PROGRAMAR

REVISTA PORTUGUESA DE PROGRAMAÇÃO • WWW.PORTUGAL-A-PROGRAMAR.ORG

EDIÇÃO #23 - MARÇO 2010 ISSN 1647-0710

INTRODUÇÃO A PROGRAMAÇÃO DE ANDROID

LUA 3ª PARTE : LINGUAGEM DE PROGRAMAÇÃO

PYTHON PROGRAMAÇÃO FUNCIONAL

índice

equipa PROGRAMAR

coordenadores
Joel Ramos
Pedro Abreu
Fernando Martins

editor
António Silva

capa
Sérgio Alves

redacção
Augusto Manzano
David Ferreira
Diogo Júnior
Jorge Paulino

equipa de revisão
Bruno Oliveira
Fernando Martins
Miguel Rentes

equipa de divulgação
David Ferreira

contacto
revistaprogramar
@portugal-a-programar.org

website
www.revista-programar.info

issn
1647-0710

Que futuro?

Quero começar por pedir desculpa aos nossos leitores, e em nome de toda a equipa, pelo atraso no lançamento desta edição da Revista Programar. Infelizmente, tal como aconteceu no lançamento da última edição da revista, não nos foi possível cumprir a data prevista. As razões são várias e as desculpas, se fossem importantes, certamente seriam ainda em maior número.
A verdade é que a Revista Programar está a passar por um momento difícil e crítico da sua existência. O número de lançamento já vai longe. Já se passaram vários anos e este projecto amadureceu, expandiu-se e tornou-se cada vez mais exigente. Os vários elementos da equipa, quer passada quer actual, sempre responderam da melhor forma que souberam aos vários desafios que nos foram sendo postos. Mas, actualmente, enfrentamos o maior desafio de todos até à data.

A Revista Programar nasceu de forma espontânea como um projecto colaborativo dos participantes na comunidade P@P dentro do fórum, herdou o mesmo foco, o mesmo público e o mesmo espírito: a partilha de conhecimentos sobre programação entre falantes da língua de Camões.
Foi com este espírito, de quem tem algo para dar sem esperar receber nada em troca, que a revista foi evoluindo e coleccionando vários sucessos. Número após número o feedback que nos ia chegando era cada vez melhor e mais positivo. Foram aparecendo pedidos sobre artigos, passámos além fronteiras, melhorámos o visual, tornámo-nos mais exigentes dentro da própria equipa e trabalhámos sempre com afinco em cada edição.

«A Revista Programar é vossa. Existe para todos vocês. Mas só existirá se houver quem faça a revista.»

Como sempre, o sucesso trouxe responsabilidades acrescidas.
Se o número de leitores cresceu significativamente, o mesmo não aconteceu com o número de colaboradores, com enfoque particular no número de autores.
O número de artigos propostos e entregues estagnou, ou diminuiu mesmo em algumas edições, e em alguns casos a qualidade dos mesmos levou à sua rejeição.
Não é necessário dizer o óbvio: sem autores não há artigos e sem artigos com qualidade não há revista.

É neste contexto que assumo a responsabilidade de colaborador da Revista Programar a partir deste número, acumulando assim este papel com o de revisor que já exercia. E é neste contexto que assumo, com toda a equipa da revista, este desafio. E é neste contexto que venho fazer um apelo de ajuda à comunidade P@P.
Necessitamos de conteúdo com qualidade para as próximas edições, sob pena de não haver próximas edições.
Apelo assim a todos quantos têm artigos escritos que os partilhem connosco. Todos quantos querem ajudar mas não têm tema, contactem-nos, temos pedidos de artigos mas não temos quem os escreva.

Creio não estar errado quando afirmo que não existe nenhuma revista centrada na programação e orientada para a comunidade de língua Portuguesa que tenha um espírito tão altruísta como a Revista Programar.
A Revista Programar é vossa. Existe para todos vocês. Mas só existirá se houver quem faça a revista.
Vem ajudar-nos a fazer a tua revista de programação favorita.

Fernando Martins

Windows Phone Series 7

http://www.windowsphone7series.com/

A MIcrosoft apresentou no *Mobile World Congress* o novo Windows Phone Series 7, e prevê a a sua estreia para o final de 2010. Concorrentes como o Android, esperam para ver se este será um salto nos sistemas operativos da Microsoft para smartphones capaz concorrer qualitativamente com eles.

Firefox começa a chegar aos telemóveis

http://www.mozilla.com/en-US/mobile/

Apesar de ainda estar disponivel para um número limitado de plataformas, o Firefox Mobile já pode ser descarregado no *site* da Mozilla**.**

Google Buzz

http://www.google.com/buzz

A Google lançou o Google Buzz, um serviço que permite partilhar actualizações de estado, fotos, links entre outras, utilizando parte da interface do Gmail. Apesar de tudo já se ouviram várias vozes contra este serviço, e que levaram a própria Google a fazer alterações ao Buzz, e permitir que o mesmo fosse configurado e desactivado de maneira mais fácil, devido a questões de privacidade.

Google Chrome 4 lançado

http://www.google.com/chrome

Foi lançada a versão 4 do browser da Google para a plataforma Windows que inclui entre outras funcionalidades a capacidade de sincronizar marcadores, apesar da Google afirmar ter mais de 1500 novas funcionalidades, nós não o pudemos comprovar.

Microsoft TechDays 2010

Já é público o próximo grande evento técnico da Microsoft. É destinado a todos os profissionais que trabalham em tecnologias de informação, sejam ligados à administração de sistemas, desenvolvimento de software, gestão de projecto ou de departamentos de sistemas de informação.

Este ano irá decorrer nos dias 20,21 e 22 de Abril, com mais de 40 oradores e sobre diversas tecnologias como Visual Studio, Silverlight, WPF, Exchange, SQL Server, Windows 7, Sharepoint, etc.

Até dia 5 de Março, Early Bird, o preço da inscrição é de 150€, sendo 225€ após essa data.

Um evento a não perder!

http://www.techdays2010.com

A Revista PROGRAMAR errou: Devido a um erro de edição o artigo "Lua - 2ª Parte" da edição anterior não foi correctamente colocado, e como tal vem nesta edição a sua versão corrigida. A todos os intervenientes as nossas desculpas.

VS2010: Auto-Implemented Properties

As Auto-Implemented Properties são uma forma simples e rápida de definir uma propriedade, sem a utilização do Get e Set. Esta opção já estava disponível no C# 3.0 e passou agora a fazer parte do VB10.

```
Public Property myProp As String

'Em vez de(expanded property):

Private _myPror As String
Public Property myPror() As String
Get
  Return _myPror
End Get
Set(ByVal value As String)
  _myPror = value
End Set
End Property
```

No entanto pode-se sempre usar o método "normal"(expanded property), caso seja necessário adicionar código nos métodos Get e Set, caso seja uma propriedade WriteOnly ou ReadOnly, etc.

