__init__.py

```
package_name = "EnneadTab"
version = "2.1"

import os
import traceback

import os
for module in os.listdir(os.path.dirname(__file__)):
    #print (module)
    if module == '__init__.py':
        continue
    if module in ["RHINO", "REVIT"]:
        __import__(module, locals(), globals())
        continue
    if module[-3:] != '.py':
        continue
    try:
        __import__(module[:-3], locals(), globals())
    except Exception as e:
        # to-do: add try because Rhino 8 traceback is not working peoperly. This
should be recheck in future Rhino 8.
        try:
            print ("Cannot import {} becasue\n\n{}".format(module,
traceback.format_exc()))
        except:
            print ("Cannot import {} becasue\n\n{}".format(module, e))
del module# delete this varible becaue it is refering to last item on the for
loop
```

ENVIRONMENT.py

```
    for i, x in enumerate(sorted(globals())):
        content = globals()[x]

        if inspect.ismodule(content):
            continue

        if not x.startswith("_") and not callable(content):
            print(x, " = ", content)

            if isinstance(content, bool):
                continue

            if not isinstance(content, list):
                content = [content]

            for item in content:
                if "\\" in item:
                    is_ok = os.path.exists(item) or os.path.isdir(item)

                    if not is_ok:
                        print("!!!!!!!!!!!!! not ok: " + item)
                    # assert is_ok


IS_AVD = is_avd()
IS_RHINO_ENVIRONMENT = is_Rhino_environment()
IS_RHINO_7 = is_Rhino_7()
IS_RHINO_8 = is_Rhino_8()
IS_GRASSHOPPER_ENVIRONMENT = is_Grasshopper_environment()
IS_REVIT_ENVIRONMENT = is_Revit_environment()
```

```
IS_RHINOINSIDEREVIT_ENVIRONMENT = is_RhinoInsideRevit_environment()


def get_app_name():
    """Get the current application name.

    Returns:
        str: The current application name.
    """
    app_name = "terminal"
    if IS_REVIT_ENVIRONMENT:
        app_name = "revit"
    elif IS_RHINO_ENVIRONMENT:
        app_name = "rhino"
    return app_name


################
if __name__ == "__main__":
    unit_test()
```

COLOR.py

```
            subject = raw_data[pointer].get("value")
            if subject in [""]:
                continue

            pointer_right = (i, j+1)
            subject_abbr = raw_data[pointer_right].get("value")

            # if no abbr(maybe due to merged cell), use subject name as abbr.
            subject_abbr = subject if subject_abbr == "" else subject_abbr

            temp_data[subject] = {"abbr": subject_abbr, "color": subject_color}

    return temp_data

def get_color_template_data(template = None):
    """Get color template data from department standards.

    Args:
        template (str, optional): The template path. Defaults to None.

    Returns:
        dict: The resulting color data.
    """
    if template:
        safe_template = FOLDER.copy_file_to_local_dump_folder(template)
    else:
        safe_template = "OFFICE STANDARD FILE TO BE MADE"

    if safe_template.endswith(".sexyDuck"):
        with open(safe_template, "r") as f:
            return json.load(f)

    if safe_template.endswith(".xlsx"):
        NOTIFICATION.messenger(main_text="Please save as .xls instead of .xlsx")
        return {}


    if safe_template.endswith(".xls"):
        import EXCEL
        raw_data = EXCEL.read_data_from_excel(safe_template,
                                              worksheet = "HEALTHCARE",
                                              return_dict=True)
```

```
        #column A and D are 0, 3 for key column
        department_data = _gather_data(raw_data, key_column = 0)
        program_data = _gather_data(raw_data, key_column = 3)


        return {"department_color_map": department_data, "program_color_map":
program_data}
```

CONFIG.py

```
"""Get and set the global settings for EnneadTab."""


import os
import DATA_FILE


GLOBAL_SETTING_FILE = "setting_{}.sexyDuck".format(
    os.environ["USERPROFILE"].split("\\")[-1]
)


def get_setting(key, default_value=None):
    """If no key provided, will return the whole dict.
    Otherwise, return the default value of this key.

        key_default_value (tuple): (key, default value), a tuple of default
result, this is used to get the key of value looking for. If do not provide this
tuple, then return the raw while data"""

    data = DATA_FILE.get_data(GLOBAL_SETTING_FILE)
    return data.get(key, default_value)


def set_setting(key, value):
    """Set the key and value to the Revit UI setting.

    Args:
        key (str): The key of the setting.
        value (str): The value of the setting.
    """

    with DATA_FILE.update_data(GLOBAL_SETTING_FILE) as data:
        data[key] = value


# simply rename the addin file register file by
# add/remove .disabled at end of the .addin file
# note need to search for all valid version folder
def enable_revit_addin(addin):
    pass

    # reload pyrevit


def disable_revit_addin(addin):
    pass

    # reload pyrevit
```

CONTROL.py

```
pass
```

COPY.py

```
"""
The main purpose of this moudle to is to handle Rhino 8 situation.
Native shutil.copyfile will fail in some cases, so we use dotnet to copy the
file.

"""
try:
    import shutil
except:
    from System.IO import File



def copyfile(src, dst):
    try:
        # Attempt to copy the file using shutil.copyfile
        shutil.copyfile(src, dst)
    except Exception as e:
        copyfile_with_dotnet(src, dst)

def copyfile_with_dotnet(src, dst):
    try:
        File.Copy(src, dst, True)  # True to overwrite if exists
        return True
    except Exception as e:
        return False


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

DATA_CONVERSION.py

```
        return System.Collections.Generic.List[DB.CurveLoop](list)
    if type == DataType.Curve:
        return System.Collections.Generic.List[DB.Curve](list)
    if type == DataType.TableCellCombinedParameterData:
        return
System.Collections.Generic.List[DB.TableCellCombinedParameterData](list)

    if type == DataType.XYZ:
        pts = System.Collections.Generic.List[DB.XYZ]()
        for pt in list:
            pts.Add(pt)
        return pts

    if type == DataType.Double:
        values = System.Collections.Generic.List[System.Double]()
        for value in list:
            values.Add(value)
```

```
        return values

    return System.Collections.Generic.List[type](list)
    # print_note("Things are not right here...type = {}".format(type))

    return False


def compare_list(A, B):
    """Compare two lists and return the unique elements in each list and the
shared elements.

