

PYTHON PARA PROCESSAMENTO DE ARQUIVOS TEXTO

GUILHERME BENCKE

PORTO ALEGRE - NOVEMBRO, 2017

OBJETIVO DO CURSO

Esse curso foi criado de forma a habilitar os times internos da empresa a ler, processar, manipular e recriar informacoes de arquivos texto sem intervencao manual e de forma que cada operacao possa ser rastreavel, reutilizavel e automatizada.

O objetivo é eliminar a edição manual de arquivos que sejam recebidos ou enviados a clientes.

OBJETIVO DO PROGRAMADOR: POR QUE BONS PROGRAMADORES SÃO PREGUICOSOS?

Preguiçosos, porque apenas programadores preguiçosos irão querer escrever os tipos de ferramentas que podem substitui-los no final. Preguiçosos, porque somente um programador preguiçoso vai evitar escrever código monótono e repetitivo e assim evitando redundância, o inimigo da manutenibilidade e flexibilidade de software. No mais, as ferramentas e processos que vem disso, disparados pela preguiça, irão aumentar a produção.

Claro, essa e apenas meia verdade. para um programador preguioso para ser um bom programador, ele (ou ela) tambem devem ser extremamente nao preguicosos quando e hora de aprender como ser preguioso, ou seja, quais ferramentas de software tornam seu trabalho mais facil, quais tecnicas evitam redundancia, e como ele pode fazer seu trabalho ter mais mantenabilidade e ser facilmente refatorado.

ARQUIVOS DE TEXTO

ARQUIVOS TEXTO SAO A INTERFACE UNIVERSAL (CITACAO)

This is the Unix philosophy: Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface.

Por mais que existam protocolos, formas eletronicas de envio e recebimento de informacao, o formato de texto em que cada linha e considerado um registro de informacao independente, ainda e a forma universao de transmissao e recebimento de dados entre computadores.

PORQUE PYTHON? - CARACTERISTICAS

"PYTHON E A FERRAMENTA IDEAL PARA PROGRAMADORES PREGUICOSOS POIS E:"

- **Uma Linguagem de Scripts:** Nao precisa de um compilador, ou da geracao de um executavel ininteligivel por um ser humano, apenas e necessario editar um arquivo texto e rodar esse arquivo texto, e esse arquivo em geral o mesmo independente de qual sistema operacional ou computador tu esta usando
- **Linguagem Simples:** Diferente de muitas linguagens que existem no mercado, o python foi pensado de forma a ser uma linguagem simples para tarefas simples e rapidas, entao diversos conceitos como Programacao Funcional, Orientacao a Objetos, estruturas complexas de dados, sao presentes no Python, mas, nao sao **Obrigatorias** de serem usadas como em outras linguagens
- **Roda em Qualquer Lugar:** O Python vem instalado em qualquer distribuicao linux e e facilmente instalado no windows baixando ele em python.org. Se utilizado exclusivamente as funcionalidades da linguagem, e garantido que qualquer programa que rode no windows, rode no linux e vice-versa.

PYTHON NAO E PARA:

"POREM O PROGRAMADOR IRA PASSAR TRABALHO SE USAR:"

- **Linguagem para processamento em modo servidor:** Um aplicativo em modo daemon, fica na maquina esperando por comandos ou dados a serem enviados, todo programa servidor numa maquina em geral roda nesse modo, todos os web servers rodam dessa forma. O Python nao e feito para ser rodado nesse modo, apesar de muitos sites famosos como o Instagram, Quora, entre outros, serem feitos com Python. Nesses sites, o python e chamado a cada requisicacao ou a cada N requisicoes, mas, jamais fica rodando no memoria por muito tempo
- **Modelagem de Dados Complexos:** Python e uma linguagem dinamicamente tipada entao e muito facil de se gerar bugs em aplicacoes com um modelo de negocio complexo como CRMs onde existe relacoes fixas entre entidades como Clientes, Contratos, Titulos, Telefones, e essas relacoes devem ser rígidas e facilmente identificaveis no codigo
- **Performance Computacional:** Python nao e uma linguagem que tem multithread nativo na maquina, e utiliza um mecanismo arcaico chamado GIL (Great Interpreter Lock) que praticamente indisponibiliza ele para aplicacoes paralelas massivas, pois indiferentemente de executar operacoes em paralelo, ele executa apenas instrucoes sequencialmente (* dando a impressao de paralelismo *). Acesso a recursos fisicos da maquina: Python nao permite acesso a recursos da maquina de forma direta, em vez disso, utiliza uma interface em C para permitir que programas em C sejam compilados com Python.

