nome in pessoas:`
4. `print (nome, "está na lista")`
5. `else:`
6. `print (nome, "não está na lista")`

Em uma primeira vista no Código 10 dá para perceber que o programa ficou bastante enxuto com relação ao código 9. Pois bem, isso foi possível pelo uso da palavra **in** na linha 3. Se traduzirmos a linha 3, seria algo como "se o nome está na lista de pessoas". Ou seja, o operador "in" automaticamente faz a busca para você e devolve verdadeiro se o valor foi encontrado e falso caso contrário.

Para finalizarmos esta aula e o estudo de *Python* nesse curso, vamos mostrar a estrutura de repetição FOR (para). Ela é mais flexível que o WHILE ao trabalharmos com listas, pois elimina a necessidade da expressão de teste. Basta que você use uma variável de controle e, com o operador "in" indique a lista que será percorrida. O que o FOR faz é, a cada repetição, atribuir o elemento atual da lista, começando com o primeiro elemento, à variável de controle até que a lista termine. O Código 9 como o uso do FOR ficaria da forma como apresentado no código 11.

Código 11 – Procura por um nome em uma lista de nomes (uso do FOR)

1. `peessoas = ['Maria', 'João', 'Pedro', 'Letícia', 'Bruno', 'Francisco']`
2. `nome = input ("Digite um nome: ")`

3. **for** x **in** pessoas:

6. if x == nome:

7. encontrado = **True**

8. **if** encontrado:

9. **print** (nome, "está na lista")

10. **else**:

11. **print** (nome, "não está na lista")

No caso do Código 11, com o uso do FOR eliminamos a necessidade de inicializar e atualizar a variável "x". Isso é feito automaticamente pelo FOR, que atribui a "x" o elemento atual de "pessoas". Assim, ao trabalhar com listas e a depender do problema, você pode ficar livre para escolher se quer usar WHILE ou FOR.

RESUMINDO

Caro aluno, esta foi a nossa última aula neste curso. Nessa aula, exploramos o conceito de repetição de trechos de código em *Python*. A técnica de repetição pode ser aplicada à resolução de diversos problemas, diminuindo a necessidade de variáveis e a repetição de código com instruções semelhantes. Vimos as estruturas WHILE e FOR para realizar a repetição. A primeira, de uso mais geral, permite realizar a repetição mesmo quando não sabemos de antemão quantas repetições serão necessárias. A segunda é aplicada a listas, agilizando o processo de percorrer toda uma lista de valores. Por falar em listas, essa importante estrutura de dados foi assunto da segunda parte desta aula. As listas facilitam o trabalho de definição e manipulação de um conjunto de valores relacionados, facilitando assim a resolução de diversos problemas.

LEITURAS COMPLEMENTARES

(BOSON TREINAMENTOS, 2015a) e (BOSON TREINAMENTOS, 2015b) Vídeo aulas sobre WHILE, FOR e outros comandos – Nas duas páginas que estão nas referências você encontra informações sobre os conceitos básicos de estruturas de repetição em *Python*. Alguns recursos que não foram trabalhados nessa aula também são explicados, como a saída da repetição com *break* e a geração de um intervalo para o FOR com o a palavra *range*.

(MELANDA, 2015a), (MELANDA, 2015b) e (DEVLINUXBR, 2015) – Nesses *sites* você também encontra explicações sobre as estruturas vistas nesta aula.

(STUMM JR., 2015) – Neste *site* você encontra exemplos de diversas funções e operadores que podem ser aplicados a listas.



AVALIANDO SEUS CONHECIMENTOS

1. Faça um algoritmo que leia **n** valores e calcule a média aritmética desses valores.
2. Faça um algoritmo que leia uma quantidade de N números que serão digitados e, ao final do processamento do algoritmo, mostre qual foi o maior número digitado.

Exemplo:

Se o usuário digitou 5 como a quantidade de números (N), devem ser lidos 5 números. Ao final, o algoritmo deve exibir qual desses cinco números é o maior. Dica: *leia o primeiro número fora da repetição e diga que ele é o maior (crie uma variável chamada maior), uma vez que foi o primeiro. Dentro da repetição você deve comparar os demais números lidos com o valor que está na variável maior. Se o valor for maior, então maior recebe esse novo valor.*

5 90 123 -232 0 → maior número = 123

3. Em uma eleição presidencial existem 4 candidatos. Os votos são informados através de código. Os dados utilizados para a contagem dos votos obedecem à seguinte codificação:

a) 1, 2, 3, 4 = Voto para os respectivos candidatos

1 para Cecília Meireles

2 para Ariano Suassuna

3 para Machado de Assis

4 para Graciliano Ramos

b) **5** para voto em branco

c) **1234** para encerrar a votação

d) Qualquer outro valor diferente dos anteriores para anular o voto

Faça um programa que calcule e escreva:

- O total de votos para cada candidato;
- O total de votos nulos;
- O total de votos em branco.

Para finalizar a entrada dos dados (a repetição) e exibir o resultado da votação, use o valor 1234 (mil duzentos e trinta e quatro).

4. Dada uma lista de inteiros, faça um programa que procure se um número está na lista. O número a ser buscado deve ser fornecido pelo usuário do programa.

5. A partir de um texto fornecido pelo usuário, conte quantos caracteres esse texto possui, sem contar os espaços em branco.

6. Leia um texto com 10 caracteres minúsculos e diga quantas vogais foram lidas.

Referências

BOSON TREINAMENTOS. **Python: loop While (Estrutura de Repetição) e Instrução break**. 2013. Disponível em: <<http://www.bosontreinamentos.com.br/programacao-em-python/19-python-loop-while-estrutura-de-repeticao-e-instrucao-break/>>. Acesso em: 01 abr. 2015.

_____. **Python: loop FOR: Estruturas de Repetição: função range**. 2013. Disponível em: <<http://www.bosontreinamentos.com.br/programacao-em-python/20-python-loop-for-estruturas-de-repeticao-funcao-range/>>. Acesso em: 01 abr. 2015.

DEVLINUXBR. **Vamos falar de Python?: Laços de repetição**. [201-?] Disponível em: <<http://devlinuxbr.blogspot.com.br/2014/09/vamos-falar-de-python-lacos-de.html>>. Acesso em: 01 abr. 2015.

MELANDA, J. C. E. **Estruturas de Repetição Parte 1: While**. [2014]. Disponível em: <<http://programeempython.blog.br/blog/estruturas-de-repeticao-parte-1-while/>>. Acesso em: 01 abr. 2015.

_____. **Estruturas de Repetição Parte 2: For**. [2014]. Disponível em: <<http://programeempython.blog.br/blog/estruturas-de-repeticao-parte-2-for/>>. Acesso em: 01 abr. 2015.

PYTHON SOFTWARE FOUNDATION. Disponível em: <<https://www.python.org/>>. Acesso em: 10 mar. 2015.

STUMM JÚNIOR, V. **Brincando com listas**. 2013. Disponível em: <<https://pythonhelp.wordpress.com/2013/06/26/brincando-com-listas/>>. Acesso em: 01 abr. 2015.

