1.3 **Loops**

- while loops
- for loops
- nested loops
- image processing

https://introcs.cs.princeton.edu
Basic building blocks for programming

any program you might want to write

objects

functions               libraries

arrays

conditionals          loops

Math                  text I/O

built-in data types   assignment statements

to infinity and beyond!
1.3 Loops

- while loops
- for loops
- nested loops
- image processing
The *while* loop

**Goal.** Repeat a certain statement (or statements).
- Evaluate a *boolean expression*. If *true*,
  - execute sequence of statements in *code block*
  - repeat
An infinite while loop

Goal. Recreate percussive beat from Queen’s “We Will Rock You.”

public class StompStompClap {
    public static void main(String[] args) {
        while (true) {
            StdAudio.play("stomp.wav");
            StdAudio.play("stomp.wav");
            StdAudio.play("clap.wav");
            StdAudio.play("rest.wav");
        }
    }
}

<table>
<thead>
<tr>
<th>effect</th>
<th>audio file</th>
<th>sound</th>
</tr>
</thead>
<tbody>
<tr>
<td>stomp</td>
<td>stomp.wav</td>
<td></td>
</tr>
<tr>
<td>clap</td>
<td>clap.wav</td>
<td></td>
</tr>
<tr>
<td>silence</td>
<td>rest.wav</td>
<td></td>
</tr>
</tbody>
</table>

<Ctrl–C> to break out of infinite loop
Counting from 1 to $n$

**Goal.** Repeat a ringtone $n$ times.

```
public class Ringtone {
    public static void main(String[] args) {
        String filename = args[0];
        int n = Integer.parseInt(args[1]);

        int i = 1;
        while (i <= n) {
            StdAudio.play(filename);
            i++;
        }
    }
}
```

```
~/cos126/loops> java-introcs Ringtone marimba.wav 1
🔔 [plays marimba ringtone once]

~/cos126/loops> java-introcs Ringtone marimba.wav 3
🔔 [plays marimba ringtone three times]

~/cos126/loops> java-introcs Ringtone sonar.wav 2
🔔 [plays sonar ringtone twice]
```
**Goal.** Repeat a ringtone $n$ times.

**Trace.** Show values of variables at end of each iteration of *while* loop.

```java
public class Ringtone {
    public static void main(String[] args) {
        String filename = args[0];
        int n = Integer.parseInt(args[1]);

        int i = 1;
        while (i <= n) {
            StdAudio.play(filename);
            i++;
        }
    }
}
```

<table>
<thead>
<tr>
<th>filename</th>
<th>n</th>
<th>i</th>
</tr>
</thead>
</table>
| "marimba.wav" | 3  | 1 | ← before loop
| "marimba.wav" | 3  | 2 |
| "marimba.wav" | 3  | 3 |
| "marimba.wav" | 3  | 4 | ← after loop

a trace of variables
(values at end of each loop iteration)
Loops: quiz 1

What does the following program do when \( n \) is 10?

A. Prints 0 to 10.

B. Print powers of 2, from \( 2^0 \) to \( 2^9 \).

C. Print powers of 2, from \( 2^0 \) to \( 2^{10} \).

D. Print powers of 2, from \( 2^0 \) to \( 2^{11} \).

E. Print powers of 2, from \( 2^1 \) to \( 2^{10} \).

```java
public class Mystery {
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        int i = 0;
        int value = 1;

        while (i <= n) {
            System.out.println(value);
            i++;
            value = value * 2;
        }
    }
}
```
### Examples of *while* loops

<table>
<thead>
<tr>
<th>computation</th>
<th>while loop</th>
</tr>
</thead>
</table>
| **print integers from n down to 1**  | `int i = n;`  
`while (i >= 1) {`  
`    System.out.println(i);`  
`    i--;`  
`}`  

  *shorthand for*  
  *i = i - 1*  

<table>
<thead>
<tr>
<th>infinite loop</th>
<th></th>
</tr>
</thead>
</table>
|               | `while (true) {`  
`    StdAudio.play("heartbeat.wav");`  
`}`  

  *curly braces are optional here since only one statement in body of loop (but better style to use curly braces)*  

<table>
<thead>
<tr>
<th>number of decimal digits in positive integer x</th>
<th></th>
</tr>
</thead>
</table>
|                                               | `int digits = 0;`  
`while (x > 0) {`  
`    x = x / 10;`  
`    digits++;`  
`}`  

  *integer division*
LOOPS

- while loops
- for loops
- nested loops
- image processing
A `for` loop (in C)

```c
#include <stdio.h>
int main(void)
{
    int count;
    for (count = 1; count <= 500; count++)
        printf("I will not throw paper airplanes in class.\n");
    return 0;
}
```

Copyright 2004, FoxTrot by Bill Amend
**The for loop**

An alternative repetition structure.