OQUE PRECISO PARA PROGRAMAR EM PYTHON...

ASSUMINDO QUE ESTAMOS TRABALHANDO NO WINDOWS E NECESSARIO:

- 1) Baixar o instalador de Python em Python.org
(Nessa versao utilizaremos a versao 2.7)
- 2) Um editor de texto, recomendo o notepad++ ou
qualquer outro para windows.
- 3) O Terminal do Windows (Command Prompt)

VIDEO DEMO, OLA MUNDO EM PYTHON

DO QUE COMPOE UM PROGRAMA EM PYTHON

Um programa de computador é um conjunto de instruções que permite que um computador receba dados e execute ações sobre os mesmos. Cada programa, independentemente de cada linguagem de programação, é composto dos seguintes componentes:

- **Variaveis:** Basicamente, um nome que guarda uma informacao que pode variar com o tempo. Uma variavel pode conter outras variaveis.
- **Funcoes:** Um conjunto de instrucoes a ser executado e que retorna ou nao um determinado valor.
- **Controle de Fluxo:** Uma instrucao de decisao a ser executado pelo programa, geralmente no formato: se isso, entao isso, senao aquilo.

OQUE E UMA VARIABEL

Um computador geralmente é composto por um processador que executa instruções de máquina, implementadas em transistores, uma memória que armazena informações e diversos dispositivos de entrada e saída de dados que gravam ou leem dados da memória do computador.

Dentro de um programa de computador é necessário representar os dados que estão armazenados na memória RAM do computador. No passado os programadores tinha uma tabela num caderno em papel em que registravam manualmente o endereço de memória de cada informação na memória, dessa forma tínhamos algo como:

```
#0000A3F -> CodCliente (4 Bytes)  
#0000A44 -> Nome (30 Bytes)
```

e assim vai

Mas, isso era muito difícil de manter e a medida que a manutenção nos programas ficou mais frequente, se tornou cada vez mais difícil manter o caderno atualizado. Dessa forma, se criou o conceito de variável, em que um nome (chamado de símbolo) será utilizado para acessar essa região de memória então podemos daí fazer assim:

```
nome = "Guilherme"
entrada = 1000
parcelas = 10
valor_parcela = 120

#E melhor, podemos acessar esses dados e
#fazer cálculos, colocando em variável:

total_acordo = entrada +
    (parcelas * valor_parcela)
```

TIPO DE VARIÁVEL

Porém, com o uso das variaveis, temos o problema que para o computador a memoria é apenas um conjunto de numeros de forma linear e nao existe distincao entre a forma ou o conteudo semantico desses dados. Para isso, se criou o conceito de tipo de dados. Cujos tipos mais comuns são:

String: Um conjunto de caracteres, como "Guilherme", em que cada caracter representa 1 byte.

Inteiro: Um numero inteiro som virgula. Ex. 100, 20, 21, 45 e assim por diante...

Ponto-Flutuante: Um numero com virgula, Ex. 100.21, 10.90, 22.34 e assim vai

Logicas: Contendo apenas os valores Sim ou Nao

CRIANDO UMA VARIÁVEL

Em Python, por ser uma linguagem de script, não é necessário que declaremos as variaveis antes de usá-las. Entao é apenas necessário atribuir um valor inicial e sair usando como nos exemplos abaixo:

```
nome = "Guilherme"
# Variavel do Tipo String
idade = 40
# Variavel do tipo inteiro
divida = 1000102.2
# Variavel do tipo ponto-flutuante
programador = True
# Variavel logica
```

MOSTRANDO O VALOR DE UMA VARIÁVEL

Para se mostrar o valor de uma variavel é necessario que se use o comando print que imprime o valor de uma ou mais variaveis alem de poder tambem combinar essa escrita com outras expressoes por exemplo:

```
nome = "Guilherme"
idade = 40
print nome

print 'idade de ', nome, ' eh ', idade, ' anos '
#Ira imprimir "idade de Guilherme eh 40 anos"

print 'Significa que ', nome, ' nasceu antes de ', \
      (datetime.today() - timedelta(days=40*365))
#Ira imprimir: Significa que Guilherme
#nasceu antes de 1977-11-3
```

