Using Excel with Python

John Strickler

Version 1.0, September 2021

Table of Contents

Chapter 1: Using openpyxl with Excel spreadsheets

Objectives

- Learn the basics of **openpyxl**
- Open **Excel** spreadsheets and extract data
- Update spreadsheets
- Create new spreadsheets
- Add styles and conditional formatting

The openpyxl module

- Provides full read/write access to Excel spreadsheets
- Creates new workbooks/worksheets
- Does not require Excel

The **openpyxl** module allows you to read, write, and create **Excel** spreadsheets.

openpyxl does not require Excel to be installed.

You can open existing workbooks or create new ones. You can do most anything that you could do manually in a spreadsheet – update or insert data, create formulas, add or change styles, even hide columns.

When you open an existing spreadsheet or create a new one, openpyxl creates an instance of `Workbook`. A Workbook contains one or more `Worksheet` objects.

From a worksheet you can access cells directly, or create a range of cells.

The data in each cell can be manipulated through its `.value` property. Other properties, such as `.font` and `.number_format`, control the display of the data.

View the full documentation at http://openpyxl.readthedocs.org/en/latest/index.html.

TIP	To save typing, import **openpyxl** as **px**.

Reading an existing spreadsheet

- Use `load_workbook()` to open file
- Get active worksheet with `WB.active`
- List all worksheets with `get_sheet_by_name()`
- Get named worksheet with `WB['worksheet-name']`

To open and read an existing spreadsheet, use the `load_workbook()` function. This returns a WorkBook object.

There are several ways to get a worksheet from a workbook. To list all the sheets by name, use `WB.get_sheet_names()`.

To get a particular worksheet, index the workbook by sheet name, e.g. `WB['sheetname']`.

A workbook is also an iterable of all the worksheets it contains, so to work on all the worksheets one at a time, you can loop over the workbook.

```
for ws in WB:
    print(ws.title)
```

The `.active` property of a workbook is the currently active worksheet. `WB.active` may be used as soon as a workbook is open.

The `.title` property of a worksheet lets you get or set the title (name) of the worksheet.

Example

px_load_worksheet.py

```
#!/usr/bin/env python
import openpyxl as px

def main():
    wb = px.load_workbook('../DATA/presidents.xlsx')

# three ways to get to a worksheet:

    # 1
    print(wb.sheetnames, '\n')
    ws = wb['US Presidents']
    print(ws, '\n')

    # 2
    for ws in wb:
        print(ws.title, ws.dimensions)
    print()

    # 3
    ws = wb.active
    print(ws, '\n')

    print(ws['B2'].value)


if __name__ == '__main__':
    main()
```

px_load_worksheet.py

```
['US Presidents', 'President Names']

<Worksheet "US Presidents">

US Presidents A1:J47
President Names B2:C47

<Worksheet "President Names">

Washington
```

Worksheet info

- Worksheet attributes
  - dimensions
  - min_row
  - max_row
  - min_column
  - max_column
  - *many others…*

Once a worksheet is opened, you can get information about the worksheet directly from the worksheet object.

The dimensions are based on the extent of the cells that actually contain data.

NOTE	Worksheets can have a maximum of 1,048,576 rows and 16,384 columns.

Example

px_worksheet_info.py

```
#!/usr/bin/env python
import openpyxl as px


def main():
    """program entry point"""
    wb = px.load_workbook('../DATA/presidents.xlsx')
    ws = wb['US Presidents']

    print("Title:", ws.title)
    print("Dimensions:", ws.dimensions)
    print("Minimum column:", ws.min_column)
    print("Minimum row:", ws.min_row)
    print("Maximum column:", ws.max_column)
    print("Maximum row:", ws.max_row)
    print("Parent:", ws.parent)
    print("Active cell:", ws.active_cell)

if __name__ == '__main__':
    main()
```

px_worksheet_info.py

```
Title: US Presidents
Dimensions: A1:J47
Minimum column: 1
Minimum row: 1
Maximum column: 10
Maximum row: 47
Parent: <openpyxl.workbook.workbook.Workbook object at 0x7f80a1015fd0>
Active cell: J48
```

