CSCA08 AMACSS Review Seminar

Tabeeb Yeamin and Mohammad Moustafa

Dictionaries

`{key: value}`

- Keys must be an immutable type, like string, int, tuple etc.
- `{[1,2,3]: ‘a’}` will fail.
- Value can be any type, can also be lists, sets, dicts etc.
- `{‘a’: [1,2,3], ‘b’: {1:(2,2,2)}}`

Dictionaries

Initialization:

- `d = {}`
- `d = dict()` initializing set => `set()`

Dictionaries are NOT ordered! (Just like sets)

- cannot do d[1]

Dictionaries

```
name_to_age = {'Alice': 30, 'Bob': 15, 'Carol': 45,
'David': 25, 'Ed': 35}
```

Access the values by using [key] or .get(key) method:

● `name_to_age['Alice']` => 30	invalid key gives error
● `name_to_age.get('Alice')` => 30	invalid key returns None
● `name_to_age.get('Alice', -1)` => 30	invalid key returns -1

Dictionaries

```
name_to_age = {‘Alice’: 30, ‘Bob’: 15, ‘Carol’: 45,
‘David’: 25, ‘Ed’: 35}
```

- You can use elemental for loops to loop through the keys

  ```
  for name in name_to_age:
      print(name_to_age[name])
  ```
  <= prints the ages

- You can also get list of the keys by doing:

  ```
  name_to_age.keys()
  ```

  Returns dict keys object, a “list-like” object, but it does not support indexing.

Files

`open(filename, mode)` : a function that returns a filehandle object

(str, str) -> (io.TextIOWrapper)

mode:

- `'r'`: reading
- `'w'`: writing (erases previous data)
- `'a'`: appending

Close the filehandle object by doing:

```
filehandle.close()
```

Files (Reading)

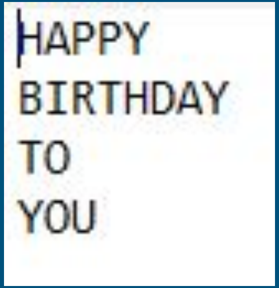
`filehandle.readline()` : 1 line from the file

`filehandle.read()`: reads whole file into a single string

`filehandle.readlines()`: reads whole file into a list (each element is one line of text)

Files Example

HBD.txt has:



```
HAPPY
BIRTHDAY
TO
YOU
```

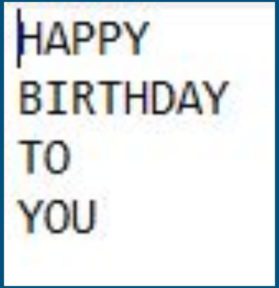
```
filehandle.readline()
```

```
filehandle.read()
```

```
filehandle.readlines()
```

Files Example

HBD.txt has:



```
HAPPY
BIRTHDAY
TO
YOU
```

`filehandle.readline()`

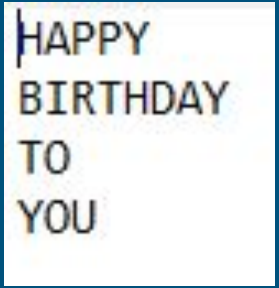
=> “HAPPY\n” (the \n will create a new line)

`filehandle.read()`

`filehandle.readlines()`

Files Example

HBD.txt has:



```
HAPPY
BIRTHDAY
TO
YOU
```

`filehandle.readline()`

=> “HAPPY\n” (the \n will create a new line)

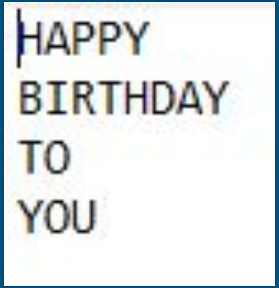
`filehandle.read()`

=> “HAPPY\n BIRTHDAY\nTO\nYOU”

`filehandle.readlines()`

Files Example

HBD.txt has:



```
HAPPY
BIRTHDAY
TO
YOU
```

`filehandle.readline()`

=> “HAPPY\n” (the \n will create a new line)

`filehandle.read()`

=> “HAPPY\n BIRTHDAY\nTO\nYOU”