CONVERTENDO DE UM TIPO PARA O OUTRO

Importante salientar que apos ter sido atribuido o valor de uma variavel, ela tem um tipo e somente podemos fazer operacoes entre variaveis de mesmo tipo, dessa forma:

```
nome + idade # Dara um erro de tipo invalido
```

Para isso é necessario converter o tipo de dados da variavel idade para um tipo compativel com nome, no caso o tipo string, entao se fizermos

```
nome + str(idade)
```

ira funcionar corretamente

As funcoes mais usadas sao:

`str()` -> Converte de um tipo qualquer para string

`int()` -> Converte de um tipo qualquer para inteiro

`float()` -> Converte de um tipo qualquer para ponto flutuante

`bool()` -> Converte de um tipo qualquer para booleano

Cuidado para converter de um tipo que possa ser convertido para outro, por exemplo:

```
int("Guilherme") # Ira dar um erro  
int("1") # Ira funcionar corretamente
```

VARIAVEIS LOGICAS

Variaveis logicas sao um dos tipos mais comuns de variaveis e são essenciais para o controle do fluxo do programa, essas variaveis podem apenas conter os valores Sim/Nao, Verdadeiro/Falso, 0 ou 1.

Sao produzidas por atribuicao simples:

```
eh_velho = True
```

Ou por Operacoes Logicas

```
eh_velho = idade > 40
```

Os operadores logicos em Python são:

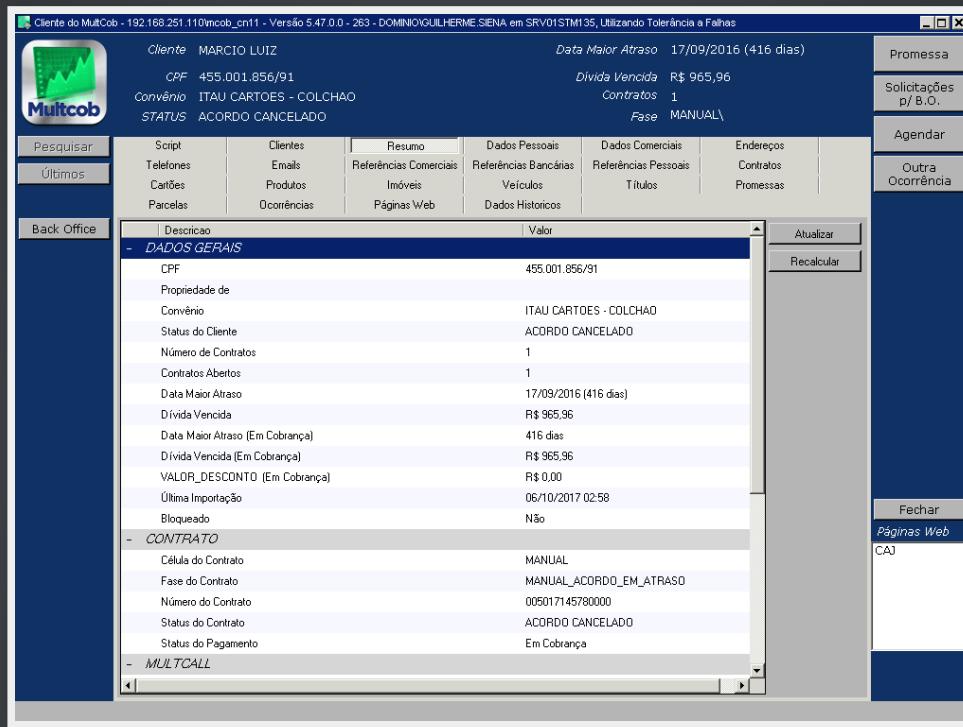
```
>, <, <=, >=, ==, !=, not , and, or
```

Podemos combinar diversas operações em uma só, da seguinte forma:

```
eh_velho = idade > 40 and \
( not nome == "Guilherme" )
```

Quer irá atribuir True a variável: eh_velho se a idade for maior que 40 e o nome não for Guilherme.

EXERCICIO #1



Abra o MultcobCli no cliente acime, crie variaveis e mostre os dados como acima.

