Integrando o Python com Excel

Felipe C. R. dos Santos

1 Integrando o Python com Excel

A **importância** do Excel atualmente é **inegável**, além de fazer parte,junto ao Word e o Power Point, da tríplice **mais popular** do Office, seu uso **vem se tornando quase que uma necessidade** não só para o uso doméstico como também o coorporativo e o acadêmico. O programador, técnico de computação, analista de sistema ou profissional de qualquer outro **ramo da TI que nunca utilizou o Excel que atire a primeira pedra**. O profissional da contabilidade, administração, recursos humanos ou gerência que nunca o utilizou, **que atire as duas primeiras pedras**.

Sua **popularidade** além de altamente **perceptível** é facilmente **explicável**: trabalhos que **demorariam dias** e exigiam alto nível de cuidado, como faze o relatório fiscal de uma empresa se torna uma tarefa de **poucas horas** e com uma segurança enorme.

Mas tudo que é bom... *Pode melhorar, não é mesmo?*

Com o Excel não é diferente. Imagine pegar essa tarefa de **poucas horas** e transformar em tarefa de **poucos minutos**!

Através da programação você consegue de forma simples automatizar toda a criação e gerenciamento de uma planilha.

E falando em programação de forma simples... Impossível não lembrar do **Python**!

Pense em como seria bom se apenas **configurando uma única vez um automatizador de planilhas**, que vá até a fonte dos seus dados, o recupere, já efetuando toda a análise necessário, e os insira na planilha, você conseguisse praticamente **computadorizar todo o seu trabalho**? Além de evitar perder tempo com tarefas repetitivas, você basicamente estaria recebendo seu salário para apenas ficar analisando a execução do seu software e, se necessário, fazer uma atualização ou outro no código.

Neste curso vamos te ensinar todo o necessário para **construir do zero uma planilha completa e bem profissional** no Excel, que consiga atender as suas demandas e te ajudar a automatizar o seu trabalho.

Então sem mais delongas, *vamos começar*?

1.1 Python + Excel = OpenPyXL

O **OpenPyXL** é uma biblioteca Python para ler/gravar arquivos do Excel *xlsx/xlsm/xltx/xltm*. Nasceu da falta de biblioteca existente para ler/escrever nativamente do Python no formato *Office Open XML*.

Em resumo, da necessidade da manipulação de arquivos Excel no Python, surgiu o OpenPyXL.

O **OpenPyXL** consegue manipular tanto ***Workbooks*** quanto ***Worksheets*** de forma completa e bem fácil, portanto, vamos utilizá-lo durante este curso.

Para instalá-lo, basta usar o comando: > pip install openpyxl

1.2 Workbooks e Worksheets

No **Excel** uma **Worksheet** é uma planilha única, composta por linhas e colunas.
Já um **Workbook** (Pasta de Trabalho) é um conjunto de **Worksheets**.

Sempre que criamos um novo arquivo no **Excel** na verdade não estamos criando uma *Planilha*, mas sim, uma *Pasta de Trabalho*. Sendo assim, criamos um **Workbook** e não um **Worsheet**.

Portanto, no Python não seria diferente, criamos primeiro um **Workbook** e dentro desse objeto criamos nossa **Worksheet** (Planilha).

Por convensão do **OpenPyXL** nós utilizamos os seguintes nomes nas variáveis:
1. ***wb*** => **w**ork**b**ook 2. ***ws*** => **w**ork**s**heet

Porém, para uma melhor didática e compreensão, usaremos os nomes **pasta** e **planilha** (respectivamente) neste livro.

2 Importanto a biblioteca

Para podermos usar a biblioteca é preciso primeiramente importá-la no nosso projeto, para isso podemos usar:
>**import *openpyxl***

Porém, como o que majoritariamente usaremos é apenas a classe **Workbook** podemos importá-lo da seguinte forma:
> **from *openpyxl* import *Workbook***

Dessa forma, conseguimos usá-la diretamente (sem necessidade de usar *openpyxl.Workbook*)

```
from openpyxl import Workbook
```

3 Criando nosso primeiro workbook

Para criar um workbook é bem simples, basta **instanciarmos** (criarmos) um novo **objeto** (variável) do tipo **Workbook**.
Para isso, basta executar o construtor da classe **Workbook**, como no código abaixo:

```
pasta = Workbook()
```

4 Salvando um arquivo

Seria **muito** sem graça se não pudéssemos salvar nosso trabalho com o OpenPyXL, não?
Felizmente, temos um jeito bem simples de salvar tudo o que fizemos em um arquivo **.xlsx**.

Para isso, basta chamarmos a função **save()** do nosso **Workbook** (pasta de trabalho) e passar como parâmetro (valor entre os parênteses) o nome do nosso arquivo com a extensão **xlsx** (obrigatóriamente).

```
pasta.save('arquivo.xlsx')
```

5 Abrindo planilhas já existentes

Muitas vezes já temos um arquivo previamente feito e queremos utilizá-lo, ao invez de criar uma nova desde o zero.

Para isso, precisamos **carregar** um **workbook**, certo? Então nada mais justo que um método chamado justamente isso, no caso **load_workbook()**.

Então, primeiramente, precisamos importar nosso método responsável por carregar workbooks:

from *openpyxl* **import** *load_workbook*

```
from openpyxl import load_workbook
```

Agora é só chamar a função **load_workbook()**, onde passamos por parâmetro o caminho e nome do nosso arquivo existente.

Lembrando que se o arquivo estiver na mesma pasta que o seu projeto, você pode apenas passar o nome dele.

```
pasta_existemte = load_workbook('arquivo_existente.xlsx')
```

6 Trabalhando com planilhas

6.1 Criando nossa primeira planilha

É sempre bom saber que **um workbook já vem com uma planilha padrão criada, chamada de *Sheet***.

Porém, também temos a opção de criar novas planilhas, através do método **create_sheet()**.
Esse método possui dois parâmetros, sendo eles:

1. **title**: nome (título) da planilha => Obrigatório;
2. **index**: índice (posição) onde queremos inserir nossa planilha => Opcional.

Caso não enviemos o parâmetro índice a planilha será inserida no final, como a última no workbook.

```
pasta.create_sheet("Criando Planilha");
```

Além disso, o método **create_sheet()** é retornável, sendo assim, podemos atribuir essa planilha que criamos para uma variável e usá-la depois:

```
planilha = pasta.create_sheet("Minha Planilha")
planilha = pasta.create_sheet("Planilha Indice",1) # Insere no␣
 ↪índice 1
```

6.2 Nome da planilha

Também podemos recuperar o título (nome) da planilha, através da propriedade **title**.

```
nome = planilha.title
print("O nome da Planilha é:",nome)
```

```
[Output]:O nome da Planilha é: Planilha Indice
```

6.3 Renomeando nossa planilha

Podemos também atribuir um valor para a propriedade **title** da planilha, dessa forma, conseguimos renomear a planilha.

```
planilha.title = 'Planilha Criada'
```

6.4 Listando planilhas

Para listarmos todas as **worksheets** (planilhas), podemos usar a propriedade **worksheets** da nossa pasta (**workbook**).

