

Laboratório de Desenvolvimento de Algoritmos

Prof. Me. Paulo Djalma Martins
paulo.martins@ceunsp.edu.br

- Exercício 1

Escreva um programa em Python que calcule o faturamento de um representante comercial que recebe R\$ 500,00 fixos e 6% de comissão sobre as vendas do mês.

Considere que ele fechou o mês com um valor de R\$ 12.398,00 em vendas.

- Resolução - Exercício 1

```
print ("Início do Programa")  
print (" ")  
Fixo = 500.00  
Vendas = 12398.00  
Comissao = 6/100  
Fat = Fixo + Vendas * Comissao  
print ("Faturamento do mês = {0:.2f}".format (Fat))  
print (" ")  
print ("Fim do Programa")
```

"Dado = {0:2f}".format(x)
f - número real, exibindo
2 casas após a vírgula

- Exercício 2

Reescreva o programa anterior alterando-o de modo que as vendas do mês sejam lidas do teclado.

- Resolução - Exercício 2

```
print ("Início do Programa")  
print (" ")  
Fixo = 500.00  
Vendas = float (input("Digite o valor de vendas: " ))  
Comissao = 6/100  
Fat = Fixo + Vendas * Comissao  
print ("Faturamento do mês = {0:.2f}".format (Fat))  
print (" ")  
print ("Fim do Programa")
```

float
Capazes de armazenar
nrs. reais positivos ou
negativos, além do zero.

Trabalhando com dados

Um treinador necessita registrar em um arquivo os tempos de seus quatro atletas, James, Sarah, Julie e Mikey.

Inicialmente o treinador precisa de uma maneira rápida de saber os três tempos mais rápidos de cada atleta.

Vamos começar lendo os dados de cada um dos arquivos em sua própria lista.

#Importar os módulos necessários
import os

#ir para a pasta onde se encontram os arquivos desejados
os.chdir('digite aqui o caminho da pasta que estão os arquivos')

```
with open('james.txt') as jaf:  
    data = jaf.readline()  
james = data.strip().split(',')
```

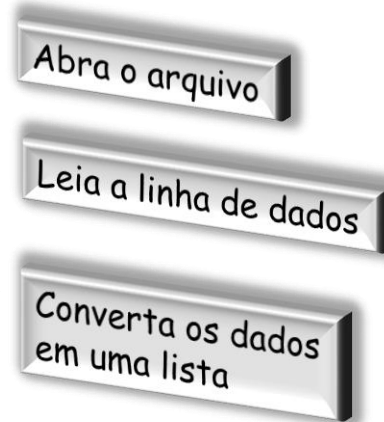
```
with open('julie.txt') as juf:  
    data = juf.readline()  
julie = data.strip().split(',')
```

```
with open('mikey.txt') as mif:  
    data = mif.readline()  
mikey = data.strip().split(',')
```

```
with open('sarah.txt') as saf:  
    data = saf.readline()
```

```
sarah = data.strip().split(',')
```

```
print(james)  
print(julie)  
print(mikey)  
print(sarah)
```



A Linha: **data.strip().split(',')** - é chamada de encadeamento de métodos.
O primeiro método **strip()** é aplicado na linha em **data**, que retira qualquer espaço em branco indesejado.
Em seguida, os resultados da retirada são processados pelo segundo método **split**, criando uma lista.

Vejam os o que acontece com seus dados quando cada uma das opções de classificação do Python é utilizada. Comece criando uma lista desordenada no shell IDLE:

```
>>> data = [6, 3, 1, 2, 4, 5]
>>> data
[6, 3, 1, 2, 4, 5]
```

Crie uma lista de dados desordenados e atribua a uma variável.

Realize uma classificação no local usando o método `sort()` que é predefinido como o padrão em todas as listas do Python:

```
>>> data.sort()
>>> data
[1, 2, 3, 4, 5, 6]
```

Execute a classificação **NO LOCAL** dos dados.

A ordem dos dados mudou.

