2. Conditionals and loops
2. Conditionals & Loops

- **Conditionals:** the `if` statement
- **Loops:** the `while` statement
- An alternative: the `for` loop
- Nesting
- Debugging
Context: basic building blocks for programming

any program you might want to write

objects
functions and modules
arrays
graphics, sound, and image I/O

conditionals and loops

Math
text I/O

primitive data types
assignment statements

Previous lecture: equivalent to a calculator

This lecture: to infinity and beyond!
**Conditionals and Loops**

**Control flow**
- The sequence of statements that are actually executed in a program.
- *Conditionals and loops* enable us to choreograph control flow.

---

**Diagram:**
- **Statement 1**
- **Statement 2**
- **Statement 3**
- **Statement 4**

**Flowchart:**
- **Boolean 1**
- **Boolean 2**

---

**Diagram labels:**
- straight-line control flow [previous lecture]
- control flow with conditionals and a loop [this lecture]
The **if** statement

Execute certain statements depending on the values of certain variables.

- Evaluate a boolean expression.
- If **true**, execute a statement.
- The **else** option: If **false**, execute a different statement.

**Example:**

```java
if (x < 0) x = -x;
```

Replaces `x` with the absolute value of `x`

**Example:**

```java
if (x > y) max = x;
else max = y;
```

Computes the maximum of `x` and `y`
Example of if statement use: simulate a coin flip

```java
public class Flip {
    public static void main(String[] args) {
        if (Math.random() < 0.5)
            System.out.println("Heads");
        else
            System.out.println("Tails");
    }
}
```

% java Flip
Heads
% java Flip
Heads
% java Flip
Tails
% java Flip
Heads
**Example of if statement use: 2-sort**

**Q.** What does this program do?

```java
public class TwoSort {
    public static void main(String[] args) {
        int a = Integer.parseInt(args[0]);
        int b = Integer.parseInt(args[1]);
        if (b < a) {
            int t = a;  // alternatives for if and else can be a sequence of statements, enclosed in braces
            a = b;
            b = t;
        }
        System.out.println(a);
        System.out.println(b);
    }
}
```

**A.** Reads two integers from the command line, then prints them out in numerical order.

```
% java TwoSort 1234 99
99
1234

% java TwoSort 99 1234
99
1234
```
Pop quiz on if statements

Q. Add code to this program that puts a, b, and c in numerical order.

```java
public class ThreeSort {
    public static void main(String[] args) {
        int a = Integer.parseInt(args[0]);
        int b = Integer.parseInt(args[1]);
        int c = Integer.parseInt(args[2]);

        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
    }
}
```

```java
% java ThreeSort 1234 99 1
1
99
1234

% java ThreeSort 99 1 1234
1
99
1234
```
Pop quiz on if statements

Q. Add code to this program that puts a, b, and c in numerical order.

```java
public class ThreeSort {
    public static void main(String[] args) {
        int a = Integer.parseInt(args[0]);
        int b = Integer.parseInt(args[1]);
        int c = Integer.parseInt(args[2]);
        if (b < a) {
            int t = a;
            a = b;
            b = t;
        }
        if (c < a) {
            int t = a;
            a = c;
            c = t;
        }
        if (c < b) {
            int t = b;
            b = c;
            c = t;
        }
        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
    }
}
```

A. % java ThreeSort 1234 99 1
1
99
1234

% java ThreeSort 99 1 1234
1
99
1234
Example of if statement use: error checks

```java
public class IntOps {
    public static void main(String[] args) {
        int a = Integer.parseInt(args[0]);
        int b = Integer.parseInt(args[1]);
        int sum = a + b;
        int prod = a * b;
        System.out.println(a + " + " + b + " = " + sum);
        System.out.println(a + " * " + b + " = " + prod);
        if (b == 0) System.out.println("Division by zero");
        else System.out.println(a + " / " + b + " = " + a / b);
        if (b == 0) System.out.println("Division by zero");
        else System.out.println(a + " % " + b + " = " + a % b);
    }
}
```

Good programming practice. Use conditionals to check for and avoid runtime errors.
Image sources

http://commons.wikimedia.org/wiki/File:Calculator_casio.jpg
http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2789164/#!po=30.0000 [181e306f1.jpg]
2. Conditionals & Loops

• Conditionals: the *if* statement
• **Loops: the** *while* statement
• An alternative: the *for* loop
• Nesting
• Debugging
The **while loop**

Execute certain statements repeatedly until certain conditions are met.

