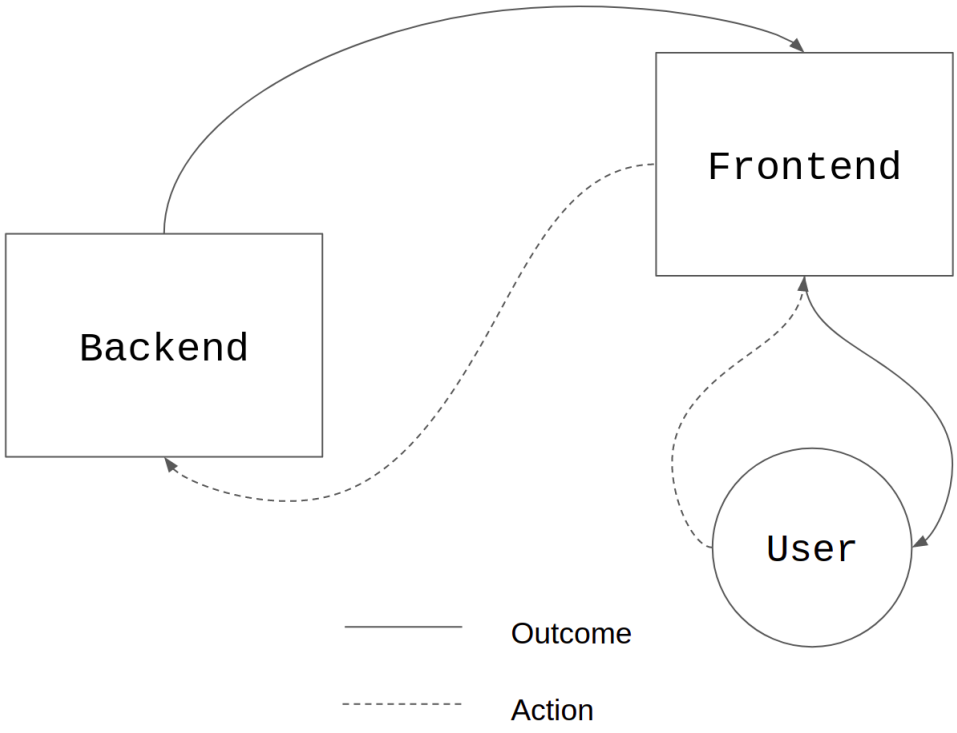
IIT Madras
BSc Degree

Home Lesson-6.5

Lesson-6.4

Dictionaries in Action: LMS

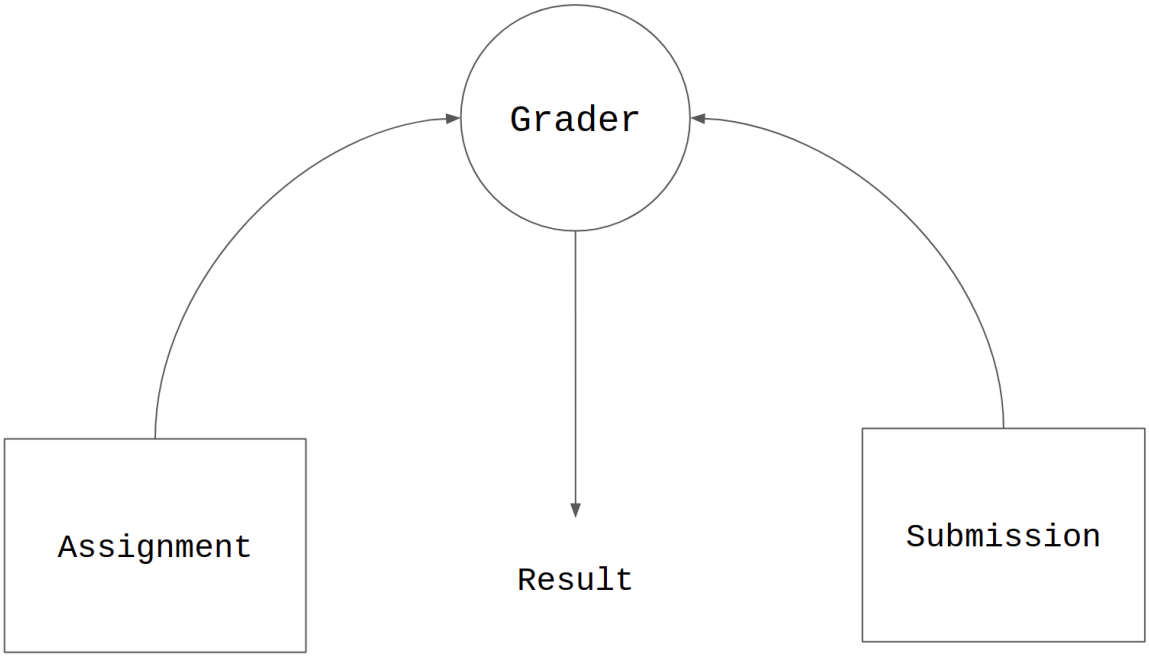
The online degree portal — our virtual classroom — is called a learning Management system (LMS). In more accessible terms, an LMS is the software application that powers the portal. Have you ever wondered how your assignment submissions get recorded and graded? This is the question that we will try to answer in this lesson. At a high level, the LMS is made of two components: frontend and backend.



