docx\_generator.py

"""

В этом модуле происходит вызов модулей парсинга для получения данных об учебных дисциплинах, затем происходит

генерация файлов РПД в формате .docx

"""

from difflib import SequenceMatcher

from docxtpl import DocxTemplate

from tkinter import filedialog

from tkinter import \*

import os

from parser\_matrix import get\_info\_from\_excel

from parser\_plane import get\_info\_from\_education\_plane

# Создание графического окна

gui\_win = Tk()

def check\_number(num: int) -> str:

"""

Функция для проверки принадлежности числа к одной из трёх групп

:param num: число, которое будет проверяться

:type num: int

:return: Возвращает одну из трёх групп

:rtype: str

"""

if num % 10 == 1 and num != 11:

return '1'

elif 1 < num % 10 < 5 and (num > 19 or num < 5):

return '2'

else:

return '3'

def main() -> None:

"""

Основная функция с циклом для графического интерфейса

"""

gui\_win.title('Генератор РПД')

gui\_win.geometry('400x200')

gui\_win.grid\_rowconfigure(0, weight=1)

gui\_win.grid\_columnconfigure(0, weight=1)

dialog\_btn = Button(gui\_win, text='Выберите директорию для генерации РПД', command=generator)

dialog\_btn.pack()

gui\_win.mainloop()

def get\_filepath() -> str:

"""

Функция для создания дилогового окна выбора директории

:return: строка выбранного пользователем пути

:rtype: str

"""

filepath = filedialog.askdirectory(initialdir=r"C:/",

title="Dialog box")

label\_path = Label(gui\_win, text="Генерация выполнена по пути " + filepath, font='italic 14')

label\_path.pack(pady=20)

return filepath

def create\_folder(filepath: str) -> None:

"""

Функция для создания директории, в которую будут сохраняться файлы

:param filepath: путь, выбранный пользователемдля генерации

:type filepath: str

:raises FileExistsError: если в выбранной директории уже существует папка с

названием generated\_files, программа вызовет ошибку, но продолжит работу

"""

try:

os.mkdir(os.path.join(filepath, "generated\_files"))

except FileExistsError:

print("Folder already created")

def set\_generator\_params(data: dict, key: str) -> dict:

"""

Функция получения информации о дисциплине по названию

:param data: данные об учебном плане

:type data: dict

:param key: название учебной дисциплины

:type key: str

:raises KeyError: если в учебном плане не была найдена переданная учебная дисциплина

:return: список данных о конкреной учебной дисциплине

:rtype: dict

"""

context\_plane = {}

try:

context\_plane = get\_info\_from\_education\_plane("../media/excel/planes/03-5190 - ВЕБ 2020 (1).xlsx")[key]

except KeyError:

for error\_key in get\_info\_from\_education\_plane("../media/excel/planes/03-5190 - ВЕБ 2020 (1).xlsx"):

if SequenceMatcher(None, key, error\_key).ratio() >= 0.75:

context\_plane = get\_info\_from\_education\_plane("../media/excel/planes/03-5190 - ВЕБ 2020 (1).xlsx")[error\_key]

break

context\_plane["program\_name"] = data[key]["program\_name"]

context\_plane["program\_code"] = data[key]["program\_code"]

context\_plane["profile\_name"] = data[key]["profile\_name"]

context\_plane["year\_start"] = data[key]["year\_start"]

context\_plane["current\_year"] = data[key]["current\_year"]

context\_plane['intensity\_ZET\_check'] = check\_number(context\_plane['intensity\_ZET'])

context\_plane['intensity\_hours\_check'] = check\_number(context\_plane['intensity\_hours'])

context\_plane['total\_homework\_hours\_check'] = check\_number(context\_plane['total\_homework\_hours'])

for i, \_ in enumerate(context\_plane['courses']):

context\_plane['courses'][i]['ZET\_check'] = check\_number(int(context\_plane['courses'][i]['ZET']))

context\_plane['courses'][i]['hours\_check'] = check\_number(int(context\_plane['courses'][i]['hours']))

context\_plane['courses'][i]['homework\_time\_check'] = check\_number(int(context\_plane['courses'][i]['homework\_time']))

return context\_plane

def fix\_docx\_tables(doc: DocxTemplate) -> None:

"""

