Fundamentos da Programação
LEIC/LETI

Aula 19

Dúvidas Projeto 1. Apresentação Projeto 2

Alberto Abad, Tagus Park, IST, 2021-22

Fundamentos da Programação
Ano letivo 2021-22
Primeiro Projeto
15 de Outubro de 2021

Buggy Data Base (BDB)

https://fenix.tecnico.ulisboa.pt/downloadFile/563568428818736/FP2122P1.20211015.pdf
(https://fenix.tecnico.ulisboa.pt/downloadFile/563568428818736/FP2122P1.20211015.pdf)

Ordem de resolução?

1. Descoberta do PIN
2. Depuração de senhas
3. Desencriptação de dados
4. Correção documentação
5. Verificação de dados

Descoberta de PIN - Q&A

- `obter_posicao` **não** precisa verificar argumentos
- `obter_digito` **não** precisa verificar argumentos
- `obter_pin` **sim** precisa verificar argumentos:
  - tuplo de entre 4 e 10 sequências de movimentos, cada movimento é uma string com 1 ou mais carateres 'C', 'B', 'E' ou 'D')
- `obter_pin` começa sempre no 5

```
In [2]: from FP2122P1_alberto import *
```

```
In [3]: obter_posicao('C', 5)
Out[3]: 2
```

```
In [4]: obter_digito('CEE', 5)
Out[4]: 1
```

```
In [132]: t = ('CEE', 'DDBBB', 'ECDBE', 'CCCCB')
          obter_pin(t)
Out[132]: (1, 9, 8, 5)
```

Depuração de senhas - Q&A

```
{'name':'john.doe', 'pass':'aabcde', 'rule':{'vals':(1,3), 'char':'a'}}
```

- `eh_utilizador` **não** precisa verificar argumentos e devolve `True` se e só se recebe:
  - um dicionário, com 3 chaves ('name', 'pass', 'rule')
  - o valor de 'name' e 'pass' são strings não vazias (qualquer carater)
  - o valor de 'rule' um dicionário com duas chaves:
    - o valor do 'vals' é um tuplo de 2 inteiros positivos (o sefundo maior ou igual que o segundo)
    - o valor de 'char' é uma única letra minúscula
- `eh_senha_valida` **não** precisa verificar argumentos (receberá sempre uma string e uma regra *bem formada*) e devolve True se e só se a senha cumpre com:
  - regras gerais: 3 vocais minúsculas e 1 carater (qualquer) seguido repetido
  - regras particulares
- `filtrar_senhas` **sim** precisa verificar argumentos:
  - se não é lista com 1 ou mais *utilizadores* levanta erro

In [65]:
```
user = {'pass': 'aa!bc.de', 'name': 'john.doe!', 'rule': {'vals': (
1, 3), 'char': 'a'}}
eh_utilizador(user)
```

Out[65]:
```
True
```

In [21]:
```
user = {'name': 'john.doe', 'pass': 'aabcde', 'rule': {'vals': 1, '
char': 'a'}}
eh_utilizador(user)
```

Out[21]:
```
False
```

In [10]:
```
eh_senha_valida('aabcde', {'vals': (1, 3), 'char': 'a'})
```

Out[10]:
```
True
```

In [9]:
```
eh_senha_valida('cdefgh', {'vals': (1, 3), 'char': 'b'})
```

Out[9]:
```
False
```

In [11]:
```
bdb = [{'name': 'john.doe', 'pass': 'aabcde', 'rule': {'vals': (1,
3), 'char': 'a'}},
       {'name': 'jane.doe', 'pass': 'cdefgh', 'rule': {'vals': (1,
3), 'char': 'b'}},
       {'name': 'jack.doe', 'pass': 'cccccc', 'rule': {'vals': (2,
9), 'char': 'c'}}]
filtrar_senhas(bdb)
```

Out[11]:
```
['jack.doe', 'jane.doe']
```

Desencriptação de dados - Q&A

```
('qgfo-qutdo-s-egoes-wzegsnfmjqz', '[abcde]', (2223,424,1316,99))
```

- `eh_entrada` **não** precisa verificar argumentos e devolve `True` se e só se recebe:
  - um tuplo de tamanho 3:
    - Primeiro elemento: string com uma ou mais palavras separadas por traços; as palavras apenas podem estar formadas por letras minúsculas (tamanho mínimo 1)
    - Segundo elemento: string com 5 letras minúsculas entre `[ ]`
    - Terceiro elemento: tuplo com dois ou mais inteiros positivos
- `obter_num_seguranca` **não** precisa verificar argumentos
- `decifrar_texto` **não** precisa verificar argumentos:
  - letras em posicão par (tendo em conta toda a string) avança `num_seguranca + 1`
  - letras em posição ímpar (tendo em conta toda a string) avança `num_seguranca - 1`
- `decifrar_bdb` **sim** precisa verificar argumentos:
  - se não é lista com 1 ou mais *entradas* levanta erro

In [78]:
```
eh_entrada(('qgfo-qutdo-s-egoes-wzegsnfmjqz', '[abcde]', (2223,424,
1316,99)))
```