Essa propriedade nos retornará uma lista com elementos do tipo **worksheet**, então podemos utilizá-las para editar a planilha futuramente.

```
planilhas = pasta.worksheets
print(planilhas)
```

```
[Output]:[<Worksheet "Sheet">, <Worksheet "Planilha Criada">,␣
↪<Worksheet "Criando
Planilha">, <Worksheet "Minha Planilha">]
```

Podemos também lista o nome de todas as planilhas, através da propriedade **sheetnames**.

Desta vez, não serão retornados elementos do tipo **worksheet**, apenas strings contendo o nome da planilha mesmo.

```
nomes_planilhas = pasta.sheetnames
print(nomes_planilhas)
```

```
[Output]:['Sheet', 'Planilha Criada', 'Criando Planilha', 'Minha␣
↪Planilha']
```

6.5 Recuperando a planilha ativa (selecionada)

Podemos também saber qual a planilha que virá ativa (selecionada) no Excel, ou seja, caso tivermos mais de uma planilha na nossa pasta, podemos informal ao Excel qual delas estará selecionada ao abrir o arquivo.

Pasa isso podemos usar a propriedade **active**.

```
planilha_atual = pasta.active
print(planilha_atual)
```

```
[Output]:<Worksheet "Sheet">
```

6.6 Obtendo planilhas

6.6.1 Obtendo planilhas por nome

Podemos também obter uma planilha da nossa pasta, ou seja, procuramos uma planilha na nossa pasta e retornamos essa planilha buscada.

Para isso podemos buscar na nossa pasta da seguinte forma:

pasta[**'nome_da_planilha'**]

Onde o nome da planilha é passado como string (entre aspas).

```
planilha = pasta['Planilha Criada']
print(planilha)
```

```
[Output]:<Worksheet "Planilha Criada">
```

6.6.2 Obtendo planilhas por nome de forma segura

Porém, se tentarmos recuperar uma planilha não existente iremos obter um erro do tipo **KeyError**, ou seja, a chave que estamos buscando (nome da planilha) não existe na nossa pasta.

```
planilha = pasta['Planilha']
```

```
[Output]:KeyError: 'Worksheet Planilha does not exist.'
```

Para isso, podemos primeiro checar se a pasta contem uma planilha com esse nome (através da propriedade **sheetnames**, lembra?).

Caso existir, então vamos obter a planilha na pasta. Do contrário, avisamos que não existe tal planilha.

```
if 'Planilha' in pasta.sheetnames:
    planilha = pasta['Planilha']
else:
    print("Não há nenhuma planilha com esse nome")
```

```
[Output]:Não há nenhuma planilha com esse nome
```

6.6.3 Obtendo planilhas pelo índice

Podemos também obter a planilha através do índice (posição) dela na nossa pasta.

Para isso, podemos usar a propriedade **worksheets** e informar o índice que queremos, como por exemplo:

planilha = pasta.worksheets[0]

`Onde 0 é o índice que queremos recuperar.`

```
indice = 1
planilha = pasta.worksheets[indice]

print(planilha)
```

```
[Output]:<Worksheet "Planilha Criada">
```

6.6.4 Obtendo planilhas pelo índice de forma segura

Podemos também enfrentar erros devido ao índice buscado ser inválido, seja ele menor do que 0 ou maior que o máximo.

```
indice = 10
pasta.worksheets[indice]
```

```
[Output]:IndexError: list index out of range
```

Para contornar isso, podemos primeiro checar se o índice está entre o intervalo mínimo **0** e o máximo (**tamanho - 1**)

```
indice = 10
if indice < len(pasta.worksheets) and indice >= 0:
    pasta.worksheets[indice]
else:
    print("Índice não está dentro no intervalo válido.\nMínimo: 0␣
 ↪| Máximo:",str(len(pasta.worksheets)-1))
```

```
[Output]:Índice não está dentro no intervalo válido.
Mínimo: 0 | Máximo: 3
```

6.7 Mudando a planilha ativa (selecionada)

Podemos também alterar a pasta ativa (selecionada) no Excel.

Assim como usamos a propriedade **active** para recuperar a planilha selecionada, podemos também usá-la para alterar a planilha atualmente ativa.

```
pasta.active = pasta['Minha Planilha']
print(pasta.active)
```

```
[Output]:<Worksheet "Minha Planilha">
```

6.8 Copiando planilhas

Podemos também criar uma cópia de uma planilha (todos seus valores de cada célula) e passar para outra planilha.

Para isso podemos usar o método **copy()** e enviar por parâmetro a planilha a ser copiada.

```
original = pasta['Minha Planilha']
copia = pasta.copy_worksheet(original)

print(copia.title)
```

```
[Output]:Minha Planilha Copy
```

6.9 Alterando a cor de fundo da aba da planilha

Por fim, podemos também editar as propriedades das planilhas, como por exemplo a cor da aba da planilha, que fica no canto inferior do Excel.

Para isso, acessamos a propriedade **sheet_properties.tabColor** da planilha e atribuir a essa propriedade o **valor hexadecimal** da nossa cor.

```
planilha.sheet_properties.tabColor = '34EBB7'
```

7 Trabalhando com células

7.1 Recuperando células

7.1.1 Recuperando uma célula

Assim como podemos pegar uma planilha através da pasta usando uma **chave** para pesquisa (igual em um objeto do tipo **dicionário**), podemos fazer o mesmo com as células, passando uma chave correspondente a ela na nossa planilha.

Para isso, trabalhos com a chave igual ao Excel, exemplos:
1. Célula **A1** => Chave = **A1**;
2. Célula **B3** => Chave = **B3**;
3. Célula **D7** => Chave = **D7**.

Ou seja, uma letra para representar a coluna e um número para representar a linha.

Dessa forma, podemos pegar a célula da seguinte maneira:

celula = planilha[**'CHAVE'**]

```
celula = planilha['A1']
print(celula)
```

```
[Output]:<Cell 'Planilha Criada'.A1>
```

7.1.2 Recuperando uma célula pelo índice

Podemos também pegar pelo índice da célula, onde informamos o valor da linha (**row**) e coluna (**colum**).

Lembrete que para facilitar, on índices de ambas iniciam em ***1***.

Dessa forma:

1. Coluna 1 x Linha 1 = A1
2. Coluna 3 x Linha 4 = C4
3. Coluna 2 x Linha 5 = B5

Agora, para pegarmos essa célula, vamos usar a função **cell()** na nossa planilha, onde passamos dois parâmetros **row** (linha) e **column** (coluna).

```
celula = planilha.cell(row=4, column=2) # Coluna 2 = B  Linha 4 =␣
 ↪4  => Célula = B4
print(celula)
```

```
[Output]:<Cell 'formula'.B4>
```

7.1.3 Recuperando o valor de uma célula

Para recuperarmos o valor de uma célula podemos usar a propriedade **value** da célula.

