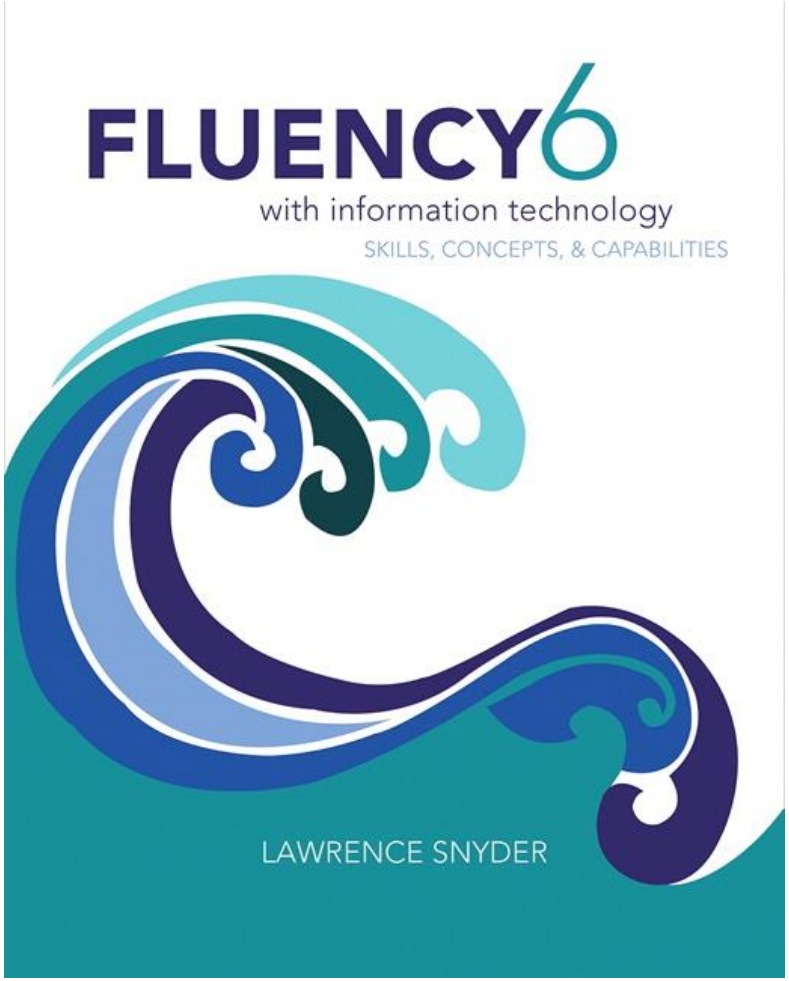

Chapter 20

Iteration Principles

Table of Contents

- Part 1: Becoming Skilled at Computing
- Part 2: Algorithms and Digitizing Information
- Part 3: Data and Information
- Part 4: Problem Solving
  - Chapter 17: Fundamental Concepts Expressed in JavaScript
  - Chapter 18: A JavaScript Program
  - Chapter 19: Programming Functions
  - Chapter 20: Iteration Principles
  - Chapter 21: A Case Study in Algorithmic Problem Solving
  - Chapter 22: Limits to Computation
  - Chapter 23: A Fluency Summary

Learning Objectives

- Trace the execution of a given for loop
- Learn a World-Famous Iteration for loop
- Discuss the structure of nested loops
- Explain the use of indexes
- List the rules for arrays; describe the syntax of an array reference
- Explain the main programming tasks for online animations

Terminology

- Repeat
  - 5 repeats means that you may have done it once followed by 4 more times
  - The first time is not considered a “repeat”
  - The second through the last are “repeats”
- Iterate
  - 5 iterations means that you do it 5 times
- Iteration means looping through a series of statements to repeat them
- In JavaScript, the main iteration statement is the for loop

for Loop Syntax [1/2]

```
for (var j = 0; j < 3; j=j+1) {
        <statement list>
}
```

General Form	Specific Example with j
<initialization>;	j = 1;
if (*<continuation>*) {	if (j <= 3) {
<statement list>;	*<statement list>*;
<next iteration>;	j = j + 1;
}	}

- Text that is not in <meta-brackets> must be given literally
- Computer completes the whole statement sequence of the <statement list> before beginning the next iteration
- 3 operations in the parentheses of the for loop control the number of times the loop iterates (called the control specification)

