ME 701 – Development of Computer Applications In Mechanical Engineering
Homework 3 – Due 9/22/2017

Instructions: Your solutions to the following should be contained in one file named `lastname_firstname_hw3.py` and uploaded to Canvas.

Problem 1 – Some Python `str` Basics

1. Let `a = "hello"` and `b = "world"`. Use `a` and `b` to produce `c = "hello world"`.
2. Figure out how to define `d = "Hello World"` starting with `c`.
3. From `d`, define `e = "Hello"` and `f = "World"` in a single line of code.
4. Let `g = 123; h = 3.141592653589793; i = 6.022e23`. Using those values, produce `j = '123|3.1416| 6.02e+23'` in just one line.
5. Let `j = 5` (or some other integer). Use that to produce `k = '0..1..2..3..4'` in just one line.

Problem 2 – A Python Oddity

Consider the following code and output

```
>>> a = [1, 2, 3]
>>> b = a
>>> a[0] = 99
>>> a
[99, 2, 3]
>>> b
[99, 2, 3]
```

In other words, a change in `a` leads to a change in `b`. This can confuse Python newbies.

1. Explain, in your own words, why this happens.
2. Offer **two** ways by which one could produce a second list `b` with the same values as `a`. At least **one** of these should require no more than one expression. For reference, an expression is just a composition of arithmetic or other operations, e.g., `sin(a**2)/4.0+1`.

Problem 3 – List Comprehension

List comprehension is a uniquely Pythonic way to construct lists without using explicit loops. Rewrite the following using list comprehension:

1.
```
# compute first 20 powers of 2
i = 0
powers = []
while i < 20:
```

```
        p = 2 ** i
        powers.append(p)
        i = i + 1
```

2.
```
    # Generate all (x, y, z) coordinates from three lists
    xpoints = [1, 2, -1]
    ypoints = [8, 4, 3, 0]
    zpoints = [0, -1]
    points = []
    for x in xpoints:
        for y in ypoints:
    for z in zpoints:
        points.append((x, y, z))
```

Problem 4 – Binary Fun

In class, we covered some basics of floating-point numbers, showing, for example, that 0.1 (in base-10) can only be represented in the base-2 (binary) system using an infinite number of bits. In other words, 0.1 cannot be represented on our computers. Your job is to write a function that computes the closest binary representation of a given base-10 number using a fixed number of bits for the fractional part (i.e., the stuff to the right of the floating point). In essence, you are extending the built-in `bin` function.

Deliverables:

1. Implement a function named `decimal_to_binary(x, n)` that accepts a floating-point number n and an integer n, and returns the binary representation of that number using at most n bits to the right of the floating point. For simplicity, return the result as a string. Example: for $x = 123.625_{10}$ and $n = 4$, you should return `'1111011.1010'`.

2. Implement a second function `binary_to_decimal(i, f)` that takes a binary number of the form `'1111011.1010'` and returns it in base-10 as a standard float.

Problem 5 – Conservation of Numbers?

Summing up the elements of an array is easy:

```
# option 1
s = 0
for i in len(a):
   s += a[i]
```

However, one could also do this:

```
# option 2
s = 0
```

```
a = sorted(a)
for i in len(a):
   s += a[i]
```

Or even this:

```
# option 3
def sumr(a):
    if len(a) <= 2:
        return sum(a)
    else:
        return sumr(a[:len(a)//2]) + sumr(a[len(a)//2:])
s = sumr(a)
```

By using `a = np.random.rand(n)` and the `Decimal` module, perform a numerical experiment that shows (1) which of these approaches is most accurate, (2) how these compare to the built-in `sum` and `np.sum` functions, and (3) how the error in the sum varies with the number of elements `n` of the array `a` being summed.

I hope that this problem highlights a basic fact: even the simplest of numerical computing tasks results in observable error!