```
function [V0, U_Cube, Policy] = SolveConsumerProblem(Asset, Income, Econom_param)

U_Cube = UtilityCube(Asset.Values, Income.Values, Econom_param);

% define vetor inicial de chutes (lin:asset col:income)
V0 = zeros(Asset.Grid.N,2);

% define a variavel de cubo do V0 que sera utilizado na interacao
V0_cube = nan(Asset.Grid.N, Asset.Grid.N, 2);

% declara o vetor de politica
pol_a_idx = nan(Asset.Grid.N, 2);

Test_param.tolerance = 0.0001;
Test_param.error = 2;
nIter = 0;
while Test_param.error > Test_param.tolerance

    % Calcula o valor experado da da funcao valor
    ETV = V0 * Income.PI';

    % Transforma o valor esperado em um Cubo para calculo
    V0_cube = repmat(reshape(ETV,[Asset.Grid.N, 1, 2]), 1, Asset.Grid.N, 1);

    % Calcula o maximo pelo operador TV
    [V1, pol_a_idx] = TV_op(U_Cube.Values, Econom_param, V0_cube);

   % Incrementa o contador de interacoes
    nIter = nIter + 1;

    % Atualiza o erro de interacao
    Test_param.error = norm(V1(:) - V0(:));

    % Atualiza o valor de V0
    V0 = V1;
end

% DEFINICAO DAS POLITICAS
Policy.AssetDomain = U_Cube.a_domain;
Policy.AssetPrime.Values = U_Cube.a_domain(pol_a_idx);
Policy.AssetPrime.Index = pol_a_idx;
Policy.Consumption.Values = ones(1,Asset.Grid.N)'*Income.Values + Asset.Values'*ones(1,↙
Income.Grid.N).*(1 + Econom_param.r) - Policy.AssetPrime.Values;


end % end of fucntion
```