Table 1. Useful Worksheet Attributes

Attribute	Data type	Description
`active_cell`	str	coordinates ("A1"-style) of active cell
`columns`	generator	iterable of all columns, as tuples of Cell objects
`dimensions`	str	coordinate range ("A1:B2") of all cells containing data
`encoding`	str	text encoding of worksheet
`max_column`	int	maximum column index (1-based)
`max_row`	int	maximum row index (1-based)
`mime_type`	str	MIME type of document
`min_column`	int	minimum column index (1-based)
`min_row`	int	minimum row index (1-based)
`parent`	Workbook	Workbook object that this worksheet belongs to
`rows`	generator	iterable of all rows, as tuples of Cell objects
`selected_cell`	str	coordinates of currently selected cell
`tables`	dict	dictionary of tables
`title`	str	title of this worksheet
`values`	generator	iterable of all values in the worksheet (actual values, not Cell objects)

NOTE min and max row/column refer to extent of cells containing data

Accessing cells

- Each cell is instance of Cell
- Attributes
  - value
  - number_format
  - font
  - *and others*
- Get cell with
  - `ws["COORDINATES"]`
  - `ws.cell(row, column)`

A worksheet consists of *cells*. There are two ways to access an individual cell:

- lookup the cell using the cell coordinates, e.g. `ws["B3"]`.
- specify the row and column as integers (1-based) using the `.cell` method of the worksheet, e.g. `ws.cell(4, 5)`. You can used named arguments for the row and column: `ws.cell(row=4, column=5)`.

In both cases, to get the actual value, use the `.value` attribute of the cell.

NOTE	Cell coordinates are case-insensitive.

Example

px_access_cells.py

```
#!/usr/bin/env python
import openpyxl as px

def main():
    wb = px.load_workbook('../DATA/presidents.xlsx')
    ws = wb['US Presidents']

    # access cell by cell name
    print(ws['A1'].value)
    print(ws['C2'].value, ws['B2'].value)
    print()

    # same, but lower-case
    print(ws['a1'].value)
    print(ws['c2'].value, ws['b2'].value)
    print()

    # access cell by row/column (1-based)
    print(
        ws.cell(row=27, column=3).value,  # "C27"
        ws.cell(row=27, column=2).value,  # "B27"
    )
    print()

    # same, without argument names
    print(
        ws.cell(27, 3).value,  # "C27"
        ws.cell(27, 2).value,  # "B27"
    )
    print()

if __name__ == '__main__':
    main()
```

px_access_cells.py

```
Term
George Washington

Term
George Washington

Theodore Roosevelt

Theodore Roosevelt
```

Table 2. Cell attributes

Attribute	Type	Description
`alignment`	openpyxl.styles.alignment.Alignment	display alignment info
`border`	openpyxl.styles.borders.Border	border info
`col_idx`	int	column index as integer
`column`	int	column index as integer
`column_letter`	str	column index as string
`comment`	any	cell comment
`coordinate`	str	coordinate of cell (e.g. ("B2")
`data_type`	str	data type code (s=str, n=numeric, etc)
`encoding`	str	text encoding (e.g. 'utf-8')
`fill`	openpyxl.styles.fills.PatternFill	fill (background) style
`font`	openpyxl.styles.fonts.Font	font color, family, style
`has_style`	bool	True if cell has style assigned
`hyperlink`	openpyxl.worksheet.hyperlink.Hyperlink	hyperlink for cell
`internal_value`	any	value of cell
`is_date`	bool	True if value is a date
`number_format`	str	Code for number format, e.g. "0.0"
`parent`	openpyxl.worksheet.worksheet.Worksheet	worksheet in which cell is located
`protection`	openpyxl.styles.protection.Protection	protection settings (e.g., hidden, locked)
`quotePrefix`	bool	character used for quoting
`row`	int	row index
`style`	str	name of style
`value`	any	actual value of cell

Getting raw values

- Use `worksheet.values`
- Returns row generator
- Each row is a tuple of column values

To iterate over all the values in the spreadsheet, use `worksheet.values`. It is a generator of the rows in the worksheet. Each element is a tuple of column values.