Redefinida `data` para seu estado original desordenado, em seguida, execute uma classificação copiada usando o BIF `sorted()`:

```
>>> data = [6, 3, 1, 2, 4, 5]
>>> data
[6, 3, 1, 2, 4, 5]
>>> data2 = sorted(data)
>>> data
[6, 3, 1, 2, 4, 5]
>>> data2
[1, 2, 3, 4, 5, 6]
```

Os dados ordenados foram redefinidos.

Execute a classificação **COPIADA** dos dados.

Da mesma forma que era.

Os dados copiados estão ordenados do menor para o maior.

#Importar os módulos necessários
import os

#ir para a pasta onde se encontram os arquivos desejados
os.chdir('digite aqui o caminho da pasta que estão os arquivos')

```
with open('james.txt') as jaf:  
    data = jaf.readline()  
james = data.strip().split(',')
```

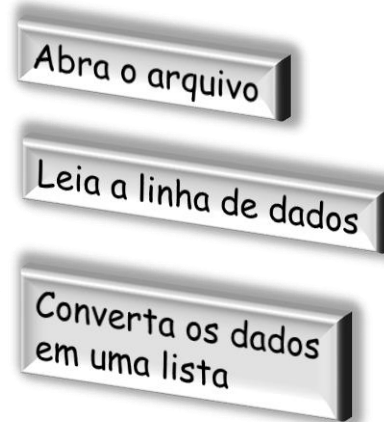
```
with open('julie.txt') as juf:  
    data = juf.readline()  
julie = data.strip().split(',')
```

```
with open('mikey.txt') as mif:  
    data = mif.readline()  
mikey = data.strip().split(',')
```

```
with open('sarah.txt') as saf:  
    data = saf.readline()
```

```
sarah = data.strip().split(',')
```

```
Print(sorted(james))  
Print(sorted(julie))  
Print(sorted(mikey))  
Print(sorted(sarah))
```



A Linha: **data.strip().split(',')** - é chamada de encadeamento de métodos.
O primeiro método **strip()** é aplicado na linha em **data**, que retira qualquer espaço em branco indesejado.
Em seguida, os resultados da retirada são processados pelo segundo método **split**, criando uma lista.

Trabalhando com dados

Parece que os valores de dados não são uniforme. O problema está nos pontos, traços e dois pontos?

Os separadores dos minutos e dos segundos estão confundindo a tecnologia de classificação do Python.

Ao gravar os tempos de seus atletas em cada um dos seus arquivos, o treinador usou as vezes, um caractere diferente para separar os minutos dos segundos.

Será necessário corrigir esse separador que foi usado.

O Python classifica as strings e quando se trata de strings, um traço vem antes de um ponto, o qual vem antes de dois pontos.

Deve-se criar uma função que obtém como entrada uma string de cada uma das listas do atleta. A função processa a string para substituir quaisquer traços ou dois pontos encontrados por um ponto e retorna a string limpa.

Se a string já tiver um ponto, não haverá necessidade de limpá-la.

Criando uma função que obtém como entrada uma string de cada uma das listas dos atletas.

```
def sanitize(time_string):
```

```
    if '-' in time_string:
```

```
        splitter = '-'
```

```
    elif ':' in time_string:
```

```
        splitter = ':'
```

```
    else:
```

```
        return(time_string)
```

```
    (mins, secs) = time_string.split(splitter)
```

```
    return(mins + '.' + secs)
```

Use o apontador "in" para verificar se a string contém um traço ou dois pontos

Não faça nada se a string não precisar ser limpa.

Divida a string para extrair as partes dos minutos e dos segundos

O programa começa aqui com a função

```
def sanitize(time_string):  
    if '-' in time_string:  
        splitter = '-'  
    elif ':' in time_string:  
        splitter = ':'  
    else:  
        return(time_string)  
    (mins, secs) = time_string.split(splitter)  
    return(mins + '.' + secs)
```

```
with open('james.txt') as jaf:  
    data = jaf.readline()  
james = data.strip().split(',')
```

```
with open('julie.txt') as juf:  
    data = juf.readline()  
julie = data.strip().split(',')
```

```
with open('mikey.txt') as mif:  
    data = mif.readline()  
mikey = data.strip().split(',')
```

```
with open('sarah.txt') as saf:  
    data = saf.readline()  
sarah = data.strip().split(',')
```