- This example uses j as the iteration variable
- Iteration variables are normal variables and must be declared

for Loop Syntax [2/2]

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

```
for (var j = 0; j < 3; j=j+1) {
        <statement list>
}
```

- <initialization> sets the iteration variable's value for the first iteration of the loop
- <continuation> has the same form as the predicate in a conditional statement
  - If the <continuation> test is false, the loop terminates and <statement list> is skipped
  - If <continuation> test is true, the < statement list> is performed
- When the statements are completed, the <next iteration> operation is performed
- <next iteration> changes iteration variable
  - Next iteration starts with the <continuation> test, performing the same sequence of operations
- Iterations proceed until the <continuation> test becomes false, terminating the loop

for Sequence

Table 20.1 The sequence of operations on j from the **for** loop with control specification (j=0; j<3; j=j+1)

Operation	Operation Result	Role
j = 0	j's value is 0	Initialize iteration variable
j < 3	true, j is *less than* 3	First <*continuation*> test, do statements, continue
j = j + 1	j's value is 1	First <*next iteration*> operation
j < 3	true, j *is less than* 3	Second <*continuation*> test, do statements, continue
j = j + 1	j's value is 2	Second <*next iteration*> operation
j < 3	true, j *is less than* 3	Third <*continuation*< test, do statements, continue
j = j + 1	j's value is 3	Third <*next iteration*> operation
j < 3	false, j *is equal to* 3	Fourth <*continuation*> test, terminate

```
text = "She said ";                 //Set text to a string
for (j = 1; j <= 3; j = j + 1) {    //Define a 3 cycle loop
   text = text + "Never! ";         //Concatenate on a string
}                                   //. . .  end of loop
alert(text);                        //Show result
```

which produces the following alert box.

WFI!

- JavaScript uses the same for loop statement as other popular PLs like Java, C++
- The following syntax is the most frequently written for loop all times

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

- We call it World-Famous Iteration (WFI)

Iteration Variables

- Iteration variables are normal variables, but just used in iteration
- Programmers tend to choose short identifiers for iteration
  - i, j, and k are the most common

Starting Point

- Iterations can begin anywhere
  - Including with negative numbers: for (j = −10; j <= 10; j = j + 1) { . . . }
  - Including fractional numbers: for (j = 2.5; j <= 6; j = j + 1) { . . . }
    - j assumes the values 2.5, 3.5, 4.5, and 5.5

Continuation/Termination Test

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

- If you can begin an iteration anywhere, you can end it anywhere
- The <continuation> test is any predicate expression having the iteration variable and resulting in a Boolean value: ex. j > 3

Step-by-Step

- <next iteration> allows you to specify how big or small the change in the iteration variable (the step or step size) ➔ j=j+1 j=j+10

Iteration Variable does Math!

- Iteration variable is often used in computations in the <statement list>
- Important that you focus on the values of the iteration variable during the loops

```
fact = 1;
 for ( j = 1; j <= 5; j = j + 1) {
      fact = fact * j;
}
```

Infinite Loops and Infinitum

- for loops are relatively error free, but still possible to create infinite loops
- Every loop in a program must have a continuation test or it never terminates!
- The 5th property of algorithms is that they must be (1) finite or (2) stop & report that no answer is possible

```
for ( j = 1 ; j <= 3; i = i + 1) {     // infinite loop example
            . . .
 }
```

- If the continuation test is based on values that don't change in the loop, the outcome of the test will never change
- The loop, then, will never end (note i and j above)

for Loop Practice: Heads/Tails

- Let's use randNum(2) from Chapter 19: It will return 0 (tails) or 1 (heads)
- And flip the "coin" 100 times
- Use WFI

```
function randNum(range) {
     return Math.floor(range*Math.random());
}
function trial (count) {
  var heads=0, tails=0;
  for (var i=0; i<count; i++ ) {
      if (randNum(2) == 1)
         heads++;
      else
         tails++;
  }
  return heads ;
}

trial(100)

/*
52
*/
```

Figure 20.1 The trial( ) declaration, and the results of a 100-flip trial.

```
function randNum(range) {
     return Math.floor(range*Math.random());
}
function trial (count) {
  var heads=0, tails=0;
  for (var i=0; i<count; i++ ) {
      if (randNum(2) == 1)
         heads++;
      else
         tails++;
  }
  return heads ;
}
var headCount, outAns = "";          //Output text is empty
for (var j=0; j < 5; j++) {
    headCount = trial(100);          //Compute a trial
    outAns = outAns + "Trial " + j   //Build answer string
       + ": " + headCount + ":" +
       (100-headCount) + '\n';
}

outAns
/*
Trial 0: 45:55
Trial 1: 43:57
Trial 2: 63:37
Trial 3: 52:48
Trial 4: 48:52

*/
```

Figure 20.2 The JavaScript program to run five trials of 100 flips each.

