

G T E M R U D U

L. Nobre G.

16 de Dezembro de 2007

Resumo

Este documento descreve um sistema Gerador de Testes de Escolha Múltipla, Resposta Única e Desconto Uniforme (GTEMRUDU) baseado em *templates* de perguntas de escolha múltipla escritas em \LaTeX e contendo código `python2.4` ou superior. Este sistema tem o objectivo triplo de facilitar a geração de enunciados de testes de escolha múltipla, facilitar a geração das respectivas chaves de resposta e dificultar a fraude por cópia entre alunos submetidos ao teste. O sistema foi testado durante o segundo semestre de 2006/2007 e durante o primeiro semestre de 2007/2008, respectivamente nas disciplinas de Física I e Análise Matemática I-B da Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, tendo-se concluído que o sistema requer um extremo cuidado na programação dos algoritmos geradores de variabilidade dos enunciados e, também, que o sistema é eficaz nos seus objectivos.

Conteúdo

1	Introdução	1
1.1	Variáveis de configuração	1
2	Tutorial	3
2.1	Exemplo básico	3
2.2	Exemplo com desordenação de opções	4
2.3	Exemplo com super-conjunto de opções	5
2.4	Exemplo com dois enunciados diferentes e opções incorrectas iguais	5
2.5	Exemplo com resultado numérico	5
2.6	Exemplo com vectores	6
3	Particularidades \LaTeX	7
4	Bibliografia	7

1 Introdução

O sistema GTEMRUDU baseia-se na receita `python`

<http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/496702>

da autoria de Tomer Filiba. Esta receita define a classe `Templite` que, por sua vez, permite gerar funções interpretadores de *templates* que têm por argumento um dicionário `python`. O sistema GTEMRUDU usa uma versão da classe `Templite` que difere da original apenas nos delimitadores de código `python`. A estrutura de ficheiros do sistema encontra-se esquematizada na Figura 1. O que há a reter desta Figura, por agora, é que o sistema tem dois programas `python`: `PeMu.py` e `TeMu.py`. O programa `PeMu.py` permite verificar o funcionamento dos algoritmos embebidos nos *templates* usando a seguinte linha de comando:

```
python PeMu.py templatefile.txt
```

