Python LAB 3, **TMMK16**
Maths for Software Engineers
Due: **2018-02-19** | Pass at: $\geq$ **2** points

Name:	Khan Muhammad Uzair
JU-sign:	khmu1784

- Labs are personal and are submitted individually to Ping Pong before midnight on due date.
- Fill code at the `YOUR_CODE_HERE`–marks in `Lab3.py` and `mse.py` to solve the problems. `Lab3.py` does `import mse.py` so it must reside in the same dir.
- Submit this assignment in pdf, your copies of `Lab3.py` and `mse.py` along with the results you obtained. These should be reproducible by running python3 on `Lab3.py` in a sandbox dir with your copy of `mse.py`.

`PLAY-part:` (not to be included in the report) Play a while as hinted in the Play-block of `Lab3.py` to get familiar with the functions from `mse.py`'s section `Sec: Binomial Coefficients and Convolutions`. Consider in particular `ConvMin(a,b)`, computing the convolution of the lists `a` and `b` truncated to `min(len(a),len(b))`, as well as `ConvPowMin_slow(a,n)` which computes the n-fold convolution of a list `a` truncated to `len(a)` elements, so that, if

```
a=[1]*4                 # a      = [1,1,1,1] <-1:1-> coeff(1+t+t^2+t^3)
ConvPowMin_slow(a,2)    # a*a    = [1,2,3,4]  --> (1+t+t^2+t^3)^2 truncated to 4 elem/deg 3
ConvPowMin_slow(a,3)    # a*a*a  = [1,3,6,10] --> (1+t+t^2+t^3)^2 truncated to 4 elem/deg 3
ConvPowMin_slow(a,1234) # a^1234 = [1, 1234, 761995, 313941940]
```

1. In `Lab3.py`: Use the `mse.py`-routines `nCr`, `Conv` or `ConvMin`, `ConvPow` or `ConvPowMin_slow` to: (1 p)

   (a) Find the No of ways to split 12 identical objects in 11 bins, or, equivalently, the number of solutions

   $$(\#)\{x_1 + x_2 + \cdots + x_{11},\ x_i \geq 0\} = \binom{?}{?},$$

   through the right binomial coefficient `nCr(?,?)`. Obtain same result by associating each container x_k with a polynomial $p(t) = 1 + t + t^2 \cdots + t^r$ (list `[1]*(r+1)`) and reading off the proper coefficient:

   $$\texttt{g=ConvPowMin(p,n); g[?] = } (c_?)(1 + t + t^2 \cdots + t^r)^m = (\#)\{x_1 + x_2 + \cdots + x_{11},\ x_i \geq 0\}.$$

   (b) Compute the coefficients of the generating function

   $$(1 + t + \cdots + t^{515})(1 + t^5 + \cdots + (t^5)^{103})(1 + t^{10} + \cdots + (t^{10})^{51})(1 + t^{20} + \cdots + (t^{20})^{25})$$

   to find in how many ways can one split \$515 into \$1, \$5, \$10 and \$20 bills?
   Modify your input to answer in how many ways can one split the \$515 if at least half of the change should be in \$20 bills and no more than \$20 of the change are \$1 bills.

   (c) In `Lab3.py`: Run the five inbedded loops: $s = 0$; `for` $0 \leq i_1 \leq i_2 \leq i_3 \leq i_4 \leq i_5 \leq 30 : \{$`s++`$\}$. Add a line obtaining the same value as `s` by computing a properly chosen binomial coefficient.

2. In `mse.py`: Devise `prMsgNo(Str)` which prints the number of all different messages contained in the string `Str` by ripping the corresponding coefficients off the *exponential* generating function. Run it on `"ZAMBEZEE", "TALLAHASEE", "MISSISSIPI"` in `Lab3.py`. E.g., `prMsgNo("ORINOCO")` should give: (1 p)

```
ORINOCO: 7 symb, (RCNI)x1 (O)x3
g_7(z) = (1+z)^4 (1+z+z^2/2!+z^3/3!)
#(Msg of len 1) = c_1(g)*1! = 5     #(Msg of len 5) = c_5(g)*5! = 480
#(Msg of len 2) = c_2(g)*2! = 21    #(Msg of len 6) = c_6(g)*6! = 840
#(Msg of len 3) = c_3(g)*3! = 73    #(Msg of len 7) = c_7(g)*7! = 840
#(Msg of len 4) = c_4(g)*4! = 208
```

3. In `mse.py`: Write a function `ConvPowMin_fast(a,n)` which computes the n-th convolution power of `a` truncated to `len(a)` by converting the exponent n into binary and *by repeated squaring*. E.g., (1 p)

   $$a^{23} = \{23 = 10111_2\} = a^{1\cdot 2^4 + 0\cdot 2^3 + 1\cdot 2^2 + 1\cdot 2^1 + 1\cdot 2^0} = (a^{16})^1 * (a^8)^0 * (a^4)^1 * (a^2)^0 * (a^1)^1,$$

   so, evaluating from right to left, the exponentiation is achieved in less than $2\log_2(n)$ convolutions.