As a user, your communicate with the frontend. The frontend is the website where you see all the content displayed. When you make an action, say clicking the submit button in a graded assignment, that action is fed to the backend as input. The backend processes this input and returns some output to the frontend, which is then displayed as the outcome of your action.

Where does Python come into the picture? It features prominently in the backend.

So how do we expect grading to work? It needs two inputs. The assignment and the submission corresponding to this assignment. It will return the result as output:



The grader can be expressed as a function:

```
def grader(assignment, submission):
    """Grading logic"""
    result = 0.0
    return result
```

The function is incomplete. We need to decide how an assignment and its corresponding submission are going to be modeled.

Assignment Model

Let us consider an assignment. It is essentially a list of problems. So, modeling an assignment breaks down to modeling a problem. A problem could have the following attributes:

Attribute	Type
id	string
question	string
type	string
options	list
answers	tuple
marks	float

For grading, we only need two attributes, the problem-id and the answers. With this, the assignment model will look like the following. The entire assignment will now be a list of dictionaries:

```
# assume that the assignment has three problems
# the assignment will be a list of dictionaries
assignment = [
                {'id': '10001', 'answers': (0, 1), 'marks': 2.0},
                {'id': '10002', 'answers': (1, ), 'marks': 1.0 },
                {'id': '10003', 'answers': (2, ), 'marks': 2.0}
             ]
```

A point to note. A singleton tuple is represented as `(<item>, )`. The comma cannot be ignored. Coming back to the assignment model, we see that there are several attributes in the table that haven't entered into the assignment dictionary since they are not relevant from the point of view of grading. They have been mentioned so that it gives a better understanding of how assignments can be modeled.

Submission Model

The submission model is slightly more involved. There are some global attributes like name of the user, the user's roll number and the time of submission. And then there are local attributes like the options selected for each problem.

Attribute	Type
name	string
roll_number	string
timestamp	string
problems	list

Let us look at a sample submission:

```
submission = {
                'name': 'Kapil Dev',
                'roll_number': 'BSC1001',
                'time': 'Sunday 18 April 2021 10:23:30 PM IST',
                'problems': [
                                {'id': '10001', 'selected': (0, 1)},
                                {'id': '10002', 'selected': (1, )},
                                {'id': '10003', 'selected': (3, )}

                            ]
             }
```

`submission` is a fairly complicated object. To begin with, it is a dictionary. The first three keys do not pose any challenges. The value of the key `'problems'` is a list of dictionaries! We could add one more level of complexity. Since a user could make multiple submissions, we could have a list of submissions! But for now, let us not complicate things any further.

Grader

The assignment is a list of dictionaries. While this is not a bad representation, the grader has to search for the problem id through this list every time it has to grade a problem. Since the problem id is unique, we can come up with a better representation for the assignment:

```
assignment_ = [
                {'id': '10001', 'answers': (0, 1), 'marks': 2.0},
                {'id': '10002', 'answers': (1, ), 'marks': 1.0 },
                {'id': '10003', 'answers': (2, ), 'marks': 2.0}
            ]
assignment = dict()
for problem in assignment_:
    problem_id = problem['id']
    answers = problem['answers']
    marks = problem['marks']
    assignment[problem_id] = {'answers': answers, 'marks': marks}
```

The assignment now looks like this:

```
assignment = {
                '10001': {
                            'answers': (0, 1),
                            'marks': 2.0
                        },
                '10002': {
                            'answers': (1, ),
                            'marks': 1.0
                        },
                '10003': {
                            'answers': (2, ),
                            'marks': 2.0
                        },
            }
```

We are now ready to complete the grader using this new assignment model:

```
def grader(assignment, submission):
    """Grading logic"""
    result = 0.0
    for problem in submission['problems']:
        problem_id = problem['id']
        selected = problem['selected']
        answers = assignment[problem_id]['answers']
        if answers == selected:
            result += assignment[problem_id]['marks']
    return result
```