```
-----------------------------------------------------------------

Programmer Documentation.

Holds documentation for all functions used in the system and
information about the nose testing suite.


Authors: JT Kashuba, Noah Kruss, Logan Levitre, Zeke Petersen,
         River Veek


Group: TBD


Last Modified: 3/10/21
-----------------------------------------------------------------
```

==========

Nose Testing

==========

First, ensure that nose is installed by entering the following command into the terminal:

```
pip3 install nose
```

(Note that '`-v`' is the verbose flag. It outputs the docstring of the current test function)

To run all nosetests in the testing/ directory (from '`tbd/`' directory):

```
nosetests -v testing/*
```

To run all nosetests from a certain file (from '`tbd/`' directory):

```
nosetests -v testing/<test_file>
```

To run single test class (from '`tbd/`' directory):

```
nosetests -v testing/<test_file>:<class name>
```

To run single test module (single test from within a class; from ‘`tbd/`' directory):

```
nosetests -v testing/<test_file>:<class name>.<module name>
```

Some helpful tips for creating new nosetests:

- Be sure to add a one-sentence docstring describing the test function. As stated above, this is what will be outputted when the tests are run with the '`-v`' flag.
- Nose will only recognize that a function is a nose test if it has the word 'test' in the test function's name.
- A test function needs to contain an 'assert' statement; this is what allows the test function to equate to True or False.

For more information, refer to the nose documentation:
https://nose.readthedocs.io/en/latest/

==========

Student Object

==========

__init__(self)

Function to initialize Student class object

Parameters:

identifier: String input of the student ID of student

summer: Boolean indicator of whether the student is willing to take courses over the summer (defaults to False)

desired_grad_date: Tuple in the form (year: int, term: int) (defaults to (4,2))

max_credits_per_term: Int input that tells the system what the max number of credits the student will want in each term of their generated plan

Returns:

None

add_degree(self, degree_obj)

Function for adding a degree from the Student object (add degree to student.degree_list)

Parameters:

degree_obj: Degree object to be added to the Student object

Returns:

None

remove_degree(self, degree_name)

Function for removing a degree from the Student object (add degree to student.degree_list)

Parameters:

degree_name: String name of Degree object to be added to the Student object

Returns:

None

add_course(self, course_name, year, term)

Function that adds a course from the course object from self.student plan with position being determined by inputted "year" and "term"

Parameters:

course_name: String name of the course to add to the student plan

year: Int value that indicates the year in which to add the course in self.plan

1 = 1st year

2 = 2nd year

…

term: Int value that indicates the term in which to find the course in self.plan

0 = Fall

1 = Winter

2 = Spring

3 = Summer

Returns:

None

remove_course(self, course_name, year, term)

Function that removes a course from the course object from self.student plan with position determined by inputted "year" and "term"

Parameters:

self

course_name: String name of the course to add to the student plan

year: Int value that indicates the year in which to add the course in self.plan

1 = 1st year

2 = 2nd year

…

term: Int value that indicates the term in which to find the course in self.plan

0 = Fall

1 = Winter

2 = Spring

3 = Summer

Returns:

None

get_plan(self)

Function to generate a possible plan for the student to complete all of the requirements for the degrees they are taking.

Calls:

degree_planning.generate_plan()

Parameters:

None

Returns:

plan: Dictionary of plan information in the following form

{1: [[course_1, course_2, ...], [course_3, course_4, ...], [], []],

2: [[], [], [], []],

3: [[], [], [], []],

4: [[], [], [], []],

…

}

Note: course objects are stored within the lists.

key:value pairs represent year:courses_taken_that_year, where the inner lists represent each term within that year. The ordering is Fall, Winter, Spring, Summer.

get_course_list(self)

Function to return a list of the string names of all course objects that are within the degrees stored within self.degree_list

Parameters:

None

Returns:

course_list: List of course names

==========

Term Object

==========

__init__(self, term_name)

Function to initialize a Term class object

Parameters:

term_name: String input of the term

Options include: “Fall”, “Winter”, “Spring”, “Summer”

Returns:

None

==========

Course Object

==========

__init__(self, name, course_num, num_credits, pre_reqs, terms, difficulty)

Function to initialize a Course class object

Parameters:

name: String input of the unique name for the course object

course_num: Int value of the identifier number for the course

num_credits: Int value of the number of credits the University assigns to the course

pre_reqs: List of Course objects that are required to be taken by a Student before this one

terms: List of Term objects

difficulty: Int value representing how difficult the course is on a scale of 1 to 5

Returns:

None

==========

Degree Object

==========

__init__(self, name)

Function to initialize a Degree class object

Parameters:

name: String identifier of Degree object

Returns:

None

calc_pre_req_nums(self)