(a) Complete checks in `Lab3.py` comparing the results of `ConvPowMin_slow` and `ConvPowMin_fast` on random data. Once passed, place in `mse.py` an alias pointer `ConvPowMin = ConvPowMin_fast`.

(b) In `Lab3.py`: Use the skeleton loop to compare the right element `ConvPowMin(p,n)[?]` in the convolution power p^n and the binomial coefficient `nCr(?,?)` for a proper chosen list p using that

$$(c_r)(1+t+\cdots+t^r)^n = \binom{n-1+r}{r}.$$

(Play) In `Lab3.py`: Use the wrapper function `prClock_F1_vs_F2` provided in `mse.py` to compare the timings of a^n computed by `ConvPowMin_slow(a,n)` and `ConvPowMin_fast(a,n)` as n increases.

If an algorithm's time complexity is of order $T(n) = \mathcal{O}(n^k)$ as a function of some typical size n, then

$$T(n) \sim Cn^k$$

for some positive constants C and k. Taking logarithm on both sides gives

$$\log(T) \sim \log(C) + k\log(n),$$

and hence a plot of $\log(n)$ vs $\log(T)$ plot of clocked exec times should loosely follow a straight line of gradient k. This constant can be read off by least squares linear fit using `numpy.polyfit(x,y,deg)`.

4. In `Lab3.py`: Use the `mse.py` (Sec: Clock) wrappers `prClock_F_n` and `prLogRegrPlot` to clock, plot $\log(n)$ vs $\log(T)$ and compute the exponent k (the gradient of the linear fit) as shown in `Lab3.py` for the following functions: (1 p)

   (a) `np.linalg.inv()` by inverting random $n \times n$-matrices, a typical $\mathcal{O}(n^3)$ task;
   (b) `mse.py`'s `Conv(p_n,q_n)`, the convolution of two lists of length n, a typical $\mathcal{O}(n^2)$ task
   (c) `ConvPow(p,n)`, the n-th convolution power of a fixed-length list p, an $\sim \mathcal{O}(n^2)$ task;
   (d) `ConvPowMin_slow(p,n)`, the `len(p)`–truncated n-th power of a fixed-length list, an $\mathcal{O}(n^1)$ task;
   (e) `ConvPowMin_fast(p,n)`, the `len(p)`–truncated fast n-th power, an $\mathcal{O}(\log(n))$ task;

   Include the plots (as .pdf or .png) in your report.

(Play) With some of the `mse.py` (in Sec: Base integer) functions `lsPrimes_lt(n)`, to list the primes less than n (Erastostenes sieve, $\sim$ 240 BC), the pair of functions `lsPfactors(n)` and `lsPfactors_once(n)`, which list the prime factors of n, with and without multiplicities:

```
lsPrimes_lt(17)     # --> [2,3,5,7,11,13];
lsPfactors(36)      # --> [2, 2, 3, 3];
lsPfactors_once(36) # --> [2, 3].
```

The function `gcd,x,y = EuklidExt(a,b)` realizes the extending Euklides algorithm to find the $\gcd(a,b)$, as well as $(x,y) \in \mathbb{Z}$, solving the linear congruence (Bézout identity):

$$ax + by = \gcd(a,b).$$

Sample output (as $17 \perp 25 \;\Rightarrow\; \gcd(17,25) = 1 = 3\cdot 17 - 2\cdot 25$):

```
x,y,gcd = EuklidExt(17,25)  # --> gcd = 1, x = 3, y = -2
```

thus giving the pairs of mutual inverses $[3]\cdot[17] = [1]$ in U_{25} and $[-2][25] = [15][8] = [1]$ in U_{17}.

5. In `mse.py`: Write the functions `lsUn(n)`, listing $U_n = \{0 < k < n : k \perp n\}$, all co-primes to n, e.g., (1 p)

```
lsUn(36)   --> [1, 5, 7, 11, 13, 17, 19, 23, 25, 29, 31, 35],
```

   and `Euler_phi(n)`, returning their number, $\phi(n) = |U_n|$. Set $\phi(0) = 0$ and $\phi(1) = 1$. For $n > 1$, use that if n is factored in prime factors (use `lsPfactors(n)`), then

$$\phi(n) = \phi(p_1^{e_1}\cdots p_k^{e_k}) = \prod_1^k (p_j - 1)p_j^{e_j-1},\ n > 1,\ \text{e.g., } \phi(36) = \phi(2^2\cdot 3^2) = 1\cdot 2^1\cdot 2\cdot 3^1 = 12.$$

   Run the tests in `Lab3.py` checking whether `len(lsUn(n))==Euler_phi(n)` on random data.

6. Devise a function `x=Zn_Inv(a,n)` returning the (least positive, $1 < x < n$) inverse $[x]_n = [a^{-1}]_n$ in U_n if $a \in U_n$ and `None` otherwise. Basically, a wrapper around `x,y,gcd=EuklExt(a,n)`, as (1 p)

$$ax + ny = 1 \Leftrightarrow [ax]_n = [1]_n \Leftrightarrow [x]_n = [a^{-1}]_n$$

Run the tests in `Lab3.py` to check that $[ax]_n = [1]_n$ on random data.