## Programming

- Why is programming fun?
  - First is the sheer joy of making things. As the child delights in his mud pie, so the adult enjoys building things, especially things of his own design. I think this delight must be an image of God's delight in making things, a delight shown in the distinctness and newness of each leaf and each snowflake.

Source: Frederick P. Brooks, Jr. *The Mythical Man-Month: Essays on Software Engineering.*

## Homework

- By today you should have read
  - Chapters 20 and 21 in *Fluency*

## Once is Not Enough

*Iteration Principles*

## Iteration: *Play It Again, Sam*

- The process of repetition:
  - looping through a series of statements to repeat them

*Again and again, and again*

*Repetition is good*

## FOR LOOPS

## The for Loop Basic Syntax

```
for (<initialization>; <continuation>; <next
iteration>) {
        <statement list>
}
```

- Text that is not in *metabrackets* <> must be given literally
- The whole sequence of statements in the statement list is performed for each iteration
    - Computer completes the whole statement sequence of the <statement list> before beginning the next iteration

## The Iteration Variable

- *Control specification*: the three operations in the parentheses of the for loop
  - Control the number of times the loop iterates
  - by using an *iteration variable* (must be declared)

## JavaScript Rules for for Loops (cont'd)

- The World-Famous Iteration
  - JavaScript uses the same `for` loop statement as other programming languages, so thousands of loops with this structure are written every day:

    ```
    for ( j = 0; j < n; j++ ) {…}
    ```

  - Most frequently written `for` loop of all time
  - Easy to see iteration count:
    - Always *n* times

## The Iteration Variable (cont'd)

- Example:

```
for ( j = 1 ; j <= 3 ; j = j + 1) {
   <statement list>
}
```

- Here's what happens:
  - The first operation is the <*initialization*>
    - Sets the iteration variable's value for the first iteration of the loop. Done only once.
  - The next operation is <*continuation*>
    - Test. If the test has a false outcome, the <statement list> is skipped.
    - If the test has a true outcome, the <statement list> is performed. When the statements are complete, the
  - <*next iteration*> operation is performed
    - Repeats with the continuation test, performs same sequence of steps.

