Reverse WWPP

```
def f1(x, y):
    print(x + y)
    return y + x

lst1 = list(range(5))

class Act:
    def __init__(self, x):
        self.y = x

    def update(self, y):
        self.x = y
        return self.y
    def swap(self, y):
        self.x, self.y = self.x + y, self.y - y
    def __next__(self):
        self.x += self.y
        if self.x > 10:
            raise StopIteration
        return self.x
    def __iter__(self):
        self.x - 1
        return self
```

Code	Python output
`f1(3, 1)`	`4` `4`
`f1([3], [[1]])`	`[3, [1]]` `[[1], 3]`
`lst1[1:3] + [f1([2], [3])]`	`[2, 3]` `[1, 2, [3, 2]]`

`act = Act(1)`
`Act.swap(act, act.update(6))`
`[act.x, act.y]`
	`[7, 0]`

`it = Act(2)`
`print(it.update(3))`
`it.swap(-1)`
`[x * 2 + 1 for x in it]`
	`2`
	`[11, 17]`

An Environmental Disaster

Solution

```
f = lambda x: lambda x: x * 2
x = f(2)(2)
```

Squash the Bugs Version 1!

Each of the functions below have a specific number of bugs. Read through the functions and doctests. First, write down what the **function currently returns** with the input given. After that, write down **each bug you find and how to fix it**. Please do not write down an alternate solution, rather, you should try to find bugs in the solutions provided.
What the code currently returns may or may not be the same as what is shown on the doctests.

Q1 How often?- 3 Bugs

```
def frequency(text):
     """
```

Returns a dictionary containing how often each word in the text appears. Do not worry about punctuation/ multiple whitespaces. Assume split() splits the text string into a list at each whitespace. *Hint: one of the bugs would require adding a line of code with a function call*

```
    >>> str = “bananas and apples and ApPlEs and oranges”
    >>> frequency(text)
    {“bananas”: 1, “and”:3, “apples”:2, “oranges”:1}
    >>> str = “cs cs 88”
    >>> frequency(text)
    {“cs”: 2, “88”:1}
    """
     dict_word = {}
     testList = text.split()
     for word in textList:
          if word in dict_word:
               dict_word[word] = dict_word[word] + 1
               return dict_word
          dict_word[word] = 1
     return dict_word
```

What does the code **currently** return with no changes?
frequency(“cs cs 88”)

Demonstrate the bug:

What are the **3** bugs and how do you fix them?

1.
2.
3.

Currently returns: {“cs” : 2}

Demonstrate the bug: Many solutions, any input with it’s output was acceptable as long as the function did not work as intended:

Example that was given credit: frequency(“we love cs88 cs88 Cs88”): {“we”:1, “love”:1, “cs88”:2}
-incorrect because answer should be {“we”:1, “love”:1, “cs88”:3}

Example that was not given credit: frequency(“abcd”): {“abcd”:1}
-this is incorrect as it does not demonstrate a bug

There are three bugs in this question. The first bug is that the return in the if statement should be removed. Otherwise, the function will stop at the first word that is found a second time. The second one is that dict_word[word] needs to be in an else statement, otherwise, each word will always have a frequency of 1. The third bug is that string comparison does not ignore case, so it is required that there is a text.lower() to make sure that apples and ApPlEs are both treated the same.

PLEASE note that an explanation AND fix was required for full credit and partial credit was given if only one of the two was provided.

Squash the Bugs Version 2!

Each of the functions below have a specific number of bugs. Read through the functions and doctests. First, write down what the **function currently returns** with the input given. After that, write down **each bug you find and how to fix it**. Please do not write down an alternate solution, rather, you should try to find bugs in the solutions provided.
What the code currently returns may or may not be the same as what is shown on the doctests.

Q1 How often?- 3 Bugs
```
def frequency(sentence):
```

```
    """
Returns a dictionary containing how often each word in the text
appears. Do not worry about punctuation/ multiple whitespaces. Assume
split() splits the text string into a list at each whitespace. Hint:
one of the bugs would require adding a line of code with a function
call
    >>> str = “bananas and apples and ApPlEs and oranges”
    >>> frequency(text)
    {“bananas”: 1, “and”:3, “apples”:2, “oranges”:1}
    >>> str = “cs cs 88”
    >>> frequency(text)
    {“cs”: 2, “88”:1}
    """
     wordMap = {}
     words = sentence.split()
     for word in sentence:
          if word in wordMap:
               wordMap[word] = wordMap[word] + 1
               return wordMap
          wordMap[word] = 1
     return wordMap
```

What does the code **currently** return with no changes?
frequency(“cs cs 88”)

Demonstrate the bug:

What are the **3** bugs and how do you fix them?
1.
2.
3.

Currently returns: {"c":2, 's':1, ' ':1}

Demonstrate the bug: Many solutions, any input with it’s output was acceptable as long as the function did not work as intended. Anything, other than a one character string, would have worked as long as the actual and expected outputs were correct.

There are actually 4 bugs in this question. Three were required for full credit. The first bug is that the return in the if statement should be removed. Otherwise, the function will stop at the first word that is found a second time. The second one is that wordMap[word]

needs to be in an else statement, otherwise, each word will always have a frequency of 1. The third bug is that string comparison does not ignore case, so it is required that there is a sentence.lower() to make sure that apples and ApPlEs are both treated the same. The fourth bug is that the for loop should iterate through words, otherwise, the loop iterates through each of the characters in sentence.

PLEASE note that an explanation AND fix was required for full credit and partial credit was given if only one of the two was provided.

Just Bugs...Not Murder Hornets

a. WITH BUGS, necessary corrections in red

```
class Exam():
    course = 'cs88'
    instructors = ['Gerald Friedland', 'Michael Ball']
    tas = ['Alex Kassil', 'Brian Mi', 'Julia Yu', 'Alec Kan', 'Cameron
Malloy', 'Shreya Kannan', 'Sophia Qin', 'Srinath Goli', 'Vandana
Ganesh']
    time = 90
    def __init__(self, name, student_id):
        self.name = name
        self.sid = student_id
        self.score = {}

    def grade(self, points_correct, points_total):
        grade = points_correct/points_total
        self.score[student_id] = grade

    def view_score(self):
        return score

class Question():
    extra_credit = False
    def grade(self, points_correct, points_total):
        if self.extra_credit == False:
            Exam.grade(points_correct, points_total)
        else:
            self.score[self.sid] += points_correct


oski_exam = Exam('Oski Bear', 123456789)
oski_q1 = Question('Oski Bear', 123456789)
oski_q1.grade(24,36)
oski_q1.score
>>> {123456789: 0.6666666666666666}
oski_exam.score
>>> {}
```

b. Solution

```
class Exam():
    course = 'cs88'
    instructors = ['Gerald Friedland', 'Michael Ball']
    tas = ['Alex Kassil', 'Brian Mi', 'Julia Yu', 'Alec Kan', 'Cameron
Malloy', 'Shreya Kannan', 'Sophia Qin', 'Srinath Goli', 'Vandana
Ganesh']
    time = 90
    def __init__(self, name, student_id):
        self.name = name
        self.sid = student_id
        self.score = {}

    def grade(self, points_correct, points_total):
        grade = points_correct/points_total
        self.score[self.sid] = grade

    def view_score(self):
        return self.score

class Question(Exam):
    extra_credit = False
    def grade(self, points_correct, points_total):
        if self.extra_credit == False:
            Exam.grade(self, points_correct, points_total)
        else:
            self.score[self.sid] += points_correct
```

Summoner's School for SQL

As a student of CS88, you've been noticed for your sharp SQL skills and magical powers in coding by the legends of Summoner's Rift. They offer you the title of "Master of Data," a title renowned by many, but only if you can complete the following SQL challenges. To aid you in your mission, you have given two tables: **champions** and **positions**.

```
CREATE TABLE champions AS
     SELECT "Aatrox" AS name, 580 AS health, 38 AS armor UNION
     SELECT "Ahri", 526, 20.88 UNION
     SELECT "Akali", 575, 23 UNION
     SELECT "Alistar", 575, 44 UNION
     SELECT "Amumu", 613.12, 33 UNION
     SELECT "Ashe", 539, 26 UNION
     SELECT "Jax", 592.8, 36 UNION
     SELECT "Lux", 490, 18.72 UNION
     SELECT "Malphite", 574.2, 37 UNION
     SELECT "Vayne", 515, 23 UNION
     SELECT "Yasuo", 523, 30;

CREATE TABLE positions AS
     SELECT "Aatrox" AS name, "Top" AS position UNION
     SELECT "Ahri", "Middle" UNION
     SELECT "Akali", "Top" UNION
     SELECT "Akali", "Middle" UNION
     SELECT "Alistar", "Support" UNION
     SELECT "Amumu", "Jungle" UNION
     SELECT "Ashe", "Bottom" UNION
     SELECT "Jax", "Top" UNION
     SELECT "Jax", "Jungle" UNION
     SELECT "Lux", "Middle" UNION
     SELECT "Lux", "Support" UNION
     SELECT "Malphite", "Top" UNION
     SELECT "Malphite", "Jungle" UNION
     SELECT "Malphite", "Support" UNION
     SELECT "Vayne", "Top" UNION
```

```
    SELECT "Vayne", "Bottom" UNION
    SELECT "Yasuo", "Top" UNION
    SELECT "Yasuo", "Mid";
```

1. Write a query that selects the name and health of the champion with the greatest health. You may assume health values are unique.

```
CREATE TABLE health AS
    SELECT name, health FROM champions
    ORDER BY health DESC LIMIT 1;

Output:
sqlite> SELECT * FROM health;
Amumu | 613.12
```

2. Write a query that selects the names of champions who are played in the jungle and have armor greater than 35

```
CREATE TABLE jungle AS
    SELECT champions.name FROM champions, positions
    WHERE champions.name = positions.name
    AND position = "Jungle" AND armor > 35;

Output:
sqlite> SELECT * FROM jungle;
Jax | 36
Malphite | 37
```

3. Write a query that selects the five positions and the number of champions that are played in each position sorted by the number of champions in descending order, meaning the position with the highest number of champions is at the top.

```
CREATE TABLE roles AS
    SELECT position, COUNT(*) as count FROM positions
    GROUP BY position ORDER BY count DESC;
```

```
Output:
sqlite> SELECT * FROM roles;
Top | 6
Jungle | 3
Middle | 3
Support | 3
Bottom | 2
Mid | 1
```

Iteration Generates Success

Given a list of single digits, create an Iterator that iterates through all the possible **two digit numbers** that can be created from digits in the list. The __init__ method and __iter__ method have been completed for you.

```
class BuildNum:
     """
     >>> for i in BuildNum([1, 2, 3]):
     ...     print(i)
     ...
     11
     12
     13
     21
     22
     23
     31
     32
     33
     """
    def __init__(self, lst):
        self.lst = lst
        self.i = 0
        self.j = 0

    def __next__(self):
        if self.i == len(self.lst):
            raise StopIteration
        result = self.lst[self.i] * 10 + self.lst[self.j]
        if self.j == len(self.lst) - 1:
            self.i += 1
            self.j = 0
        else:
            self.j += 1
        return result

    def __iter__(self):
        return self
```

Given a list of single digits, create a generator function that yields all the two digit numbers that can be created from the given list of digits and only have repeated digits, or a repdigit (ie. 11, 22, 33, … 99). You may use BuildNum and assume it has been implemented correctly for this part.

```
def rep_digit(lst):
     """
     >>> for i in rep_digit([1, 2, 3]):
     ...     print(i)
     ...
     11
     22
     33
     """
     for i in BuildNum(lst):
          if (i % 10 == i // 10 % 10):
               yield i
```

88’s Bizzare Social Network

You are making a social network where users can make profiles with their name and age. Users can keep track of their friends by adding them to their friends list. You decide that in order to make money you make users pay for premium accounts. Default profiles are limited to one friend, while premium accounts can have up to 1000. In addition, premium accounts have the ability to leave comments on other user’s profiles. Whenever a profile is created it is added to a list that contains all profiles ever created called all_profiles. Fill out the respective Profile and Premium classes to create your social network.

```
class Profile():
     “””
     >>>joseph = Profile("Joseph", 19)
     >>>jonathan = Profile("Jonathan", 20)
     >>>dio = Premium("Dio",21) #dio has a premium account
     >>>dio.name #profiles have an associated name
     “Dio”
     >>>dio.age #profiles have an associated age
     21
     >>>len(Profile.all_profiles) #all_profiles should contain
all three profiles: joseph, jonathan, and dio
     3

     >>>joseph.add_friend(jonathan)
     >>>len(joseph.friends)
     1
     >>>joseph.add_friend(dio) #default profiles can only have
one friend, so adding a second friend should print “Friends list
full!”
     Friends list full!

     >>>dio.add_friend(joseph)
     >>>dio.add_friend(jonathan)
     >>>len(dio.friends) #premium profiles can have up to 1000
friends
     2
```

```
     >>>dio.comment(jonathan, "Oh? You're Approaching Me?")
     >>>jonathan.comments #premium profiles can leave comments
on any profile.  Comments are recorded in a list.
     ["Oh? You're Approaching Me?"]
     >>>jonathan.comment(dio, "What?") #Default accounts cannot
comment and instead should print Need premium account to
comment"
     "Need premium account to comment"

     “””
    all_profiles = []
    limit = 1
    def __init__(self, name, age):
        self.friends = []
        self.comments = []
        self.name = name
        self.age = age
        Profile.all_profiles.append(self)

    def add_friend(self, friend):
        if(len(self.friends) < self.limit):
            self.friends.append(friend)
        else:
            print("Friends list full!")

    def comment(self, account, msg):
        print("Need premium account to comment.")

class Premium(Profile):
    def __init__(self, name, age):
        super().__init__(name, age)
        self.limit = 1000

    def comment(self, account, msg):
        account.comments.append(msg)
```

*Not sure how open ended to make the question so it could probably be adjusted by amount of given code

Perfect SymmeTREE

Q1: We will call a symmetric decreasing tree a tree where every node has 2 branches and the values in every node decrease by 1 every time the depth increases by 1. Write a function **symmetric_decreasing_tree** that given n, constructs a symmetric decreasing tree where the node at depth 0 has a value n, every node at depth 1 has a value n - 1, and so on, and all the leaves of the tree should have value 1. Assume that n >= 1.

```
def symmetric_decreasing_tree(n):
    """
    >>> symmetric_decreasing_tree(3)
    Tree(3, [Tree(2, [Tree(1), Tree(1)]), Tree(2, [Tree(1), Tree(1)])])
    >>> symmetric_decreasing_tree(1)
    Tree(1)
    """
    if n == 1:
        return Tree(n)
    else:
        branches = [symmetric_decreasing_tree(n-1) for i in range(2)]
        return Tree(n, branches)
```

Q2: A 'tree'cherous villain wants to ruin our symmetric decreasing tree by removing valuable information from it and polluting the tree with extra unlucky nodes. The villain has provided a linked list of numbers he wants to erase from the tree. Help the villain finish this task by completing the following methods.

a) Implement a helper function **contains_val** that returns whether or not a linked list contains a value.

```
def contains_val(lnk, val):
    """
    >>> contains_val(Link(4, Link(7, Link(1, Link(3)))), 7)
    True
    >>> contains_val(Link(5, Link(6, Link(8))), 3)
    False
    """
    if lnk is Link.empty:
        return False
    else:
```

```
        return lnk.first == val or contains_val(lnk.rest, val)
```

b) Implement the function **destroy_tree** by changing any value in the tree that appears in the supplied linked list to 0. You may use the method implemented in part (a).

```
def destroy_tree(t, lnk):
    """
    >>> t3 = symmetric_decreasing_tree(3)
    >>> bad_nodes = Link(3, Link(5, Link(1)))
    >>> destroy_tree(t3, bad_nodes)
    >>> t3
    Tree(0, [Tree(2, [Tree(0), Tree(0)]), Tree(2, [Tree(0), Tree(0)])])
    """
    if contains_val(lnk, t.entry):
        t.entry = 0

    for b in t.branches:
        destroy_tree(b, lnk)
```

c) Implement the function **add_unlucky_leaves** that, given a list of unlucky numbers, mutates the original tree by adding all of these numbers as branches to each of the original leaves in the tree. You can assume the list of unlucky numbers is **not** a nested list.

```
def add_unlucky_leaves(t, unlucky_nums):
    """
    >>> t = Tree(0, [Tree(2, [Tree(1), Tree(5)])])
    >>> add_unlucky_leaves(t, [13, 6, 17])
    >>> t
     Tree(0, [Tree(2, [Tree(1, [Tree(13), Tree(6), Tree(17)]), Tree(5,
[Tree(13), Tree(6), Tree(17)])])])
    """
    if t.is_leaf():
        t.branches = [Tree(val) for val in unlucky_nums]
    else:
        for b in t.branches:
            add_unlucky_leaves(b, unlucky_nums)
```