Crash Course: Repetition Structures 2

ST. MARY'S HIGH SCHOOL

In this crash course

ST. MARY'S HIGH SCHOOL

Different types of loops – While loop

- This is the syntax or formula for a while loop:

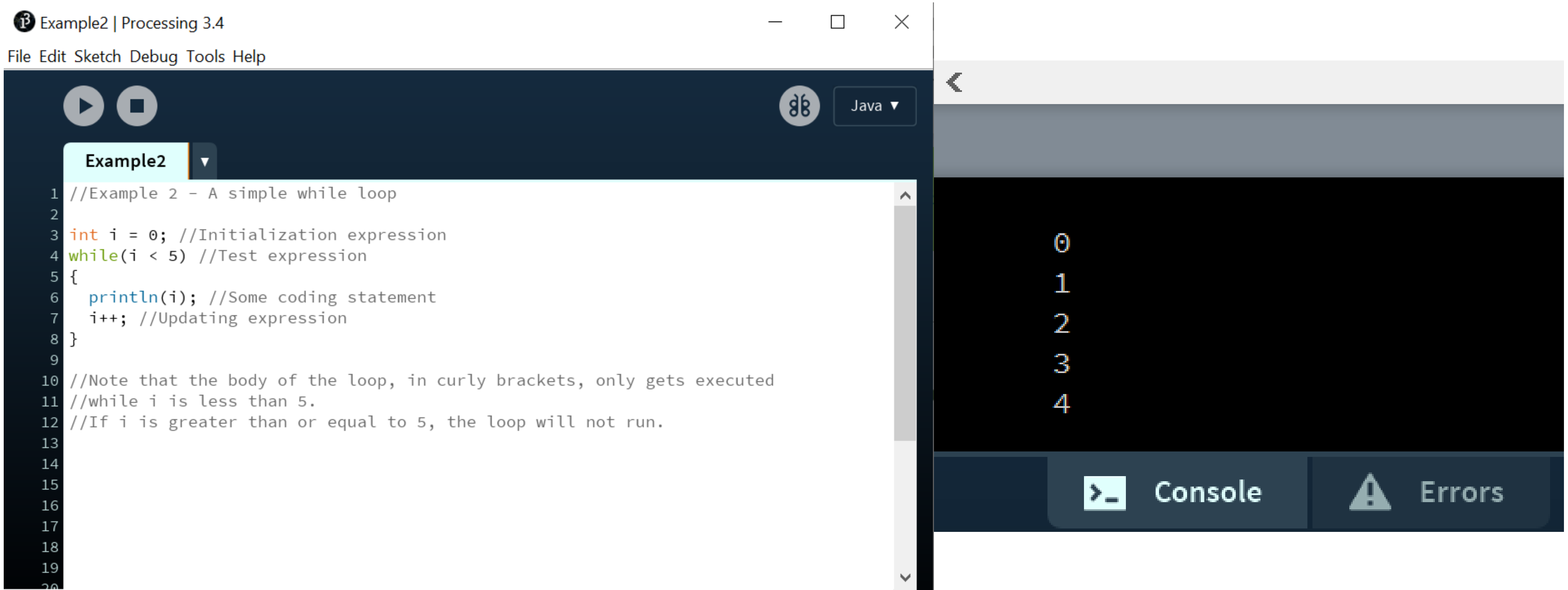
```
//Anatomy of a while loop

initialization_expression;

while(test_expression)
{
            //Some coding statement(s);
             updating_expression;
}

//Only while the test_expression is true do we iterate
//As soon as it becomes false, we exit the loop
//Note that if you only have one statement in the body
//of the loop, curly brackets {} are not necessary
//Notice that there is no semicolon at the end
```

While loop – Example 2

- Let's start with a very simple example
- Let's say you want to print the five consecutive numbers from 0 through 4 to the console

While loop – Example 2

- We can do it like this:



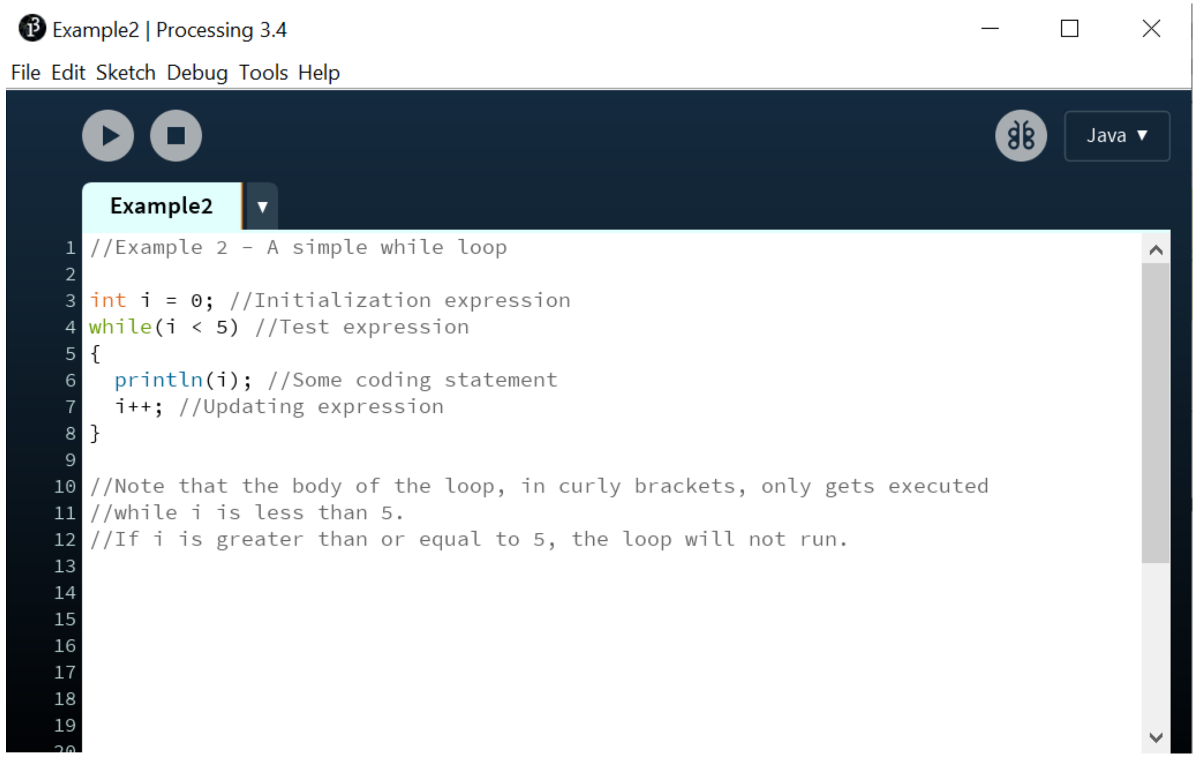
```
//Example 2 - A simple while loop

int i = 0; //Initialization expression
while(i < 5) //Test expression
{
  println(i); //Some coding statement
  i++; //Updating expression
}

//Note that the body of the loop, in curly brackets, only gets executed
//while i is less than 5.
//If i is greater than or equal to 5, the loop will not run.
```

```
0
1
2
3
4
```

ST. MARY'S HIGH SCHOOL

While loop – Example 2

- So what's going on here?
- Let's step through each iteration



```
//Example 2 - A simple while loop

int i = 0; //Initialization expression
while(i < 5) //Test expression
{
  println(i); //Some coding statement
  i++; //Updating expression
}

//Note that the body of the loop, in curly brackets, only gets executed
//while i is less than 5.
//If i is greater than or equal to 5, the loop will not run.
```