Quando é definida uma auto-implemented property, é criado internamente (não visível) um backing field, ou seja, se a propriedade for designada como "myProp", será criada uma variável "_myProp". Isto quer dizer, que se tentarmos criar uma variável com o mesmo nome, iremos ter o seguinte erro:

property 'myProp' implicitly defines '_myProp', which conflicts with a member of the same name in class 'Form1'

Podemos ainda inicializar a propriedade por mais complexa que seja a expressão:

```
Public Property myProp1 As String = "programar"
Public Property myProp2 As Integer = 123
Public Property myProp3 As List(Of Integer) = Enumerable.Range(0, 20).ToList
Public Property myProp4() As String = New String({"a", "b", "c"})
```

VS2010 : Implicit Line Continuation

Quando uma linha de código é demasiado extensa, é necessário muitas vezes saltar de linha, de modo a que o código fique todo visível e seja mais simples de interpretar. Para o fazer é usado um "_" (underscore):

```
MessageBox.Show("Isto é um teste", _
                My.Application.Info.Title, _
                MessageBoxButtons.OK, _
                MessageBoxIcon.Warning)
```

Ora isto é algo que no Visual Studio 2010 deixou de ser necessário, podendo agora ser utilizado:

```
MessageBox.Show("Isto é um teste",
                My.Application.Info.Title,
                MessageBoxButtons.OK,
                MessageBoxIcon.Warning)
```

Obviamente que isto não pode ser efectuado em todas as situações, pois o compilador precisa de saber identificar se estamos a saltar de linha ou a iniciar uma nova instrução.

Estas são algumas das situações onde é possível o fazer:

- Após uma virgula ","
- Após um parêntese aberto "(" ou antes de um fechado ")"
- Após uma chaveta recta "{" ou antes de uma chaveta recta "}"
- Após uma concatenação "&"
- Após uma atribuição "=, &=, :=, +=, -=, *=, /=, \=, ^=, <<=, >>="
- Após um operador binário "+, -, /, *, Mod, <>, <, >, <=, >=, ^, >>, <<, And, AndAlso, Or, OrElse, Like, Xor"
- Após um operador Is ou IsNot

Entre muitas outras situações, como LINQ, XML Literals, etc.

Não te esqueças, esta página pode ser tua!
http://www.revista-programar.info/front/yourpage

Introdução à Programação para Android

O que é o Android?

O Android é um sistema operativo para dispositivos móveis desenvolvido por um consórcio de 50 empresas denominado Open Handset Alliance. Deste consórcio fazem parte empresas como a Google, HTC, Motorola, Intel, Samsung, LG, entre outras, unidas com o objectivo principal de desenvolver novas normas abertas para dispositivos móveis. O primeiro projecto opensource a ser criado por este consórcio foi efectivamente o Android. A apresentação ao público do Android ocorreu em 5 de Novembro de 2007 e o código foi distribuído com uma licença Apache 2.0 e GPLv2.

Para ser possível aos programadores desenvolverem aplicações que utilizam as capacidades do Android, é disponibilizado um SDK que permite a criação de aplicações utilizando a linguagem de programação Java. As aplicações escritas em Java são executadas pela Dalvik Virtual Machine que é uma implementação de uma Virtual Machine desenhada e optimizada para dispositivos móveis. As aplicações podem utilizar várias funcionalidades disponibilizadas pela API existentes na plataforma Android. Algumas dessas funcionalidades são dependentes do hardware.

Funcionalidades:

- Framework de aplicações - permite a reutilização e substituição de componentes
- Dalvik virtual machine - máquina virtual optimizada para dispositivos móveis
- Browser Integrado - baseado no motor de código aberto WebKit
- Gráficos Optimizados - suportado por uma biblioteca gráfica 2D e de gráficos 3D (OpenGL ES 1.0)
- SQLite - para armazenamento de dados estruturados
- Suporte Media - suporte comum para áudio, vídeo e imagem
- GSM - (dependente do hardware)
- Bluetooth, EDGE, 3G, e WiFi - (dependente do hardware)
- Camera, GPS, bússola e acelerómetro - (dependente do hardware)

Ferramentas usadas para desenvolver aplicações

Para desenvolver aplicações é necessário fazer download do SDK do site do Android (http://android.com), e de preferência ter o IDE Eclipse (http://www.eclipse.org/) instalado e configurado para desenvolver aplicações em Java. Para este IDE existe um plugin que permite uma maior rapidez e eficácia no desenvolvimento de aplicações. O plugin que dá pelo nome de Android Development Tools (ADT) também precisa de ser descarregado do site do Android e pode ser instalado através do gestor de software do eclipse. Caso existam dificuldades na instalação, no link seguinte estão detalhados todos os passos que devem ser seguidos para a instalação e configuração do SDK e IDE: http://developer.android.com/sdk/index.html

Criar uma aplicação para Android

O primeiro passo para a criação de uma aplicação é a criação de um novo projecto Android no eclipse através do wizard. Depois de se indicar o nome do projecto, nome da aplicação e o package da mesma, é criado um projecto com uma aplicação "Hello World" que serve de base para a criação da aplicação que se pretende desenvolver. Nesta base criada pelo wizard, está presente todo o esqueleto de directorias e ficheiros que são necessários para a aplicação.

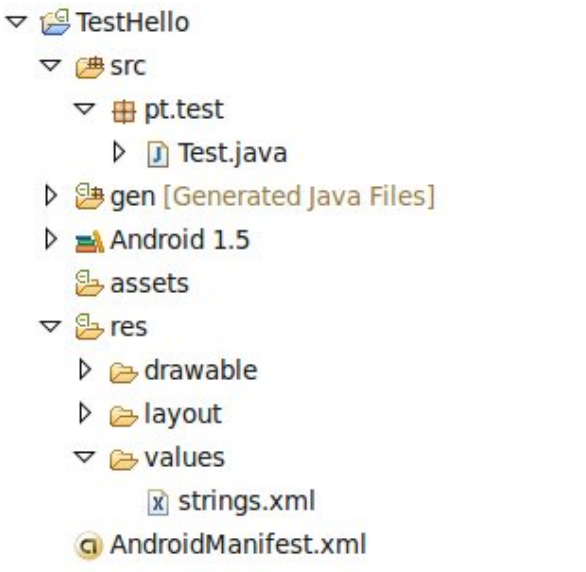

Fig.1- Directorias e ficheiros de uma aplicação

Dentro da directoria **src** estão presentes os ficheiros com o código Java da aplicação. Estes ficheiros encontram-se organizados e distribuídos segundo a estrutura dos packages de Java que a aplicação possui.