- Perform an **initialization step**.  
  Typically, declaring and initializing the value of a variable
- Evaluate a **boolean expression**. If **true**,  
  - execute sequence of statements in code block  
  - perform an **update step**. Typically, updating the value of a variable  
  - repeat

```latex
\textbf{for} \ (<\textit{init}>; <\textit{boolean expression}>; <\textit{update}>) \{  
  <\textit{statement} 1>  
  <\textit{statement} 2>  
\}
```

*for loop template*
Counting from 1 to $n$

**Goal.** Play a WAV file $n$ times.

```
public class MusicLoop {
    public static void main(String[] args) {
        String filename = args[0];
        int n = Integer.parseInt(args[1]);

        for (int i = 1; i <= n; i++) {
            StdAudio.play(filename);
        }
    }
}
```

```bash
~/cos126/loops> java-introcs MusicLoop heartbeat.wav 1
[plays heartbeat once]
```

```bash
~/cos126/loops> java-introcs MusicLoop heartbeat.wav 9999999
[plays heartbeat repeatedly]
```

```bash
~/cos126/loops> java-introcs MusicLoop AmenBreak.wav 10
[plays The Winstons "Amen Break" drum break 10 times]
```

Identical behavior as Ringtone.java

Among most sampled tracks in music history
### Examples of *for* loops

<table>
<thead>
<tr>
<th>computation</th>
<th>for loop</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>factorial</strong></td>
<td>int product = 1;</td>
</tr>
<tr>
<td>(1 × 2 × 3 × ... × n)</td>
<td>for (int i = 1; i &lt;= n; i++) { product = product * i; }</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
<tr>
<td><strong>print integers</strong></td>
<td>for (int i = n; i &gt;= 1; i--) { System.out.println(i); }</td>
</tr>
<tr>
<td><em>from n down to 1</em></td>
<td>}</td>
</tr>
<tr>
<td><strong>infinite loop</strong></td>
<td>for (;;) { StdAudio.play(&quot;heartbeat.wav&quot;); }</td>
</tr>
<tr>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

- Curly braces are optional since if body consists of only one statement (but better style to include).
- Empty initialization and update (but better style to use *while* loop).
Q. Which value does the following program print when n is 3?

A. 0 1 2 3 2 1 0
B. 0 1 0 2 0 1 0
C. 0 1 0 2 0 1 0 3
D. 0 1 0 2 0 1 0 3 0 1 0 2 0 1 0

```java
public class Ruler {
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);

        String ruler = "0";
        for (int i = 1; i <= n; i++) {
            ruler = ruler + " " + i + " " + ruler;
        }

        System.out.println(ruler);
    }
}
```
While loop vs. for loop

**Fact.** Any *while* loop can be replaced with a *for* loop, and vice versa.

**Q.** Which one should I use?

**A.** Guiding principle: use loop construct that leads to clearer code.

**Rule-of-thumb.** Use a *for* loop when you know the number of iterations ahead of time.

```java
int i = 1;
while (i <= n) {
    StdAudio.play(filename);
    i++;
}
```

*while loop*

```java
for (int i = 1; i <= n; i++) {
    StdAudio.play(filename);
}
```

*equivalent for loop (except i not accessible after loop)*

*code controlling loop localized to one place*
LOOPS

- while loops
- for loops
- nested loops
- image processing
Suppose $m = 10$ and $n = 5$. How many lines of output does the following program produce?

A. 10  
B. 15  
C. 50  
D. 55  
E. 60
Gambler’s ruin problem

**Gambler’s ruin.** A gambler starts with $\text{stake}$ and places $\$1$ fair bets.

- Outcome 1 (win): gambler reaches $\text{goal}$.
- Outcome 2 (loss): gambler goes broke with $\$0$.