VARIAVEIS COMPOSTAS E ESTRUTURAS DE DADOS

Como podemos ver, usamos variaveis para representar posicoes na memoria, porém declarar variaveis de forma individual é bastante penoso e ineficiente, dessa forma é necessario que possamos declarar conjuntos de variaveis e melhor variaveis que contem outras variaveis.

Python tem varias estruturas de dados, mas, nesse curso iremos analisar os mais basicos que sao:

Python tem varias estruturas de dados, mas, nesse curso iremos analisar os mais basicos que sao:

Lista:Uma lista é um conjunto de variaveis de mesmo tipo, por exemplo: Lista de CPFs, Lista de Contratos, Lista de Telefones e assim por diante, por exemplo:

```
CPFs = ['121232131',
        '9832231',
        '93468335',
        '0001234562']
```

Dicionario: Um conjunto de variaveis identificado por chave. Cada chave é unica e pode conter um tipo de dados distinto. Esse tipo de dado eh fundamental para que possamos ter formas de dados complexas como os dados de um cliente. Por exemplo:

```
Cliente = {  
    'Nome' : 'Guilherme',  
    'Idade': 40,  
    'Divida' : 1212.09,  
    'Endereco' : 'Rua XYZ, 60'  
}
```

Lista de Dicionarios: Podemos entao ter uma lista de dicionarios para poder manipular esses dados complexos:

```
lista_clientes = [
    { 'Nome' : 'Guilherme',
      'Idade': 40,
      'Divida' : 1212.09,
      'Endereco' : 'Rua XYZ, 60' },
    { 'Nome' : 'Manoel',
      'Idade': 30,
      'Divida' : 912.19,
      'Endereco' : 'Rua ABC, 60' },
    { 'Nome' : 'Felipe',
      'Idade': 21,
      'Divida' : 2122.09,
      'Endereco' : 'Rua 123, 60' }
]
```

Dicionario de Listas: Podemos tambem para dados muitos complexos ter listas como valores de um dionario:

```
cliente_crm = {  
    'Contratos' : ['010210/12',  
                  '332122/12',  
                  '898543/54'],  
    'Telefones' : ['514544333',  
                  '3299987121',  
                  '3199878121']  
}
```

OPERACOES SOBRE UMA LISTA

Para criar uma lista, apenas abrimos e fechamos colchetes:

```
lista_telefones = []
```

Para adicionar valores a lista, podemos simplesmente somar uma lista a outra lista, por exemplo

```
lista_telefones = lista_telefones + \
                  ['5133254546', '32999754433']
```

Para saber o numero de entradas numa lista, usamos a funcao len():

```
print len(lista_telefones) # ira imprimir 2
```

Todo o acesso a valores é por indice, iniciando por zero, entao podemos fazer assim:

```
print lista_telefones[1] # ira imprimir 32999754433
```

Para remover um item, podemos usar o comando `del` com o índice da lista:

```
del lista_telefones[0] # ira remover '5133254546'
```

OPERACOES AVANCADAS DE LISTA

Podemos usar slicing para criar listas de listas, por exemplo:

```
valores = ['1','2','3','4','5','6']

print valores[3:]
#Ira imprimir ['1','2','3']

print valores[:3]
#Ira imprimir ['4','5','6']

print valores[3:5]
#Ira imprimir ['4','5']

print valores[-1]
#Ira imprimir ['6'],
#indices negativos sao de tras pra frente
```

Muitas vezes recebemos uma string e precisamos criar uma lista separados por um caracter especial, como uma virgula no caso dos arquivos CSV. Para isso, usamos o comando split do tipo string:

```
"10|11|12|13|14|15".split("|")
# ira retornar: ['10','11','12','13','14','15']
```

Da mesma forma que o comando join faz o inverso:

```
"|".join(['10','11','12','13','14','15'])  
# ira retornar: "10|11|12|13|14|15"
```

OPERACOES SOBRE UM DICIONARIO

Operar um dicionario é mais simples que uma lista, é apenas acessar pelo nome da chave.