Function that calculates (and sets the integer value for) the number of courses that each course in the Degree object relies upon (is a pre-req for)

Parameters:

None

Returns:

None

add_course(self, name, course_num, num_credits, pre_reqs, terms, is_core, difficulty)

Function for creating a new Course object and adding it to the degree

Note: Course must have all pre-req's already added to the degree

Calls:

Course(name, course_num, num_credits, pre_reqs, terms, difficulty)

Parameters:

name: String input of the unique name for the course object

course_num: Int value of the identifier number for the course

num_credits: Int value of the number of credits the University assigns to the course

pre_reqs: List of course names that are required to be taken by a Student before this one

terms: List of Term objects

is_core: Boolean indicator denoting if the course is required for a student to take in order to complete the major. Defaults to False

difficulty: Int value representing how difficult the course is on a scale of 1 to 5

Returns:

None

remove_course(self, name)

Function for removing the Course object with target name from the Degree object

Parameters:

name: String input of the unique name for target Course object

Returns:

None

get_course(self, target_course_name)

Function to get the Course object in the Degree object that has name == target_course_name

Parameters:

name: String input of the unique name for target Course object

Returns:

course: Course object of target course

==========

Degree Planning

==========

generate_plan(student)

Function for generating a forecast degree plan for a student to meet all requirements of their degrees

Calls:

sort_pre_req(course)

add_course_to_forecast(plan, unmet_courses, current_term, student)

Parameters:

student- Student object of the student to generate the plan for

Returns:

plan: Dictionary of plan information in the following form

{1: [[course_1, course_2, ...], [course_3, course_4, ...], [], []],

2: [[], [], [], []],

3: [[], [], [], []],

4: [[], [], [], []],

…

}

Note: course objects are stored within the lists.

key:value pairs represent year:courses_taken_that_year, where the inner lists represent each term within that year. The ordering is Fall, Winter, Spring, Summer.

sort_pre_req(course)

Function for specifying the parameter to sort a course list by. This function gives a sort key of course.pre_reqs_num

Parameters:

course: Course object

Returns:

pre_req_num: Int value of course.pre_req_num

add_course_to_forecast(plan, unmet_courses, current_term, student)

Function adding the courses from unmet_courses into the forecast_plan

Calls:

get_next_course(forecasted_courses_taken, unmet_course_list, current_term)

increment_year(plan, year)

Parameters:

plan: Dictionary of plan information in the following form

{1: [[course_1, course_2, ...], [course_3, course_4, ...], [], []],

2: [[], [], [], []],

3: [[], [], [], []],

4: [[], [], [], []],

…

}

Note: course objects are stored within the lists.

key:value pairs represent year:courses_taken_that_year, where the inner lists represent each term within that year. The ordering is Fall, Winter, Spring, Summer.

unmet_courses: List of Course objects that need to be added to the plan

current_term: Tuple of the term the student is entering (year, term)

student: Student object forecast is being performed for

Returns:

None

get_next_course(forecasted_courses_taken, unmet_course_list, current_term)

Function for determining what course should be added to a student plan next

Parameters:

forecasted_courses_taken: List of Course objects that are currently in the students plan for previous term

unmet_course_list: List of Course objects that still need to be taken by the student

current_term: Int value representing the term next_course is going to be taken in

0 = Fall

1 = Winter

2 = Spring

3 = Summer

Returns:

next_course: Course object of the next course that should be taken

increment_year(plan, year)

Function incrementing the year key being used to index through a degree plan. If the incremented key is out of range of the keys within the degree plan then a new year gets added to the degree plan

Parameters:

plan: Dictionary of plan information in the following form

{1: [[course_1, course_2, ...], [course_3, course_4, ...], [], []],

2: [[], [], [], []],

3: [[], [], [], []],

4: [[], [], [], []],

…

}

Note: course objects are stored within the lists.

the key:value pairs represent year:courses_taken_that_year, where inner lists represent each term within that year.

The ordering is Fall, Winter, Spring, Summer.

year: Integer key for plan dictionary

Returns:

None

print_plan(plan)

Function for printing a degree plan to the terminal (Used for progress testing)

Parameters:

plan: Dictionary of plan information in the following form

{1: [[course_1, course_2, ...], [course_3, course_4, ...], [], []],

2: [[], [], [], []],

3: [[], [], [], []],

4: [[], [], [], []],

…

}

Note: course objects are stored within the lists.

key:value pairs represent year:courses_taken_that_year, where the inner lists represent each term within that year.