Lebrando que: Caso não tenha nada retorna **None**.

Como até agora no tutorial não inicializamos nenhum valor pra célula, o valor então será o padrão (**None**).

```
celula = planilha['B4']
valor = celula.value
print(valor)
```

```
[Output]:None
```

Agora na próxima etapa nós já atribuimos o valor de 10 para célula (você verá isso em breve).

Vamos ver o código de recuperar o valor novamente então?

```
valor = celula.value
print(valor)
```

```
[Output]:10
```

7.1.4 Recuperando o valor de uma célula pelo índice

Podemos também recuperar uma célula através do indice dela (como vimos anteriormente) e como o retorno também será um objeto do tipo **cell**, podemos igualmente recuperar o valor da propriedade **value**.

```
celula = planilha.cell(row=4, column=2) # Coluna 2 = B  Row 4 = 4ª linha  => Célula = B4
print(celula.value)
```

```
[Output]:10
```

7.2 Inserindo valores

7.2.1 Inserindo um valor em uma célula

Já para atribuir um valor para uma célula, podemos novamente pegar a **propriedade** value e atribuir um valor a isso.

Como por exemplo:

planilha[**'Chave'**].***value*** = 'valor'

```
planilha['A1'].value = 'Sou o A1'
print(planilha['A1'].value)
```

```
[Output]:Sou o A1
```

Maaaas... Podemos também fazer de uma forma mais **resumida**.

Para isso, podemos apenas pegar a célula e atribuir um valor diretamente ao objeto **cell**, não precisando pegar a propriedade **value**.

Ou seja... Para **inserirmos** um valor, a gente pode artribuir diretamente na variável e automaticamente esse valor será inserido na propriedade **value**:

planilha[**'Chave'**] = 'valor'

```
planilha['B3'] = 3
print(planilha['B3'].value)
```

```
[Output]:3
```

7.2.2 Inserindo um valor em uma célula pelo índice

Também podemos usar o método **cell()** (que já usamos anteriormente para recuperar uma planilha) e usar um terceiro parâmetro, chamado **value**. O valor passado neste parâmetro será inserido como valor para a célula:

```
celula = planilha.cell(row=4, column=2, value = 'Bê Quatro')
print(celula.value)
```

```
[Output]:Bê Quatro
```

7.3 Recuperando células em um intervalo

7.3.1 Intervalo entre células

Além de recuperarmos o valor de apenas uma célula, podemos pegar os valores de um intervalo de céluda de uma só vez.

Por exemplo, em vez de pegarmos o valor das celulas **A1, A2, A3, A4, B1, B2, B3 e B4**, podemos pegar de uma vez só todas as células do **intervalo** entre **A1 e B4**, que daria no mesmo e seria bem mais prático.

Para isso, podemos usar a função de **fatiar** do python, que pega apenas uma **fatia** (parte) da nossa sequência.
As fatias no **Python** funcionam da seguinte maneira: ['**Posição Inicial**' : '**Posição Final**'].

Ou seja, para pegar entre o intervalo **A1 e B2**, iniciando em **A1** e terminando em **B2**, temos o seguinte:

['**A1**' : '**B2**']

Então para pegar esse intervalo de célula podemos apenas pedir ao Python para retornar essa fatia da planilha:

intervalo_celulas = planilhas['**A1**' : '**B2**']

```
intervalo_celulas = planilha['A1':'B2']
print(intervalo_celulas)
```

```
[Output]:((<Cell 'Planilha Criada'.A1>, <Cell 'Planilha Criada'.
 ↪B1>), (<Cell 'Planilha
Criada'.A2>, <Cell 'Planilha Criada'.B2>))
```

7.3.2 Intervalo entre linhas

Podemos também em vez de usar a chave que representa a célula (*Coluna e Linha, ex: A1,B2,C3...*) para fatiar, podemos usar o valor que representa as **linhas** que queremos recuperar, dessa forma, podemos pegar **todas as células nessas linhas**.

Como as linhas são representadas por números, para pegar todas as células das linhas 2 e 3 usamos:

intervalo_linhas = planilha['2':'3']

```
intervalo_linhas = planilha['2':'3']
print(intervalo_linhas)
```

```
[Output]:((<Cell 'Planilha Criada'.A2>, <Cell 'Planilha Criada'.
 ↪B2>), (<Cell 'Planilha
Criada'.A3>, <Cell 'Planilha Criada'.B3>))
```

Como alternativa, podemos também **iterar** as linhas, usando a função **iter_rows()**, onde temos os seguintes 4 parâmetros:

1. **min_row**: A linha inicial (linha mínima)
2. **min_col**: A coluna inicial (coluna mínima)
3. **max_row**: A linha final (linha máxima)
4. **max_col**: A coluna final (coluna máxima)

Sendo assim, para pegar entre o intervalo **A1** e **C4** precisamos pega entre as células **Coluna1 x Linha1** e **Coluna3 x Linha4**.
Dessa forma:

1. **min_row**: 1
2. **min_col**: 1
3. **max_row**: 4
4. **max_col**: 3

```
for linha in planilha.iter_rows(min_row=1, min_col=1, max_row=4,␣
 ↪max_col=3):
    print("Linha: ",linha)
```

```
[Output]:Linha:  (<Cell 'Planilha Criada'.A1>, <Cell 'Planilha␣
 ↪Criada'.B1>, <Cell
'Planilha Criada'.C1>)
Linha:  (<Cell 'Planilha Criada'.A2>, <Cell 'Planilha Criada'.B2>,␣
 ↪<Cell
'Planilha Criada'.C2>)
Linha:  (<Cell 'Planilha Criada'.A3>, <Cell 'Planilha Criada'.B3>,␣
 ↪<Cell
'Planilha Criada'.C3>)
Linha:  (<Cell 'Planilha Criada'.A4>, <Cell 'Planilha Criada'.B4>,␣
 ↪<Cell
'Planilha Criada'.C4>)
```

7.3.3 Intervalo entre colunas

Assim como fizemos com as linhas, podemos também **fatiar as colunas**, ou seja, podemos pegar toas as células de um determinado intervalo de colunas.

A única diferença é que **colunas são letras**, então a primeira coluna vale **A**, a segunda vale **B**, a terceira **C** e por assim vai...