Only populated cells will be part of the returned data.

Example

px_raw_values.py

```
#!/usr/bin/env python
import openpyxl as px

def main():
    wb = px.load_workbook('../DATA/presidents.xlsx')
    ws = wb['US Presidents']  ①
    headers = next(ws.values)   ②
    for row in ws.values:  ③
        print(row[:5])   ④


if __name__ == '__main__':
    main()
```

① get active sheet

② read first row from generator

③ loop over rows in generator

④ print first 5 elements of row tuple

px_raw_values.py

```
('Term', 'Last Name', 'First Name', 'Birth Date', 'Death Date')
(1, 'Washington', 'George', '1732-02-22', '1799-12-14')
(2, 'Adams', 'John', '1735-10-30', '1826-07-04')
(3, 'Jefferson', 'Thomas', '1743-04-13', '1826-07-04')
(4, 'Madison', 'James', '1751-03-16', '1836-06-28')
(5, 'Monroe', 'James', '1758-04-28', '1831-07-04')
(6, 'Adams', 'John Quincy', '1767-07-11', '1848-02-23')
(7, 'Jackson', 'Andrew', '1767-03-15', '1845-06-08')
(8, 'Van Buren', 'Martin', '1782-12-05', '1862-07-24')
(9, 'Harrison', 'William Henry', '1773-02-09', '1841-04-04')
```

```
...
```

```
(37, 'Nixon', 'Richard Milhous', '1913-01-09', '1994-04-22')
(38, 'Ford', 'Gerald Rudolph', '1913-07-14', '2006-12-26')
(39, 'Carter', "James Earl 'Jimmy'", '1924-10-01', 'NONE')
(40, 'Reagan', 'Ronald Wilson', '1911-02-06', '2004-06-05')
(41, 'Bush', 'George Herbert Walker', '1924-06-12', datetime.datetime(2018, 11, 30, 0,
0))
(42, 'Clinton', "William Jefferson 'Bill'", '1946-08-19', 'NONE')
(43, 'Bush', 'George Walker', '1946-07-06', 'NONE')
(44, 'Obama', 'Barack Hussein', '1961-08-04', 'NONE')
(45, 'Trump', 'Donald J', '1946-06-14', 'NONE')
(46, 'Biden', 'Joseph Robinette', datetime.datetime(1942, 11, 10, 0, 0), 'NONE')
```

Working with ranges

- Range represents a rectangle of cells
- Use slice notation
- Iterate through rows, then columns

To get a range of cells, use slice notation on the worksheet object and standard cell notation, e.g. `WS['A1':'M9']` or `WS['A1:M9']`. Note that the range can consist of one string containing the range, or two strings separated by `:`.

The range is a virtual list of rows, and so can be iterated over. Each element of a row is a Cell object. Use the .value attribute to get or set the cell value.

Example

px_get_ranges.py

```
#!/usr/bin/env python
import openpyxl as px


def main():
    """program entry point"""
    wb = px.load_workbook('../DATA/presidents.xlsx')
    ws = wb['US Presidents']

    print_first_and_last_names(ws)


def print_first_and_last_names(ws):
    """Print first and last names of all presidents"""
    pres_range = ws['B2':'C47']  # cell range
    for row in pres_range:  # row object
        print(row[1].value, row[0].value)


if __name__ == '__main__':
    main()
```

px_get_ranges.py

```
George Washington
John Adams
Thomas Jefferson
James Madison
James Monroe
John Quincy Adams
Andrew Jackson
Martin Van Buren
William Henry Harrison
John Tyler
```

...

Modifying a worksheet

- Assign to cells
  - WS.cell(row=ROW, column=COLUMN).value = *value*
  - WS.cell(ROW, COLUMN).value = *value*
  - WS[*coordinate*] = *value*

To modify a worksheet, you can either iterate through rows and columns as described above, or assign to the `.value` attribute of individual cells using either `WS.cell()` or `WS["coordinates"]`.

Use `ws.append(iterable)` to append a new row to the spreadsheet.

Use `workbook.save('name.xlsx')` to save the changes. To save to the original workbook, use its name.

