# Birthday Problem

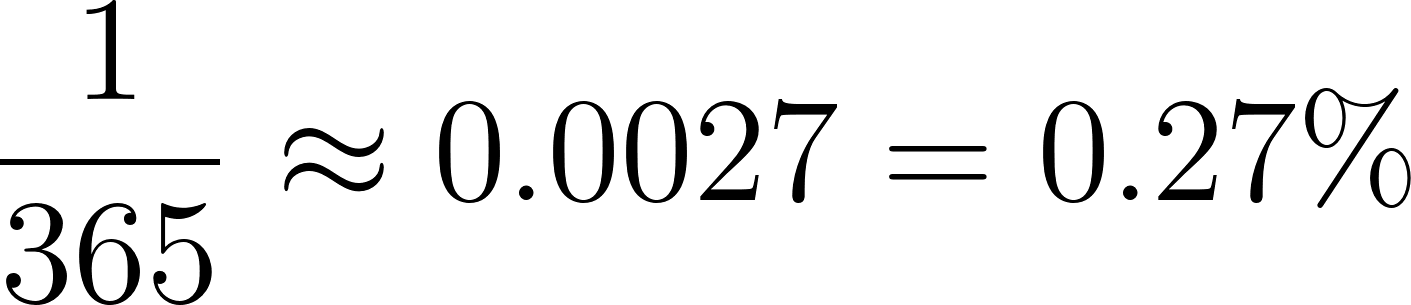
[The Birthday Problem](https://en.wikipedia.org/wiki/Birthday_problem), also known as the Birthday Paradox, is a classic problem in probability theory that deals with the likelihood of at least two people in a group sharing the same birthday. Although it's called a "paradox," it's not a true paradox but rather a surprising and counterintuitive result in probability. The Birthday Problem was first introduced by the Hungarian mathematician [Richard von Mises](https://en.wikipedia.org/wiki/Richard_von_Mises) in 1939. It gained popularity as an illustration of how human intuition often fails to grasp the true nature of probability and chance.

The problem can be stated as follows:

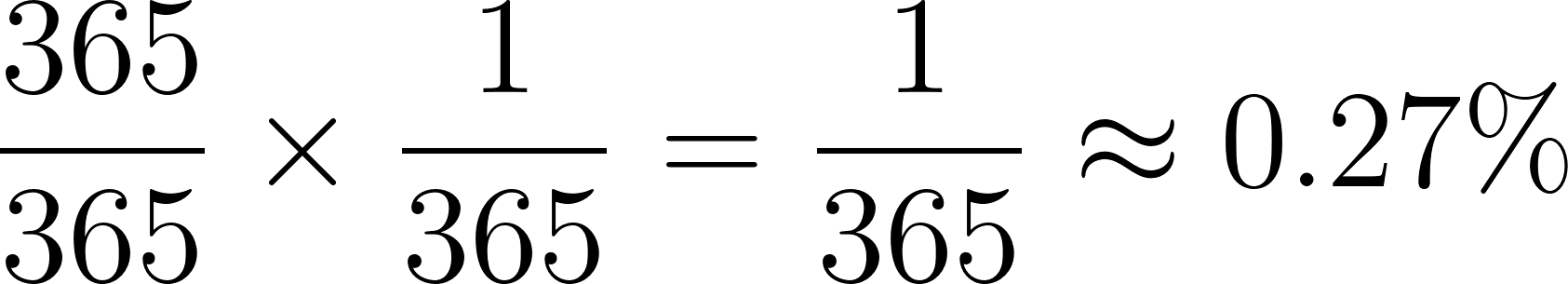
In a group of people, what is the probability that at least two of them have the same birthday? For simplicity, we assume there are 365 days in a year (ignoring leap years), and all birthdays are equally likely.

Solution for 2 people in the room:

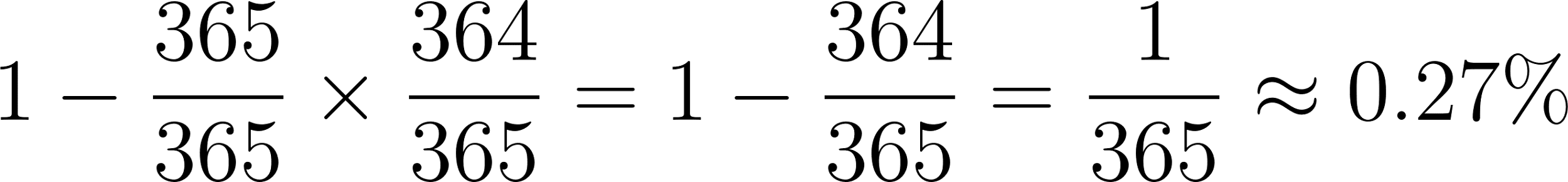
There are 365 possible birthdays for the first person, and only 1 of those days would result in a shared birthday with the second person. Thus, the probability of a shared birthday is

[](https://www.codecogs.com/eqnedit.php?latex=%5Cdfrac%7B1%7D%7B365%7D%20%5Capprox%200.0027%3D0.27%25#0)

A slightly different way to think about it is the first person has 365 days to “choose” from out of 365. The second person then only has 1 way.

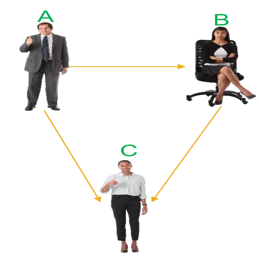
[](https://www.codecogs.com/eqnedit.php?latex=%5Cdfrac%7B365%7D%7B365%7D%20%5Ctimes%20%5Cfrac%7B1%7D%7B365%7D%3D%5Cfrac%7B1%7D%7B365%7D%5Capprox%200.27%25#0)

Yet another way to think about it (trust me) is to find the probability that they have different birthdays and then subtract that from 1 to find the probability that they share a birthday. The first person can have any of the 365 days as their birthday. The second person has 364 days to choose from. The resulting calculation is

[](https://www.codecogs.com/eqnedit.php?latex=1-%5Cdfrac%7B365%7D%7B365%7D%5Ctimes%20%5Cdfrac%7B364%7D%7B365%7D%3D1-%5Cdfrac%7B364%7D%7B365%7D%3D%5Cdfrac%7B1%7D%7B365%7D%5Capprox%200.27%25#0)

Solution for 3 people in the room:

When there are three people in the room, it is harder to use the first two methods we used for 2 people in the room. This is because you must do multiple comparisons between sets of different people (similar problem to the [Handshake Problem](https://mathworld.wolfram.com/HandshakeProblem.html)).

