Primitive data types, expressions, and variables
How the computer sees the world

- Internally, the computer stores everything in terms of 1’s and 0’s
  - Example:
    - `h` → 0110100
    - "hi" → 01101000110101
    - 104 → 0110100

- How can the computer tell the difference between an `h` and 104?
Data types

- **data type**: A category of data values.
  - Example: integer, real number, string

- Data types are divided into two classes:
  - **primitive types**: Java's built-in *simple* data types for numbers, text characters, and logic.
  - **object types**: Coming soon!
Primitive types

Java has eight primitive types. Here are two examples:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>integers</td>
<td>42, -3, 0, 926394</td>
</tr>
<tr>
<td>double</td>
<td>real numbers</td>
<td>3.4, -2.53, 91.4e3</td>
</tr>
</tbody>
</table>

Numbers with a decimal point are treated as real numbers.

Question: Isn’t every integer a real number? Why bother?
Integer or real number?

- Which category is more appropriate?

<table>
<thead>
<tr>
<th>integer (int)</th>
<th>real number (double)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. Temperature in degrees Celsius
2. The population of lemmings
3. Your grade point average
4. A person's age in years
5. A person's weight in pounds
6. A person's height in meters
7. Number of miles traveled
8. Number of dry days in the past month
9. Your locker number
10. Number of seconds left in a game
11. The sum of a group of integers
12. The average of a group of integers

# Other Primitive Data Types

<table>
<thead>
<tr>
<th><strong>Discrete Types</strong></th>
<th><strong>Continuous Types</strong></th>
<th><strong>Non-numeric Types</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>float</td>
<td>boolean</td>
</tr>
<tr>
<td>short</td>
<td>double</td>
<td>char</td>
</tr>
<tr>
<td>int</td>
<td></td>
<td></td>
</tr>
<tr>
<td>long</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Data Type Representations

<table>
<thead>
<tr>
<th>Type</th>
<th>Representation</th>
<th>Bits</th>
<th>Bytes</th>
<th>#Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>True or False</td>
<td>1</td>
<td>N/A</td>
<td>2</td>
</tr>
<tr>
<td>char</td>
<td>‘a’ or ‘7’ or ‘\n’</td>
<td>16</td>
<td>2</td>
<td>$2^{16} = 65,536$</td>
</tr>
<tr>
<td>byte</td>
<td>…,-2,-1,0,1,2,…</td>
<td>8</td>
<td>1</td>
<td>$2^8 = 256$</td>
</tr>
<tr>
<td>short</td>
<td>…,-2,-1,0,1,2,…</td>
<td>16</td>
<td>2</td>
<td>$2^{16} = 65,536$</td>
</tr>
<tr>
<td>int</td>
<td>…,-2,-1,0,1,2,…</td>
<td>16</td>
<td>2</td>
<td>&gt; 4.29 million</td>
</tr>
<tr>
<td>long</td>
<td>…,-2,-1,0,1,2,…</td>
<td>8</td>
<td>1</td>
<td>&gt; 18 quintillion</td>
</tr>
<tr>
<td>float</td>
<td>0.0, 10.5, -100.7</td>
<td>32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>double</td>
<td>0.0, 10.5, -100.7</td>
<td>64</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Manipulating data via expressions

- **expression**: A data value or a set of operations that produces a value.
  - Examples:
    - $1 + 4 \times 3$
    - $3$
    - "CSE142"
    - $(1 + 2) \% 3 \times 4$
Operators

- Arithmetic operators we will use:
  - + addition
  - - subtraction or negation
  - * multiplication
  - / division
  - % modulus, a.k.a. remainder
Evaluating expressions

- When Java executes a program and encounters an expression, the expression is *evaluated* (i.e., computed).
  - Example: \(3 \times 4\) evaluates to 12

- `System.out.println(3 * 4)` prints 12 (after evaluating \(3 \times 4\))
  - How could we print the text \(3 \times 4\) on the console?
Evaluating expressions: Integer division

- When dividing integers, the result is also an integer: the quotient.
  - Example: \( 14 \div 4 \) evaluates to 3, not 3.5 (truncate the number)

- Examples:
  - \( 1425 \div 27 \) is 52
  - \( 35 \div 5 \) is 7
  - \( 84 \div 10 \) is 8
  - \( 156 \div 100 \) is 1
  - \( 24 \div 0 \) is illegal (what do you think happens?)
Evaluating expressions: The modulus (%)

- The modulus computes the remainder from a division of integers.
  - Example: $14 \% 4$ is 2
  - $1425 \% 27$ is 21

\[
\begin{array}{c}
4 \overline{)} 14 \\
\quad \underline{12} \\
\quad 2
\end{array}
\quad
\begin{array}{c}
27 \overline{)} 1425 \\
\quad \underline{135} \\
\quad 75 \\
\quad \underline{54} \\
\quad 21
\end{array}
\]

- What are the results of the following expressions?
  - $45 \% 6$
  - $4 \% 2$
  - $8 \% 20$
  - $11 \% 0$
Applying the modulus

- What expression obtains...
  - the last digit (unit’s place) of a number?
    - Example: From 230857, obtain the 7.