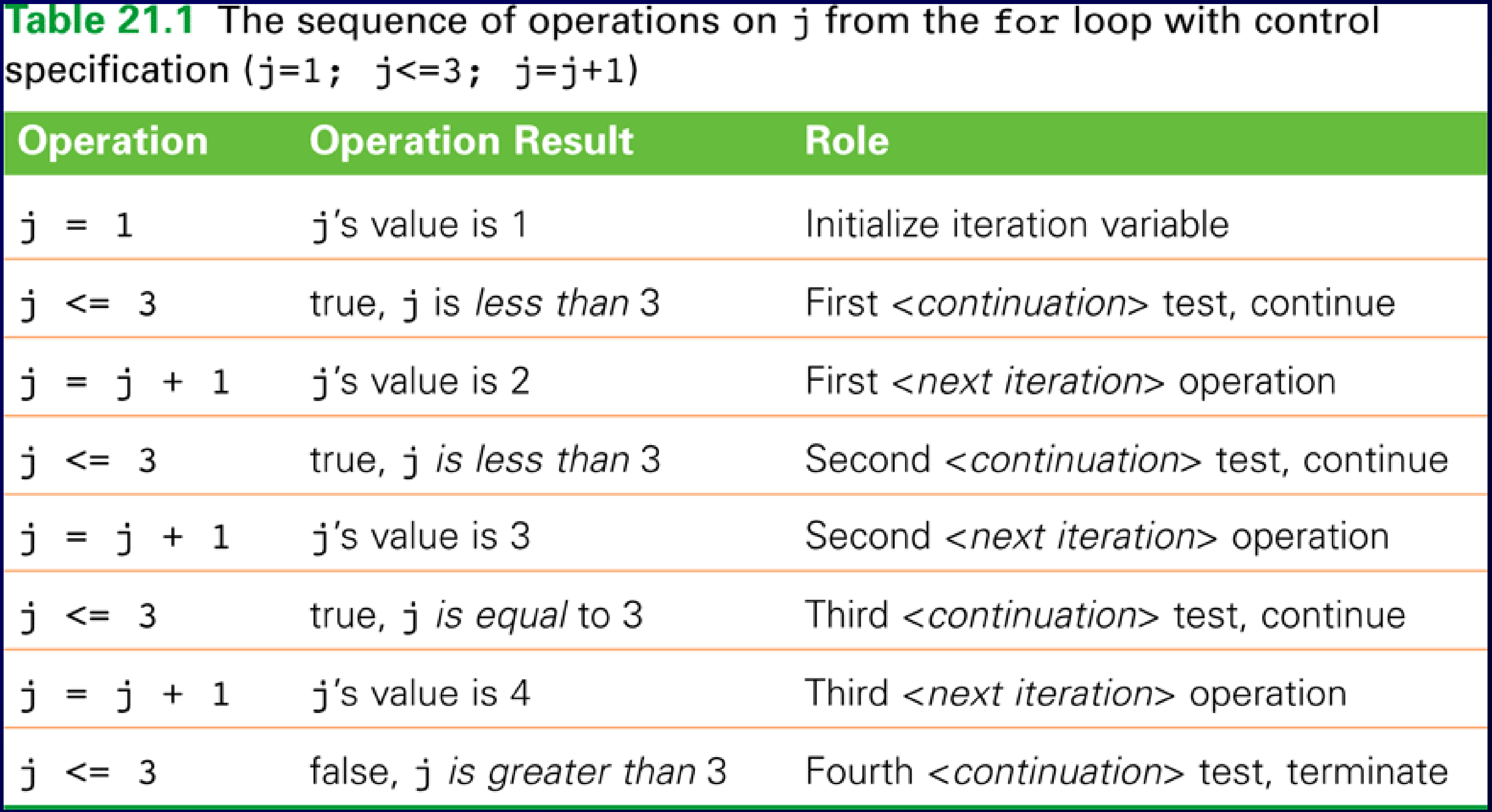

Table 21.1 The sequence of operations on `j` from the `for` loop with control specification `(j=1; j<=3; j=j+1)`

| Operation | Operation Result | Role |
|---|---|---|
| `j = 1` | `j`'s value is 1 | Initialize iteration variable |
| `j <= 3` | true, `j` is *less than* 3 | First <*continuation*> test, continue |
| `j = j + 1` | `j`'s value is 2 | First <*next iteration*> operation |
| `j <= 3` | true, `j` *is less than* 3 | Second <*continuation*> test, continue |
| `j = j + 1` | `j`'s value is 3 | Second <*next iteration*> operation |
| `j <= 3` | true, `j` *is equal* to 3 | Third <*continuation*> test, continue |
| `j = j + 1` | `j`'s value is 4 | Third <*next iteration*> operation |
| `j <= 3` | false, `j` *is greater than* 3 | Fourth <*continuation*> test, terminate |

## How a for Loop Works

- Consider a computation on declared variables j and text

```
text = "She said ";
for ( j = 1; j <= 3; j = j + 1 ) {
    text = text + "Never! ";
}
alert(text);
```

## How a for Loop Works

- Consider a computation on declared variables j and text

```
text = "She said ";
for ( j = 1; j <= 3; j = j + 1 ) {
    text = text + "Never! ";
}
alert(text);
```

**Starting point**

## How a for Loop Works

- Consider a computation on declared variables j and text

```
text = "She said ";
for ( j = 1; j <= 3; j = j + 1 ) {
    text = text + "Never! ";
}
alert(text);
```

**Stop condition**

# How a for Loop Works

- Consider a computation on declared variables j and text

```
text = "She said ";
for ( j = 1; j <= 3; j = j + 1 ) {
    text = text + "Never! ";
}
alert(text);
```

**Step size or increment**

## How a for Loop Works

- Demo:

```
text = "She said ";
for ( j = 1; j <= 3; j = j + 1 ) {
    text = text + "Never! ";
}
alert(text);
```

## JavaScript Rules for for Loops

- The Iteration Variable: **j = 1;**
  - Must be declared, and follow rules for variable identifiers
  - i, j, and k are the most common choices
- The Starting Point
  - Iteration can begin anywhere, including negative numbers

## JavaScript Rules for for Loops (cont'd)

- Continuation/Termination Test **`j <= 3`**
  - *Test* is any expression resulting in a Boolean value (true/false)
  - Continuation must involve iteration variable to avoid infinite loop
- Step Size **`j = j + 1`**
  - Amount of change from one iteration to the next
  - Often called the *increment* or *decrement*