Функция удаления пустых строк из таблиц в генерируемом файле

:param doc: объект генерируемого файлы

:type doc: DocxTemplate

"""

for i in range(len(doc.tables)):

table = doc.tables[i].\_tbl

for row in doc.tables[i].rows:

if len(row.cells[0].text.strip()) == 0 and len(set(row.cells)) == 1:

table.remove(row.\_tr)

def generate\_docx(data: dict, key: str, filepath: str) -> None:

"""

Функция генерации и сохранения файла РПД

:param data: данные учебного плана по всем дисциплинам

:type data: dict

:param key: название учебной дисциплины, по которой генерируется РПД

:type key: str

:param filepath: путь сохранения файла, выбранный пользователем

:type filepath: str

"""

context\_plane = set\_generator\_params(data, key)

doc = DocxTemplate("../templates/template.docx")

doc.render(dict(data[key], \*\*context\_plane))

fix\_docx\_tables(doc)

doc.save("{}/generated\_files/{}.docx".format(filepath, key))

def generator() -> None:

"""

Функция для парсинга и генерации РПД по всем учебным дисциплинам

"""

filepath = get\_filepath()

create\_folder(filepath)

data, key\_data = get\_info\_from\_excel(

"../media/excel/matrices/09\_03\_01\_Информатика\_и\_ВТ,\_Матрица\_ВЕБ\_технологии\_2020.xlsx")

for key in key\_data:

generate\_docx(data, key, filepath)

if \_\_name\_\_ == '\_\_main\_\_':

main()

parser\_plane.py

"""

Модуль парсинга учебного плана

"""

from typing import Union

import xlrd

def get\_matrix(filename: str) -> list:

"""

Функция получения матрицы предметов из файла

:param filename: путь к файлу, из которого получается матрица

:type filename: str

:return: список данных, полученных из файла

:rtype: list

"""

xls = xlrd.open\_workbook(filename)

xls = xls.sheet\_by\_index(0)

return [

[str(xls.cell\_value(i, j)).strip() for j in range(xls.ncols)]

for i in range(xls.nrows)

]

def number\_to\_words(n: int) -> str:

"""

Метод преобразования числа в словесное его представление

:param n: число, по котрому необходимо получить название

:type n: int

:return: строка с полным названием числа

:rtype: str

"""

less\_than\_ten = {1: 'первом', 2: 'втором', 3: 'третьем', 4: 'четвёртом',

5: 'пятом', 6: 'шестом', 7: 'седьмом', 8: 'восьмом',

9: 'девятом'}

ten = {10: 'десятом'}

from\_eleven\_to\_nineteen = {11: 'одиннадцатом', 12: 'двенадцатом',

13: 'тринадцатом', 14: 'четырнадцатом',

15: 'пятнадцатом', 16: 'шестнадцатом',

17: 'семнадцатом', 18: 'восемнадцатом',

19: 'девятнадцатом'}

n1 = n % 10

n2 = n - n1

if n < 10:

return less\_than\_ten.get(n)

elif 10 < n < 20:

return from\_eleven\_to\_nineteen.get(n)

elif n >= 10 and n in ten:

return ten.get(n)

else:

return ten.get(n2) + ' ' + less\_than\_ten.get(n1)

def hours\_to\_zet(h: int) -> Union[int, float]:

"""

Метод перевода часов в ZET-единицы

:param h: количество часов

:type h: int

:return: количество ZET-единиц от полученных часов

:rtype: Union[int, float]

"""

z = round(h / 36, 1)

if z == int(z):

return int(z)

else:

return z

def get\_courses(arr: list, imp\_cols: dict, met: str = 'moduls') -> list:

"""

Функция получения данных о дисциплине (семестр, курс, вид зачётного мероприятия, часы)

:param arr: список всех данных о дисциплине

:type arr: list

:param imp\_cols: набор параметров, которые необходимо получить

:type imp\_cols: dict

:param met: вид дисциплины (практика, элктивная, основная), defaults to 'moduls'

:type met: str

:return: список необходимых данных о дисциплине

:rtype: list

"""

if met == 'practice':

sems = [

to\_int(el) for el in arr[imp\_cols['exam']].replace(' ', '').split(',')

] + [

to\_int(el) for el in arr[imp\_cols['credit']].replace(' ', '').split(',')