Continuação do programa

```
clean_james = []  
clean_julie = []  
clean_mikey = []  
clean_sarah = []
```

Crie quatro listas novas
vazias inicialmente



```
for each_t in james:  
    clean_james.append(sanitize(each_t))
```

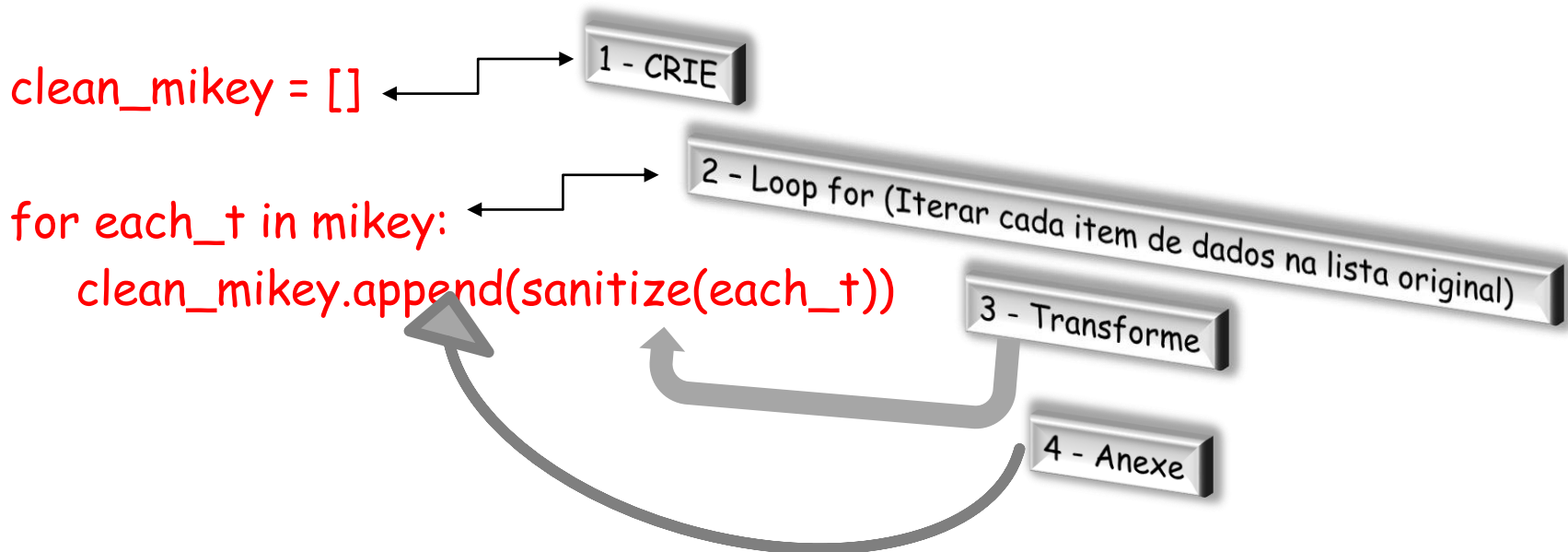
```
for each_t in julie:  
    clean_julie.append(sanitize(each_t))
```

```
for each_t in mikey:  
    clean_mikey.append(sanitize(each_t))
```

```
for each_t in sarah:  
    clean_sarah.append(sanitize(each_t))
```

