BOOLEAN OPERATIONS
AND CONDITIONALS
Topics
Topics

- Boolean Operations
Topics

- Boolean Operations
  - Shorting vs. Non-Shorting
Topics

- **Boolean Operations**
  - Shorting vs. Non-Shorting
  - Combination of Boolean Operations
Topics

- Boolean Operations
  - Shorting vs. Non-Shorting
  - Combination of Boolean Operations
    - Order of Operations
Topics

- **Boolean Operations**
  - Shorting vs. Non-Shorting
  - Combination of Boolean Operations
    - Order of Operations

- **Comparison Operators**
Topics

- Boolean Operations
  - Shorting vs. Non-Shorting
  - Combination of Boolean Operations
    - Order of Operations

- Comparison Operators

- Conditional Statements
Topics

- **Boolean Operations**
  - Shorting vs. Non-Shorting
  - Combination of Boolean Operations
    - Order of Operations

- **Comparison Operators**

- **Conditional Statements**
  - if
Topics

- **Boolean Operations**
  - Shorting vs. Non-Shorting
  - Combination of Boolean Operations
    - Order of Operations

- **Comparison Operators**

- **Conditional Statements**
  - if
  - if...else if...else
Topics

- **Boolean Operations**
  - Shorting vs. Non-Shorting
  - Combination of Boolean Operations
    - Order of Operations

- **Comparison Operators**

- **Conditional Statements**
  - if
  - if...else if...else
  - switch
Boolean Operations
Boolean Operations

- Operations that combine and compare bools
Boolean Operations

- Operations that combine and compare bools
  - ! The NOT Operator
Boolean Operations

- Operations that combine and compare bools
  - ! The NOT Operator
  - && The AND Operator
Boolean Operations

- **Operations that combine and compare bools**
  - `!` The NOT Operator
  - `&&` The AND Operator
  - `||` The OR Operator
Boolean Operations
Boolean Operations

- ! The NOT Operator
Boolean Operations

- The NOT Operator
  - Pronounced either "not" or "bang"
Boolean Operations

- The NOT Operator
  - Pronounced either "not" or "bang"
  - Reverses value of the bool
The NOT Operator

- Pronounced either "not" or "bang"
- Reverses value of the bool

```java
print( !true );  // Outputs: false
print( !false );  // Outputs: true
print( !(true) ); // Outputs: true (the double negative of true)
```
Boolean Operations

- **The NOT Operator**
  - Pronounced either "not" or "bang"
  - Reverses value of the bool
    
    ```
    print( !true );       // Outputs: false
    print( !false );      // Outputs: true
    print( !(!true) );    // Outputs: true (the double negative of true)
    ```
  - Also called the "logical negation operator"
Boolean Operations

- The NOT Operator
  - Pronounced either "not" or "bang"
  - Reverses value of the bool
    
    ```
    print( !true ); // Outputs: false
    print( !false ); // Outputs: true
    print( !(true) ); // Outputs: true (the double negative of true)
    ```
  - Also called the "logical negation operator"
    • This differentiates it from ~, the bitwise not operator
Boolean Operations
Boolean Operations

- &&  The AND Operator
Boolean Operations

- **&& The AND Operator**
  - Returns true only if both operands are true
Boolean Operations

**&&  The AND Operator**

- Returns true only if both operands are true

```javascript
print( false && false );    // false
print( false && true  );    // false
print( true  && false );    // false
print( true  && true  );    // true
```
Boolean Operations
Boolean Operations

- The OR Operator
Boolean Operations

- **The OR Operator**
  - Returns true if either operand is true
Boolean Operations

- **The OR Operator**
  - Returns true if either operand is true

```
print( false && false );    // false
print( false && true  );    // true
print( true  && false );    // true
print( true  && true  );    // true
```
Boolean Operations

- **The OR Operator**
  - Returns true if either operand is true
    
    ```
    print( false && false );    // false
    print( false && true  );    // true
    print( true  && false );    // true
    print( true  && true  );    // true
    ```
  - | (the pipe) is Shift-Backslash
Boolean Operations

- **The OR Operator**
  - Returns true if either operand is true

```javascript
print( false && false );    // false
print( false && true  );    // true
print( true  && false );    // true
print( true  && true  );    // true
```

- | (the pipe) is Shift-Backslash
  - Just above the return or enter key on a US keyboard
Boolean Operations
Boolean Operations

