Problem 1: Getting Started with Loops

Repetition is what computers do best—and what people enjoy **not** having to do!

For this problem, you'll write a couple of short functions that use loops.

Make a copy of this trinket in order to start this problem of the homework. Then, to get into the design mindset of using *loops*, take a look at the factorial function in the trinket, which uses a `for` loop:

Looping function to write #1: `power(b,p)`

To start, read over and run the above starter code. You should see the tests pass with the expected results.

Then, either copy-and-alter that function—or create a new function based on that model named `power(b,p)`, which should:

- Take in any numeric value (base) `b`
- Take in any nonnegative integer (power) `p`
- Return the value of `b**p`
- Use `for` loop! (writing `return b**p` is not in the spirit of this problem)
- For this problem, `power(0,0)` should return `1.0` (some mathematicians might object)

Here are a few tests to paste and use:

```
print "power(2,5): should be 32 == ", power(2,5)
print "power(5,2): should be 25 == ", power(5,2)
print "power(42,0): should be 1 == ",
power(42,0)
print "power(0,42): should be 0 == ",
power(0,42)
print "power(0,0): should be 1 == ", power(0,0)
```

Hint: The value of `n` in factorial is analogous to the value of `p` (the power) in the `power` function. (The base is simply used as the multiplied value)

Looping function to write #2: `summedOdds(L)`

Here is a loop-based `summed` function we looked at in class—and try it out!

Then, either using this as a template or copying-and-pasting, create a loop-based function named `summedOdds(L)`, which should:

- Take in any list of integers `L`. You may assume the input list includes only integers.
- Return the sum of all of the **odd** elements in `L` using a loop!

Hint: Use an `if` statement inside the loop and only add to `result` under the correct conditions.

Here are a couple of tests to paste and try:

```
print "summedOdds( [4,5,6] ): should be 5 == ", summedOdds( [4,5,6] )
print "summedOdds( range(3,10) ): should be 24 == ", summedOdds( range(3,10)
)
```

Looping function to write #3: `untilARepeat( high )`

This function will let you explore what is sometimes called the *birthday paradox*: even in a very small group of people (23 or more), there is greater than a 50-50 chance that a pair of those people share a birthday! This is well-explained here, including a nice explanation of why it seems paradoxical.

First, paste and try out the number-guessing `while`-loop example from class:

Then, write a variant of this (using the above function as a guide or, simply, copy-and-alter) named `untilARepeat( high )`, which:

- Keeps a list `L` of all of the integers guessed. Start with `L = [ ]`!
- Keeps looping as long as all of the elements in `L` are unique (no repeats).
    - Use a `while` loop!
    - Use the `uniq(L)` function that's provided below—it returns a boolean.
- Within the `while` loop:
    - Make a guess in the `range(0,high)`
    - Count the number of guesses (add one each time to some kind of counting variable)
    - Add the guess on to the end of the list `L` (see below for a hint on this)
- When the `while` loop finishes, the function should return the number of guesses needed until it gets a repeat

Hints!

- Remember that you can add 1 to a variable with `count += 1`.
- Similarly, you can add a new element `42` to the end of a list `L` with the line `L = L + [42]`. The element needs to be in brackets, because lists can be added only to lists.
- **Hint**: Feel free to use this `uniq(L)` function provided here—you may have written this in the previous

```
def uniq( L ):
    """ returns whether all elements in L are unique
        input: L, a list of any elements
        output: True, if all elements in L are
unique,
             or False, if there is any repeated
element
    """
    if len(L) == 0:
        return True
    elif L[0] in L[1:]:
        return False
    else:
        return uniq( L[1:] ) # recursion is OK, too!
```

homework!

Once you've written your `untilARepeat` function, you should confirm that `untilARepeat(365)` produces surpringingly small numbers! For example,

```
>>> untilARepeat( 365
)
25
>>> untilARepeat( 365
)
8
>>> untilARepeat( 365
)
23
>>> untilARepeat( 365
)
33
```

Then try running 1,000 times with a list comprehension! You can try even larger numbers, like 10,000 or even 100,000 if you're working offline, but don't try values that high in trinket—it will freeze your browser.

```
L = [ untilARepeat( 365 ) for i in range(1000)
]
```

and find the average and max and min of `L`:

```
>>> sum(L)/1000.0
>>> max(L)
>>> min(L)
>>> 42 in L
```

See if these values match your intuition!

Submit Homework 8, Problem 1

25.0/25.0 points (graded)

To submit your Homework 8, Problem 1 code, you'll need to copy it from your trinket or file and paste it into the box

below. After you've pasted your code below, click the "Check" button.

IMPORTANT: Make sure that there aren't spaces at the beginning of your code, and that you copied all of the characters. If there are extra spaces or you are missing spaces, our server won't be able to run your code and we won't be able to give you any of the points you deserve for your hard work.

```
1


2


3


4


5


6


7


8


9
import random
10


11
```

```
12
def power(b,p):
13
    """ loop-based power function
14
        input: b:any numeric value (base), p:any nonnegative integer
(power)
15
        output: the value of
b**p
16
"""
17
    result = 1
18
    for x in range(p):
19
        result = result * b
20
    return result
```

21

22

23

24

25

Press ESC then TAB or click outside of the code editor to exit
correct

correct

Test results
CORRECT See full outputSee full output

You have used 1 of 3 attempts Some problems have options such as save, reset, hints, or show answer. These options follow the Submit button.