Chapter 2
Fundamental Data Types

JavaScript scripting with BeanShell
What is a data type?

- A data type is a set of values and a set of operations on these values

**Example: Integers**
- infinite discrete set of values
- operations (+, -, *, DIV, MOD, etc.)

**Example: Real numbers**
- infinite continuous set of values like 3.21, π, √2
- operations (+, -, *, /, etc)
DIV and MOD (Euclid)

- If \( n \) and \( d \) are non-negative integers and \( d \neq 0 \) there are unique integers \( q \) called the quotient and \( r \) called the remainder such that \( 0 \leq r < d \) and \( n = dq + r \)
- Pseudo-code: \( q \) is \( n \text{ DIV} d \), \( r \) is \( n \text{ MOD} d \)
- Java: \( q \) is given by \( n \div d \)
  \( r \) is given by \( r \% d \)
- Example \( 27 \div 5 \) is \( 5 \) and \( 27 \% 5 \) is \( 2 \)
Computer data types

- Integers and real numbers are infinite sets.
- We can only represent a finite number of them inside any computer which has a finite amount of memory.
- Therefore we can only represent a finite subset of integers.
- Similarly we can only represent a finite subset of real numbers.
Java primitive numeric types

<table>
<thead>
<tr>
<th>Type</th>
<th>bits</th>
<th>Minimum value</th>
<th>Maximum value</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>8</td>
<td>-128</td>
<td>127</td>
</tr>
<tr>
<td>short</td>
<td>16</td>
<td>-32768</td>
<td>32767</td>
</tr>
<tr>
<td>char</td>
<td>16</td>
<td>0</td>
<td>65535</td>
</tr>
<tr>
<td>int</td>
<td>32</td>
<td>-2147483648 = $-2^{31}$</td>
<td>2147483647 = $2^{31} - 1$</td>
</tr>
<tr>
<td>long</td>
<td>64</td>
<td>-9223372036854775808</td>
<td>9223372036854775807</td>
</tr>
<tr>
<td>float</td>
<td>32</td>
<td>$\pm 1.40 \times 10^{-45}$</td>
<td>$\pm 3.40 \times 10^{38}$</td>
</tr>
<tr>
<td>double</td>
<td>64</td>
<td>$\pm 4.94 \times 10^{-324}$</td>
<td>$\pm 1.80 \times 10^{308}$</td>
</tr>
</tbody>
</table>

We will mostly use the **int** and **double** numeric types.
Overflow and Underflow

- The result of an arithmetic operation on integer or floating point numbers can be too large to fit in the number of bits used to represent the answer

- This is called **overflow**

- There is also **underflow** to zero

- For floating point numbers (e.g. `double`) there can also be **round-off error**
Truncation and round-off error

- **Truncation error** occurs when a number with an infinite decimal expansion is stored as a 32 or 64 bit binary number:

- **Examples**
  \[ \pi = 3.1415926535897932384... \]
  0.1 cannot be stored exactly in binary

- **Round-off** error occurs when performing arithmetic operations.
Integer, floating point literals

- **integer literals**
  - int: 0, 10, -37, 56
  - long: 0L, 10L, 12111231212L (L or l)

- **floating point literals**
  - float: 3.4F, -4.56F, 5.467E-17F (F or f)
  - double: 3.4, 3.4D, -4.56, 5.43E23 (D or d)

**Note:** if no suffix is used, `double` is assumed
Declaring Variables

- A **variable** has a name and a type and corresponds to a storage location in computer memory which can hold a value of the type (e.g., 32 bits for an **int**).

- To **declare** a variable means to specify its type and its name and optionally its initial value.
Declaring variables in Java

```java
int width;
double area;
```

These are examples of uninitialized variables

Java is a strongly typed language

Computer Memory
Initializing variables (2 ways)

- When they are declared
  ```
  int width = 5;
  double area = 3.1416;
  ```
- Later, using assignment
  ```
  width = 5;
  area = 3.1416;
  ```
Multiple declarations

Several variable declarations can be included in one statement:

Example

- `double radius, area, circumference;`

Example

- `double radius = 2.0, area, circumference;`
Multiple assignments

A common value can be assigned to several variables in one assignment statement

Example

\[ a = b = c = 0; \]

is equivalent to

\[ a = 0; \]
\[ b = 0; \]
\[ c = 0; \]
Choosing variable names

- Begin them with a lower case letter
- Other characters can be letters and numbers
- Cannot have spaces
- Capitalize the first letter of interior words
- Example:

```
numberOfStudents, not numberofstudents
totalCents, not totalcents or TOTALCENTS
```
Constants

