In[211] := (* C cookies, 5 People function *)
Ccookies5people[numC_] := Module[{},
    (* numC is the number of cookies to divide among 5 people *)
    x = Sum[
        Sum[Sum[Sum[Sum[
            If[a1 + a2 + a3 + a4 + a5 = numC, 1, 0],
                {a5, 0, numC}],
            {a4, 0, numC}],
        {a3, 0, numC}],
    {a2, 0, numC}],
    {a1, 0, numC}];
    Return[x];
];
In[214] := Timing[Ccookies5people[40]]
Out[214] = {122.726, 135751}

In[215] := Ccookies5peopleSlightlyFaster[numC_] := Module[{},
    x = Sum[
        Sum[Sum[Sum[Sum[1,
            {a4, 0, numC - (a1 + a2 + a3)}],
        {a3, 0, numC}],
    {a2, 0, numC}],
    {a1, 0, numC}];
    Return[x];
];
In[218] := Timing[Ccookies5peopleSlightlyFaster[40]]
Out[218] = {2.66762, 135751}

In[216] := Ccookies5peoplefaster[numC_] := Module[{},
    x = Sum[
        Sum[Sum[Sum[1, {a4, 0, numC - (a1 + a2 + a3)}],
            {a3, 0, numC - (a1 + a2)}],
    {a2, 0, numC - a1}],
    {a1, 0, numC}];
    Return[x];
];
In[219] := Timing[Ccookies5peoplefaster[200]]
Out[219] = {6.61444, 70058751}
\textbf{In[219]} = CcookiesPpeople[numC_, numP_] := Module[{},
(*numC is number of cookies, numP is number of people *)
count = 0; (* sets the number of successes to 0*)
(* key
idea: count from 0 to (numC+1)^{(numP-1)-1 in base numC+1 *)
(* the digits are going to be 0, 1, ..., numC *)
(* if the sum is at most numC then
  can give the remaining cookies to new person*)
(* IntegerDigits converts n to a list of numP-1 digits *)
(* the second argument is the base,
  the third pads 0 digits in front *)
(* this counts POORLY, but advantage is
don't need to know number of people*)
For[n = 0, n \leq (numC + 1)^{(numP - 1) - 1}, n++,
{
  x = IntegerDigits[n, numC + 1, numP - 1];
  If[Sum[x[[i]], {i, 1, numP - 1}] \leq numC, count = count + 1];
}
Print[count];
];

\textbf{In[221]} = Timing[CcookiesPpeople[40, 5]]

135751

\textbf{Out[221]} = {18.3301, Null}

\textbf{In[222]} = For[c = 1, c \leq 20, c++, Print[Ccookies5peoplefaster[c]]]
CODE TO LOOK AT FRACTIONS
FROM 1 thru 9, each digit once
list = {};  
For[j = 1, j ≤ 9, j++, list = AppendTo[list, j]];  
permlist = Permutations[list];  
For[n = 1, n ≤ 9!, n++,  
  {  
    x = permlist[[n]];  
    If[(x[[1]] / (10 x[[2]] + x[[3]])) +  
       (x[[4]] / (10 x[[5]] + x[[6]])) +  
       (x[[7]] / (10 x[[8]] + x[[9]])) == 2, Print[x]];  
}];

Subset problem: Set of N distinct elements
Take any subset, A (N)
Take a subset of A (N), B (A (N)). How many ways can you do this?

Clear[n];
subsetfunction[n_] := Module[{},  
  count = 0;  
  list = {};  
  For[j = 1, j ≤ n, j++, list = AppendTo[list, j]];  
  sublist = Subsets[list, n];  
  For[m = 1, m ≤ Length[sublist],  
    m++, count = count + 2^Length[sublist[[m]]]];  
  (*Print[count];*)  
  Return[count];  
};

For[num = 0, num ≤ 10, num++, Print[num, " ", subsetfunction[num]]]
Math / Stat 341 : Fall 2015 :
sjm1 @williams.edu

(* Computing a 5-0 trump split among two hands *)
desk = {}; (* initialize deck to empty *)
(* assign five 1s to the deck; the 1s represent the trump suit *)
(* then we assign 21 0s, these are the non-trump *)
(* taking time and coding well can save you a LOT of trouble *)
For[n = 1, n ≤ 5, n++, desk = AppendTo[deck, 1]];
For[n = 6, n ≤ 26, n++, desk = AppendTo[deck, 0]];
Length[deck] (* makes sure got 26 cards *)
(* should have this in the program so we make sure we use the right deck, and thus will paste it below! *)
trumpsplit[numdo_] := Module[{},
  count = 0;
  deck = {}; (* initialize deck to empty *)
  For[n = 1, n ≤ 5, n++, deck = AppendTo[deck, 1]];
  For[n = 6, n ≤ 26, n++, deck = AppendTo[deck, 0]];
  For[n = 1, n ≤ numdo, n++, (* main loop of code *)
    hand = RandomSample[deck, 13]; (* randomly choose 13 cards *)
    numtrump = Sum[hand[[k]], {k, 1, 13}];
    (* note numtrump is 0 or 5 if we have a 5-0 split *)
    If[numtrump = 0 || numtrump = 5, count = count + 1];
  ]; (* end of n loop *)
  Print["Two theories: 2(1/2)^5 gave ", 6.25, ", other gave 3.913%."];
  Print["We observe ", 100. count/numdo, ","];
]

