

# Diário de bordo

Você se lembra de quando mudou a versão do Firefox pela última vez? E por que instalou aquele outro programa que às vezes parece não servir para nada? Minha memória não é muito boa, por isso uso um diário.

por José María Ruiz e Pedro Orantes



Justyna Furmaniczkyk - www.sxc.hu

“**D**iário de bordo, data estelar 2123...”

Em todos os livros sobre administração de sistemas, recomenda-se termos um pequeno “diário de bordo” onde possamos anotar as ações potencialmente perigosas que realizamos. Dessa maneira, poderemos supostamente recriar passo a passo os eventos que nos levaram a um desastre, e portanto ir desfazendo-os em ordem inversa.

A dura realidade é que nem todo mundo usa esses cadernos. É duro ter que deixar o teclado e pegar a caneta para escrever... Não estamos na era dos computadores? Não íamos abolir o papel?

Muitas pessoas usam um *blog* em sua própria máquina ou na Internet para anotar detalhes ou notícias que lhes são interessantes. Outros inclusive publicam

seus arquivos de configuração, para poder acessá-los sempre.

Mas o que fazer se quisermos as anotações só para nós? E se a máquina que estivermos acessando não possuir um servidor web com o software adequado configurado para ter um blog? E se não quisermos nos preocupar com toda essa parafernália?

Algumas aplicações como o *KPIM* já incorporam a opção de um diário pessoal, mas não funcionam de maneira remota, a não ser que tenhamos uma conexão de rede com muita largura de banda.

Que opções nos restam? Podemos voltar nossos olhos para a antiga era dos computadores, quando as interfaces funcionavam exclusivamente a partir de um console de texto. Tais interfaces

ainda são usadas em várias aplicações, pelo fato de serem mais simples de usar. É mais fácil automatizar o ato de teclar três vezes **[Tab]** do que mover o mouse; e isso funciona melhor remotamente, mesmo em conexões lentas.

Vamos desenhar e programar um diário de bordo em Python, que utilizará *ncurses* para a interface texto e o *gdbm* para armazenar as entradas por data.

## Desenho do caderno

Começaremos nosso projeto dando uma olhada nas bibliotecas em que vamos nos basear. A *ncurses* foi desenvolvida para abstrair, ocultar e simplificar a gestão de terminais de texto. Cada fabricante oferecia em seu terminal de

texto características distintas dos demais, forçadas na maioria das vezes por uma feroz competição. Isso transformava em uma tortura a simples tarefa de trocar um terminal por outro, exigindo na maioria das vezes a modificação do programa. A *curses* permitiu executar programas sem levar em conta as diferenças entre os terminais. Não só isso, mas também simplificou enormemente a gestão de interfaces de texto como veremos mais adiante.

O *gdbm* é uma base de “dados”. Coloquei entre aspas porque na verdade só nos permite armazenar dados, recuperá-los e realizar buscas, porém sem usar *SQL*, e sim chamadas a bibliotecas. O *gdbm* é uma família de bibliotecas que nos permitem armazenar dados em um arquivo e administrá-los como se fossem um dicionário ou *hash* em Python. Cada entrada é composta por uma *chave* e um *valor* associado. Se não tivermos que realizar buscas complexas, o *gdbm* será nossa melhor opção.

Basicamente, temos que montar uma interface que divida a tela em duas partes. Em uma, deverão ser mostradas as datas armazenadas, e deve ser possível recorrer a elas; a outra deverá exibir o texto relacionado à data indicada.

As ações serão:

- ▶ Navegar pelas entradas
- ▶ Criar uma entrada
- ▶ Editar uma entrada
- ▶ Apagar uma entrada
- ▶ Sair

Cada uma das ações irá corresponder a uma combinação de teclas. Começaremos criando os objetos que administram os dados e posteriormente a interface com o usuário.