- Evaluate a boolean expression.
- If true, execute a sequence of statements.
- Repeat.

**Example:**
```java
int i = 0;
int v = 1;
while (i <= n) {
    System.out.println(v);
    i = i + 1;
    v = 2 * v;
}
```

Prints the powers of two from $2^0$ to $2^n$.

[stay tuned for a trace]
Example of while loop use: print powers of two

A trace is a table of variable values after each statement.

### Example Class

```java
public class PowersOfTwo {
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        int i = 0;
        int v = 1;
        while (i <= n) {
            System.out.println(v);
            i = i + 1;
            v = 2 * v;
        }
    }
}
```

Prints the powers of two from $2^0$ to $2^n$.

### Trace Table

<table>
<thead>
<tr>
<th>i</th>
<th>v</th>
<th>i &lt;= n</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>true</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>true</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>true</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>true</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>true</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
<td>true</td>
</tr>
<tr>
<td>6</td>
<td>64</td>
<td>true</td>
</tr>
<tr>
<td>7</td>
<td>128</td>
<td>false</td>
</tr>
</tbody>
</table>

% java PowersOfTwo 6

1
2
4
8
16
32
64
Pop quiz on while loops

Q. Anything wrong with the following code?

```java
public class PQwhile {
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        int i = 0;
        int v = 1;
        while (i <= n) {
            System.out.println(v);
            i = i + 1;
            v = 2 * v;
        }
    }
}
```
Pop quiz on while loops

Q. Anything wrong with the following code?

```java
public class PQwhile {
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        int i = 0;
        int v = 1;
        while (i <= n) {
            System.out.println(v);
            i = i + 1;
            v = 2 * v;
        }
    }
}
```

A. Yes! Needs braces.

Q. What does it do (without the braces)?

A. Goes into an infinite loop.

% java PQwhile 6
1
1
1
1
1
1
1
1
1
1
1
1

challenge: figure out how to stop it on your computer
Example of **while** loop use: implement `Math.sqrt()`

**Goal.** Implement square root function.

Newton-Raphson method to compute \( \sqrt{c} \)

- Initialize \( t_0 = c \).
- Repeat until \( t_i = c/t_i \) (up to desired precision):
  - Set \( t_{i+1} \) to be the average of \( t_i \) and \( c/t_i \).

\[
\begin{array}{|c|c|c|c|}
\hline
i & t_i & 2/t_i & \text{average} \\
\hline
0 & 2 & 1 & 1.5 \\
1 & 1.5 & 1.3333333 & 1.4166667 \\
2 & 1.4166667 & 1.4117647 & 1.4142157 \\
3 & 1.4142157 & 1.4142114 & 1.4142136 \\
4 & 1.4142136 & 1.4142136 & 1.4142136 \\
\hline
\end{array}
\]

computing the square root of 2 to seven places

Many students actually look forward to Mr. Atwadder’s math tests.

\[
\text{The square root of 9 is 3.} \\
\text{A) True.} \\
\text{B) False.} \\
\text{C) Who cares?}
\]
Example of while loop use: implement Math.sqrt()

Newton-Raphson method to compute $\sqrt{c}$

- Initialize $t_0 = c$.
- Repeat until $t_i = \frac{c}{t_i}$ (up to desired precision):
  
  Set $t_{i+1}$ to be the average of $t_i$ and $\frac{c}{t_i}$.

```java
public class Sqrt {
    public static void main(String[] args) {
        double EPS = 1E-15; // error tolerance (15 places)
        double c = Double.parseDouble(args[0]);
        double t = c;
        while (Math.abs(t - c/t) > t*EPS) {
            t = (c/t + t) / 2.0;
        }
        System.out.println(t);
    }
}
```

% java Sqrt 60481729.0 7777.0
% java Sqrt 2.0 1.414213562373095

Isaac Newton
1642-1727

Scientists studied computation well before the onset of the computer.
Newton-Raphson method

Explanation (some math omitted)

- Goal: find root of function $f(x)$ (value of $x$ for which $f(x) = 0$).
- Start with estimate $t_0$.
- Draw line tangent to curve at $x = t_i$.
- Set $t_{i+1}$ to be $x$-coordinate where line hits $x$-axis.
- Repeat until desired precision.