In the image above, we need to consider the following scenarios:

A has the same birthday as B.

A has the same birthday as C.

B has the same birthday as C.

A, B, and C all have the same birthday.

Another issue is that there is some overlap (e.g., A having the same birthday as B would also be counted in the A, B, and C all having the same birthday situation). In order to calculate the probability, we need to consider the following events:

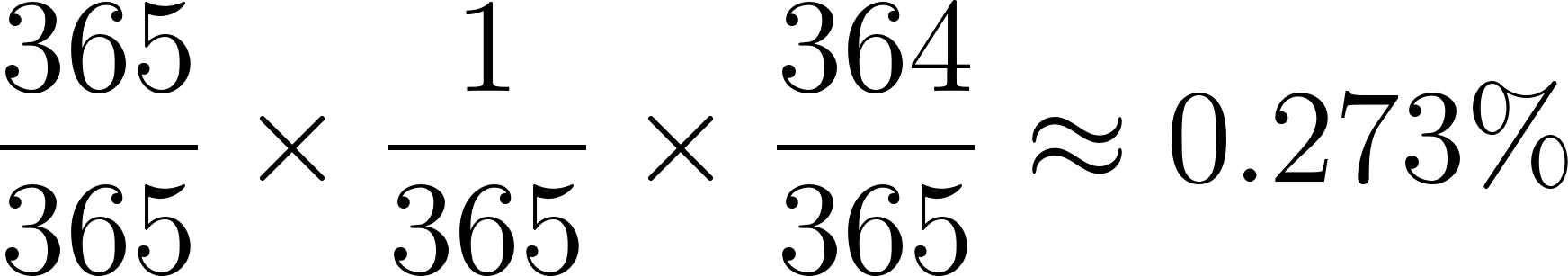
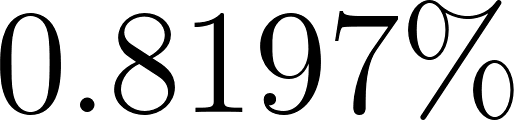
A has the same birthday as B and C has a different birthday.

A has the same birthday as C and B has a different birthday

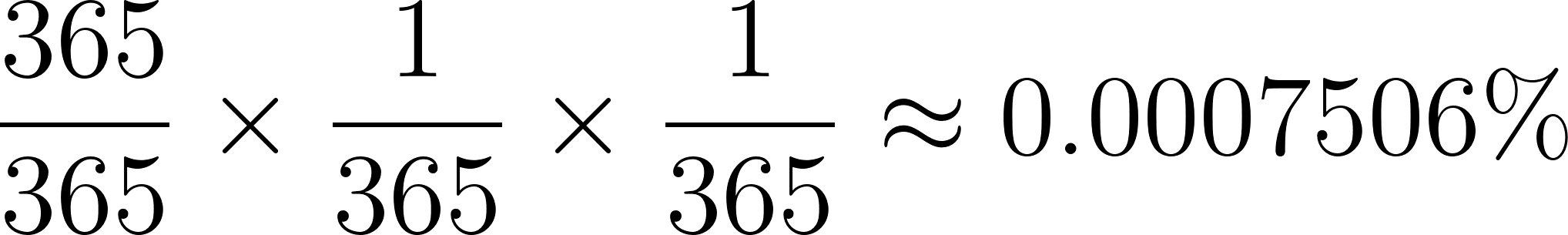
B has the same birthday as C and A has a different birthday.

A, B, and C all have the same birthday.

The first three will all result in the same probability. To calculate this, the first person can have a birthday on any of the 365 days. The second person has only one day that can be their birthday if they are to have the same birthday as the first person. The third person can have their birthday on any of the remaining 364 days.

[](https://www.codecogs.com/eqnedit.php?latex=%5Cdfrac%7B365%7D%7B365%7D%5Ctimes%20%5Cdfrac%7B1%7D%7B365%7D%5Ctimes%20%5Cdfrac%7B364%7D%7B365%7D%20%5Capprox%200.273%25#0) (this occurs three times so the total probability is [](https://www.codecogs.com/eqnedit.php?latex=0.8197%25#0))

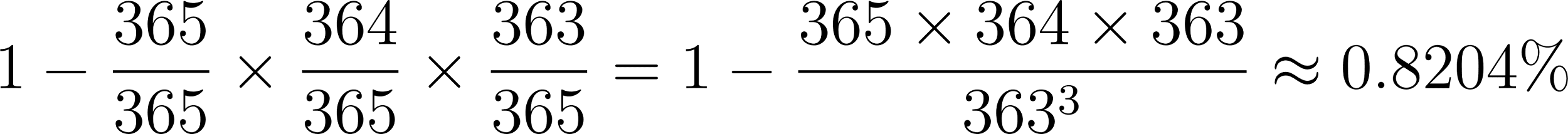
For the scenario of all three having the same birthday, the first person can have a birthday on any of the 365 days. The second and third person must have the same birthday so they can only have 1 day available for their birthday.

[](https://www.codecogs.com/eqnedit.php?latex=%5Cdfrac%7B365%7D%7B365%7D%5Ctimes%20%5Cdfrac%7B1%7D%7B365%7D%5Ctimes%20%5Cdfrac%7B1%7D%7B365%7D%20%5Capprox%200.0007506%25#0)

Adding these together we get [](https://www.codecogs.com/eqnedit.php?latex=0.8204%25#0).