Math.floor() & Math.random() :

Built-in Math objec의 method들

Nested Loops…Loop in a Loop

- All programming languages allow loops to nest
- Inner and outer loops must use different iteration variables or else they will interfere with each other

```
var headCount, outAns = "", aster;                    //Output text is empty
for (var j=0; j < 5; j++) {

    headCount = trial(100);                           //Compute a trial
    outAns = outAns + "Trial " + (j+1) + ': ';        //Build answer string
    aster = "";                                       //Initialize
    for (var k=0; k < Math.abs(headCount-50); k++) {  //Loop by difference
        aster = aster + "*";                          //Add * for each one
    }
    outAns = outAns + aster + "\n";                   //Include in output
}

outAns

/*
Trial 1: **
Trial 2: ****
Trial 3: ***
Trial 4: **
Trial 5: *

*/
```

Number of Heads is 52

Figure 20.3 JavaScript for a program using three iterations (one not shown) to display the results for five trials using asterisk diagram.

```
<script>
  function randNum( range) {
    return Math.floor( range * Math.random( ));
  }
  function coinFlip ( ) {
    if (randNum(2) == 1)
      return "us1heads.jpg";
    else
      return "us1tails.jpg";
  }
  document.write("<div style='margin:50px'>")
  for (var j=0; j<5; j++) {
    for (var i=0; i<7; i++) {
    document.write('<img src=" ' + coinFlip( )
          + ' " width="50" />');
    }
    document.write('<br/>');
  }
  document.write('</div>');
</script>
```

Figure 20.4 Nested loops, with the inner loop iterating seven times and the outer loop iterating five times.

- documents.write() 의 괄호안에 문장이 있으면 문장을 쓰고 HTML Tag가 있으면 HTML을 수행한다
- …… : 화면에 block을 잡아주는 division tag

Indexing of Sequenced Data

- We are acquainted with indexed data
  - Class 5, Rocky 3, Apollo 13, World War 2
- Indexing is the process of creating a sequence of names by associating a base name with a number
- Each indexed item is called an element of the base-named sequence
- An index is enclosed in [square brackets] in JavaScript
  - Apollo[13], Apollo[14]….
  - Rocky[1], Rocky[2], Rocky[3]….
- The simple way is to use Array in JavaScript

JavaScript Arrays [1/2]

var <variable> = new Array(<number of elements>)

- Notice that Array starts with an uppercase “A”
- Ex. var week = new Array(7);
  - new Array(7) specifies that the variable “week” will be an array variable
  - number in parentheses gives the number of array elements
- To refer to number of elements in an array, we use <variable>.length
- Array indexing begins at 0
- Greatest index of an array is <number of elements> − 1 (because the origin is 0)
- Array reference consists of array name with an index [enclosed in brackets]
  - week[0], week[1], week[2], week[3], week[4], week[5], week[6]

JavaScript Arrays [2/2]

```
1  var dwarf = new Array(7);      //Declarations use parens
2  var deux = 2;                  //Create a value for examples
3  dwarf[0] = "Happy";            //References use brackets
4  dwarf[1] = "Sleepy";           //Index by a constant
5  dwarf[deux] = "Dopey";         //Index by a variable
6  dwarf[deux + 1] = "Sneezy";    //Index by an expression
7  dwarf[2*deux] = "Bashful";
8  dwarf[3*deux - 1] = "Grumpy";
9  dwarf[10-(2*deux)] = "Doc";
10
11 dwarf
12
13 /*
14 Happy,Sleepy,Dopey,Sneezy,Bashful,Grumpy,Doc
15 */
```

```
var week = new Array(7);
12 for(var i = 0; i < week.length; i++) {
13     week[i] = dwarf[i] + " & " + dwarf[(i+1)%7] + " do dishes";
14 }
15
16 week
17
18 /*
19 Happy & Sleepy do dishes,
20 Sleepy & Dopey do dishes,
21 Dopey & Sneezy do dishes,
22 Sneezy & Bashful do dishes,
23 Bashful & Grumpy do dishes,
24 Grumpy & Doc do dishes,
25 Doc & Happy do dishes
26 */
```