Root: $f(x) = 0$

Use $f(x) = x^2 - c$ for $\sqrt{c}$

$y = f(x)$

$t_{i+3}$ $t_{i+2}$ $t_{i+1}$ $t_i$
Image sources

http://www.sciencecartoonsplus.com
2. Conditionals & Loops

- Conditionals: the if statement
- Loops: the while statement
- An alternative: the for loop
- Nesting
- Debugging
The **for** loop

An alternative repetition structure.

- Evaluate an *initialization statement*.
- Evaluate a *boolean expression*.
- If true, execute a *sequence of statements*, then execute an *increment statement*.
- Repeat.

**Example:**

```java
int v = 1;
for (int i = 0; i <= n; i++) {
    System.out.println( i + " " + v );
    v = 2*v;
}
```

Prints the powers of two from $2^0$ to $2^n$

Why? Can provide code that is more compact and understandable.

**Every for loop has an equivalent while loop:**

```java
int v = 1;
int i = 0;
while (i <= n) {
    System.out.println( i + " " + v );
    v = 2*v;
    i++;
}
```
Examples of for loop use

```
int sum = 0;
for (int i = 1; i <= N; i++)
    sum += i;
System.out.println(sum);
```

 Computes sum (1 + 2 + 3 + ... + N)

```
long product = 1;
for (int i = 1; i <= N; i++)
    product *= i;
System.out.println(product);
```

 Computes N! = 1 * 2 * 3 * ... * N

```
for (int k = 0; k <= N; k++)
    System.out.println(k + " " + 2*Math.PI*k/N);
```

 Prints a table of function values

```
int v = 1;
while (v <= N/2)
    v = 2*v;
System.out.println(v);
```

 Prints largest power of 2 less than or equal to N
Example of for loop use: subdivisions of a ruler

Create subdivisions of a ruler to $1/N$ inches.

- Initialize `ruler` to one space.
- For each value $i$ from 1 to $N$:
  sandwich $i$ between two copies of `ruler`.

```java
public class Ruler {
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        String ruler = " ";
        for (int i = 1; i <= N; i++)
            ruler = ruler + i + ruler;
        System.out.println(ruler);
    }
}
```

Note: Small program can produce huge amount of output.
Pop quiz on for loops

Q. What does the following program print?

```java
public class PQfor {
    public static void main(String[] args) {
        int f = 0, g = 1;
        for (int i = 0; i <= 10; i++) {
            System.out.println(f);
            f = f + g;
            g = f - g;
        }
    }
}
```
Pop quiz on for loops

Q. What does the following program print?

```java
public class PQfor {
    public static void main(String[] args) {
        int f = 0, g = 1;
        for (int i = 0; i <= 10; i++) {
            System.out.println(f);
            f = f + g;
            g = f - g;
        }
    }
}
```

Beginning-of-loop trace

<table>
<thead>
<tr>
<th>i</th>
<th>f</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>13</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>21</td>
<td>13</td>
</tr>
<tr>
<td>9</td>
<td>34</td>
<td>21</td>
</tr>
<tr>
<td>10</td>
<td>55</td>
<td>34</td>
</tr>
</tbody>
</table>

A. values printed
2. Conditionals & Loops

- Conditionals: the if statement
- Loops: the while statement
- An alternative: the for loop
- Nesting
- Debugging
Nesting conditionals and loops

Nesting

• Any “statement” within a conditional or loop may itself be a conditional or a loop statement.
• Enables complex control flows.
• Adds to challenge of debugging.

Example:

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

[ Stay tuned for an explanation of this code. ]
Example of nesting conditionals: Tax rate calculation

**Goal.** Given income, calculate proper tax rate.

<table>
<thead>
<tr>
<th>income</th>
<th>rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 – $47,450</td>
<td>22%</td>
</tr>
<tr>
<td>$47,450 – $114,649</td>
<td>25%</td>
</tr>
<tr>
<td>$114,650 – $174,699</td>
<td>28%</td>
</tr>
<tr>
<td>$174,700 – $311,949</td>
<td>33%</td>
</tr>
<tr>
<td>$311,950 +</td>
<td>35%</td>
</tr>
</tbody>
</table>

```java
if (income < 47450) rate = 0.22;
else {
    if (income < 114650) rate = 0.25;
    else {
        if (income < 174700) rate = 0.28;
        else {
            if (income < 311950) rate = 0.33;
            else               rate = 0.35;
        }
    }
}
```
Pop quiz on nested if statements