## Armazenamento de dados

Devemos manter o texto associado a uma data e hora em algum lugar. Com a febre atual pelas bases de dados relacionais, raramente menciona-se a existência de outras bases de dados que não cumpram o padrão relacional nem *SQL*.

É realmente necessário um motor relacional e *SQL* para qualquer coisa que precisemos armazenar? Claro que não. Infelizmente, “quem só tem martelo pensa que tudo é prego”.

O problema está na definição de “banco de dados” – que a *gdbm* é, sem muita sofisticação. Basicamente, ela

nos permite armazenar chaves e valores associados às mesmas, assim como recuperar o valor ou apagar as chaves, simplesmente isso.

A biblioteca *gdbm* necessita de um arquivo para depositar os dados a serem armazenados. Assim, teremos que dar a ele o nome de um arquivo e indicar como queremos que você o trate. O arquivo pode ser aberto para a inclusão de novos dados ou criado novamente, mesmo que já exista um com o mesmo nome.

Uma vez aberto o arquivo, o objeto *gdbm* se comporta como um armazenador qualquer. Podemos fazer uso da sintaxe `[]`, à qual já nos acostumamos na maioria das linguagens de programação.

Como podemos observar no **exemplo 1**, o uso da biblioteca *gdbm* é realmente simples. Ela se comporta como uma lista, com todas as suas operações. O leitor deve ter se perguntado ao ver o código: “Onde está o truque? Se a *gdbm* representa uma base de dados, por que pode fazer uso da sintaxe `[]`?”

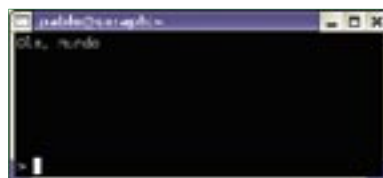
A resposta é que no Python a sintaxe `[]` é o que em inglês se chama de “syntactic sugar”. Uma forma de tradução é dizer que é uma maneira de tornar visualmente agradável a chamada a certas funções de linguagem.

Podemos incorporar `[]` a um de nossos objetos e fazer com que se comporte como uma lista? A resposta é sim, e não há nada de complicado nisso.

O Python reserva uma série de métodos ao uso especial. Entre eles estão:

```
▶ def __len__(self)
▶ def __setitem__(self, chave, valor)
▶ def __getitem__(self, chave)
▶ def __delitem__(self, chave)
```

Esses quatro métodos são mascarados posteriormente pelo Python, da maneira mostrada no **quadro 1**. Portanto, podemos mascarar as ações de um objeto de forma que ele seja usado como se fosse um dicionário. E é exatamente isso que vamos fazer com nosso objeto *Armazen* que abriga o dicionário, acrescentando novas ações. O leitor pode comprovar o código no **exemplo 2**. ▶



**Figura 1** Olá Mundo em nosso primeiro programa com *curses*.

## Exemplo 1: Uso da *gdbm*

```
01 >>> import gdbm
02 >>> dados = gdbm.open('visitantes','c') # cria o arquivo
03 >>> dados["Joao Jose"] = "virah terça-feira"
04 >>> dados["Joao Jose"]
05 'virah terça-feira'
06 >>> dados.close()
07 >>>
08 >>> dados = gdbm.open('visitantes')
09 >>> dados["Joao Jose"]
10 'virah terça-feira'
11 >>> dados.keys()
12 ['Joao Jose']
13 >>> for chave in dados.keys():
14 ...     print "[" + chave + "]" -> + dados[chave]
15 ...
16 [Joao Jose] -> virah terça-feira
17 >>> dados.close()
```