TIP Assigning to *cell* is a shortcut for assigning to *cell*.value(). That is, you can say `ws['B4'] = 10`.

NOTE See the later section on inserting and moving rows and columns.

Example

px_modify_sheet.py

```
#!/usr/bin/env python
from datetime import date
import openpyxl as px


def main():
    """program entry point"""
    wb = px.load_workbook('../DATA/presidents.xlsx')
    ws = wb['US Presidents']

    add_age_at_inauguration(ws)

    wb.save('presidents1.xlsx')  # save as ...


def make_date(date_str):
    """Convert date string returned by CELL.value into Python date object"""
    year, month, day = date_str.split('-')
    return date(int(year), int(month), int(day))


def add_age_at_inauguration(ws):
    """Add a new column with age of inauguration"""
    new_col = ws.max_column + 1
    print(new_col)
    ws.cell(row=1, column=new_col).value = 'Age at Inauguration'
    for row in range(2, 47):
        birth_date = make_date(ws.cell(row=row, column=4).value)  # treat date as string
        inaugural_date = make_date(ws.cell(row=row, column=8).value)
        raw_age_took_office = inaugural_date - birth_date
        age_took_office = raw_age_took_office.days / 365.25
        ws.cell(row=row, column=new_col).value = age_took_office


if __name__ == '__main__':
    main()
```

Working with formulas

- Assign to cell value as a string
- Be sure to start with '='

To add or update a formula, assign the formula as a string to the cell value.

NOTE Remember that **openpyxl** can not *recalculate* a worksheet.

Example

px_formulas.py

```
#!/usr/bin/env python
import openpyxl as px

def main():
    """program entry point"""
    wb = px.load_workbook('../DATA/presidents.xlsx')
    ws = wb['US Presidents']

    add_age_at_inauguration(ws)

    wb.save('presidents_formula.xlsx')

def add_age_at_inauguration(ws):
    """Add a new column with age of inauguration"""
    new_col = ws.max_column + 1
    print(new_col)
    ws.cell(row=1, column=new_col).value = 'Age at Inauguration'
    for row in range(2, 47):
        new_cell = ws.cell(row=row, column=new_col)
        new_cell.value = '=(H{0}-D{0})/365.25'.format(row)
        new_cell.number_format = '0.0'


if __name__ == '__main__':
    main()
```

Creating a new spreadsheet

- Use the Workbook() function
- One worksheet created by default
- Add worksheets with WB.create_sheet(n)
- Copy worksheets with WB.copy_sheet(n)
- Add data rows with WS.append(iterable)

To create a new spreadsheet file, use the Workbook() function. It creates a new workbook, with a default worksheet named "Sheet1".

Add worksheets with `WB.create_sheet(n)`. The parameter indicates where to insert the new worksheet; if not specified, it is appended.

To get or set the name of the worksheet, use its `.title` property.

To easily add rows to the worksheet, use `WS.append(iterable)`, where *iterable* is an iterable of column values.

Example

px_create_worksheet.py

```
#!/usr/bin/env python
import openpyxl as px

fruits = [
    "pomegranate", "cherry", "apricot", "date", "apple", "lemon",
    "kiwi", "orange", "lime", "watermelon", "guava", "papaya",
    "fig", "pear", "banana", "tamarind", "persimmon", "elderberry",
    "peach", "blueberry", "lychee", "grape"
]

wb = px.Workbook()

ws = wb.active

ws.title = 'fruits'

ws.append(['Fruit', 'Length'])

for fruit in fruits:
    ws.append([fruit, len(fruit)])

# hard way
# for i, fruit in enumerate(fruits, 1):
#     ws.cell(row=i, column=1).value = fruit
#     ws.cell(row=i, column=2).value = len(fruit)

wb.save('fruits.xlsx')
```

Inserting, Deleting, and moving cells

- Insert
  - `ws.insert_rows(row_index, num_rows=1)`
  - `ws.insert_cols(col_index, num_cols=1)`
- Delete
  - `ws.delete_rows(row_index, num_rows)`
  - `ws.delete_cols(col_index, num_cols)`
- Move
  - `ws.move_range(range, rows=row_delta, cols=col_delta)`
- Append
  - `ws.append(iterable)`

To insert one or more blank rows or columns, use `ws.insert_rows()` or `ws.insert_cols()`. The first argument is the positional index of the row or column (1-based), and the second argument is how many columns to insert.