Já na directoria **res** podem ser encontradas três novas directorias. A directoria **drawable** é onde são colocados todos os recursos que estejam relacionados com imagem. É nesta directoria que devem ser adicionadas todas as imagens que se pretende usar na aplicação, como por exemplo: fundos, ícones, etc..

Na directoria **layout** são guardados todos os layouts que a aplicação pode utilizar. Um layout é um simples ficheiro xml que contém a estrutura e os elementos que vão ser mostrados pela aplicação. Nestes ficheiros é possível definir os elementos disponibilizados pela plataforma Android (como TextViews, EditText, etc) que se pretende incorporar na aplicação assim como as propriedades de cada um desses elementos com vista a criar o design que se pretende obter na aplicação.

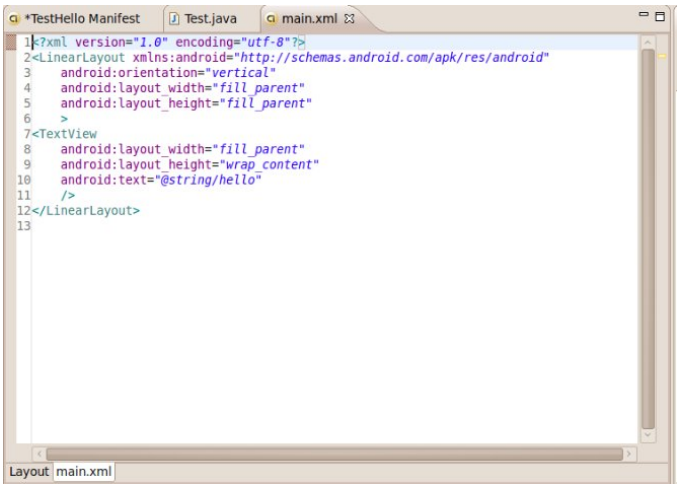

Fig. 2 – Exemplo do conteúdo do layout da aplicação Hello World

Dentro da directoria **values** devem ser guardados todos os valores constantes que a aplicação vai utilizar. Um exemplo desses valores, são todas as strings usadas na aplicação. Assim, é possível guardar todas as strings usadas na aplicação dentro do ficheiro **strings.xml** e usa-las na aplicação simplesmente usando uma referência. Com esta organização torna-se fácil para o programador fazer uma internacionalização/localização de uma aplicação. Deste modo também é possível que o sistema operativo defina a língua a utilizar na aplicação que mais se adequa ao idioma do utilizador. Esta funcionalidade é conseguida através da indicação de especificações nas directorias. Basicamente se o programador tiver a aplicação disponível em duas línguas, como por exemplo Inglês e Português, apenas necessita de criar duas pastas: **values-en** e **values-pt**, e quando a aplicação estiver em execução num dispositivo configurado com língua Portuguesa, a aplicação irá ser executada usando as strings contidas no ficheiro **strings.xml** dentro da directoria **values-pt**.

O mesmo efeito pode ser conseguido com os layouts e imagens a usar na aplicação. Neste caso não tem tanta relevância a língua utilizada no dispositivo, mas sim a orientação em que o dispositivo se encontra. Como normalmente os ecrãs dos dispositivos não são quadrados, isto obriga a que sejam criados layouts diferentes e a utilização de imagens diferentes para a execução da aplicação em landscape ou portrait. Para isso apenas precisamos separar os diferentes layouts e imagens específicas para cada uma das orientações nas directorias **layout**, **layout-land**, **drawable-port** e **drawable-land**.

A última directoria que necessita de ser referenciada é a directoria **gen**, onde está disponível o ficheiro **R.java** que contém uma referência para todos os recursos existentes na aplicação assim como todos os recursos criados. Este ficheiro é criado automaticamente cada vez que a aplicação é compilada. De salientar também que é este ficheiro que permite agilizar o processo de referência, integração e utilização de todos os recursos existentes nas directorias mencionadas previamente, na aplicação.

O único ficheiro comum destas directorias é um ficheiro xml com o nome **AndroidManifest**. Neste ficheiro é onde são indicadas todas as características da aplicação. Desde a versão, nome e icon da aplicação, até às permissões que a aplicação precisa para utilizar os recursos do dispositivo.

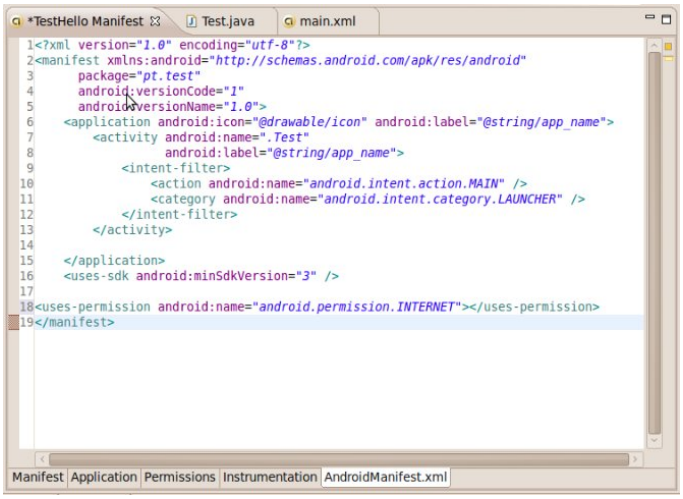

Fig. 3 – Exemplo do conteúdo do ficheiro AndroidManifest.xml

Onde tudo começa!

Quando uma aplicação entra em execução, o primeiro método a ser chamado é o método onCreate disponibilizado pela classe Activity. Uma aplicação tem de ser constituida pelo menos por uma activity. No fundo uma activity pode ser associada visualmente, ao ecrã que se visualiza quando essa activity está em execução. A criação de mais que uma activity permite que uma aplicação seja dividida em módulos(normalmente estão associadas aos casos de

utilização), e sejam criadas várias activities que compõem a aplicação e que podem inclusive permitir a reutilização destas por aplicações externas.

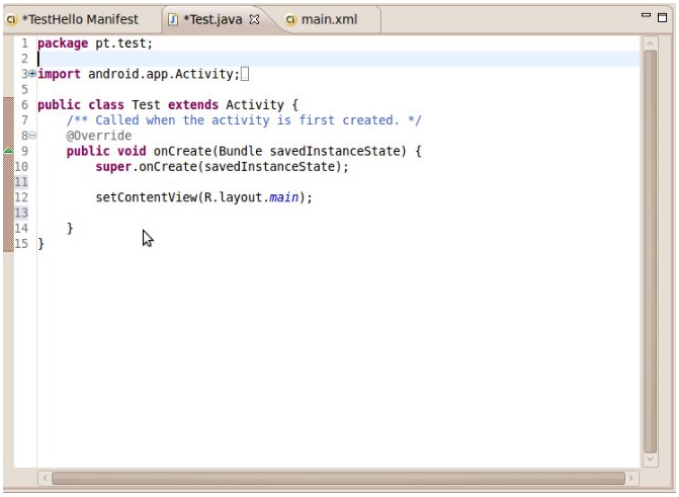

Fig. 4 – Código da classe que implementa a Activity

Para criar uma activity é necessário criar uma classe que estende da classe Activity dentro do package criado pelo wizard e com o nome que pretendemos. Nesta classe é extremamente importante a criação e indicação de dois métodos:

• A implementação do método onCreate que deve fazer Override ao já implementado pela classe Activity.

