MusicComputation
Choose meaningful variable names to make your program more readable. For example, use `income`, instead of `num`.

Stick to lower-case variable names. For example, use `income`, instead of `INCOME`. Variables that are all capitals usually indicate a constant (more on this soon.)

Use proper case for all words after the first in a variable name. For example, instead of `totalcommissions`, use `totalCommissions`.

Avoid redefining identifiers previously defined in the Processing API.
Syntax Errors

• Caused when the compiler cannot recognize a statement.
• These are violations of the language
• The compiler normally issues an error message to help the programmer locate and fix it
• Also called compile errors or compile-time errors.
• For example, if you forget a semi-colon, you will get a syntax error.
Run-time Errors

- The compiler cannot pick up on runtime errors. Therefore they happen at runtime.

- Runtime errors fall into two categories.
  - Fatal runtime errors: These errors cause your program to crash.
  - Logic errors: The program can run but the results are not correct.
review

div mod worksheet
Boolean values

• Processing provides a type just for true and false evaluation.

• Named after George Boole, the English mathematician who published “An investigation into the Laws of Thought” in 1854 which began Boolean logic.

• Any Boolean expression will evaluate to either true or false.

• We declare boolean variables like any other variable:

  ```java
  boolean b;
  ```
## Relational Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>==</td>
<td>Equal to</td>
</tr>
<tr>
<td>!=</td>
<td>Not Equal to</td>
</tr>
</tbody>
</table>
boolean boolVar;

boolVar = false;
println ("boolVar: " + boolVar);

int a = 10;
int b = 10;
boolVar = ( a == b);
println ("boolVar: " + boolVar);

println (a == b);
println (a != b);
println (a < b);
println (a <= b);
println (a > b);
println (a >= b);
Equality v. Assignment

• Remember Gets not Equals!
  \[ \text{( grade = 100 )} \]
  Will not evaluate to true or false

• In this case, we are using a single = character. (We really want to use ==)
Control Structures

• Control the flow of a program
• Normally, in Processing, statements are executed in sequential order
• Control structures allow the programmer to specify that a statement other than the next be executed
  – i.e. programmer can control what’s executed in which order
Three Basic Control Structures

• All programs can be written with just these types of structures
  – *Sequence structure*
    • Statements run one after the other
  – *Selection structure*
    • Depending on a condition, do one thing; otherwise, do something else
    • Examples in Processing: if, if else, and switch.
  – *Repetition structure*
    • Repeat some actions over and over
    • Examples in Processing: for loops, while loops, and do/while loops.
The **if** structure

- **Pseudocode:**
  
  ```plaintext
  if some Boolean expression is true
do this
  ```

- **Example:**
  
  ```java
  if ( x == y )
  {
    println(" x is equal to y!" ) ;
  }
  ```

- **Every procedural / OO programming language has some form of an if statement.**
Another if example

if ( temperature >= 85 )
{
    println( "It is hot out!" );
}

if

Flow Chart

temperature >= 85

true

print "It is hot"

false
if with a twist: if else

• Pseudocode:
  if some Boolean expression is true
    do this
  otherwise
    do something else

• Example:

  if ( grade >= 65 )
    println( "You passed!" );
  else
    println( "You failed!" );
if/else Flow Chart

if grade >= 60
    Print "You passed"
else
    Print "You failed"

True

False
Blocks

• To run several lines of code together, you must include them within a block

• For example:

```java
if ( grade >= 60 )
{
    println ( "You passed!" );
    println ( "Congratulations!" );
}
```
Indentation

- Everything within the block of code (even if it is an implicit block because we only use one statement) should be indented – helps you see the block at a quick glance.

- Avoid writing code like this:
  ```java
  if (grade >= 65) {
    println("You passed!!!\n");
    println ("Congratulations!\n");
  }
  ```

- This is valid code, but it is not easy to view the block: *bad style*
Common error: misplaced semi-colon

- Remember, Processing requires that you use a semicolon to terminate a statement.
- A complete if statement is formed as follows:
  ```
  if (boolean expression)
      Statement or block of code;
  ```
- Therefore, if you place a semicolon after the conditional as in
  ```
  if (boolean expression);
      Statement or block of code;
  ```
The compiler will interpret the semicolon as a null statement. In other words, nothing will happen if the expression evaluates to true and the statement or block of code will be executed whether or not the boolean expression is true.
nested if statements

• When one if/else structure is contained inside another if/else structure it is called a nested if/else.

```java
if (grade > 60) {
    if (grade > 70)
        println("You passed");
    else
        println("You passed but need a tutor");
} else
    println("You failed");
```
else if

• Usually you try to nest within the else statement. Note the indentation.

```java
if (grade > 70)
    println("You passed");
else if (grade > 60)
    println("You passed but need a tutor");
else
    println("You failed");
```
Using boolean variables as flags

• You may want to use boolean variables to hold the value of a boolean expression.

• For example:
  
  boolean passed = (grade >= 65)

• Then you can use the variable later in a conditional statement:
  
  if (passed)
      println ("You passed");
  else
      println ("You failed");
&& - Logical AND

((boolean exp a) && (boolean exp b))

• When using the and operator (&&), both expression a and b must be true for the compound statement to be true.

<table>
<thead>
<tr>
<th>truth table</th>
</tr>
</thead>
<tbody>
<tr>
<td>FALSE</td>
</tr>
<tr>
<td>FALSE</td>
</tr>
<tr>
<td>TRUE</td>
</tr>
</tbody>
</table>

• For example:

((total > 50) && (status == 0))
|| - Logical OR

((boolean exp a) || (boolean exp b))

• When using the or operator (||), at least one expression a or b must be true for the compound statement to be true.

truth table

<table>
<thead>
<tr>
<th></th>
<th>FALSE</th>
<th>TRUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

• For example:

((total > 50) || (status == 0))
^ - Exclusive OR
((boolean exp a) ^ (boolean exp b))

• When using the exclusive or operator (^), at least one expression a or b must have opposite Boolean values for the compound statement to be true.

  truth table

<table>
<thead>
<tr>
<th></th>
<th>FALSE</th>
<th>TRUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

• For example:
((person1 == 1) ^ (person2 == 1))
logical negation !

! (a)

- Reverses the truth or falsity of expression a Boolean expression
- ! has high precedence so you must use parenthesis
- You can avoid using the logical negation by expressing the condition differently with an appropriate relational operator. However, in the case of complex expressions, it is sometimes easier to use negation.
- Note: its a unary operator
Unconditional vs. Conditional Boolean Operators

• Processing provides us with a second “and” operator (a single “&”) and a second “or” operator (a single “|”).

• The unconditional operators guarantee that both expressions will be evaluated.

• In this class, you should just use the conditional operators since we will not rely on the execution of the second expression.
switch

• Processing has another conditional structures called switch