**Q1.** What are the chances of winning?

**Q2.** How many bets until win or loss?

**One approach.** [Monte Carlo simulation]

- Perform one experiment using simulated coin flips.
- Repeat experiment many times and collect statistics.
Gambler’s ruin problem: one experiment

**Warmup.** Simulate one experiment.

```java
public class GamblerWarmup {
    public static void main(String[] args) {
        int stake = Integer.parseInt(args[0]);
        int goal = Integer.parseInt(args[1]);

        int cash = stake;
        System.out.println(cash);
        while ((cash > 0) && (cash < goal)) {
            if (Math.random() < 0.5) cash++;
            else cash--;
            System.out.println(cash);
        }
    }
}
```

```bash
~/loops> java GamblerWarmup 4 10
4
5
4
3
4
3
1
2
1
0
```
Monte Carlo simulation of gambler’s ruin problem

```java
public class Gambler {
    public static void main(String[] args) {
        int stake = Integer.parseInt(args[0]);
        int goal = Integer.parseInt(args[1]);
        int trials = Integer.parseInt(args[2]);

        int wins = 0;
        for (int t = 1; t <= trials; t++) {
            int cash = stake;
            while ((cash > 0) && (cash < goal)) {
                if (Math.random() < 0.5) cash++;
                else cash--;
            }
            if (cash == goal) wins++;
        }
        System.out.println(wins + " wins of " + trials);
    }
}
```

~/$cos26/loops> java Gambler 5 25 1000
191 wins of 1000

~/$cos26/loops> java Gambler 5 25 1000
183 wins of 1000
Digression: simulation vs. mathematical analysis

Facts. [known via probability theory]

- Probability of winning = \( \text{stake} \div \text{goal} \).
- Expected number of bets = \( \text{stake} \times (\text{goal} - \text{stake}) \).

Ex. [\( \text{stake} = 500, \text{goal} = 2500 \)]

- 20% chance of winning.
- Expect to make 1 million bets per experiment.

Remarks.

- For gambler’s ruin, mathematical analysis is well known.
- Computer simulation agrees with math.
- For more complicated variants, math may be beyond reach.
- Monte Carlo simulations widely used in STEM.
Integer factorization

**Goal.** Given a positive integer $n$, find its prime factorization.

\[
98 = 2 \times 7 \times 7 \\
3,757,208 = 2 \times 2 \times 2 \times 7 \times 13 \times 13 \times 397 \\
11,111,111,111,111,111 = 2,071,723 \times 5,363,222,357
\]

Grade-school factoring algorithm.

**FACTOR**($n$)

Consider each potential divisor $d$ between 2 and $n$:

- **while** $d$ is a divisor of $n$:
  - **print** $d$
  - $n \leftarrow n / d$

Integer factorization

public class Factors {
    public static void main(String[] args) {
        long n = Long.parseLong(args[0]);
        for (long d = 2; d <= n; d++) {
            while (n % d == 0) {
                System.out.print(d + " ");
                n = n / d;
            }
        }
        System.out.println();
    }
}

Remark 1. Uses long instead of int to support integers between $-2^{63}$ and $2^{63} - 1$.
Remark 2. Way too slow to break cryptography.

can be sped up substantially by stopping when $d > \sqrt{n}$
(but still way too slow)

~/cos126/loops> java Factors 98
  2 7 7

~/cos126/loops> java Factors 3757208
  2 2 2 7 13 13 397

~/cos126/loops> java Factors 97
  97

~/cos126/loops> java Factors 1111111111111111
2071723 536322235

takes a few seconds
LOOPS

- while loops
- for loops
- nested loops
- image processing
A picture is a width–by–height grid of pixels; each pixel has a color.

Image-processing conventions.

- Pixel \((i, j)\) means column \(i\) and row \(j\).
- Pixel \((0, 0)\) is upper–left.

warning: different conventions from matrices and Cartesian coordinates
**RGB color model**

**Color** is a sensation in the eye from electromagnetic radiation.

**RGB color model.** Popular format for representing color on digital displays.
- Color is composed of red, green, and blue components.
- Each color component is an integer between 0 to 255.

