The Birthday Problem

Relevant Formulas:

The probability of at least one match among *m* people is represented by $P_x(m) := 1 - \frac{n!/(n-m)!}{n^m}$

In[1]:=
```
P[n_, m_] := 1 - (n! / (n - m)!) / n^m
?? P
```

Out[2]=

Symbol	
Global`P	
Definitions	
P[n_, m_] := $1 - \frac{n!}{(n-m)!\, n^m}$	
Full Name	Global`P

The probability that the m^{th} person made the first match is represented by $p_x(m) := \frac{n!/(n-(m-1))! \cdot (m-1)}{n^m}$

In[3]:=
```
p[n_, m_] := (n! / (n - (m - 1))!) * ((m - 1) / n^m)
?? p
```

Out[4]=

Symbol	
Global`p	
Definitions	
p[n_, m_] := $\frac{n!\,(m-1)}{(n-(m-1))!\, n^m}$	
Full Name	Global`p

The natural formula for the expected value is the cumulative product of the probability of each event and its value: $\sum_{i=a}^{b} i \cdot p_x(i)$

In[5]:=
```
naturalExpected[n_] := Sum[m * n! * (m - 1) / ((n - (m - 1))! * n^m), {m, 1, n + 1}]
?? naturalExpected
```

Out[6]=

Symbol

Global`naturalExpected

Definitions

naturalExpected[n_] := $\sum_{m=1}^{n+1} \frac{m\, n!\, (m-1)}{(n-(m-1))!\, n^m}$

Full Name Global`naturalExpected

The alternative formula for the expected value uses the cumulative probability of failure:

$$a + \sum_{i=a}^{b} 1 - P_x(i)$$

In[7]:=
```
expected[n_] := 1 + Sum[n! / ((n - m)! * n^m), {m, 1, n}]
?? expected
```

Out[8]=

Symbol

Global`expected

Definitions

expected[n_] := 1 + $\sum_{m=1}^{n} \frac{n!}{(n-m)!\, n^m}$

Full Name Global`expected

We can see that both expectation value formulas are equivalent via simplification:

In[9]:=
```
FullSimplify[naturalExpected[x] == expected[x]]
```

Out[9]= True

However, the alternative formula is much more suitable for large calculations (efficiency and avoiding underflow).

Calculations:

In[10]:=
```
n = 365
```

Out[10]= 365

The link between expected value and probability:

We are interested in the link between the expected value and probability, which would equate to asking:

We know we can calculate the probability of a match (=at least one match) for any given number of people in the room,

but what <u>probability threshold</u> do we need to reach before we can **expect** to have a match? - Not an easy question.
Some may try to argue that the threshold lies at 50%, but there seems to be no apparent reason as to why we should
expect a match when the chance of a match reaches 50%. We could argue the same for 75% or 90% - it's arbitrary.
But a well-defined threshold does exist - for every probability experiment, in fact - it's simply the <u>probability of the expected value</u> itself.
All we have to do in the birthday problem is calculate the expected value of people needed for a match given *n* days in a year,
then calculate the probability of actually having a match with a room full of that many people.

```
In[11]:= N[expected[n]]
Out[11]= 24.6166
```

```
In[12]:= N[P[n, N[expected[n]]]]
Out[12]= 0.557151
```

We observe that our P_x can be applied to non-integer values despite the birthday problem theoretically being a discrete probability experiment.
However, Mathematica can have some issues with underflow for p_x and P_x for some non-integer values (large fractions), so it's helpful to define interpolated functions for both:

```
In[13]:= pInt = Interpolation[p[n, #] & /@ Range[n + 1]]
         PInt = Interpolation[P[n, #] & /@ Range[n + 1]]
Out[13]= InterpolatingFunction[ Domain: {{1, 366}} Output: scalar ]
Out[14]= InterpolatingFunction[ Domain: {{1, 366}} Output: scalar ]
```

Still, we can trust that P_x is correct even for non-integer values, as we get a nearly identical result when using its interpolated counterpart:

```
In[15]:= N[PInt[expected[n]]]
Out[15]= 0.557151
```

This essentially means that we trust the gamma function, which is used for non-integer factorials, so the same can also be said for p_x:

```
In[16]:= {N[p[n, N[expected[n]]]], N[pInt[expected[n]]]}
Out[16]= {0.0306359, 0.0306359}
```

Sadly, Mathematica can't compute the numeric limit of this probability as the days in a year approach infinity, but it seems to be very close to 54%.

```
In[17]:= Timing[N[Limit[P[x, expected[x]], x → 100000]]] {"seconds elapsed", "calculated limit"}
Out[17]= {0.84375 seconds elapsed, 0.544837 calculated limit}
```

However, when setting the limit to infinity, Mathematica was able to find a much faster equation to immediately calculate the probability threshold given *n* days in a year:

In[18]:=
```
thresh[n_] :=
 1 - (n^(-e^n * n^(-n) * Gamma[n + 1, n]) * n!) / ((n - e^n * n^(-n) * Gamma[n + 1, n])!)
?? thresh
```

Out[19]=

Symbol

Global`thresh

Definitions

$$\text{thresh}[n\_] := 1 - \frac{n^{-e^n n^{-n}\,\text{Gamma}[n+1,n]}\; n!}{\left(n - e^n n^{-n}\,\text{Gamma}[n+1,n]\right)!}$$

Full Name Global`thresh

In[20]:=
```
FullSimplify[P[x, expected[x]] == thresh[x]]
```

Out[20]= True

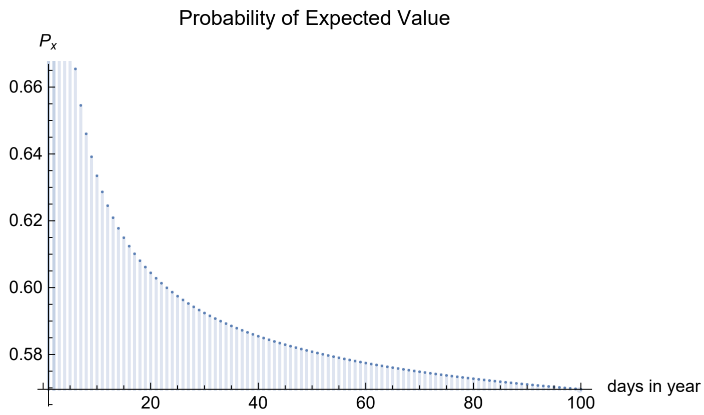
In[21]:=
```
Timing[N[thresh[100000]]] {"seconds elapsed", "calculated value"}
```

Out[21]= {0.3125 seconds elapsed, 0.544837 calculated value}

This isn't without its drawbacks, though. Due to its composition, it has underflow issues for some values, including integer ones.
For this reason, it's safer to visualize the trend of the probability threshold as *n* increases with our original P_x or its interpolated counterpart.

In[22]:=
```
DiscretePlot[P[x, N[expected[x]]], {x, 1, 100},
 PlotLabel → "Probability of Expected Value", AxesLabel → {"days in year", "Px"}]
```

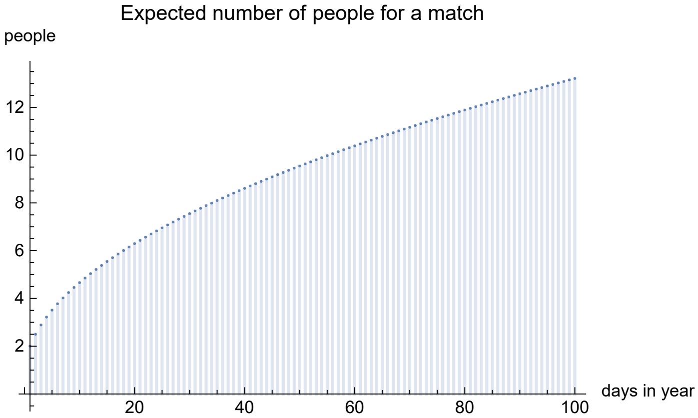
Out[22]=



The expected number of people to let into a room until a match occurs grows according to the square root of the number of days in the year:

In[23]:=
```
DiscretePlot[expected[x], {x, 1, 100},
 PlotLabel → "Expected number of people for a match",
 AxesLabel → {"days in year", "people"}]
```

Out[23]=



This is one of the most practical takeaways of the birthday problem.
If we know how the expected value scales as *n* increases,
we can calculate the complexity *O* of an algorithm involving the birthday problem,
and make quick, reliable calculations for different versions of the birthday problem in general.