- Shorting vs. Non-Shorting Boolean Operators
Boolean Operations

- **Shorting vs. Non-Shorting Boolean Operators**
  - `&&` and `||` are *shorting operators*
Boolean Operations

- **Shorting vs. Non-Shorting Boolean Operators**
  - `&&` and `||` are *shorting operators*
    - If the first operand of `&&` is false, the second is not evaluated
Boolean Operations

- **Shorting vs. Non-Shorting Boolean Operators**
  - `&&` and `||` are *shorting operators*
    - If the first operand of `&&` is false, the second is not evaluated
    - If the first operand of `||` is true, the second is not evaluated
Boolean Operations

- **Shorting vs. Non-Shorting Boolean Operators**
  - && and || are *shorting operators*
    - If the first operand of && is false, the second is not evaluated
    - If the first operand of || is true, the second is not evaluated
  - & and | are *non-shorting operators*
Boolean Operations

- **Shorting vs. Non-Shorting Boolean Operators**
  - `&&` and `||` are *shorting operators*
    - If the first operand of `&&` is false, the second is not evaluated
    - If the first operand of `||` is true, the second is not evaluated
  - `&` and `|` are *non-shorting operators*
    - Both operands are evaluated regardless of value
Boolean Operations

- **Shorting vs. Non-Shorting Boolean Operators**

  - `&&` and `||` are *shorting operators*
    - If the first operand of `&&` is false, the second is not evaluated
    - If the first operand of `||` is true, the second is not evaluated
  
  - `&` and `|` are *non-shorting operators*
    - Both operands are evaluated regardless of value

  - `&` and `|` are also bitwise operators
Boolean Operations

- **Shorting vs. Non-Shorting Boolean Operators**
  - `&&` and `||` are *shorting operators*
    - If the first operand of `&&` is false, the second is not evaluated
    - If the first operand of `||` is true, the second is not evaluated
  - `&` and `|` are *non-shorting operators*
    - Both operands are evaluated regardless of value
  - `&` and `|` are also bitwise operators
    - `&` and `|` compare each bit of the values passed into them
Boolean Operations

- **Shorting vs. Non-Shorting Boolean Operators**
  - `&&` and `||` are *shorting operators*
    - If the first operand of `&&` is false, the second is not evaluated
    - If the first operand of `||` is true, the second is not evaluated
  - `&` and `|` are *non-shorting operators*
    - Both operands are evaluated regardless of value
  - `&` and `|` are also bitwise operators
    - `&` and `|` compare each bit of the values passed into them
    - Bitwise operators will be used much later when dealing with Unity layers and collisions
Boolean Operations
Boolean Operations

- Combining Boolean Operations
Boolean Operations

- Combining Boolean Operations
  - Can combine several on a single line
Boolean Operations

- Combining Boolean Operations
  - Can combine several on a single line
    
    ```cpp
define bool tf = true || false && true;
```
Combining Boolean Operations

- Can combine several on a single line
  
  ```
  bool tf = true || false && true;
  ```

- Must follow order of operations
Boolean Operations

- **Combining Boolean Operations**
  - Can combine several on a single line
    ```
    bool tf = true || false && true;
    ```
  - Must follow *order of operations*
    ```
    ! NOT
    & Non-Shorting AND / Bitwise AND
    | Non-Shorting OR / Bitwise OR
    && AND
    || OR
    ```
Boolean Operations

- **Combining Boolean Operations**
  - Can combine several on a single line
    ```
    bool tf = true || false && true;
    ```
  - **Must follow order of operations**
    ```
    ! NOT
    & Non-Shorting AND / Bitwise AND
    | Non-Shorting OR / Bitwise OR
    && AND
    || OR
    ```
  - The line above would be interpreted as:
Boolean Operations

- Combining Boolean Operations
  - Can combine several on a single line
    ```
    bool tf = true || false && true;
    ```
  - Must follow order of operations
    ```
    ! NOT
    & Non-Shorting AND / Bitwise AND
    | Non-Shorting OR / Bitwise OR
    && AND
    || OR
    ```
  - The line above would be interpreted as:
    ```
    bool tf = true || (false && true);  // true
    ```
## Boolean Operations

### Combining Boolean Operations

- Can combine several on a single line
  
  ```
  bool tf = true || false && true;
  ```

- **Must follow order of operations**