Caso queira pegar todas as células das colunas de A até C, podemos usar:

intervalo_colunas = planilha['**A**' : '**C**']

```
intervalo_colunas = planilha['A':'C']
print(intervalo_colunas)
```

```
[Output]:((<Cell 'formula'.A1>, <Cell 'formula'.A2>, <Cell␣
 ↪'formula'.A3>, <Cell
'formula'.A4>), (<Cell 'formula'.B1>, <Cell 'formula'.B2>, <Cell␣
 ↪'formula'.B3>,
<Cell 'formula'.B4>), (<Cell 'formula'.C1>, <Cell 'formula'.C2>,␣
 ↪<Cell
'formula'.C3>, <Cell 'formula'.C4>))
```

Podemos também usar o método **iter_cols()**, para iterarmos as colunas, assim como o **iter_rows()** para as linhas (igual fizemos anteriormente), inclusive tendo os mesmos parâmetros:

Veja um exemplo:

```
for coluna in planilha.iter_cols(min_row=1, min_col=1, max_row=4,␣
 ↪max_col=3):
    print("Coluna: ",coluna)
```

```
[Output]:Coluna:  (<Cell 'Planilha Criada'.A1>, <Cell 'Planilha␣
 ↪Criada'.A2>, <Cell
'Planilha Criada'.A3>, <Cell 'Planilha Criada'.A4>)
Coluna:  (<Cell 'Planilha Criada'.B1>, <Cell 'Planilha Criada'.B2>,␣
 ↪<Cell
'Planilha Criada'.B3>, <Cell 'Planilha Criada'.B4>)
Coluna:  (<Cell 'Planilha Criada'.C1>, <Cell 'Planilha Criada'.C2>,␣
 ↪<Cell
'Planilha Criada'.C3>, <Cell 'Planilha Criada'.C4>)
```

7.3.4 Todas células de uma linha

Já para pegar todas as células de uma só linha podemos pegar como **chave de pesquisa** justamente aquela linha que queremos, como por exemplo:

```
intervalo_celulas_coluna = planilha['2']
print(intervalo_celulas_coluna)
```

```
[Output]:(<Cell 'Planilha Criada'.A2>, <Cell 'Planilha Criada'.B2>,␣
 ↪<Cell 'Planilha
Criada'.C2>)
```

7.3.5 Todas células de uma coluna

Do mesmo jeito, podemos pegar todas as células de uma só coluna, também passando só a **chave de pesquisa** dessa coluna, como por exemplo:

```
intervalo_celulas_coluna = planilha['A']
print(intervalo_celulas_coluna)
```

```
[Output]:(<Cell 'Planilha Criada'.A1>, <Cell 'Planilha Criada'.A2>,␣
 ↪<Cell 'Planilha
Criada'.A3>, <Cell 'Planilha Criada'.A4>)
```

7.3.6 Todas as linhas de uma planilha

Agora se quisermos uma lista de todas as linhas de uma planilha podemos usar o *generator* **planilha.rows**, criando uma lista (ou tupla, caso queira) a partir deste *generator*, veja um exemplo:

```
intervalo_todas_linhas = list(planilha.rows)
print(intervalo_todas_linhas)
```

```
[Output]:[(<Cell 'Planilha Criada'.A1>, <Cell 'Planilha Criada'.
 ↪B1>, <Cell 'Planilha
Criada'.C1>), (<Cell 'Planilha Criada'.A2>, <Cell 'Planilha Criada'.
 ↪B2>, <Cell
'Planilha Criada'.C2>), (<Cell 'Planilha Criada'.A3>, <Cell␣
 ↪'Planilha
Criada'.B3>, <Cell 'Planilha Criada'.C3>), (<Cell 'Planilha Criada'.
 ↪A4>, <Cell
'Planilha Criada'.B4>, <Cell 'Planilha Criada'.C4>)]
```

7.3.7 Todas as colunas de uma planilha

Agora se quisermos uma lista de todas as colunas de uma planilha podemos usar o *generator* **planilha.columns**, criando uma lista (ou tupla, caso queira) a partir deste *generator*, veja um exemplo:

```
intervalo_todas_colunas = list(planilha.columns)
print(intervalo_todas_colunas)
```

```
[Output]:[(<Cell 'Planilha Criada'.A1>, <Cell 'Planilha Criada'.
 ↪A2>, <Cell 'Planilha
Criada'.A3>, <Cell 'Planilha Criada'.A4>), (<Cell 'Planilha Criada'.
 ↪B1>, <Cell
'Planilha Criada'.B2>, <Cell 'Planilha Criada'.B3>, <Cell 'Planilha
Criada'.B4>), (<Cell 'Planilha Criada'.C1>, <Cell 'Planilha Criada'.
 ↪C2>, <Cell
'Planilha Criada'.C3>, <Cell 'Planilha Criada'.C4>)]
```

7.3.8 Todos os valores de uma planilha

Podemos também pegar todos os **valores** de uma planilha, ou seja, não pegamos de fato a célula, mas sim o valor dela.

Para isso, podemos usar a propriedade **values** da planilha.

Lembrando que caso alguma posição não tenha um valor atribuido ainda, o seu valor será *None*.

```
for linha in planilha.values:
    for valor in linha:
        print(valor)
```

```
[Output]:1
None
C1
A2
2
None
Três
B
C3
None
None
:)
```

7.3.9 De forma tabelada

Aproveitando a propriedade **values** da planilha para pegar os **valores** de **todas as células**, podemos notar que ao printar os valores nós temos algo bem estranho e feio, sendo difícil de entender.

Porém... Há um jeito fácil de contornar esse problema, podemos **tabelas** o output através da formatação da string.

Usando o método **format** da classe string nós podemos usar *'{:<25}'* para alinhas a o dado a esquerda usando um tamanho fixo de 25 caracteres.

```
for linha in planilha.values:
    for valor in linha:
        if valor == None: valor ='None'
        print('{:<25}'.format(valor),end=' ')
    print()
```

```
[Output]:
1                         None                      C1
A2                        2                         None
Três                      B                         C3
None                      None                      :)
```

7.4 'Condição' de existência de uma célula

Quando criamos uma nova planilha ela vem **vazia**, ou seja, não existe nenhuma célula nela.

As células começam a existir assim que elas são acessadas, chega para **recuperar** um valor ou para **atribuir** um valor.

Ou seja... Caso acessarmos a célula **B3** a planilha irá ser '**criada**' até lá, então começará a ter as células:

```
A1, A2, A3, B1, B2 e B3.
```

Agora se acessarmos a célula **C4**, notaremos que a linha 4 não existe, bem como a coluna 3, dessa forma precisamos criá-las. Sendo assim passaremos a ter:

```
A1, A2, A3, A4, B1, B2, B3, B4, C1, C2, C3 e C4
```

Portanto, é necessário ter um cuidado extra na hora de acessar células sem precisar, pois se acessarmos a célula **J10** iremos precisar armazenas na nossa memória 10 linhas e 10 colunas, ou seja, 100 células serão armazenadas na nossa memória.

```
nova_pasta = Workbook()
nova_planilha = nova_pasta.active
print(f"Linhas: {len(list(nova_planilha.rows))} x Colunas:␣
 ↪{len(list(nova_planilha.columns))}")

celula = nova_planilha['B3']
print(f"Linhas: {len(list(nova_planilha.rows))} x Colunas:␣
 ↪{len(list(nova_planilha.columns))}")

nova_planilha['C4'] = 'A'
print(f"Linhas: {len(list(nova_planilha.rows))} x Colunas:␣
 ↪{len(list(nova_planilha.columns))}")
```

```
[Output]:
Linhas: 0 x Colunas: 0
Linhas: 3 x Colunas: 2
Linhas: 4 x Colunas: 3
```

7.5 Mesclar e separar células

7.5.1 Mesclar

Podemos também mesclar e separar células, o que acaba sendo útil para agrupamento de dados ou criação de títulos com fontes de grande tamanho.