In[24]:=
```
expectedSimplified[n_] := ⅇ^n * n^(-n) * Gamma[n + 1, n]
?? expectedSimplified
```

Out[25]=

Symbol
Global`expectedSimplified
Definitions expectedSimplified[n_] := $e^n n^{-n}$ Gamma[n + 1, n]
Full Name Global`expectedSimplified

In[26]:=
```
FullSimplify[expected[x] == expectedSimplified[x]]
```

Out[26]= True

In[27]:=
```
t = Table[{x, N[expected[x]]}, {x, 1, 1000}];
```

In[28]:=
```
FindFormula[t, x]
```

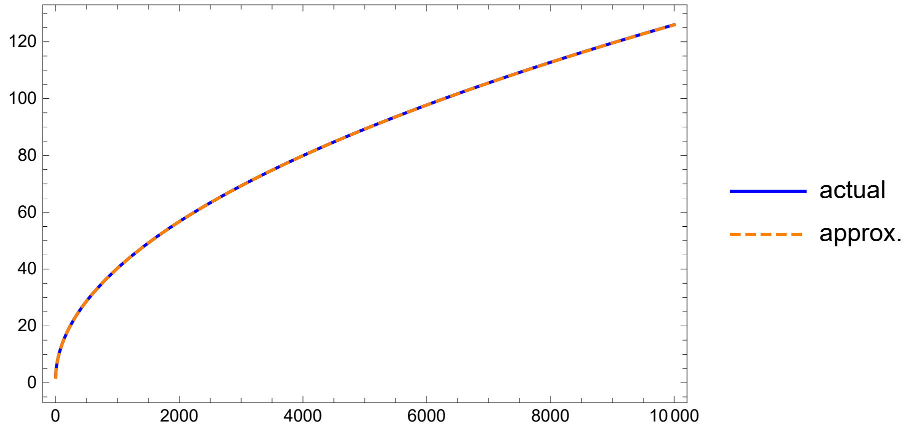
Out[28]= $0.684004 + 1.25279\, x^{0.5}$

In[29]:=
```
f[x_] = 0.684004 + 1.25279 * Sqrt[x]
```

Out[29]= $0.684004 + 1.25279 \sqrt{x}$

In[30]:=
```
Plot[{expected[x], f[x]}, {x, 1, 10000}, Frame → True,
 PlotStyle → {Blue, Directive[Orange, Dashed]},
 PlotLegends → {"actual", "approx."}]
```

Out[30]=



Important note - Everything can be done using the probability distribution p_x:

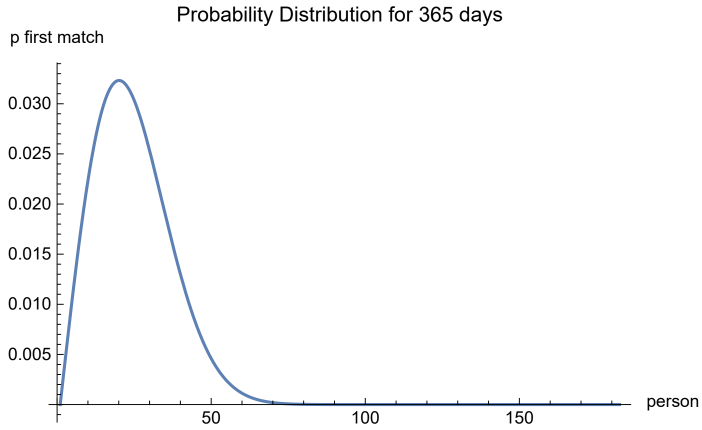
But be warned: We are dealing with a discrete probability distribution, meaning that there are limits to our analytical capabilities with p_x.

p_x may be calculable for most non-integer values, but underflow issues worsen in its derivative and become unavoidable in its integral.

We can calculate approximations by using our interpolation of p_x, but it has its limits regarding accuracy as well, particularly when it's integrated.

In[31]:=
```
Plot[p[n, x], {x, 1, n / 2},
 PlotLabel → StringForm["Probability Distribution for `` days", n],
 AxesLabel → {"person", "p first match"}]
```

Out[31]=



Mode:

The mode of a probability distribution is defined to be its maximum.

In this case, the mode can be understood as the person that is most likely to create the first match after entering the room.

We know that p_x can be applied to non-integer values, but finding a maximum analytically is too hard for Mathematica (due to underflow).

An easy workaround is to find the maximum of our interpolated p_x:

```
In[32]:= max = FindMaximum[pInt[x], x]
Out[32]= {0.0323209, {x → 20.1088}}
```

Thus, the mode is:

```
In[33]:= mode = x /. max[[2]]
Out[33]= 20.1088
```

We can and should also find the mode for discrete values:

```
In[34]:= modeDiscrete = Extract[{Floor[mode], Ceiling[mode]},
           Position[{p[n, Floor[mode]], p[n, Ceiling[mode]]},
            Max[{p[n, Floor[mode]], p[n, Ceiling[mode]]}]]]
Out[34]= {20}
```

As well as the corresponding probability:

```
In[35]:= N[p[n, #] & /@ modeDiscrete]
Out[35]= {0.0323199}
```

Note: We are using tuples in the discrete case, since there can be two discrete modes for some values of n (e.g. 342).