  ```
  ! NOT
  & Non-Shorting AND / Bitwise AND
  | Non-Shorting OR / Bitwise OR
  && AND
  || OR
  ```

- The line above would be interpreted as:
  
  ```
  bool tf = true || (false && true); // true
  ```

- It's best to always use parentheses to enforce the order in which you want the evaluation to take place!
Comparison Operators
Comparison Operators

- Allow the comparison of two values
Comparison Operators

- Allow the comparison of two values
- Return a bool (either true or false)
Comparison Operators

- Allow the comparison of two values
- Return a bool (either true or false)
  
  `==` Is Equal To
Comparison Operators

- Allow the comparison of two values
- Return a bool (either `true` or `false`)

  ==  Is Equal To
  !=  Not Equal To
Comparison Operators

- Allow the comparison of two values
- Return a bool (either true or false)

==  Is Equal To
!=  Not Equal To
>   Greater Than
Comparison Operators

- Allow the comparison of two values
- Return a bool (either true or false)

== Is Equal To
!= Not Equal To
> Greater Than
< Less Than
Comparison Operators

- Allow the comparison of two values
- Return a bool (either true or false)

== Is Equal To
!= Not Equal To
> Greater Than
< Less Than
>= Greater Than or Equal To
Comparison Operators

- Allow the comparison of two values
- Return a bool (either `true` or `false`)

- `==` Is Equal To
- `!=` Not Equal To
- `>` Greater Than
- `<` Less Than
- `>=` Greater Than or Equal To
- `<=` Less Than or Equal To
COMPARISON BY VALUE OR REFERENCE
Simple variables are compared by value
Simple variables are compared by value
- bool, int, float, char, string, Vector3, Color, Quaternion
COMPARISON BY VALUE OR REFERENCE

- Simple variables are compared by value
  - bool, int, float, char, string, Vector3, Color, Quaternion

- More complex variables are compared by reference
### COMPARISON BY VALUE OR REFERENCE

- **Simple variables are compared by value**
  - bool, int, float, char, string, Vector3, Color, Quaternion

- **More complex variables are compared by reference**
  - When variables are compared by reference, the comparison is not of their internal values but of whether they point to the same location in memory
COMPARISON BY VALUE OR REFERENCE

- Simple variables are compared by value
  - bool, int, float, char, string, Vector3, Color, Quaternion

- More complex variables are compared by reference
  - When variables are compared by reference, the comparison is not of their internal values but of whether they point to the same location in memory
  - GameObject, Material, Renderer, HelloWorld (and other C# classes you write)
COMPARISON BY VALUE OR REFERENCE

- **Simple variables are compared by value**
  - bool, int, float, char, string, Vector3, Color, Quaternion

- **More complex variables are compared by reference**
  - When variables are compared by reference, the comparison is not of their internal values but of whether they point to the same location in memory
  - GameObject, Material, Renderer, HelloWorld (and other C# classes you write)

```csharp
1 GameObject go0 = Instantiate( boxPrefab ) as GameObject;
2 GameObject go1 = Instantiate( boxPrefab ) as GameObject;
3 GameObject go2 = go0;
4 print( go0 == go1 ); // Output: false
5 print( go0 == go2 ); // Output: true
```
Comparison Operators
Comparison Operators

- `==` Is Equal To
Comparison Operators

- `==` Is Equal To
  - Returns true if the values or references compared are equivalent
Comparison Operators

- **== Is Equal To**
  - Returns true if the values or references compared are equivalent

```java
print( 10 == 10 );          // Outputs: True
print( 20 == 10 );          // Outputs: False
print( 1.23f == 3.14f );    // Outputs: False
print( 1.23f == 1.23f );    // Outputs: True
print( 3.14f == Mathf.PI ); // Outputs: False
// Mathf.PI has more decimal places than 3.14f
```
Comparison Operators

- **==** Is Equal To
  - Returns true if the values or references compared are equivalent