- the last 4 digits of a Social Security Number?
  - Example: From 658236489, obtain 6489.

- the second-to-last digit (ten’s place) of a number?
  - Example: From 7342, obtain the 4.
Applying the modulus

- How can we use the \( \% \) operator to determine whether a number is odd?

- How about if a number is divisible by, say, 27?
Precision in real numbers

- The computer internally represents real numbers in an imprecise way.

**Example:**

```java
System.out.println(0.1 + 0.2);
```

- The output is `0.3000000000000004!`
Precedence

- **precedence**: Order in which operations are computed in an expression.
  - Operators on the same level are evaluated from left to right.
    - Example: \( 1 - 2 + 3 \) is 2 (not -4)
  - Spacing does not affect order of evaluation.
    - Example: \( 1+3 \times 4-2 \) is 11

<table>
<thead>
<tr>
<th>Parentheses</th>
<th>()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplication, Division, Mod</td>
<td>* / %</td>
</tr>
<tr>
<td>Addition, Subtraction</td>
<td>+ –</td>
</tr>
</tbody>
</table>
Precedence examples

\[
1 \times 2 + 3 \times 5 / 4
\]

\[
1 + 2 \times 5 - 4
\]

\[
1 + 0 \times 5 - 4
\]

\[
1 - 3
\]
Mixing integers and real numbers

- When an operator is used on an integer and a real number, the result is a real number.
  - Examples:
    
    \[
    \begin{align*}
    4.2 \times 3 & \text{ is } 12.6 \\
    1 / 2.0 & \text{ is } 0.5
    \end{align*}
    \]

- The conversion occurs on a *per-operator* basis. It affects only its two operands.
  - Notice how \(3 / 2\) is still 1 above, not 1.5.
Concatenation: Operating on strings

- **String concatenation**: Using the + operator between a string and another value to make a longer string.

**Examples:**

```
"hello" + 42 is "hello42"
1 + "abc" + 2 is "1abc2"
"abc" + 1 + 2 is "abc12"
1 + 2 + "abc" is "3abc"
"abc" + 9 * 3 is "abc27" (what happened here?)
"1" + 1 is "11"
4 - 1 + "abc" is "3abc"
```

"abc" + 4 - 1 causes a compiler error. Why?
Exercise: Combining String and Math Expressions

Write a program to print out the following output. Use math expressions to calculate the last two numbers.

Your grade on test 1 was 95.1
Your grade on test 2 was 71.9
Your grade on test 3 was 82.6
Your total points: 249.6
Your average: 83.2
Question

- `int`s are stored in 4 bytes (32 bits)
- In 32 bits, we can store at most $2^{32}$ different numbers
- What happens if we take the largest of these, and add 1 to it?
  - ERROR!
  - This is known as overflow: trying to store something that does not fit into the bits reserved for a data type.
  - Overflow errors are NOT automatically detected!
    - It’s the programmer’s responsibility to prevent these.
  - The actual result in this case is a negative number.
Overflow example

```java
int n = 2000000000;
System.out.println(n * n);
// output: -1651507200
```

- the result of n*n is 4,000,000,000,000,000,000,000,000 which needs 64-bits:

```
--------- high-order bytes ---------
00110111 10000010 11011010 11001110
--------- low order bytes ---------
10011101 10010000 00000000 00000000
```

- In the case of overflow, Java discards the high-order bytes, retaining only the low-order ones
- In this case, the low order bytes represent 1651507200, and since the right most bit is a 1 the sign value is negative.
Another question:

- What happens if we create a `double` value of 1.0, and then keep dividing it by 10?

- Answer: eventually, it becomes 0.0

- This is known as `underflow`: a condition where a calculated value is smaller than what can be represented using the number of bytes assigned to its type.

- Again, Java does not detect this error; it’s up to the programmer to handle it.
What was the answer again?

- Evaluating expressions are somewhat like using the computer as a calculator.
  - A good calculator has "memory" keys to store and retrieve a computed value.
Variables

- **variable**: A piece of your computer's memory that is given a name and type and can store a value.
  - Usage:
    - compute an expression's result
    - store that result into a variable
    - use that variable later in the program

- Variables are a bit like preset stations on a car stereo:
Declaring variables

- To create a variable, it must be *declared*.

- Variable declaration syntax:
  
  `<type> <name>;`

- Convention: Variable identifiers follow the same rules as method names.