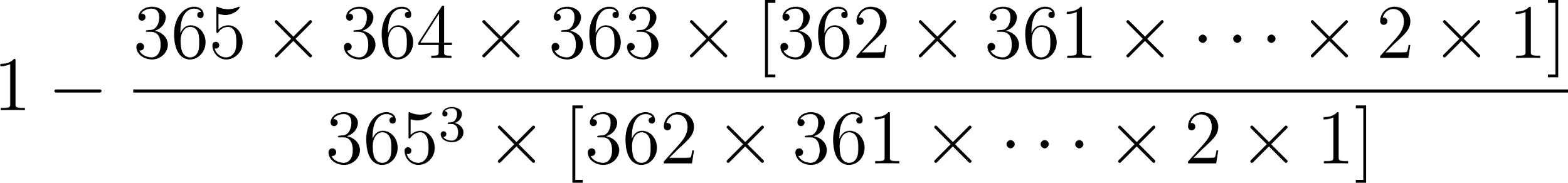
Imagine adding another person and another person. This method will get annoying extremely quickly as you need to account for all of the different combinations.

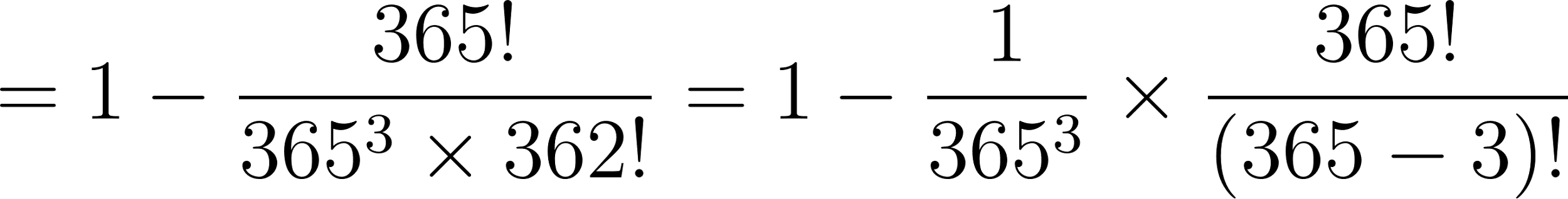
For three people, we can calculate the probability that they all have different birthdays and then subtract that from 1 to find the probability of at least one shared birthday. The first person can have any of the 365 days as their birthday. The second person has a 364/365 chance of not sharing a birthday with the first person, and the third person has a 363/365 chance of not sharing a birthday with the other two.

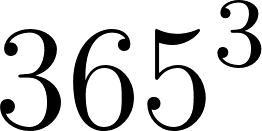
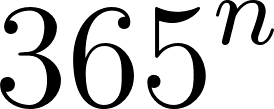
[](https://www.codecogs.com/eqnedit.php?latex=1-%5Cfrac%7B365%7D%7B365%7D%20%5Ctimes%20%5Cfrac%7B364%7D%7B365%7D%20%5Ctimes%20%5Cfrac%7B363%7D%7B365%7D%3D1-%5Cdfrac%7B365%5Ctimes%20364%20%5Ctimes%20363%7D%7B363%5E3%7D%20%5Capprox%200.8204%25#0)

Amazingly, we get the same answer we got doing it the harder way. Finding the probability of the complementary event (that no one shares a birthday) is much easier to calculate.

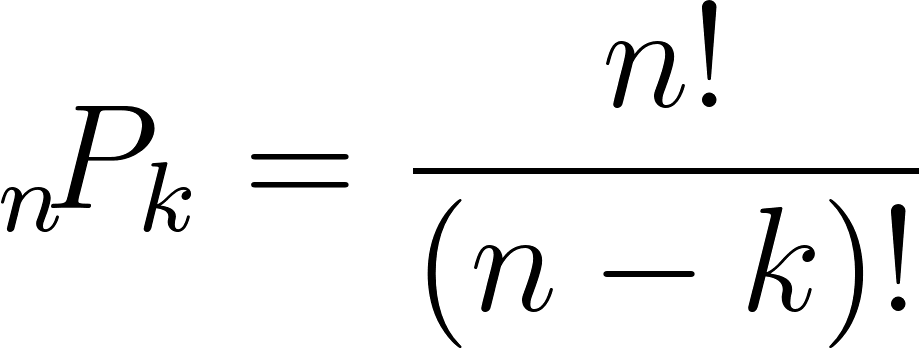
Let’s get creative with our result from the 3 person scenario. We can rewrite the expression above the following way:

[](https://www.codecogs.com/eqnedit.php?latex=1-%5Cdfrac%7B365%20%5Ctimes%20364%20%5Ctimes%20363%20%5Ctimes%20%5B362%20%5Ctimes%20361%20%5Ctimes%20%5Cdots%20%5Ctimes%202%20%5Ctimes%201%5D%7D%7B365%5E3%20%5Ctimes%20%5B362%20%5Ctimes%20361%20%5Ctimes%20%5Cdots%20%5Ctimes%202%20%5Ctimes%201%5D%7D#0)

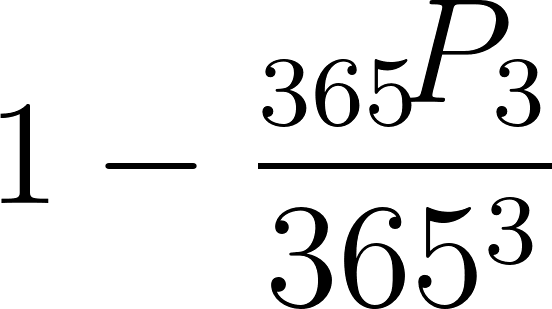
[](https://www.codecogs.com/eqnedit.php?latex=%3D1-%5Cdfrac%7B365!%7D%7B365%5E3%20%5Ctimes%20362!%7D%3D1-%5Cdfrac%7B1%7D%7B365%5E3%7D%5Ctimes%20%5Cdfrac%7B365!%7D%7B(365-3)!%7D#0)

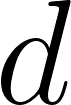
From the resulting expression we see [](https://www.codecogs.com/eqnedit.php?latex=365%5E3#0) as this comes from there being 3 people. For any group of [](https://www.codecogs.com/eqnedit.php?latex=n#0) people, the denominator in this expression will need to be [](https://www.codecogs.com/eqnedit.php?latex=365%5En#0). The next fraction has the number of days factorial and the denominator has the number of days minus the number of people factorial.