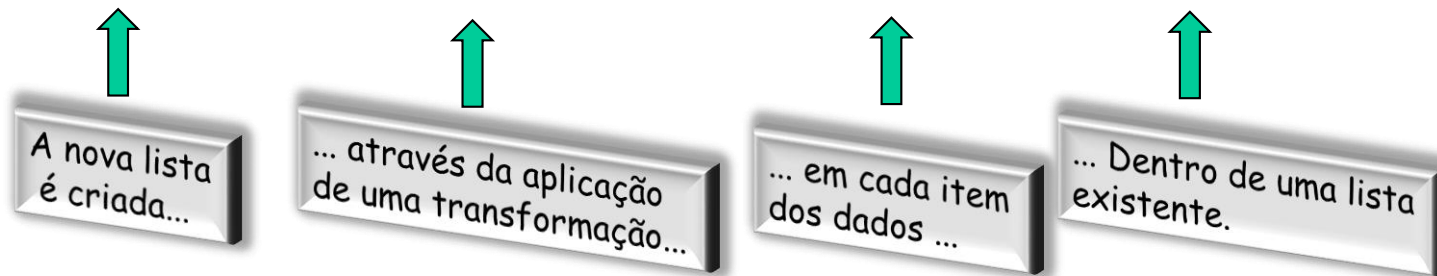
```
print(sorted(clean_james))  
print(sorted(clean_julie))  
print(sorted(clean_mikey))  
print(sorted(clean_sarah))
```

Compreendendo as listas



Aqui está a mesma funcionalidade como uma **compreensão** da lista, que envolve a criação de uma nova Lista especificando a **transformação** que será aplicada a cada um dos itens de dados dentro de uma Lista existente.

```
clean_mikey = [sanitize (each_t) for each_t in mikey]
```



O interessante é que a transformação foi reduzida a uma *única linha de código*. Além disso, não há nenhuma necessidade de especificar o uso do método **append()**, pois esta ação está implícita na compreensão da lista.

Exemplos de compreensão de listas.

Abra seu shell IDLE e acompanhe essas transformações de uma linha.

Comece a transformar uma lista de minutos em uma lista de segundos:


```
>>> min = [1, 2, 3]
>>> seg = [m * 60 for m in min]
>>> seg
[60, 120, 180]
```



Basta multiplicar os valores
do minuto por 60

Dada uma lista de strings minúsculas e misturadas, é fácil transformar as strings em MAIÚSCULAS

```
>>> texto = ['Eu', 'não', 'gosto', 'de spam']
>>> maiuscula = [s.upper() for s in texto]
>>> maiuscula
['EU', 'NÃO', 'GOSTO', 'DE SPAM']
```

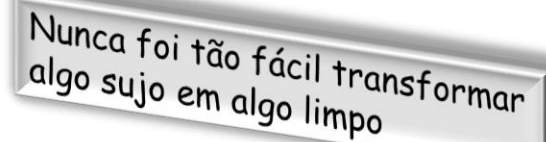


Cada string vem com o
método "upper()"

Exemplos de compreensão de listas.

Vamos usar a função `sanitize()` para transformar alguns dados da lista nas horas formatadas corretamente.

```
>>> dirty = ['2-22', '2:22', '2.22']  
>>> clean = [sanitize(t) for t in dirty]  
>>> clean  
['2.22', '2.22', '2.22']
```



Nunca foi tão fácil transformar algo sujo em algo limpo

Também é possível atribuir os resultados da transformação da lista de volta no identificador original.

Este exemplo transforma uma lista de strings em números de ponto flutuante, então substitui os dados da lista original

```
>>> clean = [float(s) for s in clean]  
>>> clean  
[2.22, 2.22, 2.22]
```



"float()" converte em ponto flutuante

Exemplos de compreensão de listas.

A transformação pode ser uma cadeia de funções:

```
>>> clean = [float(sanitize(t)) for t in ['2-22', '3:33', '4.44']]
```

```
>>> clean
```

```
[2.22, 3.33, 4.44]
```



Combinar transformações nos itens de dados é suportado também!

Agora que você conhece as compreensões da lista, vamos escrever quatro delas para processar as quatro listas de valores do tempo do treinador. Transforme cada uma de suas listas em uma versão ordenada e limpa delas mesmas.