    Args:
        A (list): The first list.
        B (list): The second list.
    """
    unique_A = [x for x in A if x not in B]
    unique_B = [x for x in B if x not in A]
    shared = [x for x in A if x in B]


def unit_test():
    # print all the enumerations of DataType
    print("All DataType in class:")
    for i, type in enumerate(dir(DataType)):
        if type.startswith("__"):
            continue
        print("{}: {}".format(type, getattr(DataType, type)))
    pass


if __name__ == "__main__":
    unit_test()
    pass
```

DATA_FILE.py

```
        # temporarily hands control back to the caller, allowing them to modify
data.
        yield data
        # Once the block inside the with statement is complete,
        # control returns to the context manager, which writes the modified data
back to the file.

        set_data(data, file_name, is_local)

    except Exception as e:
        try:
            print(
                "An error occurred when updating
data:\n{}".format(traceback.format_exc())
            )
        except:
            print (e)


########################################


STICKY_FILE = "sticky.SexyDuck"


def get_sticky(sticky_name, default_value_if_no_sticky=None):
    """Get longterm sticky information.
    Args:
```

```
        sticky_name (str): The name of the sticky.
        default_value_if_no_sticky (any, optional): The default value to return
if the sticky does not exist. Defaults to None.

    Returns:
        any : get the value of the longterm sticky
    """

    data = get_data(STICKY_FILE)
    if sticky_name not in data.keys():
        set_sticky(sticky_name, default_value_if_no_sticky)
        return default_value_if_no_sticky
    return data[sticky_name]


def set_sticky(sticky_name, value_to_write):
    """Set a long term sticky. The long term sticky will not be cleared after
the application is closed.

    Args:
        sticky_name (str): The name of the sticky.
        value_to_write (any): The value to write
    """
    with update_data(STICKY_FILE) as data:
        data[sticky_name] = value_to_write
```

DOCUMENTATION.py

```
def show_tip_rhino():
    """Show a random tip for Rhino. Not implemented yet.
    """
    print("TO_DO: use tool lookup data")

def tip_of_day():
    """Show a random tip of the day.
    """
    if random.random() < 0.8:
        return
    if ENVIRONMENT.is_Revit_environment():
        if random.random() < 0.95:
            show_tip_revit()
        else:
            show_scott_tip()
    if ENVIRONMENT.is_Rhino_environment():
        show_tip_rhino()


def unit_test():
    # tip_of_day()
    pass


def print_documentation_book_for_review_revit():
    """Print all the tips in a book or webpage to check spelling and doc
updates."""
    show_tip_revit(is_random_single=False)

    OUTPUT.display_output_on_browser()

def show_floating_box_warning():
    """Show an informational message for floating a box window.
    """
    import NOTIFICATION
    NOTIFICATION.duck_pop(main_text="Click has no use for this button. Just hold
down on the arrow and drag to make the window floating.\nThis will always stay
```

```
on top even when changed to another tab.")


def get_floating_box_documentation():
    """Return an informational message for floating a box window.
    """
    return "Hold down on the arrow and drag to make the window floating. This
will always stay on top even when changed to another tab."




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

DUCK.py

```
"""the dancing call duck"""
import EXE


def quack ():
    EXE.try_open_app("EnneaDuck.exe")
```

EMAIL.py

```
                ),
            )

    if ENVIRONMENT.IS_RHINO_ENVIRONMENT:
        developer_emails = USER.get_rhino_developer_emails()

    if USER.IS_DEVELOPER:
        developer_emails = [USER.get_EA_email_address()]

    email(
        receiver_email_list=developer_emails,
        body=body,
        subject=subject_line,
        body_folder_link_list=None,
        body_image_link_list=None,
        attachment_list=None,
    )


def email_to_self(
    subject="EnneadTab Auto Email to Self",
    body=None,
    body_folder_link_list=None,
    body_image_link_list=None,
    attachment_list=None,
):
    """Send email to self.

    Args:
        subject (str, optional): Subject of the email. Defaults to "EnneadTab
Auto Email to Self".
```

```
        body (str, optional): Body of the email. Defaults to None.
        body_folder_link_list (list, optional): List of folder links to be
included in the email body. Defaults to None.
        body_image_link_list (list, optional): List of image links to be
included in the email body. Defaults to None.
        attachment_list (list, optional): List of file paths to be attached to
the email. Defaults to None
    """
    email(
        receiver_email_list=[USER.get_EA_email_address()],
        subject=subject,
        body=body,
        body_folder_link_list=body_folder_link_list,
        body_image_link_list=body_image_link_list,
        attachment_list=attachment_list,
    )


def unit_test():
    email_to_self(
        subject="Test Email for compiler",
        body="Happy Howdy. This is a quick email test to see if the base
communication still working",
    )
```

EMOJI.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
"""Get emojis from the emoji library."""

import io
import random
import DOCUMENTATION

def get_all_emojis():
    """Get all emojis from the emoji library.