Para isso podemos usar o método **merge_cells()** e passar como parâmetro uma **string** contendo o intervalo (fatia) a se mesclar.

Por exemplo:

planilha.merge_cells('A2:D2')

Para mesclar as células entre **A2** e **D2**.

PS: Precisamos notar que agora **não** é de fato uma fatia passada por parâmetro (formato [**'A2'** : **'D2'**]), mas sim, uma **string** representando essa fatia (formato **'A2:D2'**).

```
planilha.merge_cells('A2:D2')
```

7.5.2 Separar (desfazer mesclagem)

Para **desfazer** essa mesclagem, podemos usar o método **unmerge_cells()**, que pode ser utilizado do mesmo jeito do que o método **merge_cells()**, sendo os mesmos parâmetros e formato, a única diferenteça é que agora nós estamos **separando** as células previamente mescladas.

```
planilha.unmerge_cells('A2:D2')
```

7.6 Modificando o tipo do valor numérico

Podemos também modificar o **tipo** do valor numérico, ou seja, o jeito de **fomatar** aquele dado.

Dessa maneira, podemos fazer com que uma célula seja uma **data** (formato yyyy-mm-dd h:mm:ss), o que no **Excel** nos mostrará 2019-09-03 para o dia 3 de setembro de 2019.

Além disso, podemos criar **formatos customizados**, como por exemplo, **'# ##0.000'** o que fará o número ***'1234.5'*** ser mostrado como ***'1 2345.500'***.

Por padrão, todos os números inicial no formato **'General' (geral)**.

```
import datetime
planilha['A4'] = datetime.datetime(2019,9,3)
print(planilha['A4'].number_format)

planilha['A5'] = 2.05
print(planilha['A5'].number_format)

planilha['A7'] = 1245.25
planilha['A7'].number_format = '# ##0.000'   # 1 234.500
print(planilha['A7'].number_format)
```

```
[Output]:yyyy-mm-dd h:mm:ss
General
# ##0.000
```

7.7 Inserindo Imagens

Além de dados textuais e numéricos (que querendo ou não são caracteres, igual aos textos) podemos também ter imagens inseridas nas nossas planilhas.

Em primeiro lugar, precisamos importar a classe que controlará essa imagem para usarmos, para isso, podemos importar **Image** de **openpyxl.drawing.image**:

from* *openpyxl.drawing.image*** **import** ***Image***

```
from openpyxl.drawing.image import Image
```

Pronto, agora podemos já usar as imagens, para isso, primeiro vamos instanciar um objeto do tipo **Image**, passando por parâmetro o caminho para uma imagem e em seguida usar o método **add_image()**, passando como parâmetro (respectivamente) a imagem e a **chave** (posição Coluna e Linha) da célula para inserirmos, como por exemplo:

```
imagem = Image('c:\\logo-bylearn.jpg')
planilha.add_image(imagem, 'D5')
```

7.8 Criando um gráfico

Uma planilha matemática sem gráficos? Então não é uma planilha matemática!

Sabendo disso, o **OpenPyXL** obviamente nos trás a possibilidade de criar gráficos e inserí-los nas nossas planilhas.

Dentre os princiais tipos de gráficos temos:

1. Bar => Barras;
2. Column => Colunas;
3. Line => Linhas;
4. Pie => Pizza (setores);
5. Surface => Superfície.

Caso queira ver todos os outros tipos, você pode consultar a documentação: https://bit.ly/2luaLiy

Para criamos os gráficos primeiro importar:

1. O tipo de gráfico que queremos => neste exemplo queremos um gráfico de barras, portanto, importaremos **BarChart**;
2. Referências => Que servirá para definirmos o intervalo que pegaremos para criamos nossa série, importaremos **References**;
3. Series => Agora sim, a nossa série de fato, a que será inserida no gráfico, importaremos **Series**.

Importaremos tudo isso de **openpyxl.char**, sendo assim:

```
from openpyxl.chart import BarChart, Reference, Series
```

Para este exemplo iremos criar uma nova planilha e populá-la (inserir dados nela):

```
planilha = pasta.create_sheet('Gráficos')

#Popular 10 linhas da planilha com 1 valor em cada uma (crescente).
 ↪
for i in range(10):
    planilha.append([i])
```

Agora, para criar o gráfico faremos os procedimentos a seguir:

1. Criar o gráfico:
   1.1 Para isso usaremos o construtor de BarChart, por ser nossa escolha de gráfico.
2. Criar a série (dados a serem inseridos):
   2.1 Para isso, pegaremos essa série a partir de uma referência que possui os parâmetros:
   2.1.1 worksheet: a planilha de onde pegaremos a refência;
   2.1.2 min_col: coluna inicial (mínima); 2.1.3 max_col: coluna máxima (final);
   2.1.4 min_row: linha inicial (mínima); 2.1.5 max_row: linha final (máxima);
3. Adicionar a série ao nosso gráfico:
   3.1 Através do método **add_data()** do nosso gráfico, passando nossa série por parâmetro.
4. Adicionar o gráfico na planilha:
   4.1 Através do método **add_char()** passando por parâmetro (respectivamente) o gráfico a ser inserido e a chave da célula (coluna e linha).

Vejamos o exemplo:

```
grafico = BarChart()

serie = Reference(planilha, min_col=1, min_row=1, max_col=1,␣
 ↪max_row=10)

grafico.add_data(serie)

planilha.add_chart(grafico,'E15')
```

Podemos também ter gráficos mais completos, com títulos e eixos nomeados.