Para criar um dicionario:

```
cliente = {}
```

Para atribuir um valor:

```
cliente['nome'] = "Guilherme"  
cliente['idade'] = 40  
cliente['divida'] = 1200.50
```

Para acessar o valor:

```
print(cliente['idade'])
```

Para remover a chave eh apenas usar o operador del

```
del cliente['idade']
```

LISTA DE DICIONARIOS

Como vimos, podemos usar dicionários para criar estruturas de dados mais complexos. No caso de ter vários dados, podemos ter uma lista de dicionários, como abaixo:

```
lista_clientes = [
    { 'Nome' : 'Guilherme', 'Idade': 40,
      'Divida' : 1212.09,
      'Endereco' : 'Rua XYZ, 60' },
    { 'Nome' : 'Manoel', 'Idade': 30,
      'Divida' : 912.19,
      'Endereco' : 'Rua ABC, 60' },
    { 'Nome' : 'Felipe', 'Idade': 21,
      'Divida' : 2122.09,
      'Endereco' : 'Rua 123, 60' }
]

print lista_clientes[1]['Nome']
# Vai imprimir "Manoel"
```

DICIONARIOS DE LISTAS

Podemos usar tambem listas nos dicionarios para aumentar a complexidade dos dados.

```
cliente_crm = {
    'Contratos' :
        ['010210/12', '332122/12',
         '898543/54'],
    'Telefones' :
        ['514544333', '3299987121',
         '3199878121']
}

print ",".join(cliente_crm['Contratos'])
# vai imprimir "010210/12,332122/12,898543/54"
```

EXERCICIO #2

The image displays two side-by-side screenshots of the MultcobCLI application interface. Both screens show a header with the client information: Client MARCIO LUIZ, CPF: 455.001.856/91, Convênio: ITAU CARTOES - COLCHAO, and STATUS: ACORDO CANCELADO. The left screen shows a main search panel with tabs for Pesquisar, Últimos, Script, Clientes, Resumo, Dados Pessoais, Dados Comerciais, Endereços, Contratos, Promessas, Solidarizações p/B.O., Agendar, and Outra Ocorrência. Below this is a table for 'Back Office' with columns for Tipo De Logradouro, Logradouro, Número, Complemento, Bairro, Cidade, and UF. It lists four entries: R LUIZ GONZAGA MEROTTO, R LUIZ GONZAGA MERCATO, R PE ACACIO DUARTE, and R PRINCIPAL. A detailed address entry form is visible on the right, showing fields for Logradouro (R LUIZ GONZAGA MEROTTO), Número (20), Complemento (CS), Bairro (RETIRO), Cidade (JUIZ DE FORA), and UF (MG). The right screenshot shows a similar layout but with a more detailed view of the 'Contratos' section, displaying a table of contracts with columns like Número, Quantidade De Contratos, Data Da Entrada, Valor Da Entrada, Qtd Parcelas, Operador, Propriedade (Unidade), and Status. It lists six contracts from 20161230-1 to 20161230-6, all marked as 'Importação'. Below this is a 'Promessa' table with columns for Número, Data De Vencimento, Valor No Vencimento, and Status. It lists one entry: 20161230-1 with a value of R\$ 72,81 and status 'Pago'. At the bottom, there is a note about a base contract and an ID da Base.

Abra o MultcobCli no cliente acima, estruture um dicionario que contenha os dados acima e mostre os dados como na tela acima.

MANIPULACAO DE ARQUIVOS

Um arquivo do ponto de vista do python é apenas uma sequencia de caracteres com tantos caracteres de controle e mostraveis na tela.

Uma linha é terminada pelo caracter "\n"

Cada coluna pode ser tanto delimitada pela distancia do ultimo caracter de quebra de linha ou por um caracter especifico.

LENDΟ UM ARQUIVO

Para abrir um arquivo usamos um comando do tipo open que recebe tanto o nome do arquivo quanto o proposito para a abertura dele.

```
arq = open("cpfs.txt", "r")
```

No caso acima, abrimos o arquivo para leitura. é importante fechar o arquivo apos o uso para que possa ser liberado. Para evitar que esquecamos de fechar o arquivo, é aconselhavel que usamos o comando com um conjunto do tipo with, assim:

```
with open("cpfs.txt", "r") as arq:  
    #... Faz algo ...
```

Para facilitar a nossa vida, o python permite ler o arquivo e criar uma lista de linhas do arquivo

```
linhas_cpf = []
with open("cpfs.txt","r") as arq:
    linhas_cpf = arq.readlines()
```

O código acima irá abrir o arquivo, ler as linhas, e colocar as linhas na lista `linhas_cpf`, e finalmente fechar o arquivo.