Magic Decider

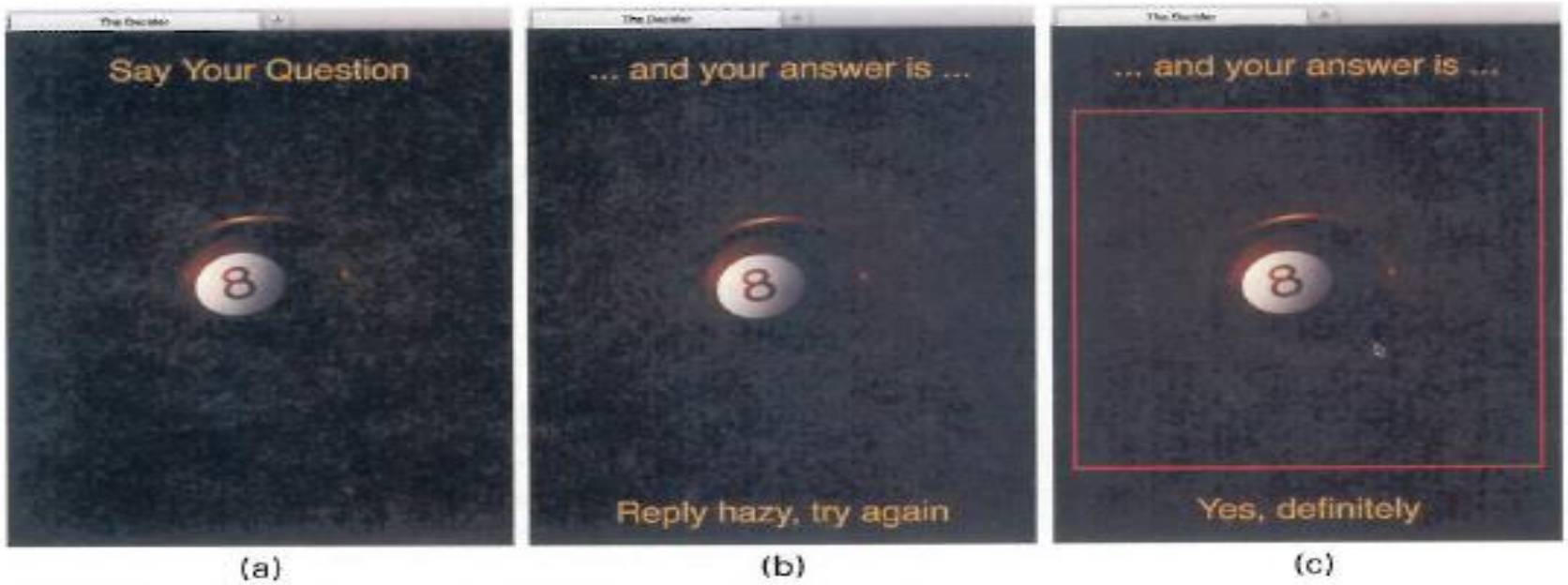

Figure 20.5 Magic Decider displays (a) the initial question request, (b) the UI after tapping or clicking on the image, and (c) the image with a visible border to show its extent.

```html
<!doctype html>
<html>
 <head>
  <meta charset="UTF-8"><title>The Decider</title>
  <style>
    body {background-color:black; color:orange;
          text-align:center; font-family:helvetica}
    button {margin:0; padding:0; background-color:black;
            border-style:none}
    p      {font-size:x-large; }
  </style>
  <script>
    var respond = new Array(
      "It is certain", "It is decidedly so", "Without a doubt",
      "Yes, definitely", "You may rely on it", "As I see it, yes",
      "Most likely", "Outlook good", "Yes", "Signs point to yes",
      "Reply hazy, try again", "Concentrate, and ask again",
      "Better not tell you now", "Cannot predict now",
      "Concentrate and ask again", "Don't count on it",
      "My reply is, no", "My sources say, no", "Outlook not so good",
      "Very doubtful");
    function randNum( range ) {
        return Math.floor( range * Math.random( ));
    }
  </script>
 </head>
 <body>
  <p id="ask"> Say Your Question</p>
  <p><button
     onclick="document.getElementById('ask').innerHTML='... and your answer is ... ';
              document.getElementById('tell').innerHTML=respond[randNum(20)]">
     <img src="8-ball.jpg" alt="8 Ball" width="300"/></button></p>
  <p id="tell" > </p>
 </body>
</html>
```