O programa começa aqui com a função

```
def sanitize(time_string):  
    if '-' in time_string:  
        splitter = '-'  
    elif ':' in time_string:  
        splitter = ':'  
    else:  
        return(time_string)  
    (mins, secs) = time_string.split(splitter)  
    return(mins + '.' + secs)
```

```
with open('james.txt') as jaf:  
    data = jaf.readline()  
    james = data.strip().split(',')
```

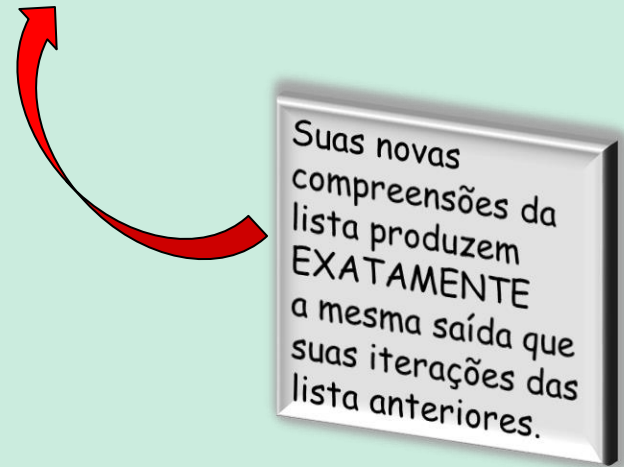
```
with open('julie.txt') as juf:  
    data = juf.readline()  
    julie = data.strip().split(',')
```

```
with open('mikey.txt') as mif:  
    data = mif.readline()  
    mikey = data.strip().split(',')
```

```
with open('sarah.txt') as saf:  
    data = saf.readline()  
    sarah = data.strip().split(',')
```

Continuação do programa

```
print(sorted ( [sanitize(t) for t in james] ))  
print(sorted ( [sanitize(t) for t in julie] ))  
print(sorted ( [sanitize(t) for t in mikey] ))  
print(sorted ( [sanitize(t) for t in sarah] ))
```



- Exercício 3

A sequência de Fibonacci é uma sequência de números inteiros que tem as seguintes regras de formação:

Os dois primeiros termos são 0 e 1; do terceiro em diante cada termo é a soma dos dois anteriores.

Escreva uma programa em Python que leia um número inteiro N , em seguida, mostre na tela os N primeiros termos da sequência de Fibonacci. Faça o programa de modo que N seja no mínimo 2.

- Resolução - Exercício 3

```
print ("Sequência de Fibonacci \n")
```

```
# leitura do número de termos
```

```
N = 0
```

```
while N < 2:
```

```
    try:
```

```
        N = int(input("Digite N ( > 1 ): "))
```

```
        if N < 2:
```

```
            print ("Digite N  > = 2")
```

```
    except:
```

```
        print ("O dado digitado deve ser um número inteiro.")
```

```
A = 0
```

```
B = 1
```

```
print ("0, 1, ", end="")      # exibe os dois primeiros termos
```

```
i=0
```

```
while i < N-2:                # o laço tem que exibir N - 2  termos
```

```
    C = A + B
```

```
    print ("{}", ".format (C), end="")  # end=""  suprime a mudança de linha na exibição em tela
```

```
    A = B
```

```
    B = C
```

```
    i += 1
```

```
print ("\n\n Fim do Programa ")
```

- Exercício 3

A sequência de Fibonacci é uma sequência de números inteiros que tem as seguintes regras de formação:

Os dois primeiros termos são 0 e 1; do terceiro em diante cada termo é a soma dos dois anteriores.

Escreva uma programa em Python que leia um número inteiro N, em seguida, mostre na tela os N primeiros termos da sequência de Fibonacci. Faça o programa de modo que N seja no mínimo 2.

PRODUZIR OS TRÊS MELHORES TEMPOS DE CADA ATLETA E REMOVA AS DUPLICATAS

Comece criando uma nova lista chamada `unique_james`, depois preencha com os itens de dados únicos encontrados em `james`. Além disso, forneça o código para exibir somente os três primeiros melhores tempos de James

Pode-se considerar o uso do operador `not in`.

```
unique_james = []  
for each_t in james:  
    if each_t not in unique_james:  
        unique_james.append(each_t)  
  
print(unique_james[0:3])
```

Crie a lista vazia para manter os
itens de dados únicos

Itere os dados existentes

Se o item de dados ainda NÃO
estiver na nova lista

... Acrescente o item de dados
único à nova lista.