## Exemplo 2: *armazen.py*

```
01 #!/usr/bin/python
02
03 import gdbm
04 class Armazen:
05     def __init__(self,nome):
06         self.bd = gdbm.open(nome,'c')
07
08     def busca_palavra(self, palavra):
09         chaves = self.entradadas()
10         encontradas = []
11
12         for chave in chaves:
13             conteudo = self.conteudo(chave)
14             if palavra in conteudo:
15                 encontradas.push(chave)
16
17         return encontradas
18
19     def entradas(self):
20         a = self.bd.keys()
21         if not a:
22             a = []
23         return a
24
25     def fecha(self):
26         self.bd.close()
27
28     def __len__(self):
29         return len(self.entradadas())
30
31     def __setitem__(self, chave, valor):
32         self.bd[chave] = valor
33
34     def __getitem__(self, chave):
35         return self.bd[chave]
36
37     def __delitem__(self, chave):
38         del self.bd[chave]
```

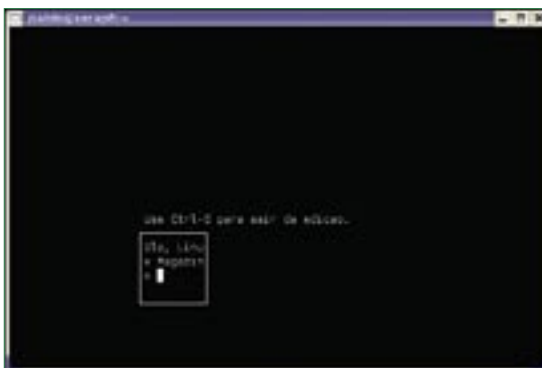


Figura 2 Um quadro de texto *curses*.

## Curses

*Curses* são algumas bibliotecas de baixo nível. As abstrações que elas criam são muito básicas: preparar o console, criar “janelas” (que nada têm a ver com as janelas gráficas), escrever nessas janelas, receber caracteres e algo mais.

Por esse motivo, são muito complicadas de manipular. Fazer coisas chamativas costuma precisar de muito código. Por isso, vamos nos concentrar em uma interface simples. Nosso programa será modular: terá um modo de “navegação” e um de “edição”, da mesma forma que o editor *Vi*.

## Desenho principal

Vamos começar a inicializar a *curses*. Infelizmente, isso também nos faz perder o controle do nosso console Python,

visto que impede o seu funcionamento. Por isso, pede-se ao leitor que execute todas as ações relacionadas a *curses* a partir de um programa Python executável (lembre-se de fazer o `chmod +x <programa>`). Podemos ver um programa que inicializa o console com *curses* no **exemplo 3**.

Depois, escrevemos um “Olá mundo” e a tela é atualizada. Po-

demos ver o resultado na **figura 1**. Se não atualizarmos a tela, a *curses* não mostrará nada. No **exemplo 3**, `stdscr` representa toda a tela. É possível criar subjanelas e realizar atualizações seletivas como poderemos comprovar no código do programa.

Uma vez realizadas as operações, prosseguimos deixando a tela em uma configuração correta, o que é realizado pelas quatro últimas chamadas de funções.

O objeto `diario` criará por sua vez um objeto `GUI`, que administra a interface, e o objeto `Armazem`, que se encarrega de administrar a base de dados.

O objeto `Armazem` é passado para o `GUI` como parâmetro, em sua criação. E a missão de `GUI` é exatamente responder aos eventos que o usuário enviar mediante um laço infinito.

## Duas janelas

Nosso programa vai dispor de duas janelas. A maior faz o papel de “quadro”, onde podemos ver as anotações feitas no momento. Podemos nos mover para cima e para baixo através dele. Para indicar qual é a data selecionada, iremos destacá-la em negrito e sublinhá-la.

A segunda janela servirá de barra de ajuda e status. Ao mudarmos o status, por exemplo, ao editar, isso se refletirá lá. É o mesmo modo de trabalho de que se orgulha o *Vim*.

As janelas devem dividir a tela de maneira que não se sobreponham. A janela de um terminal tem 80 colunas de largura e 25 linhas de altura. Deixaremos uma linha abaixo, que será usada para mostrar informações. O restante das 24 fileiras se encarregarão de mostrar as entradas armazenadas.

