logical operators:

• an expression like \( 1 \leq x \leq 10 \) needs 2 clauses to test in java
  \[(1 \leq x) \land (x \leq 10)\]

• logical operators are:
  
  \[
  \begin{align*}
  ! & \quad \text{not} \\
  && & \text{and} \\
  || & \quad \text{or} \\
  ^ & \quad \text{exclusive or (one is true but not both)}
  \end{align*}
  \]

• truth tables

| A | B | A&&B | A || B | A^B |
|---|---|------|-------|-----|
| T | T |   T  | T     |   F |
| T | F |   F  | T     |   T |
| F | T |   F  | T     |   T |
| F | F |   F  | F     |   F |
logical operators (cont.):

• De Morgan's laws
  \[ ! (a \land b) \text{ same as } !a \lor !b \]

• short-circuiting the \textit{if} clauses:
  Suppose statement is
  \[ \text{if } (p1 \land p2) \]
  Java evaluates \( p1 \) first. If it is false, no need to evaluate \( p2 \)

• What would be the rule for \( p1 \) for “\( || \)”? For “\( ^\lor \)”?

• Useful example: \[ \text{if } (x \neq 0 \land 1/x < 10.) \] (otherwise use nested \textit{if}s)
? Operator (conditional)

\[ booleanExpression \ ? \ expr1 : expr2; \]

- Can use in assignments statement to fit on one line instead of if/else

- Examples:
  
  - \[ absVal = (x > 0) \ ? \ x : -x; \]
  
  - \[ maxVal = (num1 > num2) \ ? \ num1 : num2; \]
  
  - \[ System.out.println( (num\%2 == 0) \ ? \ “even” : “odd”); \]
Switch statement:

```
int dayOfWeek = 4;  // 0 is Sunday, 1 is Monday, etc.
System.out.print("Today is ");

switch (dayOfWeek) {
    case 0: System.out.println("Sunday"); break;
    case 1: System.out.println("Monday"); break;
    case 2: System.out.println("Tuesday"); break;
    case 3: System.out.println("Wednesday"); break;
    case 4: System.out.println("Thursday"); break;
    case 5: System.out.println("Friday"); break;
    case 6: System.out.println("Saturday"); break;
    default: System.out.println("Error: unrecognized day of week");
}
```

- default and break statements not necessary
- but what happens without break statement?
- replaces long sequences of nested if's, but not as flexible
What is Unicode?

• we’ve discussed representing integers; what about characters?

• a character is a ‘primitive’ type (like int, double, …);
  
  \[ \text{char } c = 'M'; \]

• like numbers, characters are simple a sequence of binary digits; it’s the compiler interpretation that distinguishes between them
  
  \[ \text{char } c = (\text{char}) 16; \]

• alphabet is sequential; ‘A’ comes before ‘a’.
  
  can use as follows: suppose char c = ‘m’;
  
  \[ \text{if } ((c \geq 'a') && (c \leq 'z')) \text{ System.out.println(' lowercase');} \]

• Unicode is 16 bits, (ASCII was 8) but still need more;
  
  ignoring supplemental for now (signaled by \ucccc, c a hex digit)

• a String is an Object, not a primitive type, made up of zero or more characters plus more
### ASCII table (subset of Unicode)

