Import modules

In [1]:
```
import threading
import time
import xlwings as xw
```

Search Term Worksheet

In [2]:
```
class ExcelWorksheet:
    def __init__(self, \
                 ws_index=0, \
                 title_font="Lato", \
                 content_font="Source Sans Pro", \
                 large_font_size = 18, \
                 medium_font_size = 14, \
                 small_font_size = 11):

        # Connect to excel
        self.xlApp = xw.apps.active
        if self.xlApp is None: raise Exception("Excel must be open to
continue!")

        # Test workbook connection
        self.connection_tested = False
        t1 = threading.Thread(target=self.connection_timer)
        t2 = threading.Thread(target=self.test_connection)
        t1.start(), t2.start()
        t1.join(), t2.join()

        # Connect to workbook
        if self.xlApp is None: raise Exception("Excels' data entry mode
successfully deactivated.  Please rerun code.")
        self.wb = xw.books.active

        # Select worksheet and set range variables
        self.ws = self.worksheet_connection(ws_index)
        self.titlebar = self.ws.range("B2:F4")
        self.titlebar_data = self.ws.range("D3")
        self.headings_data = self.ws.range("B5:F5")
        self.table_data = self.ws.range("B6:F100")
```

```
        # Font types and font sizes
        self.title_font =  title_font
        self.content_font = content_font
        self.large_font_size = large_font_size
        self.medium_font_size = medium_font_size
        self.small_font_size = small_font_size


    def connection_timer(self):
        # Throws an error if test connection hangs
        time.sleep(0.1)
        if not self.connection_tested:
            self.xlApp = None
            raise Exception("Excel is in data entry mode! Go to excel and
press the 'esc' key to deactivate.")


    def test_connection(self):
        # Try connecting to the active workbook
        wb = xw.books.active
        wb = None
        self.connection_tested = True


    def worksheet_connection(self, ws_index):
        # Select requested worksheet or maek a new worksheet
        try:
            self.ws = self.wb.sheets[ws_index]
        except:
            self.ws = self.wb.sheets.add(name=None, before=self.wb.sheets[0])

        # If selected worksheet has existing data make a new worksheet
        data_range = self.ws.used_range.address.replace("$", "").split(":")
        if len(data_range) > 1: self.ws = self.wb.sheets.add(name=None,
before=self.wb.sheets[0])
        try:
            self.ws.name = "Search Terms"
        except Exception:
            print("Can't name worksheet 'Search Terms' as name alreday in
use.")
        finally:
            return self.ws
```

```
    def column_headings(self, headings = ["Search Terms 1", "Search Terms 2",
"Search Terms 3"], widths = [39.43, 40.43, 36.86]):
        # Insert blank-named spacing columns
        for idx in range(len(headings), -1, -1):
            headings.insert(idx, "")
            widths.insert(idx, 0.5)

        # Add column headings / widths to worksheet
        for idx in range(0, len(headings)):
            col_indx = idx +1
            self.ws.range(5, col_indx).value = headings[idx]
            self.ws.range(5, col_indx).column_width = widths[idx]


    def format_worksheet(self):
        # Add text
        self.titlebar_data.value = "Search Terms"

        # Font types
        self.titlebar_data.font.name = self.title_font
        self.headings_data.font.name = self.title_font
        self.table_data.font.name = self.content_font

        # Font sizes
        self.titlebar_data.font.size = self.large_font_size
        self.headings_data.font.size = self.medium_font_size
        self.table_data.font.size = self.small_font_size

        # Font (bold, italic, wrap)
        self.titlebar_data.font.bold = True
        self.headings_data.font.bold = True
        self.table_data.wrap_text = True

        # Font and background colour
        self.titlebar.api.Interior.Color = 2836736 # leeds_green
        self.titlebar_data.font.color = 16777215 # full_white

        # Horizontal alignment
        self.titlebar_data.api.HorizontalAlignment = -4108  # xlHAlignCenter
        self.headings_data.api.HorizontalAlignment = -4108  # xlHAlignCenter
        self.table_data.api.HorizontalAlignment = -4108  # xlHAlignCenter

        # Vertical alignment
```

```
        self.titlebar_data.api.VerticalAlignment = -4108 # xlVAlignCenter
        self.headings_data.api.VerticalAlignment = -4108 # xlVAlignCenter
        self.table_data.api.VerticalAlignment = -4108 # xlVAlignCenter

        # Row heights
        self.ws.range("1:1").row_height = 5
        self.ws.range("2:4").row_height = 21
        self.headings_data.row_height = 24
        self.table_data.row_height = 18

        # Feeze titlebar and headings rows
        self.ws.range("6:6").select()
        self.xlApp.api.ActiveWindow.FreezePanes = True
        self.ws.range("B6").select()


    def make_sheet(self):
        self.column_headings()
        self.format_worksheet()


def main():
    excel_worksheet = ExcelWorksheet()
    excel_worksheet.make_sheet()

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

Search Strings

In [4]:

```
class TransferData:
    def __init__(self, \
                 target_cells = ("B6", "D6", "F6"), \
                 column_A_width= 0.5, \
                 row_1_height = 5, \
                 title_font="Lato", \
                 content_font="Source Sans Pro", \
                 large_font_size = 18, \
                 medium_font_size = 14, \
                 small_font_size = 11):

        # Connect to excel
        self.xlApp = xw.apps.active
```

```
        if self.xlApp is None: raise Exception("Excel must be open to
continue!")