• E a chamada do método setContentView, dentro do método onCreate, indicando qual o layout definido na pasta layout que a activity vai mostrar no ecrã.

A partir daqui, depois de ser chamado o método setContentView, o programador deve tomar conta da aplicação e implementar a lógica de negócio / ou requisitos que pretender. Existem muitos outros métodos que a classe activity implementa que permitem aceder a um conjunto vasto de funcionalidades da plataforma Android e que podem ser utilizados pelo programador. Para o programador testar a aplicação, o SDK disponibiliza um emulador que permite ao programador verificar e testar a aplicação.

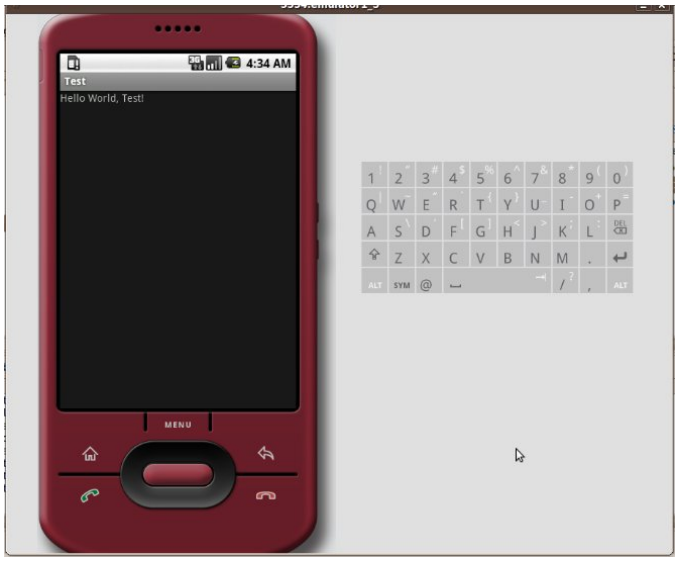

Fig. 5– Emulador a correr a aplicação

Procurar Ajuda

Numa fase inicial os programadores de aplicações Android têm bastantes recursos onde encontrar ajuda. A melhor referência que os programadores podem seguir quando estão a desenvolver, é a página que é disponibilizada pelo Android (http://developer.android.com/) onde podem ser encontrados guias, documentação da API e exemplos. Para além disso, pode ser encontrada ajuda no google-groups de programadores de Android (http://developer.android.com/resources/community-groups.html), no stackoverflow (http://stackoverflow.com/questions/tagged/android), no IRC (canal #android e #android-dev no host irc.freenode.net) e também no twitter (http://twitter.com/androiddev). Em portugal também existe uma comunidade portugesa de Android onde se pode encontrar ajuda e alguma documentação e bastantes e bons artigos sobre Android(http://androidpt.com/).

SOBRE O AUTOR

O Diogo estuda actualmente na Faculdade de Engenharia da Universidade do Porto, e está no seu 5º ano. Desde Junho de 2008 que tem também trabalhado como programador no Instituto de Investigação Fraunhofer Portugal, onde tem criado, para além de outras coisas, várias aplicações para o Android. É um entusiasta das tecnologias abertas e adora o Linux.

diogo.junior@portugal-a-programar.org

Diogo Júnior

LUA - Linguagem de Programação - Parte 2

No primeiro artigo desta série foram apresentadas informações sobre o uso inicial da linguagem de programação Lua. Foram então abordados pontos relacionados com o desenvolvimento da linguagem, sua aquisição e exemplos de programação sequencial. Neste artigo será enfatizada a programação com tomada de decisão.

Decisões e Condições

Na sua forma mais ampla, decisão é o acto ou efeito de decidir, de deliberar, de julgar, de tomar uma decisão.

Para uma decisão ser tomada esta necessita estar calcada sobre uma proposição, que na esfera da área da computação é tratada como sendo uma condição.

A condição é por sua natureza própria o estado ou a situação de algo. Dentro da esfera computacional pode-se dizer que condição é a relação lógica entre os elementos que formam a proposição para a tomada de uma decisão e que a resposta da tomada de uma decisão poderá ser verdadeira ou falsa.

A relação lógica entre esses elementos ocorre sob dois aspectos:

- Variável versus variável;
- Variável versus constante.

A forma para se estabelecer uma relação lógica permitida é conseguida com o uso de operadores relacionais.

Operadores Relacionais

Os operadores relacionais são ferramentas responsáveis por permitir a definição da relação entre elementos que formam uma condição, à qual será utilizada para a execução do processo de tomada de uma decisão.

A tabela seguinte apresenta os operadores relacionais que podem ser utilizados com a linguagem Lua.

Simbolo	Significado
=	igual a
<>	diferente de
>	maior que
<	menor que
>=	maior ou igual que
<=	menor ou igual que

Os operadores relacionais possuem a mesma prioridade. Assim sendo, não há necessidade de se preocupar em alterar a sua prioridade quando a utilizar numa expressão lógica.

Desvio Condicional Simples

Para se fazer uso de tomada de decisão com desvio condicional simples é necessário fazer uso de instrução:

if (condição) then [bloco] end
que possui a sintaxe:

```
if (condição) then
  bloco de acção
end
instruções executadas após
condição ser falsa ou após a execução
do bloco de acção
quando a condição for
verdadeira.
```

Se condição for verdadeira, então serão executadas todas as instruções definidas na área de bloco de acção que estiver entre if (condição) then e end. Se condição for falsa o bloco de acção não será executado transferindo a execução do programa para após end.

No sentido de exemplificar este tipo de acção, o programa seguinte efectua a leitura de dois valores inteiros, e apresenta os valores por ordem crescente. Considere então o seguinte código:

```
-- inicio do programa COND1

  io.write("Entre 1o. valor: ")
  A = io.read("*number")
  io.write("Entre 2o. valor: ")
  B = io.read("*number")
  if (A > B) then
    A, B = B, A
  end
  io.write("Os valores sao: ");
  print(A .. " e " .. B);

-- fim do programa COND1
```

Pode escrever o código do programa anterior num editor de texto da sua preferência, grave-o com o nome condicao01.lua e execute-o através da linha de comandos: lua 5.1 condicao01.lua.

