PROCEDIMIENTO

Antes que nada, consideramos generar las 3 monedas con las que cuenta Harvey. Si bien no necesitamos todas para el ejercicio (a) nos pareció más organizado.

```
N <- 1000
### MONEDAS
x <- runif(N)
y <- runif(N)
moneda1 <- 1*(x<0.5)
moneda2<- 1*(y<0.5)
moneda3 <- rep(1,N)
```

Vemos así que quedan generadas dos monedas cuyas tiradas tienen igual probabilidad de cara como de ceca (moneda1 y moneda2), y una otra en la que siempre sale cara (moneda3). Para esto fue necesario asignar un valor N arbitrario que representa la cantidad de tiradas, en este caso 1000. Estas monedas son listas generadas aleatoriamente de N posiciones, donde se asigna un 1 o un 0, cara o ceca respectivamente, excepto la moneda3, en donde forzamos que sea una lista de 1 (no hay azar).

Ahora sí, procediendo al ejercicio (a), A partir de la función "runif(N)" generamos N cantidad de tiros de manera aleatoria. Nos ayuda a decidir a partir de un valor aleatorio que moneda estamos tirando. Creamos luego una lista de N posiciones en la que si la moneda indicada en la i-ésima posición es cara (o 1) almacenaremos un 1, si es ceca (o 0) almacenaremos un 0.

Finalmente sumaremos todos estos valores y dividiremos por la cantidad de tiradas. Nos quedó la simulación.

Pruebo 3 veces, y los valores que nos retorna son: `[1] 0.669, [2] 0.654, [3] 0.677.` Concluimos con que la simulación fue realizada correctamente ya que buscábamos un output bien cercano a 2/3 que es la probabilidad real.

El ejercicio (b) es bastante más rebuscado, y precisamos un contador **c**, una nueva lista de N elementos aleatorios **z** y una lista de N elementos llamada **resultados** inicializada con todos 0 a la que iremos llenando. La idea de lo que está dentro del 'for' fue en principio dividir en tercios la probabilidad de estar tirando con cada una de las monedas, después discriminamos para tomar en cuenta SOLO si la tirada vale 1 y caso contrario a **c** restarle 1, guardando así la cantidad total de caras.

```
c <- N
z <- runif(N)
resultados <- rep(0,N)
for (i in 1:N){
  if(z[i]<=1/3){
    monedaRandom <- "moneda1"
  }else if(z[i]<=2/3){
    monedaRandom <- "moneda2"
  }else monedaRandom <- "moneda3"
    if ((monedaRandom == "moneda1") & (moneda1[i]==1)){
      resultados[i] <- 1
    }else if ((monedaRandom == "moneda2") & (moneda2[i]==1)){
      resultados[i] <- 1
    }else if (monedaRandom == "moneda3"){
      resultados[i] <- 0
    }else{
      c <- c-1
```

```
    }
}
prob_moneda1y2 = sum(resultados)/c
```

En otras palabras:

1) Estamos eligiendo al azar la moneda
2) Si es la 1 o la 2 nos fijamos si es cara, y guardaremos un 1 en la i-esima posición de la lista resultados.
3) Si es la moneda 3 dejaremos el 0
4) Si ningún punto anterior se cumple, es porque es ceca y no nos sirve, por lo que al contador le restaremos 1. Algo así como que "anularemos" esa tirada del total N que teníamos.
5) Por último, y lo más fácil. Sumamos los casos de éxito (cara) y dividimos solo por los que nos sirven, pues la lista resultados quedará con muchos "ceros" de tiros que no sirvieron por no haber cumplido la premisa de haber salido cara en el primer tiro. Es decir, la lista 'resultados' aún contiene N elementos, pero nos sirven c

y el output es: `[1] 0.5089021, [2] 0.4894578, [3] 0.502907`, dando aproximadamente 0.5 que fue el resultado que nos dio realizándolo a mano.

NOTA: En lugar de volver a tirar cada moneda N cantidad de veces dentro del 'for', por un tema de optimización de programación, aprovechamos que los valores de las monedas son aleatorios, y usamos los valores de estas listas como si fueran las primeras tiradas. Es lo mismo, pero nos permitirá usar N más grandes sin prender fuego el procesador!

Por último, probemos lo resultados para varios valores de N.

Para N=100:

```
prob_Cara ---> [1] 0.37, [2] 0.47, [3] 0.59
prob_moneda1y2 ---> [1] 0.45, [2] 0.58, [3] 0.5
```

Para N=1000:

```
prob_Cara ---> [1] 0.52, [2] 0.48, [3] 0.51
prob_moneda1y2 ---> [1] 0.49, [2] 0.48, [3] 0.52
```

Para N=100000:

```
prob_Cara ---> [1] 0.5, [2] 0.5, [3] 0.5
prob_moneda1y2 ---> [1] 0.5, [2] 0.5, [3] 0.5
```

Tal como se suponía, al agrandar N la simulación tenderá al valor exacto de la probabilidad, mientras vemos que para N chicos es poco confiable por ser una muestra más pequeña