    Returns:
        list: List of emojis.
    """
    with io.open(DOCUMENTATION.get_text_path_by_name('_emoji_text.txt'), "r",
encoding = "utf8") as f:
        lines = f.readlines()
    return [x.replace("\n", "") for x  in lines if x != "\n"]


def pick_emoji_text():
    """Pick an emoji text from the displayed list and copy it to the clipboard.
    """
    lines = get_all_emojis()
    from pyrevit import forms
    sel = forms.SelectFromList.show(lines, select_multiple = False, title = "Go
wild")
    if not sel:
        return

    forms.ask_for_string(default = sel,
                         prompt = 'Copy below text to anywhere, maybe SheetName
or Schedule',
                         title = 'pick_emoji_text')


def random_emoji():
    """Pick a random emoji.
```

```
    """
    lines = get_all_emojis()

    random.shuffle(lines)
    return lines[0].replace("\n", "")
```

ENCOURAGING.py

```
import textwrap


import NOTIFICATION
import DOCUMENTATION
import ENVIRONMENT
import CONFIG

def is_hate_encouraging():
    """Check if the user has enabled encouraging messages.

    Returns:
        bool: True if the user has enabled encouraging messages.
    """
    return not CONFIG.get_setting("radio_bt_popup_full", False)

def get_all_warming_quotes():
    """Get all encouraging quotes from the quote library.

    Returns:
        list: All encouraging quotes.
    """
    with io.open(DOCUMENTATION.get_text_path_by_name('_warming_quotes.txt'),
"r", encoding = "utf8") as f:
        lines = f.readlines()
    return [x.replace("\n", "") for x  in lines if x != "\n"]


def random_warming_quote():
    """Get a random encouraging quote from the quote library.

    Returns:
        str: A random encouraging quote
    """
    lines = get_all_warming_quotes()
    random.shuffle(lines)
    return lines[0].replace("\n", "")


def warming_quote():
    """Display a random encouraging quote.
    """
    quote = random_warming_quote()


    # Wrap this text.
    wrapper = textwrap.TextWrapper(width = 100)
    quote = wrapper.fill(text = quote)


    NOTIFICATION.messenger(main_text = quote, animation_stay_duration = 10)
```

ERROR_HANDLE.py

```
                subject_line = "EnneadTab Auto Error Log"
                if is_silent:
                    subject_line += "(Silent)"
                try:
                    EMAIL.email_error(error_time + error, func.__name__,
USER.USER_NAME, subject_line=subject_line)
                except Exception as e:
                    print_note("Cannot send email: {}".format(e))

                if not is_silent:

                    error += "\n\n######If you have EnneadTab UI window open,
just close the original EnneadTab window. Do no more action, otherwise the
program might crash.##########\n##########Not sure what to do? Msg Sen Zhang, you
have dicovered a important bug and we need to fix it ASAP!!!!!########"
                    error_file =
FOLDER.get_EA_dump_folder_file("error_general_log.txt")
                    try:
                        with open(error_file, "w") as f:
                            f.write(error)
                    except IOError as e:
                        print_note(e)

                    output = OUTPUT.get_output()
                    output.write(error_time, OUTPUT.Style.SubTitle)
                    output.write(error)
                    output.insert_division()
                    output.plot()

                if ENVIRONMENT.IS_REVIT_ENVIRONMENT and not is_silent:
                    NOTIFICATION.messenger(
                        main_text="!Critical Warning, close all Revit UI window
from EnneadTab and reach to Sen Zhang.")

        error_wrapper.original_function = func
        return error_wrapper
    return decorator



def print_note(string):
    """For non-developers this is never printed."""

    if USER.is_EnneadTab_developer():

        try:
            from pyrevit import script
            string = str(string)
            script.get_output().print_md(
                "***[Dev Debug Only Note]***:{}".format(string))
        except Exception as e:

            print("[Dev Debug Only Note]:{}".format(string))
```

EXCEL.py

```
    },
    "add_two": {
        "1, 2": 3,
        "3, 4": 7,
        "5, 6": 11,
```

```
    },
    "add_to_list": {
        "[1, 2, 3]": [2, 3, 4],
        "[4, 5, 6]": [5, 6, 7],
    },
    "flip_dict": {
        "{'a': 1, 'b': 2}": {1: 'a', 2: 'b'},
        "{'x': 3, 'y': 4}": {3: 'x', 4: 'y'},
    },
    "num_and_letter": {
        "1, 'A'": 1,
        "2, 'B'": 3,
        "3, 'C'": 5,
    },
}


# Old unit test function
# def unit_test():
#     return
#     import xlrd
#     import WEB

#     # Replace this with your SharePoint URL
#     sharepoint_url = "https://enneadarch-
my.sharepoint.com/:x:/g/personal/scott_mackenzie_ennead_com/Eey-
gTYaVIdGuU9Jg65gig8BIUBmc32Aie-0nNsjVSgUfQ?rtime=4PY2woUX3Eg"

#     # Open the Excel file from the URL
#     str = WEB.get_request(sharepoint_url)
#     print(str)
#     workbook = xlrd.open_workbook(file_contents=str)

#     # Select the first sheet (you can change the sheet index as needed)
#     sheet = workbook.sheet_by_index(0)

#     # Iterate through rows and print each row
#     for row_num in range(sheet.nrows):
#         row = sheet.row_values(row_num)
#         print(row)


##################  MAIN  #################

if __name__ == "__main__":
    filename = __file__
    UNIT_TEST.pretty_test(test_dict, filename)
```

EXE.py

```
    exe_name = exe_name.replace(".exe", "")
    exe = ENVIRONMENT.EXE_PRODUCT_FOLDER + "\\{}.exe".format(exe_name)


    def get_ignore_age(file):
        if "OS_Installer" in file or "AutoStartup" in file:
            return 60*5
        return 60*60*24
    if safe_open:
        if not os.path.exists(exe):
            raise Exception("Only work for stanfle along exe, not for foldered
exe.[{}] not exist".format(exe))
        temp_exe_name = "_temp_exe_{}_{}.exe".format(exe_name, int(time.time()))
        temp_exe = FOLDER.DUMP_FOLDER + "\\" + temp_exe_name
        # print (temp_exe)
```