`filehandle.readlines()`

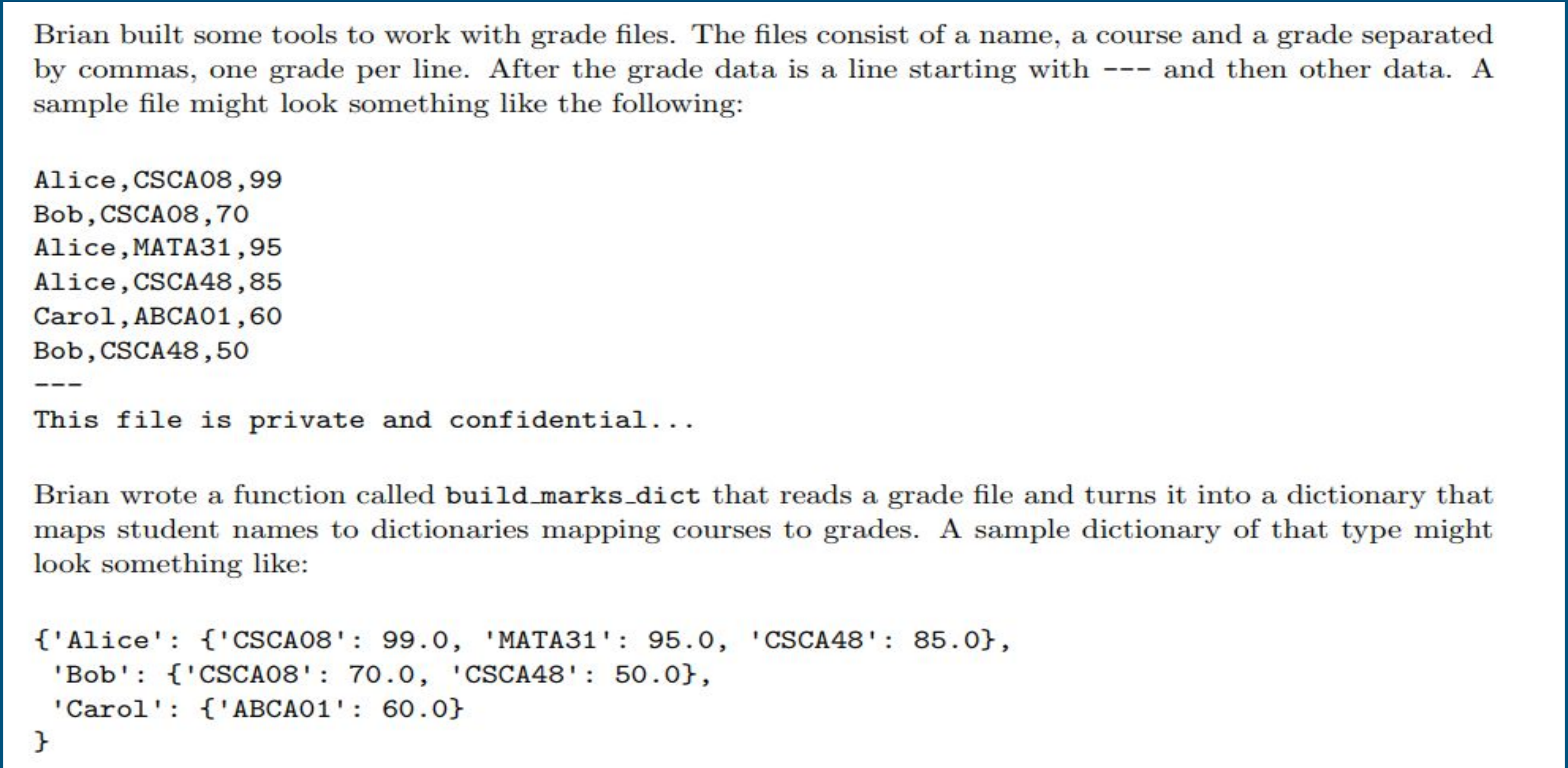
=> [“HAPPY\n”, “BIRTHDAY\n”, “TO\n”, “YOU”]

Files (Writing)

`filehandle.write():`

- Only takes in strings
- To create new lines you must include `“\n”`
- You can also use `“\t”` to tab
- Make sure to close your file afterwards, or it may not write

File + Dictionaries Example (2017 A08 TT2)

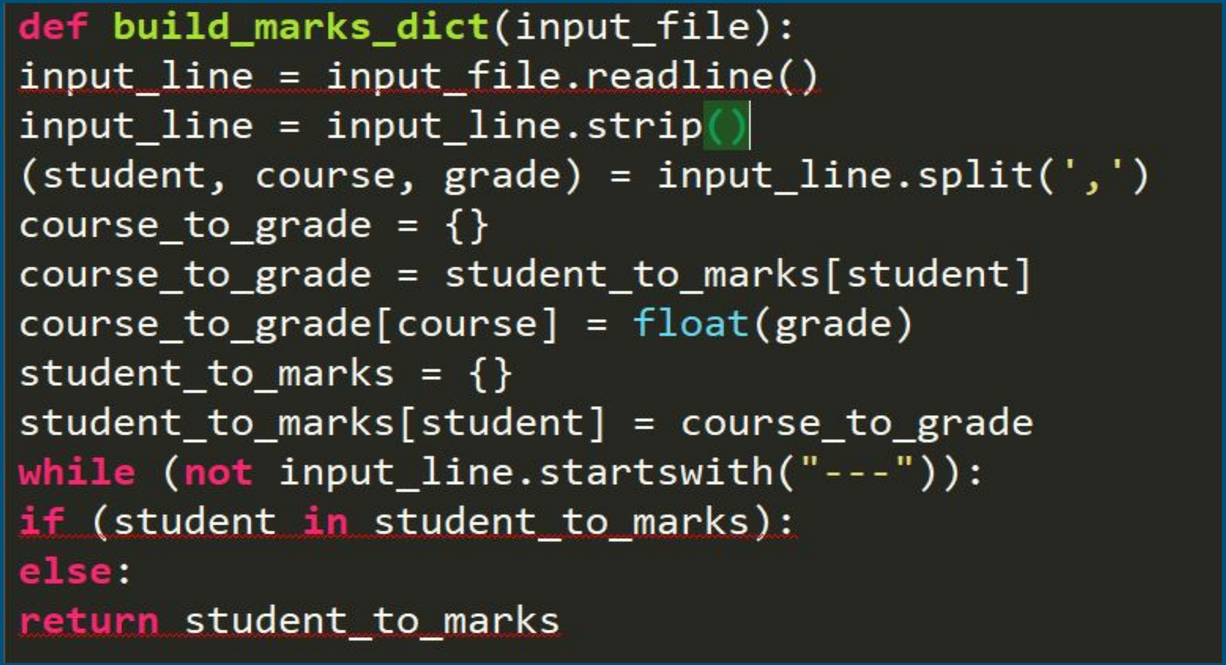


Brian built some tools to work with grade files. The files consist of a name, a course and a grade separated by commas, one grade per line. After the grade data is a line starting with --- and then other data. A sample file might look something like the following:

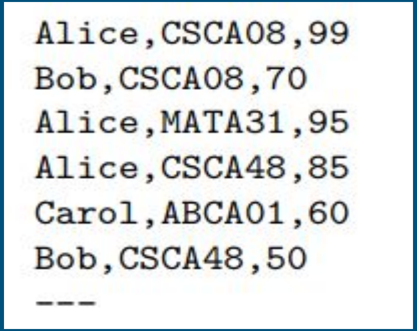
```
Alice,CSCA08,99
Bob,CSCA08,70
Alice,MATA31,95
Alice,CSCA48,85
Carol,ABCA01,60
Bob,CSCA48,50
---
This file is private and confidential...
```

Brian wrote a function called `build_marks_dict` that reads a grade file and turns it into a dictionary that maps student names to dictionaries mapping courses to grades. A sample dictionary of that type might look something like:

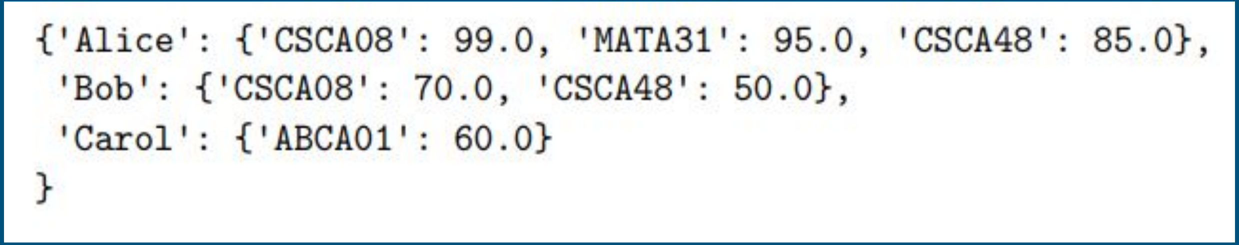
```
{'Alice': {'CSCA08': 99.0, 'MATA31': 95.0, 'CSCA48': 85.0},
 'Bob': {'CSCA08': 70.0, 'CSCA48': 50.0},
 'Carol': {'ABCA01': 60.0}
}
```



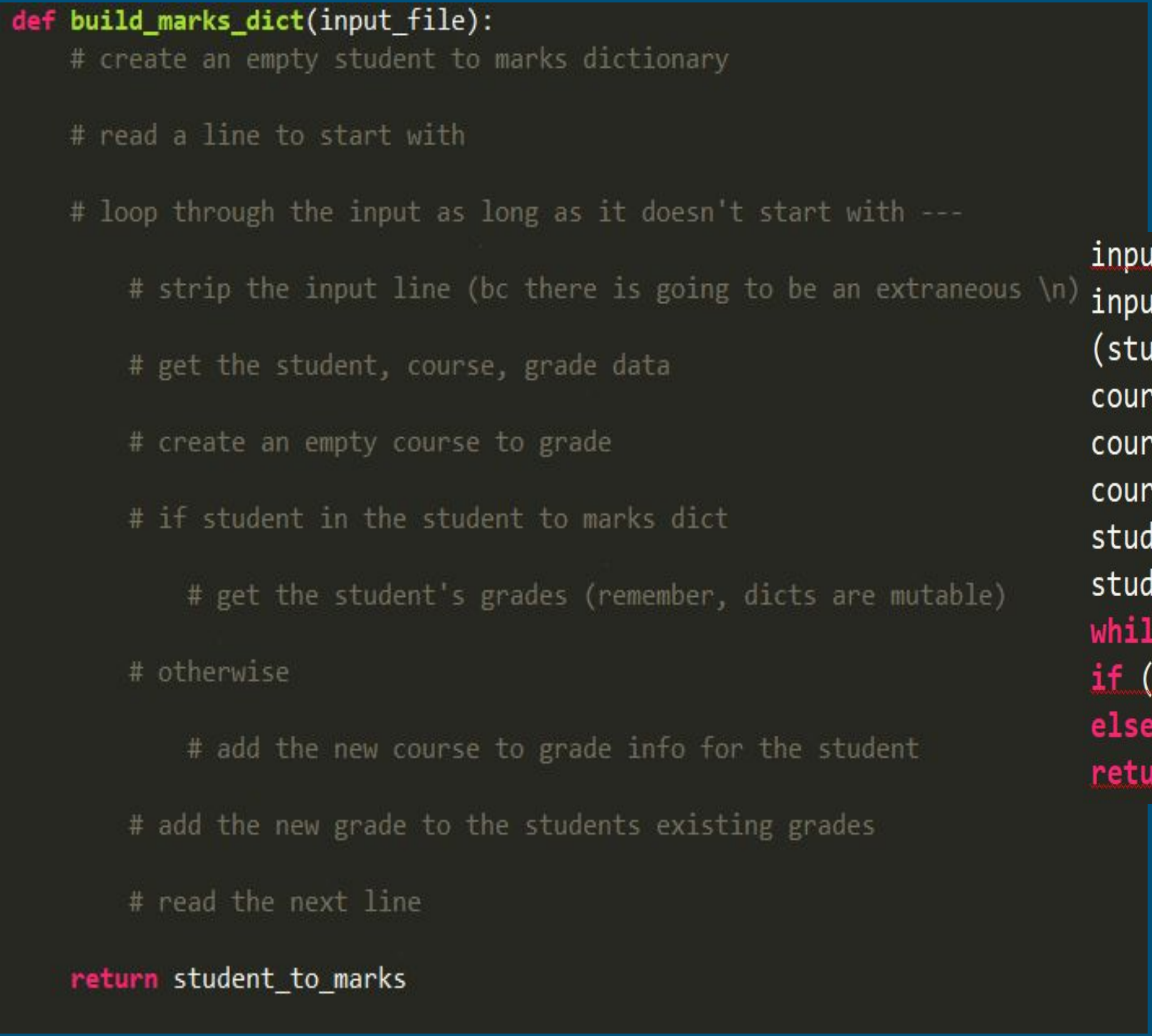
```
def build_marks_dict(input_file):
input_line = input_file.readline()
input_line = input_line.strip()
(student, course, grade) = input_line.split(',')
course_to_grade = {}
course_to_grade = student_to_marks[student]
course_to_grade[course] = float(grade)
student_to_marks = {}
student_to_marks[student] = course_to_grade
while (not input_line.startswith("---")):
if (student in student_to_marks):
else:
return student_to_marks
```



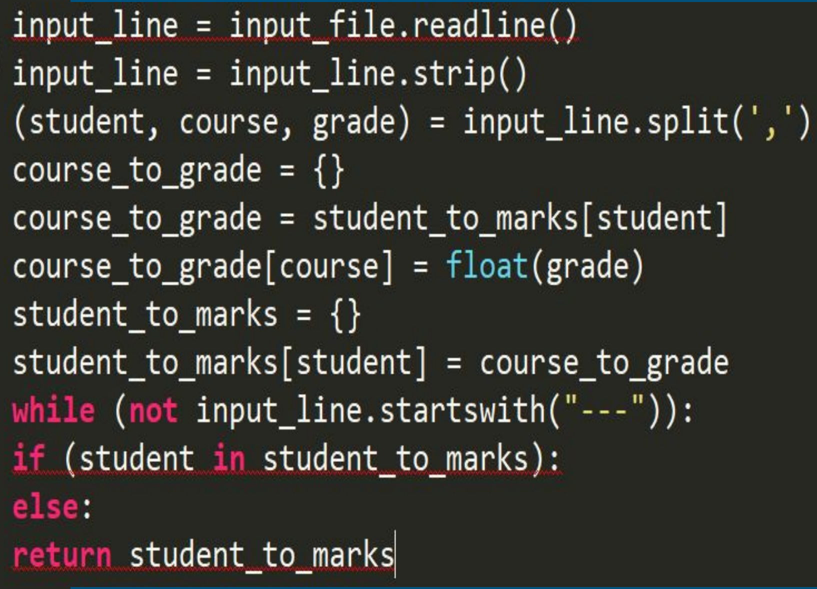
```
Alice,CSCA08,99
Bob,CSCA08,70
Alice,MATA31,95
Alice,CSCA48,85
Carol,ABCA01,60
Bob,CSCA48,50
---
```



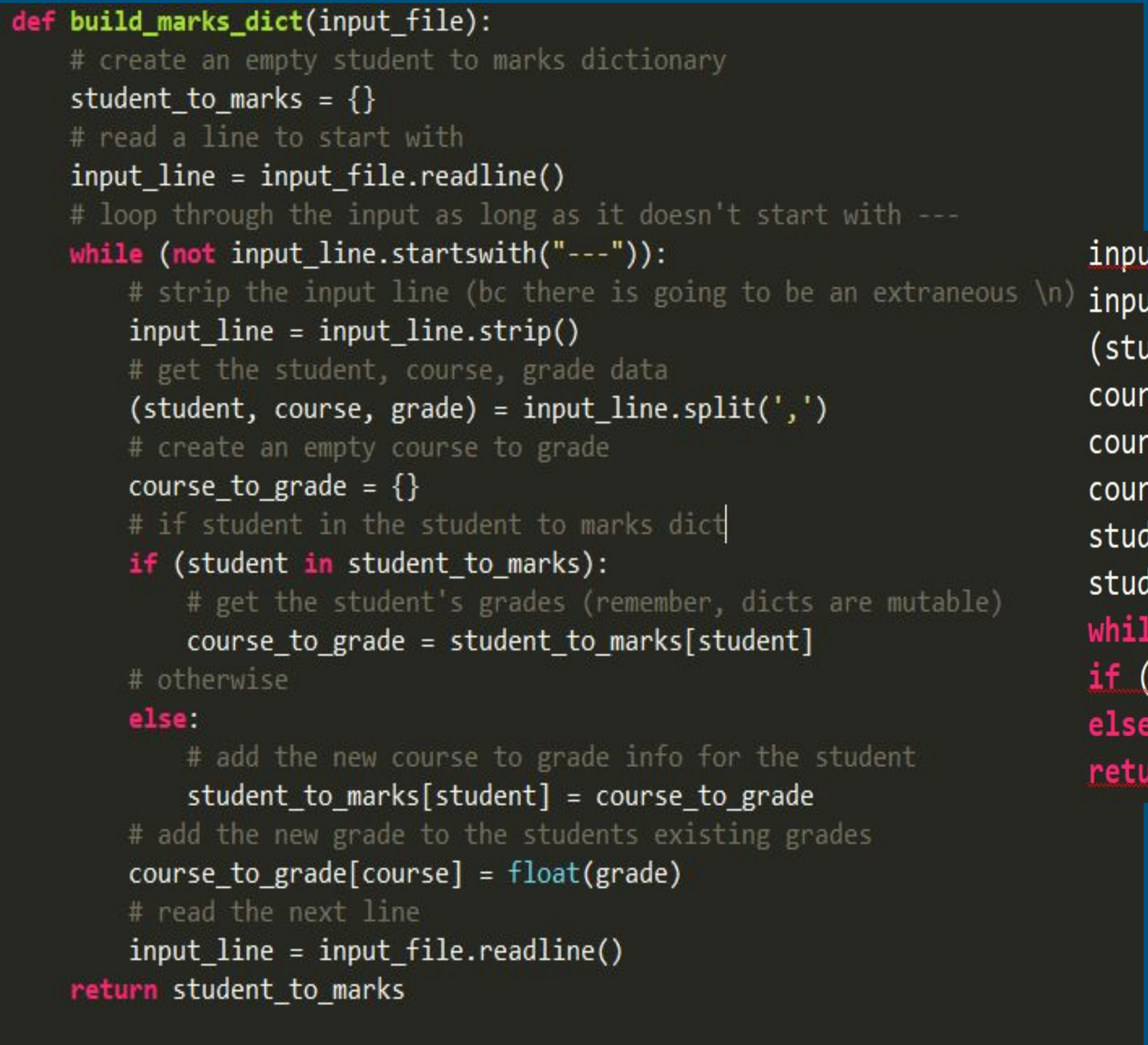
```
{'Alice': {'CSCA08': 99.0, 'MATA31': 95.0, 'CSCA48': 85.0},
 'Bob': {'CSCA08': 70.0, 'CSCA48': 50.0},
 'Carol': {'ABCA01': 60.0}
}
```



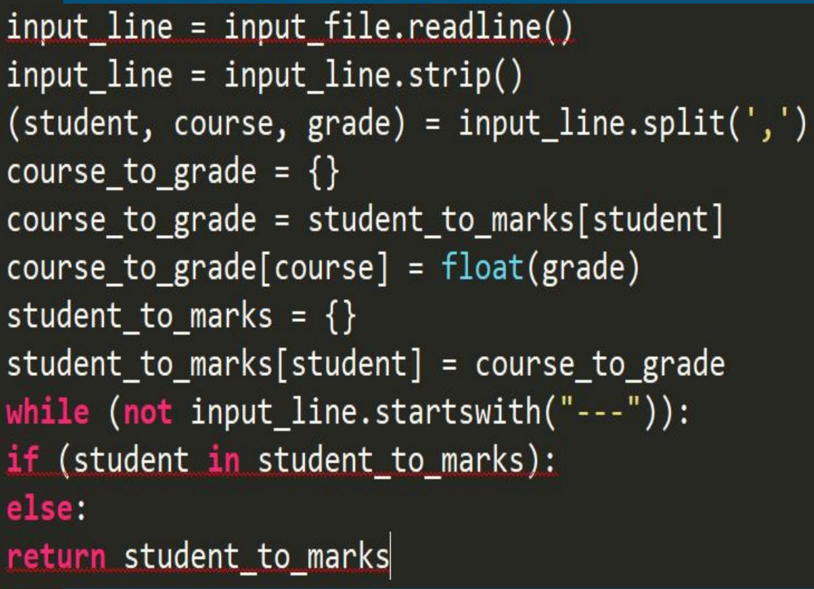
```python
def build_marks_dict(input_file):
    # create an empty student to marks dictionary

    # read a line to start with

    # loop through the input as long as it doesn't start with ---

        # strip the input line (bc there is going to be an extraneous \n)

        # get the student, course, grade data

        # create an empty course to grade

        # if student in the student to marks dict

            # get the student's grades (remember, dicts are mutable)

        # otherwise

            # add the new course to grade info for the student

        # add the new grade to the students existing grades

        # read the next line

    return student_to_marks
```



```python
input_line = input_file.readline()
input_line = input_line.strip()
(student, course, grade) = input_line.split(',')
course_to_grade = {}
course_to_grade = student_to_marks[student]
course_to_grade[course] = float(grade)
student_to_marks = {}
student_to_marks[student] = course_to_grade
while (not input_line.startswith("---")):
if (student in student_to_marks):
else:
return student_to_marks
```



```python
def build_marks_dict(input_file):
    # create an empty student to marks dictionary
    student_to_marks = {}
    # read a line to start with
    input_line = input_file.readline()
    # loop through the input as long as it doesn't start with ---
    while (not input_line.startswith("---")):
        # strip the input line (bc there is going to be an extraneous \n)
        input_line = input_line.strip()
        # get the student, course, grade data
        (student, course, grade) = input_line.split(',')
        # create an empty course to grade
        course_to_grade = {}
        # if student in the student to marks dict
        if (student in student_to_marks):
            # get the student's grades (remember, dicts are mutable)
            course_to_grade = student_to_marks[student]
        # otherwise
        else:
            # add the new course to grade info for the student
            student_to_marks[student] = course_to_grade
        # add the new grade to the students existing grades
        course_to_grade[course] = float(grade)
        # read the next line
        input_line = input_file.readline()
    return student_to_marks
```



```python
input_line = input_file.readline()
input_line = input_line.strip()
(student, course, grade) = input_line.split(',')
course_to_grade = {}
course_to_grade = student_to_marks[student]
course_to_grade[course] = float(grade)
student_to_marks = {}
student_to_marks[student] = course_to_grade
while (not input_line.startswith("---")):
if (student in student_to_marks):
else:
return student_to_marks
```

UnitTesting (Number Ranges)

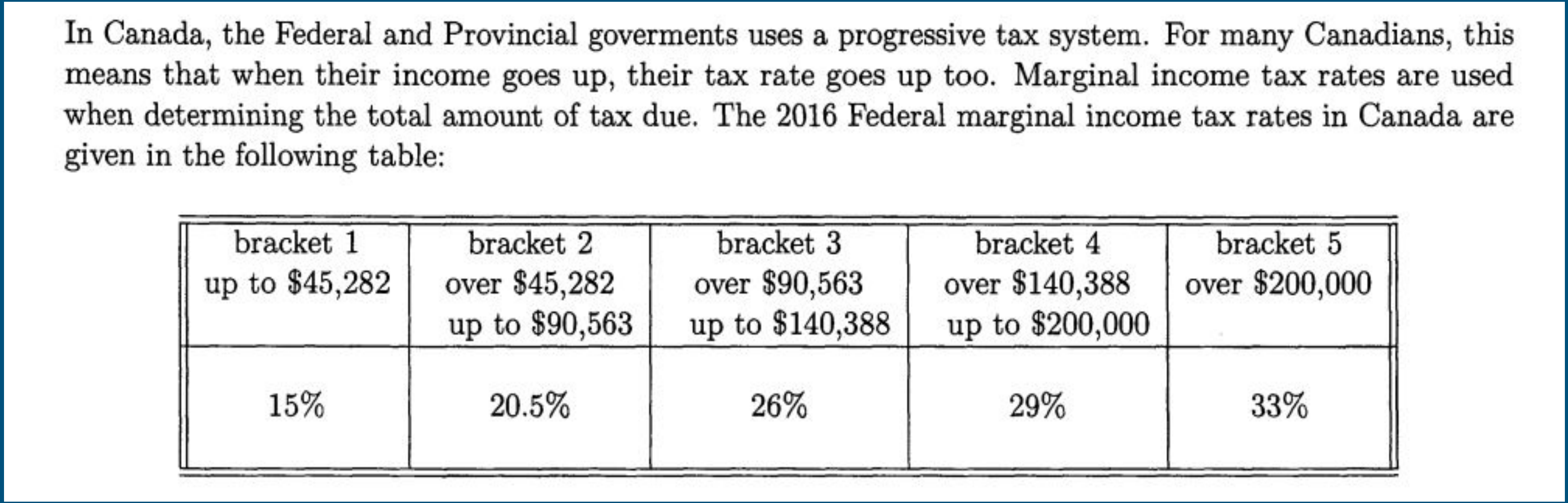
Ex: 0 <= n <= 13

Test the edge cases, a number in between, and then above and below the range

- n = 0
- n = 13
- 0 < n < 13
- n > 13
- n < 0 (If valid input)

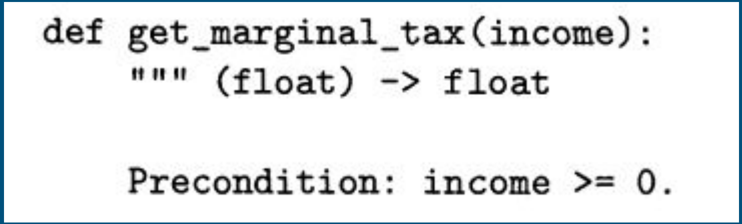
You don’t need to test invalid cases. I.e. if n refers to age, n >= 0 should be a REQ.

UnitTesting Ex 1 (April 2017 Final)

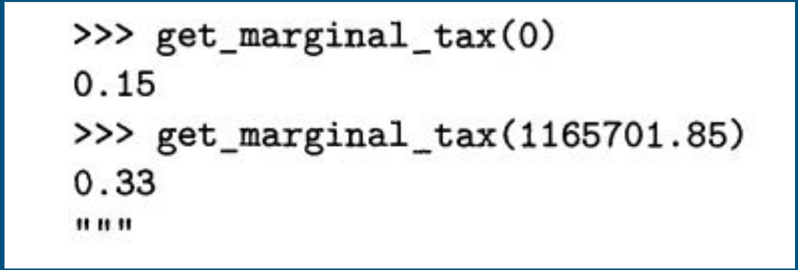


In Canada, the Federal and Provincial goverments uses a progressive tax system. For many Canadians, this means that when their income goes up, their tax rate goes up too. Marginal income tax rates are used when determining the total amount of tax due. The 2016 Federal marginal income tax rates in Canada are given in the following table:

bracket 1 up to \$45,282	bracket 2 over \$45,282 up to \$90,563	bracket 3 over \$90,563 up to \$140,388	bracket 4 over \$140,388 up to \$200,000	bracket 5 over \$200,000
15%	20.5%	26%	29%	33%



```
def get_marginal_tax(income):
    """ (float) -> float

    Precondition: income >= 0.
```



```
>>> get_marginal_tax(0)
0.15
>>> get_marginal_tax(1165701.85)
0.33
"""
```

UnitTesting Ex 1 (April 2017 Final)

Test Case Description	Income ($)	Return value
0	0	0.15
In bracket 1	20, 000	0.15
Bracket 1 upper edge case	45, 282	0.15
In bracket 2	60, 000	0.205
Bracket 2 upper edge case	90, 563	0.205
In bracket 3	110, 000	0.26
Bracket 3 upper edge case	140, 388	0.26
In bracket 4	160, 000	0.29
Bracket 4 upper edge case	200, 000	0.29
Bracket 5	201, 000	0.33

UnitTesting Strings/Lists/Dicts etc.

- Empty
- One character/element
- More than one character/element (May have to divide further depending on the question)

UnitTesting Strings Example

```
def is_palindrome(string: str) -> bool:
    ‘’’ Returns True iff string is a palindrome
   Precondition: 0 <= len(string) <= 3
    ‘’’
```

UnitTesting Strings Example

len(string)	strings
0	“”
1	“a”
2	“aa”, “ab”
3	“aaa”, “aba”, “abb”, “aab”, “abc”