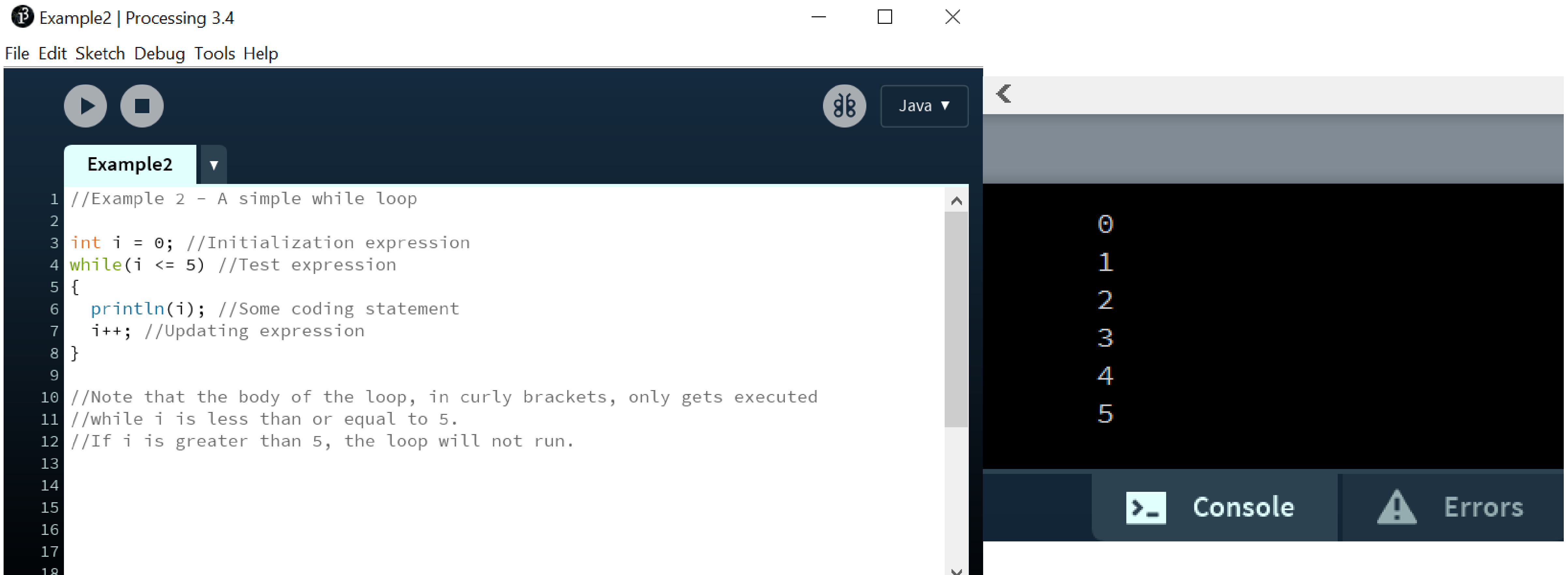
Notice that the loop runs for five iterations: namely i = 0, 1, 2, 3, 4

i = 0	Is i < 5?	Yes, 0 < 5, print	Increment i
i = 1	Is i < 5?	Yes, 1 < 5, print	Increment i
i = 2	Is i < 5?	Yes, 2 < 5, print	Increment i
i = 3	Is i < 5?	Yes, 3 < 5, print	Increment i
i = 4	Is i < 5?	Yes, 4 < 5, print	Increment i
i = 5	Is i < 5?	No, 5 = 5, stop	

ST. MARY'S HIGH SCHOOL

While loop – Example 2

- What happens when we use <= instead of <?



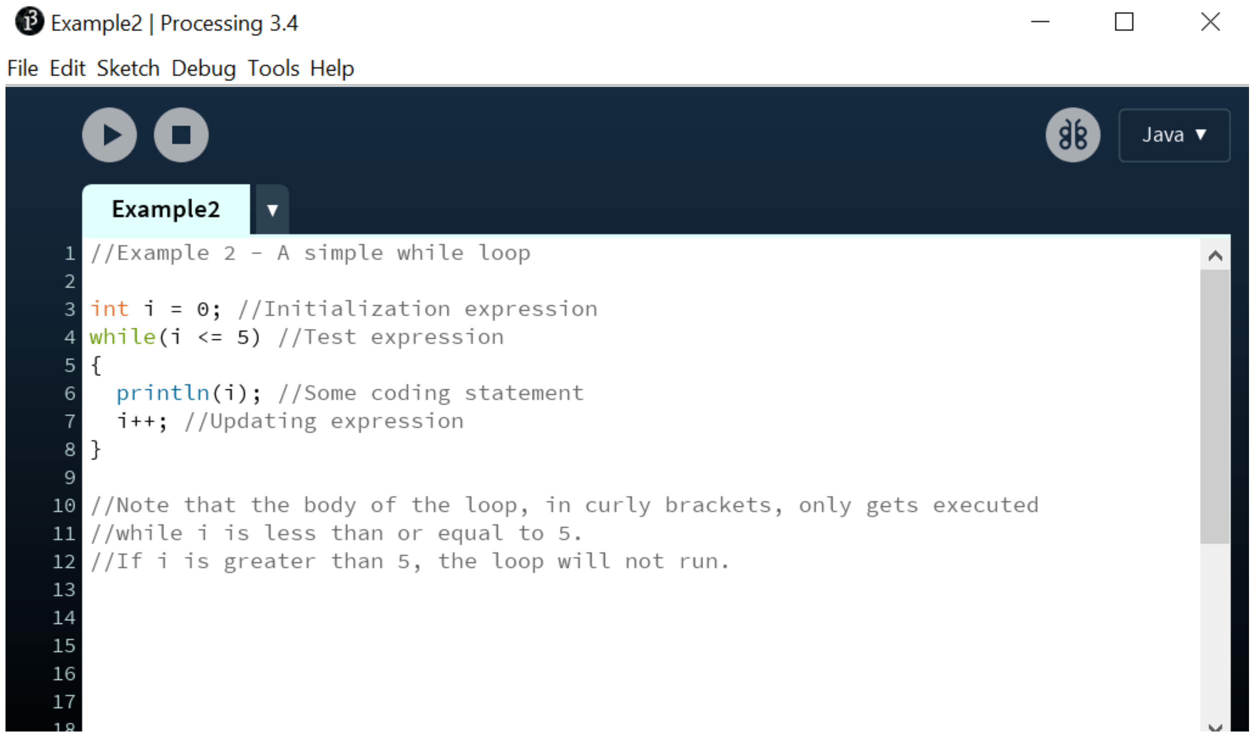
```
//Example 2 - A simple while loop

int i = 0; //Initialization expression
while(i <= 5) //Test expression
{
  println(i); //Some coding statement
  i++; //Updating expression
}

//Note that the body of the loop, in curly brackets, only gets executed
//while i is less than or equal to 5.
//If i is greater than 5, the loop will not run.
```

```
0
1
2
3
4
5
```

While loop – Example 2

Notice that the loop runs for six iterations: namely i = 0, 1, 2, 3, 4, 5

- So what's going on here?
- Let's step through each iteration



Example2 | Processing 3.4

File Edit Sketch Debug Tools Help

Example2

```
//Example 2 - A simple while loop

int i = 0; //Initialization expression
while(i <= 5) //Test expression
{
  println(i); //Some coding statement
  i++; //Updating expression
}

//Note that the body of the loop, in curly brackets, only gets executed
//while i is less than or equal to 5.
//If i is greater than 5, the loop will not run.
```