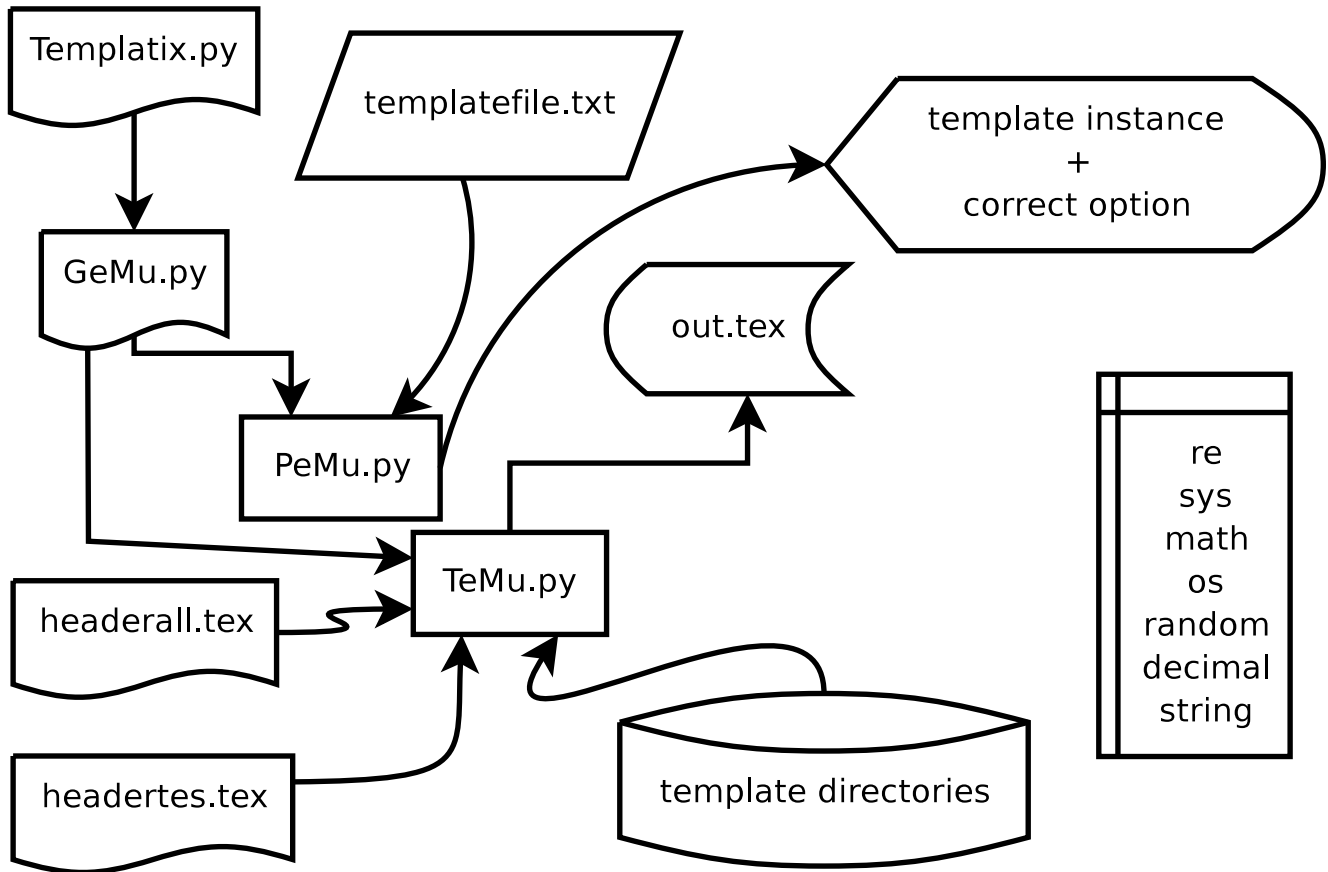


Figura 1: Esquema da estrutura de ficheiros do sistema GTEMURUDU.

O programa TeMu.py é o gerador central do sistema GTEMURUDU. A geração de uma colecção de testes de escolha múltipla, em que cada teste constitui um arranjo aleatório de um determinado conjunto de perguntas de escolha múltipla, tem lugar quando se executa o seguinte comando:

```
python TeMu.py
```

A colecção de testes de escolha múltipla é guardada num único ficheiro $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ (que, por defeito, tem o nome out.tex). A configuração deste ficheiro de saída é definida através de determinadas variáveis do programa TeMu.py (e através de determinados ficheiros que o mesmo programa lê).

1.1 Variáveis de configuração

É logo no início do programa TeMu.py que são afectadas as variáveis de configuração do ficheiro de saída:

```

1  from GeMu import *
2  random.seed( 1234567890 )
3  numberofkeypages = 5
4  pg = 12
5  questionsdirectories = ["pergs07","pergs08","pergs09"]
6  questionsnumbersperd = [15,10,5]
7  headerall = open( "headerall.tex" ).read()
8  headertes = open( "headertes.tex" ).read()
9  finaldocument = open( "out.tex" , 'w' )
10 printkeychars = 1
11 tabularheader = """\begin{tabular}[c]{|l|*{16}{c|}}
12 \hline

```

```

13  \\\hline
14  &_1&_2&_3&_4&_5&_6&_7&_8&_9&_10&_11&_12&_13&_14&_15&_
    16\\\\
15  \\\hline
16  \\\hline\n”””

```

Vamos ver este bloco de instruções `python`, instrução por instrução:

1. O ficheiro `GeMu.py` contém todas as definições comuns aos dois programas. De particular importância para os algoritmos dos *templates* é a utilização das bibliotecas-padrão `math`, `random` e `decimal` (recomendamos a consulta da respectiva documentação). A biblioteca-padrão `re` é utilizada para o caso de ser necessário trocar as marcas de separação entre unidades e décimas de pontos para vírgulas. As diversas funções definidas no ficheiro `GeMu.py` são apresentadas em §2.
2. A biblioteca-padrão `random` constitui a infraestrutura sobre a qual a variabilidade dos testes é construída, no entanto, acontece haver erros nos *templates* que só são detectados depois de ser gerada uma colecção de testes e respectivas chaves. Por esta razão, é necessário garantir que se consegue gerar uma colecção “igual” (em que a ordenação dos testes, perguntas e opções é a mesma) após a correcção dos erros detectados. Essa garantia é fornecida pela semente do gerador de números pseudo-aleatórios.
3. A variável `numberofkeypages` define o número de páginas dedicadas à lista de chaves de resposta. A lista de chaves vem sempre no fim da colecção de testes.
4. A variável `pg` define o número de chaves por página. Esta variável tem que ser definida manualmente porque o autor do sistema prescindiu de desenvolver um método automático para o fazer. O número de testes (e de chaves) da colecção é o produto `numberofkeypages*pg`.
5. Os *templates* são seleccionados de entre os ficheiros com extensão ‘`.txt`’ incluídos nas directorias da lista `questionsdirectories`.
6. Cada teste será constituído por um número de perguntas de cada uma das directorias anteriores determinado pelo número correspondente na lista `questionsnumbersperd` (este número não pode ser superior ao número de *templates* contidos na directoria respectiva). As perguntas são desordenadas em cada directoria. As directorias não são desordenadas.
7. A variável `headerall` guarda o ficheiro que contém o cabeçalho do ficheiro de saída (ver também §3).
8. A variável `headertes` guarda o ficheiro que contém o cabeçalho de cada teste.
9. A variável `finaldocument` define o nome do ficheiro de saída.
10. A variável `printkeychars`, caso seja igual a 1, faz com que a chave de soluções apresente as letras das opções correctas. Caso não seja igual a 1, a chave de soluções apresenta os números das opções correctas (situação em que o cabeçalho do ficheiro de saída deve incluir as linhas seguintes).

```

\renewcommand{\theenumi}{\alph{enumi}}
\renewcommand{\theenumii}{\arabic{enumii}}

```

11. Finalmente, a variável `tabularheader`, contém o código L^AT_EX correspondente aos cabeçalhos da tabela de respostas e da tabela de chaves de soluções (este código é inserido no lugar onde se encontrar a cadeia “`XXXcabecatabelXXX`” no ficheiro `headertes.tex`, ou seja, no ficheiro lido para a variável `headertes`).

2 Tutorial

Serão fornecidos exemplos de complexidade incremental.

2.1 Exemplo básico

Começemos com um exemplo de pergunta de escolha múltipla simples e divertido.

Os professores de Física são:

1. Burros
2. Bestas
3. Camelos
4. Animais

A única diferença obrigatória entre o código `LATEX`, necessário para escrever a pergunta acima, e o código de um *template* é a informação relativa à opção correcta. Essa informação é fornecida através da seguinte linha `python`: `certa[0]=3` (em `python` a contagem começa em zero e, portanto, a quarta opção tem o número três). Como os *templates* assumem que o código de defeito é `LATEX`, os conjuntos de linhas de código `python` têm que ser delimitados com `pyB`, no início, e com `pyE`, no fim. Desta forma, o *template* da pergunta acima é:

Os professores de Física são:

```
\begin{enumerate}
\item Burros
\item Bestas
\item Camelos
\item Animais
\end{enumerate} pyB
certa[0]=3
pyE
```

Templates deste género são pouco eficazes excepto se existirem em grande número para um teste com muito poucas perguntas. É, em geral, conveniente introduzir variabilidade nos enunciados das perguntas.

2.2 Exemplo com desordenação de opções

Vamos pegar no exemplo anterior e introduzir-lhe variabilidade na ordenação das opções de resposta.

Os professores de Física são:

```
\begin{enumerate} pyB
subst = [" Animais", " Burros", " Bestas", " Camelos"]
unsorted = shuffle( subst )
certa[0] = unsorted[0]
opts = unsorted[1]
for i in opts:
    emit( "\n\\item ", i )
pyE
\end{enumerate}
```

Esta variabilidade é conseguida através da função `shuffle` que admite uma lista como argumento e retorna não só a lista baralhada mas também a posição em que o seu primeiro elemento ficou. Por esta razão, o primeiro elemento da lista fornecida como argumento tem que corresponder à opção correcta. A função `shuffle` encontra-se definida na biblioteca `python GeMu.py` onde são concentradas todas as funções globais a que os *templates* poderão aceder. Outra função a realçar é a `emit` que se encontra definida no interior da classe `Templite`. Esta função é análoga ao `print` mas funciona no código embebido nos *templates* admitindo como argumento uma sequência de expressões. Outro aspecto a ter em consideração é que os caracteres “`\`”, omnipresentes em `LATEX`, têm um significado especial em `python` (e noutras linguagens de

programação) quando surgem em cadeias de caracteres (*strings*). O seu significado especial é aniquilado quando se duplicam e quando se precede a cadeia de caracteres pela letra “r” (que significa *raw string*).