De início, criaremos outra planilha. Repare desta vez que os dados possuem uma primeira linha que virá a ser usada como título para esses dados e as demais linhas com valores numéricos:

```
planilha = pasta.create_sheet('Gráfico completo')

# Criamos os dados para o gráfico
dados = [
    ('Número original','Quadrado','Cubo'),
    (1,1,1),
    (2,4,8),
    (3,9,27),
    (4,16,64),
    (5,25,125)
]

# Inserimos os dados de cada linha no gráfico
for linha in dados:
    planilha.append(linha)
```

Agora editaremos melhor o gráfico mudando:

1. O título (propriedade **title**);
2. O título de seu eixo_x (propriedade **x_axis.title**);
3. O título do seu eixo_y (propriedade **y_axis.title**).

```
# Criamos um gráfico de barras e setamos seu nome e título de␣
 ↪eixos (X e Y)
grafico = BarChart()
grafico.title = "Numero ao Quadrado e Cubo"
grafico.y_axis.title = 'Valor resultado'
grafico.x_axis.title = 'Numero original'
```

Agora iremos configurar a série, configurando as referências e as categorias

Referências: Setamos as referencias (posições na planilha dos valores para o gráfico e categorias):

1. Neste caso, os valores são os quadrados (1,4,9,16 e 25) e cubos (1,8,27,64,125);

2. As categorias são os numeros 'bases' dos quais esses valores são o quadrado/cubo (1,2,3,4,5).

3. Valores são os mostrados nas barras do gráfico (eixo Y);
   1.1 Então pegamos a partir da segunda coluna (eliminar as categorias).

4. Categorias são as mostradas na parte inferior do gráfico (eixo X);
   2.2 Então pegamos só da primeira coluna, que é onde estão as categorias.

```
valores = Reference(planilha, min_col=2, min_row=1, max_row=6,␣
 ↪max_col=3) # Colunas 2 e 3, da linha 1 até 6
categorias = Reference(planilha, min_col=1, min_row=2, max_row=6,␣
 ↪max_col=1) # Apenas coluna 1, da linha 1 até 6
```

Por fim, iremos configurar os ados no gráfico, configurando os valores, os títulos e as categorias.

Analisando os dados temos que:

1. Nós temos os números bases (os quais vamos elevar ao quadrado e ao cubo) => categoria (ficará na parte de baixo);

2. Os números quadrados => Um tipo de valor a ser analisado;
3. Os números ao cubo => Outro tipo de valor a ser analisado.

Ou seja:

Tanto os números quadrados quanto os ao cubo são os valores e a primeira linha de cada um possui um título, então vamos usar tais linhas como título, através do parâmetro **titles_from_data** sendo setado como ***True*** no médoto **add_data()**.

Por fim, para as categorias temos o método **set_categories** no nosso gráfico, onde mandamos as categorias por parâmetro.

```
grafico.add_data(valores, titles_from_data=True)
grafico.set_categories(categorias)
```

Por fim, para já testarmos tudo o que inserimos, iremos salvar o arquivo:

```
pasta.save('arquivo.xlsx')
```

7.9 Usando fórmulas

Podemos também usar as fórmulas disponíveis no **Excel**, como a de soma, por exemplo.

Infelizmente, temos uma limitação no OpenPyXL onde ele aceita **apenas fórmulas em inglês**.

Dessa maneira, a soma que é **'=SOMA(1,5)'** se torna **'=SUM(1,5')**.

Para inserir uma fórmula uma célula, basta alterar o seu valor para ser uma string contendo tal fórmula, como por exemplo:

```
planilha = pasta.create_sheet('formula')
planilha['C3'] = "=SUM(1,5)"
```

8 Trabalhando com estilos

Completando nosso aprendizado iremos aprender como estilizar células, mudando seu preenchimento, borda, alinhamento, fonte e cores.

Para isso importaremos as seguintes classes:

1. PatternFill;
2. Border;
3. Side;
4. Alignment;
5. Protection;
6. Font;
7. Color.

Tudo isso no namespace **openpyxl.styles**, da seguinte forma:

```
from openpyxl.styles import PatternFill, Border, Side, Alignment,␣
↪Protection, Font, Color
```

8.1 Propriedades dos estilos

Todos os estilos precisam ser **criados**, não é possível **editar** um já pronto.

Dessa forma, precisamos, por exemplo, criar uma nova fonte para atribuíla como a fonte atual de uma planilha específica.

Todas as classes mensionadas acima possuem um construtor próprio onde pode ser passado todas as propriedades dela.

8.1.1 Fonte

Para customizarmos uma fonte temos as seguintes propriedades:

1. name => Nome da fonte;
2. size => Tamanho da fonte;
3. bold => Se há ou não negrito;
4. itali => Se há ou não itálico;
5. vertAlign => Alinhamento vertical;
6. underline => Se há um tracedo embaixo da célula (underline);
7. strike => Se há um traçado no meio da célula;
8. color => Cor da fonte (hexadecimal).

Por padrão, as fontes possuem a seguinte configuração, onde você só precisa enviar por parâmetro as configurações que queira editar:

```
font = Font(name='Calibri',
            size=11,
            bold=False,
            italic=False,
            vertAlign=None,
            underline='none',
            strike=False,
            color='FF000000')
```

8.1.2 Preenchimento por padrão

Para customizarmos um preenchimento temos as seguintes propriedades:

1. fill_type => tipo do preenchimento => ex: 'solid';
2. fgColor => cor do foreground (plano da frente);
3. bgColor => cor do background (plano de fundo).

Por padrão, os preenchimentos possuem a seguinte configuração, onde você só precisa enviar por parâmetro as configurações que queira editar:

```
fill = PatternFill(fill_type=None,
                   fgColor='FFFFFFFF',
                   bgColor='FF000000')
```

8.1.3 Borda

Já para customizarmos a borda é um pouco diferente...

Nós primeiro teremos que decidir o que da borda vamos editar, sendo eles:

1. left => Esquerda;
2. right => Direita;
3. top => Cima;
4. bottom => Baixo;
5. diagonal => Diagonal;
6. outline => Contorno;
7. vertical => Vertical;
8. horizontal => Horizontal;
9. diagonal_direction => Direção da Diagonal.

Com exceção do *'diagonal_direction'* que é um inteiro que define a direção da diagonal, nós temos que passar um objeto do tipo **Side**, que por sua vez possui:

1. border_style => Estilo da borda;
2. color => Cor (em Hexadecimal).

Por padrão, as bordas possuem a seguinte configuração, onde você só precisa enviar por parâmetro as configurações que queira editar:

```
border = Border(left=Side(border_style=None,
                          color='FF000000'),
                right=Side(border_style=None,
                           color='FF000000'),
                top=Side(border_style=None,
                         color='FF000000'),
                bottom=Side(border_style=None,
                            color='FF000000'),
                diagonal=Side(border_style=None,
                              color='FF000000'),
                diagonal_direction=0,
                outline=Side(border_style=None,
                             color='FF000000'),
                vertical=Side(border_style=None,
                              color='FF000000'),
                horizontal=Side(border_style=None,
                               color='FF000000')
                )
```

8.1.4 Alinhamento

Para customizarmos um preenchimento temos as seguintes propriedades:

1. horizontal => tipo do preenchimento => ex: 'solid';
2. vertical => cor do foreground (plano da frente);

3. text_rotation => cor do background (plano de fundo);

4. wrap_text => quebrar linha automaticamente (não ultrapassar limite horizontal);

5. shrink_to_fit => enconlher a fonte para caber na célula;

6. indent => identar a célula.