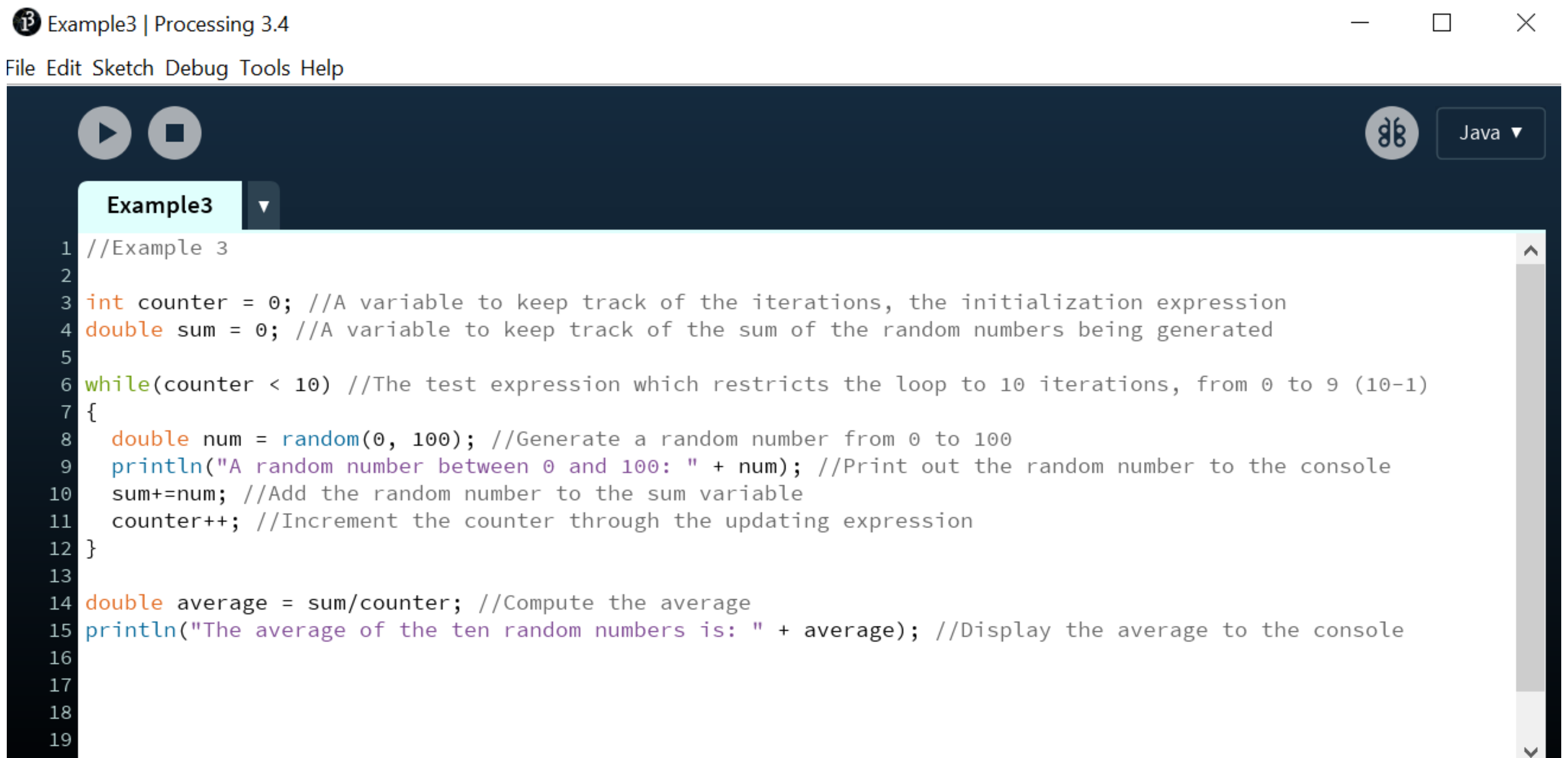
i = 0	Is i <= 5?	Yes, 0 < 5, print	Increment i
i = 1	Is i < =5?	Yes, 1 < 5, print	Increment i
i = 2	Is i <= 5?	Yes, 2 < 5, print	Increment i
i = 3	Is i <= 5?	Yes, 3 < 5, print	Increment i
i = 4	Is i <= 5?	Yes, 4 < 5, print	Increment i
i = 5	Is i <= 5?	Yes, 5 = 5, print,	Increment i
i = 6	Is i <= 5?	No, 6 > 5, stop	

While loop – Example 2

- So the difference was that with <=, you got another iteration
- In general, to iterate n times, you could start with i = 0 and continue while i < n or start with i = 1 and continue while i <= n or even start with i = 0 and continue while i <= n - 1
- In general, to iterate n+1 times, you could start with i = 0 and continue while i <= n or start with i = 0 and continue while i < n + 1

While loop – Example 3

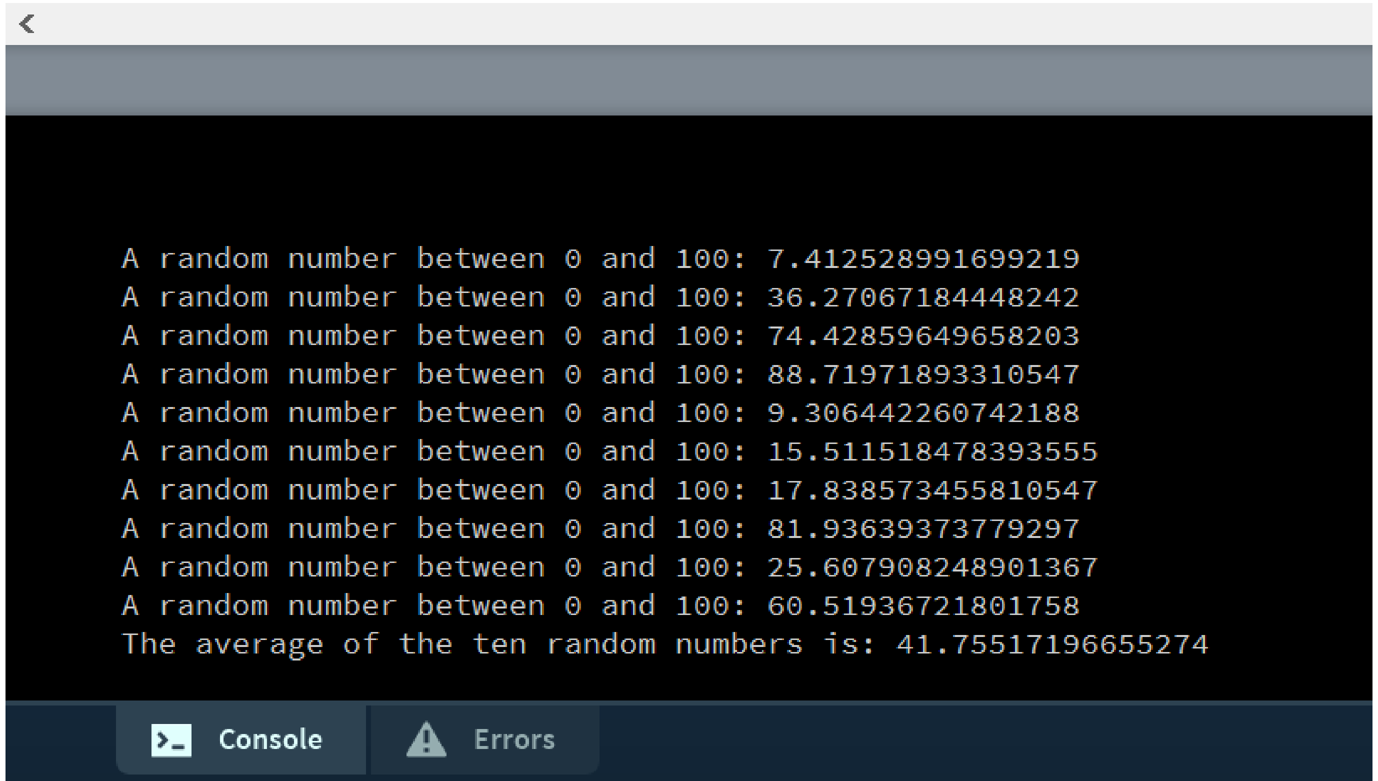
- Write a program that generates 10 numbers from 0 to 100 and then averages those numbers



```
//Example 3

int counter = 0; //A variable to keep track of the iterations, the initialization expression
double sum = 0; //A variable to keep track of the sum of the random numbers being generated

while(counter < 10) //The test expression which restricts the loop to 10 iterations, from 0 to 9 (10-1)
{
  double num = random(0, 100); //Generate a random number from 0 to 100
  println("A random number between 0 and 100: " + num); //Print out the random number to the console
  sum+=num; //Add the random number to the sum variable
  counter++; //Increment the counter through the updating expression
}

double average = sum/counter; //Compute the average
println("The average of the ten random numbers is: " + average); //Display the average to the console
```