```
        COPY.copyfile(exe, temp_exe)
        os.startfile(temp_exe)
        for file in os.listdir(FOLDER.DUMP_FOLDER):
            if file.startswith("_temp_exe_"):
                # ignore if this temp file is less than 1 day old, unless it is
OS_installer or AutoStartup
                if time.time() -
os.path.getmtime(os.path.join(FOLDER.DUMP_FOLDER, file)) < get_ignore_age(file):
                    continue
                try:
                    os.remove(os.path.join(FOLDER.DUMP_FOLDER, file))
                except:
                    pass
        return True



    if os.path.exists(exe):
        os.startfile(exe)
        return True
    foldered_exe = ENVIRONMENT.EXE_PRODUCT_FOLDER +
"\\{0}\\{0}.exe".format(exe_name)
    if os.path.exists(foldered_exe):
        os.startfile(foldered_exe)
        return True

    if legacy_name:
        if try_open_app(legacy_name):
            return True

    if USER.IS_DEVELOPER:
        print ("[Developer only log]No exe found in the location.")
        print (exe)
        print (foldered_exe)
        NOTIFICATION.messenger("No exe found!!!\n{}".format(exe_name))
    return False
```

FOLDER.py

```
                print(
                    "Cannot delete file [{}] becasue error:
{}".format(current_file, e)
                )
    return count


def secure_filename_in_folder(output_folder, desired_name, extension):
    """Ensure proper formatting of file name in output folder.
    Commonly used with Revit jpg exports, as Revit will change the file names.

    Args:
        output_folder (str): Folder to search.
        desired_name (str): The desired name of the file. Will use this name in
search pattern. Do not include extension!
        extension (str): File extension to lock search to. Include DOT! (e.g.
".jpg")
    """

    try:
        os.remove(os.path.join(output_folder, desired_name + extension))
    except:
        pass

    # print keyword
    keyword = " - Sheet - "
```

```
    for file_name in os.listdir(output_folder):
        if desired_name in file_name and extension in file_name.lower():
            new_name = desired_name

            # this prefix allow longer path limit
            old_path = "\\\\?\\{}\\{}".format(output_folder, file_name)
            new_path = "\\\\?\\{}\\{}".format(output_folder, new_name +
extension)
            try:
                os.rename(old_path, new_path)

            except:
                try:
                    os.rename(
                        os.path.join(output_folder, file_name),
                        os.path.join(output_folder, new_name + extension),
                    )

                except Exception as e:
                    print(
                        "filename clean up failed: skip {} becasue: {}".format(
                            file_name, e
                        )
                    )

if __name__ == "__main__":
    pass
```

GUI.py

```
"""Manipulate the GUI of apps by searching for image patterns."""

import EXE
import DATA_FILE

def simulate_click_on_image(image):
    """Add search json to find this image on screen and try to click on it."""

    with DATA_FILE.update_data("auto_click_data.sexyDuck") as data:
        if "ref_images" not in data:
            data["ref_images"] = []
        data["ref_images"].append(image)

    EXE.try_open_app("AutoClicker.exe")
```

HOLIDAY.py

```
def greeting_mid_moon():
    d0 = datetime.datetime(2023, 9, 28)
    today = datetime.datetime.now()
    d1 = datetime.datetime(2023, 10, 10)

    if not (d0 < today < d1):
        return

    image = "mid moon.jpg"
    image_file = __file__.split("ENNEAD.extension")[
        0
```

```
    ] + "ENNEAD.extension\\bin\{}".format(image)

    image = "moon-cake-drawing.png"
    moon_cake_image_file = __file__.split("ENNEAD.extension")[
        0
    ] + "ENNEAD.extension\\bin\{}".format(image)

    # print image_file
    output = script.get_output()
    output.print_image(image_file)
    output.set_width(1200)
    output.set_height(900)
    output.self_destruct(100)
    output.center()
    output.set_title("Greeting from EnneadTab")
    output.print_md("# Happy Mid-Autumn Festival, everybody!")
    output.print_md(
        "## Also known as the Moon-Festival, it is a family reunion holiday
shared in many east asian culture."
    )
    output.print_md(
        "## An important part is the moon-cake. You may find the technical
drawing below."
    )
    output.print_image(moon_cake_image_file)

    file = "sound effect_chinese_new_year.wav"
    SOUND.play_sound(file)

    if random.random() > 0.2:
        return
    dest_file = FOLDER.get_EA_dump_folder_file("Moon Festival.html")
    output.save_contents(dest_file)
    output.close()
    os.startfile(dest_file)


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

IMAGE.py

```
    files = [
        os.path.join(ENVIRONMENT.IMAGE_FOLDER, f)
        for f in os.listdir(ENVIRONMENT.IMAGE_FOLDER)
        if f.startswith(prefix)
    ]
    file = random.choice(files)
    return file


def average_RGB(R, G, B):
    """Average the RGB values of a pixel to simplify it to greyscale.

    Args:
        R (int): Red. 0-255.
        G (int): Blue. 0-255.
        B (int): Green. 0-255.

    Returns:
        int: Average of the RGB values.
    """
    return (R + G + B) / 3
def convert_image_to_greyscale(original_image_path, new_image_path=None):
    """Convert an image to greyscale.
```

```
    Args:
        original_image_path (str): The full path to the image to convert.
        new_image_path (str): The full path to save the new image. If None, the
original image will be overwritten. Careful: defaults to None!
    """
    if new_image_path is None:
        new_image_path = original_image_path
    image = SD.Image.FromFile(original_image_path)
    for x in range(image.Width):
        for y in range(image.Height):
            pixel_color = image.GetPixel(x, y)
            R = pixel_color.R
            G = pixel_color.G
            B = pixel_color.B
            A = pixel_color.A
            new_color = SD.Color.FromArgb(
                A, average_RGB(R, G, B), average_RGB(R, G, B), average_RGB(R, G,
B)
            )
            image.SetPixel(x, y, new_color)
    image.Save(new_image_path)
    return image


if __name__ == "__main__":
    pass
```