```javascript
print( 10 == 10 );          // Outputs: True
print( 20 == 10 );          // Outputs: False
print( 1.23f == 3.14f );    // Outputs: False
print( 1.23f == 1.23f );    // Outputs: True
print( 3.14f == Mathf.PI ); // Outputs: False
  // Mathf.PI has more decimal places than 3.14f
```

- Do NOT confuse `==` and `=`
Comparison Operators

- **== Is Equal To**
  - Returns true if the values or references compared are equivalent
  
  ```
  print( 10 == 10 );          // Outputs: True
  print( 20 == 10 );          // Outputs: False
  print( 1.23f == 3.14f );    // Outputs: False
  print( 1.23f == 1.23f );    // Outputs: True
  print( 3.14f == Mathf.PI ); // Outputs: False
  // Mathf.PI has more decimal places than 3.14f
  ```
  
- Do NOT confuse `==` and `=`
  
  `==` The *comparison* operator
Comparison Operators

- **== Is Equal To**
  - Returns true if the values or references compared are equivalent
    
    ```
    print( 10 == 10 );          // Outputs: True
    print( 20 == 10 );          // Outputs: False
    print( 1.23f == 3.14f );    // Outputs: False
    print( 1.23f == 1.23f );    // Outputs: True
    print( 3.14f == Mathf.PI ); // Outputs: False
    // Mathf.PI has more decimal places than 3.14f
    ```
  - Do NOT confuse `==` and `=`
    
    `==` The *comparison* operator
    
    `=` The *assignment* operator
Comparison Operators
Comparison Operators

- `!=` Not Equal To
Comparison Operators

- !=  Not Equal To
  - Returns true if the values or references compared are NOT equivalent
### Comparison Operators

- **!= Not Equal To**
  - Returns true if the values or references compared are NOT equivalent

```java
print( 10 != 10 );          // Outputs: False
print( 20 != 10 );          // Outputs: True
print( 1.23f != 3.14f );    // Outputs: True
print( 1.23f != 1.23f );    // Outputs: False
print( 3.14f != Mathf.PI ); // Outputs: True
```
Comparison Operators
Comparison Operators

- > Greater Than
Comparison Operators

- >  Greater Than
  - Returns true if the first operand is greater than the second
Comparison Operators

- **Greater Than**
  - Returns true if the first operand is greater than the second

  ```
  print( 10 > 10 );          // Outputs: False
  print( 20 > 10 );          // Outputs: True
  print( 1.23f > 3.14f );    // Outputs: False
  print( 1.23f > 1.23f );    // Outputs: False
  print( 3.14f > 1.23f );    // Outputs: True
  ```
Comparison Operators

- **>** Greater Than
  - Returns true if the first operand is greater than the second

  ```
  print( 10 > 10 );          // Outputs: False
  print( 20 > 10 );          // Outputs: True
  print( 1.23f > 3.14f );    // Outputs: False
  print( 1.23f > 1.23f );    // Outputs: False
  print( 3.14f > 1.23f );    // Outputs: True
  ```

- **<** Less Than
Comparison Operators

- **Greater Than**
  - Returns true if the first operand is greater than the second

```java
print( 10 > 10 ); // Outputs: False
print( 20 > 10 ); // Outputs: True
print( 1.23f > 3.14f ); // Outputs: False
print( 1.23f > 1.23f ); // Outputs: False
print( 3.14f > 1.23f ); // Outputs: True
```

- **Less Than**
  - Returns true if the first operand is less than the second
Comparison Operators

- **Greater Than**
  - Returns true if the first operand is greater than the second

  ```
  print( 10 > 10 );          // Outputs: False
  print( 20 > 10 );          // Outputs: True
  print( 1.23f > 3.14f );    // Outputs: False
  print( 1.23f > 1.23f );    // Outputs: False
  print( 3.14f > 1.23f );    // Outputs: True
  ```

- **Less Than**
  - Returns true if the first operand is less than the second