<table>
<thead>
<tr>
<th>name</th>
<th>red</th>
<th>green</th>
<th>blue</th>
<th>color</th>
</tr>
</thead>
<tbody>
<tr>
<td>red</td>
<td>255</td>
<td>0</td>
<td>0</td>
<td>red</td>
</tr>
<tr>
<td>green</td>
<td>0</td>
<td>255</td>
<td>0</td>
<td>green</td>
</tr>
<tr>
<td>blue</td>
<td>0</td>
<td>0</td>
<td>255</td>
<td>blue</td>
</tr>
<tr>
<td>black</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>black</td>
</tr>
<tr>
<td>white</td>
<td>255</td>
<td>255</td>
<td>255</td>
<td>white</td>
</tr>
<tr>
<td>yellow</td>
<td>255</td>
<td>255</td>
<td>0</td>
<td>yellow</td>
</tr>
<tr>
<td>magenta</td>
<td>255</td>
<td>0</td>
<td>255</td>
<td>magenta</td>
</tr>
<tr>
<td>cyan</td>
<td>0</td>
<td>255</td>
<td>255</td>
<td>cyan</td>
</tr>
<tr>
<td>book blue</td>
<td>0</td>
<td>64</td>
<td>128</td>
<td>book blue</td>
</tr>
</tbody>
</table>

![RGB color diagram]
Grayscale

**Goal.** Convert color image to grayscale.

- RGB color is a shade of gray when \( R = G = B \).
- To convert RGB color to grayscale, use **luminance** for \( R \), \( G \), and \( B \) values:

\[
Y = 0.299 R + 0.587 G + 0.114 B
\]

<table>
<thead>
<tr>
<th>name</th>
<th>red</th>
<th>green</th>
<th>blue</th>
<th>color</th>
<th>lum</th>
<th>gray</th>
</tr>
</thead>
<tbody>
<tr>
<td>red</td>
<td>255</td>
<td>0</td>
<td>0</td>
<td>red</td>
<td>76</td>
<td></td>
</tr>
<tr>
<td>green</td>
<td>0</td>
<td>255</td>
<td>0</td>
<td>green</td>
<td>150</td>
<td></td>
</tr>
<tr>
<td>blue</td>
<td>0</td>
<td>0</td>
<td>255</td>
<td>blue</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>black</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>black</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>white</td>
<td>255</td>
<td>255</td>
<td>255</td>
<td>white</td>
<td>255</td>
<td></td>
</tr>
<tr>
<td>yellow</td>
<td>255</td>
<td>255</td>
<td>0</td>
<td>yellow</td>
<td>226</td>
<td></td>
</tr>
<tr>
<td>magenta</td>
<td>255</td>
<td>0</td>
<td>255</td>
<td>magenta</td>
<td>105</td>
<td></td>
</tr>
<tr>
<td>cyan</td>
<td>0</td>
<td>255</td>
<td>255</td>
<td>cyan</td>
<td>179</td>
<td></td>
</tr>
<tr>
<td>book blue</td>
<td>0</td>
<td>64</td>
<td>128</td>
<td>book blue</td>
<td>52</td>
<td></td>
</tr>
</tbody>
</table>

**fundamental operation in computer graphics and vision**
**StdPicture.** Our library for manipulating images.

```java
public class StdPicture {

  static void read(String filename)  // initialize picture from filename

  static void save(String filename)  // save picture to filename

  static int width()  // width of picture

  static int height()  // height of picture

  static int getRed(int col, int row)  // red component of pixel (col, row)

  static int getGreen(int col, int row)  // green component of pixel (col, row)

  static int getBlue(int col, int row)  // blue component of pixel (col, row)

  static void setRGB(int col, int row, int r, int g, int b)  // set color of pixel (col, row) to (r, g, b)

  ...

  ...

  supported file formats:
  JPEG, PNG, GIF, TIFF, BMP

available with javac-introcs
and java-introcs commands
```

**Available with javac-introcs and java-introcs commands**
public class Grayscale {
    public static void main(String[] args) {
        String filename = args[0];
        StdPicture.read(filename);
        int width = StdPicture.width();
        int height = StdPicture.height();

        for (int col = 0; col < width; col++) {
            for (int row = 0; row < height; row++) {
                int r = StdPicture.getRed(col, row);
                int g = StdPicture.getGreen(col, row);
                int b = StdPicture.getBlue(col, row);
                int y = (int) (Math.round(0.299*r + 0.587*g + 0.114*b));
                StdPicture.setRGB(col, row, y, y, y);
            }
        }

        StdPicture.show();
    }
}
Image processing: color image filters

- original
- grayscale
- sepia
- duotone
- brighter
- darker
- RGB layers
- negative
Image processing: shape masks

original

rounded rectangle

oval

heart

puzzle piece

tiger
Flip an image horizontally

**Goal.** Flip an image horizontally, like looking into a mirror. on Zoom, Instagram, TikTok, …

https://www.wikihow.com/Inverted-Filter
Deja Vu challenge

Disclaimer. COS 126 is not liable for damage to self-esteem.
Flip an image horizontally: demo