- Constants, like variables, have a name and a type but their initial value cannot change.
- The keyword `final` identifies a constant.
- Examples (note the naming convention)

```java
final double PI = 3.141592653589793;
final double CM_PER_INCH = 2.54;
final int MARGIN_WIDTH = 5;
```
Arithmetic Operations (1)

- + is used for addition
- - is used for subtraction
- * is used for multiplication
- / is used for both integer division and floating point division
  - for integers / is like DIV (gives quotient)
  - for floating point numbers it is a real division:

  5/2 gives 2 but 5.0/2.0 gives 2.5
Arithmetic Operations (2)

- `%` is used in Java to find the remainder for an integer division. In pseudo-code we use **MOD** to denote this operation.

**Examples:**

\[
\begin{align*}
5 \div 2 & \text{ is 2 and } 5 \mod 2 \text{ is 1} \\
25 \div 7 & \text{ is 3 and } 25 \mod 7 \text{ is 4}
\end{align*}
\]

- `totalCents / 25` is the number of quarters
- `totalCents % 25` is the remaining cents
### Expression examples (1)

<table>
<thead>
<tr>
<th>Expression</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a + bc - 4$</td>
<td>$a + b\cdot c - 4$</td>
</tr>
<tr>
<td>$\frac{1}{2}(a + b) (c - 7)$</td>
<td>$(a + b) \cdot (c - 7.0) / 2.0$</td>
</tr>
<tr>
<td>$\frac{9}{5} c + 32$</td>
<td>$(9.0 / 5.0) \cdot c + 32.0$</td>
</tr>
<tr>
<td>$\frac{5}{9} (f - 32)$</td>
<td>$(5.0 / 9.0) \cdot (f - 32.0)$</td>
</tr>
<tr>
<td>$a^2 b^2 + \frac{c^3}{a + b}$</td>
<td>$a \cdot a \cdot b \cdot b + c \cdot c \cdot c / (a + b)$</td>
</tr>
</tbody>
</table>
Expression examples (2)

\[ 3x^2 - 2x + 4 \]

\[ 3.0 \times x \times x - 2.0 \times x + 4.0 \]

\[ 1.3 + x(3.4 - x(2.5 + 4.2x)) \]

\[ 1.3 + x \times (3.4 - x \times (2.5 + 4.2 \times x)) \]

\[ s(s-a)(s-b)(s-c) \]

\[ s \times (s-a) \times (s-b) \times (s-c) \]

\[ \sqrt{a^2 + b^2 - 2ab \cos \gamma} \]

\[ \text{Math.sqrt}(a*a + b*b - 2*a*b*\text{Math.cos}(\text{gamma})) \]
Precedence Rules (1)

- *, %, and / have the same precedence and they have a higher precedence than +, -. They are done in the order in which they appear.

- **Example**: a + b*c/d
  - the multiplication is done first, then the division
Precedence Rules (2)

- +, – have the same precedence. They are done in the order in which they appear

Example: \( a + b \times c - d \)

- the multiplication is done first,
- then the addition is done,
- then the subtraction is done.
Precedence Rules (3)

- Parentheses have the highest precedence.

- **Example**: \((a + b) \times (c - d)\)
  - the addition is done first,
  - then the subtraction is done,
  - and finally the multiplication is done.
Assignment Statements (1)

- An assignment statement is used to give a value to a variable. It is the most common kind of statement in programs.

Example: (assume radius, area, circumference have been declared as double variables)

```java
radius = 2.0;
area = Math.PI * radius * radius;
circumference = 2.0 * Math.PI * radius;
```
Assignment Statements (2)

- dollars and cents example:
  - assume all variables are of type `int`

```java
totalCents = 3517;
dollars = totalCents / 100;
cents = totalCents % 100;
```

- dollars will have the value 35
- cents will have the value 17
Assignment Statements (3)

- extract digits of a 3 digit number like 123
  - assume all variables are of type int

```
n = 123;
hundreds = n / 100;
remainder = n % 100;
tens = remainder / 10;
units = remainder % 10;
```
Other arithmetic operations

\( k++ \); is the same as \( k = k + 1 \);

\( k-- \); is the same as \( k = k - 1 \);

there are also \( ++k \) and \( --k \) (can be different)

\( x += d \); is the same as \( x = x + d \);

\( x -= d \); is the same as \( x = x - d \);

\( x *= d \); is the same as \( x = x * d \);

