**In Data 8, we have three methods of generating random values.**

1. np.random.choice(array, sample\_size=1)  
     
    Returns an array of length sample\_size consisting of items randomly sampled with replacement from the array. If you don't specify the sample\_size argument, it returns just one randomly chosen item.
2. tbl.sample(n, with\_replacement=False)  
     
   This function takes a sample size n (an integer), and an optional argument with\_replacement, which is a boolean. This function returns a new table where n rows are randomly sampled from the original table; by default, n=tbl.num\_rows. Default is with\_replacement=True. For sampling without replacement, use argument with\_replacement=False.
3. sample\_proportions(sample\_size, model\_proportions)  
     
   sample\_size should be an integer, model\_proportions an array of probabilities that sum up to 1. The function samples sample\_size objects from the distribution specified by model\_proportions. It returns an array with the same size as model\_proportions. Each item in the array corresponds to the proportion of times that item was sampled out of the sample\_size times. Both the input model\_proportions and output sum to 1.

**Each of these functions can be used interchangeably to sample with replacement. To sample without replacement, we use tbl.sample(n, with\_replacement=False).**

**Let's see how we can use these functions interchangeably.**

To simulate 1000 coin flips of a fair coin, we could write:

1. np.random.choice(make\_array("Heads", "Tails"), 1000)  
   This will return an array of 1000 string values that are either "Heads" or "Tails".
2. Table().with\_column("Outcome", make\_array("Heads", "Tails")).sample(1000)  
   This will return a table with 1000 rows, and a column "Outcome". Each row will have either the string "Heads" or the string "Tails" in it.
3. sample\_proportions(1000, make\_array(0.5, 0.5))  
   This will return an array with two elements in it, the first item will be the simulated proportion of heads in 1000 tosses, the second item will be the simulated proportion of tails in those 1000 tosses.

To count the number of times the face with 3 dots appears in 500 rolls of a fair dice we could write any of the following:

1. rolls = np.random.choice(np.arange(1, 7), 500)  
   num\_3 = np.count\_nonzero(rolls == 3)
2. dice = Table().with\_column("Face", np.arange(1, 7))  
   rolls = dice.sample(500)  
   num\_3 = rolls.where("Face", are.equal\_to(3)).num\_rows
3. model\_proportions = make\_array(1/6, 1/6, 1/6, 1/6, 1/6, 1/6)  
   rolls = sample\_proportions(500, model\_proportions)  
   num\_3 = rolls.item(2) \* 500

**When should we use a specific function?**

Because each of these functions are interchangeable there are no set rules for which function to use. That being said, we can save ourselves some time by following these tips:

1. If we have some model\_proportions specified in our problem, sample\_proportions will probably be the simplest function to use!
2. If we are dealing with resampling values or our data is stored in a table already then tbl.sample will probably be the most convenient function to use.
3. If we are choosing from an array of values, np.random.choice is often the easiest function to implement, or we can store the values in a table and then use tbl.sample.
4. When shuffling the labels for A/B testing we should always use tbl.sample(with\_replacement=False) because this is the only way we can sample without replacement.

These are only general tips--if you want more specific examples of when you might use each function, check out the textbook! For example, [section 10.2](https://www.inferentialthinking.com/chapters/10/2/Sampling_from_a_Population.html) has an example of using tbl.sample to sample from a table of flights, and [section 11.2](https://www.inferentialthinking.com/chapters/11/2/Multiple_Categories.html) has an illustrative example on when to use sample\_proportions.