HTML document에 있는 문장들을 바꾸고 싶다면!!!!

var respond = new Array {“ab”, “fcd”, “bbb”,....}
// respond array의 initialization

HTML document의 모든 tag의 element는 document.getElementById() 로 접근가능

document.getElementById(“zz”).innerHTML = “xxyy”
// Tag의 Id가 zz인 문장을 “xxyy”로 교체

Figure 20.6 The HTML for the Magic Decider highlighting its three-paragraph structure.

The Busy Animation

- Movies & cartoons animate by the rapid display of many pictures known as frames
- Human visual perception is relatively slow so it's fooled into observing smooth motion when the display rate is about 30 fps (frame/sec) or 30 Hz
- Iteration, arrays, and indexing can be used for animation

- Animation in JavaScript requires 3 things:
  - Using a timer to initiate animation events
  - Prefetching the frames of the animation
  - Redrawing a Web page image

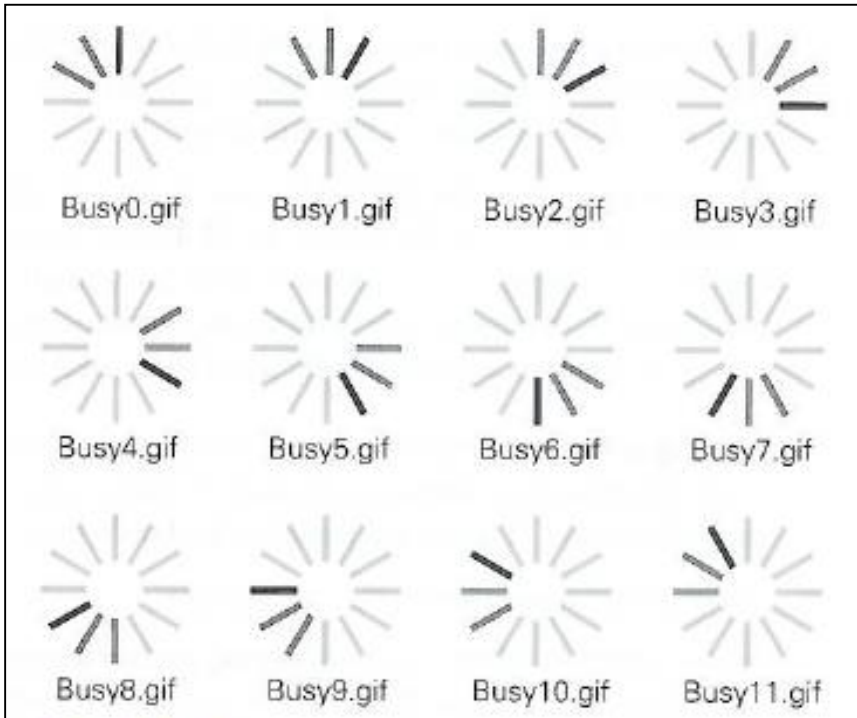

Figure 20.7 The .gif images for the Busy Animation. These files are available at pearsonhighered.com/snyder.

1. Using a Timer

- Graphics and animations in web browsers are event driven:
  - sit idle until an event occurs,
  - then they act,
  - and then idly wait for next event…repeat
- Suppose animation requires constant actions every 30 milliseconds!
  - 33 images / 1 sec is for smooth motion change for human visual perception
- The command to set a timer is setTimeout("<event handler>", <duration>)
  - setTimeout("animate( )", 30): 30 ms later the computer runs the animate( )
  - Animation's time unit is 1/000 sec (millisecond)
- Using a handle variable to refer to a Timer: timerID = setTimeout("animate()", 30);
  - To cancel timerID and stop animate(): clearTimeout(timerID);

2. Prefetching Images

- 12 icon image files are usually stored in separate directory "gifpix"
- Initializing to an Image Object:
  - Elements of the array must be initialized to a blank instance of an image object

```
var pics = new Array (12);
```

```
for (i = 0; i < pics.length; i++) {
    pics[i] = new Image( );
}
```

- Using the src attribute of an image
  - tag in HTML
  - Browser saves the name, gets the file, stores it in memory
  - 12 icon images:
    gifpix/Busy0.gif
    gifpix/Busy1.gif
    ….
    gifpix/Busy11.gif

```
pics[0].src = "gifpix/Busy0.gif"
```

```
for (var i = 0; i < pics.length; i++) {
  pics[i].src = "gifpix/Busy" + i + ".gif";
}
```

- pics[i].src = "gixpix/Busy" + I + ".gif" (Prefetching) is not visible on the screen
- <img src= "…">: is visible on the screen

3. Start Button and Stop Button

- Display first image at first (doesn't need to be animated yet)
- Buttons are be used to start (setTimeout()) and stop (clearTimeout()) animation

```
<body style="text-align:center"><p>
    <img src="gifpix/Busy0.gif" alt="spinner"/> <!--Initial Frame -->
    <form>
      <input type="button" value="Start"
          onclick='setTimeout("animate()",30);'/>
      <input type="button" value="Stop"
          onclick='clearTimeout(timerID);'/>
    </form>
</p></body>
```

- To animate (overwrite image with next sequenced image):
  - Loading an image one-at-a-time from outside directory is too slow
  - Get all images first, store them locally, then display them
- Images in an array are already numbered

4. Redrawing an Image

- To animate we need to overwrite it with the images that were prefetched
- When is encountered the HTML document, browser fills its images in document.images as an array

```
<img src="gifpix/Busy0.gif" alt="spinner"/>
```

- To change initial frame, write:

```
document.images[0].src = pics[i].src;
```

- Defining the animate( ) event Handler
  - 12 images must be changed, cyclically, one every 30 ms
  - animate( ) event handler overwrites the image, sets up for the next frame, and sets the timer to call itself again:

```
function animate () {
    document.images[0].src = pics[frame].src;
    frame = (frame + 1)%12;
    timerID = setTimeout ("animate()", 30 ));
}
```

Complete Busy Animation

```
<!doctype html>
<html>
  <head>
   <meta charset="UTF-8"/><title>Spinner</title>
   <style>
     body {text-align:center}
   </style>
   <script>
    var frame = 0;                          //Frame counter
    var timerID;                            //Timer handle var
    var pics = new Array(12);               //Array for prefetched gifs
    function animate( ) {
    document.images[0].src = pics[frame].src;
    frame = (frame+1)%12;
    timerID = setTimeout("animate( )", 100);
}
    for (var i = 0; i < pics.length; i++) {
      pics[i] = new Image( );
    }
    for (var i = 0; i < pics.length; i++) {
      pics[i].src = "gifpix/Busy" + i + ".gif";
    }
   </script>
  </head>
  <body>
    <img src="gifpix/Busy0.gif" alt="spinner"/>
    <form>
      <input type="button" value="Start"
          onclick='setTimeout("animate( )",100);'/>
      <input type="button" value="Stop"
          onclick='clearTimeout(timerID);'/>
    </form>
  </body>
</html>
```

Figure 20.8 The Busy Animation program, assuming that the 12 .gif files are stored in a folder called gifpix.

new Image() 는 image object 생성

document.images는 현재 HTML document에 있는 image들을 array로 가지고 있는다

document.images[0].src 에 새로운 image가 저장되면 img tag의 src가 변경되면서 img tag가 수행됨

setTimeout(code, delay)는1/000초단위의 delay이후에 code를 실행하는 JavaScript의 built-in method

clearTimeout()은 setTimeout()으로 실행중인 code를 정지시킴

보조자료

document.images

`document.images` returns a collection of the images in the current HTML document.

Syntax

```
var htmlCollection = document.images;
```

Example

```
var ilist = document.images;

for(var i = 0; i < ilist.length; i++) {
    if(ilist[i].src == "banner.gif") {
        // found the banner
    }
}
```

Notes

`document.images.length` – property, returns the number of images on the page.

`document.images` is part of DOM HTML, and it only works for HTML documents.