Q. Anything wrong with the following code?

```java
class PQif {
    public static void main(String[] args) {
        double income = Double.parseDouble(args[0]);
        double rate = 0.35;
        if (income < 47450) rate = 0.22;
        if (income < 114650) rate = 0.25;
        if (income < 174700) rate = 0.28;
        if (income < 311950) rate = 0.33;
        System.out.println(rate);
    }
}
```
Pop quiz on nested if statements

Q. Anything wrong with the following code?

```java
public class PQif {
    public static void main(String[] args) {
        double income = Double.parseDouble(args[0]);
        double rate = 0.35;
        if (income < 47450) rate = 0.22;
        if (income < 114650) rate = 0.25;
        if (income < 174700) rate = 0.28;
        if (income < 311950) rate = 0.33;
        System.out.println(rate);
    }
}
```

A. Yes! Need else clauses. Without them, code is equivalent to:

```java
if (income < 311950) rate = 0.33;
else if (income < 174700) rate = 0.28;
else if (income < 114650) rate = 0.25;
else if (income < 47450) rate = 0.22;
else rate = 0.35;
```

Note. Braces are not needed in this case, but BE CAREFUL when nesting if-else statements because of potential ambiguity (see Q&A p. 75).
A gambler starts with $stake and places $1 fair bets.
- Outcome 1 (loss): Gambler goes broke with $0.
- Outcome 2 (win): Gambler reaches $goal.

Q. What are the chances of winning?
Q. How many bets until win or loss?

One approach: Monte Carlo simulation.
- Use a simulated coin flip.
- Repeat and compute statistics.
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 = 0; t < trials; t++) {
            int cash = stake;
            while (cash > 0 && cash < goal) {
                if (Math.random() < 0.5) cash++;
                else cash--;
            }
            if (t == goal) wins++;
        }
        System.out.println(wins + " wins of " + trials);
    }
}

Example of nesting conditionals and loops: Simulate gambler's ruin

Gambler's ruin simulation

• Get command-line arguments.
• Run all the experiments.
• Run one experiment.
• Make one bet.
• If goal met, count the win.
• Print #wins and # trials.

% java Gambler 5 25 1000
191 wins of 1000
**Digression: simulation and analysis**

**Facts (known via mathematical analysis for centuries)**
- Probability of winning = \( \text{stake} \div \text{goal} \).
- Expected number of bets = \( \text{stake} \times \text{desired gain} \).

**Example**
- 20% chance of turning $500 into $2500. \( \frac{500}{2500} = 20\% \)
- Expect to make 1 *million* $1 bets. \( 500 \times (2500 - 500) = 1,000,000 \)

**Remarks**
- Computer simulation can help validate mathematical analysis.
- For this problem, mathematical analysis is simpler (if you know the math).
- For more complicated variants, computer simulation may be the *best* plan of attack.
Image sources

2. Conditionals & Loops

• Conditionals: the if statement
• Loops: the while statement
• An alternative: the for loop
• Nesting
• Debugging
Debugging

is 99% of program development in any programming language, *even for experts.*

**Bug:** A mistake in a program.  
**Debugging:** The process of eliminating bugs.

“As soon as we started programming, we found out to our surprise that it wasn’t as easy to get programs right as we had thought. I can remember the exact instant when I realized that a large part of my life from then on was going to be spent in finding mistakes in my own programs.”

– Maurice Wilkes

**Impossible ideal:** "Please compile, execute, and debug my program." Why is this impossible? Stay tuned.

**Bottom line:** Programming is primarily a *process* of finding and fixing mistakes.
Debugging is challenging because conditionals and loops *dramatically increase* the number of possible outcomes.

<table>
<thead>
<tr>
<th>program structure</th>
<th>no loops</th>
<th>$n$ conditionals</th>
<th>1 loop</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of possible execution sequences</td>
<td>1</td>
<td>$2^n$</td>
<td>no limit</td>
</tr>
</tbody>
</table>

Most programs contain *numerous* conditionals and loops, with nesting.

**Good news.** Conditionals and loops provide structure that helps us understand our programs.

Old and low-level languages have a *goto* statement that provides arbitrary structure. Eliminating *gos* was controversial until Edsgar Dijkstra published the famous note "*Goto considered harmful*" in 1968.

> "The quality of programmers is a decreasing function of the number of goto statements in the programs they produce."

— Edsgar Dijkstra
Debugging a program: a running example

**Problem:** Factor a large integer \( n \).

**Application:** Cryptography.

**Surprising fact:** Security of internet commerce depends on difficulty of factoring large integers.

**Method**

- Consider each integer \( i \) less than \( n \)

- While \( i \) divides \( n \) evenly
  - Print \( i \) (it is a factor of \( n \)).
  - Replace \( n \) with \( n/i \).