## Experiments with Flipping Coins

- To practice for loops, we experiment with flipping electronic coins
- We can use the function `randNum(2)`, which returns either 0 (tails) or 1 (heads)
- Set up an iteration in which our randNum() function is performed 100 times, and statistics gathered

## Experiments with Flipping Coins (cont'd)

```
<html><head><title>Coin Flips</title></head>
<body><script language='JavaScript'>
var heads=0, tails=0;                      //Counters
var i;                                     //Iteration variable
for (i=0; i<100; i++ ){
   if (randNum(2) == 1)
      heads++;
   else
      tails++;
}
alert("Heads: " + heads + " and Tails: " + tails);
function randNum(range) {
   return Math.floor(range*Math.random());
}
</script></body></html>
```

- Demo...

## Experiments with Flipping Coins (cont'd)

- `i` ranges from 0 to 99, so the loop iterates 100 times
- Conditional statement checks and records the outcome of random number generation
- When random number is 1, count of heads is increased by 1 ( `heads++;` )
- When random number is 0, count of tails is increased by 1 ( `tails++;` )

## Experiments with Flipping Coins (cont'd)

- A Nested Loop
  - To run several trials, consider the entire loop we just looked at as one Trial
  - Create another for loop containing this Trial unit, adding a couple of needed statements
  - We have a loop within a loop (*nested loop*) which causes the Trial loop (0-99) to run five times

## Experiments with Flipping Coins (cont'd)

```
var heads = 0, tails = 0;
var i, j;                                       //Iteration vars
for (j = 0; j < 5; j++){                        //Outer loop start
   for (i=0; i<100; i++){                          //Trial line 1
      if (randNum(2) == 1)                         //Trial line 2
         heads++;                                  //Trial line 3
      else                                         //Trial line 4
         tails++;                                  //Trial line 5
   }                                               //Trial line 6
   alert("Heads: "+heads+" and Tails: "+tails);
                                                   //Trial line 7
   heads = 0; tails = 0;                        //Additional
}                                               //Outer loop end
```

- Demo....

## Experiments with Flipping Coins (cont'd)

- A Diagram of Results
  - To show how far off a perfect 50-50 score a trial is, display with diagram
  - Compute the distance from 50-50 and show that number using asterisks

```
text = text + 'Trial ' + j + ': ';
for (i = 0; i < (Math.abs(heads-50)); i++) {
    text = text + '*';
}
text = text + '\n';
alert(text);
```

*Creating and using lists, or arrays*

## INDEXING

## Indexing

- Process of creating a sequence of names by associating a base name with a number (like Apollo 13 or Henry VIII)
  - Each indexed item is called an element of the base-named sequence
- Index Syntax
  - index number is enclosed in square brackets [ ]
- Iterations can be used to refer to all elements of a name
  - A[j] for successive iterations over j referring to different elements of A

## Indexing (cont'd)

- *Index* ***Origin***
  - The point at which indexing begins (the least index)
  - In life, the first element may begin with 1, or have no number (Queen Elizabeth)
  - JavaScript *always* uses index origin 0

## Rules for Arrays

- Arrays are normal variables initialized by
```
    new Array (<number of elements>);
```
- <number of elements> is number of items in array
- Array indexing begins at 0
- Greatest index is <number of elements> - 1
- Number of elements is array length
- Index values range from 0 to (length – 1)

## Array Reference Syntax

- Array reference is array name together with index enclosed in brackets (non-negative integer or expression or variable that resolves to non-negative integer)

```
array[i]
```

- World-Famous Iteration, or 0-origin loop iteration, is perfect for arrays

## JavaScript Rules for for Loops (cont'd)

- The World-Famous Iteration for looping through an array:

```
for ( i = 0; i < fruits.length; i++ )
{
      alert(fruits[i]);
}
```

- `.length` is a built-in JavaScript property that always gives you the length of an array.

## Reflections

- Write for 10 minutes on this topic:
  - First describe and then compare and contrast
    - Dante and
    - The Students server
  - Be sure to answer these questions:
    - How are they connected?
    - How do you access each one?

## Homework

- Read *Fluency* chapter 22 for Friday!
- Quiz 4 Thursday and Friday
  - See email for details on what to review