JOKE.py

```
if random.random() < chance:
    prank_dvd()


if random.random() < chance:
    DUCK.quack()


def april_fool():
    return

    y, m, d = TIME.get_date_as_tuple(return_string=False)

    marker_file = FOLDER.get_EA_dump_folder_file("2024_april_fooled3.stupid")

    if m == 4 and d in [1, 2] and random.random() < 0.2 :


        if os.path.exists(marker_file):
            return
        dice = random.random()
        if dice < 0.2:
            prank_ph()
        # elif dice < 0.45:
        #     NOTIFICATION.duck_pop(random_joke())
        # elif dice < 0.48:
        #     NOTIFICATION.messenger(random_loading_message())
        elif dice < 0.95:
            max = 10 if USER.USER_NAME in TARGETS else 5
            for _ in range(random.randint(3, max)):
                prank_dvd()

        else:
            SOUND.play_meme_sound()
        with open(marker_file, 'w') as f:
            f.write("You have been pranked.")
```

```
        # FUN.EnneaDuck.quack()


april_fool()

if __name__ == "__main__":

    prank_dvd()
```

KEYBOARD.py

```
pass
```

LEADER_BOARD.py

```
@FOLDER.backup_data(GLOBAL_SETTING_FILE , "setting")
def update_account(event_key):
    event_data = PRICE.get(event_key)
    if not event_data:
        raise "Cannot find event key {}".format(event_key)
    with DATA_FILE.update_data(GLOBAL_SETTING_FILE) as data:

        if "money" not in data.keys():
            data["money"] = 100
        data["money"] += event_data.get("money_delta")
        return data["money"]

def get_money():
    return update_account(0)




def print_leader_board():
    pass

def print_history():
    pass

def manual_transaction():
    pass


def get_data_by_name():
    pass

def set_data_by_name():
    pass

def daily_reward():
    pass

def sync_queue_cut_in_line():
    pass
def sync_queue_wait_in_line():
```

```
    pass

def sync_gap_too_long():
    pass

def open_doc_with_warning_count():
    pass
```

LOG.py

```
        function: The decorated function.
    """
    # If a script has multiple aliases, just use the lonest one as the record.
    if isinstance(func_name_as_record, list):
        func_name_as_record = max(func_name_as_record, key=len)

    def decorator(func):
        def wrapper(*args, **kwargs):
            with DATA_FILE.update_data(LOG_FILE_NAME) as data:
                t_start = time.time()
                out = func(*args, **kwargs)
                t_end = time.time()
                if not data:
                    data = dict()
                data[TIME.get_formatted_current_time()] = {
                    "application": ENVIRONMENT.get_app_name(),
                    "function_name": func_name_as_record.replace("\n", " "),
                    "arguments": args,
                    "result": str(out),
                    "script_path": script_path,
                    "duration": TIME.get_readable_time(t_end - t_start),
                }

            return out

        return wrapper

    return decorator


def read_log(user_name=USER.USER_NAME):
    """Read the log file of a specific user.

    Args:
        user_name (str, optional): The name of the user. Defaults to current
user.
    """
    data = DATA_FILE.get_data(LOG_FILE_NAME)
    print("Printing user log from <{}>".format(user_name))
    import pprint

    pprint.pprint(data, indent=4)


def unit_test():
    pass


############################################################
if __name__ == "__main__":
    unit_test()
```

MATH.py

```
pass
```

MODULE_HELPER.py

```
    Args:
        folder (str): The folder name for the button script, in EnneadTab
sources codes folder.
        file_name (str): The file name for the button script, without the .py
extension.
        func_name (str): The function name to run in the button script. To run
entire script, use "file_name".
        *args: Positional arguments to pass to the function.
    """

    root = ENVIRONMENT.RHINO_FOLDER
    module_path = "{}\\{}".format(root, locator)

    # this is to handle only one senario---the speciall installer rui folder
structure
    if not os.path.exists(module_path):
        module_path = "{}\\RHINO\\{}".format(ENVIRONMENT.CORE_FOLDER, locator)

    # add the folder of the module to the system path for referencing additional
modules
    module_folder = os.path.dirname(module_path)
    if module_folder not in sys.path:
        sys.path.append(module_folder)

    # add core lib
    if ENVIRONMENT.LIB_FOLDER not in sys.path:
        sys.path.append(ENVIRONMENT.LIB_FOLDER)

    # ensure core can load
    from EnneadTab import ERROR_HANDLE

    head, tail = os.path.split(module_path)
    func_name = tail.replace(".py", "")
    module_name = FOLDER.get_file_name_from_path(module_path).replace(".py", "")
    ref_module = imp.load_source(module_name, module_path)

    func = getattr(ref_module, func_name, None)
    if func is None:
        for surfix in ["_left", "_right"]:
            func = getattr(ref_module, func_name.replace(surfix, ""), None)
            if func is not None:
                break
        else:
            NOTIFICATION.messenger(
                main_text="Oooops, cannot find the func <{}> in source
code.\nContact SZ and let him know. Thx!".format(
                    func_name
                )
            )
            return

    # no longer decide to aapkly pre error
    # cather here. so the scripte structure can be same for revit and rhino
    # also make it possoble to call alis run
    # directly in the future and still get proper logging and error handle
    func(*args, **kwargs)
```