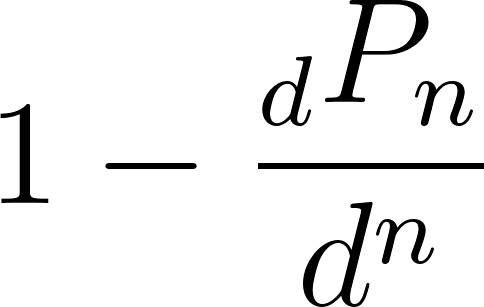
The definition of a [permutation](https://en.wikipedia.org/wiki/Permutation) is

[](https://www.codecogs.com/eqnedit.php?latex=%7B%7D_%7Bn%7D%5C!P_%7Bk%7D%3D%5Cdfrac%7Bn!%7D%7B(n-k)!%7D#0)

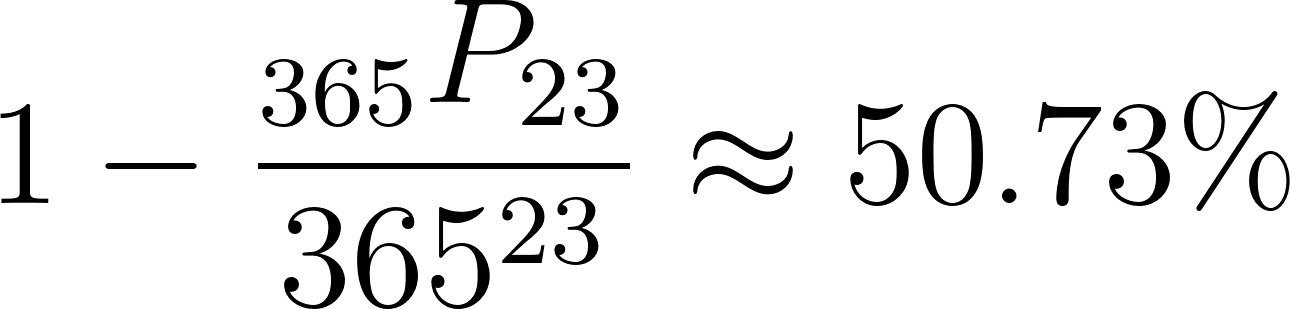
We can rewrite the expression above as

[](https://www.codecogs.com/eqnedit.php?latex=1-%5Cdfrac%7B%7B%7D_%7B365%7D%5C!P_%7B3%7D%7D%7B365%5E3%7D#0)

To generalize for [](https://www.codecogs.com/eqnedit.php?latex=n#0) people and [](https://www.codecogs.com/eqnedit.php?latex=d#0) days, the probability that [](https://www.codecogs.com/eqnedit.php?latex=n#0) people share the same birthday is

[](https://www.codecogs.com/eqnedit.php?latex=1-%5Cdfrac%7B%7B%7D_%7Bd%7DP_%7Bn%7D%7D%7Bd%5En%7D#0)

As [](https://www.codecogs.com/eqnedit.php?latex=n#0) increases, the probability of at least one shared birthday grows rapidly. For example, with a group of 23 people, the probability of at least one shared birthday is over 50%. This result is often surprising because it's much lower than what our intuition suggests, which is why the Birthday Problem is considered a fascinating illustration of probability theory.

[](https://www.codecogs.com/eqnedit.php?latex=1-%5Cdfrac%7B%7B%7D_%7B365%7DP_%7B23%7D%7D%7B365%5E%7B23%7D%7D%5Capprox%2050.73%25#0)

## Birthday Problem R Code

This R code demonstrates how to calculate the probability of at least two people sharing a birthday in a group (the Birthday Problem) using various methods, including built-in functions, custom functions, and simulations. It covers groups of 2 and 3 people. The code starts with preliminary results for demonstration purposes, using the built-in pbirthday function to calculate the probability for 2 people. It then calculates the same probability using basic arithmetic and a custom function p\_no\_match. After that, the code simulates the Birthday Problem for 2 people by generating random birthdays and checking for duplicates. It repeats this simulation 100,000 times (1e5) to approximate the probability. It then compares the probability obtained through simulation to the exact probability calculated using pbirthday. The code continues by doing the same for a group of 3 people. It calculates the probability using pbirthday, basic arithmetic, and simulation. It also compares the simulation probability to the exact probability. Finally, the code introduces a custom function my\_pbirthday that uses the permutation formula to calculate the probability.

*# R has a built in function, pbirthday*  
*# Check ?pbirthday for documentation*  
message("Preliminary results for demonstration")

## Preliminary results for demonstration

pbirthday(n = 2, classes = 365)

## [1] 0.002739726

1 / 365

## [1] 0.002739726

1 - (365/365)\*(364/365)

## [1] 0.002739726

*# Custom function to calculate probability there are no matches*  
*# with d days and n people*  
p\_no\_match <- **function**(d, n) {  
 prod(d:(d - n + 1)) / d^n   
}  
1 - p\_no\_match(365,2)

## [1] 0.002739726

message("\n\n----- 2 People in Room -----")

##   
##   
## ----- 2 People in Room -----

*# Simulation approach*  
*# Generate a sample of 2 days and check if they match*  
set.seed(6644)  
(birthdays <- sort(sample(365, size = 25, replace = TRUE)))

## [1] 7 8 11 28 38 62 66 80 87 120 135 155 171 176 186 187 204 213 229  
## [20] 266 276 283 288 312 328

message(paste("Any duplicated birthdays? ", any(duplicated(birthdays))))

## Any duplicated birthdays? FALSE

*# Do this many times to approximate the probability calculated above*  
check\_sample <- **function**(d, n) {  
 birthdays <- sample(d, size = n, replace = TRUE)  
 any(duplicated(birthdays))  
}  
  
results <- replicate(1e5, check\_sample(365, 2))  
message(paste("Probability from simulation: ", mean(results)))

## Probability from simulation: 0.0028

message(paste("Probabiltiy from prbirthday: ", pbirthday(n = 2, classes = 365)))

## Probabiltiy from prbirthday: 0.00273972602739725

*# See how far off we were*  
diff <- mean(results) - pbirthday(n = 2, classes = 365)  
message(paste("Difference between simulation and exact: ", diff))