While loop – Example 3

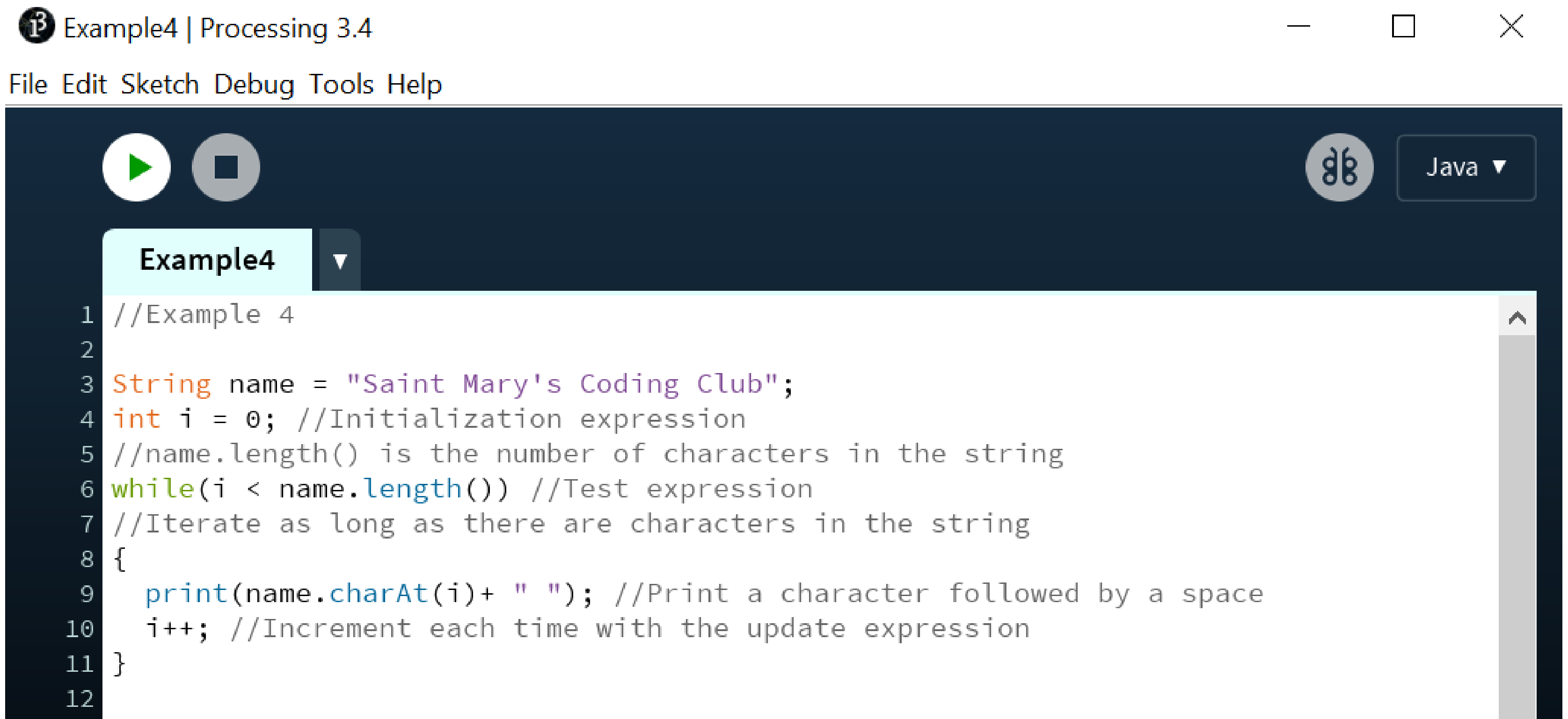
- Write a program that generates 10 numbers from 0 to 100 and then averages those numbers



```
A random number between 0 and 100: 7.412528991699219
A random number between 0 and 100: 36.27067184448242
A random number between 0 and 100: 74.42859649658203
A random number between 0 and 100: 88.71971893310547
A random number between 0 and 100: 9.306442260742188
A random number between 0 and 100: 15.511518478393555
A random number between 0 and 100: 17.838573455810547
A random number between 0 and 100: 81.93639373779297
A random number between 0 and 100: 25.607908248901367
A random number between 0 and 100: 60.51936721801758
The average of the ten random numbers is: 41.75517196655274
```

While loop – Example 4

- Write a program that takes a string and prints it out character by character with spaces in between

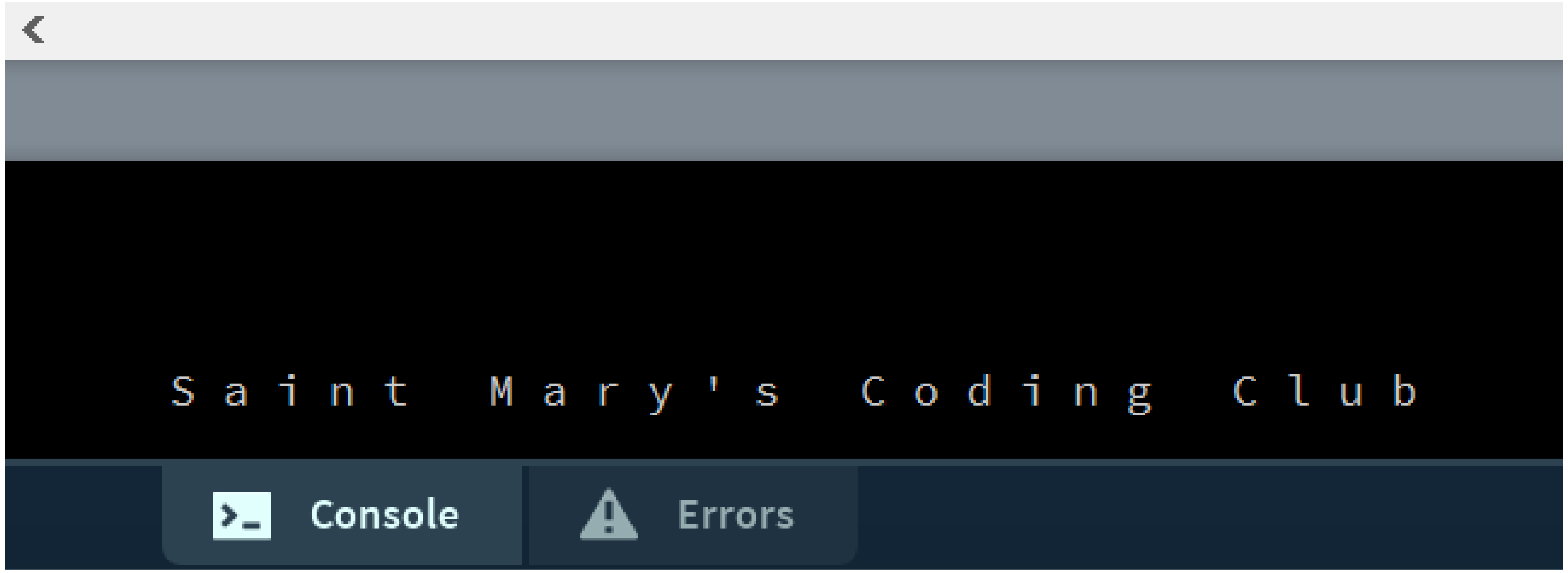


```
//Example 4

String name = "Saint Mary's Coding Club";
int i = 0; //Initialization expression
//name.length() is the number of characters in the string
while(i < name.length()) //Test expression
//Iterate as long as there are characters in the string
{
  print(name.charAt(i)+ " "); //Print a character followed by a space
  i++; //Increment each time with the update expression
}

```

ST. MARY'S HIGH SCHOOL

While loop – Example 4

- Write a program that takes a string and prints it out character by character with spaces in between



```
S a i n t   M a r y ' s   C o d i n g   C l u b
```

Different types of loops – For loop

- This is the syntax or formula for a for loop:

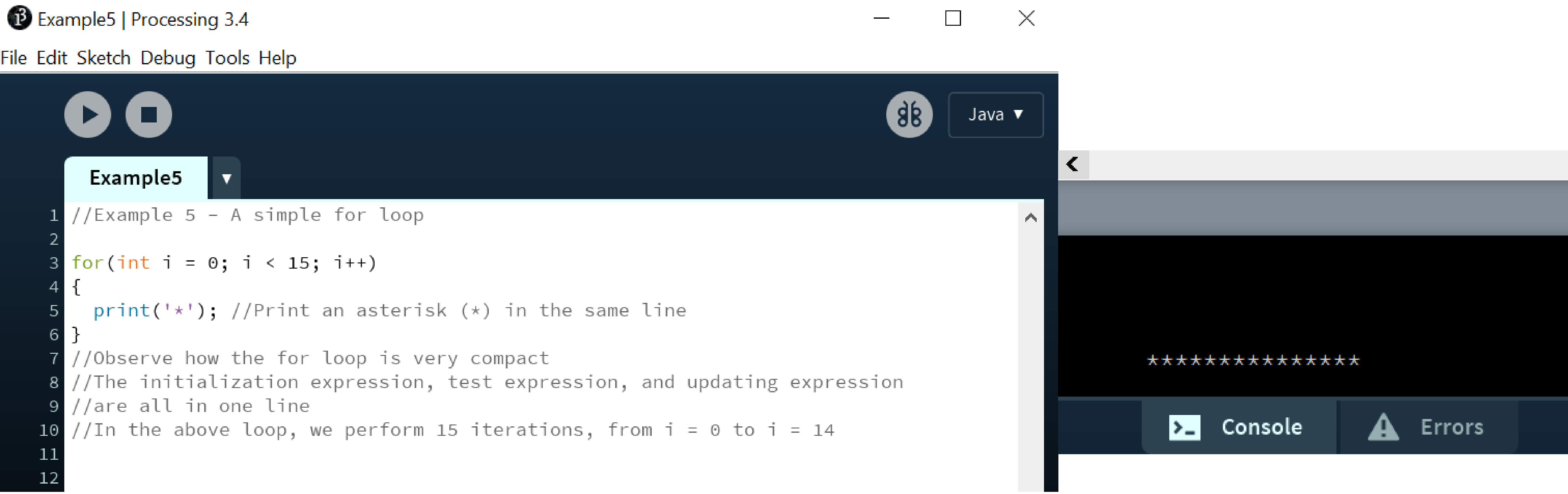
```
//Anatomy of a for loop

for(initialization_expression; test_expression; updating_expression)
{
        //Some coding statement(s);
}

//Only while the test_expression is true do we iterate
//As soon as it becomes false, we exit the loop
//Note that if you only have one statement in the body
//of the loop, curly brackets {} are not necessary
//Notice that there is no semicolon at the end
```