Divida os três primeiros itens de
dados da lista e exiba-os na tela

O programa começa aqui com a função

```
def sanitize(time_string):
    if '-' in time_string:
        splitter = '-'
    elif ':' in time_string:
        splitter = ':'
    else:
        return(time_string)
    (mins, secs) = time_string.split(splitter)
    return(mins + '.' + secs)

with open('james.txt') as jaf:
    data = jaf.readline()
    james = data.strip().split(',')

with open('julie.txt') as juf:
    data = juf.readline()
    julie = data.strip().split(',')

with open('mikey.txt') as mif:
    data = mif.readline()
    mikey = data.strip().split(',')

with open('sarah.txt') as saf:
    data = saf.readline()
    sarah = data.strip().split(',')

```

Continuação do programa

```
print(sorted ( [sanitize(t) for t in james] ))
print(sorted ( [sanitize(t) for t in julie] ))
print(sorted ( [sanitize(t) for t in mikey] ))
print(sorted ( [sanitize(t) for t in sarah] ))

unique_james = []
for each_t in james:
    if each_t not in unique_james:
        unique_james.append(each_t)
print(unique_james[0:3])
unique_julie = []
for each_t in julie:
    if each_t not in unique_julie:
        unique_julie.append(each_t)
print(unique_julie[0:3])
unique_mikey = []
for each_t in mikey:
    if each_t not in unique_mikey:
        unique_mikey.append(each_t)
print(unique_mikey[0:3])
unique_sarah = []
for each_t in sarah:
    if each_t not in unique_sarah:
        unique_sarah.append(each_t)
print(unique_sarah[0:3])

```

Trabalhando com CONJUNTOS - `set()`

Além das listas, o Python vem também com a estrutura de dados do conjunto, que se comporta como os conjuntos aprendidos nas aulas de matemática.

As características predominantes dos conjuntos, no Python, são que os itens de dados em um conjunto são desordenados, e as *duplicatas não são permitidas*.

Se tentar adicionar um item de dados a um conjunto que já contém o item de dados, o Python simplesmente irá ignorá-lo.

Para criar um conjunto vazio usa-se `set()`:

Crie um conjunto novo e vazio, e atribua-o a uma variável

`distancias = set()`

Também é possível criar e preencher um conjunto em uma única etapa.

Pode-se fornecer uma lista de valores de dados entre colchetes, ou especificar uma lista existente como um argumento para o `set()`.

`distancias = {10.6, 11, 8, 10.6, "two", 7}`

Qualquer duplicata na lista fornecida de valores é ignorada. Neste exemplo o segundo item 10.6 é ignorado.

`Distancias = set(james)`

Qualquer duplicata na lista "james" é ignorada.

Crie uma função para abertura dos arquivos com with open

```
def get_coach_data(filename):  
    try:  
        with open(filename) as f:  
            data = f.readline()  
            return(data.strip().split(',') )  
    except IOError as ioerr:  
        print('File error: ' + str(ioerr))  
        return(None)
```

def sanitize(time_string):

if '-' in time_string:

splitter = '-'

elif ':' in time_string:

splitter = ':'

else:

return(time_string)

(mins, secs) = time_string.split(splitter)

return(mins + '.' + secs)

def get_coach_data(filename):

try:

with open(filename) as f:

data = f.readline()

return(data.strip().split(','))

except IOError as ioerr:

print('File error: ' + str(ioerr))

return(None)

james = get_coach_data('james.txt')

julie = get_coach_data('julie.txt')

mikey = get_coach_data('mikey.txt')

sarah = get_coach_data('sarah.txt')

print(sorted(set([sanitize(t) for t in james]))[0:3])

print(sorted(set([sanitize(t) for t in julie]))[0:3])

print(sorted(set([sanitize(t) for t in mikey]))[0:3])

print(sorted(set([sanitize(t) for t in sarah]))[0:3])

Função **sanitize**

Função
get_coach_data
Para abertura dos
arquivos

Trabalhando com novo arquivo texto, que contém mais dados

Agora no arquivo texto, além dos tempos, temos o nome do atleta e também a data de nascimento.

