REMINDERS

• A06 is due on 10 AM Wednesday, June 27th
  – 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
  – 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)*
for item in collection:
    *** body of loop ***

• The body of the for loop will execute
  \text{len}(\text{collection}) \times \text{times}, \text{once for every element in}\n  \text{collection}

• Similar to \text{map}; \text{goes through every element in the}\n  \text{collection}

• Be careful of mutating your \text{collection} inside the loop!
  
  – \text{Never mutate the same collection that you are iterating over.}
WHILE LOOP VERSION OF A FOR LOOP

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

```
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 `len(collection1)` times.

- 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:

- \texttt{i to n} $\Rightarrow$ integer
- \texttt{l} $\Rightarrow$ list
- \texttt{s} $\Rightarrow$ string
- \texttt{c} $\Rightarrow$ characters (strings of length 1)
  - \textit{You are always allowed to use other meaningful names}

\texttt{i, j, k} convention for integer counters are in fact inherited from \textit{Fortran}. In Fortran, integer variables had to start with the letters \texttt{i} through \texttt{n}.
  - \textit{This is only for interest, materials on this will not be tested on exam.}
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']) => True
```
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:**

```
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 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:

- `make_list(0) => ["" ]`
- `make_list(3) => ["", "1", "22", "333"]`
Write a function called `valid_input` that consumes a string to be used as the prompt, `prompt`, a list of strings of valid inputs, `valid`, and a positive integer `max_guess`.

The function should continuously prompt the user for input until the user enters a value in the list `valid`, and then return that value, or print a message when maximum number of guess is reached. If the user enters an invalid value, the function will let them know by printing: "Invalid input. Try again." to the screen. If maximum number of guess is reached, the function will print "Maximum number of guess is reached" and return `None` in this case.
For example:

If the user enters "6", "5", and "3",
valid_input("Enter a digit < 5: ",
        ["0", "1", "2", "3", "4"], 5) => "3"

and the following is printed:

Enter a digit < 5: 5
Invalid input. Try again.
Enter a digit < 5: 3

Note: You may assume that the user enters input that is the correct type.