Por padrão, os preenchimentos possuem a seguinte configuração, onde você só precisa enviar por parâmetro as configurações que queira editar:

```
alignment=Alignment(horizontal='general',
                    vertical='bottom',
                    text_rotation=0,
                    wrap_text=False,
                    shrink_to_fit=False,
                    indent=0)
```

8.1.5 Formato Numérico

Esse já é conhecido aqui do curso, usamores apenas propriedade **number_format** para definirmos o tipo de formatação para aquele número.

Assim como já vimos, o padrão é '***General***'.

```
number_format = 'General'
```

8.1.6 Proteção

Para customizarmos a 'proteção' temos as seguintes propriedades:

1. locked => se a célula é bloqueada à edição;
2. hidden => se a célula é oculta.

Por padrão, as proteções possuem a seguinte configuração, onde você só precisa enviar por parâmetro as configurações que queira editar:

```
protection = Protection(locked=True,
                        hidden=False)
```

8.2 Estilizando células

Antes estilizarmos as células nós precisamos primeiro ter uma variável/objeto daquela célula. Para isso, nós já sabemos como fazer, é apenas atribuir o valor de uma célula para uma determinada variável (*obter um planilha*).

Agora podemos acessar as propriedades de estilo (como **fill** e **font**) e alterá-las atribuindo o valor de alguma customização nossa, ex:

fonte = Font(color='FFBB00')
preenchimento = PatternFill(fill_type='solid', fgColor='FFFF00FF')

Dessa forma, criamos uma nova fonte e um novo preenchimento.
Agora vamos setar essas customizações:

celula_a1.fill = preenchimento
celula_a1.font = fonte

```
celula_a1 = planilha['A1']
celula_b3 = planilha['B3']
celula_c4 = planilha['C4']

fonte = Font(color='FFBB00')
preenchimento = PatternFill(fill_type='solid', fgColor='FFFF00FF')

celula_a1.fill = preenchimento
celula_a1.font = fonte
celula_b3.font = fonte
celula_c4.font = Font(color='FFBBFF', italic = True)

celula_a1.value = "Testando"
celula_b3.value = "nossos"
celula_c4.value = "estilos"
```

8.3 Copiando estilos

Criar estilos são chatos, nós sabemos... Ainda mais quando a alteração é mínima.

Por sorte, temos como copiar um estilo a partir de um outro.

Para isso, primeiro precsamos importar o método **copy()** do namespace de mesmo nome (**copy**):

```
from copy import copy
```

Agora basta chamarmos o método **copy()** e enviar como parâmetro o estilo que queremos copiar. Tal método terá como retorno um estilo do mesmo tipo que foi passado como parâmetro, ex:

Se passarmos uma fonte como parâmentro, teremos uma fonte como retorno:

fonte2 = copy(fonte1)

```
fonte1 = Font(name='Verdana',size=16,color='FFBB00')
fonte2 = copy(fonte1)
fonte2.bold = True

print('\n Fonte:'\n',fonte2)
print('\n Fonte2:'\n',fonte2)
```

```
[Output]:    Fonte1:

<openpyxl.styles.fonts.Font object>
Parameters:
name='Verdana', charset=None, family=None, b=False, i=False,␣
 ↪strike=None,
outline=None, shadow=None, condense=None, color=<openpyxl.styles.
 ↪colors.Color
object>
Parameters:
rgb='00FFBB00', indexed=None, auto=None, theme=None, tint=0.0,␣
 ↪type='rgb',
extend=None, sz=16.0, u=None, vertAlign=None, scheme=None



  Fonte2:

 <openpyxl.styles.fonts.Font object>
Parameters:
name='Verdana', charset=None, family=None, b=True, i=False,␣
 ↪strike=None,
outline=None, shadow=None, condense=None, color=<openpyxl.styles.
 ↪colors.Color
object>
Parameters:
rgb='00FFBB00', indexed=None, auto=None, theme=None, tint=0.0,␣
 ↪type='rgb',
extend=None, sz=16.0, u=None, vertAlign=None, scheme=None
```

8.4 Estilizando Intervalos

8.4.1 Estilizando determinadas colunas

Podemos também estilizar diretamente um certo intervalo de colunas, como por exemplo, todas as células da coluna **A**.

Para isso basta chamar a propriedade **column_dimensions** e passar como chave o valor da coluna (letra), como por exemplo:

coluna_a = planilha.column_dimensions[‘A’]

Agora vamos atribuir uma font para toda essa coluna:

coluna_a.font = Font(bold=True)

```
coluna_a = planilha.column_dimensions['A']
coluna_a.font = Font(bold=True)
```

8.4.2 Estilizando determinadas linhas

Podemos também estilizar diretamente um certo intervalo de linhas, como por exemplo, todas as células da linha **1**.

Para isso basta chamar a propriedade **row_dimensions** e passar como chave o valor da linha (número), como por exemplo:

linha_1 = planilha.row_dimensions['1']

Agora vamos atribuir uma font para toda essa coluna:

linha_1.font = Font(bold=True)

```
linha_1 = planilha.row_dimensions[1]
linha_1.font = Font(underline="single")
```

9 Configurações de impressão

Por fim, vamos aprender a configurar a impressão da nossa planilha.

Embora não é muito usual imprimir planilhas, as vezes precisaremos... E imprimir 150 linhas e 200 colunas para mostrar apenas 5 células além de exagero torna a visualização horrível, então vamos aprender a configurar impressões?

9.1 Opções de alinhamento

A primeira configuração que veremos são as opções de alinhamento, que temos duas:

1. print_options.horizontalCentered => Se a impressão será centralizada horizontalmente na folha;

2. print_options.verticalCentered => Se a impressão será centralizada verticalmente na folha;

```
planilha.print_options.horizontalCentered = True
planilha.print_options.verticalCentered = True
```

9.2 Cabeçalhos e Rodapés

Agora temos os rodapés e capeçalhos para definir.

Primeiros vamos definir os tipos de cabeçalhos e rodapés:

1. Cabeçalhos:
    - oddHeader => Cabeçalho de páginas ímpares;
    - evenHeader => Cabeçalho de páginas pares;
    - firstHeader => Cabeçalho da primeira página.
2. Rodapés:
    - oddFooter => Rodapé de páginas ímpares;
    - evenFooter => Rodapé de páginas pares;
    - firstFooter => Rodapé da primeira página.

Agora vamos ver as posições que nós temos:

1. left => Esquerda;
2. right => Direita;
3. center => Central.

Pronto! Mas... Qual o motivo de aprendermos isso?

Bom... para editarmos os cabeçalhos e rodapés precisamos da combinação de ambos, sendo primeiro o tipo a ser editado e o segundo a posição.

Vamos supor que queremos os cabeçalhos das páginas pares na posição esquerda, então teremos a propriedade **oddHeader.left** da nossa planilha.