## Deslocamento das entradas

A janela de dados permitirá que nos movamos para cima e para baixo pelas entradas. Como vamos conseguir recriar esse movimento? A solução é capturar as teclas dos cursores “Para cima” e “Para baixo”. Ao pressionar uma delas, incrementaremos ou decrementaremos uma variável que estabelece a posição da entrada selecionada em cada momento, e voltaremos a desenhar, ou escrever, a tela de dados. Mas não faremos isso de qualquer jeito.

Queremos que o efeito seja bonito, de forma que sempre tentaremos mostrar um número fixo de entradas acima e abaixo da nossa. Como temos a posição da entrada selecionada, um simples cálculo nos permite selecionar quais entradas queremos mostrar.

As listas em Python têm uma funcionalidade que nos será muito útil. Usando a sintaxe `lista[começo:fim]`, podemos extrair os elementos entre `começo` e `fim`, formando uma nova lista. Simplesmente temos que selecionar aqueles que estejam a uma distância fixa do selecionado e usá-los como `começo` e `fim`.

Podemos ver o código que realiza essa ação no método `mostra_dados(self)` da `GUI`, no **exemplo 6**.

### Exemplo 3: “Olá mundo” com *curses*

```
01 #!/usr/bin/python
02 # -*- coding: UTF-8
03
04 import curses
05 from time import sleep
06
07 # Inicialização da tela
08 stdscr=curses.initscr()
09 curses.noecho()
10 curses.cbreak()
11 stdscr.keypad(1)
12
13 # Escrevendo algo
14 stdscr.addstr("\n01a, mundo\n\n",0)
15 stdscr.refresh()
16
17 # Esperando 4 segundos antes de...
18 sleep(4)
19
20 # Limpar a tela
21 stdscr.keypad(1)
22 curses.echo()
23 curses.nocbreak()
24 curses.endwin()
```

## Textbox

O Python dispõe de uma ferramenta muito útil para a edição de textos com `curses`. Infelizmente, apesar de seu poder, ela possui alguns inconvenientes que demonstraremos mais adiante.

Essa ferramenta é o objeto `Textbox`, que se encontra na biblioteca `curses.textpad`. `Textbox` nos permite editar um texto dentro de uma janela, e utilizar muitas das combinações de teclas que o editor de textos *Emacs* suporta. Assim, um **[Ctrl]+[E]** os leva ao final da linha que estivermos editando, e **[Ctrl]+[D]** apaga o caractere sobre o qual nos encontramos.

Vamos utilizar um `Textbox` para percorrer o texto que o usuário quer introduzir. Infelizmente, ele possui uma limitação devido ao seu desenho. Se, ao editarmos o texto, pressionarmos repetidas vezes a seta para a esquerda, deslocando-nos até encontrar a borda da janela, o programa irá falhar.

O suporte a `curses` no Python baseia-se nas bibliotecas originais escritas em C e, como já dissemos, são de muito baixo nível. A implementação do `Textbox` é realmente básica e não controla todos os parâmetros. Ainda assim, ela será de alguma serventia.

Um exemplo de utilização do `Textbox` aparece em seu próprio código-fonte (ver o [exemplo 4](#) e a [figura 2](#)). Esse código gera um retângulo com bordas (usando a função `rectangle` do `curses.textpad`) e nos solicita que escrevamos algo no mesmo. Para terminar, devemos pressionar **[Ctrl]+[G]** e o escrito é mostrado mais abaixo. Se experimentarmos, comprovaremos que não podemos sair do retângulo a ser editado.

Na [figura 3](#) vemos como integramos o `Textbox` em nosso programa. Aumentamos o número de colunas para permitir a introdução de mensagens mais longas.

## Gerenciamento de comandos

Existem muitas maneiras de se fazer um gerenciador de comandos. A mais típica consiste em fazer uma estrutura `switch` ou uma grande quantidade de `ifs` agrupados, cada um dos quais responde a uma tecla ou combinação diferente. O código gerado chega a ficar ilegível quando o número de comandos ultrapassa dez.

