```python
# ... to the only wise God

# This is the home of functions that implements simple petroleum engineering computations.

######## A function to compute real gas density ########
# Note: pressure must be in psia and temperature in degree Rankine

def gas_density(gravity, pressure = 14.7, temperature = 520, z = 1):
    density = (2.70*pressure*gravity)/(z*temperature)
    return round(density, 4)

######## A function to estimate bubble point pressure, pb ########
# Note: this function only works if solution gas-oil ratio at a pressure above bubble point (i.e. Rsi (=Rsb)) is known
# Note that temperature is in degree Fahreiheit

def bubble_pressure(temperature, pressure, gas_gravity, oil_gravity, rsb):
    api = (141.5/oil_gravity)-131.5
    y = (0.00091*temperature)-(0.0125*api)
    pb = (18*(10**y))*((rsb/gas_gravity)**0.83)
    return round(pb,2)



######## A function to compute solution gas-oil ratio, Rs ########
# Note: temperature must be in degree Fahreiheit

def sol_gor(temperature, pressure, gas_gravity, oil_gravity, pb): # where pb is bubble point pressure.
    api = (141.5/oil_gravity)-131.5
    y = (0.00091*temperature)-(0.0125*api)
    if pressure<pb:
        rs = gas_gravity*(((pressure)/(18*(10**y)))**1.205)
        return round(rs,2)
    else:
        rsb = gas_gravity*(((pb)/(18*(10**y)))**1.205)
        return round(rsb,2)


########  A function to compute oil formation volume factor, Bo ########
# Note: temperature must be in degree Fahreiheit

def fvf(pressure, temperature, gas_gravity, oil_gravity, pb = None, rs = None, co = None):

    # calling function bubble_pressure if neccessary (i.e. if pb is not specified)
    if pb is None:
        pb = bubble_pressure(temperature, pressure, gas_gravity, oil_gravity, rs)

    # calling function sol_gor if neccessary (i.e. if rs is not specified)
    if rs is None:
        rs = sol_gor(temperature, pressure, gas_gravity, oil_gravity, pb)

    # calculating F parameter
    F = (rs*((gas_gravity/oil_gravity)**0.5))+(1.25*temperature)

    if pressure > pb:
        bob = 0.9759+(0.00012*(F**1.2)) # assuming gas_gravity and oil_gravity are constant for all pressures above pb
```

```
        # importing needed library
        import math
        bo = bob*(math.exp(co*(pb-pressure)))
    else:
        bo = 0.9759+(0.00012*(F**1.2))

    return round(bo, 4)

########  A function to compute Stock Tank Oil Initially In-Place (STOIIP), N ########
def stoiip(area, thickness, poro, sw, boi):
    N = (7758*area*thickness*poro*(1-sw))/boi
    return round(N, 2)

########  A function to compute Stock Tank Oil Initially In-Place (STOIIP), N ########
# This function accepts a single argument
def stoiip_2(data):
    N = (7758*data['area']*data['thickness']*data['poro']*(1-data['swi']))/data['boi']
    return round(N, 2)


########  A function to compute STOIIP for all blocks in a discretized reservoir, and returns the value total STOIIP and a list of block STOIIP ########
def stoiip_discretized(Lx, Ly, h, nx, ny, boi, poro_list, swi_list):

    # discretizing the reservoir
    delta_x = Lx/nx
    delta_y = Ly/ny

    # calculating the area per block
    area = delta_x*delta_y

    total_stoiip = 0
    stoiip_list =[]

    # the 'for' loop
    for j in range(1,ny+1):
        for i in range(1,nx+1):
            block_n_order = (nx*(j-1))+i
            poro = poro_list[(block_n_order - 1)]
            sw = swi_list[(block_n_order - 1)]
            block_stoiip = (7758*area*h*poro*(1-sw))/boi
            stoiip_list.append(block_stoiip)
            total_stoiip = total_stoiip + block_stoiip
    return total_stoiip, stoiip_list


########  A function to compute STOIIP for all blocks in a discretized reservoir, and returns the value total STOIIP and a dictionary of block STOIIP ########
def stoiip_discretized_2(Lx, Ly, h, nx, ny, boi, poro_list, swi_list):

    # discretizing the reservoir
    delta_x = Lx/nx
    delta_y = Ly/ny

    # calculating the area per block
    area = delta_x*delta_y

```

```
    total_stoiip = 0
    stoiip_dict ={}

    # the 'for' loop
    for j in range(1,ny+1):
        for i in range(1,nx+1):
            block_n_order = (nx*(j-1))+i
            block_label = 'Block'+str(block_n_order) # to be used as key in stoiip_dict
            poro = poro_list[(block_n_order - 1)]
            sw = swi_list[(block_n_order - 1)]
            block_stoiip = (7758*area*h*poro*(1-sw))/boi
            stoiip_dict[block_label] = block_stoiip
            total_stoiip = total_stoiip + block_stoiip
    return (total_stoiip, stoiip_dict)



########  A function to implement the oil material balance equation (MBE) and
# compute oil produced (np) for all blocks in a discretized reservoir,
# and returns the value total oil produced and a list of block np ########
def np(nx, ny, nz, N, Pb, bob, co, ce, boi, pi_list, pnow_list):

    total_np = 0
    np_list = []

    # the 'for' loops
    for k in range(1,nz+1):
        # fetching pi for the layer from the pi_list
        pi = pi_list[k-1]
        for j in range(1,ny+1):
            for i in range(1,nx+1):
                block_n_order = (nx*ny*(k-1)) + (nx*(j-1)) + i
                # fething pnow for the block from the pnow_list
                Pnow = pnow_list[block_n_order-1]
                bo = bob*(1 - (co*(Pnow - Pb)))
                block_np = (N*boi*ce*(Pi - Pnow))/bo
                total_np = total_np + block_np
                np_list.append(block_np)
    return total_np, np_list

```