**Goal.** Flip an image horizontally, like looking into a mirror.

**Algorithm.** For each pixel \((col, row)\), swap with pixel \((width - col - 1, row)\).

![Original image and flipped image](image-url)
Flip an image horizontally: implementation

**Goal.** Flip an image horizontally, like looking into a mirror.

**Algorithm.** For each pixel \((col, row)\), swap with pixel \((width - col - 1, row)\).

```java
for (int col = 0; col < width / 2; col++) {
    for (int row = 0; row < height; row++) {
        int r1 = StdPicture.getRed(col, row);
        int g1 = StdPicture.getGreen(col, row);
        int b1 = StdPicture.getBlue(col, row);
        int r2 = StdPicture.getRed(width - col - 1, row);
        int g2 = StdPicture.getGreen(width - col - 1, row);
        int b2 = StdPicture.getBlue(width - col - 1, row);
        StdPicture.setRGB(col, row, r2, g2, b2);
        StdPicture.setRGB(width - col - 1, row, r1, g1, b1);
    }
}
StdPicture.show();
```
What image does the following code fragment produce?

A. Original image.
B. Horizontal flip.
C. Vertical flip.

```java
for (int row = 0; row < height; row++) {
    for (int col = 0; col < width / 2; col++) {
        int r1 = StdPicture.getRed(col, row);
        int g1 = StdPicture.getGreen(col, row);
        int b1 = StdPicture.getBlue(col, row);
        int r2 = StdPicture.getRed(width - col - 1, row);
        int g2 = StdPicture.getGreen(width - col - 1, row);
        int b2 = StdPicture.getBlue(width - col - 1, row);
        StdPicture.setRGB(col, row, r2, g2, b2);
        StdPicture.setRGB(width - col - 1, row, r1, g1, b1);
    }
}
StdPicture.show();
```
Summary

**Iteration.** Use *while* and *for* loops to repeat code in a program.

**Nested iteration.** Body of loop contains another loop.

**Image processing.** An image is a 2D grid of pixels, each of which has a color.
<table>
<thead>
<tr>
<th>media</th>
<th>source</th>
<th>license</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buzz Lightyear</td>
<td>alphacoders.com</td>
<td></td>
</tr>
<tr>
<td>Rainbow Infinity</td>
<td>Adobe Stock</td>
<td>education license</td>
</tr>
<tr>
<td>Stomp–Stomp–Clap</td>
<td>Queen</td>
<td></td>
</tr>
<tr>
<td>Ringtone Icon</td>
<td>Wikimedia</td>
<td>public domain</td>
</tr>
<tr>
<td>Marimba Ringtone</td>
<td>Apple iPhone</td>
<td></td>
</tr>
<tr>
<td>Sonar Ringtone</td>
<td>Apple iPhone</td>
<td></td>
</tr>
<tr>
<td>Coin Toss</td>
<td>clipground.com</td>
<td>CC BY 4.0</td>
</tr>
<tr>
<td>Paper Airplanes</td>
<td>FoxTrot by Bill Amend</td>
<td></td>
</tr>
<tr>
<td>Heartbeat</td>
<td>freesound.org</td>
<td>CC BY 4.0</td>
</tr>
<tr>
<td>Amen Break</td>
<td>The Winstons</td>
<td></td>
</tr>
<tr>
<td>media</td>
<td>source</td>
<td>license</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>Russian Nesting Dolls</td>
<td>Adobe Stock</td>
<td>education license</td>
</tr>
<tr>
<td>Cryptography Icon</td>
<td>Adobe Stock</td>
<td>education license</td>
</tr>
<tr>
<td>Image Processing Icon</td>
<td>Adobe Stock</td>
<td>education license</td>
</tr>
<tr>
<td>Mandrill</td>
<td>USC SIPI Image Database</td>
<td></td>
</tr>
<tr>
<td>Johnson Arch</td>
<td>Danielle Alio Capparella</td>
<td>by photographer</td>
</tr>
<tr>
<td>RGB Color Model</td>
<td>Wikimedia</td>
<td>Kopimi</td>
</tr>
<tr>
<td>LGBTQ+ Eye</td>
<td>Wikimedia</td>
<td>CC BY 2.0</td>
</tr>
<tr>
<td>Inverted Filter</td>
<td>WikiHow</td>
<td></td>
</tr>
</tbody>
</table>