Desvio Condicional Composto

Para se utilizar a tomada de decisão com desvio condicional composto é necessário utilizar a instrução:

if (condição) then [bloco1] else [bloco2] end,
que possui a sintaxe:

```
if (condição) then
  bloco de acção 1
else
  bloco de acção 2
end
instruções executadas após
condição ser verdadeira ou
condição ser falsa
```

Se a condição for verdadeira, então serão executadas todas as instruções definidas na área de bloco de acção que estiver entre if (condição) then e else, ou seja, as instruções definidas no bloco de acção 1. Caso a condição seja falsa serão executas todas as instruções que estejam definidas na área de bloco de acção que esteja contida entre else e end, ou seja, as instruções que estejam definidas no bloco de acção 2. Após a execução de um desvio condicional composto a execução do programa continua após end.

No sentido de exemplificar este tipo de acção o programa seguinte efectua a leitura de duas notas escolares, calcula a média aritmética das notas e apresenta a mensagem "Aprovado" caso a condição seja maior ou igual a 5. Se a condição for menor que 5 apresenta a mensagem "Reprovado". Junto de cada mensagem será apresentado o valor da média.

```
-- inicio do programa COND2

  io.write("Entre 1a. nota: ")
  N1 = io.read("*number")
  io.write("Entre 2a. nota: ")
  N2 = io.read("*number")
  MD = (N1 + N2) / 2
  if (MD >= 5) then
    io.write("Aprovado, ")
  else
    io.write("Reprovado, ")
  end
  print(string.format("%5.2f", MD))

-- fim do programa COND2
```

Em seguida escreva o código de programa anterior num editor de texto de sua preferência, gravando-o com o nome condicao02.lua. Posteriormente execute-o com a linha de comando lua 5.1 condicao02.lua.

Desvio Condicional Sequencial

Para se fazer uso de tomada de decisão com desvio condicional sequencial é necessário utilizar a instrução:

if (condição1) then [bloco1] elseif (condição2) then [bloco2] ... elseif (condiçãoN) then [blocoN] else [blocoN+1] end,
que possui a sintaxe:

```
if (condição1) then
  bloco de acção 1
elseif (condição2) then
  bloco de acção 2
  (...)
elseif (condiçãoN) then
  bloco de acção N
else
  bloco de acção N + 1
end
instruções executadas após
condição sequencial ser executada
```

Se condição1 for verdadeira, então serão executadas as instruções que estejam posicionadas na área de bloco de acção entre if (condição1) then e elseif (condição2) then, ou seja, as instruções definidas no bloco de acção 1. Se condição2 for falsa, então serão executadas as instruções que estejam posicionadas na área de bloco de acção entre if (condição2) then e elseif (condiçãoN) then e assim sucessivamente. Não sendo as condições verdadeiras, serão executadas as instruções existentes entre else e end definidas como bloco de acção N+1.
No sentido de exemplificar este tipo de acção o programa seguinte efectua a leitura de um valor numérico entre 1 e 12, correspondente ao nome do mês do ano de um calendário.

```
-- inicio do programa COND3

  io.write("Entre valor: ")
  N = io.read("*number")
  if (N == 1) then
    print("Janeiro")
  elseif (N == 2) then
    print("Fevereiro")
  elseif (N == 3) then
    print("Marco")
  elseif (N == 4) then
    print("Abril")
```

```
  elseif (N == 5) then
    print("Maio")
  elseif (N == 6) then
    print("Junho")
  elseif (N == 7) then
    print("Julho")
  elseif (N == 8) then
    print("Agosto")
  elseif (N == 9) then
    print("Setembro")
  elseif (N == 10) then
    print("Outubro")
  elseif (N == 11) then
    print("Novembro")
  elseif (N == 12) then
    print("Dezembro")
  else
    print("Mes invalido")
  end

-- fim do programa COND3
```

Operadores Lógicos

Existem ocasiões em que é necessário trabalhar com o relacionamento de duas ou mais condições ao mesmo tempo para a tomada de uma única decisão. Para esses casos é necessário proceder à utilização dos operadores lógicos.

Os operadores lógicos utilizados com a linguagem Lua são três: and (operador lógico de conjunção), or (operador lógico de disjunção) e not (operador lógico de negação). Em alguns casos, o uso de operadores lógicos evita a utilização de muitas condições if...then encadeados.

O nível de prioridade entre operadores lógicos obedece a seguinte ordem: or, and e not.

O operador lógico de conjunção and é utilizado quando dois relacionamentos lógicos de uma decisão necessitam ser verdadeiros para obter um resultado lógico verdadeiro; caso contrário, o resultado do valor lógico retornado será falso. A tabela-verdade para o operador lógico and, considerando o uso de duas condições, pode retornar um dos seguintes resultados lógicos:

Condição 1	Condição 2	Resultado
Falsa	Falsa	Falso
Verdadeira	Falsa	Falso
Falsa	Verdadeira	Falso
Verdadeira	Verdadeira	Verdadeiro

O operador and faz com que o resultado lógico seja verdadeiro quando todas as condições envolvidas na decisão forem também verdadeiras, gerando assim um resultado lógico verdadeiro.

No sentido de exemplificar este tipo de acção o programa seguinte efectua a leitura de um valor numérico entre 1 e 9 e apresenta a mensagem informando se o valor está no

```
-- inicio do programa COND4

  io.write("Entre numero: ")
  N = io.read("*number")
  if (N >= 1) and (N <= 9) then
    print("Valor na faixa de 1 a 9")
  else
    print("Valor fora da faixa")
  end

-- fim do programa COND4
```

intervalo 1 a 9 ou se está fora dessa faixa de valores.
O exemplo anterior mostra, a utilização do operador de conjunção and, que somente permitirá a apresentação da mensagem: Valor na faixa de 1 a 9, caso o valor fornecido para a variável N esteja entre 1 e 9. Qualquer valor fornecido fora da faixa definida apresentará a mensagem: Valor fora da faixa.

O operador lógico de disjunção or é utilizado quando pelo menos um dos relacionamentos lógicos de uma decisão necessita de ser verdadeiro para se obter um resultado lógico verdadeiro; caso contrário, o valor do resultado lógico retornado será falso. A tabela-verdade para o operador lógico or, considerando o uso de duas condições, pode retornar um dos seguintes resultados lógicos:

Condição 1	Condição 2	Resultado
Falsa	Falsa	Falso
Verdadeira	Falsa	Verdadeiro
Falsa	Verdadeira	Verdadeiro
Verdadeira	Verdadeira	Verdadeiro

O operador or faz com que o resultado lógico seja verdadeiro quando pelo menos uma das condições envolvidas na decisão for verdadeira, fornecendo um resultado lógico verdadeiro.