Timing[trumpsplit[1000000]]
Two theories: 2(1/2)^5 gave 6.25%, other gave 3.9%.
We observe 3.9166.
{11.2945, Null}

(* Getting exactly two kings *)
twokings[numdo_] := Module[{},
  deck = {}; (* initialize deck to empty *)
  (* 1 is a king, 0 non-king *)
  For[n = 1, n ≤ 4, n++, deck = AppendTo[deck, 1]];
  For[n = 5, n ≤ 52, n++, deck = AppendTo[deck, 0]];
  count = 0; (* initialize num of successes to 0 *)
  For[n = 1, n ≤ numdo, n++,
    hand = RandomSample[deck, 5]; (* 5 card hand *)
    numkings = Sum[hand[[k]], {k, 1, 5}];
    If[numkings = 2, count = count + 1];
  ]; (* end of n loop *)
  Print["Theory predicts prob exactly two kings is ",
    100.0 Binomial[4, 2] Binomial[48, 3] / Binomial[52, 5], ","];
  Print["Observed probability is ", 100.0 count/numdo, ","];
]

Timing[twokings[1000000]]
Theory predicts prob exactly two kings is 3.99298.
Observed probability is 3.9965.
{6.94204, Null}
(52)

(* calculating probability of a full house, queens and kings *)
(* probability is VERY small so must do a lot of simulations! *)
(* sadly the more you want to compute, the worse Mathematica is *)
(* this is not a hard code, don't really need the special fns here *)
(* would want to shift to another language that is better *)

fullkingqueens[numdo_] := Module[{},
  deck = {};
  (* initialize deck to empty *)
  (* 10 is a queen, 1 is a king, 0 non-king *)
  For[n = 1, n ≤ 4, n++, deck = AppendTo[deck, 1]];  
  For[n = 5, n ≤ 8, n++, deck = AppendTo[deck, 10]];  
  For[n = 9, n ≤ 52, n++, deck = AppendTo[deck, 0]];  
  count = 0; (* initialize num of successes to 0 *)
  For[n = 1, n ≤ numdo, n++,
    {hand = RandomSample[deck, 5]; (* 5 card hand *)
      numkings = Sum[hand[[k]], {k, 1, 5}];
      (* want full house of Qs and Ks *)
      (* sum is either 23 or 32! *)
      If[numkings = 32 || numkings == 23, count = count + 1];
    }]; (* end of n loop *)
  Print["Theory predicts prob full house (Qs and Ks) is ",
    100.0 Binomial[2, 1] Binomial[4, 3] Binomial[4, 2] / Binomial[52, 5], "."];  
  Print["Observed probability is ", 100.0 count / numdo, "."];  

];

Timing[fullkingqueens[10 000 000]]

Theory predicts prob full house (Qs and Ks) is 0.00184689.
Observed probability is 0.00168.

{71.9165, Null}

Timing[fullkingqueens[40 000 000]]

Theory predicts prob full house (Qs and Ks) is 0.00184689.
Observed probability is 0.0018925.

{298.945, Null}
fullhouse[numdo_] := Module[{},
    count = 0; (* this is our count variable, records successes *)
    deck = {}; (* initializing deck; cards are 1, 10, 100, 1000, ... *)
    For[n = 1, n ≤ 13, n++,
        For[j = 1, j ≤ 4, j++,
            deck = AppendTo[deck, 10^(n-1)]
        ]; (* end of for loops, deck created *)
    For[n = 1, n ≤ numdo, n++,
        {hand = RandomSample[deck, 5]; (* randomly chooses 5 cards *)
            value = Sum[hand[[k]], {k, 1, 5}]; (* sums the value of cards *)
            (* next few lines is a nice trick. if our hand is 10002011 this means *)
            (* we have one number repeated twice, the 1000; and have one 1, one 10, *)
            (* and one 1000000. notice how easy it is to check! the only way to have *)
            (* a full house is to have one 2 and one 3 among digits, or the product of *)
            (* non-zero digits is a 6. Easier to do MemberQ to check on 2, 3 but show both *)
            valuelist = IntegerDigits[value];
            If[
                MemberQ[valuelist, 2] == True && MemberQ[valuelist, 3] == True, count = count+1];
            (* this is how to do as a product of digits *)
            (* ----------------- *)(* trimlist = {};
            For[j = 1, j ≤ Length[valuelist], j++,
                If[valuelist[[j]] > 1, trimlist = AppendTo[trimlist,valuelist[[j]]]
            ];
            If[Product[trimlist[[j]],{j,1,Length[trimlist]}] == 6, count = count+1];
            --------------- *)
        ]; (* end of n loop *)
        Print["Prob of a full house: ", 100.0 Binomial[13, 2]
        Print["Observed prob: ", 100.0 count/numdo, "]%."];
    ];

Timing[fullhouse[2000000]]

Prob of a full house: 0.144058%.
Observed prob: 0.14615%.
{69.4828, Null}