The ordering is Fall, Winter, Spring, Summer.

year: Integer key for plan dictionary

Returns:

None

==========

Pickling

==========

save_record(student_id, student_object)

Creates a new pickle file of the name <student_id>

Parameters:

student_id: String representation of the 95 number of the student

Student_object: Student class object

Returns:

None

load_record(student_id)

Returns a Python object (in this case, a Student) from the file named <student_id> if it exists and None otherwise

Parameters:

student_id: String representation of the 95 number of the student

Returns:

Student: class object previously stored in a pickle file

OR

None

delete_record(student_id)

Deletes the file named <student_id> in the pickles directory if it exists

Parameters:

student_id: String representation of the 95 number of the student

Returns:

None

create_studentID_list()

Returns a list of all pickle file names (as strings) in pickles/ directory

Parameters:

None

Returns:

List: string list of pickle file names from the pickles/ directory

==========

JavaScript

==========

toggleLogin(name)

On users' input, restructures DOM to present main functions used to create a four-year plan.

Parameters:

name: user's student Id number to be displayed

Returns:

None

toggleLoginNewUser()

Collects value from input of new user's Id and passes it to toggle Login.

Parameters:

None

Returns:

None

addClass()

Creates Table row and adds the user's current selection of Course, Term and Year to that row

Parameters:

None

Returns:

None

removeClass()

Finds and deletes Row containing the Course, Term and Year selected from the table

Parameters:

None

Returns:

None

saveTable()

Collects the data within the table to a List of Lists & clears table of data

Parameters:

None

Returns:

List: A List of Lists containing 3 indexes: Course, Term, Year (in that order)

saveID()

Gets the input value the user enters for their student Id

Parameters:

None

Returns:

String: string representation of the student Id

showAlert(message, alerttype)

Creates and inserts Bootstrap Alerts into HTML for 3 seconds, then removes it

Parameters:

message: String message to be displayed in Alert

alerttype: String of type that will create alert based on what is put into

parameter (possible alerttypes are defined in Bootstrap Documentation)

Returns:

Div: Creates alert box in HTML lasting 3 seconds before being removed

existingUser(name)

Sends existing user ID data to flask using Ajax

Parameters:

String name: the students Id number in string format

Returns:

None

==========

Flask

==========

@app.route("/")

@app.route("/index", methods=['POST'])

def index()

Main routing function for rendering ui.html. Passes a static list of terms and years, a variable list of pickle file names, and a list of courses

Parameters:

None

Returns:

renders ui.html

@app.route("/forecast", methods=['POST'])

def forecast()

Main routing function for rendering forecast.html, contains logic for rerouting based on user input

Parameters:

None

Returns:

renders forecast.html

format_rows_to_columns(forecast_dict)

Helper function to reformat list of courses from Student object's

self.plan (student_object.py) into a format appropriate for forecast.html

i.e., rows from student object will be converted into columns for forecast.html

Parameters:

forecast_dict: dictionary of plan information in the following form:

{1: [[course_1, course_2, ...], [course_3, course_4, ...], [], []],

2: [[], [], [], []],

3: [[], [], [], []],

4: [[], [], [], []],

...

}

Returns:

List: 3-Layer deep list (i.e. A list containing lists which contain lists)

example = [

[["CIS 210", "CIS 211", "CIS 212", ""], ["CIS 110", "CIS 111", "CIS 199", ""]],

[["CIS 313", "CIS 315", "CIS 425", ""], ["CIS 314", "MATH 343", "CIS 471", ""]]

]

This example list represents a possible course-plan for the 1st and 2nd year,

where Year 1 Fall Term the student is taking CIS 210 and CIS 110

Year 1 Winter Term the student is taking CIS 211 and CIS 111

Year 1 Spring Term the student is taking CIS 212 and CIS 199

Year 1 Summer Term the student is taking no courses

Year 2 Fall Term the student is taking CIS 313 and CIS 314

Year 2 Winter Term the student is taking CIS 315 and MATH 343

Year 2 Spring Term the student is taking CIS 425 and CIS 471

Year 2 Summer Term the student is taking no courses

gen_courses()

Helper function for populating the course dropdown by creating a default student object

Parameters:

None

Returns:

List: All courses in a Gen_Ed degree and CIS degree object

==========

Default Degree Objects

==========

create_CIS_major()

Function created to instantiate the CIS_major Degree object in the back end which specifies all courses required of CIS majors at the University of Oregon. This was created to get the system working in the narrow scope of *only* handling CIS majors at UO. In order to generalize the system for many majors/minors there are two obvious options: following this format to manually create all of the desired major/minor objects in the back end, or generalize the system by allowing the user to create their own Degree object from a drop-down menu of all available courses at their University.

Parameters:

None

Returns:

Degree object

Create_Gen_Ed()

Function created to instantiate the General Education (Gen_Ed) Degree object in the back end which specifies any courses required of all majors at the University of Oregon.

Parameters:

None

Returns:

Degree object