Se for usar "split()" para extrair os dados de Sarah para uma lista, o primeiro item de dado será o nome de Sarah, o segundo será sua data de nascimento e o resto serão os dados do tempo de Sarah.

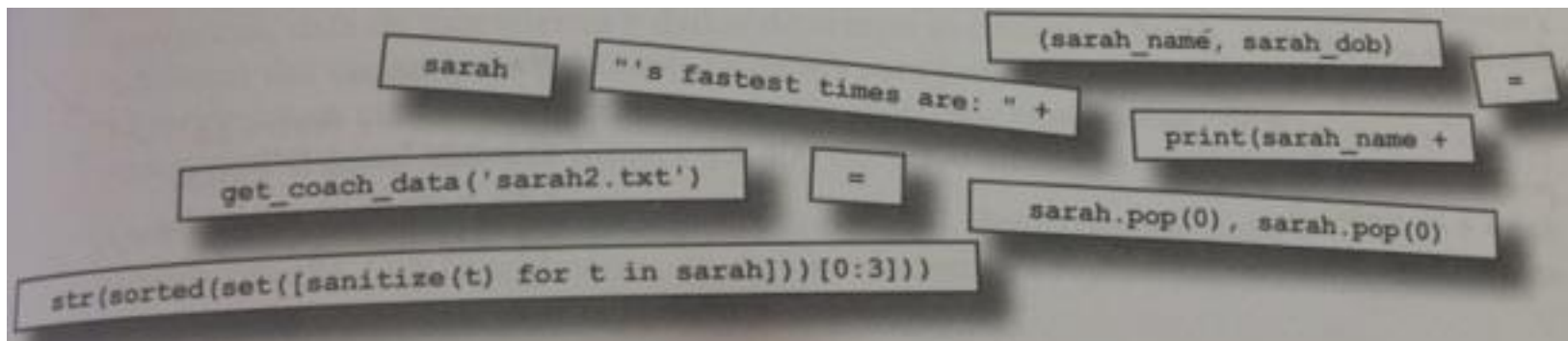
Vamos utilizar o método pop() para remover e retornar um item de dados a partir da localização da lista especificada.

Como ficaria o programa para ler os dados do novo arquivo da atleta Sarah?

1º. → Função sanitize → Para extrair os tempos

2º. → Função get_coach_data → Para abertura do arquivo

3º. → Abaixo, reorganize o código para implementar o processamento da lista requerido para extrair e processar os três tempos mais rápidos de Sarah.



```
def sanitize(time_string):  
    if '-' in time_string:  
        splitter = '-'  
    elif ':' in time_string:  
        splitter = ':'  
    else:  
        return(time_string)  
    (mins, secs) = time_string.split(splitter)  
    return(mins + '.' + secs)
```

Função **sanitize**

```
def get_coach_data(filename):  
    try:  
        with open(filename) as f:  
            data = f.readline()  
        return(data.strip().split(','))  
    except IOError as ioerr:  
        print('File error: ' + str(ioerr))  
        return(None)
```

Função
get_coach_data
Para abertura dos
arquivos

```
sarah = get_coach_data('sarah2.txt')
```

```
(sarah_name, sarah_dataNasc) = sarah.pop(0), sarah.pop(0)
```

```
print (sarah_name + " 'Os tempos mais rápidos são: " + str(sorted(set([sanitize(t) for t in sarah]))[0:3]))
```

A chamada "**pop(0)**" retorna e remove os dados da frente de uma lista. Duas chamadas para "**pop(0)**" remove os dois primeiros valores de dados e os atribui às variáveis nomeadas.

Trabalhando com DICIONÁRIO

No uso de lista para atleta Sarah, foi usado três variáveis, sendo uma para o nome da atleta, outra para data de nascimento e outra para o tempo.