NOTIFICATION.py

```
    if not isinstance(main_text, str):
        main_text = str(main_text)

    data = {}
    data["main_text"] = main_text
    data["animation_in_duration"] = 0.5
    data["animation_stay_duration"] = animation_stay_duration
    data["animation_fade_duration"] = 2
    data["width"] = width
    data["height"] = height or 150 + str(main_text).count("\n") * 40
    data["image"] = image
    data["x_offset"] = x_offset

    DATA_FILE.set_data(data, "messenger_data.sexyDuck")

    EXE.try_open_app("Messenger")


def duck_pop(main_text = None):
    """Pop a duck to the user, which disappears after a few seconds.

    Args:
        main_text (str, optional): The message to show. Defaults to "Quack!".
    """
    if is_hate_duck_pop():
        messenger(main_text)
        return

    if not main_text:
        main_text = "Quack!"

    data = {}
    data["main_text"] = main_text

    # when the ranking is ready, can progress to make better ranked duck
    data["duck_image"] = IMAGE.get_one_image_path_by_prefix("duck_pop")
    data["explosion_gif"] = IMAGE.get_image_path_by_name("duck_explosion.gif")
    data["audio"] = SOUND.get_one_audio_path_by_prefix("duck")
    DATA_FILE.set_data(data, "DUCK_POP.sexyDuck")

    EXE.try_open_app("DuckPop", legacy_name="Duck_Pop")


def unit_test():
    duck_pop("Hello, Ennead!")
    messenger("Hello Ennead!")

if __name__ == "__main__":
    duck_pop("Hello, world!")
    messenger("Hello world")
```

OUTPUT.py

```
    output.write("Sample text in 'Title' style",Style.Title)
    output.write("Sample text in 'SubTitle' style",Style.SubTitle)
```

```
    output.write("Sample text in default style")
    output.write("sample text in foot note style(this is not working yet)",
Style.Footnote)

    output.insert_division()
    output.write("\n\n")
    output.insert_division()

    output.write("Trying to print list as item list")
    test_list = ["A", "B", "C", 99, 440, 123]
    output.write(test_list)
    output.write("Trying to print list as str")
    output.write(test_list, as_str=True)

    output.insert_division()


    output.write("Trying to print a random meme image")
    output.write(IMAGE.get_one_image_path_by_prefix("meme"))


    output.insert_division()


    output.write("Trying to print a button")
    output.write("bt_sample button")


    output.insert_division()
    new_output = get_output()
    new_output.write("This is a new output object but should write to same old
output window")
    new_output.plot()



def display_output_on_browser():
    if not ENVIRONMENT.IS_REVIT_ENVIRONMENT:
        NOTIFICATION.messenger("currently only support Revit Env")
        return
    from pyrevit import script
    dest_file = FOLDER.get_EA_dump_folder_file("EnneadTab Output.html")
    output = script.get_output()
    output.save_contents(dest_file)
    output.close()
    os.startfile(dest_file)

##########################################################
if __name__ == "__main__":
    unit_test()
```

OVERLOAD.py

```
"""example on overload handling

import System.Collections.Generic.IEnumerable as IEnumerable


for srf in srfs:
    splitBrep = srf.Split.Overloads[IEnumerable[Rhino.Geometry.Curve],
System.Double](cutters, tol)



wrong example.....
```

```
if you have a out parameter, iron python will return as tuple instead using it
in args
success, family_ref = project_doc.LoadFamily.Overloads[str,
DB.IFamilyLoadOptions](temp_path, loading_opt, family_ref)
"""

import os
```

PDF.py

```
    return pdf_path


def pdfs2pdf(combined_pdf_file_path, list_of_filepaths, reorder = False):
    """merge multiple pdfs to single pdf.

    Args:
        combined_pdf_file_path (str): path for final product
        list_of_filepaths (list): list of l=path for the input pdfs
        reorder (bool, optional): reorder the pdf alphabetically. Defaults to
False.
    """
    from PyPDF2 import PdfFileMerger

    merger = PdfFileMerger()

    if reorder:
        list_of_filepaths.sort()

    for filepath in list_of_filepaths:
        merger.append(filepath)

    merger.write(combined_pdf_file_path)
    merger.close()


def images2pdf(combined_pdf_file_path, list_of_filepaths, reorder = False):
    """merge multiple images to single pdf.

    Args:
        combined_pdf_file_path (str): path for final product
        list_of_filepaths (list): list of l=path for the input images
        reorder (bool, optional): reorder the pdf alphabetically. Defaults to
False.
    """
    from PyPDF2 import PdfFileMerger

    from PIL import Image
    merger = PdfFileMerger()

    if reorder:
        list_of_filepaths.sort()

    for filepath in list_of_filepaths:
        with Image.open(filepath) as img:
            merger.append(img)

    merger.write(combined_pdf_file_path)
    merger.close()
```

SAMPLE_FILE.py

```
"""used to retrive sample file in rhino, reivt and excel.
Good the sharing template"""

import os
import ENVIRONMENT
import NOTIFICATION

def get_file(file_name):

    path = "{}\\{}\\{}".format(ENVIRONMENT.DOCUMENT_FOLDER,
ENVIRONMENT.get_app_name(), file_name)
    if os.path.exists(path):
        return path
    NOTIFICATION.messenger("Cannot find [{}]".format(file_name))
```

SECRET.py

```
    data = get_dev_dict()
    developer_data = data.get(developer_name)
    if not developer_data:
        return
    return developer_data.get(key)


def get_dev_dict():
    """Get the dictionary of developers from the secret file.

    Returns:
        dict: The dictionary of developers.
    """
    developer_file = "ENNEADTAB_DEVELOPERS.secret"
    L_drive_file_path = os.path.join(ENVIRONMENT.DB_FOLDER, developer_file)
    if ENVIRONMENT.IS_OFFLINE_MODE:
        return DATA_FILE.get_data(developer_file)
    return DATA_FILE.get_data(L_drive_file_path)



def unit_test():
    """Unit test for the SECRET module."""
    import pprint

    app_names = [
        "chatgpt_api_key",
        "translator_api_key",
        "reporter_api_key",
        "clone_helper",
        "miro_oauth",
    ]

    print("######### API KEY TEST #########")
    for app_name in app_names:
        print("{name}: {key}".format(name=app_name, key=get_api_key(app_name)))
    print("######### DEV DICT TEST #########")
    pprint.pprint(get_dev_dict())
    print("######### DEV INFO TEST #########")
    for dev_name in get_dev_dict().keys():
        for key in get_dev_dict()[dev_name].keys():
            print(
                "{dev_name}: {key}: {value}".format(
                    dev_name=dev_name, key=key, value=get_dev_info(dev_name,
```

```
key)
                )
            )


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

SHORT_CUT.py