## Difference between simulation and exact: 6.0273972602751e-05

*# Now try for 3 people in the same room*  
message("\n\n----- 3 People in Room -----")

##   
##   
## ----- 3 People in Room -----

message(paste("Probability from pbirthday: ", pbirthday(n = 3, classes = 365)))

## Probability from pbirthday: 0.00820416588478134

message(paste("Probability from doing 1 minus method: ",  
 1 - (365/365)\*(364/365)\*(363/365)))

## Probability from doing 1 minus method: 0.00820416588478134

results <- replicate(1e5, check\_sample(365, 3))  
message(paste("Probability from simulation: ", mean(results)))

## Probability from simulation: 0.00824

*# See how far off we were*  
diff <- mean(results) - pbirthday(n = 3, classes = 365)  
message(paste("Difference between simulation and exact: ", diff))

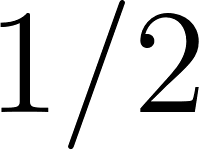
## Difference between simulation and exact: 3.5834115218656e-05

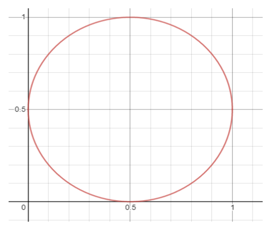
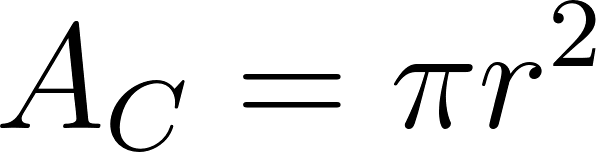
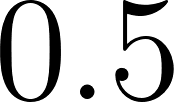
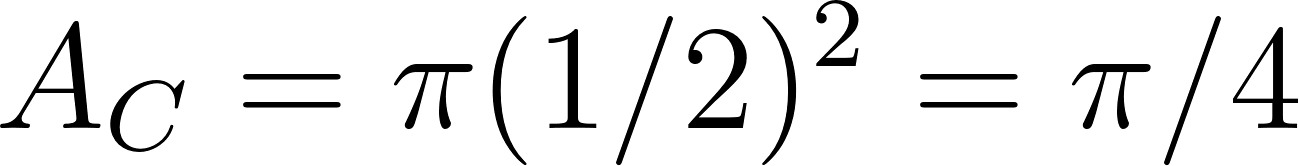
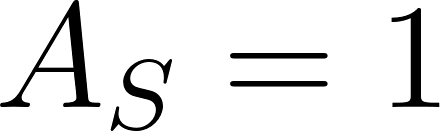
*# Using permutation formula*  
perm <- **function**(n, k) {  
 prod(n:(n-k+1))  
}  
my\_pbirthday <- **function**(n, classes) {  
 1 - perm(classes, n) / classes^n  
}  
  
message(paste("Probability from doing 1 minus method in a custom function: ",  
 my\_pbirthday(n = 3, classes = 365)))

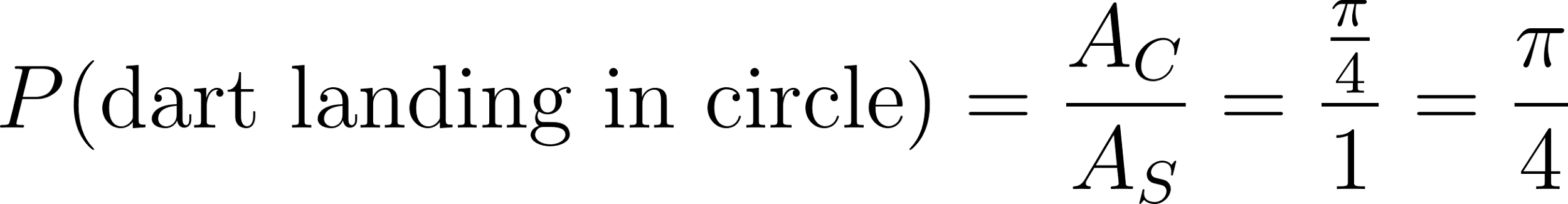
## Probability from doing 1 minus method in a custom function: 0.00820416588478134

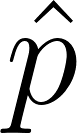
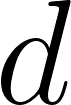
# Let’s Make Some Pi

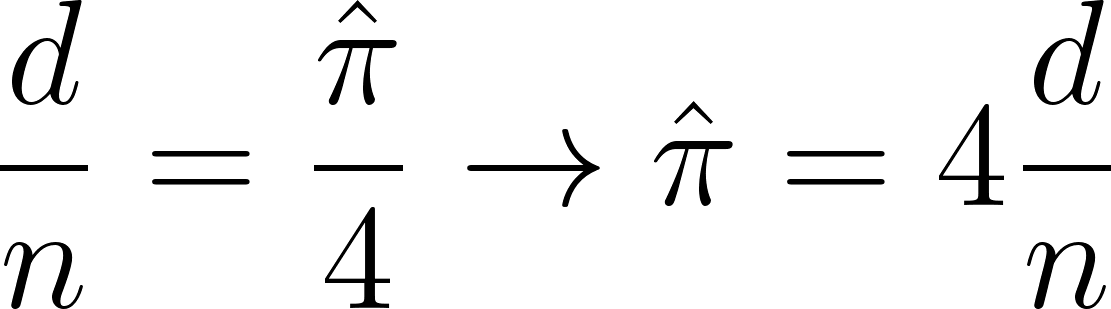
Estimating pi by throwing random darts at a circle inscribed in a unit square is a classic example of the Monte Carlo method, a computational technique that uses random sampling to solve mathematical problems. The method of estimating pi in this way is also known as the "[Buffon's needle problem](https://docs.google.com/document/u/0/d/1ZclRnU_9WXWKNiKzPauxuGqVrtAFfbVUXJYUfb-4BtQ/edit)" or the "[dart method](https://en.wikipedia.org/wiki/Approximations_of_%CF%80#Summing_a_circle's_area)."