To delete rows or columns, use `ws.delete_rows()` or `ws.delete_cols()`. The first argument is the index of the first row or column to delete; the second is the number of rows or columns.

To move a range of rows and columns, use `ws.move_range()`. The first argument is a range string such as `A1:F10`. Add named arguments `rows` and `cols` to specify how many cells to move. Positive values move down or right, and negative values move up or left. Existing data will be overwritten at the new location of the moved cells.

To append a row of data to a worksheet, pass an iterable to `ws.append()`.

Example

px_insert_delete_move.py

```
#!/usr/bin/env python
import openpyxl as px

RAW_DATA = [47, "Mouse", "Mickey", None, None, "Anaheim", "California", "2025-01-20",
None, "Imagineer"]

def main():
    """program entry point"""
    wb = px.load_workbook('../DATA/presidents.xlsx')
    ws = wb['US Presidents']

    insert_cells(ws)
    delete_cells(ws)
    move_cells(ws)
    append_cells(ws)

    print(ws.dimensions)

    wb.save('presidents_insert_delete_move.xlsx')

def append_cells(ws):
    ws.append(RAW_DATA)

def insert_cells(ws):
    ws.insert_rows(1, 3)  # insert three rows at top
    ws.insert_cols(5)  # insert one col at position 5

def delete_cells(ws):
    ws.delete_rows(15,5)
    ws.delete_cols(6)

def move_cells(ws):
    ws.move_range('A43:K45', rows=6, cols=3)


if __name__ == '__main__':
    main()
```

Hiding and freezing columns and sheets

- Hide
  - `ws.column_dimensions[column]`
  - `ws.column_dimensions.group(column, ⋯)`
- Freeze
  - `ws.freeze_panes = 'coordinate'`
- Hide entire sheet
  - `ws.sheet_state = 'hidden'`

You can hide a column by using the `.column_dimensions` property of a worksheet. Specify a column letter inside square brackets, and assign `True` to the `.hidden` property. To hide multiple columns, use `.column_dimensions.group()`. Specify start and end columns, and set the `hidden` argument to `True`

```
ws.column_dimensions['M'].hidden = True
ws.column_dimensions.group(start="C", end="F", hidden=True)
```

To freeze rows and columns for scrolling, assign the coordinates of the first row and column that you want to scroll to `ws.freeze_panes`. For example, to freeze the first 3 columns and start scrolling with column 'D', use

```
ws.freeze_panes = 'A4'
```

You can also hide a worksheet by setting `ws.sheet_state` to `"hidden"`.

Example

px_hide_freeze.py

```
#!/usr/bin/env python
import openpyxl as px


def main():
    """program entry point"""
    wb = px.load_workbook('../DATA/presidents.xlsx')
    ws = wb['US Presidents']

    hide_columns(ws)

    wb.save('presidents_hidden.xlsx')

    wb = px.load_workbook('../DATA/presidents.xlsx')
    ws = wb['US Presidents']

    freeze_columns(ws)

    create_hidden_sheet(wb)

    wb.save('presidents_frozen.xlsx')

    wb = px.load_workbook('../DATA/presidents.xlsx')

    create_hidden_sheet(wb)

    wb.save('presidents_hidden_sheet.xlsx')


def hide_columns(ws):
    """Hide single columnn and multiple columns"""

    # hide birthplace column
    ws.column_dimensions['F'].hidden = True

    # hide inauguration columns
    ws.column_dimensions.group(start='H', end='I', hidden=True)

def freeze_columns(ws):
    """Freeze the first 2 columns"""
    ws.freeze_panes = "C1"

def create_hidden_sheet(wb):
    """Add a hidden worksheet to the workbook"""
```

```
    ws = wb.create_sheet(title="secret plans")
    ws.sheet_state = "hidden"


if __name__ == '__main__':
    main()
```

Setting Styles

- Must be set on each cell individually
- Cannot change, once assigned (but can be replaced)
- Copy style to make changes

Each cell has a group of attributes that control its styles and formatting. Most of these have a corresponding class; to change styles, create an instance of the appropriate class and assign it to the attribute.

