TUTORIAL 9

DICTIONARIES AND CLASSES
REMINDER

• Assignment 07 is due Wednesday, July 17th at 10 AM
  – Please come to office hours EARLY!!! 😊
REVIEW

• Dictionaries
• Classes
  - __init__
  - __repr__
  - __eq__
  - class methods
d = {key1:value1, key2:value2, ...}

- Each element has a key (a way to look up info) and a value associated with the key
- Unordered collection (with each element being a key-value pair)
- Like a REAL dictionary (a real dictionary is a word-definition pair; word = key, definition = value)
USEFUL DICTIONARIES FUNCTIONS

- $d[k]$ → Get the value of $k$
- $d[k]=v$ → Add key-value pair or change value to be $v$ if $k$ already exists in $d$
- $d.keys()$ → Creates a view of all the keys in $d$
- $d.values()$ → Creates a view of all the values in $d$
- $d.pop(k)$ → Removes key-value pair of $k$ from $d$ and returns the value associated with $k$
- $k in d$ → returns $True$ if $k$ is a key in $d$
CLASSES

• Python’s version of Racket structures
• Allows related information to be grouped together
• We’ll use __init__, __repr__, and __eq__ with the class
• We'll also write new class methods
class name:

def __init__(self, f1, f2, ...):
    self.field1 = f1
    self.field2 = f2
    ...
    ...

• Creates an object of this class:
    
x = name(field1_val, field2_val, ...)

• Call the fields by:  x.field1

• Racket’s version:
  (define-struct name (field1_val field2_val ...))
  (name-field1 x)
If we try to print a class object, we’d get something like

```python
<__main__.name instance at 0x12361c0>
```

We can print a more informative message using the `__repr__` command within the class definition

```python
def __repr__(self):
    return "name: {0},{1},…"\n        .format(self.field1,
                self.field2,…)```

Think of `__repr__` as "represents"

Very similar to `__str__`
def __eq__(self, other):
    return isinstance(other, name) and \
    self.field1 == other.field1 and \
    self.field2 == other.field2 and \
    ...            ...

• It will allow you to compare objects to see if they have same fields:

  \[ x == y \Rightarrow \text{True} \]
class name:
    def __init__(self, f1, f2, ...):...
    def __repr__(self):...
    def __eq__(self, other):...

    def foo(self, ...):
        # Access field values: self.field1, ...
        # fn may update field values, use field values
        # for calculations, print information, or
        # return information
Write a function `list_multiples` that consumes a string `s` and returns a list in *alphabetical order* containing every character in `s` that appears more than once. Use dictionaries.

Examples:

```
list_multiples("abcd") => []
list_multiples("bacaba") => ["a", "b"]
list_multiples("gtddyucaadsa") => ["a", "d"]
```
Write a function `xor` that consumes two dictionaries (d1 and d2) and returns a dictionary.

The returned dictionary will contain all the keys that appear in exactly one of d1 or d2 (but not both).

The value associated with each key will be the same as the one found in the original dictionary.
EXAMPLES

d1 = {1:'a', 2:'b', 3:'c', 4:'d'}
d2 = {5:'e', 6:'f', 7:'g', 8:'h'}

xor(d1,d2) => {1:'a', 2:'b', 3:'c', 4:'d',
5:'e', 6:'f', 7:'g', 8:'h'}


d3 = {5:'q', 6:'l', 7:'c', 8:'e'}

xor(d2,d3) => {}  


d4 = {1:'a', 3:'f', 8:'u', 9:'t'}

xor(d1,d4) => {2:'b', 4:'d', 8:'u', 9:'t'}
The remaining questions will use the following class:

A **Student** is a class with fields **name**, **faculty**, **program**, **year**, and **courses**

- **name** is a non-empty string representing the student’s full name;
- **faculty** is a non-empty string representing the student’s faculty;
  - Full version: e.g. "Environment" rather than "Env"
- **program** is a non-empty string representing the person’s program (or major);
- **year** is a natural number representing the student’s academic year;
- **courses** is a list of strings representing the courses the student is taking in the current term;
EXAMPLES OF STUDENT OBJECTS:


- Dan_W = student("Dan Wolczuk", "Mathematics", "Pure Mathematics", 1, ["MATH 148", "MATH 146", "CS 116"])  

- Logan_S = student("Logan Stanley", "Science", "Chemistry", 1, ["CHEM 120", "MATH 127", "PHYS 111"])
Write a class method `add_courses` in the Student class, which consumes a Student object, `self`, and a list of strings, `courses`. It adds the courses in `courses` to the student’s list of courses and prints a message indicating the number of courses the student is now taking.

Examples:

Paul_S.add_courses(['"HLTH 230"']) will print "Paul Shen is currently taking 6 course(s)." and adds "HLTH 230" to Paul_S.courses

Nicole_V.add_courses([]) will print "Nicole Velocci is currently taking 3 course(s)." and adds nothing to Nicole_V.courses
Write a function `organize_by_year` outside the class, which consumes a list of `Student` objects, `loS`, and returns a dictionary where the keys will be natural numbers associating with the students’ years and its associated values is a list of names of the `Student` in the corresponding year.

Example:
L = [Paul_S, Nicole_V, Dan_W, Logan_S]
organize_by_year(L)
=> {1:["Dan Wolczuk", "Logan Stanley"],
    2:["Paul Shen", "Nicole Velocci"]}

Write a function `is_same_faculty` that consumes a non-empty list of students, `los`, and returns True if all the students belongs in the same faculty. Otherwise, the method returns False.

Example:

```
Mathies = [Nicole_V, Dan_W]

is_same_faculty(Mathies) => True

is_same_faculty([Nicole_V]) => True

is_same_faculty([Paul_S, Logan_S]) => False
```