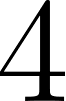
[The Monte Carlo method](https://en.wikipedia.org/wiki/Monte_Carlo_method) was named after the famous casino in Monaco, as it relies on random sampling and probability, much like gambling. The technique was developed during the 1940s by [Stanislaw Ulam](https://en.wikipedia.org/wiki/Stanislaw_Ulam), [John von Neumann](https://en.wikipedia.org/wiki/John_von_Neumann), and [Nicholas Metropolis](https://en.wikipedia.org/wiki/Nicholas_Metropolis) while working on the [Manhattan Project](https://en.wikipedia.org/wiki/Manhattan_Project). However, the idea of using random sampling to estimate pi dates back to the 18th century, with the work of [Georges-Louis Leclerc, Comte de Buffon](https://en.wikipedia.org/wiki/Georges-Louis_Leclerc,_Comte_de_Buffon), a French mathematician, and naturalist. The problem can be stated as follows: Given a circle of radius [](https://www.codecogs.com/eqnedit.php?latex=1%2F2#0)inscribed in a square of side length 1, we can estimate the value of pi by throwing random darts at the square and counting the number of darts that land inside the circle.

The circle's area can be represented by the formula [](https://www.codecogs.com/eqnedit.php?latex=A_C%20%3D%20%5Cpi%20r%5E2#0), where [](https://www.codecogs.com/eqnedit.php?latex=r#0) is the radius. Since the radius of the circle is [](https://www.codecogs.com/eqnedit.php?latex=0.5#0), the circle's area is [](https://www.codecogs.com/eqnedit.php?latex=A_C%20%3D%20%5Cpi(1%2F2)%5E2%20%3D%7B%5Cpi%7D%2F%7B4%7D#0). The square's area is [](https://www.codecogs.com/eqnedit.php?latex=%20A_S%20%3D%201#0).

[](https://www.codecogs.com/eqnedit.php?latex=P(%5Ctext%7Bdart%20landing%20in%20circle%7D)%3D%5Cdfrac%7BA_C%7D%7BA_S%7D%3D%5Cdfrac%7B%5Cfrac%7B%5Cpi%7D%7B4%7D%7D%7B1%7D%3D%5Cdfrac%7B%5Cpi%7D%7B4%7D" \l "0)

We can get an estimate of the probability [](https://www.codecogs.com/eqnedit.php?latex=%5Chat%7Bp%7D" \l "0) by “throwing” random darts at the square and finding the proportion that land in the circle. Let [](https://www.codecogs.com/eqnedit.php?latex=d#0) be the number of darts that land in the circle and [](https://www.codecogs.com/eqnedit.php?latex=n#0) be the total number of darts thrown.

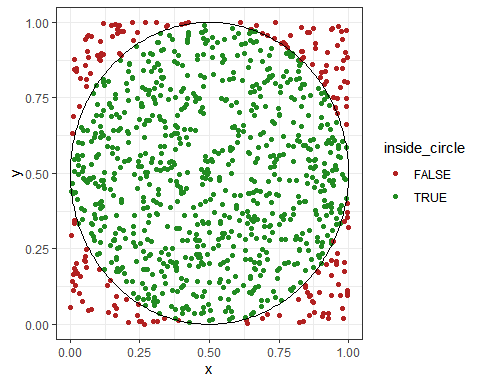
[](https://www.codecogs.com/eqnedit.php?latex=%5Cdfrac%7Bd%7D%7Bn%7D%3D%5Cfrac%7B%5Chat%7B%5Cpi%7D%7D%7B4%7D%20%5Crightarrow%20%7B%5Chat%7B%5Cpi%7D%7D%3D4%5Cfrac%7Bd%7D%7Bn%7D#0)

The estimate of [](https://www.codecogs.com/eqnedit.php?latex=%5Cpi#0) is [](https://www.codecogs.com/eqnedit.php?latex=4#0) times the proportion of darts that land in the circle.

## Let’s Make Some Pi R Code

This R code demonstrates how to approximate the value of pi using the Monte Carlo method, by simulating random darts thrown at a circle inscribed in a square, as previously discussed. The code includes visualizations using the [ggplot2](https://ggplot2.tidyverse.org/) and [ggforce](https://ggforce.data-imaginist.com/) packages.

* Load the required libraries (ggplot2 and ggforce).
* Generate 1000 random x and y coordinates in the range of 0 to 1.
* Determine if the points lie inside the circle using the [distance formula](https://en.wikipedia.org/wiki/Distance#Straight-line_or_Euclidean_distance.)
* Create a scatter plot of the points, color-coded based on whether they are inside or outside the circle.
* Calculate the proportion of points inside the circle and the approximation of pi by multiplying the proportion by 4.
* Create a custom function get\_pi() that takes the number of points to be simulated as input, generates random x and y coordinates, calculates the proportion of points inside the circle and returns the approximation of pi.
* Run the get\_pi() function with 500 points and observe the result.
* Run the simulation of 500 points 10,000 times (1e4), and store the results in a variable pis.
* Analyze the distribution of pi approximations using summary statistics, histogram, and quantiles.
* Repeat steps 8 and 9 with a higher number of points (10,000) to observe the improvement in the accuracy of the pi approximation.

*# If you don't have the following packages, then install them by uncommenting*  
*# the following two lines.*  
*# install.packages("ggplot2")*  
*# install.packages("ggforce")*  
library(ggplot2)  
library(ggforce)  
  
*# Generate many uniform xs and ys between 0 and 1*  
x <- runif(1e3)  
y <- runif(1e3)  
  
*# Logical indicating whether or not the point "landed" inside the circle*  
inside\_circle <- (x - 0.5)^2 + (y - 0.5)^2 < 0.5^2  
  
*# Graph to show the points and whether or not they landed inside the circle*  
ggplot(data.frame(x = x, y = y, inside\_circle = inside\_circle)) +  
 geom\_point(aes(x = x, y = y, color = inside\_circle)) +  
 geom\_circle(aes(x0 = 0.5, y0 = 0.5, r = 0.5)) +  
 scale\_color\_manual(values = c("firebrick", "forestgreen")) +  
 theme\_bw()

*# Find proportion by dividing number inside circle divided by total number*  
sum(inside\_circle == TRUE) / length(inside\_circle)