You can also make a copy of an existing style object, and just change the attributes you need.

Example

px_styles.py

```
#!/usr/bin/env python
import openpyxl as px

def main():
    """program entry point"""
    wb = px.load_workbook('../DATA/presidents.xlsx')
    ws = wb['US Presidents']

    update_last_names(ws)

    wb.save('presidents_styles.xlsx')

def update_last_names(ws):
    """Make the last name column blue and bold"""
    for row in ws['B2:B47']:
        cell = row[0]
        cell.value = cell.value.upper()
        cell.font = px.styles.Font(color='FF0000FF')

if __name__ == '__main__':
    main()
```

Table 3. openpyxl Cell Style Attributes

Cell attribute	Class	Parameters	Default value
font	Font		
		name	'Calibri'
		size	11
		bold	False
		italic	False
		vertAlign	None
		underline	'none'
		strike	False
		color	'FF000000'
fill	PatternFill		
		fill_type	None
		start_color	'FFFFFFFF'
		end_color	'FF000000'
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	
		outline	Side(border_style=None, color='FF000000')
		vertical	Side(border_style=None, color='FF000000')
		horizontal	Side(border_style=None, color='FF000000')
alignment	Alignment		

Cell attribute	Class	Parameters	Default value
		horizontal	'general'
		vertical	'bottom'
		text_rotation	0
		wrap_text	False
		shrink_to_fit	False
		indent	0
number_format	None	N/A	'General'
protection	Protection		
		locked	True
		hidden	False

Conditional Formatting

- Apply styles per values
- Types
  - Builtin
  - Standard
  - Custom
- Components
  - Differential Style
  - Rule
    - Formula

Conditional formatting means applying styles to cells based on their values. In openpyxl, conditional formatting can be a little complicated.

There are three kinds of rules for conditional formatting:

- Builtin — predefined rules with predefined styles
- Standard — predefined rules with custom styles
- Custom — custom rules with custom styles

Because this is complicated, there are some convenience functions for generating some formats.

Components

Formatting requres *styles*, which are either builtin or configured via a `DifferentialStyle` object, and *rules*, which are embedded in the formats. For custom rules, you provide a *formula* that defines when the rule should be used.

Builtin formats

There are three conditional formats: `ColorScale`, `IconSet`, and `DataBar`. These formats contain various settings, which compare the value to ann integer using one of these types: `num`, `percent`, `max`, `min`, `formula`, or `percentile`.

ColorScale

`ColorScale` provides a rule for a gradient from one color to another for the values within a range. You can add a second `ColorScale` for two gradients.

The convenience function for `ColorScale` is `openpyxl.formatting.rule.ColorScaleRule()`.

IconSet

`IconSet` provides a rule for applying different icons to different values.

The convenience function for `IconSet` is `openpyxl.formatting.rule.IconSetRule()`.

DataBar

`DataBar` provides a rule for adding "data bars", similar to the bars used by mobile phones to indicate signal strength.

The convenience function for `DataBar` is `openpyxl.formatting.rule.DataBarRule()`.