No sentido de exemplificar este tipo de acção o programa seguinte efectua a leitura do sexo biológico de um ser humano e apresenta uma mensagem informando se o sexo fornecido é ou não válido..

```
-- inicio do programa COND5

  io.write("Entre seu sexo: ")
  S = io.read()
  if (S == "m") or (S == "f") then
    print("Sexo valido")
  else
    print("Sexo invalido")
  end

-- fim do programa COND 5
```

O exemplo acima demonstra a utilização do operador lógico or, que somente permite a apresentação da mensagem "Sexo válido", caso o valor fornecido para a variável S seja m ou f. Qualquer outro valor fornecido apresentará a mensagem "Sexo invalido".

Condição	Resultado
Falsa	Não Falso
Verdadeira	Não Verdadeiro

O operador lógico not é utilizado quando se necessita de estabelecer a inversão do valor de um determinado resultado lógico de uma decisão. É possível obter valores: não verdadeiro e não falso. O operador lógico not inverte o resultado lógico de uma condição. A tabela-verdade para o operador lógico not, que é utilizado posteriormente a uma condição, pode retornar um dos seguintes resultados lógicos:

O operador not faz com o resultado lógico de uma determinada decisão seja invertido.

No sentido de exemplificar este tipo de acção o programa seguinte efectua a leitura de um valor numérico e apresenta o valor, informado caso este valor não seja menor que 3.

```
-- inicio do programa COND6

  io.write("Entre um numero: ")
  NUMERO = io.read("*number")
  if not (NUMERO <= 3) then
    print( NUMERO)
  end

-- fim do programa COND6
```

Conclusão

Neste artigo foi dado ênfase nas acções de processamento lógico baseado em tomada de decisão por meio de desvio condicional simples, composto e sequencial, além do uso dos operadores relacionais e lógicos.

No próximo artigo será tratado as questões relacionadas ao uso dos laços de repetição.

SOBRE O AUTOR

Natural da Cidade de São Paulo, Augusto Manzano tem experiência em ensino e desenvolvimento de programação de software desde 1986. É professor da rede federal de ensino no Brasil, no Instituto Federal de Educação, Ciência e Tecnologia. É também autor, possuindo na sua carreira várias obras publicadas na área da computação.

augusto.manzano@portugal-a-programar.org

Augusto Manzano

LUA - Linguagem de Programação - Parte 3

No segundo artigo desta série foram apresentadas informações sobre o uso do mecanismo de tomadas de decisão. Foi abordado o tema sobre a questão decisão e condição, sobre o uso dos operadores lógicos/relacionais, além da apresentação sobre desvios simples e compostos. Neste artigo será a vez do tema relacionado ao uso de laços de repetição.

Laços de Repetição

A técnica de uso laço de repetição é tema importante em qualquer das linguagens de programação existentes, pois torna-se necessário efectivar a acção contínua e controlada de um determinado trecho de código de programa.

É conhecido que os laços de repetição em algumas linguagens de programação podem efectuar basicamente três tipos de operações de repetição (há outras linguagens que possuem até mais de três formas).

Os laços de repetição podem ser efectivados de duas maneiras: forma interactiva ou forma iterativa. Laços de repetição interactivos caracterizam-se por serem controlados por condições, podendo possuir a interferência de um utilizador; já laços iterativos são controlados por definição e uso de variável de controlo.

A linguagem de programação LUA faz uso de três laços de repetição, sendo interactivos os laços construídos com os comandos while e repeat; laços iterativos são criados por meio do comando for.

Os laços interactivos são classificados em duas categorias, sendo laços do tipo pré-teste (quando a condição de controlo é definida no início do laço) e pós-teste (quando a condição de controlo é definida no fim do laço).

Laço Pré-Teste

A estrutura de laço pré-teste realiza o teste lógico no início do laço, antes de executar alguma acção verificando se é permitido executar o trecho de instruções subordinado à condição em uso.

A utilização do teste lógico no início do laço é conseguida na linguagem LUA por meio da instrução:

while (condição) do [bloco] end.

Esta estrutura pode executar um certo conjunto de instruções, enquanto a condição permanecer verdadeira. No momento em que essa condição se torna falsa, o processamento da rotina é desviado para fora do laço. Sendo a condição falsa logo no início do laço, as instruções contidas dentro do laço são ignoradas. Esta instrução deve ser escrita:

```
while (condição) do
instruções executadas
enquanto a condição for verdadeira
end
instruções executadas
após o término do laço
```

Como exemplo de uso de laço com condição do tipo pré-teste simulando um laço do tipo iterativo, considere o desenvolvimento de um programa que apresente como resultado a tabuada de um número qualquer.

```
-- inicio do programa LACO 1

  io.write("Entre um valor numerico:
")
  N = io.read("*number")
  I = 1
  while (I <= 10) do
    R = N * I
    io.write(string.format("%2d", N))
    io.write(" X ")
    io.write(string.format("%2d", I))
    io.write(" = ")
    io.write(string.format("%3d", R),
"\n")
    I = I + 1
  end

-- fim do programa LACO 1
```

Em seguida escreva o código do programa anterior num editor de texto de sua preferência, gravando-o com o nome laco01.lua e execute-o com a linha de comando lua 5.1 laco01.lua.

Imagine outra situação, em que o utilizador deseja executar o programa de tabuada várias vezes, mas não sabe, ao certo, quantas vezes deve repetir o trecho de programa. Neste caso, observe o exemplo seguinte, no qual se faz uso do

contexto de aplicação do laço interactivo:

```
-- inicio do programa LACO 2

  RESP = 1
  while (RESP == 1) do
    io.write("Entre um valor numerico:
")
    N = io.read("*number")
    I = 1
    while (I <= 10) do
      R = N * I
      io.write(string.format("%2d", N))
      io.write(" X ")
      io.write(string.format("%2d", I))
      io.write(" = ")
      io.write(string.format("%3d", R),
"\n")
      I = I + 1
    end
    io.write("Nova tabuada? [1] - Sim
ou [2] - Nao: ")
    RESP = io.read("*number")
  end

-- fim do programa LACO 2
```

Em seguida escreva o código de programa anterior num editor de texto de sua preferência, gravando-o com o nome laco02.lua e execute-o com a linha de comando lua 5.1 laco02.lua.

Laço Pós-Teste

A estrutura de laço pós-teste realiza o teste lógico no final do laço, executando o trecho de instruções até que a condição do laço se torne verdadeira. A utilização de teste lógico no final do laço é conseguida com a instrução:

repeat [bloco] until (condição).