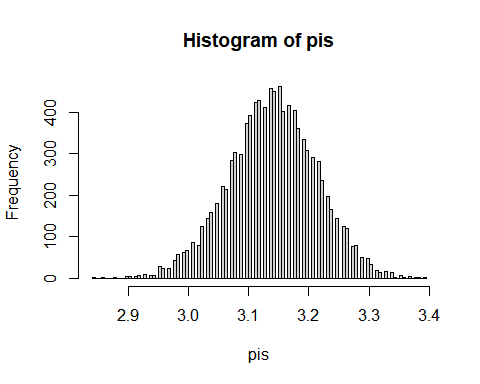
## [1] 0.811

*# Alternative is to take the mean of the logical vectors (TRUE = 1, FALSE = 0)*  
mean(inside\_circle)

## [1] 0.811

*# Approximation of pi is proportion times 4*  
mean(inside\_circle) \* 4

## [1] 3.244

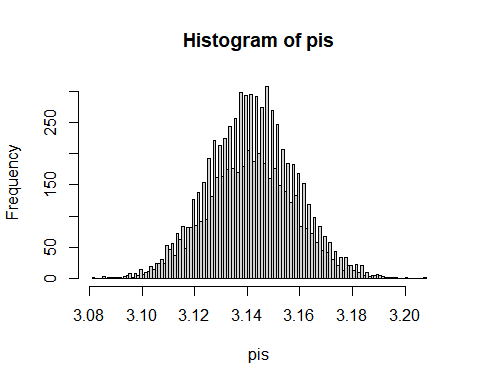
*# Function that runs one simulation of throwing n\_points and calculates*  
*# the approximation for pi by taking 4 times the proportion of darts*  
*# that land inside the circle*  
get\_pi <- **function**(n\_points) {  
 x <- runif(n\_points)  
 y <- runif(n\_points)  
 inside\_circle <- (x - 0.5)^2 + (y - 0.5)^2 < 0.5^2  
 mean(inside\_circle) \* 4  
}  
  
*# See what happens if select 500 points*  
get\_pi(500)

## [1] 3.12

*# Run the simulation of 500 points 1000 times*  
pis <- replicate(1e4, get\_pi(500))  
  
*# Look at the distribution of pi approximations*  
summary(pis)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 2.840 3.096 3.144 3.140 3.192 3.392

hist(pis, 100)

quantile(pis, probs = c(0.025, 0.5, 0.975))

## 2.5% 50% 97.5%   
## 2.992 3.144 3.280

*# Run the simulation with many, many more points*  
pis <- replicate(1e4, get\_pi(1e4))  
summary(pis)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 3.082 3.130 3.142 3.142 3.152 3.207

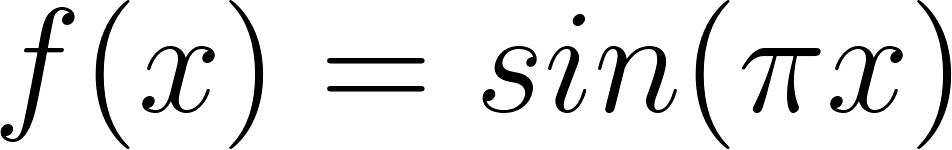
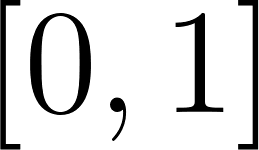
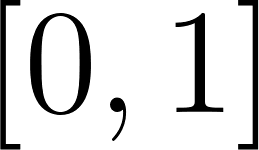
hist(pis, 100)

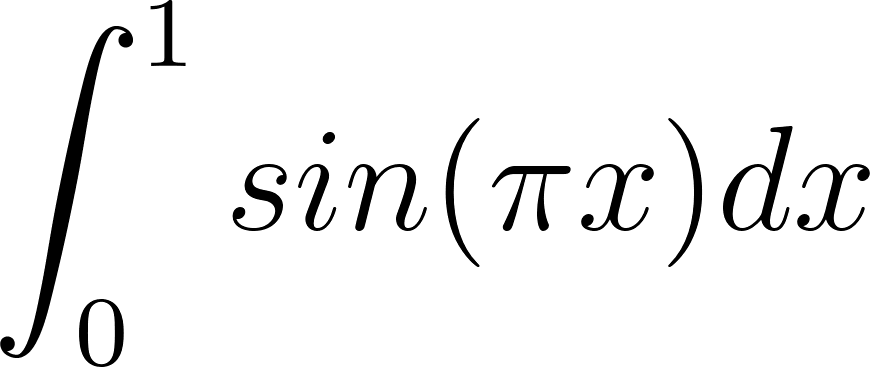
quantile(pis, probs = c(0.025, 0.5, 0.975))

## 2.5% 50% 97.5%   
## 3.1096 3.1416 3.1744

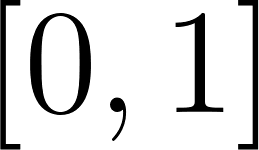
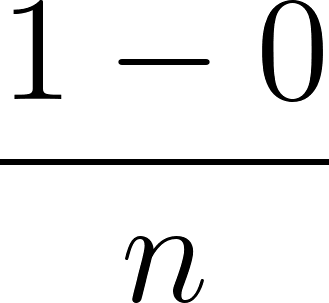
# Fun With Calculus

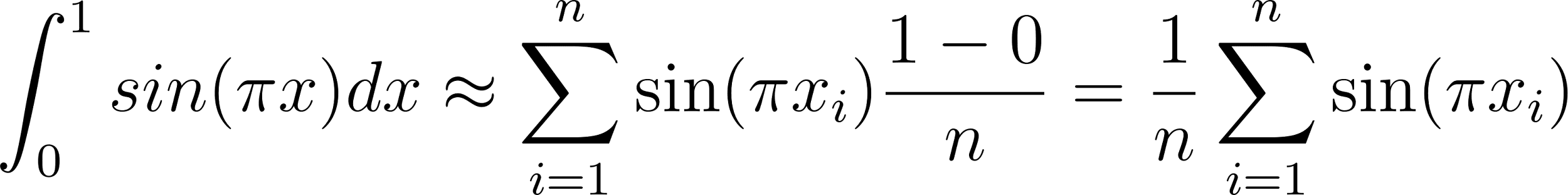
[Monte Carlo integration](https://en.wikipedia.org/wiki/Monte_Carlo_integration) is a numerical integration technique that uses random sampling to approximate the definite integral of a function over a specific interval. It is especially useful for high-dimensional integrals and complex functions where traditional methods like [Riemann sums](https://en.wikipedia.org/wiki/Riemann_sum) or more advanced techniques like [Simpson's rule](https://en.wikipedia.org/wiki/Simpson%27s_rule) or [Gaussian quadrature](https://en.wikipedia.org/wiki/Gaussian_quadrature) become cumbersome.