<table>
<thead>
<tr>
<th>Dec</th>
<th>Hr</th>
<th>Oct</th>
<th>Char</th>
<th>ASCII Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>NUL (null)</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>01</td>
<td>SOH (start of heading)</td>
<td>0001</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>02</td>
<td>STX (start of text)</td>
<td>0002</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>03</td>
<td>ETX (end of text)</td>
<td>0003</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>04</td>
<td>EOT (end of transmission)</td>
<td>0004</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>05</td>
<td>ENQ (enquiry)</td>
<td>0005</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>06</td>
<td>ACK (acknowledge)</td>
<td>0006</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>07</td>
<td>BEL (bell)</td>
<td>0007</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>08</td>
<td>BS (backspace)</td>
<td>0008</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>09</td>
<td>HT (vertical tab)</td>
<td>0009</td>
</tr>
<tr>
<td>10</td>
<td>A</td>
<td>10</td>
<td>LF (NL line feed, new line)</td>
<td>0010</td>
</tr>
<tr>
<td>11</td>
<td>B</td>
<td>11</td>
<td>VT (vertical tab)</td>
<td>0011</td>
</tr>
<tr>
<td>12</td>
<td>C</td>
<td>12</td>
<td>FF (NF form feed, new page)</td>
<td>0012</td>
</tr>
<tr>
<td>13</td>
<td>D</td>
<td>13</td>
<td>CR (carriage return)</td>
<td>0013</td>
</tr>
<tr>
<td>14</td>
<td>E</td>
<td>14</td>
<td>SO (shift out)</td>
<td>0014</td>
</tr>
<tr>
<td>15</td>
<td>F</td>
<td>15</td>
<td>SI (shift in)</td>
<td>0015</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>00</td>
<td>DLE (data link escape)</td>
<td>0016</td>
</tr>
<tr>
<td>17</td>
<td>17</td>
<td>01</td>
<td>DC1 (device control 1)</td>
<td>0017</td>
</tr>
<tr>
<td>18</td>
<td>18</td>
<td>02</td>
<td>DC2 (device control 2)</td>
<td>0018</td>
</tr>
<tr>
<td>19</td>
<td>19</td>
<td>03</td>
<td>DC3 (device control 3)</td>
<td>0019</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>04</td>
<td>DC4 (device control 4)</td>
<td>0020</td>
</tr>
<tr>
<td>21</td>
<td>21</td>
<td>05</td>
<td>NAK (negative acknowledge)</td>
<td>0021</td>
</tr>
<tr>
<td>22</td>
<td>22</td>
<td>06</td>
<td>SYN (synchronous idle)</td>
<td>0022</td>
</tr>
<tr>
<td>23</td>
<td>23</td>
<td>07</td>
<td>ETB (end of transmission block)</td>
<td>0023</td>
</tr>
<tr>
<td>24</td>
<td>24</td>
<td>08</td>
<td>CAN (cancel)</td>
<td>0024</td>
</tr>
<tr>
<td>25</td>
<td>25</td>
<td>09</td>
<td>EM (end of medium)</td>
<td>0025</td>
</tr>
<tr>
<td>26</td>
<td>26</td>
<td>0A</td>
<td>SUB (substitute)</td>
<td>0026</td>
</tr>
<tr>
<td>27</td>
<td>27</td>
<td>0B</td>
<td>ESC (escape)</td>
<td>0027</td>
</tr>
<tr>
<td>28</td>
<td>28</td>
<td>0C</td>
<td>FS (file separator)</td>
<td>0028</td>
</tr>
<tr>
<td>29</td>
<td>29</td>
<td>0D</td>
<td>GS (group separator)</td>
<td>0029</td>
</tr>
<tr>
<td>30</td>
<td>30</td>
<td>0E</td>
<td>RS (record separator)</td>
<td>0030</td>
</tr>
<tr>
<td>31</td>
<td>31</td>
<td>0F</td>
<td>US (unit separator)</td>
<td>0031</td>
</tr>
</tbody>
</table>

- **numbers are sequential**
- **control characters for printing part of ASCII (`\t, \n, bell`);**
Strings

- a String is a **reference** to a sequence of characters

  For example, suppose

  ```java
  String name = "Marsha";
  ```

  unlike primitive variables, which hold the value in memory itself
  a String variable *refers* to another place in memory

  ```java
  String name                                "Marsha"
  ```

- to get the individual characters use

  ```java
  name.charAt(charPosition) for example, name.charAt[4] is ‘h’
  ```

- length of string if name.length()

  **length is one larger than last character’s position**
Comparing Strings

• since not a primitive variable type, can’t use  ==

• string example: suppose

```java
String s1 = “Marsha”;
String s2 = “marsha”;

s1.equals(s2) returns false;  s2.equals(s1) also false
but s1.equalsIgnoreCase(s2) is true
```

• other useful string methods:

```java
s1.compareTo(s2) returns int: -1 if s1< s2; 0 if equal; 1 if s1 > s2
s1.indexOf(ch) returns int: index of first occurrence of character ch in s1
s1.substring(beginIndex, endIndex) returns substring between beginIndex and endIndex-1
```

• other available string methods in java API  https://docs.oracle.com/javase/8/docs/api/