```
pass
```

SOUND.py

```
            self.file = file


            # Create a flag to control the music playback
            self.stop_flag = threading.Event()

            # Create a thread to play music
            self.music_thread = threading.Thread(target=self.play)

            # Start the music thread
            self.music_thread.start()

        def stop(self):

            # Set the stop flag to terminate the music thread
            self.stop_flag.set()

            # Wait for the music thread to finish
            self.music_thread.join()

            print("Music stopped.")
            pass


def sys_alert():
    #play window alert sound
    import winsound
    duration = 100  # milliseconds
    freqs = [440,
            500,
            600,
            900]# Hz
    for i,f in enumerate(freqs):
        if i == len(freqs)-1:
            duration = 400
        winsound.Beep(f, duration)
#############
if __name__ == "__main__":
    print(__file__ + "   -----OK!")
    # unit_test()
    file = "sound_effect_spring"
    # play_sound(file)
    # play_sound()
    test_play_all_sounds()
```

```
    # sys_alert()
```

SPEAK.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import random
import DATA_FILE
import EXE
import CONFIG

def random_speak(lines, chance=1.0):
    if random.random() <= chance:
        random.shuffle(lines)
        speak(lines[0])


def is_hate_talkie():
    return not CONFIG.get_setting("toggle_bt_is_talkie", False)


def speak(text, language='en', accent='com'):
    """
    #language = 'zh-CN'
    #language = 'zh-TW'
    #language = 'en'

    #accent = 'co.uk'
    #accent = 'co.in'
    #accent = 'com'
    """
    if is_hate_talkie():
        return

    if not text:
        return

    data = {}
    data["text"] = text
    data["language"] = language
    data["accent"] = accent
    DATA_FILE.set_data(data, "text2speech.sexyDuck")



    EXE.try_open_app("Speaker")


def unit_test():
    speak("I like to move it move it!")


if __name__ == "__main__":
    speak("This is a test?")
```

TASK.py

```
import subprocess
```

```
class TaskScheduler:


    def add_scheduled_task(self, task_name, exe_path):
        # Format the command to add to the Task Scheduler
        command = 'schtasks /create /tn "{}" /tr "{}" /sc daily /st
00:00'.format(task_name, exe_path)

        try:
            # Run the command
            subprocess.check_call(command, shell=True)
            print ("Task scheduled successfully: {}".format(task_name))
        except subprocess.CalledProcessError as e:
            print ("Failed to schedule task:", e)

    def remove_scheduled_task(self, task_name):
        # Format the command to delete the task from the Task Scheduler
        command = 'schtasks /delete /tn "{}" /f'.format(task_name)

        try:
            # Run the command
            subprocess.check_call(command, shell=True)
            print ("Task '{}' removed successfully.".format(task_name))
        except subprocess.CalledProcessError as e:
            print ("Failed to remove scheduled task:", e)
```

TEXT.py

```
import ENVIRONMENT

import sys
sys.path.append(ENVIRONMENT.DEPENDENCY_FOLDER)

from termcolor import colored # pyright: ignore
from COLOR import TextColorEnum


def colored_text(text, color = TextColorEnum.Cyan, on_color=None, attrs=None):
    """Colorize text.

    Available text colors:
        red, green, yellow, blue, magenta, cyan, white.

    Available text highlights:
        on_red, on_green, on_yellow, on_blue, on_magenta, on_cyan, on_white.

    Available attributes:
        bold, dark, underline, blink, reverse, concealed.

    Example:
        colored('Hello, World!', 'red', 'on_grey', ['blue', 'blink'])
        colored('Hello, World!', 'green')
    """


    if "colored" not in globals():
        # in some terminal run, it cannot read the dependecy folder so cannot
load the colored moudle
        return text
    return colored(text, color, on_color, attrs)



def unit_test():
    print (colored_text("Test dfault color text"))
```

```
    print (colored_text("test green", TextColorEnum.Green))#,
attrs=[TextColorEnum.Blue, 'blink']))



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

TIME.py

```
            #print("The Elapsed event was raised at", datetime.datetime.now())

            if self.current_count > 0:

                self.timer = threading.Timer(self.interval, self.on_timed_event)
                self.timer.start()
            else:
                print("Timer stopped after", self.life_span, "seconds")
                self.stop_timer()

        def stop_timer(self):
            if self.timer:
                self.timer.cancel()

        def begin(self):
            print("Timer begins!")
            self.timer = threading.Timer(self.interval, self.on_timed_event)
            self.timer.start()
            #print(self.timer.is_alive())



def get_revit_uptime():
    import ENVIRONMENT
    if not ENVIRONMENT.IS_REVIT_ENVIRONMENT:
        return "Not in Revit"
    from pyrevit.coreutils import envvars
    if not envvars.get_pyrevit_env_var("APP_UPTIME"):
        update_revit_uptime()
    uptime = time.time() - envvars.get_pyrevit_env_var("APP_UPTIME")
    uptime = get_readable_time(uptime)
    return uptime


def update_revit_uptime():
    from pyrevit.coreutils import envvars
    envvars.set_pyrevit_env_var("APP_UPTIME", time.time())

def unit_test():

    print ("Current Revit UpTime = {}".format(get_revit_uptime()))

    print ("Current Time = {}".format(get_formatted_current_time()))

if __name__ == "__main__":
    timer_example = AutoTimer(life_span=10,
                              show_progress=True,
                              interval= 0.1)
    timer_example.begin()
```

TIMESHEET.py