Há uma maneira muito mais elegante de combater esse problema, mas não é tão fácil fazer uso dela em todas as linguagens. Felizmente, o Python nos permite uma implementação muito simples. A idéia é a seguinte: cada comando está associado a uma série de ações a serem realizadas. Vincularemos as ações de cada comando a um método respectivo do nosso objeto “GUI”. Até aqui, tudo é bastante normal. Agora vem a magia.

O Python nos permite invocar métodos de objetos usando seu nome. Se declarmos o objeto `Pessoa`:

```
>>> class Pessoa:
...     def fala(self):
...         print "01a, mundo"
...
>>>
```

Poderemos invocar o método `fala` usando uma cadeia com seu nome através do método `getattr()`, que precisa da instância do objeto e do método a ser invocado. Ele devolve, por assim dizer, uma referência ao método em questão, que funciona da mesma maneira. Como dizem por aí, “uma imagem vale mais que mil palavras”.



Figura 3 Inserção de uma nova entrada no diário de bordo.

## Exemplo 4: Uso de Textbox

```
01 #!/usr/bin/python
02 # -*- coding: UTF-8 -*-
03
04 import curses
05 import curses.textpad as textpad
06
07 # Inicialização da tela
08 stdscr=curses.initscr()
09 curses.noecho()
10 curses.cbreak()
11 stdscr.keypad(1)
12
13 ncols, nlinhas = 9, 4
14 uly, ulx = 15, 20
15 stdscr.addstr(uly-2, ulx, 'Use Ctrl-G para sair da edicao.')
16
17 win = curses.newwin(nlinhas, ncols, uly, ulx)
18 textpad.rectangle(stdscr, uly-1, ulx-1, uly + nlinhas,
19                 ulx + ncols)
20 textpad.Textbox(win).edit()
21
22 # Limpar a tela
23 stdscr.keypad(1)
24 curses.echo()
25 curses.nocbreak()
26 curses.endwin()
```

## Exemplo 5: Método executa\_comando(self,ch)

```
01 # Espera-se que ANTES voce tenha executado
02 # import curses
03
04 def executa_comando(self,ch):
05     "Processa as teclas recebidas"
06     if curses.ascii.isprint(ch):
07         for comando in self.comandos:
08             if comando[0] == chr(ch):
09                 (getattr(self,comando[1]))()
10                 break
11     else:
12         if ch in (curses.ascii.DLE, curses.KEY_UP):
13             self.incr_pos_datas()
14             self.redesenha()
15         elif ch in (curses.ascii.S0, curses.KEY_DOWN):
16             self.decr_pos_datas()
17             self.redesenha()
18     self.atualiza()
19     return 1
```

```
>>> fulano = Pessoa()
>>> fulano.fala()
Ola, mundo
>>> (getattr(fulano, "fala"))()
Ola, mundo
>>>
```

Vamos criar uma lista de listas, cada uma das quais contendo dois elementos. O primeiro será um caractere, e o segundo, o nome do método a ser invocado. Dessa maneira, nosso gerenciador de comandos se reduz a um código que recebe um caractere, compara-o ao primeiro elemento de cada entrada em sua lista de comandos e, se encontrar uma coincidência, executa o comando associado (exemplo 5).

Como podemos ver no código, comprova-se se o caractere recebido é “imprimível” e, posteriormente, faz-se a busca na lista de comandos. Em caso de coincidência, é feita a execução usando-se `self` como instância. Dessa maneira, é possível manipular o funcionamento a respeito de a quais caracteres o programa responde, sem termos que modificar seu código-fonte. Isso nos dá muita flexibilidade e é menos propenso a erros.