We want to approximate the integral of the function [](https://www.codecogs.com/eqnedit.php?latex=f(x)%20%3D%20sin(%5Cpi%20x)#0) over the interval [](https://www.codecogs.com/eqnedit.php?latex=%5B0%2C%201%5D#0) using Monte Carlo integration. The definite integral of [](https://www.codecogs.com/eqnedit.php?latex=f(x)#0) over [](https://www.codecogs.com/eqnedit.php?latex=%5B0%2C%201%5D#0) can be represented as:

[](https://www.codecogs.com/eqnedit.php?latex=%5Cint_0%5E1%20sin(%5Cpi%20x)%20dx#0)

Monte Carlo integration can be used to approximate this integral as follows:

* Choose a large number of random points ([](https://www.codecogs.com/eqnedit.php?latex=x_i#0)) uniformly distributed over the interval [](https://www.codecogs.com/eqnedit.php?latex=%5B0%2C%201%5D#0).
* Evaluate the function [](https://www.codecogs.com/eqnedit.php?latex=f(x_i)#0) for each random point [](https://www.codecogs.com/eqnedit.php?latex=x_i#0).
* Consider the width of each rectangle to be [](https://www.codecogs.com/eqnedit.php?latex=%5Cfrac%7B1-0%7D%7Bn%7D#0) where [](https://www.codecogs.com/eqnedit.php?latex=n#0) is the number of random points and the height of each rectangle as [](https://www.codecogs.com/eqnedit.php?latex=f(x_i)#0).
* Calculate the sum of all the heights and widths. This is the approximate value of the integral or area under the curve of [](https://www.codecogs.com/eqnedit.php?latex=f(x)#0)

[](https://www.codecogs.com/eqnedit.php?latex=%5Cint_0%5E1%20sin(%5Cpi%20x)dx%20%5Capprox%20%5Csum_%7Bi%3D1%7D%5En%20%5Csin(%5Cpi%20x_i)%5Cfrac%7B1-0%7D%7Bn%7D%3D%5Cfrac%7B1%7D%7Bn%7D%5Csum_%7Bi%3D1%7D%5En%20%5Csin(%5Cpi%20x_i)#0)

As the number of random points increases, the approximation of the integral converges to the true value.

## Fun With Calculus R Code

This R code demonstrates the Monte Carlo integration method to approximate the integral of the function f(x) = sin(pi \* x) over the interval [0, 1]. The code also visualizes the Monte Carlo integration process using ggplot2. Here's a brief description of each part of the code:

* Load the ggplot2 library for visualization.
* Define the function f(x) = sin(pi \* x).
* Create a function called monte\_carlo\_integration that takes the number of random points (n) as an input and performs the following steps:
  + Generate n random points (x) uniformly distributed over the interval [0, 1].
  + Evaluate the function f(x) for each random point x.
  + Calculate the sum of all the function values (f(x)) and multiply it by the length of the interval (b - a) divided by the total number of random points (n) to get the integral approximation.
  + Return the random points (x), function values (f\_x), and the integral approximation.
* Perform the Monte Carlo integration for 1e7 random points and print the integral approximation.
* Create a ggplot2 visualization that shows the function curve in red, the random points in blue, and vertical dashed lines connecting 50 of the random points to the x-axis.

The code also prints the exact answer using R's built-in integrate function, allowing you to compare the Monte Carlo approximation with the true integral value.

*# Load the ggplot2 library for visualization*  
library(ggplot2)  
  
set.seed(1)  
*# Define the function f(x) = sin(pi \* x)*  
f <- **function**(x) {  
 sin(pi \* x)  
}  
  
*# Monte Carlo integration*  
monte\_carlo\_integration <- **function**(n) {  
 *# Generate n random points uniformly distributed over [0, 1]*  
 x <- runif(n)  
 *# Evaluate the function f(x) for each random point x*  
 f\_x <- f(x)  
 *# Take the value of the function times the width of each rectangle and sum*  
 integral\_approximation <- (1/n) \* sum(f(x))  
 return(list(x = x, f\_x = f\_x,   
 integral\_approximation = integral\_approximation))  
}  
  
*# Number of random points*  
n <- 1e7  
  
*# Approximate the integral using Monte Carlo integration*  
result <- monte\_carlo\_integration(n)  
integral\_approximation <- result$integral\_approximation  
  
print(paste("Integral approximation using", n, "random points:", integral\_approximation))

## [1] "Integral approximation using 1e+07 random points: 0.636563788515587"

print(paste("Exact answer", integrate(f, 0, 1)$value))

## [1] "Exact answer 0.636619772367581"

*# Select a random sample of 50 points for visualization*  
sample\_size <- 50  
sample\_indices <- sample(1:n, sample\_size)  
  
*# Create a data frame for ggplot with the random sample*  
data <- data.frame(x = result$x[sample\_indices], f\_x = result$f\_x[sample\_indices])  
  
*# Create the visualization*  
ggplot(data, aes(x = x, y = f\_x)) +  
 geom\_point(color = "blue", alpha = 0.5) +  
 geom\_segment(aes(x = x, y = 0, xend = x, yend = f\_x), linetype = "dashed", alpha = 0.5) +  
 stat\_function(fun = f, geom = "line", color = "red") +  
 labs(title = "Monte Carlo Integration",  
 subtitle = paste("Integral Approximation:", round(integral\_approximation,4)),  
 x = "x", y = "f(x)") + theme\_minimal()

