Due: Wednesday Nov. 1st, 2017 at 10 am

Assignment Guidelines

- This assignment covers material in Module 06.
- **NO helper functions are allowed for this assignment**
- **Do NOT use recursion or abstract list functions (map and filter). All repetition must be performed using iteration (while and for loops only). Solutions using recursion will receive a grade of 0.**
- You don't need to submit any design recipe for Q1 and Q2. However, you are encouraged to test your solutions.
- Submission details:
  - Solutions to these questions must be placed in files `a06q1.py`, `a06q2.py`, `a06q3.py`, and `a06q4.py`.
  - You must be using Python 3 or higher.
  - Download the interface file from the course Web page to ensure that all function names are spelled correctly and each function has the correct number and order of parameters.
  - All solutions must be submitted to MarkUs. No solutions will be accepted through email, even if you are having issues with MarkUs.
  - Verify using MarkUs and your basic test results that your files were properly submitted and are readable on MarkUs.
  - For full style marks, your program must follow the Python section of the CS116 Style Guide.
  - Be sure to review the Academic Integrity policy on the Assignments page
- Download the testing module from the course web page. Include `import check` in each solution file.
- Restrictions:
  - Do not import any modules other than `math` and `check`.
  - Do not use any other Python functions not discussed in class or explicitly allowed elsewhere. See the allowable functions post on Piazza.
  - While you may use global *constants* in your solutions, do **not** use global *variables* for anything other than testing.
  - Read each question carefully for additional restrictions or tips.

The solutions you submit must be entirely your own work. Do not look up either full or partial solutions on the Internet or in printed sources.

1.

Both parts should be submitted in one file

Part a)

Write a Python function `calc_exp` that consumes three positive natural numbers `a`, `b` and `n`, and returns the result of $(b+b^2+b^3+\ldots.+b^n) + (a+1)(a+2)\ldots..(a+n)$

For example,

```
calc_exp(10, 10,1) => 21
```

`calc_exp(1,3,4) => 240 ,` because $3+3^2+3^3+3^4+(1+1)(1+2)(1+3)(1+4)=240$

Part b)

Write a Python function `list_stat` that consumes a list named `lst` and returns a list containing exactly 5 natural numbers in the order that follows:

- The number of integers in `lst`
- The number of floats in `lst`
- The number of booleans in `lst`
- The number of strings in `lst`
- The number of all other types in `lst`

For example,

```
list_stat([3, "wow", -3.967, True, True, False, "nice"]) =>
[1, 1, 3, 2, 0]
list_stat(["good", [3,4], [10]]) => [0, 0, 0, 1, 2]
```

2.

All four parts should be submitted in one file

Part a) Write a Python function `create_odds` that consumes a natural number (`target`) and returns a list of all positive odd integers that are <= `target` in ascending order.

For example:

```
create_odds(8) => [1, 3, 5, 7]
create_odds(0) => [ ]
```

Part b) Write a Python function `build_special_list` that consumes a natural number (`n`) and returns a list following the pattern `[[1], [1,2], … , [1,2,3,…,n]]`.

For example:

```
build_special_list(6) => [[1], [1, 2], [1, 2, 3], [1, 2, 3,
4], [1, 2, 3, 4, 5], [1, 2, 3, 4, 5, 6]]
build_special_list(0) => []
build_special_list(1) => [[1]]
```

Part c) Write a Python function `divisibles` that consumes a natural number `n` and returns a list of all divisibles of `n` (that are less than `n`) in ascending order.
For example:

```
divisibles(16) => [1, 2, 4, 8]
divisibles(0) => []
divisibles(1) => []
divisibles(19) => [1]
```

Part d) Write a Python function `update_list` that consumes a list of integers `nlst`, and two integers `val` and `newval`, and mutates `nlst` by changing all the occurrences of `val` in `nlst` to `newval`. You may assume that `val != newval`. The function also returns how many times the change occurred.
For example:
if `nl` is `[]` then `update_list(nl, 5, 10) => 0` and `nl` is unchanged
if `nl` is `[3, 10, 5, 10, -4]` then `update_list(nl, 10, 7) => 2`
and `nl` is mutated to `[3,7, 5, 7, -4]`

3. Note: You are not allowed to use sort method or define your own sort function for this question.

Write a Python function `most_frequent` that consumes a non-empty list of integers, `nlst` (which may contain duplicate values) and returns a list of the most frequently occurring integer(s).
Note that more than one integer may appear the same number of times. The first occurrence of the frequently occurring integer in `nlst` determines the order in the list that is returned.
For example,

```
most_frequent([16, 0, 15, 16, 15, -10, 7])  => [16, 15]
most_frequent([16, 0, 15, 16, 15, -10, 7, 16])  => [16].
```

4. Note: You are not allowed to use *for* loop for this question, you must use *while*.

The digit sum of a number is found by, first, summing its digits. If the sum is greater than 9, then the digits of the sum are added. This process is repeated until a single digit number is obtained. The digit sum of 602 is 8 since 6 + 0 + 2 = 8, and 8 is a single digit number. The digit sum of 897 is 6 since 8 + 9 + 7 = 24, and 2 + 4 = 6 .

Write a Python function `digit_sum` that consumes a natural number (`num`) and returns its digit sum, while printing the process of calculation then the final answer as follows:
For example,

`digit_sum(8) => 8`, and prints:

```
8
```

`digit_sum(897) => 6`, and prints:

```
8+9+7= 24
2+4= 6
6
```