**Rationale:**

1. Any factor of \( n/i \) is a factor of \( n \).
2. \( i \) may be a factor of \( n/i \).

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

**This program has bugs!**
Debugging a program: syntax errors

Is your program a legal Java program?
• Java compiler can help you find out.
• Find the first compiler error (if any).
• Repeat.
• Result: An executable Factors.class file

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

% javac Factors.java
Factors.java:5: ';' expected
    long n = Long.parseLong(args[0]);
         ^

...% javac Factors.java
Factors.java:6: cannot find symbol
    for (int i = 0; i < n; i++) {
                 ^

...% javac Factors.java
Factors.java:6: cannot find symbol
    for (int i = 0; i < n; i++) {
                 ^

This legal program still has bugs!
Does your legal Java program do what you want it to do?

- You need to run it to find out.
- Find the first runtime error (if any).
- Fix and repeat.

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

% javac Factors.java
% java Factors
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 0
   at Factors.main(Factors.java:5)

% java Factors 98
Exception in thread "main"
java.lang.ArithmeticException: / by zero
   at Factors.main(Factors.java:8)

% java Factors 98
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
98 = 2 × 7× 7 ✓

% java Factors 98
2 7 7%
This working program still has bugs!
Debugging a program: testing

Does your legal Java program always do what you want it to do?
  • You need to test on many types of inputs it to find out.
  • Add trace code to find the first error.
  • Fix the error.
  • Repeat.

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

% java Factors 98
2 7 7%
% java Factors 5
??? no output
% java Factors 6
2
2
?? where's the 3?

AHA! Need to print out n (if it is not 1).
Debugging a program: testing

Does your legal Java program *always* do what you want it to do?

• You need to test on many types of inputs it to find out.
• Add trace code to find the first error.
• Fix the error.
• Repeat.

```java
public class Factors {
    public static void main(String[] args) {
        long n = Long.parseLong(args[0]);
        for (int i = 2; i < n; i++) {
            while (n % i == 0) {
                System.out.print(i + " ");
                n = n / i;
            }
        }
        if (n > 1) System.out.println(n);
        else System.out.println();
    }
}
```

Note: This working program still has a bug (stay tuned).
Is your working Java program fast enough to solve your problem?
• You need to test it on increasing problem sizes to find out.
• May need to change the algorithm to fix it.
• Repeat.

Method
• Consider each integer \( i \leq n/i \)
• While \( i \) divides \( n \) evenly
  print \( i \) (it is a factor of \( n \))
  replace \( n \) with \( n/i \).

change the *algorithm*: no need to check when \( i \cdot i > n \) since all smaller factors already checked

% java Factors 1111111
11 73 101 137
% java Factors 111111111111
21649 513239
% java Factors 1111111111111111
11 239 4649 909091
% java Factors 11111111111111111
2071723 5363222357 \( \text{immediate} \)

public class Factors
{
    public static void main(String[] args)
    {
        long n = Long.parseLong(args[0]);
        for (int i = 2; i <= n/i; i++)
        {
            while (n % i == 0)
            {
                System.out.print(i + " ");
                n = n / i;
            }
        }
        if (n > 1) System.out.println(n);
        else System.out.println();
    }
}
Q. How large an integer can I factor?

public class Factors {
    public static void main(String[] args) {
        long n = Long.parseLong(args[0]);
        for (int i = 2; i <= n/i; i++) {
            while (n % i == 0) {
                System.out.print(i + " ");
                n = n / i;
            }
        }
        if (n > 1) System.out.println(n);
        else       System.out.println();
    }
}

%
% java Factors 9201111169755555703
9201111169755555703

digits in largest factor | i < N | i <= N/i
---|---|---
3 | instant | instant
6 | instant | instant
9 | 77 seconds | instant
12 | 21 hours† | instant
15 | 2.4 years† | 2.7 seconds
18 | 2.4 millenia† | 92 seconds

† estimated, using analytic number theory

Lesson. Performance matters!

Note. Internet commerce is still secure: it depends on the difficulty of factoring 200-digit integers.
Debugging your program: summary

Program development is a *four*-step process, with feedback.

**EDIT** your program.

**COMPILE** your program to create an executable file.

**RUN** your program to test that it works as you imagined.

**TEST** your program on realistic and real input data.

**SUBMIT** your program for independent testing and approval.
2. Conditionals & Loops