For loop – Example 5

- Let’s start with a very simple example
- Let’s say you want to print 15 asterisks (*) in the same line (not a new line)

ST. MARY'S HIGH SCHOOL

For loop – Example 5

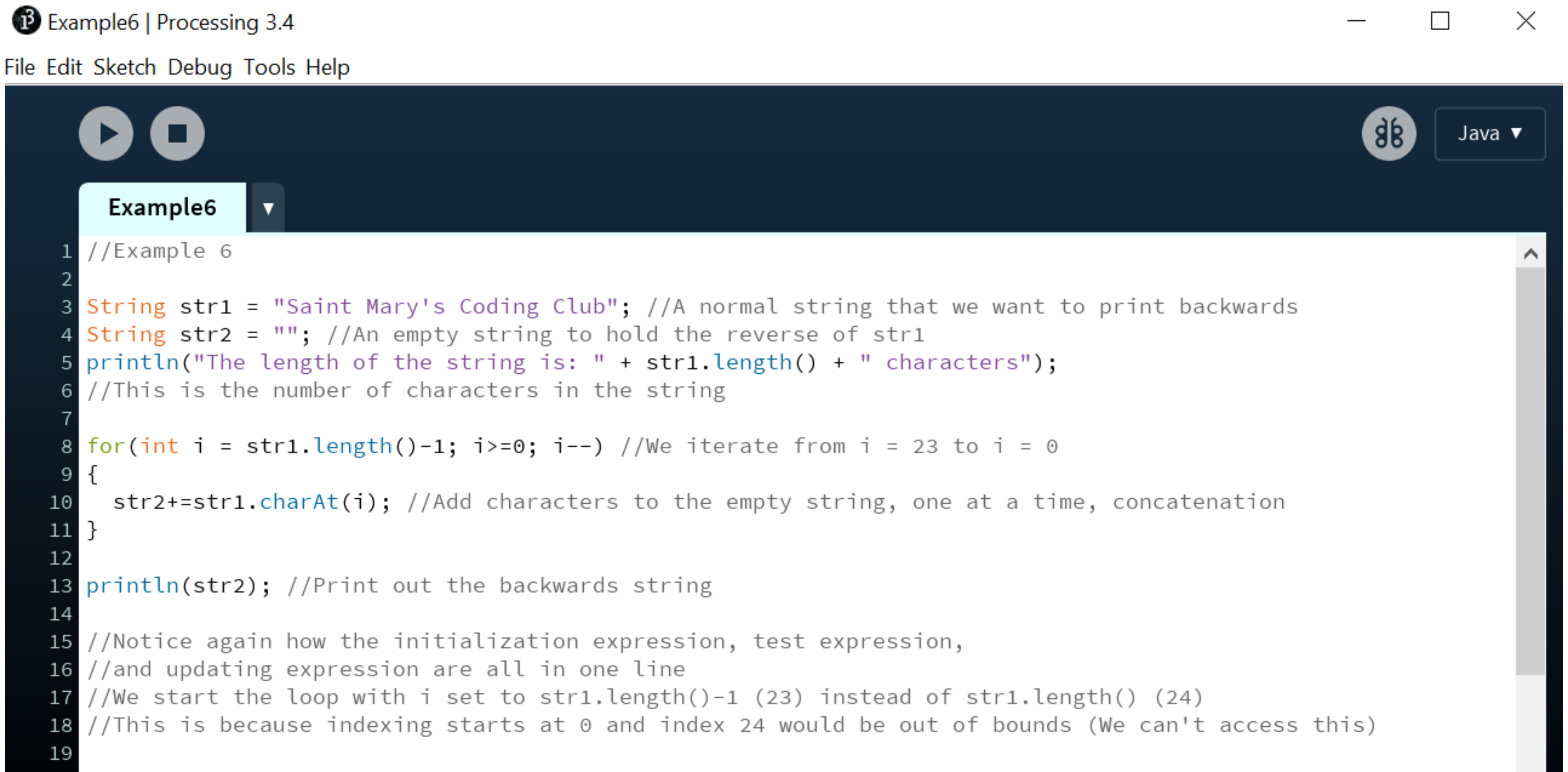


```
//Example 5 - A simple for loop

for(int i = 0; i < 15; i++)
{
  print('*'); //Print an asterisk (*) in the same line
}
//Observe how the for loop is very compact
//The initialization expression, test expression, and updating expression
//are all in one line
//In the above loop, we perform 15 iterations, from i = 0 to i = 14


```

```
***************
```

For loop – Example 6

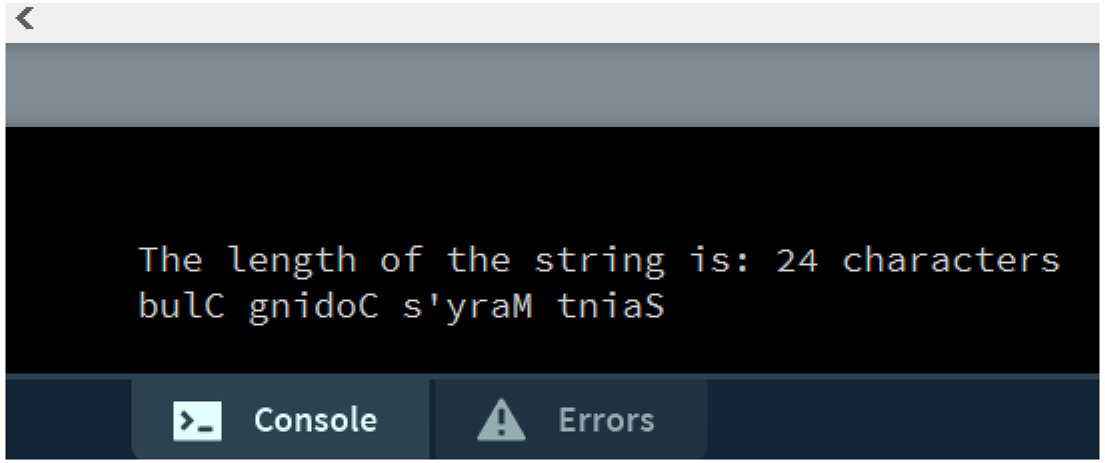
- Write a program that prints a string backwards



```
//Example 6

String str1 = "Saint Mary's Coding Club"; //A normal string that we want to print backwards
String str2 = ""; //An empty string to hold the reverse of str1
println("The length of the string is: " + str1.length() + " characters");
//This is the number of characters in the string

for(int i = str1.length()-1; i>=0; i--) //We iterate from i = 23 to i = 0
{
  str2+=str1.charAt(i); //Add characters to the empty string, one at a time, concatenation
}

println(str2); //Print out the backwards string

//Notice again how the initialization expression, test expression,
//and updating expression are all in one line
//We start the loop with i set to str1.length()-1 (23) instead of str1.length() (24)
//This is because indexing starts at 0 and index 24 would be out of bounds (We can't access this)