\( x /= d \); is the same as \( x = x / d \);
Unary and binary operations

- + and - can be binary or unary operators

- **binary operators** (two operands)
  - **Example:** a + b and a - b

- **unary operations** (one operand)
  - **Example:** +a and -a
BeanShell

- Interactive Java scripting environment
- Has a WorkSpace for typing Java statements and commands
- Has `print` and `show` commands for displaying results
- Has a WorkSpace editor that can be used to edit and evaluate Java statements.
BeanShell Examples (1)

```java
bsh % double radius, area, circ;
bsh % radius = 3.0;
bsh % area = Math.PI * radius * radius;
bsh % circ = 2.0 * Math.PI * radius;
bsh % print(area);
28.274333882308138
bsh % print(circ);
18.84955592153876
bsh %
```
intermediate results will be displayed

```
bsh % show();
<true>

bsh % double radius, area, circ;
bsh % radius = 3.0;
<3.0>

bsh % area = Math.PI * radius * radius;
<28.27433882308138>

bsh % circ = 2.0 * Math.PI * radius;
<18.84955592153876>

bsh %
```
BeanShell Examples (3)

```java
bsh % int totalCents, cents, dollars;
bsh % totalCents = 3527;
<3527>
bsh % dollars = totalCents / 100;
<35>
bsh % cents = totalCents % 100;
<27>
bsh %
```
BeanShell Examples (4)

bsh % int n = 123, remainder, hundreds, tens, units;

bsh % hundreds = n / 100;
<1>

bsh % remainder = n % 100;
<23>

bsh % tens = remainder / 10;
<2>

bsh % units = remainder % 10;
<3>
BeanShell Examples (5)

```java
bsh % int i = 3;
bsh % int j = 4;
bsh % i++;
<3>
bsh % print(i);
4
bsh % j--;
<4>
bsh % print(j);
3
```

automatically displayed values are the values before the increment or decrement is applied
Implicit type conversion (1)

bsh % int i = 1;
bsh % double d, e;
bsh % d = i;   
<1>
bsh % e = i + 3.55; 
<4.55>

a double is 64 bits
an int is 32 bits
assignment is valid since every
int will fit in a double

Here the value of i is automatically
converted to a double value since
again there is no loss of precision
Implicit type conversion (2)

Here we are attempting to assign a 64-bit double value to a 32-bit int which cannot always be done without losing information.

```
bsh % int j;
bsh % i = e;

// Error: Can't assign double to int: ...
```

Here the int value is converted to a double value and the result is added to e. This is valid but again the assignment to an int can result in loss of information.

```
bsh % j = i + e;

// Error: Can't assign double to int: ...
```
Explicit type conversion

bsh % i = (int) e;
<4>

bsh % j = i + (int) e;
<8>

Here we are explicitly type casting the double values to int values. The compiler allows this even though there can be loss of information.

This is useful when we want to throw away the fractional part of a double number to obtain an integer. This operation is called truncation.
Truncation example (1)

bsh % i = (int) 12345.5434;
<12345>

This assignment is useful since the integer part of the double number can be stored in an int variable

bsh % i = (int) 12345678912343.5;
<2147483647>

This assignment is not useful since the integer part of the double number is too large to store in an int variable. We get garbage as a result.
random gives double number \( r \) such that \( 0.0 \leq r < 1.0 \)

Then right sides are one of \( 1, 2, \ldots, 10 \)
bsh % double d = 1.11111111111111;
bsh % float f;
bsh % f = d;
// Error: Can't assign double to float
bsh % f = (float) d;
<1.1111112>
bsh % d = 1e-66;
<1.0E-66>
bsh % f = (float) d;
<0.0>
bsh % d = 1e66;
<1.0E66>
bsh % f = (float) d;
<Infinity>

loss of precision

float exponents can't be larger than about 38
The Math class

The Math class is the home of many useful mathematical functions.

- `Math.sqrt(x)` computes square root of `x`
- `Math.round(x)` rounds `x` to nearest `long`
- `Math.pow(x,y)` computes `x` to power `y`
- `Math.sin(x)` computes sine of `x` in radians
Rounding doubles (1)

To round a **double** number \( x \) to the nearest integer use \texttt{Math.round} but note that it returns a **long** (64-bit integer) so a type cast is needed to convert to \texttt{int}:

```java
int i = (int) Math.round(x);
```