## Uso do programa

O uso do programa foi simplificado ao máximo possível. Seu aspecto é apresentado na figura 4. Ao pressionarmos **[N]**, cria-se uma nova entrada com data e hora. Caso já exista uma entrada com essas mesmas data e hora, nada é feito. Com **[N]**, elimina-se uma entrada, e com **[E]** podemos editá-la. Quando se está introduzindo um texto, ele é armazenado pressionado-se **[Ctrl]+[G]**. Para sair, deve-se pressionar **[Q]**, e com as setas “Para cima” e “Para baixo”, nos deslocamos pelo programa.

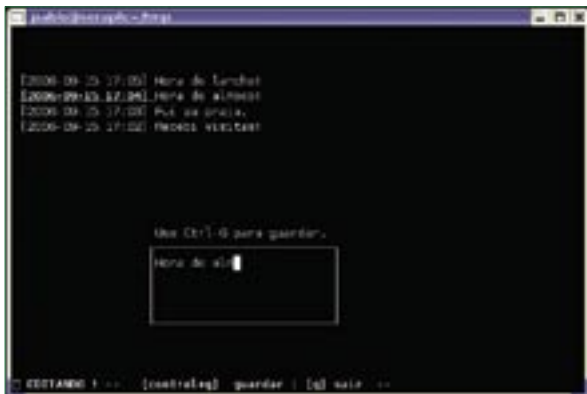


Figura 4 Visão do diário de bordo com várias entradas.