Um dos problemas da introdução de variabilidade na ordenação das opções de resposta encontra-se na dificuldade de de corrigir erros na chave de soluções que só tenham sido detectados depois dos alunos terem sido submetidos ao teste. No entanto, o sistema facilita a correcção desses erros de duas formas: (i) desde que não se altere a semente aleatória (`random.seed` em `TeMu.py`) nem o número de *templates* disponíveis nem o número de opções de resposta em algum *template*, também não se altera a ordenação geral do teste; (ii) caso se detecte que se definiu como incorrecta a opção de resposta correcta, deve-se voltar a gerar a colecção de testes, substituindo `certa[0]=unsorted[0]` por `certa[0]=unsorted[2][X]` em que X é a posição da opção correcta.

2.3 Exemplo com super-conjunto de opções

Vamos continuar a apresentar perguntas de escolha múltipla com quatro opções de resposta mas, desta vez, vamos seleccionar três opções incorrectas de um conjunto com sete elementos.

```
Os professores de Física são :
\begin{enumerate} pyB
correc=[" Animais "]
incorr=[" Burros", " Bestas", " Camelos", " Nabos", " Lesmas", " Ratos", " Parasitas "]
subst = correc + collectio( incorr , 3 )
unsorted = shuffle( subst )
certa[0] = unsorted[0]
opts = unsorted[1]
for i in opts:
    emit( "\n\item ", i )
pyE
\end{enumerate}
```

Este exemplo usa a função `collectio` que admite uma lista e um número inteiro inferior à dimensão da lista como argumentos e retorna uma lista baralhada com esse número de elementos.

2.4 Exemplo com dois enunciados diferentes e opções incorrectas iguais

Desta vez explicitamos a utilização do método `random()` que produz um número aleatório entre zero e um.

```
Os professores de pyB
if random.random() < 0.5:
    prof="Física"
    corr="Animais"
else:
    prof="Matemática"
    corr="Pessoas"

emit( prof )
pyE
são:\begin{enumerate} pyB
correc=[corr]
incorr=[" Burros", " Bestas", " Camelos", " Nabos", " Lesmas", " Ratos", " Parasitas "]
subst = correc + collectio( incorr , 3 )
unsorted = shuffle( subst )
certa[0] = unsorted[0]
opts = unsorted[1]
```

```

for i in opts:
    emit( "\n\\item ", i )
pyE
\end{enumerate}

```

Obviamente que um mesmo tipo de enunciado pode ser gerado com diferentes tipos de *template*.

2.5 Exemplo com resultado numérico

```

Quanto é metade de pyB
val = int( 10 + 10*random.random() )
emit( val, "?" )
higmar = int( 15*random.random() )/10.0
subst = [0.5*val,0.5*val+higmar,0.5*val+higmar-0.5,0.5*val+higmar-1.0]
unsorted = shuffle( subst )
certa[0] = unsorted[0]
opts = unsorted[1]
emit( "\n\\begin{enumerate}" )
for i in opts:
    emit( "\n\\item ", presentnumber(i) )
pyE
\end{enumerate}

```

A função `presentnumber` também se encontra definida na biblioteca python [GeMu.py](#). O código deste exemplo poderia ser simplificado com a função `orderedfour` ou `integerfour`.