- Examples:

  ```c
  int x;
  double myGPA;
  int varName;
  ```
Declaring variables

- Declaring a variable sets aside a piece of memory in which you can store a value.

```c
int x;
int y;
```

- Inside the computer:

```
x: [ ] y: [ ]
```

(The memory still has no value yet.)
Identifiers: Say my name!

- **identifier**: A name given to an entity in a program such as a class or method.
  - Identifiers allow us to refer to the entities.

- **Examples (in bold)**:
  - `public class Hello`
  - `public static void main`
  - `double salary`

- **Conventions for naming in Java (which we will follow)**:
  - `classes`: capitalize each word (`ClassName`)
  - `everything else`: capitalize each word after the first (`myLastName`)
Identifiers: Syntax

- First character must be a letter, _ or $
- Following characters can be any of those or a number

Examples:
- **legal**: susan  second_place  _myName  TheCure  ANSWER_IS_42  $variable  method1  myMethod  name2

- **illegal**: me+u  49er  question?  side-swipe  hi  thereph.d  jim's  2%milk  suzy@yahoo.com

- **Remember**: Java is case-sensitive (name is different from Name)
Identifiers: Keywords

- **keyword**: An identifier that you cannot use, because it already has a reserved meaning in the Java language.

- Complete list of Java keywords:
  
<table>
<thead>
<tr>
<th>abstract</th>
<th>default</th>
<th>if</th>
<th>private</th>
<th>this</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>do</td>
<td>implements</td>
<td>protected</td>
<td>throw</td>
</tr>
<tr>
<td>break</td>
<td>double</td>
<td>import</td>
<td>public</td>
<td>throws</td>
</tr>
<tr>
<td>byte</td>
<td>else</td>
<td>instanceof</td>
<td>return</td>
<td>transient</td>
</tr>
<tr>
<td>case</td>
<td>extends</td>
<td>int</td>
<td>short</td>
<td>try</td>
</tr>
<tr>
<td>catch</td>
<td>final</td>
<td>interface</td>
<td>static</td>
<td>void</td>
</tr>
<tr>
<td>char</td>
<td>finally</td>
<td>long</td>
<td>strictfp</td>
<td>volatile</td>
</tr>
<tr>
<td>class</td>
<td>float</td>
<td>native</td>
<td>super</td>
<td>while</td>
</tr>
<tr>
<td>const</td>
<td>for</td>
<td>new</td>
<td>switch</td>
<td></td>
</tr>
<tr>
<td>continue</td>
<td>goto</td>
<td>package</td>
<td>synchronized</td>
<td></td>
</tr>
</tbody>
</table>

- NB: Because Java is case-sensitive, you could technically use `Class` or `cLaSs` as identifiers, but this is very confusing and thus strongly discouraged.
Setting variables

- **assignment statement**: A Java statement that stores a value into a variable.
  - Variables must be declared before they can be assigned a value.

Assignment statement syntax:
```
<variable> = <expression>;
```

Examples:
```
x = 2 * 4;       x: 8
myGPA = 3.25;
myGPA = 3.25;
```
Setting variables

- A variable can be assigned a value more than once.

- Example:

```java
int x;
x = 3;
System.out.println(x);   // 3

x = 4 + 7;
System.out.println(x);   // 11
```
Using variables

- Once a variable has been assigned a value, it can be used in any expression.

```java
int x;
x = 2 * 4;
System.out.println(x * 5 - 1);
```

- The above has output equivalent to:

```java
System.out.println(8 * 5 - 1);
```

- What happens when a variable is used on both sides of an assignment statement?

```java
int x;
x = 3;
x = x + 2; // what happens?
```
Errors in coding

- **ERROR:** Declaring two variables with the same name
  - Example:
    
    ```java
    int x;
    int x; // ERROR: x already exists
    ```

- **ERROR:** Reading a variable’s value before it has been assigned
  - Example:
    
    ```java
    int x;
    System.out.println(x); // ERROR: x has no value
    ```
Assignment vs. algebra

- The assignment statement is not an algebraic equation!