## Exemplo 6: diário.py

```
001 #!/usr/bin/python
002 # -*- coding: UTF-8 -*-
003
004 import curses
005 import curses.ascii
006 import curses.textpad
007 import time
008 import os.path
009 import string
010 import sys
011 import armazen
012
013 class GUI:
014     """Interface com o usuario."""
015
016     def __init__(self,dados):
017
018         self.registra_comandos()
019
020         self.dados = dados
021         self.pos_datas = len(self.dados) -
022     1
023
024         self.gera_janelas()
025
026         self.banner("-- [n] nova |
027     [e] editar | [s] sair --")
028
029         self.desenha_datas()
030
031         self.atualiza_tela()
032
033     def gera_janelas(self):
034         """Gera as janelas iniciais"""
035         self.scr_datas = stdscr.subwin(
036     23, 80, 0, 0)
037         self.scr_info = stdscr.subwin(
038     1,80,23,0)
039
040     def registra_comandos(self):
041         """Armazena a letra e o comando
042     associado"""
043         self.comandos = [['a','apagar'],
044     ['e','editar'],
045     ['n',
046     'nova_entrada'],
047     ['s','sair'],
048     ['t','estado']]
049
050     def executa_comando(self, ch):
051         """Processa as teclas recebidas"""
052         if curses.ascii.isprint(ch):
053             for comando in self.comandos:
054                 if comando[0] == chr(ch):
055                     (getattr(self,
056     comando[1]))()
057                     break
058             else:
059                 if ch in (curses.ascii.DLE,
060     curses.KEY_UP):
061                     self.incr_pos_datas()
062                     self.redesenha()
063                 elif ch in (curses.ascii.SO,
064     curses.KEY_DOWN):
065                     self.decr_pos_datas()
066                     self.redesenha()
067
068                 self.atualiza_tela()
069                 return 1
070
071     def data(self):
072         return time.strftime("%Y-%m-%d
073     %H:%M")
074
075     def long_data(self):
076         caixa_de_texto = 2 # o | esquerdo
077     e o | direito
078
079         return len(self.data()) +
080     caixa_de_texto
081
082     def long_linha_de_texto(self):
083         return (80 - self.
084     long_data())
085
086     def banner(self, texto):
087         """Mostra o texto na regioao do
088     banner"""
089         self.scr_info.clear()
090         self.scr_info.addstr(texto,
091     curses.A_BOLD)
092         self.scr_info.refresh()
093
094     def desenha_datas(self):
095         """Gera a listagem das datas
096     na esquerda"""
097         self.scr_datas.clear()
098         pos_x = 3
099
100         # 8 elementos acima e abaixo
101         min = self.pos_datas - 8
102         max = self.pos_datas + 8
103
104         if max > len(self.dados):
105             max = len(self.dados)
106             min = min + (max -
107     len(self.dados))
108         if min < 0:
109             max = max + (-min)
110             min = 0
111
112         if len(self.dados) > 0:
113             # Marcamos em negrito a
114     data sobre a qual estah o cursor
115             # Para isso, iteramos,
116     escrevendo as datas. Quando as
117             # encontramos, passamos
118     a ela o atributo de negrito.
119             datas = self.listagem_
120     datas()
121             for data in datas[min:
122     max]:
123                 data_temp =
124     "["+data+" "
125                 if datas[self.pos_
126     datas] == data:
127                     # Encontramos
128                     # nossa data!!!
129                     self.scr_datas.
130     addstr(pos_x,1,data_temp , curses.
131     A_BOLD | curses.A_UNDERLINE)
132                 else:
133                     self.scr_datas.
134     addstr(pos_x,1,data_temp, 0)
135                 self.scr_datas.
136     addstr(pos_x,len(data_temp),self.
137     dados[data],0)
138                 pos_x = pos_x + 1
139             self.atualiza_tela()
140
141     def editar(self):
142         """Mostra um quadro para
143     introduzir o texto e armazena-o"""
144         if len(self.dados) == 0:
145             return
146         else:
147             self.banner(" EDITANDO !
148     -- [control+g] guardar | [s]
```

```

↳ sair --")
129     datas = self.listagem_
↳ datas()
130     texto = self.
↳ dados[datas[self.pos_datas]]
131     self.atualiza_tela()
132
133     # Capturamos o novo
↳ texto.
134
135     ncols, nlinhas = 25, 4
136     uly, ulx = 15, 20
137     stdscr.addstr(uly-2,
↳ ulx, "Use Ctrl-G para guardar.")
138     win = curses.
↳ newwin(nlinhas, ncols, uly, ulx)
139     curses.textpad.
↳ rectangle(stdscr, uly-1, ulx-1, uly
↳ + nlinhas, ulx + ncols)
140     stdscr.refresh()
141     novo_texto= curses.
↳ textpad.Textbox(win).edit()
142     stdscr.refresh()
143
144     novo_texto = novo_texto.
↳ replace('\n','')
145     # Guardamos o novo texto
146     self.dados[datas[self.
↳ pos_datas]] = novo_texto
147
148     self.banner("--- [n] nova
↳ | [e] editar | [s] sair --")
149
150     def apagar(self):
151         "Elimina a entrada
↳ selecionada"
152         if len(self.dados) > 0:
153             datas = self.listagem_
↳ datas()
154             del self.dados[datas
↳ [self.pos_datas]]
155             self.atualiza_pos_datas()
156             self.redesenha()
157             self.atualiza_tela()
158
159     def redesenha(self):
160         "Redesenha a tela"
161         self.desenha_dados()
162         self.atualiza_tela()
163
164     def atualiza_tela(self):
165         self.scr_datos.refresh()
166
167     def listagem_dados(self):
168         "Retorna a listagem das
↳ datas ordenada"
169         datas = self.dados.entradas()
170         datas.sort(reverse=1)
171         return datas
172
173     def decr_pos_datas(self):
174         "Move a data selecionada
↳ para cima"
175         if self.pos_datas >= (len
↳ (self.dados) - 1):
176             self.pos_datas = len
↳ (self.dados) - 1
177         else:
178             self.pos_datas += 1
179
180     def incr_pos_datas(self):
181         "Move a data selecionada
↳ para baixo"
182         if self.pos_datas <= 0:
183             self.pos_datas = 0
184         else:
185             self.pos_datas -= 1
186
187     def atualiza_pos_datas
↳ (self,data=""):
188         "Realiza as trocas apropri
↳ adas quando se elimina uma data"
189         if data == "":
190             self.decr_pos_datas()
191         else:
192             datas = self.listagem_dados()
193             cont = 0
194             resultado = 0
195             for f in datas:
196                 if f == data:
197                     resultado = cont
198                     break
199                 cont += 1
200
201             self.pos_datas = resultado
202
203     self.redesenha()
204     self.atualiza_tela()
205
206     def nova_entrada(self):
207         "Introduz uma nova data"
208         datas = self.listagem_dados()
209         data = self.data()
210         if not data in datas:
211             self.dados[data] = "Texto
↳ vazio"
212             self.atualiza_pos_datas(data)
213             self.redesenha()
214             self.atualiza_tela()
215             self.editar()
216
217     def sair(self):
218         fecha_curses(stdscr)
219         sys.exit(0)
220
221     def estado(self):
222         "Mostra informacoes gerais"
223         cadeia = ""
224         datas = self.listagem_dados()
225         cont = 0
226         for data in datas:
227             cadeia += "["+str(cont)+"]"
↳ "+data+"]"
228             cont += 1
229
230         cadeia = "[P:"+str(self.pos_datas)
↳ "+"]"+"--[L:"+str(len(self.dados))+"]" +
↳ cadeia
231         self.banner(cadeia)
232
233 class Diário:
234     def __init__(self):
235         self.dados = armazen.Armazem
↳ ('diario')
236         self.gui = GUI(self.dados)
237         self.laco_infinito()
238
239     def laco_infinito(self):
240         while 1:
241             c = stdscr.getch()
242             n = self.gui.executa_comando(c)
243             if not n:
244                 break
245
246     def inicia_curses(stdscr):
247         curses.noecho()
248         curses.cbreak()
249         stdscr.keypad(1)
250
251     def fecha_curses(stdscr):
252         stdscr.keypad(0)
253         curses.echo()
254         curses.nocbreak()
255         curses.endwin()
256
257 if __name__ == '__main__':
258     stdscr=curses.initscr()
259     inicia_curses(stdscr)
260     diario = Diário()
261     fecha_curses(stdscr)