```

For loop – Example 6

- Write a program that prints a string backwards

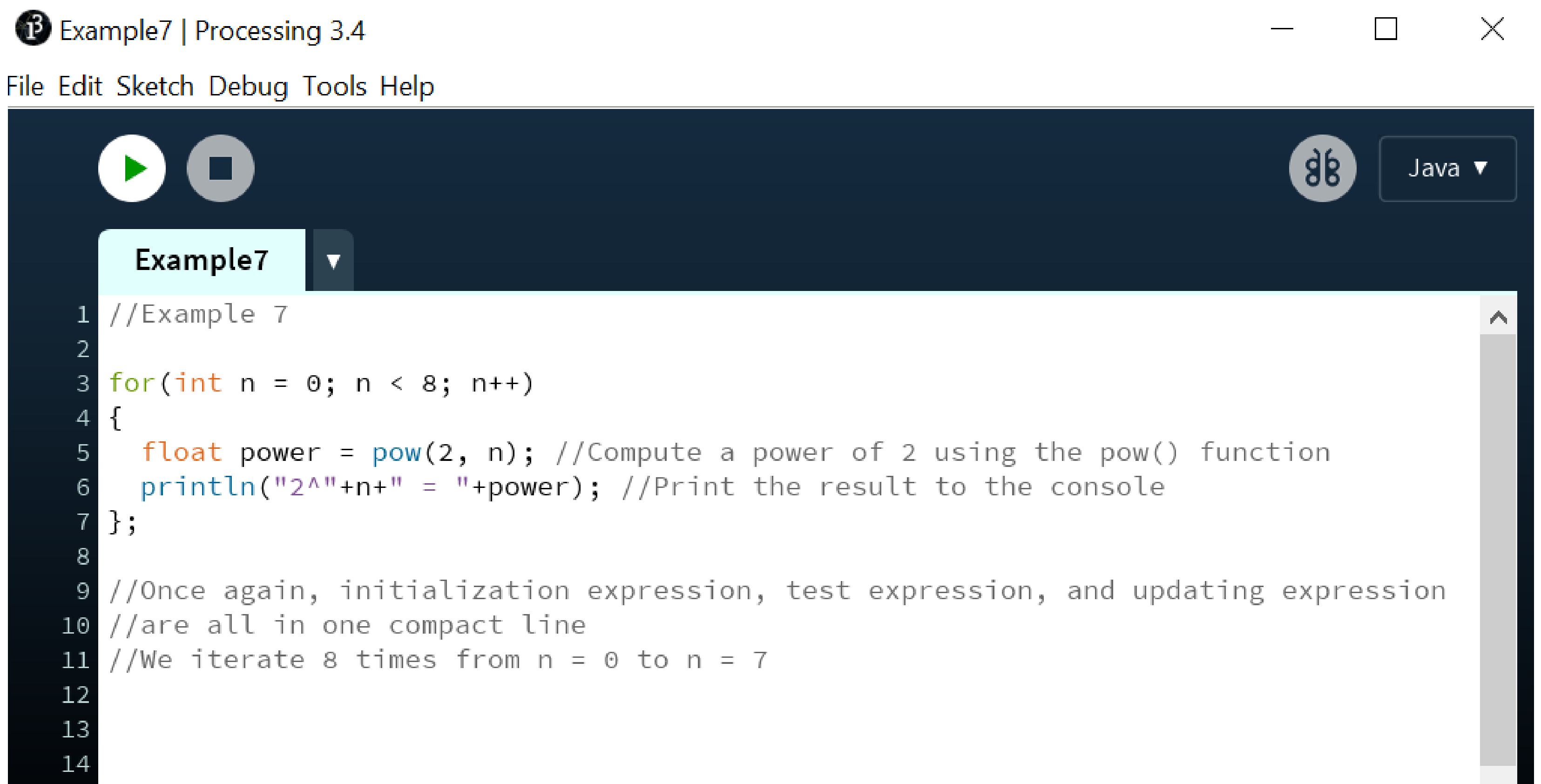


```
The length of the string is: 24 characters
bulC gnidoC s'yraM tniaS
```

Character	S	a	i	n	t		M	a	r	y	‘	s		C	o	d	i	n	g		C	l	u	b
Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

For loop – Example 7

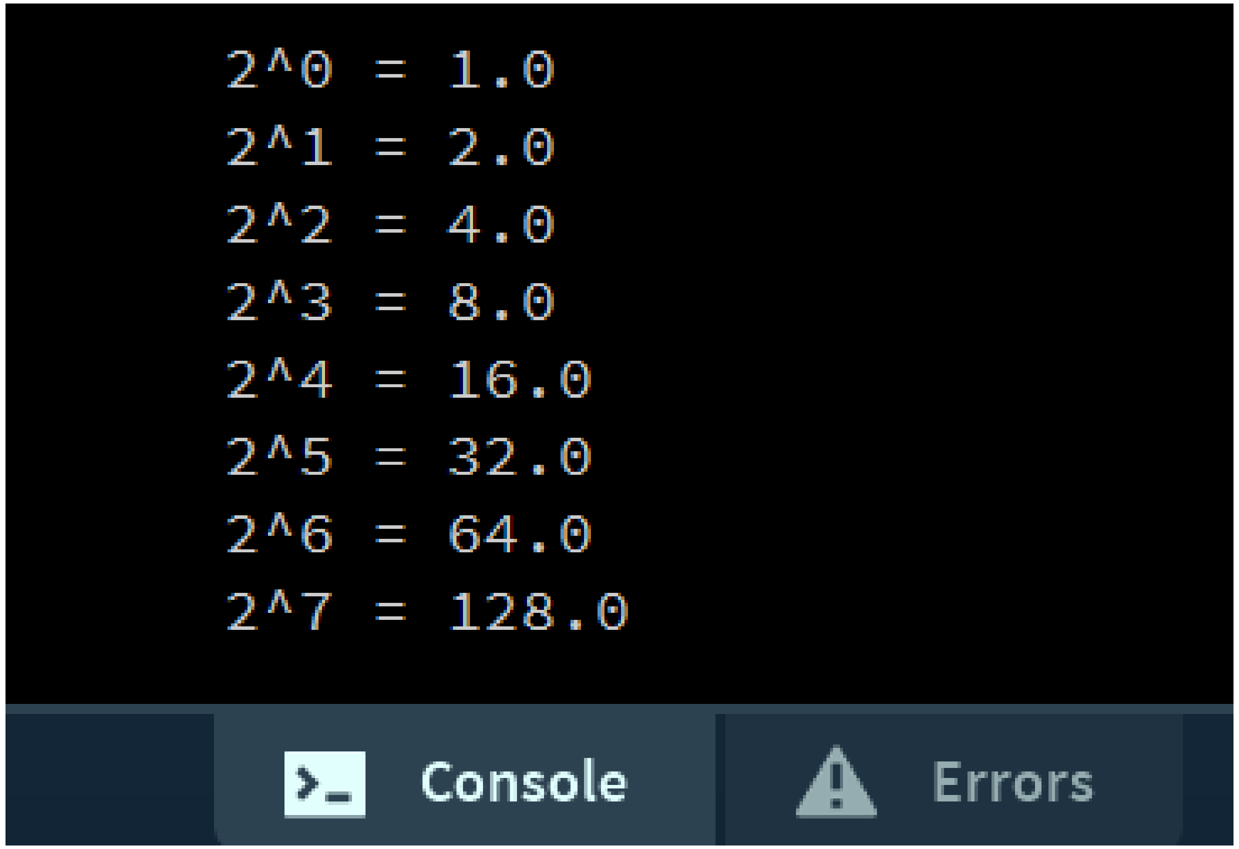
- Write a program that prints out the first 8 powers of 2 from 2^0 to 2^7



```
//Example 7

for(int n = 0; n < 8; n++)
{
  float power = pow(2, n); //Compute a power of 2 using the pow() function
  println("2^"+n+" = "+power); //Print the result to the console
};

//Once again, initialization expression, test expression, and updating expression
//are all in one compact line
//We iterate 8 times from n = 0 to n = 7
```

For loop – Example 7

- Write a program that prints out the first 8 powers of 2 from 2^0 to 2^7



```
2^0 = 1.0
2^1 = 2.0
2^2 = 4.0
2^3 = 8.0
2^4 = 16.0
2^5 = 32.0
2^6 = 64.0
2^7 = 128.0
```

Different types of loops – Do while loop

- This is the syntax or formula for a do while loop:

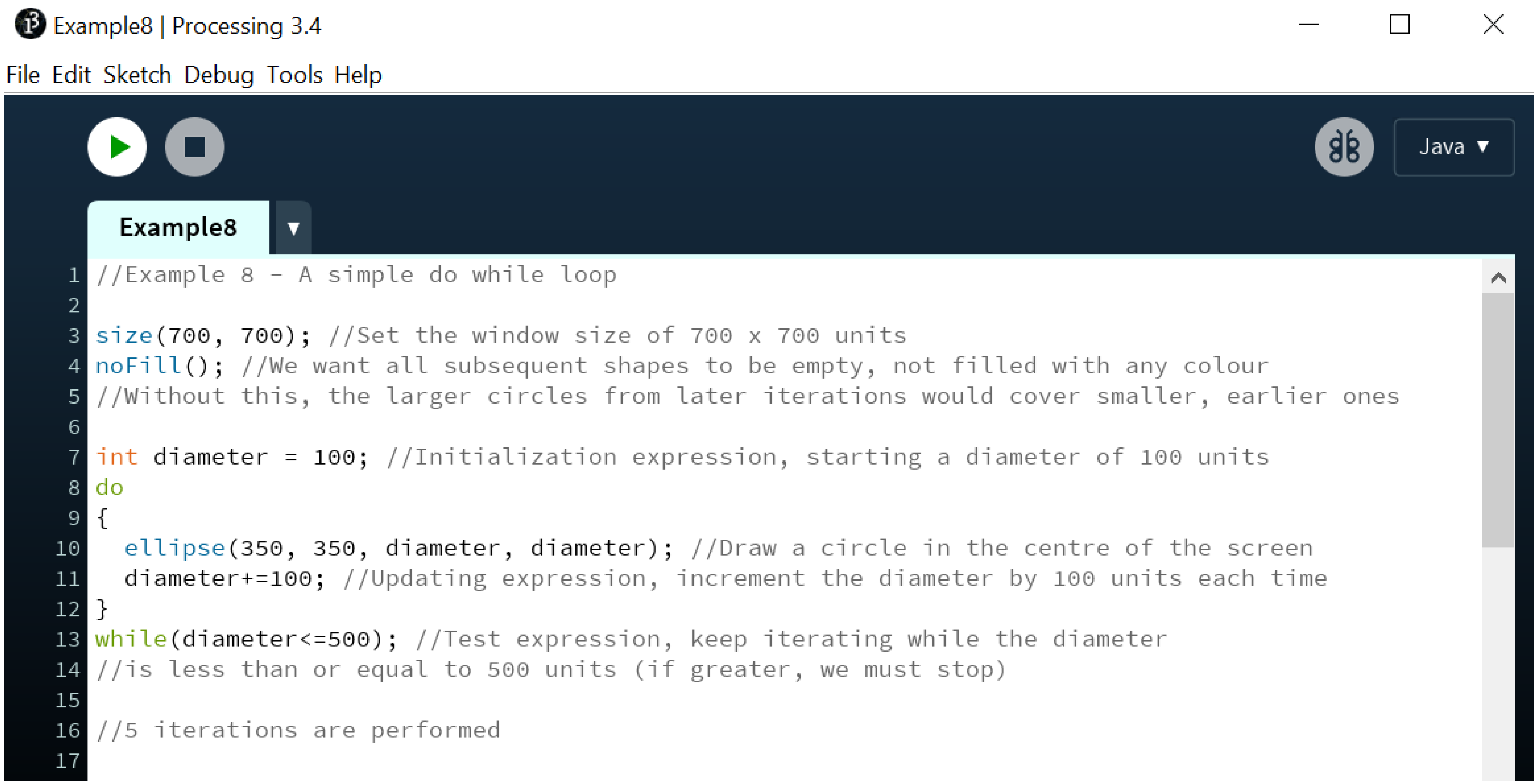
```
//Anatomy of a do while loop

initialization_expression;

do
{
        //Some coding statement(s);
        updating_expression;
}
while(test_expression);

//Only while the test_expression is true do we iterate
//As soon as it becomes false, we exit the loop
//Note that if you only have one statement in the body
//of the loop, curly brackets {} are not necessary
//A do while loop always does one iteration by default
//Notice that a semicolon is required at the end
```

Do while loop – Example 8

- Let’s do an example
- Write a program to draw a series of concentric circles that get larger with each iteration

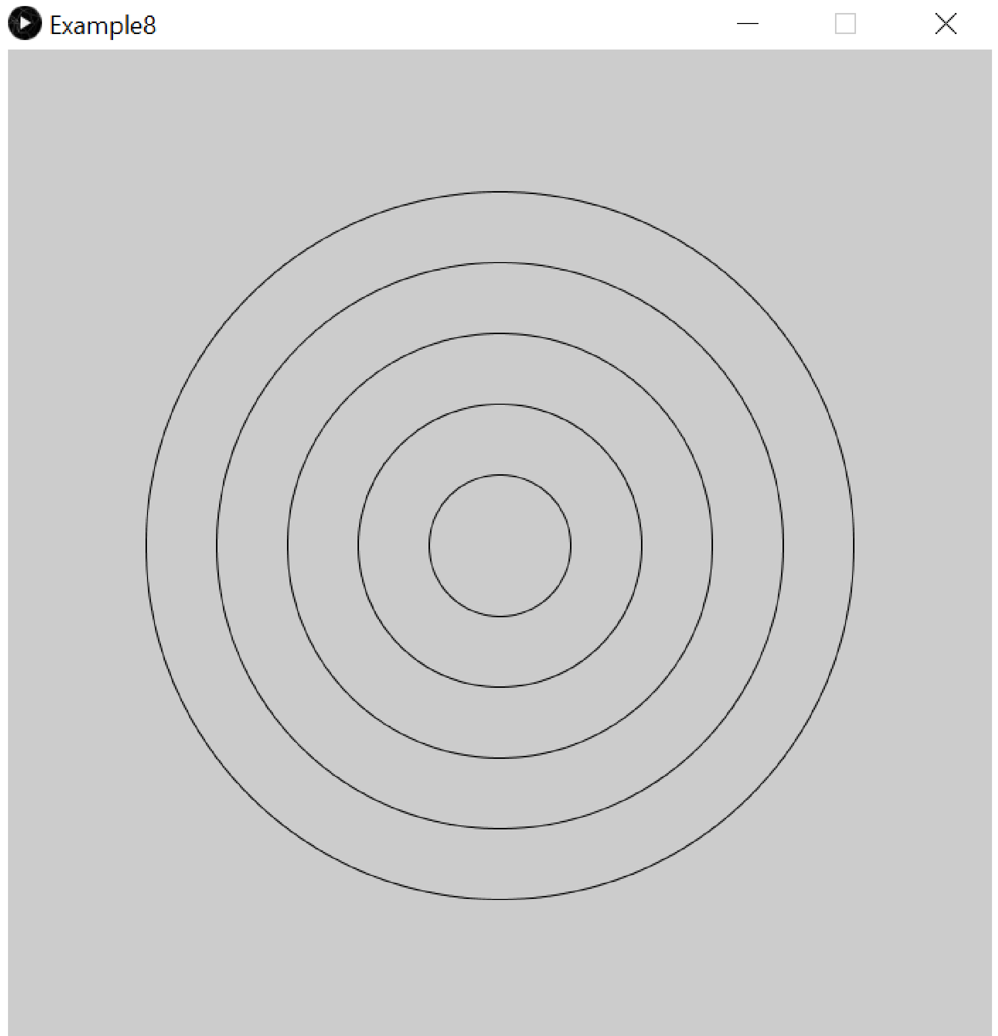
ST. MARY'S HIGH SCHOOL

Do while loop – Example 8



```
//Example 8 - A simple do while loop

size(700, 700); //Set the window size of 700 x 700 units
noFill(); //We want all subsequent shapes to be empty, not filled with any colour
//Without this, the larger circles from later iterations would cover smaller, earlier ones

int diameter = 100; //Initialization expression, starting a diameter of 100 units
do
{
  ellipse(350, 350, diameter, diameter); //Draw a circle in the centre of the screen
  diameter+=100; //Updating expression, increment the diameter by 100 units each time
}
while(diameter<=500); //Test expression, keep iterating while the diameter
//is less than or equal to 500 units (if greater, we must stop)

//5 iterations are performed