        # Test workbook connection
        self.connection_tested = False
        t1 = threading.Thread(target=self.connection_timer)
        t2 = threading.Thread(target=self.test_connection)
        t1.start(), t2.start()
        t1.join(), t2.join()

        # Connect to workbook
        if self.xlApp is None: raise Exception("Excels' data entry mode
successfully deactivated.  Please rerun code.")
        self.wb = xw.books.active

        # Set cell ranges
        self.cell_1 = target_cells[0]
        self.cell_2 = target_cells[1]
        self.cell_3 = target_cells[2]

        # Set width/height first column/row
        self.column_A_width = column_A_width
        self.row_1_height = row_1_height

        # Set font types and font sizes
        self.title_font =  title_font
        self.content_font = content_font
        self.large_font_size = large_font_size
        self.medium_font_size = medium_font_size
        self.small_font_size = small_font_size


    def connection_timer(self):
        # Throws an error if test connection hangs
        time.sleep(0.1)
        if not self.connection_tested:
            self.xlApp = None
            raise Exception("Excel is in data entry mode! Go to excel and
press the 'esc' key to deactivate.")


    def test_connection(self):
        # Try connecting to the active workbook
        wb = xw.books.active
```

```
        wb = None
        self.connection_tested = True


    def import_data(self):
        try:
            self.ws = self.wb.sheets["Search Terms"]
        except:
            raise Exception("A 'Search Terms' worksheet could not be found")
        else:
            self.search_terms_1 =
self.ws.range(self.cell_1).current_region.value[1:]
            self.search_terms_2 =
self.ws.range(self.cell_2).current_region.value[1:]
            self.search_terms_3 =
self.ws.range(self.cell_3).current_region.value[1:]


    def export_data(self, ws_title, ws_index, complete_search_string,
counter):
        try:
            self.wb.sheets[ws_index].select()
        except:
            self.wb.sheets.add(name=ws_index, after=self.wb.sheets.count)
        finally:
            self.ws = self.wb.sheets.active

        if counter == 0:
            # First row and column
            self.ws.range("A:A").column_width = self.column_A_width
            self.ws.range("1:1").row_height = self.row_1_height

            # Title bar
            self.ws.range("B2:B4").api.Interior.Color = 2836736 # leeds_green
            self.ws.range("2:4").row_height = 21

            # Body text format
            self.ws.range("B:B").font.name = self.content_font
            self.ws.range("B:B").font.size = self.small_font_size = 11
            self.ws.range("B:B").wrap_text = True
            self.ws.range("B:B").api.VerticalAlignment = -4160 #(xlVAlignTop)
            self.ws.range("B:B").column_width = 150

            # Heading text format
```

```
            self.ws.range("B3").font.name = self.title_font
            self.ws.range("B3").font.size = self.large_font_size
            self.ws.range("B3").font.bold = True
            self.ws.range("B3").font.color = 255,255,255 # full_white

            # Title string
            self.ws.range("B3").value = ws_title

        # Add search string or strings
        row_idx = str(counter + 5)
        self.ws.range(f"{row_idx}:{row_idx}").row_height = 120
        self.ws.range(f"B{row_idx}").value = complete_search_string

        # Final cell selection
        self.ws.range("B7").select()


class SearchString:
    def __init__(self):
        # Import search terms from excel using TrasferData class
        self.transfer_data = TransferData()
        self.transfer_data.import_data()

        # Create basic search strings
        if self.transfer_data.search_terms_1[0] is None: raise
Exception("Search Terms 1 is empty!")
        if self.transfer_data.search_terms_2[0] is None: raise
Exception("Search Terms 2 is empty!")
        if self.transfer_data.search_terms_3[0] is None: raise
Exception("Search Terms 3 is empty!")

        self.string_1 = " or ".join([f'"{term}"' for term in
self.transfer_data.search_terms_1])
        self.string_2 = " or ".join([f'"{term}"' for term in
self.transfer_data.search_terms_2])
        self.string_3 = " or ".join([f'"{term}"' for term in
self.transfer_data.search_terms_3])


    def cinhal(self, search_type=["TI", "AB"]):
        counter = 0
        while search_type:
            search_string_1 = (f"{search_type[0]}({self.string_1})")
            search_string_2 = (f"({self.string_2})")
```

```
            search_string_3 = (f"({self.string_3})")
            complete_search_string = " AND ".join([search_string_1,
search_string_2, search_string_3])
            self.transfer_data.export_data("Cinhal String", "Cinhal",
complete_search_string, counter)
            del search_type[0]
            counter += 1


    def medline_and_embase(self, search_type = [".m_titl.", ".ab",
".ti,ab."]):
        counter = 0
        while search_type:
            search_string_1 = (f"({self.string_1}){search_type[0]} ")
            search_string_2 = (f"({self.string_2}){search_type[0]} ")
            search_string_3 = (f"({self.string_3}){search_type[0]} ")
            complete_search_string = " AND ".join([search_string_1,
search_string_2, search_string_3])
            self.transfer_data.export_data("Medline and Embase String",
"Medline & Embase", complete_search_string, counter)
            del search_type[0]
            counter += 1


    def scopus(self):
        search_string_1 = (f"((({self.string_1})")
        search_string_2 = (f"({self.string_2})")
        search_string_3 = (f"({self.string_3})))")
        complete_search_string = " AND ".join([search_string_1,
search_string_2, search_string_3])
        self.transfer_data.export_data("Scopus String", "Scopus",
complete_search_string, 0)


    def web_of_science(self):
        search_string_1 = (f"({self.string_1})")
        search_string_2 = (f"({self.string_2})")
        search_string_3 = (f"({self.string_3})")
        complete_search_string = " AND ".join([search_string_1,
search_string_2, search_string_3])
        self.transfer_data.export_data("Web of Science String", "Web
Science", complete_search_string, 0)
```

```
    def make_search_strings(self):
        self.cinhal()
        self.medline_and_embase()
        self.scopus()
        self.web_of_science()

def main():
    search_string = SearchString()
    search_string.make_search_strings()

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