Due: Wednesday Sep. 20th, 2017 at 10 am

Assignment Guidelines

- This assignment covers material in Module 01.
- Submission details:
  - Solutions to these questions must be placed in files `a01q1.py`, `a01q2.py`, `a01q3.py`, and `a01q4.py`.
  - You must be using Python 3 or higher. Do NOT use Python 2.
  - 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.
  - When a function produces a floating point value, you must use `check.within` for your testing. Unless told otherwise, you may use a tolerance of 0.001 in your tests.
- Restrictions:
  - Do not import any modules other than `math` and `check`.
  - Do not use Python constructs from later modules (e.g. loops and lists). Do not use any other Python functions not discussed in class or explicitly allowed elsewhere. See the allowable functions post on Piazza. You are always allowed to define your own helper functions, as long as they meet the assignment restrictions.
  - 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.

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.

From Racket to Python …

1. Consider the following three functions which have been defined in Racket. This question deals with converting Racket code into its Python counterpart. For this problem, convert the given Racket code into Python within your a01q1.py file. Do NOT submit any *Racket* code for this problem. You are required to define all 3 functions within the same file.
   Note: You are NOT required to do any design recipe for this question!

```
;; f1: Nat Nat -> Nat
(define (f1 x y)
    (quotient (sqr (* y 5)) x))

;; f2: Nat Nat -> Int
(define (f2 a b)
    (expt (+ a b) (remainder a 10)))

;; f3: Nat -> Num
(define (f3 n)
   (* (sqrt (* 2 pi n)) (expt (/ n e) (- n 1))))
```

From Calculation to Python …

2. In probability theory, the **normal distribution** is an incredibly important function, which is commonly referred to as the bell curve. The equation of this function is given as:

$$f(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} * \frac{1}{e^{\frac{(x-\mu)^2}{2\sigma^2}}}$$

   Where σ is the standard deviation, μ is the mean, and π and *e* are the well-known constants. Write a Python function called `normal_distribution` which consumes 3 positive floating point numbers, namely `x`, `mean` and `std_dev`, and returns the corresponding value associated with the normal distribution. Do NOT round your answer! Do a direct translation. You must use `check.within` with a tolerance value of `0.001`. You may find the math module very helpful for this question. For example:

```
normal_distribution(3.0, 5.0, 2.0) => 0.12098536225957168
```

   Note: You do not have to get `0.12098536225957168` exactly as in the example due to different machine precisions.

From Mathematical Tricks to Python …

3. Write a Python function `forever_15` that consumes n, a natural number and performs the following calculations, and then returns the resulting value. In order to receive any correctness marks, your function must perform the given operations (i.e. you cannot simply return `15`). Note: The function will return an integer, not a floating point number!

For this mathematical trick, you must follow (and implement) these steps:

- Think of a positive integer *(this will be* n*)*.
- Multiply it with 3.
- Add 45 to this result
- Then multiply by 2.
- Divide this result by 6.
- Subtract the original number from it.

The answer will always be 15.

For example:

```
forever_15(20) => 15
forever_15(150) => 15
```

4. Write a Python function `min3` that consumes three integers (`a, b,` and `c`) and returns the minimal value among the three consumed ones without using `max`, `min`, or conditions in Python. You are not allowed to import math module for this question. Only simple mathematical operations such as `* - + / % // abs` can be used.
   For example:
   ```
   min3(1, 1, 1) => 1
   min3(4, 14, -3) => -3
   ```
   Note: Maybe you want to review/use your mathematical knowledge regarding how to find minimum between two integers (following the mentioned restrictions above) and then apply the method on three integers.