```

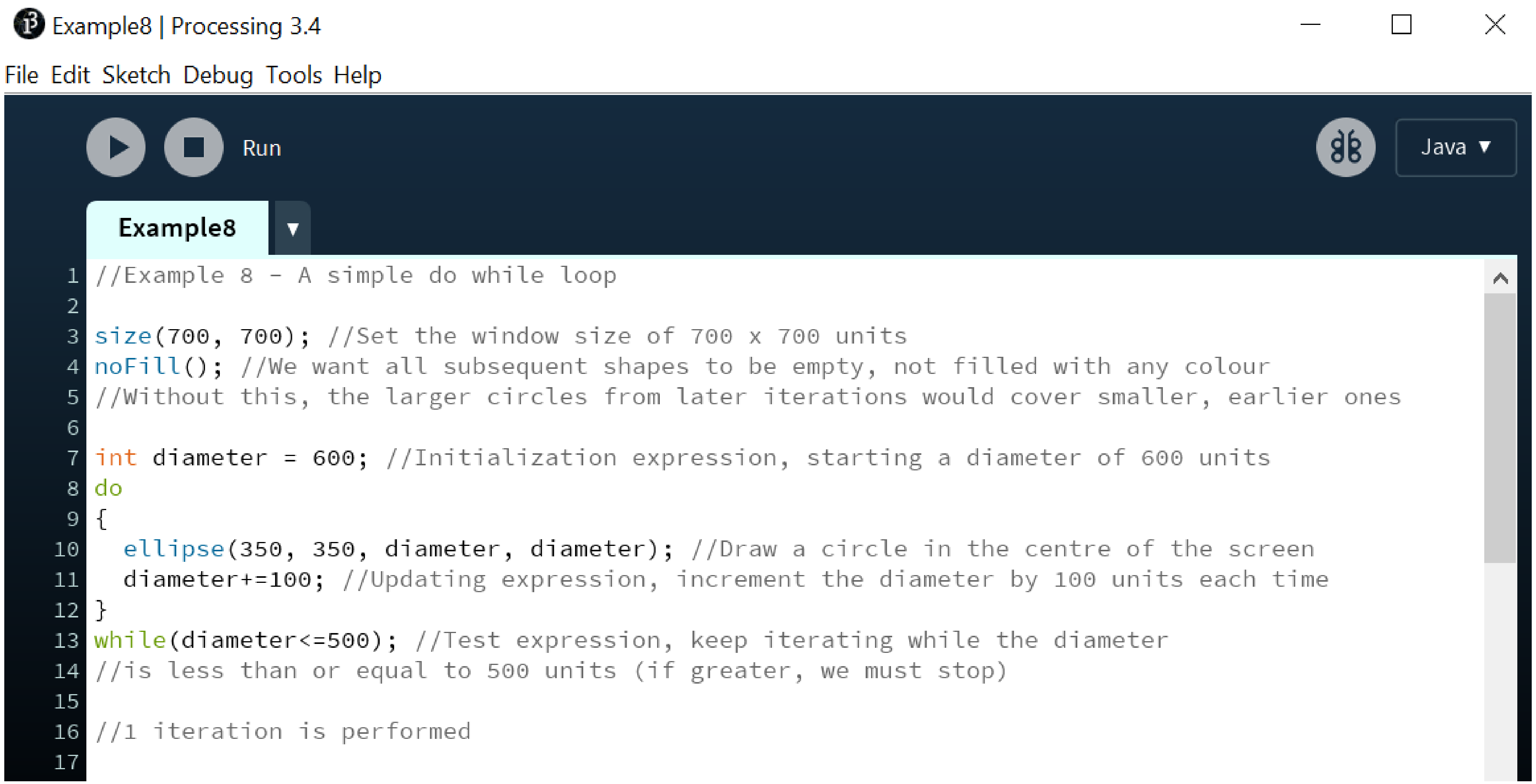
Do while loop – Example 8



Do while loop – Example 8

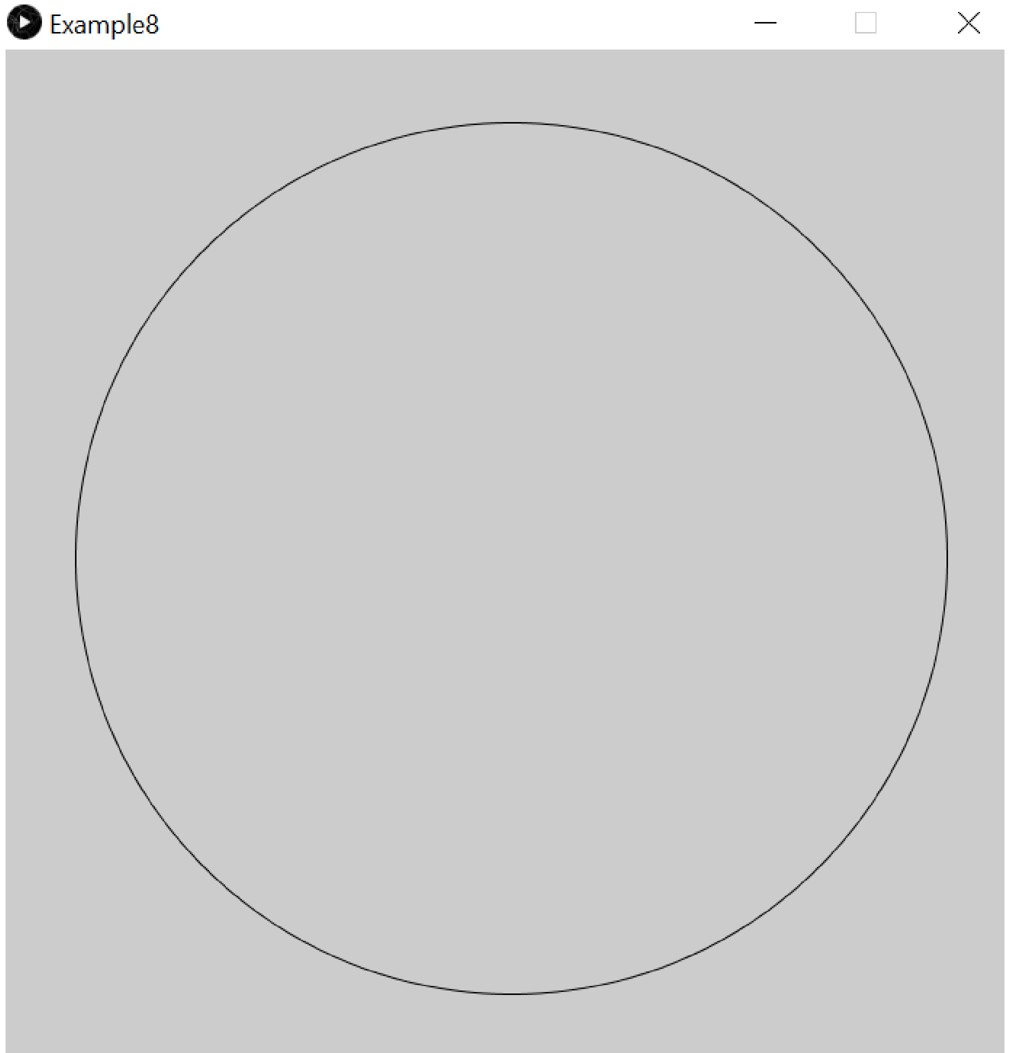
- Now what would happen if the initial value exceeded the condition?
- What if the starting diameter was greater than 500, say 600?

Do while loop – Example 8



```
//Example 8 - A simple do while loop

size(700, 700); //Set the window size of 700 x 700 units
noFill(); //We want all subsequent shapes to be empty, not filled with any colour
//Without this, the larger circles from later iterations would cover smaller, earlier ones

int diameter = 600; //Initialization expression, starting a diameter of 600 units
do
{
  ellipse(350, 350, diameter, diameter); //Draw a circle in the centre of the screen
  diameter+=100; //Updating expression, increment the diameter by 100 units each time
}
while(diameter<=500); //Test expression, keep iterating while the diameter
//is less than or equal to 500 units (if greater, we must stop)

//1 iteration is performed

```

Do while loop – Example 8



Do while loop – Example 8

- Why did it work?
- A do while loop is fundamentally different from while loops and for loops because it is a post-condition loop
- This means that it will always execute the code in the body of the loop once before checking any condition
- For loops and while loops on the other hand are pre-condition loops that first make sure a condition is satisfied before proceeding with the body of the loop

Which type of loop should I use?

- Most of the time, a programming task can be accomplished with any one of the three loops we discussed
- A while loop, a for loop, and a do while loop can basically achieve the same thing
- However, there are cases when it is better to use one over the other
- As mentioned previously, the do while loop is a post-condition loop
- On the other hand, the while and for loops are pre-condition loops
- Depending on the application, one loop may suit your purposes better

When to use a while loop

- In general, a while loop is used for looping until a condition is satisfied and when you are unsure how many iterations need to be performed
- For example, if you have a program that requires a user to guess a mystery number
- You might have a while loop that allows them to keep guessing until they get it right
- We don’t know how many tries this will take but we iterate until they correctly guess the mystery number and the condition is fulfilled

When to use a for loop

- In general, a for loop is used for looping until a condition is satisfied but you know exactly how many iterations are required
- Thinking back to the previous example where a user had to guess an unknown number
- A for loop might be used to allow them to make say 10 guesses
- We don’t care if they get it right or not, the loop performs exactly 10 iterations and that’s it

When to use a do while loop

- In general, a do while loop executes the content of the loop once before checking any conditions
- This might come in handy for particular applications
- For example, consider a software distribution company offering a one month free trial on their product
- Normally, you have to pay first before being issued the software
- Using a do while loop, a user would automatically get access to the software for the first month but after that, the program should make sure they paid first before renewing their subscription
- In the first iteration, it’s free so no need to check for payment, no need to check a condition (but we will start checking after that)

Summary

- In this crash course, we learned about the three main types of loops and when they should be used
- The while loop is a pre-condition loop used when you want to satisfy a condition, but you don’t exactly know how many iterations are required
- The for loop is a pre-condition loop used when you want to satisfy a condition but you know exactly how many iterations need to happen
- The do while loop is post-condition loop which always runs one iteration before checking any test expression