Out[78]:
```
True
```

In [32]:
```

```

Out[32]:
```
False
```

In [17]:
```
decifrar_texto('qgfo-qutdo-s-egoes-wzegsnfmjqz', 325)
```

Out[17]:
```
'esta cifra e quase inquebravel'
```

In [131]:
```
bdb = [('qgfo-qutdo-s-egoes-wzegsnfmjqz', '[abcde]', (2223,424,1316
,99)),
       ('lctlgukvzwy-ji-xxwmzguggw', '[abxyz]', (2388, 367, 5999)),
       ('nyccjoj-vfrex-ncalml', '[xxxxx]', (50, 404))]
decifrar_bdb(bdb)
```

Out[131]:
```
['esta cifra e quase inquebravel',
 'fundamentos da programayq',
 'entrada muito errada']
```

Correção documentação - Q&A

- `corrigir_palavra` **não** precisa verificar argumentos
- `eh_anagrama` **não** precisa verificar argumentos:
  - ignora "case"
  - uma palavra sempre é anagrama de si propria
- `corrigir doc` **sim** precisa verificar argumentos e levanta erro se:
  - não é string, com uma ou mais palavra separadas por espaços
  - cada palavra deve estar formada apenas por 1 ou mais letras (minísculas ou maiúsculas).
- `corrigir doc` se não levanta erro:

  1. Corrige todas as palavras
  2. Começando pelo início da string corrigida testa cada palavra:
     - se já apareceu antes uma palavra anagrama diferente de si própria (ignorando *case*), apaga a palavra
     - se já apareceu antes uma palavra igual (ignorando *case*), mantem a palavra
     - se não apareceu antes nenhuma palavra anagrama, mantem a palavra

```
In [ ]: corrigir_palavra('cCdatabasacCADde')
```

```
In [82]: eh_anagrama('caso', 'SACO')
```

```
Out[82]: True
```

```
In [92]: doc = 'BuAaXOoxiIKoOkggyrFfhHXxR duJjUTtaCcmMtaAGga eEMmtxXOjUuJQqQ
         Hhqoada JlLjbaoOsuUeYy cChgGvValLCwMmWBbclLsNn LyYlMmwmMrRrongTtoOk
         yYcCK daRfFKkLlhHrtZKqQkkvVKza'
         corrigir_doc(doc)
```

```
Out[92]: 'Buggy data base has wrong data'
```

```
In [86]: doc = 'data outra daTA outra TADA'
         corrigir_doc(doc)
```

```
Out[86]: 'data outra daTA outra'
```

Verificação de dados - Q&A

```
('qgfo-qutdo-s-egoes-wzegsnfmjqz', '[abcde]', (2223,424,1316,99))
```

- `eh_entrada` **não** precisa verificar argumentos e devolve `True` se e só se recebe:
  - um tuplo de tamanho 3:
    - Primeiro elemento: string com uma ou mais palavras separadas por traços; as palavras apenas podem estar formadas por letras minúsculas (tamanho mínimo 1)
    - Segundo elemento: string com 5 letras minúsculas entre `[ ]`
    - Terceiro elemento: tuplo com dois ou mais inteiros positivos
- `validar_cifra` **não** precisa verificar argumentos:
  - recebe sempre uma cifra e um checksum
  - devolve `True` ou `False` dependendo se o checksum é o correto da cifra
- `filtrar_bdb` **sim** precisa verificar argumentos:
  - se não é lista com 1 ou mais *entradas* levanta erro

In [113]:
```
eh_entrada(('xxasxadcasfsefascfasfadfadfafafaa-b-c-d-e-f-g-h', '[xx
xxx]', (950, 300)))
```

Out[113]:
```
True
```

In [118]:
```
entrada = ('zzzdg-h-a-b-c-d-e-f-x-y-', '[zdabc]', (950, 300))

validar_cifra(entrada[0], entrada[1])
```

Out[118]:
```
True
```

In [130]:
```
bdb = [('aaaaa-bbb-zx-ysz-xy', '[xxxxz]', (950, 300)),
       ('a-b-c-d-e-f-g-h', '[abcde]', (124, 325, 7)),
       ('entrada-muito-errada', '[abcde]', (50, 404))]
filtrar_bdb(bdb)
```

Out[130]:
```
[('aaaaa-bbb-zx-ysz-xy', '[xxxxz]', (950, 300)),
 ('entrada-muito-errada', '[abcde]', (50, 404))]
```

TÉCNICO LISBOA

Fundamentos da Programação
Ano letivo 2021-22
Segundo Projeto
29 de Outubro de 2021

O Prado

https://fenix.tecnico.ulisboa.pt/downloadFile/563568428820899/FP2122P2.20211029.pdf
(https://fenix.tecnico.ulisboa.pt/downloadFile/563568428820899/FP2122P2.20211029.pdf)

TADs no projeto

TAD posicao

O TAD posicao é usado para representar uma posiçãoo `(x, y)` de um prado arbitrariamente grande, sendo `x` e `y` dois valores inteiros não negativos.

TAD animal

O TAD animal é usado para representar os animais que habitam o prado, existindo de dois tipos: predadores e presas. Os predadores são caracterizados pela espécie, idade, frequência de reprodução, fome e frequência de alimentação. As presas são apenas caracterizadas pela espécie, idade e frequência de reprodução.

TAD prado

O TAD prado é usado para representar o mapa do ecossistema e as animais que se encontram dentro.

In [ ]: