1 **Processing: Looping Statements**

CST112

2 **Algorithms**

- Procedure for solving problem:
  1. *Actions* to be executed
  2. *Order* in which actions are executed
- The *order* of the elements of an algorithm are very important …
  - Even if the order *appears* insignificant, errors can have far-reaching results

3 **Research of Bohm and Jacopini**

- They proposed that all programs can be written in terms of three control structures:
  - The type of instructions that specify the order in which statements in a program is executed
  1. Sequence structure—step by step in order
  2. Selection (decision or conditional) structure
  3. Repetition (iteration or loop) structure—repeating blocks of statements

4 **Iteration (Loop) Statements**

- Provides *repeated* execution of a block of statements
- The loop continues:
  1. A specified number of times (counter-controlled) or …
  2. While a *condition* is met (sentinel-controlled)
- Also called repetition or do while structure
  - *Meaning* do the loop while condition is True

5 **Iteration (Loop) Pattern**

- One or more statements in a block are executed repeatedly
- Loop continues while certain condition is true

```
while (Another rectangle?)
{
    Set random fill color;
    Set size 10 pixels smaller;
    Draw rectangle;
}
```

6 **The for Loop (Page 1)**

- Implements a loop by *counting* a specific number of iterations (repetitions)
  - Counter-controlled looping
- Appropriate when exact number of loop repetitions *is known*
- Format:

```
for (initialize; booleanExpression; increment)
{
    Statement(s) to be repeated;
}
```

7 **The for Loop (Page 2)**

- Example (three expressions in the parentheses):

for (ctr = 0; ctr <= 9; ctr = ctr + 1)

- The *initialize* component (ctr = 0)
  - Value assigned to a *counter* variable when the loop is first encountered in the program
- The *booleanExpression* component (ctr <= 9)
  - *Relation condition* which is tested to determine if the loop should be entered again
- The *increment* component (ctr = ctr + 1)
  - Indicates by what value the counter *changes* at the beginning of each subsequent loop

19 **Strings (Page 1)**

- A string is a sequence of characters
- Strings are always defined inside double quotes ("abc")
  - Alternately characters (type char) are always defined inside single quotes ('a') and may only contain a single character

20 **Strings (Page 2)**

- The class String includes methods for:
  - Examining individual characters within strings
  - Comparing strings
  - Searching strings
  - Extracting parts of strings
  - Converting an entire string to uppercase or lowercase
  - Etc.

21 **The text() Function (Page 1)**

- "Draws" data to the screen
  - The data may be *text*, a char, an int or a float
- Displays the information specified in the first parameter on the screen in the position specified by the additional two parameters
- By default the text displays starting from and then to the right of the positions coordinates
- The fill() function controls the color of the text (default always is white)

22 **The text() Function (Page 2)**

- Format:

  text(*theData*, *xCoordinate*, *yCoordinate*);
- Examples:

  text("Hello", mouseX, mouseY);

  text(ctr, 100, 100);

26 **Relationship of draw() and mousePressed() functions**

- A draw() function is needed so the output from the mousePressed() function will be visible
- This is true even if draw() is empty; otherwise there will be no output to the Processing output window

27 **Assignment Operators**

- Also known as op equals operators
- Assigns an *updated* value to a variable

Operator	Example	Explanation
+=	ctr += 1;	ctr = ctr + 1;
-=	ctr -= 17;	ctr = ctr - 17;
*=	ctr *= 8;	ctr = ctr * 8;
/=	ctr /= 5;	ctr = ctr / 5;

30 Unary Operators (Page 1)

- Unary operators update variables values by adding (increment operator) or subtracting (decrement operator) value of 1 to (or from)

Operator	Example	Explanation
++	ctr++;	ctr = ctr + 1; (post)
++	++ctr;	ctr = ctr + 1; (pre)
--	ctr--;	ctr = ctr - 1; (post)
--	--ctr;	ctr = ctr - 1; (pre)

31 Unary Operators (Page 2)

- If operator is a *prefix*, the value is returned *after* it is increased or decreased:
  - When the variable ctr = 5:
    ```
    newVar = ++ctr;  // newVar will be 6
    ```
- If operator is a *suffix*, the value is returned *before*:
  - When the variable ctr = 5:
    ```
    newVar = ctr++;  // newVar will be 5
    ```
- Final value of ctr in *both* cases will be 6

36 Variable Scope (Page 1)

- Variables are recognized in the block in which they are declared …
  - Including loop and conditional blocks
- And all subordinate blocks
- If a variable is declared before the first function (i.e. before the setup() function) or outside of any function, that variable has global scope
  - It is recognized inside every function in application

37 Variable Scope (Page 2)

- In the following example, the variable ctr only is accessible inside the mousePressed() function
  - It would not be accessible in any other function:

```
void mousePressed()
{
    int ctr;

    for (ctr = 0; ctr < 10;  ctr++)
    {
        println(ctr);
    }
}
```

40 Variable Scope (Page 3)

- In the following example, the variable ctr only is accessible inside the for block:

```
for (int ctr = 0; ctr < 10;  ctr++)
```

```
{
    println(ctr);
}
println(ctr);
```

- The last statement would result in a compile error

48 **The while Loop (Page 1)**

- Continues to repeat a loop as long as a *controlling condition* is true (pre-test)
- The variable controlling the condition is updated by logic within the loop
  - Exact number of loops is *usually unknown*
- Format:

```
while (booleanExpression)
{
    Statement(s) to be repeated as long as the booleanExpression is true;
}
```

49 **The while Loop (Page 2)**

- Example:

```
int ctr = 600;
while (ctr > 0)
{
    rect(width / 2, width / 2, ctr, ctr);
    ctr -= 10;
}
```

51 **Remember the if Format …**

- Format of the if statement:

```
if (booleanExpression)
{
    Statement(s) to be executed if the booleanExpression is true;
}
```

- Format of the while statement:

```
while (booleanExpression)
{
    Statement(s) to be repeated as long as the booleanExpression is true;
}
```

52 **Comparing for and while**

- The for example:

```
for (ctr = 1; ctr <= 10; ctr++)
{
    println(ctr);
}
```

- The while example :

```
ctr = 1;   // Initialize
while (ctr <= 10)  // Boolean test
{
    println(ctr);
    ctr++;  // Increment
}
```

57 **The do while Loop (Page 1)**

- Continues to repeat a loop as long as a *controlling condition* is true
- Performs the test at the *conclusion* of the execution of each loop (post-test)
- Format:

```
do
{
    Statement(s) to be repeated as long as the booleanExpression is true;
}
while (booleanExpression);
```

58 **The do while Loop (Page 2)**

- Example:

```
do
{
    rect(mySize / 2, mySize / 2, ctr, ctr);
    ctr -= 10;
}
while (ctr > 0);
```

- A do while loop always executes *at least one* time

59 **Comparing while and do while**

- The while Format:

```
while (booleanExpression)
{
    Statement(s) to be repeated as long as the booleanExpression is true;
}
```

- The do while Format:

```
do
{
    Statement(s) to be repeated as long as the booleanExpression is true;
} while (booleanExpression);
```

64 **Looping Inside draw() Function**

- Effectively any loop structure placed inside the draw() function is a nested loop (an inner loop inside an outer loop)
- Remember that *Processing* only updates the output display at the conclusion of each draw() function
    - Therefore changes that occur during draw() do not render until each of its iterations is done executing