```
                if starting_time and end_time:
                    duration = end_time - starting_time
                    temp[date] = duration
                    proj_dict[doc_name] = temp

        for proj_name, proj_info in sorted(proj_dict.items()):
            total_duration = sum(proj_info.values())
            table_data.append([proj_name] +
[TIME.get_readable_time(proj_info.get(date, 0)) if proj_info.get(date, 0) != 0
else "N/A" for date in sorted(valid_dates)] +
[TIME.get_readable_time(total_duration)])

        output.print_table(table_data=table_data,
                           title="Revit Timesheet",
                           columns=["Proj. Name"] + sorted(valid_dates) +
["Total Hour"])

    all_dates = sorted(log_data.keys())
    seg_max = 10
    for i in range(0, len(all_dates), seg_max):
        if i + seg_max < len(all_dates):
            dates = all_dates[i:i + seg_max]
        else:
            dates = all_dates[i:]
        print_table(dates)



def _update_time_sheet_by_software(doc_name, software):
    with DATA_FILE.update_data(TIMESHEET_DATA_FILE) as data:
        software_data = data.get(software, {})
        today = time.strftime("%Y-%m-%d")

        today_data = software_data.get(today, {})
        current_doc_data = today_data.get(doc_name, {})
        if "starting_time" not in current_doc_data:
            current_doc_data["starting_time"] = time.time()
        current_doc_data.update({"end_time": time.time()})
        today_data[doc_name] = current_doc_data

        software_data[today] = today_data
        data[software] = software_data


def unit_test():
    update_timesheet("test_project_revit_1")
    update_timesheet("test_project_revit_2")
    update_timesheet("test_project_rhino_1")

    print_timesheet_detail()


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

UI.py

```
pass
```

UNIT_TEST.py

```
                self.process_folder(module_path)
                continue

            if not module_file.endswith(".py"):
                continue
            module_name = module_file.split(".")[0]
            if module_name in IGNORE_LIST:
                continue
            try:
                module = imp.load_source(module_name, module_path)
            except:
                try:
                    import importlib

                    module = importlib.import_module(module_name)
                except Exception as e:
                    print(
                        "\n\nSomething is worng when importing [{}]
becasue:\n\n++++++{}++++++\n\n\n".format(
                            print_text_in_highlight_color(module_name,
ok=False),
                            traceback.format_exc(),
                        )
                    )
                    continue

            if not self.try_run_unit_test(module):
                self.failed_module.append(module_name)


def test_core_module():
    tester = UnitTest()

    tester.process_folder(ENVIRONMENT.CORE_FOLDER)
    if len(tester.failed_module) > 0:
        print("\n\n\nbelow modules are failed.")
        print("\n--".join(tester.failed_module))
        raise TooManyFailedModuleException

    OUTPUT.display_output_on_browser()


class TooManyFailedModuleException(BaseException):
    def __init__(self):
        super().__init__(
            "There are too many failed module during unit-test for the core
module."
        )


if __name__ == "__main__":
    test_core_module()
    pass
```

USER.py

```
def get_rhino_developer_emails():
    out = []
    for developer_data in EnneadTab_DEVELOPERS.values():
        if len(developer_data["system_id"]) == 0:
            continue
        out += developer_data["email"]
    return out

def get_revit_developer_emails():
    out = []
    for developer_data in EnneadTab_DEVELOPERS.values():
        if len(developer_data["autodesk_id"]) == 0:
            continue
        out += developer_data["email"]
    return out


def unit_test():
    import inspect
    import pprint
    # get all the global varibales in the current script
    for i, x in enumerate(sorted(globals())):
        content = globals()[x]

        if inspect.ismodule(content):

            continue
        if not x.startswith("_") and not callable(content):
            if isinstance(content,dict):
                print(x, " = ")
                pprint.pprint(content)
            else:
                print(x, " = ", content)

    print ("current user [{}] is a developer? {}".format(USER_NAME,
UNIT_TEST.print_boolean_in_color(is_EnneadTab_developer())))
    print ("my system name = {}".format(USER_NAME))
    print ("my autodesk name = {}".format(get_autodesk_user_name()))
    print ("Am I a developer?
{}".format(UNIT_TEST.print_boolean_in_color(IS_DEVELOPER)))


    system_usernames, autodesk_usernames = get_usernames_from_developers()
    print ("all system_usernames = {}".format(system_usernames))
    print ("all autodesk_usernames = {}".format(autodesk_usernames))
    print ("all rhino developer emails =
{}".format(get_rhino_developer_emails()))
    print ("all revit developer emails =
{}".format(get_revit_developer_emails()))
###############
if __name__ == "__main__":
    unit_test()
```

VERSION_CONTROL.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-


import os

import EXE
import ENVIRONMENT
```

```
import NOTIFICATION



def update_EA_dist():
    EXE.try_open_app("EnneadTab_OS_Installer", safe_open=True)
    EXE.try_open_app("RegisterAutoStartup", safe_open=True)



def show_last_success_update_time():
    records = [file for file in os.listdir(ENVIRONMENT.ECO_SYS_FOLDER) if
file.endswith(".duck") and not "_ERROR" in file]
    if len(records) == 0:
        NOTIFICATION.messenger("Not successful update recently.\nYour life
sucks.")
        return
    records.sort()
    record = records[-1]
    with open(os.path.join(ENVIRONMENT.ECO_SYS_FOLDER, record)) as f:
        commit_line = f.readlines()[-1].replace("\n","")
    NOTIFICATION.messenger("Most recent update
at:{}\n{}".format(record.replace(".duck", ""),
                                                                    commit_line))

    pass

def unit_test():
    update_EA_dist()



if __name__ == "__main__":
    unit_test()
    show_last_success_update_time()
```

WEB.py

```
pass
```