  ```
  print( 10 < 10 );          // Outputs: True
  print( 20 < 10 );          // Outputs: False
  print( 1.23f < 3.14f );    // Outputs: True
  print( 1.23f < 1.23f );    // Outputs: True
  print( 3.14f < 1.23f );    // Outputs: False
  ```
Comparison Operators
Comparison Operators

- `>=` Greater Than or Equal To
Comparison Operators

- >= Greater Than or Equal To
  
  - True if the 1\textsuperscript{st} operand is greater than or equal to the 2\textsuperscript{nd}
Comparison Operators

- **>= Greater Than or Equal To**
  - True if the 1\textsuperscript{st} operand is greater than or equal to the 2\textsuperscript{nd}
  
  ```
  print( 10 >= 10 );          // Outputs: True
  print( 20 >= 10 );          // Outputs: True
  print( 1.23f >= 3.14f );    // Outputs: False
  print( 1.23f >= 1.23f );    // Outputs: True
  print( 3.14f >= 1.23f );    // Outputs: True
  ```
Comparison Operators

- **>=** Greater Than or Equal To
  - True if the 1st operand is greater than or equal to the 2nd
    
    ```
    print( 10 >= 10 ); // Outputs: True
    print( 20 >= 10 ); // Outputs: True
    print( 1.23f >= 3.14f ); // Outputs: False
    print( 1.23f >= 1.23f ); // Outputs: True
    print( 3.14f >= 1.23f ); // Outputs: True
    ```

- **<=** Less Than or Equal To
Comparison Operators

- **>= Greater Than or Equal To**
  - True if the 1st operand is greater than or equal to the 2nd
    
    ```
    print( 10 >= 10 );          // Outputs: True
    print( 20 >= 10 );          // Outputs: True
    print( 1.23f >= 3.14f );    // Outputs: False
    print( 1.23f >= 1.23f );    // Outputs: True
    print( 3.14f >= 1.23f );    // Outputs: True
    ```

- **<= Less Than or Equal To**
  - True if the 1st operand is less than or equal to the 2nd
Comparison Operators

- **>= Greater Than or Equal To**
  - True if the 1\textsuperscript{st} operand is greater than or equal to the 2\textsuperscript{nd}

  ```
  print( 10 >= 10 );       // Outputs: True
  print( 20 >= 10 );       // Outputs: True
  print( 1.23f >= 3.14f ); // Outputs: False
  print( 1.23f >= 1.23f ); // Outputs: True
  print( 3.14f >= 1.23f ); // Outputs: True
  ```

- **<= Less Than or Equal To**
  - True if the 1\textsuperscript{st} operand is less than or equal to the 2\textsuperscript{nd}

  ```
  print( 10 <= 10 );       // Outputs: True
  print( 20 <= 10 );       // Outputs: False
  print( 1.23f <= 3.14f ); // Outputs: True
  print( 1.23f <= 1.23f ); // Outputs: True
  print( 3.14f <= 1.23f ); // Outputs: False
  ```
Conditional Statements
Conditional Statements

- Control Flow Within Your Programs
Conditional Statements

- Control Flow Within Your Programs
  
  ```python
  if
  ```
Conditional Statements

- Control Flow Within Your Programs

  if

  if / else
Conditional Statements

- Control Flow Within Your Programs

  if
  if / else
  if / else if / else
Conditional Statements

- Control Flow Within Your Programs

  if

  if / else

  if / else if / else

  switch
Conditional Statements

- Control Flow Within Your Programs
  
  ```
  if
  if / else
  if / else if / else
  switch
  ```

- Can be combined with Boolean operations
Conditional Statements

- **Control Flow Within Your Programs**
  
  ```
  if
  if / else
  if / else if / else
  switch
  ```

- Can be combined with Boolean operations

- Make use of *braces* `{ }`
Conditional Statements
Conditional Statements

- **if** Performs code within braces if the argument within parentheses is true
Conditional Statements

- **if** Performs code within braces if the argument within parentheses is true

```java
if (true) {
    print( "This line will print." );
}

if (false) {
    print( "This line will NOT print." );
}

// The output of this code will be:
```
Conditional Statements

- **if** Performs code within braces if the argument within parentheses is true

```java
if (true) {
    print( "This line will print." );
}

if (false) {
    print( "This line will NOT print." );
}

// The output of this code will be:
// This line will print.
Conditional Statements

- **if** Performs code within braces if the argument within parentheses is true

  ```java
  if (true) {
    print( "This line will print." );
  }

  if (false) {
    print( "This line will NOT print." );
  }
  
  // The output of this code will be:
  // This line will print.
  
  - All the code within the braces of the if statement executes
Conditional Statements
Conditional Statements

- Combining if statements with boolean operations
Conditional Statements

- **Combining if statements with boolean operations**

```c
bool night = true;
bool fullMoon = false;

if (night) {
    print( "It's night." );
}
if (!fullMoon) {
    print( "The moon is not full." );
}
if (night && fullMoon) {
    print( "Beware werewolves!!" );
}
if (night && !fullMoon) {
    print( "No werewolves tonight. (Whew!)" );
}