As listas são ótimas, mas elas nem sempre são a melhor estrutura de dados para toda situação. A atleta Sarah por exemplo, além dos tempos, possui outros tipos de dados.

Há uma estrutura definida aqui: o nome do atleta, a data de nascimento, então a lista de tempos.

Vamos criar a parte dos dados do tempo em outra estrutura de dados, que associa todos os dados para um atleta a uma única variável.

Vamos usar um **dicionário** Python, que *associa os valores dos dados a chaves*.

As chaves:

Nome → "Sarah Sweeney"

Nasc → "2002-6-17"

Tempo → [2:58, 2.58, 2:39, 2-25, 2-55,]

Trabalhando com DICIONÁRIO

Exemplos de dicionário Python

Abra o IDLE e comece criando dois dicionários vazios, um usando chaves e outro usando uma função de fábrica:

```
>>> oliveira = { }
```

```
>>> silva = dict()
```

Ambas técnicas
criam um dicionário
vazio.

```
>>> type (oliveira)
```

```
>>> type (silva)
```

Confirmar com
"type", se foi
aberto como um
dicionário.

Trabalhando com DICIONÁRIO

Adicione alguns dados a ambos os dicionários *associando valores às chaves*.

Observe que a estrutura real dos dados está apresentada aqui, já que cada dicionário tem Nome e uma lista de Ocupações.

Note também que o dicionário silva está sendo criado ao mesmo tempo.


```
>>> oliveira ['Nome'] = 'Caio Oliveira'
```

```
>>> oliveira ['Ocupacao'] = ['Ator', 'Comediante', 'Escritor', 'Produtor filmes']
```

```
>>> silva = { 'Nome': 'Marco Silva', 'Ocupacao': ['Comediante', 'Ator', 'Escritor', 'tv'] }
```

```
>>> silva ['Nome']
```

```
>>> oliveira ['Ocupacao'] [-1]
```



Use números para acessar um item da lista armazenada em uma determinada chave do dicionário. Pense nisso como um "encadeamento de índices" e leia da direita para a esquerda

Trabalhando com DICIONÁRIO

Adicionando alguns dados sobre o local de nascimento a cada dicionário:

```
>>> silva ['LocalNasc'] = "Maceio, Alagoas, Brasil"  
>>> oliveira ['LocalNasc'] = "Manaus, Amazonas, Brasil"
```

```
>>> silva
```

```
>>> oliveira
```



Verifique agora quais dados estão nos dicionários

Trabalhando com DICIONÁRIO

Aplicar conceito de dicionário

Vamos continuar concentrados nos dados de Sarah.

Substitua o código que usa listas pelo novo código que usa um dicionário para manter e processar os dados de Sarah.

1º. → Função `sanitize` → Para extrair os tempos

2º. → Função `get_coach_data` → Para abertura do arquivo

3º. → Aqui iremos remover o código que usa a listas e vamos substituir por dicionário.

def sanitize(time_string):

if '-' in time_string:

splitter = '-'

elif ':' in time_string:

splitter = ':'

else:

return(time_string)

(mins, secs) = time_string.split(splitter)

return(mins + '.' + secs)

def get_coach_data(filename):

try:

with open(filename) as f:

data = f.readline()

return(data.strip().split(','))

except IOError as ioerr:

print('File error: ' + str(ioerr))

return(None)

sarah = get_coach_data('sarah2.txt')

sarah_data = { }

sarah_data ['Nome'] = sarah.pop(0)

sarah_data ['DataNasc'] = sarah.pop(0)

sarah_data ['Tempos'] = sarah

print (sarah_data ['Nome'] + "Os tempos mais rápidos são: " +

str(sorted(set([sanitize(t) for t in sarah_data['Tempos']])) [0:3]))

Função **sanitize**

Função
get_coach_data
Para abertura dos
arquivos

Criando dicionário vazio

Bibliografia

1) Administração de Redes com Scripts

Costa, Daniel Gouveia

Brasport

2) Python

Barry, Paul

Alta Books, 2012

3) Python3 – Conceitos e Aplicações

Banin, Sergio Luiz

Érica, 2018