Notes for Stat 430 – Design of Experiments – Lab #2 Sept. 17, 2012.

Agenda:
sample() function
print() function
for() loop
Randomization test – Single Run
Randomization test – Monte Carlo

Useful files: http://www.sfu.ca/~jackd/Stat430/Wk02_Code.txt

Sample Function

Sample() lets you select a random element or number of elements from a list. With it you can…

Pick a number from 1 to 10

```
sample(1:10, size=1)
```

Pick four numbers from 1 to 10

```
sample(1:10, size=4)
```

Pick four numbers from 1 to 10, you can use the same number twice

```
sample(1:10, size=4, replace=TRUE)
```

You can have multiples of the same number. In this case you'll end up with multiples in your sample even though replacement is FALSE.

```
temp = c(1,1,1,1,1,2,2,2,2,2)
sample(temp, size=4)
```

Print Function

The print function makes R print to the screen whatever is inside the ( )

```
print("Hello World!")
```

We can also assign variables to print

```
course = "Stat 430"
print(course)
```

```
today <- "Sept 16 2012"
print(today)
```

Note that = and <- do the same thing

We could also make a list of numbers or words (strings) and print it

```
list1 = 1:5
list2 <- c("A","B","C","D")   # c( , , , ) is used to make a list from scratch
list3 = c(4,-1,10^12,6*8)
print(list1)
print(list2)
print(list3)
```

For Loops

The print function can be used in other structures like the for loop.

```
for(k in 1:5)
{
      ## things in here will happen 5 times
}
```

Everything inside the { } will happen for each number in the list after the word "in".
The list 1:5 has 5 numbers, so everything happens five times.
Every time we go through it is called an iteration.

```
for(k in 1:5)
{
     print("That's one iteration")
}
```

K is the name of the counting variable. It's just a name, so you can make it anything you want.

```
for(badger in 1:5)
{
     print("Printing this 5 times")
```

```
}
```

The counting variable can itself be used in a for loop.

```
for(count in 1:5)
{
      print(count * 20)
}
```

Randomization Test - Single Run

For loops are very good for when we want to do the same thing many times. The randomization test (a.k.a. permutation test) is the perfect example of this.

About the randomization test:

- H_0: The population means are the same. (Same as t-test H_0)
- P-value The probability of getting a sample with as much evidence against H_0 (or more evidence) as you observed if the null is actually true.
- It is non-parametric. (i.e. It doesn't assume the data is a specific distribution like normal.)

In practice, H_0 means that all the sampled points came from a common pool of values. What you observe is one of many permutations of that those samples could have been arranged into two groups.

The p-value is the proportion of permutations that show as much of a difference in the means of the two samples as your observation.
If the difference is large, then not many permutations will be able to produce a larger difference and your p-value will be small.

We have values from a sample of size 8 and another of size 12.
How many combinations do we need to look at?

There are $\frac{20!}{8! \times 12!}$ combinations. We could compute this in R with:

```
factorial(20) / (factorial(12) * factorial(8) )
```

or...

```
choose(20,8)
```

or...

```
choose(20,12)
```

Checking 125,970 combinations, even by computer, is hard, but it gets worse quickly as the samples get larger. (If we had samples of size 14 and 17, there would be `choose(31,14)` , more than 265 million combinations.)

Instead of checking all 125,970 in the example above, we can check a randomly selected group (SRS) of the combinations and use that as an approximation because it will converge to the correct answer.

First, define the two samples, and a common pool of their values.

```
group1 = c(17, 15, 20, 27, 22, 20, 20, 18)
group2 = c(18, 15, 9, 18, 13, 17 ,7, 11, 12, 15, 26, 24)
pool = c(group1, group2)
```
This makes a list of both groups together, a pool of 20 values.

Next, we randomly select one of the 125,970 combinations by taking a sample of size 8 from the numbers 1 to 20 (without replacement). This decides which 8 of the 20 values from the pool will be in our random group of 8. The remaining 12 values go into the group of 12.

```
samp = sample(1:20,size=8)
rndm_grp1 = pool[samp]
rndm_grp2 = pool[-samp]
```

Let the alternative hypothesis was that group1's mean is higher than group2's mean.
If H0 is to be rejected in favour of this alternative, then the difference between group1's mean and group2's mean should be greater than the difference between most random groups.

```
group_diff = mean(group1) - mean(group2)
random_diff = mean(rndm_grp1) - mean(rndm_grp2)
```

Check if the group difference is larger (R will tell you if a statement is true or false)

```
group_diff > random_diff
```

Randomization Test – Monte Carlo

This is pretty advanced, feel free to just copy-paste this for now and change bits of the code to understand how it works.

```
# Set up the group difference and a counter before we start
random_was_bigger = 0
group_diff = mean(group1) - mean(group2)

# For each of 10000 iterations...
for(k in 1:10000)
{
	### Make the combination and assign the values to two groups
	samp = sample(1:20,size=8)
	rndm_g1 = pool[samp]
	rndm_grp2 = pool[-samp]

	### Find the differences in the means of the random groups
	random_diff = mean(rndm_grp1) - mean(rndm_grp2)

	### If the mean difference in the random groups was the greater difference...
	if(random_diff > group_diff)
	{
		### Count 1 more time that the random group was bigger
		random_was_bigger = random_was_bigger + 1
	}
}

### The p-value is the proportion of times the random group had a bigger mean difference
random_was_bigger / 10000  # divided by 10000 because there were 10000 iterations.
```

This method of simulating something many times to estimate a value is called a **Monte Carlo** method. We didn't use a formula to find the p-value, we directly estimate the probability of seeing the effect size under H_0 by simulating how often it came up.