UVIC Timetable Builder

Peter Wilson

Sep 21, 2021

INTRODUCTION

Timeline Creator: Timeline Creator built in python3 for UVIC students with a CSC or ENG background

CHAPTER

ONE

UVIC TIMETABLE CREATOR

1.1 First define courses in excel file

The excel file contains specified courses with pertinent information for code success, define code as needed for your program. We then import necessary packages.

```
import sys
import shutil
import os

from term import term
from student import student
from course import course

from openpyxl import load_workbook
from ipysheet import from_dataframe, to_dataframe
import pandas as pd
```

Define importCourses function then import courses from excel sheet

```
def importCourses(excel_file,sheet_name):
    """
    imports data from excel file path
    """
    df = pd.read_excel(excel_file,sheet_name=sheet_name)
    raw_courses = df.values.tolist()
    courses = {}
    for name, offer_times, prereqs, completed in raw_courses:
        name = name.replace(' ','')
        ors = False
        if 'or' in prereqs:
            print('Not implemented yet')
        elif 'None' in prereqs:
            Prerequisites = []
        else:
            Prerequisites = prereqs.replace(' ','').split(',')
        courses[name]=course(name, offerTimes=offer_times, prereqs=Prerequisites,
↪completed=completed)

    return courses
```

```
excel_file = 'Courses.xlsx'
sheet_name = 'Sheet5'
courses = importCourses(excel_file,sheet_name)
```

1.2 Create instance of student and import courses

We use the 'printStudent()' function to view details about that object

```
Peter = student('Peter',courses=courses)
Peter.printStudent()
```

```
Peter
Completed: 24
Todo: 14
Courses: 38
```

1.3 Finally, create term instances

in first case we will add one class using 'addClasses()'

```
Summer = term('Su',num_classes=1, courses={})
Summer.addClasses({'SENG275':courses['SENG275']}, Peter)
```

```
Courses in Summer Semester:
SENG275
```

1.4 In the following terms we will use 'autoBuildTerm()'

```
Fall = term('F',num_classes=5, courses={'SENG360':courses['SENG360']})
Fall.autoBuildTerm(Peter)
```

```
Courses in Fall Semester:
SENG360
SENG321
CSC360
CSC370
SENG350
ECE360
```

```
Spring = term('Sp',num_classes=5, courses={})
Spring.autoBuildTerm(Peter)
```

```
Courses in Spring Semester:
SENG371
CSC320
```

(continues on next page)

(continued from previous page)

```
ECE455/CSC460
SENG401
```

```
Summer2 = term('Su',num_classes=5, courses={})
Summer2.autoBuildTerm(Peter)
```

```
Courses in Summer Semester:
SENG426
SENG440
SENG499
```

1.5 Finally, we print the student to view details

```
Peter.printStudent()
```

```
Peter
Completed: 38
Todo: 0
Courses: 38
```

CHAPTER

TWO

COURSE

class course.course(*name: str*, *offerTimes=[]*, *prereqs=[]*, *completed=False*, *importance=0*)

The course class represents a singular course and its pertinent information, such as offer times, prerequisites, and if it has been completed.

Parameters

- **name** – The name of the course (with no spaces)
- **offerTimes** – A list containing strings of which terms the course is offered (ex. [‘Sp’,’Su’])
- **prereqs** – A list of course names that are prerequisites
- **completed** – Boolean of if the course is Completed
- **importance** – integer value of importance of course (how many classes require this class, calculated later)

Returns None

printCourse()

Prints course information :param: None :returns: None

printPrereqs()

Prints prereqs :param: None :returns: None

setCompleted(*student*)

Sets course as completed :param student: student object from student.py :returns: None

setImportance(*value*)

Sets importance of course :param value: Integer value # classes that require this course :returns: None

takeCourseBool(*student*, *term*)

Parameters

- **student** – student object from student.py
- **term** – term the course wants to be taken (ex ‘Sp’)

Returns Whether the student can take this course that term

CHAPTER

THREE

STUDENT

class student.**student**(*name: str*, *courses={}*, *completed={}*, *todo={}*)

Class describing a student. A student contains the courses required by their program, the completed courses, and the ones to complete. Each course is a course object described by course.py. The courses in the student are stored in dictionaries. If you wanted to view the course 'SENG265' you would enter courses['SENG265'], which will return the course object.

Parameters

- **name** – Name of student (honestly worthless and never used)
- **courses** – A dictionary of course (ex. courses['SENG265'] will return the course object for SENG265)
- **completed** – Dictionary of completed courses (probably doesnt need to be a dictionary but is currently)
- **todo** – Dictionary of courses that have not been completed (also probably doesnt need to be a dictionary but currently is)

Returns None

courseValues()

Sets importance for each course depending on this students program (ie the course they are in) :param: None :returns: None

createCourseLists()

After student is passed in with dict of courses this function sorts to completed and todo courses :returns: None

getPrereqs(*courseName*)

Gets prequisites of courses :param courseName: name of course you want prereqs from :returns: list of prerequisites titles

printStudent()

Prints student information :param: None :returns: None

updateCompleted()

Updates students completed courses :param: None :returns: None

CHAPTER

FOUR

TERM

term.**sortClasses**(*possible_courses*)
Function used in 'autoBuildTerm()' to sort courses based on importance :param possible_courses: List of course objects :returns: Sorted list based on importance

class term.**term**(*name: str*, *num_classes=5*, *courses={}*)
term class representing each term (ex. 'Sp' or 'Su') that a student may want to take classes

Parameters

- **name** – The name of the term (ex 'Sp')
- **num_classes** – The amount of classes wanted to be taken that term
- **courses** – Dictionary containing courses, with keys set to the name of the course (ex. courses['SENG265'])

Returns None

addClasses(*courses*, *student*)
Overwrites courses in term and sets to the new courses passed in argument :param courses: Dictionary containing course objects :param student: Student object from student.py :returns: None

autoBuildTerm(*student*)
Builds a term of classes from a student object with the highest importance courses as priority. :param student: Student object from student.py :returns: None

printCourses()
Prints courses from the term :param: None :returns: None

updateCourses(*student*)
After the term is built, this updates the student object to set the Courses as completed. :param student: Student object from student.py :returns: None

CHAPTER

FIVE

WORKSPACE

workspace.**importCourses**(*excel_file*, *sheet_name*)
imports data from excel file path

workspace.**main**()
Given workspace to mess with code lol

PYTHON MODULE INDEX

INDEX