Example

px_conditional_styles.py

```
#!/usr/bin/env python
import openpyxl as px
from openpyxl.formatting.rule import (
    Rule, ColorScale, FormatObject, IconSet, DataBar
)

from openpyxl.styles import Font, PatternFill, Color
from openpyxl.styles.differential import DifferentialStyle

CONDITIONAL_CONFIG = {
    'Republican': {
        'font_color': "FF0000",
        'fill': "FFC0CB",
    },
    'Democratic': {
        'font_color': "0000FF",
        'fill': "ADD8E6",
    },
    'Whig': {
        'font_color': "008000",
        'fill': "98FB98",
    }
}

def main():
    """program entry point"""
    wb = px.load_workbook('../DATA/presidents.xlsx')
    ws = wb['US Presidents']

    colorscale_values(ws)

    color_potus_parties(ws)

    icon_values(ws)

    wb.save('presidents_conditional.xlsx')

    wb = px.load_workbook('../DATA/columns_of_numbers.xlsx')
    icon_values(wb.active)
    databar_values(wb.active)
    wb.save('columns_with_icons.xlsx')

def colorscale_values(ws):
    """
```

```
    Add conditional style to the "TERM" column using a builtin
    type.

    :param ws: the worksheet
    :return: None
    """
    first = FormatObject(type="min")
    last = FormatObject(type="max")
    colors = [Color('AA0000'), Color('00AA00')]
    cs2 = ColorScale(cfvo=[first, last], color=colors)
    rule = Rule(type='colorScale', colorScale=cs2)

    last_row = ws.max_row
    ws.conditional_formatting.add(f'A2:A{last_row}', rule)



def color_potus_parties(ws):
    """
        Make Republicans red and Democrats blue, etc.

        This is a custom rule with a custom formula.

        :param ws: Worksheet to format
        :returns: None
    """
    for text, config in CONDITIONAL_CONFIG.items():
        font = Font(color=config['font_color'])
        fill = PatternFill(bgColor=config['fill'])
        dxf = DifferentialStyle(font=font, fill=fill)

        # make a rule for this condition
        rule = Rule(type="expression", dxf=dxf)

        # add an Excel formula to the rule. Cell must be first cell of
        # range; otherwise formatting is offset by difference from first
        # cell to specified cell
        #
        # can use any Excel text operations here
        rule.formula=[f'EXACT("{text}",$J2)']

        # add the rule to desired range
        ws.conditional_formatting.add('J2:J47', rule)

def icon_values(ws):
    """
    Add icons for numeric values in column.
```

```
    :param ws: worksheet to format
    :return: None
    """
    thresholds = [0, 33, 67]
    icons = [FormatObject(type='percent', val=t) for t in thresholds]
    iconset = IconSet(iconSet='3TrafficLights1', cfvo=icons)
    rule = Rule(type='iconSet', iconSet=iconset)
    format_range = f"A2:A{ws.max_row}"
    ws.conditional_formatting.add(format_range, rule)

def databar_values(ws):
    """
    Add conditional databars to worksheet.

    :param ws: worksheet to format
    :return: None
    """
    first = FormatObject(type='min')
    second = FormatObject(type='max')
    data_bar = DataBar(cfvo=[first, second], color="638EC6")
    rule = Rule(type='dataBar', dataBar = data_bar)
    format_range = f"F2:F{ws.max_row}"
    ws.column_dimensions['F'].width = 25  # make column wider for data bar
    ws.conditional_formatting.add(format_range, rule)


if __name__ == '__main__':
    main()
```

Chapter 1 Exercises

Exercise 1-1 (age_of_geeks.py,age_of_geeks_formula.py)

Write a script to compute the average age of the people on the worksheet 'people' in **computer_people.xlsx**. First, you'll have to calculate the age from the birth date. (Some of the people in the worksheet have died. Just include them for purposes of this exercise).

TIP TO calculate the age, get today's date (`datetime.datetime.now()`), subtract the DOB cell value from today's date. This gets a **timedelta** object. Use the **days** attribute of the timedelta divided by 365 to get the age.

NOTE Another way to do this lab (if Excel is available) is to use this Excel formula: `=DATEDIF(DOB,TODAY(),"y")` where DOB is the cell containing the birthdate, such as "D2". Add an additional column. Then add a formula to average the values in this new column. Since OpenPyXL can't recalculate a sheet, open the sheet in Excel to the the results. (You could also use Libre Office or Open Office to open the workbook).

Print out the average.

Exercise 1-2 (knights_to_spreadsheet.py, knights_to_spreadsheet_extra.py

Write a script to create a new spreadsheet with data from the knights.txt file. The first row of the spreadsheet should have the column headings:

Name, Title, Favorite Color, Quest, Comment

The data should start after that.

Save the workbook as knights.xlsx.

NOTE For extra fun, make the headers bold, the name fields red, and the comments in italics.

Index