- `<variable> = <expression>;` means:
  - "store the value of `<expression>` into `<variable>`"

- Some people read `x = 3 * 4;` as
  - "x gets the value of `3 * 4`"

- **ERROR:** `3 = 1 + 2;` is an illegal statement, because `3` is not a variable.
Assignment and types

- A variable can only store a value of its own type.
  - Example:
    ```
    int x;
    x = 2.5;   // ERROR: x can only store int
    ```

- An int value can be stored in a double variable. Why?
  - The value is converted into the equivalent real number.
  - Example:
    ```
    double myGPA;   myGPA: 2.0
    myGPA = 2;
    ```
Legal Assignments

Diagram showing the hierarchy of types:
- boolean
- double
- float
- long
- int
  - char
  - short
  - byte
Assignment exercise

What is the output of the following Java code?

```java
int x;
x = 3;
int y;
y = x;
x = 5;
System.out.println(x);
System.out.println(y);
```
Assignment exercise

- What is the output of the following Java code?
  ```java
  int number;
  number = 2 + 3 * 4;
  System.out.println(number - 1);
  number = 16 % 6;
  System.out.println(2 * number);
  ```

- What is the output of the following Java code?
  ```java
  double average;
  average = (11 + 8) / 2;
  System.out.println(average);
  average = (5 + average * 2) / 2;
  System.out.println(average);
  ```
Shortcut: Declaring and initializing

- A variable can be declared and assigned an initial value in the same statement.

- Declaration_INITIALIZATION statement syntax:
  
  `<type>  <name>  =  <expression> ;`

  - Examples:
    
    `double  myGPA  =  3.95;`
    
    `int  x  =  (11  %  3)  +  12;`
Shortcut: Declaring many variables at once

- It is legal to declare multiple variables on one line:
  ```
  <type>  <name>,  <name>,  ... ,  <name>;
  ```
  - Examples:
    ```
    int a, b, c;
    double x, y;
    ```

- It is also legal to declare/initialize several at once:
  ```
  <type>  <name> = <expression>,  ... ,  <name> = <expression>;
  ```
  - Examples:
    ```
    int a = 2, b = 3, c = -4;
    double grade = 3.5, delta = 0.1;
    ```

- NB: The variables must be of the same type.
Java has several shortcut operators that allow you to quickly modify a variable's value.

<table>
<thead>
<tr>
<th>Shorthand</th>
<th>Equivalent longer version</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;variable&gt;</code> <code>+=</code> <code>&lt;exp&gt;</code> <code>;</code></td>
<td><code>&lt;variable&gt;</code> <code>=</code> <code>&lt;variable&gt;</code> <code>+</code> ( <code>&lt;exp&gt;</code> );</td>
</tr>
<tr>
<td><code>&lt;variable&gt;</code> <code>-=</code> <code>&lt;exp&gt;</code> <code>;</code></td>
<td><code>&lt;variable&gt;</code> <code>=</code> <code>&lt;variable&gt;</code> <code>+</code> ( <code>&lt;exp&gt;</code> );</td>
</tr>
<tr>
<td><code>&lt;variable&gt;</code> <code>*=</code> <code>&lt;exp&gt;</code> <code>;</code></td>
<td><code>&lt;variable&gt;</code> <code>=</code> <code>&lt;variable&gt;</code> <code>*</code> ( <code>&lt;exp&gt;</code> );</td>
</tr>
<tr>
<td><code>&lt;variable&gt;</code> <code>/=</code> <code>&lt;exp&gt;</code> <code>;</code></td>
<td><code>&lt;variable&gt;</code> <code>=</code> <code>&lt;variable&gt;</code> <code>/</code> ( <code>&lt;exp&gt;</code> );</td>
</tr>
<tr>
<td><code>&lt;variable&gt;</code> <code>%=</code> <code>&lt;exp&gt;</code> <code>;</code></td>
<td><code>&lt;variable&gt;</code> <code>=</code> <code>&lt;variable&gt;</code> <code>%</code> ( <code>&lt;exp&gt;</code> );</td>
</tr>
</tbody>
</table>

Examples:
- `x += 3 - 4;` // `x = x + (3 - 4);`
- `gpa -= 0.5;` // `gpa = gpa - (0.5);`
- `number *= 2;` // `number = number * (2);`
Shortcut: Increment and decrement

- Incrementing and decrementing is used often enough that they have a special shortcut operator!

<table>
<thead>
<tr>
<th>Shorthand</th>
<th>Equivalent longer version</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;variable&gt;</code>++;</td>
<td><code>&lt;variable&gt; = &lt;variable&gt; + 1;</code></td>
</tr>
<tr>
<td><code>&lt;variable&gt;</code>--;</td>
<td><code>&lt;variable&gt; = &lt;variable&gt; - 1;</code></td>
</tr>
</tbody>
</table>

- Examples:
  ```
  int x = 2;
  x++;  // x = x + 1;
  // x now stores 3

  double gpa = 2.5;
  gpa++;  // gpa = gpa + 1;
  // gpa now stores 3.5
  ```
Putting it all together: Exercise

- Write a program that stores the following data:
  - Section 001 has 27 students.
  - Section 002 has 28 students.
  - Section 003 has 11 students.
  - Section 004 has 9 students.
  - The average number of students per section.

- Have your program print the following:

  There are 27 students in Section 001.
  ...
  There are <?> total students.
  There are an average of <?> students per section.