]

sems = list(filter(lambda x: x != 0, sems))

return [{

'semester': number\_to\_words(sem),

'course': number\_to\_words(int(round(sem / 2 + 0.1))),

'test': 'экзамен' if str(sem) in arr[imp\_cols['exam']] else 'зачет',

'hours': to\_int(arr[imp\_cols['ZET']]) \* 36,

'ZET': to\_int(arr[imp\_cols['ZET']]),

'homework\_time': 0,

} for sem in sems]

if met == 'elective':

sem = to\_int(arr[imp\_cols['elective\_sem']])

hours = to\_int(arr[imp\_cols['elective\_hours']])

return [{

'semester': number\_to\_words(sem),

'course': number\_to\_words(int(round(sem / 2 + 0.1))),

'test': 'зачет',

'hours': hours,

'ZET': hours\_to\_zet(hours),

'homework\_time': 0,

}]

courses = list(map(lambda el: [el[0] + 1, el[1]], enumerate(arr[imp\_cols['sems']::])))

courses = list(filter(lambda el: el[1] != '', courses))

courses = list(map(lambda el: [el[0], float(el[1])], courses))

courses\_count = len(courses)

all\_homework = to\_int(arr[imp\_cols['homework']])

homeworks = [[sem, time / 2] for sem, time in courses]

if sum([el[1] for el in homeworks]) != all\_homework:

homeworks = [[sem, time] for sem, time in courses]

if sum([el[1] for el in homeworks]) != all\_homework:

div = all\_homework // courses\_count

ost = all\_homework % courses\_count

homeworks = [[sem, div] for sem, \_ in courses]

idx = [el[1] for el in courses].index(max([el[1] for el in courses]))

homeworks[idx][1] += ost

homeworks = dict(homeworks)

res = []

for sem, time in courses:

res += [{}]

res[-1]['semester'] = number\_to\_words(sem)

res[-1]['course'] = number\_to\_words(int(round(sem / 2 + 0.1)))

res[-1]['test'] = 'экзамен' if str(sem) in arr[imp\_cols['exam']] else 'зачет'

res[-1]['homework\_time'] = to\_int(homeworks[sem])

res[-1]['hours'] = to\_int(time) + res[-1]['homework\_time']

res[-1]['ZET'] = hours\_to\_zet(to\_int(res[-1]['hours']))

return res

def to\_int(x: str) -> int:

"""

Функция преобразования численных данных из текста в число

:param x: текст для преобразования

:type x: str

:raises ValueError: если полученные данные не представлены числом в текстовом формате, то вызывается ошибка,

а функция возвращает 0

:return: полученное числовое значение

:rtype: int

"""

try:

return int(float(x))

except ValueError as exc:

# print(exc)

return 0

def find\_from\_matrix(dct: dict, matrix: list, idx: int = 0) -> dict:

"""

Функция поиска данных о дисциплине

:param dct: список параметров для поиска

:type dct: dict

:param matrix: матрица, из которой получаются данные

:type matrix: list

:param idx: переменна для разделения различных типов запросов на поиск, defaults to 0

:type idx: int

:return: список данных о дисциплине по параметрам

:rtype: dict

"""

rev = dict([[val, key] for key, val in dct.items()])

res = {}

for i in range(len(matrix)):

for j in range(len(matrix[0])):

if matrix[i][j] in dct.values() and not rev[matrix[i][j]] in res.keys():

res[rev[matrix[i][j]]] = i if idx == 0 else j

return res

def get\_info\_from\_education\_plane(filename: str) -> dict:

"""

Функция получения информации из учебного плана

:param filename: путь к файлу с данными

:type filename: str

:return: список всей полученной информации

:rtype: dict

"""

matrix = get\_matrix(filename)

imp\_rows = find\_from\_matrix({

'subjects': 'Обязательная часть',

'practice': 'Б.2',

'elective': 'Факультативные дисциплины',

}, matrix, 0)

imp\_cols = find\_from\_matrix({

'credit': 'зачетов',

'exam': 'экзаменов',

'hours': 'ВСЕГО по структуре',

'ZET': 'Всего, ЗЕТ',

'homework': 'Самостоятельная работа',

'sems': 'Распределение по курсам и семестрам, ауд. час.',

'subjects': 'Обязательная часть',

'B.1': 'Б.1',

'elective': 'Факультативные дисциплины',

'elective\_sem': 'Семестр',

'elective\_hours': 'Ауд. часов',

}, matrix, 1)

for i in range(len(matrix))[imp\_rows['subjects']::]:

matrix[i][imp\_cols['subjects']] = matrix[i][imp\_cols['subjects']].split('\*')[0].strip()

data = {}

for i in range(imp\_rows['subjects'], imp\_rows['practice']):

if matrix[i][imp\_cols['hours']] != '' and matrix[i][imp\_cols['B.1']] == '':

key = matrix[i][imp\_cols['subjects']]

data[key] = {}

data[key]['intensity\_hours'] = to\_int(matrix[i][imp\_cols['hours']])

data[key]['intensity\_ZET'] = hours\_to\_zet(to\_int(data[key]['intensity\_hours']))

data[key]['total\_homework\_hours'] = to\_int(matrix[i][imp\_cols['homework']])

data[key]['courses'] = get\_courses(matrix[i], imp\_cols, met='moduls')

for i in range(imp\_rows['practice'], imp\_rows['elective']):

if matrix[i][imp\_cols['B.1']] == '':

key = matrix[i][imp\_cols['subjects']]

data[key] = {}

data[key]['intensity\_ZET'] = to\_int(matrix[i][imp\_cols['ZET']])

data[key]['intensity\_hours'] = data[key]['intensity\_ZET'] \* 36

data[key]['total\_homework\_hours'] = 0

data[key]['courses'] = get\_courses(matrix[i], imp\_cols, met='practice')

i = imp\_rows['elective'] + 1

while matrix[i][imp\_cols['elective']] != '':

key = matrix[i][imp\_cols['elective']]

data[key] = {}

data[key]['intensity\_hours'] = to\_int(matrix[i][imp\_cols['elective\_hours']])

data[key]['intensity\_ZET'] = hours\_to\_zet(data[key]['intensity\_hours'])

data[key]['total\_homework\_hours'] = 0

data[key]['courses'] = get\_courses(matrix[i], imp\_cols, met='elective')

i += 1

return data

parser\_matrix.py

"""

Модуль парсинга матрицы компетенций

"""

import datetime

from typing import Union

import openpyxl

import xlrd

def get\_parents(matrix: list, r: int) -> list[str, str, str]:

"""

Функция получения компетенций

:param matrix: матрица данных

:type matrix: list

:param r: строка компетенции

:type r: int

:return: список данных о компетенции

:rtype: list[3]

"""

scd = matrix[r][1].replace('\n', '')

fst = matrix[r][0].replace('\n', '')

return [fst, scd, matrix[r][2]]

def get\_info\_for\_table(matrix: list, rng: list, c: int) -> list:

"""

Функция получения информации для составления таблицы компетенций

:param matrix: матрица данных

:type matrix: list

:param rng: список строк для каждой компетенции

:type rng: list

:param c: строка названия компетенции

:type c: int

:return: список данных для построения таблицы компетенций

:rtype: list

"""

res = [

{

'competency\_code': '',

'competency\_name': '',

'indicators': [['', set()]]

}

]

for r in rng:

if matrix[r][c] == '+':

f\_code, s\_code, t\_code = get\_parents(matrix, r)

if f\_code == '' or s\_code == '' or t\_code == '':

continue

code, name = [el.strip() for el in list(filter(bool, f\_code.split('.')))]

if res[-1]['competency\_code'] != code:

res.append({

'competency\_code': code,

'competency\_name': name,

'indicators': [['', set()]]

})

res[-1]['indicators'][0][0] = s\_code

res[-1]['indicators'][0][1].add(t\_code)

else:

if res[-1]['indicators'][-1][0] != s\_code:

res[-1]['indicators'].append(['', set()])

res[-1]['indicators'][-1][0] = s\_code

res[-1]['indicators'][-1][1].add(t\_code)

else:

res[-1]['indicators'][-1][1].add(t\_code)

del res[0]

return res

def get\_ranges(matrix: list) -> list:

"""

Функция получения строк, относящихся к одной компетенции

:param matrix: матрица данных

:type matrix: list

:return: список номеров строк для компетенции

:rtype: list

"""

rows = len(matrix)

skill\_types = []

k = 0

for i in range(rows)[1::]:

if matrix[i][2] == matrix[i][1] == matrix[i][0] and matrix[i][0] != '':

skill\_types += [range(k, i)]

k = i

del skill\_types[0]

skill\_types += [range(k, rows)]

return skill\_types

def parse\_title(txt: str) -> dict[str, Union[str, int]]:

"""

Функция парсинга заголовка файла

:param txt: строка заголовка файла

:type txt: str

:return: список данных, полученных из заголовка

:rtype: dict[str, Union[str, int]]

"""

import re

res = {}

txt = re.sub('[»«]', '"', txt)

txt = re.sub('[\n,]', '', txt)

mas = txt.split('"')

res['profile\_name'] = mas[5]

res['program\_code'] = mas[3]

txt = re.sub('["]', ' ', txt)

mas = txt.split()

for el in mas:

if el.count('.') == 2:

res['program\_code'] = el + ' ' + res['program\_code']

if el.count('/') == 1:

temp = el.split('/')

res['year\_start'] = temp[0]

res['year\_end'] = temp[1]

return res

def get\_matrix(filename: str) -> list:

"""

Функция получения матрицы из файла

:param filename: путь к файлу

:type filename: str

:return: список данных из файла

:rtype: list

"""

xls = xlrd.open\_workbook(filename)

xls = xls.sheet\_by\_index(0)

mx\_row, mx\_column = xls.nrows, xls.ncols

wb = openpyxl.load\_workbook(filename)

sheet = wb.get\_sheet\_by\_name(wb.get\_sheet\_names()[0])

all\_data = []

for row\_index in range(1, mx\_row + 1):

row = []

for col\_index in range(1, mx\_column + 1):

vals = sheet.cell(row\_index, col\_index).value

if vals is None:

for crange in sheet.merged\_cells:

clo, rlo, chi, rhi = crange.bounds

top\_value = sheet.cell(rlo, clo).value

if rlo <= row\_index <= rhi and clo <= col\_index <= chi:

vals = top\_value

break

row.append(vals)

if len(list(filter(bool, row))) > 0:

all\_data.append(row)

for i in range(len(all\_data)):

for j in range(len(all\_data[0])):

if all\_data[i][j] is None:

all\_data[i][j] = ''

all\_data[i][j] = str(all\_data[i][j]).strip()

return all\_data

def get\_data\_from\_matrix(data: dict, title: dict, matrix: list, skill\_types: list, c: int, key: str) -> dict:

"""

Функция заполнения списка данных по шаблону

:param data: данные, получаемые из матрицы

:type data: dict

:param title: данные из заголовка файла

:type title: dict

:param matrix: данные из матрицы данных файла

:type matrix: list

:param skill\_types: список строк компетенции

:type skill\_types: list

:param c: текущий столбец парсинка

:type c: int

:param key: название учебной дисциплины

:type key: str

:return: оюновлённые данные из матрицы

:rtype: dict

"""

data[key] = {}

data[key]['program\_name'] = key

data[key]['profile\_name'] = title['profile\_name']

data[key]['program\_code'] = title['program\_code']

data[key]['year\_start'] = title['year\_start']

data[key]['year\_end'] = title['year\_end']

data[key]['current\_year'] = str(datetime.date.today().year)

data[key]['part\_type'] = str.lower(matrix[1][c])

universal\_competences = get\_info\_for\_table(matrix, skill\_types[0], c)

general\_professional\_competencies = get\_info\_for\_table(matrix, skill\_types[1], c)

professional\_competencies = get\_info\_for\_table(matrix, skill\_types[2], c)

if len(universal\_competences) > 0:

data[key]['universal\_competences'] = universal\_competences

if len(general\_professional\_competencies) > 0:

data[key]['general\_professional\_competencies'] = general\_professional\_competencies

if len(professional\_competencies) > 0:

data[key]['professional\_competencies'] = professional\_competencies

return data

def get\_info\_from\_excel(filename: str) -> tuple[dict, list]:

"""

Функция получения информации из матрицы компетенций

:param filename: путь к файлу

:type filename: str

:return: список полученных данных и список названий учебных дисциплин

:rtype: tuple[dict, list]

"""

matrix = get\_matrix(filename)

skill\_types = get\_ranges(matrix)

for i in range(len(matrix))[::-1]:

if len(list(filter(bool, matrix[i]))) == 0:

del matrix[i]

cols = len(matrix[0])

title = parse\_title(matrix[0][0])

data = {}

key\_data = []

for c in range(cols)[3::]:

key = matrix[2][c]

if key == '':

continue

key\_data.append(key)

data = get\_data\_from\_matrix(data, title, matrix, skill\_types, c, key)

return data, key\_data