Agora, nós podemos editar as propriedades deste cabeçalho, sendo eles:

1. text => texto;
2. size => tamanho da fonte;
3. font => nome da fonte;
4. color => cor da fonte.

```
planilha.oddHeader.left.text = "Isso é uma página par"
planilha.oddHeader.left.size = 14
planilha.oddHeader.left.font = "Tahoma,Bold"
planilha.oddHeader.left.color = "FFE9FF"
```

9.3 Área de Impressão

Por fim, temos a área de impressão, que nos informa quais serão as células que irão aparecer na impressão.

Todas as células fora deste intervalo não aparecerão na hora de imprimir.

Para isso editaremos a propriedade **print_area** da nossa planilha, onde iremos atribuir uma string especificando a 'fatia' da planilha a se cortar como, por exemplo, **'A1:F10'** para o intervalo entre **A1** e **F10**.

Veja o exemplo abaixo:

```
planilha.print_area = 'A1:F10'
```

10 Desafio - Planilha Completa do Zero

Finalmente!

Depois de muito esforço e dedicação nós chegamos ao final do livro!

Agora que tal criarmos uma planilha utilizando boa parte do que aprendemos?

No código abaixo vamos criar um nova pasta (**workbook**), criar algumas planilhas, edtar as cores das abas da planilha, inserir valores nas células, inserir imagens, trabalhar com gráficos e salvar nosso arquivo.

Vamos lá?

```
# Fazemos os imports necessários
from openpyxl.workbook import Workbook
from openpyxl.drawing.image import Image
from openpyxl.chart import BarChart, Reference, Series

# Criamos um novo Workbook
pasta = Workbook()

# Renomeamos a planilha inicial e mudamos suas cores
planilha_numeros = pasta.worksheets[0]
planilha_numeros.title = 'Numeros'
planilha_numeros.sheet_properties.tabColor = '8334EB'

# Criaremos outras planilhas e também mudaremos as cores delas
planilha_bylearn = pasta.create_sheet('ByLearn')
planilha_bylearn.sheet_properties.tabColor = '34EBB7'

planilha_imagem = pasta.create_sheet('Imagem')
planilha_imagem.sheet_properties.tabColor = '80BF82'

planilha_grafico = pasta.create_sheet('Graficos')
planilha_grafico.sheet_properties.tabColor = 'CF9A4C'

# Inserindo valores iniciando por um e aumentando em 2 a cada␣
 ↪celula
# 10 números por linhas,  Durante 5 linhas

i = 0
for linha in range(5):
    for coluna in range(10):
        # Para cada célula nas 10 colunas e 5 linhas que queremos,␣
 ↪vamos setar um valor crescente
        celula = planilha_numeros.cell(row=linha+1,␣
 ↪column=coluna+1, value = i)
        i += 2;

# Criamos os cabeçalhos
planilha_bylearn['A1'] = 'Rede Social'
```

```
planilha_bylearn['B1'] =  'Link'

# As listas (todas válidas, podem seguir a gente lá :D)
lista = [['Facebook','@ByLearn'],
['Insta','@ByLearn'],
['Youtube','@ByLearn'],
['Blog','blog.bylearn.com.br']]

# Adicionamos uma sublista por vez, uma para cada linha
for linha in lista:
    planilha_bylearn.append(linha)

# Carregamos uma imagem do disco e inserimos na posição A1
imagem = Image('c://logo-bylearn.jpg')
planilha_imagem.add_image(imagem,'A1')

# Criamos os dados para o gráfico
dados = [
    ('Prova','Competidor1', 'Competidor2'),
    ('Futebol',10, 30),
    ('Basquete',40, 60),
    ('Vôlei',50, 70),
    ('Baseball',20, 10),
    ('Corrida',10, 40),
    ('Handball',50, 30),
]

# Inserimos os dados de cada linha no gráfico
for linha in dados:
    planilha_grafico.append(linha)


# Criamos um gráfico de barras e setamos seu nome e título de␣
 ↪eixos (X e Y)
grafico = BarChart()
grafico.title = "Competição Esportiva"
grafico.y_axis.title = 'Pontuação'
grafico.x_axis.title = 'Prova'

# Setamos as referencias (posições na planilha dos valores para o␣
 ↪gráfico e categorias)
valores = Reference(planilha_grafico, min_col=2, min_row=1,␣
 ↪max_row=7, max_col=3)
categoria_jogos = Reference(planilha_grafico, min_col=1,␣
 ↪min_row=2, max_row=7, max_col=1)
```

```
# Configuramos os valores do gráfico e que ele deve pegar os␣
 ↪títulos através desses valores (primeira posição de cada um)
grafico.add_data(valores, titles_from_data=True)

# Setamos as categorias
grafico.set_categories(jogos)

# Agora é só adicionar na planilha
planilha_grafico.add_chart(grafico, "A10")

# Por fim, salvamos o arquivo
pasta.save('arquivo_final.xlsx')
```

11 Documentação Oficial

O aprendizado nunca deve terminar e uma grande fonte de estudo a partir de agora será a documentação oficial.
Com todo o ensino que você aprendeu até agora no curso, você entenderá facilmente tudo o que estiver lá na documentação.

Link: https://openpyxl.readthedocs.io/en/latest

12 Extra: Conteúdos diários gratuitos

Por fim, antes de terminarmos nosso livro, precisamos te falar sobre os conteúdos diários que criamos para você, um **ByLearner**!

Temos vídeos, blogs, explicações de códigos, curiosidades, lives, minicursos, enquetes e muito mais... Inclusive os nossos *Drops*.

Caso você ainda não saiba o que é um ***Drops da ByLearn***, é simpes:
- Através da nossa didática rápida e direto ao ponto preparamos um conteúdo bem legal de uma forma bem interessante para você assistir no tempo livre.
- Sabe quando está no face/insta e surge um vídeo legal de 1 minutinho no seu feed? Ou quando surge uma imagem bem chamativa e criativa?
- Então... Agora imagine se esse video bem supimpa fosse educativo, ou se essa imagem criativa te ensinasse a programar... - É exatamente isso que é um *Drops*.

O nosso foco é conseguir o interesse do aluno e proporcionar um conhecimento novo qualquer hora do dia!

12.1 Para isso você pode nos achar em:

Facebook: @ByLearn
Instagram: @ByLearn
Youtube: @Bylearn
Twitter: @ByLearn
LinkedIn: @ByLearn
Github: @ByLearn

Bom... Você entendeu né? É tudo ***@ByLearn***, procura lá `[U+2764]`.

12.2 Para encontrar nossos cursos:

Acesse nosso site em: http://bylearn.com.br.

13 Muito obrigado por tudo! Nos vemos no próximo curso

Foi um prazer enorme criar esse livro para vocês e ensiná-lo durante todo esse curso.
Esperamos que tenha gostado e que o conteúdo tenha sido útil para você.