First.

Investigate how many times this function would be called VS the getYear function in the display-Advanced-Table

ChopYearList(): void{}

Component: Display-Advanced

Conditions:

Ascending v Descending

yearRange < or > depending on above

Cons:

Resource Intensive – constantly calculating

pros:

No real alternative, because the yearRange is going to change we will always add or subtract years.

The ascending or descending will be handled separately however, the quarterly vs annual will be dynamically changing the list as well.

Because we will be determining the period well before any function in advanced-display-table is actually called we do not have to worry about appending the Q4 etc to the array.

**solution:**

* copy displayYear to Array
* Clear displayYear
* Push values into displayYear until we hit desired value

// If Precise is desired we will have to display years and quarters

If(PRECISE) {

//Ascending

ForEach()

if ( copiedArray[i].subtring(year) <= rangeYear.value && !direction ){

//This will work regardless of quarter or annual

newArray.push(copiedArray[i])

}

If the substring year is less than or equal to the yearRange.value and we are ascending than push that value until we exceed rangeYear.value.

}

// If we want uniformity, ie oldest to newest in the case of ascending then we will need a new algorithm.

Else {

Let difference = copiedArray.substring(year).indexOf(yearRange.value))

// with values ranging from 1 – 20 in an array it would be represented as 0 – 19

//We want 10 so the index would return 9. Thererfore,we shall copy 9 instances

For(let K < difference)

{ displayList.push(

Problem:

Because we destroy the past array of displayList, once we switch to uniform we will lose our q4 q3 q2 quarter displays, and not be able to get them back.

}