A estrutura repeat...until é controlada por decisão, porém esse tipo de laço utiliza um conjunto de instruções pelo menos uma vez antes de verificar a validade da condição estabelecida. A instrução repeat...until deve ser escrita:

```
repeat
  instruções executadas até que
  a condição seja verdadeira
until (condição)
instruções executadas
após o término do laço
```

Como exemplo de uso deste tipo de laço, considere o desenvolvimento de um programa que também apresente como resultado a tabuada de um número qualquer.
Em seguida escreva o código do programa anterior num

```
-- inicio do programa LACO 3

  io.write("Entre um valor numerico:
")
  N = io.read("*number")
  I = 1
  repeat
    R = N * I
    io.write(string.format("%2d", N))
    io.write(" X ")
    io.write(string.format("%2d", I))
    io.write(" = ")
    io.write(string.format("%3d", R),
"\n")
    I = I + 1
  until (I > 10)

-- fim do programa LACO 3
```

editor de texto de sua preferência, gravando-o com o nome laco03.lua e execute-o com a linha de comando lua 5.1 laco03.lua.

Imagine outra situação, em que o utilizador deseja executar o programa de tabuada várias vezes, mas não sabe, ao certo, quantas vezes deve repetir o trecho de programa. Neste caso, observe o exemplo seguinte:

```
-- inicio do programa LACO 4

  repeat
    io.write("Entre um valor numerico:
")
    N = io.read("*number")
    I = 1
    repeat
      R = N * I
      io.write(string.format("%2d", N))
      io.write(" X ")
      io.write(string.format("%2d", I))
      io.write(" = ")
      io.write(string.format("%3d", R),
"\n")
      I = I + 1
    until (I > 10)
    io.write("Nova tabuada? [1] - Sim
ou [2] - Nao: ")
    RESP = io.read("*number")
  until (RESP ~= 1)

-- fim do programa LACO 4
```

Em seguida escreva o código de programa anterior num editor de texto de sua preferência, gravando-o com o nome laco04.lua e execute-o com a linha de comando lua 5.1 laco04.lua.

Laço com Variável de Controlo

Foram apresentadas duas formas de se definir e usar laços condicionais, uma usando o laço while...do...end e a outra usando o laço repeat...until. Esses dois tipos de laço permitem elaborar programas que venham a executar um trecho do programa um determinado número de vezes. No entanto, existe na linguagem de programação LUA uma terceira maneira de se representar laços. Trata-se do laço:

for (contador) do [bloco] end.

Os laços de repetição que possuírem um número finito de execuções, caracterizando-se em laços iterativos podem ser efectivados pelo uso da instrução: for...do...end, que é controlado por uma variável de controlo do tipo contador, podendo ser crescente ou decrescente, tendo como sintaxe para laço crescente:

```
for <variável> := <inicio>, <fim>,
<incremento> do
  instruções executadas
  dentro do laço
end
instruções executadas
após conclusão do laço
```

Como exemplo de uso, considere o desenvolvimento de um programa que apresente como resultado a tabuada de um número qualquer.

```
-- inicio do programa LACO 5

  io.write("Entre um valor numerico:
")
  N = io.read("*number")
  for I = 1, 10, 1 do
    R = N * I
    io.write(string.format("%2d", N))
    io.write(" X ")
    io.write(string.format("%2d", I))
    io.write(" = ")
    io.write(string.format("%3d", R),
"\n")
  end

-- fim do programa LACO 5
```

Em seguida escreva o código de programa anterior num editor de texto de sua preferência, gravando-o com o nome laco05.lua e execute-o com a linha de comando lua 5.1 laco05.lua.

Conclusão

Neste artigo foi dado atenção as acções de processamento baseadas no uso dos laços de repetição interactivos e iterativos.

No próximo artigo serão tratadas as questões relacionadas com o uso de variáveis indexadas.

SOBRE O AUTOR

Natural da Cidade de São Paulo, Augusto Manzano tem experiência em ensino e desenvolvimento de programação de software desde 1986. É professor da rede federal de ensino no Brasil, no Instituto Federal de Educação, Ciência e Tecnologia. É também autor, possuindo na sua carreira várias obras publicadas na área da computação.

augusto.manzano@portugal-a-programar.org

Augusto Manzano

GOSTAS DE PROGRAMAÇÃO?

É DIFÍCIL ENCONTRARES TUTORIAIS EM PORTUGUÊS DE PROGRAMAÇÃO?

SENTES FALTA DE LER O QUE TE INTERESSA?

REVISTA

PROGRAMAR

acede já a

www.revista-programar.info

e vira uma página na tua vida

um projecto

portugal-a-programar.org

Programação funcional em Python

O paradigma funcional é um paradigma que trata a computação como uma sequência de funções e não como uma sequência de acções que mudam o estado do programa. Neste paradigma não há dados mutáveis, tudo é constante: se x é definido como sendo 3, x nunca vai poder ser 4, 5 ou 6. As variáveis são apenas nomes para valores (vulgo dados), e não uma caixa para o que lá quisermos colocar dentro, seja uma bola azul ou amarela. Esta abordagem é o que diferencia o paradigma funcional do imperativo: no paradigma funcional "transforma-se" valores, aplicam-se em novas situações, no paradigma imperativo alteram-se estados (o conteúdo das variáveis por exemplo).

O Python é uma linguagem que, no meu ponto de vista, extremamente flexível, adaptando-se facilmente às nossas necessidades, e que suporta 3 paradigmas: o funcional, o imperativo e o orientado a objectos. É dada uma grande ênfase à programação orientada a objectos (uma vez que tudo em Python são objectos e a própria linguagem "obriga" a perceber a lógica da programação orientada a objectos), mas não dá tanta à programação funcional. Neste artigo pretendo apenas elucidar o leitor sobre alguns dos recursos disponibilizados pela linguagem para a programação no paradigma funcional e o seu funcionamento, estando a explicação/análise da programação funcional em si fora do objectivo deste artigo.

Caso esteja interessado em ler sobre a programação funcional em si, devo referir que o artigo da Wikipedia em inglês (http://en.wikipedia.org/wiki/Functional_programming) está bastante completo e é um bom ponto de partida.

Funções anónimas (lambda)

No Python, à semelhança de outras linguagens, existem funções anónimas. A vantagem das funções anónimas sobre as funções normais (as definidas com o def statement) em Python (e em todas as outras linguagens que suportam funções anónimas) é a possibilidade de definir rotinas sem as prender a um nome, são rotinas que não têm identidade, são abstractas, e podem ser usadas em qualquer lado. Para as funções descritas a seguir, vou usar funções anónimas, mas poderia usar igualmente funções normais. Se não conhece as funções anónimas em Python, os seguintes blocos de código são equivalentes (o primeiro usa funções anónimas, o segundo as funções ditas normais):

1º Bloco

```
transformar = lambda x, y: x**y

transformar(3, 4) #isto retorna 81
```

2º Bloco

```
def transformar(x, y):
    return x**y

transformar(3, 4)#isto também retorna 81
```

A vantagem do primeiro bloco de código sobre o segundo é o facto da função não estar presa àquele nome, ou seja, de a poder "destruir" quando quiser ou deixar de precisar dela, enquanto que com a definição tradicional não o consegue fazer.