This is a long value
Suppose that amount is a `double` number representing an amount of money such as 3.45 (3 dollars and 45 cents). Convert to total cents:

```java
int totalCents = (int) Math.round(100 * amount);
```
Using square root

\[ c = \sqrt{a^2 + b^2 - 2ab \cos \gamma} \]

\[
\text{perimeter} = a + b + c
\]

\[
s = \text{perimeter} / 2
\]

\[
\text{area} = \sqrt{s(s - a)(s - b)(s - c)}
\]

double c = Math.sqrt(a*a + b*b - 2.0*Math.cos(gamma));
double perimeter = a + b + c;
double s = perimeter / 2.0;
double area = Math.sqrt(s*(s-a)*(s-b)*(s-c));
Windchill and heat loss

- Windchill (speed $v$ in km/hr, $t$ in celsius)
  \[
  \text{double } wc = \frac{0.045 \times (5.27 \times \sqrt{v} + 10.45 - 0.28 \times v) \times (t - 33.0) + 33.0}{(10.45 + 10.0 \times \sqrt{v} - v) \times (33.0-t)};
  \]

- Heat loss (speed $v$ in m/sec, $t$ in Celsius)
  \[
  \text{double } h = \frac{0.045 \times (5.27 \times \sqrt{v} + 10.45 - 0.28 \times v) \times (t - 33.0) + 33.0}{(10.45 + 10.0 \times \sqrt{v} - v) \times (33.0-t)};
  \]
Investment example

\[ v = a \left( 1 + \frac{r}{100m} \right)^{mn} \]

- \( r \) interest rate in percent per year
- \( m \) times/year interest is compounded
- \( a \) initial investment,
- \( n \) years to invest
- \( v \) value of investment

```java
double v = 
a * Math.pow(1.0 + r/(100.0*m), m*n);
```
Rounding to 2 decimal places

- Assume $x$ is a double precision variable. Then the variable $x2$ defined by

\[
\text{double } x2 = \frac{\text{Math.round}(x \times 100.0)}{100.0};
\]

has the value of $x$ rounded to two decimal places.
Documenting Methods

- To use a function (method) we need to know the following:
  - To which class does the method belong?
  - What is the name of the method?
  - What are the formal arguments, if any, and what are their types?
  - What type of value, if any, is computed and returned when the method is called?

- The method prototype provides this information.
Method prototypes (1)

double Math.pow(double x, double y)

\( x^y \)

double \( v = a \times Math.pow(1.0 + r / (100.0 \times m), m \times n) \)

using the method

actual arguments
Method prototypes (2)

Here are some prototypes from Math

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static double sqrt(double x)</td>
<td>$\sqrt{x}$</td>
</tr>
<tr>
<td>public static double sin(double x)</td>
<td>$\sin x$</td>
</tr>
<tr>
<td>public static double cos(double x)</td>
<td>$\cos x$</td>
</tr>
<tr>
<td>public static double tan(double x)</td>
<td>$\tan x$</td>
</tr>
<tr>
<td>public static double exp(double x)</td>
<td>$e^x$</td>
</tr>
<tr>
<td>public static double log(double x)</td>
<td>$\ln x$</td>
</tr>
<tr>
<td>public static double random()</td>
<td></td>
</tr>
<tr>
<td>public static long round(double x)</td>
<td></td>
</tr>
<tr>
<td>public static double pow(double x, double y)</td>
<td>$x^y$</td>
</tr>
</tbody>
</table>
Method call expressions

double v =
    a * Math.pow(1.0 + r/(100.0*m),m*n);

int j =
    (int) Math.round(123.56);

int topNumber = (int) (10*Math.random() + 1);

double d =
    Math.pow(x,2.0/3.0) + Math.pow(y,2.0/3.0);

double c =
    Math.sqrt(a*a + b*b - 2.0*a*b*Math.cos(gamma));

The method call expressions are underlined
Terminology (1)

- simple identifier
  - Example: radius, numberOfStudents, Math
- numeric literal
  - Examples: 1, -34, 1L, -3456789212231L, 1.0F, 1.034
- variable
  - Example radius
- type
  - Examples: int, float, double
Terminology (2)

- variable declaration
  - double radius;
  - double radius = 2.0;
  - double area = Math.PI * radius*radius;
  - int n = 123, remainder, hundreds, tens, units;
  - double area, circumference;
  - double radius = 3.0, area;
Terminology (3)

- constant declaration
  - static final double CM_PER_INCH = 2.54;

- arithmetic expression
  - radius;
  - 1.0 * Math.PI * radius;
  - remainder % 10;
Terminology (4)

- assignment statement
  - radius = 2.0;
  - area = Math.PI * radius * radius;
  - a = b = c = 0.0;