2.6 Exemplo com vectores

```

pyB
flag = 0
while not flag:
    flag = 1
    varx = 1.0*int( 12*random.random()-6 )
    vary = 1.0*int( 12*random.random()-6 )
    varz = 1.0*int( 12*random.random()-6 )
    if varx:
        if (not vary) and (not varz):
            flag = 0
    else:
        if (not vary) or (not varz):
            flag = 0

rstr = vecstring( varx, vary, varz )
flag = 0
while not flag:
    flag = 1
    arx = 1.0*int( 12*random.random()-6 )
    ary = 1.0*int( 12*random.random()-6 )
    arz = 1.0*int( 12*random.random()-6 )
    if arx:
        if (not ary) and (not arz):
            flag = 0
    else:

```

```

        if (not ary) or (not arz):
            flag = 0

qstr = vecstring( arx , ary , arz )
cx = varx+arx
cy = vary+ary
cz = varz+arz
sumv = vecstring( cx , cy , cz )
thecorrect = [sumv]
la = [ vecstring(cx+1,cy,cz) , vecstring(cx,cy+1,cz) , vecstring(cx,cy,cz+1)]
lb = [ vecstring(cx+2,cy,cz) , vecstring(cx,cy+2,cz) , vecstring(cx,cy,cz+2)]
lc = [ vecstring(cx-1,cy,cz) , vecstring(cx,cy-1,cz) , vecstring(cx,cy,cz-1)]
ld = [ vecstring(cx-2,cy,cz) , vecstring(cx,cy-2,cz) , vecstring(cx,cy,cz-2)]
incorrects = collectio( la+lb+lc+ld , 3 )
subst = thecorrect+incorrects
unsorted = shuffle( subst )
certa[0] = unsorted[0]
opts = unsorted[1]
emit( """A soma do vector  $\vec{r} = %s$  com o vector  $\vec{q} = %s$  é:
\\begin{enumerate}""" % (rstr , qstr) )
for i in opts:
    emit( "\n\\item ", " $\vec{r} + \vec{q} = %s$ " % i )
pyE
\\end{enumerate}

```

A função `vecstring`, mais uma vez, também se encontra definida na biblioteca python `GeMu.py`.

3 Particularidades L^AT_EX

Tal como foi explicado anteriormente, o sistema foi desenhado para produzir um único ficheiro L^AT_EX contendo uma colecção de enunciados diferentes (embora baseados no mesmo conjunto de *templates*) e as correspondentes chaves de resposta. A colecção é gerada com o comando

```
python TeMu.py
```

Por defeito, o ficheiro L^AT_EX chama-se `out.tex`. O conteúdo deste ficheiro depende das definições contidas nos *templates*, no cabeçalho de cada teste (`headertes.tex`), no cabeçalho da colecção (`headerall.tex`) e no próprio programa `TeMu.py`. É de realçar a possibilidade de incluir nos *templates* gráficos ou desenhos parametrizáveis em METAPOST, através do pacote `emp.sty`, ou em Asymptote, através do pacote `asymtote.sty`:

<http://asymptote.sourceforge.net/doc/LaTeX-usage.html#LaTeX-usage>

Possivelmente, as macros, definidas em `headerall.tex`, que mais carecem de explicação são as que fornecem listas de opções de resposta em formatos mais compactos que os formatos-padrão L^AT_EX. Um par de macros define a lista `options` e seus itens, para opções curtas num conjunto centrado, e outro par de macros define a lista `longoptions` e seus itens, para opções longas num conjunto de tipo `quote`.

4 Bibliografia

Eric Mazur “*Peer Instruction*” 0-135-65441-6
 Randall D. Knight “*Five Easy Lessons*” 0-805-38702-1
 David M. Beazley “*Python, Essential Reference*” 0-7357-1091-0
 Alex Martelli & David Ascher “*Python Cookbook*” 0-596-00167-3

Mark Lutz *“Programming Python”* 0-596-00085-5
Alex Martelli *“Python in a Nutshell”* 0-596-10046-9
Mark Lutz & David Ascher *“Learning Python”* 0-596-00281-5
Hans Petter Langtangen *“Python Scripting for Computational Science”* 3-540-29415-5
Leslie Lamport *“ \LaTeX ”* 0-201-52983-1
Frank Mittelbach & Michel Goossens *“The \LaTeX Companion”* 0-201-36299-6
Michel Goossens, Frank Mittelbach, Sebastian Rahtz, Denis Roegel & Herbert Voss
“The \LaTeX Graphics Companion” 0-321-50892-0