Working with Styles?

Styles are used to change the look of your data while displayed on screen. Setting cell styles is done by creating new Style Objects and by assigning them to properties of cell. There are several style objects such as :

- ⇨ Font
- ⇨ Fill
- ⇨ Border
- ⇨ Alignment

Styles are defined in openpyxl.styles modules

```
from openpyxl.styles import *
```

To view the list of styles available in openpyxl.styles module

```
print(dir(openpyxl.styles)
```

Output

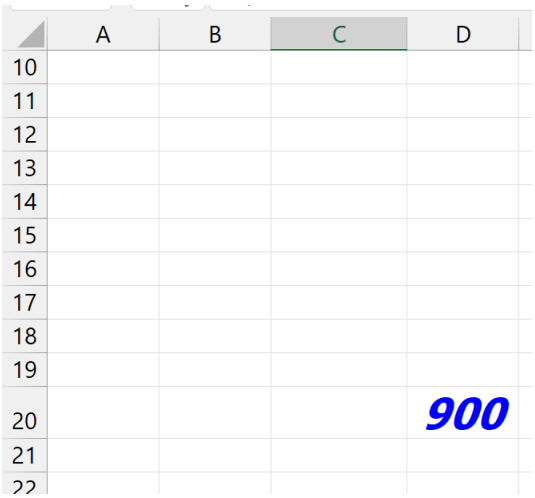
['Alignment', 'Border', 'Color', 'DEFAULT_FONT', 'Fill', 'Font', 'GradientFill', 'NamedStyle', 'NumberFormatDescriptor', 'PatternFill', 'Protection', 'Side', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__path__', '__spec__', 'alignment', 'borders', 'builtins', 'cell_style', 'colors', 'differential', 'fills', 'fonts', 'is_builtin', 'is_date_format', 'named_styles', 'numbers', 'protection', 'proxy', 'styleable', 'stylesheet', 'table']

We will have to create a Style Object with desired formatting style and then assign it to any particular cell. For Example:

```
mfont = Font(name='Tahoma', size=18, color=colors.BLUE, bold=True, italic=True, strike=False)
```

```
mcell.font = mfont
```

Output



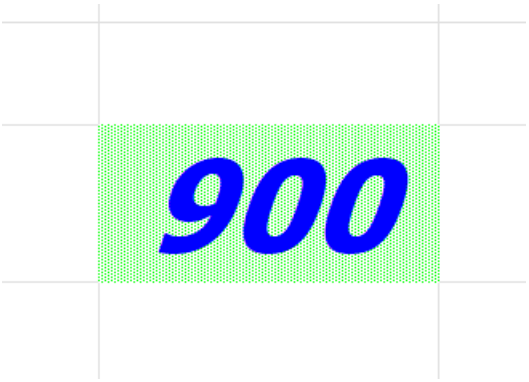
How to create a Pattern Fill ?

mfill = PatternFill(fill_type='lightGray', fgColor='00FF00')

'00FF00' is the RGB value for green

mcell.fill = mfill

Openpyxl does not have predefined color attributes like 'GREEN' ,‘YELLOW’ , ‘RED’ etc in openpyxl.styles.colors, so you should use the RGB values for colors



How to apply Border Style?

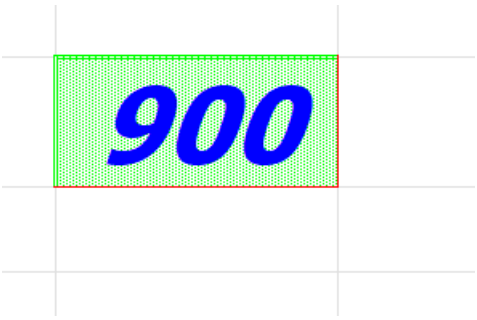
Inorder to apply border style formatting to cells, we need to create four different border objects , in case we want to apply different borders on all four sides of cell, just as shown below. To create border style, Side method is used and there we can specify, border style, border color etc attributes

dbl_border_green = Side(border_style='double', color='00FF00') # '00FF00' is the RGB value for green

thin_border_red = Side(border_style='thin', color='FF0000') # 'FF0000' is the RGB value for red

mcell.border = Border(left=dbl_border_green, right=thin_border_red, top=dbl_border_green, bottom=thin_border_red)

Output



How to set alignment of text?

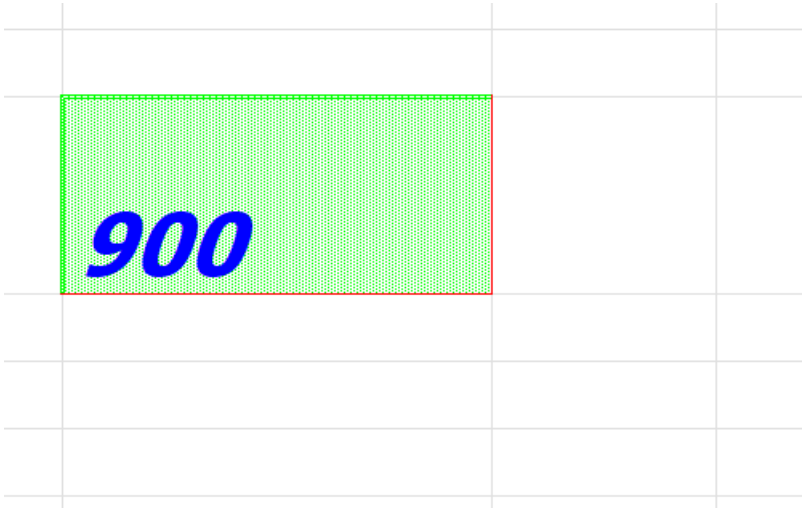
To set the alignment of cell contents, Alignment method is used and we need to specify the horizontal and vertical alignment values as follows

- Create an alignment object

  align_cell=Alignment(horizontal='left',vertical='bottom')

- Set the alignment property of cell object

  mcell.alignment=align_cell



How to copy style from one cell to another?

At first we need to import copy , as follows

from copy import copy

```
new_mcell = msheet['B2']
new_font = copy(mcell.font)
new_font.color.rgb = '00FF00'  # '00FF00' is the RGB value for green
new_mcell.font = new_font
```

Output

	A	B	C	D
1	Name	Age	Department	Salary
2	Rahul	25	A/c	12345
3	Dev	24	Computers	14526
4	Raj	22	Sales	14785

Understanding ‘copy’ module

The copy module in Python provides a way to create shallow and deep copies of objects. The copy function from this module is used to create a new object with the same content as the original object. There are two main types of copies: shallow copy and deep copy.

Shallow Copy:

A shallow copy creates a new object but does not create copies of nested objects. Instead, it copies references to the nested objects. This means that changes to the nested objects in the copied structure will be reflected in both the original and copied structures.

It can be useful when you want a new object with the same outer structure, but you are okay with sharing the nested objects.

Deep Copy:

A deep copy creates a new object and recursively copies all nested objects within the original structure. This ensures that changes to the nested objects do not affect the original structure, and vice versa.

It is useful when you want an entirely independent copy of the original object.