Not So Busy Animation: Rock-Paper-Scissor

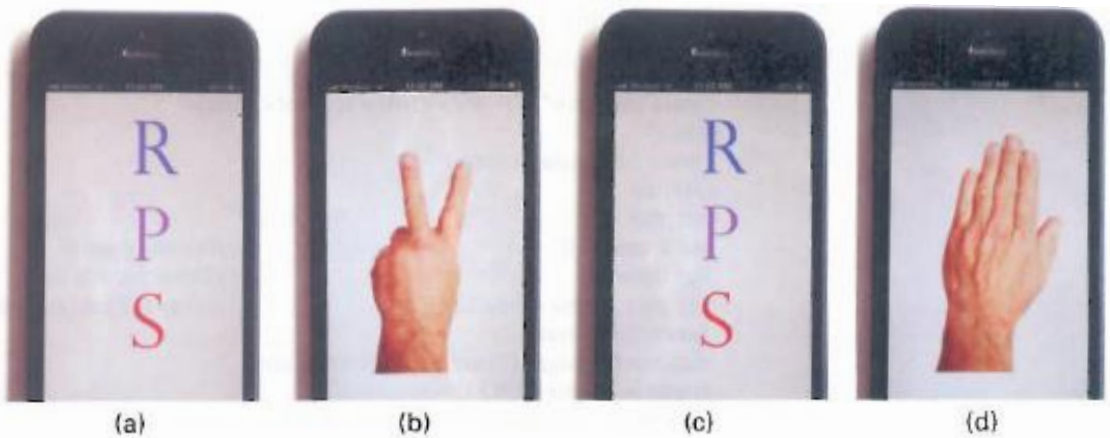

Figure 20.9 The Rock-Paper-Scissors app's operation: (a) splash image, (b) random choice, (c) return to splash for next throw, and (d) random choice. All transitions are caused by click/tap on image.

- 3 Key Ideas
  - Saving state: The app needs to remember which picture to display next
  - Prefetching: Just as the Busy Animation prefetched images and stored them locally so they could be displayed rapidly, the RPS app does the same
  - Changing document.images: We used an array known as document.images

```html
<!doctype html>
<html>
 <head><meta charset="UTF-8"/><title>RPS</title>
 <style> button {margin:0; padding:0; background-color:white;
                 border-style:none; border-width:0}
         p {text-align:center}      <!--above styling centers pic-->
 </style>
 <script>    //this code prefetches, randomizes and flips a picture
  var thro = 1;                          //alternates betw 0 and 1
  var pix = new Array(4);                //array to hold 4 pictures
  for (var i=0; i<4; i++){
    pix[i] = new Image( );               //set up element for pics
  }
  pix[0].src = "im/splash.gif";          //prefetch the 4 pics
  pix[1].src = "im/rock.gif";
  pix[2].src = "im/paper.gif";
  pix[3].src = "im/scissors.gif";
  function randNum( range ) {            //old randomizing friend
    return Math.floor( range * Math.random( ));
  }
  function rps( ) {                      //display a new image
    if (thro == 1)                       //is this a throw or reset?
      document.images[0].src             //throw, change picture
          =pix[1+randNum(3)].src;        //its random from pix 1-3
    else
      document.images[0].src             //reset, change picture
          =pix[0].src;                   //to splash picture
    thro = 1-thro;                       //flip thro for next time
  }
 </script>
 </head>
 <body><p>
   <!--The program is just a picture that acts as a button
       flipping between the splash page and a random throw-->
   <button onclick="rps( )">
     <img src="im/splash.gif"
         alt="R-P-S Throw" height="300"/>
   </button></p>
 </body>
</html>
```

Figure 20.10 The JavaScript for the Rock-Paper-Scissors app, summarizing the ideas of the last two sections.

Click할때마다 한번은 splash.gif를 보이고 한번은 rock, paper, scissor중에 random하게 보여주는 방식

Summary

- The principles of iteration ensure that every iteration contains a test and that the test is dependent on variables that change in the loop
- Programmers routinely use the World-Famous Iteration (WFI), a stylized iteration that begins at 0, tests that the iteration variable is strictly less than some limit, and increments by 1
- In indexing, we create a series of names “array” by associating a number with a base name
- Arrays and iterations can be effectively used together’
- All animations achieve the appearance of motion by rapidly displaying a series of still frames