```

## Quadro 1: Alguns métodos especiais do Python

- ▶ `__len__(self)` devolve o comprimento de nosso objeto. É invocado quando se executa `len(meu_objeto)`
- ▶ `__setitem__(self, chave, valor)` corresponde a conferir um valor em um dicionário: `meu_objeto[chave]=valor`
- ▶ `__getitem__(self, chave)` equivale a `meu_objeto[chave]` e devolve a informação armazenada em chave
- ▶ `__delitem__(self, chave)` equivale a eliminar do dicionário um par chave-valor: `meu_objeto.pop(chave)`

O arquivo de armazenamento recebe o nome `diario.db`. Se ele não existir, deverá ser criado, e se existir, usa-se esse.

## Conclusão

Mesmo que o uso de `curses` seja considerado difícil, o Python traz módulos que nos possibilitam manipulá-la com certa facilidade. Uma vez tendo compreendido o programa, sabemos que qualquer um que instale o Python poderá fazer uso de `curses`.

Sempre é possível utilizar uma série de objetos que levem a cabo tarefas de nível mais alto. Existem bibliotecas que nos proporcionam barras de menus e *widgets* mais avançados. Mas é sempre bom estar o mais próximo possível do padrão.

Na próxima vez que você tiver que fazer uma interface em modo texto, pode ser uma boa idéia dar uma oportunidade à `curses`. ■

## Mais Informações

- [1] Programa final deste artigo: <http://www.linuxmagazine.com.br/issue/24/diario.py>

## Os autores

**José María Ruiz** está atualmente fazendo seu Projeto de Conclusão de Curso em Engenharia Técnica em Informática de Sistemas, enquanto estuda Física. Há oito anos usa e desenvolve software livre e, há dois, está se especializando em FreeBSD.

**José Pedro Orantes Casado** cursa o 3º ano de Engenharia Técnica em Informática de Sistemas e, há muitos anos, usa Linux como ferramenta de trabalho e sistema de escritório.