// The output of this code will be:
```
Conditional Statements

- Combining if statements with boolean operations

```c
bool night = true;
bool fullMoon = false;

if (night) {
    print( "It's night." );
}
if (!fullMoon) {
    print( "The moon is not full." );
}
if (night && fullMoon) {
    print( "Beware werewolves!!!" );
}
if (night && !fullMoon) {
    print( "No werewolves tonight. (Whew!)" );
}

// The output of this code will be:
//     It's night.
//     The moon is not full.
//     No werewolves tonight. (Whew!)
```
Conditional Statements
Conditional Statements

- Combining if statements with comparison operators
Conditional Statements

- Combining if statements with comparison operators

```java
if (10 == 10 ) {
    print( "10 is equal to 10." );
}
if ( 10 > 20 ) {
    print( "10 is greater than 20." );
}
if ( 1.23f <= 3.14f ) {
    print( "1.23 is less than or equal to 3.14." );
}
if ( 1.23f >= 1.23f ) {
    print( "1.23 is greater than or equal to 1.23." );
}
if ( 3.14f != Mathf.PI ) {
    print( "3.14 is not equal to " + Mathf.PI + "." );
    // + can be used to concatenate strings with other data types.
    // When this happens, the other data is converted to a string.
}
```
Combining if statements with comparison operators

```java
if (10 == 10 ) {
    print( "10 is equal to 10." );
}
if ( 10 > 20 ) {
    print( "10 is greater than 20." );
}
if ( 1.23f <= 3.14f ) {
    print( "1.23 is less than or equal to 3.14." );
}
if ( 1.23f >= 1.23f ) {
    print( "1.23 is greater than or equal to 1.23." );
}
if ( 3.14f != Mathf.PI ) {
    print( "3.14 is not equal to \""+Mathf.PI\"\"." );
    // + can be used to concatenate strings with other data types.
    // When this happens, the other data is converted to a string.
}
```

- Don't accidentally use `=` in an if statement!!!
Conditional Statements
Conditional Statements

- if / else
Conditional Statements

- **if / else**
  - Performs one action if true, and another if false
Conditional Statements

- **if / else**
  - Performs one action if true, and another if false

```cpp
bool night = false;

if (night) {
    print( "It's night." );
} else {
    print( "What are you worried about?" );
}
```

// The output of this code will be:
Conditional Statements

- **if / else**
  - Performs one action if true, and another if false

```c
bool night = false;

if (night) {
    print( "It's night." );
} else {
    print( "What are you worried about?" );
}
```

// The output of this code will be:
//     What are you worried about?
Conditional Statements
Conditional Statements

- if / else if / else
Conditional Statements

- `if / else if / else`
  
  - Possible to chain several `else if` clauses
Conditional Statements

- **if / else if / else**

  - Possible to chain several else if clauses

    ```
    bool night = true;
    bool fullMoon = true;

    if (!night) { // Condition 1 (false)
        print( "It’s daytime. What are you worried about?" );
    } else if (fullMoon) { // Condition 2 (true)
        print( "Beware werewolves!!!" );
    } else { // Condition 3 (not checked)
        print( "It's night, but the moon is not full." );
    }
    ```

    // The output of this code will be:
Conditional Statements

- **if / else if / else**
  
  Possible to chain several else if clauses

```cpp
bool night = true;
bool fullMoon = true;

if (!night) { // Condition 1 (false)
    print( "It’s daytime. What are you worried about?" );
} else if (fullMoon) { // Condition 2 (true)
    print( "Beware werewolves!!!" );
} else { // Condition 3 (not checked)
    print( "It's night, but the moon is not full." );
}

// The output of this code will be:
//    Beware werewolves!!!
```
Conditional Statements
Conditional Statements

- Nested if statements
Conditional Statements

- Nested if statements

```cpp
bool night = true;
bool fullMoon = false;

if (!night) {
    print( "It’s daytime. Why are you worried about?" );
} else {
    if (fullMoon) {
        print( "Beware werewolves!!!" );
    } else {
        print( "It's night, but the moon isn't full." );
    }
}

// The output of this code will be:
```
Conditional Statements

- **Nested if statements**