Agora que temos todas as linhas numa lista, podemos trabalhar com esse arquivo.

INSTRUÇOES DE CONTROLE

Bem, agora que temos uma lista com as linhas do arquivo, precisamos ler elas, mas, com os métodos que possuímos apenas podemos fazer isso linha a linha ou fazer listas de listas.

O Python permite iterar sobre cada item da lista usando a instrução `for`. Dessa forma, podemos então executar comandos individualmente a cada linha. Como por exemplo:

```
for linha in linhas_cpf:  
    print linha
```

Esse trecho de código irá imprimir na tela todas as linhas da lista: linhas_cpf. Se desejarmos as últimas 3 linhas podemos usar slicing também.

```
for linha in linhas_cpf[-3:]:
    print linha
```

TOMANDO DECISÕES

Porem, digamos que queremos apenas imprimir as linhas cujo caracter 14 na linha seja o C, ou seja Cartao de Credito, Para isso, podemos usar a instrucao if com uma condicao logica para filtrar as linhas necessarias:

```
for linha in linhas_cpf:  
    if linha[14] == 'C':  
        print linha
```

Podemos criar uma lista a partir de uma interacao e uma selecao:

```
linhas_finais = []  
for linha in linhas_cpf:  
    if linha[14] == 'C':  
        linhas_finais.append(linha)
```

No caso acima a lista linhas_finais vai ter as linhas filtradas.

INTERANDO E TOMANDO DECISÕES

O Python tem uma funcionalidade muito legal para que possamos filtrar e criar listas a partir de listas ja existentes que é o "list comprehension".

Funciona assim:

```
linhas_finais = [x for x in linhas_cpf where x[14]=='C']
```

Ou seja, para cada linha x in linhas_cpf, me retorna a linha aonde o caracter 14 seja x. O comando acima é exatamente o mesmo que:

```
linhas_finais = []
for linha in linhas_cpf:
    if linha[14] == 'C':
        linhas_finais.append(linha)
```

GRAVANDO NO ARQUIVO

Para Escrever as linhas filtradas num arquivo, é muito facil, com o mesmo mecanismo que lemos, mas, com o comando de writelines e abrindo o arquivo para escrita com "w"

```
linhas_cpf = []
with open("cpfs.txt","r") as arq:
    linhas_cpf = arq.readlines()

linhas_finais = [x for x in linhas_cpf where x[14]=='C']
with open("cpfs_filtrados.txt","w") as arq:
    arq.writelines(linhas_finais)
```

No exemplo acima, podemos ver que lemos as linhas, filtramos as linhas lidas dos arquivo, e escrevemos as linhas num segundo arquivo.

MANIPULACAO DE STRINGS.

O Python tem diversos comandos para fazer a nossa vida facil para manipular strings, vou tentar apenas listar alguns:

TESTE DE CONTEUDO

```
nome = 'Guilherme Poisl Bencke'  
print('Poisl' in nome) # retorno True  
print(nome.startswith('Guilherme')) # retorna True  
print(nome.endswith('Bencke')) # retorno True
```

CONVERSÃO DE MAIUS.MINUSC.

```
nome = 'Guilherme Poisl Bencke'  
nome.upper() # Retorna "GUILHERME POISL BENCKE"  
nome.lower() # Retorno "guilherme poisl bencke"
```

REMOVENDO E TIRANDO CARACTERES DE INICIO/FIM

```
cpf="12234121"
cpf.rjust(20, '0') # Vai retornar 000000000000012234121

cpf="000012234121"
cpf.strip("0") $ Vai retornar 12234121
```

SABENDO A POSICAO DE UMA STRING EM OUTRA STRING

```
nome = 'Guilherme Poisl Bencke'  
nome.index("Poisl") # Retorna 10
```

EXERCICIO #3

Baixe os arquivos abaixo para a sua maquina, sao 3 CSVs, um com dados do cliente, um com telefones e outro de contratos, monte um arquivo unico contendo os 3 formatos e que use um formato de arquivo por tamanho fixo.

Nesse tamanho fixo teremos o contrato de maior valor do cliente e o telefone com o maior status (com ultimo acionamento como criterio de desempate).

Arquivos CSV