CS116 TUTORIAL

ITERATION
REMINDERS

• Assignment 6 is due at 10 AM on Wednesday, July 3rd
  – Hint: Best time to come seek help is any day that’s not the day before the due time. (Less competition!)
TODAY → LOOPS!!

• 2 Types of loops
  – while
  – for
• Nested Loops
REVIEW – WHILE LOOPS

***initialize variables***

```python
while condition:
    ***body of while, including***
    ***update of variables***
```

- The body of the while loop will execute until `condition == False`
- The `condition` is only checked before each execution of the loop body.
- Variables **MUST** be updated, otherwise there might be an infinite loop!

(Sort of like maximum recursion depth)
**REVIEW – FOR LOOPS**

```python
for item in collection:
    *** body of loop ***
```

- The body of the for loop will execute `len(collection)` times, once for every element in `collection`
- Similar to `map`; goes through every element in the `collection`

A collection is something like a list, a string, etc.
WHILE LOOP VERSION OF A FOR LOOP

```python
for item in collection:
    ***body of loop***
```

```python
i = 0
while i < len(collection):
    item = collection[i]
    ***body of loop*** (same as above)
    i = i + 1
```
**REVIEW – NESTED LOOPS**

```python
for i in collection1:
    *** body of outer for ***
    for j in collection2:
        *** body of inner for ***
```

- For each `i` in `collection1`, the inner for loop will be executed
- Examples of possible `collection1`:
  - list of nested lists
  - lists of strings

The inner for loop will be executed `len(collection1)` times.

The body of the inner for will execute `len(collection2)` times for each value of `i`. 
WHAT SHOULD MY LOOP COUNTER BE?

Examples for some meaningful counter names:

- `i to n` => integer
- `l` => List
- `s` => string
- `c` => characters (strings of length 1)

- You are always allowed to use other meaningful names

- `i, j, k` convention for integer counters are in fact inherited from Fortran. In Fortran, integer variables had to start with the letters `i` through `n`.

- This is only for interest, materials on this will not be tested on exam.
QUESTION 1 - ALLSAME_TYPE

Write a function all_same_type that consumes a list, called lst, and returns True if all members of that list are of the same type, else False.

For example:

    all_same_type([2, 5, 3]) => True
    all_same_type([2, 'R', 4.56]) => False

Note that Python's built-in type function does not distinguish between types of lists:

    i.e. type([1,2]) == type(['a', 'b'])
Write a Python function `max_even_sum` that consumes a nonempty list, `lst`. Each value in `lst` is a list of positive integers. It computes the sum of the even integers in each of the element lists in `lst`, and returns the largest out of these sums.

If an element list contains no even integers, its sum is zero.

**For example:**

```python
max_even_sum([[], [3], [2, 4, 6]]) => 12
```
Write a Python function `divisible_by_3` that consumes a Nat (called n), and returns True if n is divisible by 3, and False otherwise. You must use the following algorithm:

- The only numbers less than 10 that are divisible by 3 are: 0, 3, 6, 9
- A number is divisible by 3 if and only if the sum of its digits is also divisible by 3. If the sum of the digits of the number is greater than 10, calculate the sum of the digits of the sum, and repeat until you get a number less than 10.
Write a Python function `make_list` that consumes a natural number `n` and returns a list of strings. The produced list will look like

`["", "1", "22", "333", "4444", "55555", ... , "nnnnn...nnnn"]`

where the last element is the number `n` repeated `n` times.

For example:

```python
make_list(0) => [""]
make_list(3) => ["", "1", "22", "333"]
```