Median:

The median of a probability distribution is defined to be the point at which the cumulative probability reaches $\frac{1}{2}$.

In this case, the median can be understood as the person that brings the probability of a match over 50% after entering the room.

The second we see “cumulative probability”, we should directly think of P_x.

However, it is important to keep in mind that $P_x(x) \equiv \int_1^m p_x(x)\, dx$ for the continuous case and $P_x(m) \equiv \sum_{i=1}^{m} p_x(i)$ for the discrete case.

```
In[36]:= FullSimplify[P[n, x] == Sum[p[n, i], {i, 1, x}], x ∈ Integers]
Out[36]= True
```

This means that we can use our P_x to verify that our calculations are correct, but our goal remains to use only the probability distribution.

```
In[37]:= pIntIntegral[x_] = Integrate[pInt[x], x]
Out[37]= InterpolatingFunction[ Domain: {{1, 366}} Output: scalar ][x]
```

```
In[38]:= interpolatedMedian = x /. FindRoot[pIntIntegral[x] == 0.5, {x, expected[n]}]
Out[38]= 23.2611
```

This is where the integral of our interpolation of p_x reaches $\frac{1}{2}$, but let's verify it with P_x:

In[39]:=
```
P[n, interpolatedMedian]
```
Out[39]=
```
0.515464
```

So, P_x doesn't seem to approve completely. Let's try and find the true median using the interpolation of P_x:

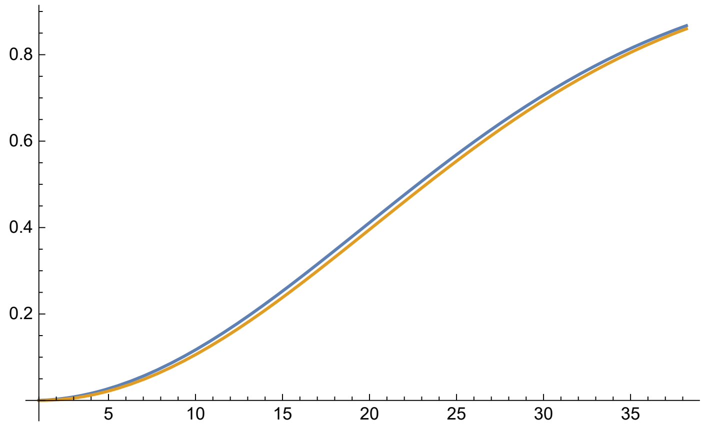
In[40]:=
```
median = x /. FindRoot[PInt[x] == 0.5, {x, expected[n]}]
```
Out[40]=
```
22.7677
```

In[41]:=
```
P[n, median]
```
Out[41]=
```
0.5
```

It stands to reason that the interpolation of P_x is more accurate than the integral of the interpolation of p_x.

We can see that they don't differ by much, but it's enough to make a notable difference when finding the median. Here are the plots of both:

In[42]:=
```
Plot[{PInt[x], pIntIntegral[x]}, {x, 1, 2 * Sqrt[n]}]
```
Out[42]=



We can and should also find the median for discrete values:

In[43]:=
```
medianDiscrete = First@FirstPosition[Accumulate[p[n, #] & /@ Range[n + 1]], x_ /; x > 0.5]
```
Out[43]=
```
23
```

Mean (Expected Value):

The expected value is the most straight forward component to be derived from the probability distribution.

In this case, the mean can be understood as the number of people we expect to need to let into the room before we have our first match.

We named it in the beginning; it's simply the product of each probability in our distribution and its value for a discrete distribution.

In[44]:=
```
mean = N[Sum[p[n, x] * x, {x, 1, n + 1}]]
```
Out[44]=
```
24.6166
```

This time around, it actually isn't necessary to calculate a "discrete mean", since a non-integer mean/expected value is well-defined,
even for discrete probability distributions. If we were to, we would just round to the nearest integer.

In[45]:=
```
meanDiscrete = Round[mean]
```
Out[45]=
```
25
```

Probability Threshold:

I defined before what I call the "Probability Threshold" to be the probability of a match at which we can rightfully say that we expect a match.
It can be calculated by taking the integral of the probability distribution from the beginning to the expected value.

In[46]:=
```
N[pIntIntegral[expected[n]]]
```
Out[46]=
```
0.54212
```

This isn't what we saw before, and we can easily see this result is very inaccurate by comparing with P_x:

In[47]:=
```
N[P[n, N[expected[n]]]]
```
Out[47]=
```
0.557151
```

This is undoubtedly the same issue as before, namely that taking the integral of our interpolation of p_x cost us too much accuracy.