Um outro exemplo:

1º Bloco

```
iguais = lambda x, y: "Sim" if x == y
else "Nao"

iguais(54, 55) #isto retorna "Nao"
```

2º Bloco

```
def iguais(x, y):
    if x == y:
        return "Sim"
    else:
        return "Nao"

iguais(54, 55)#isto também retorna "Nao"
```

Sintaxe: lambda <argumentos separados por vírgulas> : <o que a função deve retornar>

Map

O map() é uma função que recebe um ou mais objectos iteráveis, e itera os objectos aplicando-lhe uma função

definida pelo programador, retornando uma lista com os elementos modificados.

Exemplo: map(lambda x: x**2, [1, 2, 3, 4, 5])

O map neste caso vai retornar uma lista com todos os elementos da lista inicial elevados ao quadrado ([1, 4, 9, 16, 25]).

Podemos passar mais do que um objecto iterável, no entanto, a função que é passada ao map tem de receber tantos argumentos tantos os objectos iteráveis passados ao map.

Exemplo: map(lambda x, y: [x**2, str(y)], [1, 2, 3, 4, 5], [3.5, 7, 5.1, 8, 'a'])

Neste caso, o map retornaria [[1, '3.5'], [4, '7'], [9, '5.1'], [16, '8'], [25, 'a']].

Uma situação que por vezes acontece é passarmos objectos iteráveis com um número diferente de elementos. Quando o map tenta aceder ao quinto elemento de 2 objectos, e um desses objectos não o possui, é passado à função de controlo (a função que passamos ao map) um None.

Exemplo: map(lambda x, y, z: [x**2, str(y), z], [1, 2, 3, 4, 5], [3.5, 7, 5.1, 8, 'a'], [True, False])

O retorno do map neste caso seria [[1, '3.5', True], [4, '7', False], [9, '5.1', None], [16, '8', None], [25, 'a', None]].

Situações onde é possível usar o map, e as respectivas alternativas

Quando necessitamos de converter várias strings numéricas para inteiros:

```
map(lambda x: int(x), lista) #sendo lista uma lista de strings numéricas -> ['1', '3']
```

Uma alternativa seria o uso de listas por compreensão:

```
[int(x) for x in lista]
```

Ou então o uso do ciclo for:

```
novaLista = []
for x in lista:
    novaLista.append(int(x))
```

Filter

O filter() é uma função que faz exactamente o que o seu nome dá a entender: filtra um objecto, sendo-lhe passada uma função de controlo e o objecto a filtrar. Consoante o retorno da função de controlo seja True ou False, o elemento do objecto é ou não incluído no objecto a ser retornado.

Exemplo: filter(lambda x: x%2==0, [1, 2, 3, 4, 5, 6, 7])

O filter neste caso verifica se o módulo da divisão de cada um dos elementos da lista por 2 dá resto zero (ou seja, se é par ou não), e se der, retorna o elemento em questão. Neste caso, o retorno seria [2, 4, 6].

Situações onde é possível usar o filter, e as respectivas alternativas

Quando necessitamos de tirar todos os números pares de uma lista

```
filter(lambda x: x%2, lista) #sendo lista uma lista de inteiros -> [1, 2, 3]
```

Mais uma vez, uma boa alternativa seria o uso de listas por compreensão:

```
[x for x in lista if x%2]
```

Ou então um ciclo for:

```
novaLista = []
for x in lista:
    if x%2:
        novaLista.append(x)
```

Reduce

O reduce() é uma função que recebe uma função de controlo e o objecto iterável a reduzir. A função de controlo recebe dois argumentos: o que retornou na iteração anterior e o elemento actual do objecto. Opcionalmente, pode receber um terceiro argumento que vai ser passado à função de controlo na primeira iteração como o retorno da iteração anterior. No caso do terceiro argumento não ser passado, é utilizado o primeiro elemento do objecto iterável.

Exemplo: reduce(lambda x, y: x+y, [1, 2, 3, 4, 5, 6, 7, 8], 0)

Isto é equivalente a fazer 0+1+2+3+4+5+7+8 (o zero poderia ser outro valor, se tivesse passado o terceiro argumento), ou seja, a soma de todos os números até 8. O reduce neste exemplo irá retornar 36.

Situações onde é possível usar o reduce, e as respectivas alternativas

Calcular um factorial (neste caso, 9!)

```
reduce(lambda x, y: x*y, range(1, 10),1)
```

(O 1 no terceiro argumento é opcional)

Usando uma função recursiva:

```
def fact(x):
    if x > 1:
        return x*fact(x-1)
    else:
        return x

fact(9)
```

Usando um ciclo for:

```
y = 1
for x in range(1, 10):
    y *= x
```

Zip

O zip() é uma função que recebe n objectos iteráveis e gera uma lista de tuplas. Cada tupla é composta pelos elementos de cada um dos objectos daquela iteração. A lista contem x tuplas de n elementos, sendo x = min(len(objecto1), len(objecto2), ..., len(objecton)) e n o número de objectos iteráveis passados ao zip.

Exemplo: zip([1, 2], ['a', 'b', 'c'], ['A', 'B', 'C', 'D'])

Neste exemplo, o zip retorna [(1, 'a', 'A'), (2, 'b', 'B')].

Uma alternativa ao zip

Aqui fica uma função que replica o comportamento do zip que recebe uma lista com todos os objectos a iterar:

```
def zipper(x):
    returns = []
    size = min([len(y) for y in x])
    while size > 0:
        returns.append(tuple([y[size-
1] for y in x]))
        size -= 1
    return returns[::-1]
```

SOBRE O AUTOR

David Ferreira é um jovem apaixonado pelo mundo da informática. Deu os seus primeiros passos na programação aos 12 anos através da linguagem PHP, tendo hoje conhecimentos de XHTML, JavaScript, Python, redes, e outros. É um entusiasmado pelo mundo do hardware e software open-source, que o levou a explorar os sistemas GNU/Linux, com a distribuição Kurumin e depois Ubuntu.

david.ferreira@portugal-a-programar.org

David Ferreira

Queres participar na Revista PROGRAMAR? Queres integrar este projecto, escrever artigos e ajudar a tornar esta revista num marco da programação nacional?

Vai a

www.revista-programar.info

para mais informações em como participar,
ou então contacta-nos por

revistaprogramar
@portugal-a-programar.org

Precisamos do apoio de todos para tornar este projecto ainda maior...

contamos com a tua ajuda!

Equipa PROGRAMAR

Um projecto Portugal-a-Programar.org