```c
type bool = true; 
bool fullMoon = false;

if (!night) {
    print( "It's daytime. Why are you worried about?" );
} else {
    if (fullMoon) {
        print( "Beware werewolves!!!" );
    } else {
        print( "It's night, but the moon isn't full." );
    }
}
```

// The output of this code will be:
// It's night, but the moon isn't full.
Conditional Statements
Conditional Statements

- `switch` Alternative to several if statements
Conditional Statements

- **switch** Alternative to several if statements
  - Can only compare for equality
Conditional Statements

- **switch** Alternative to several if statements
  - Can only compare for equality
  - Can only compare against a single variable against literals
Conditional Statements

- `switch` Alternative to several `if` statements
  - Can only compare for equality
  - Can only compare against a single variable against literals

```java
int num = 3;
switch (num) {
    // The variable in parentheses is being compared
    case (0): // Each case is a literal that is compared against num
        print( "The number is zero." );
        break; // Each case must end with a break statement.
    case (1):
        print( "The number is one." );
        break;
    case (2):
        print( "The number is two." );
        break;
    default: // If none of the other cases are true, default will happen
        print( "The number is more than a couple." );
        break;
}
// The switch statement ends with a closing brace.
```
Conditional Statements

- **switch** Alternative to several if statements
  - Can only compare for equality
  - Can only compare against a single variable against literals

```java
int num = 3;
switch (num) {
    // The variable in parentheses is being compared
    case (0):  // Each case is a literal that is compared against num
        print( "The number is zero." );
        break;  // Each case must end with a break statement.
    case (1):
        print( "The number is one." );
        break;
    case (2):
        print( "The number is two." );
        break;
    default:  // If none of the other cases are true, default will happen
        print( "The number is more than a couple." );
        break;
}
// The switch statement ends with a closing brace.

// The output of this code is: The number is more than a couple.
```
Conditional Statements
Conditional Statements

- Switch can "fall through" to other cases
Conditional Statements

- **Switch can "fall through" to other cases**

```java
int num = 3;
switch (num) {
    case 0:
        print( "The number is zero." );
        break;
    case 1:
        print( "The number is one." );
        break;
    case 2:
        print( "The number is a couple." );
        break;
    case 3: // case (3) falls through to case (4)
    case 4: // case (4) falls through to case (5)
    case 5:
        print( "The number is a few." );
        break;
    default:
        print( "The number is more than a few." );
        break;
}
```
Conditional Statements

- Switch can "fall through" to other cases

```java
int num = 3;
switch (num) {
    case (0):
        print( "The number is zero." );
        break;
    case (1):
        print( "The number is one." );
        break;
    case (2):
        print( "The number is a couple." );
        break;
    case (3):                         // case (3) falls through to case (4)
    case (4):                         // case (4) falls through to case (5)
    case (5):
        print( "The number is a few." );
        break;
    default:
        print( "The number is more than a few." );
        break;
}

// The output of this code is: The number is a few.
```
Chapter 20 – Summary
Chapter 20 – Summary

- Boolean Operations: !    &&    ||    &    |
Chapter 20 – Summary

- Boolean Operations: ! && || & |
- Learned about "shorting operations"
Chapter 20 – Summary

- Boolean Operations: ! && || & | | & | |
- Learned about "shorting operations"
- Boolean operations can be combined
Chapter 20 – Summary

- **Boolean Operations:** ! && | | &

- Learned about "shorting operations"

- Boolean operations can be combined

- **Comparison Operators:** == != > < >= <=
Chapter 20 – Summary

- **Boolean Operations:** !    &&    ||    &    |
- **Learned about "shorting operations"**
- **Boolean operations can be combined**
- **Comparison Operators:** ==  !=  >  <  >=  <=
- **Conditional Statements:** if  if...else  switch
Chapter 20 – Summary

- Boolean Operations: !   &&   ||   &   |
- Learned about "shorting operations"
- Boolean operations can be combined
- Comparison Operators: ==   !=   >   <   >=   <=
- Conditional Statements: if   if...else   switch

  – if and switch statements can be combined in complex ways
Chapter 20 – Summary

- Boolean Operations: ! && || & |
- Learned about "shorting operations"
- Boolean operations can be combined
- Comparison Operators: == != > < >= <=
- Conditional Statements: if if...else switch
  - if and switch statements can be combined in complex ways
- Next Chapter: Loops in C# code!