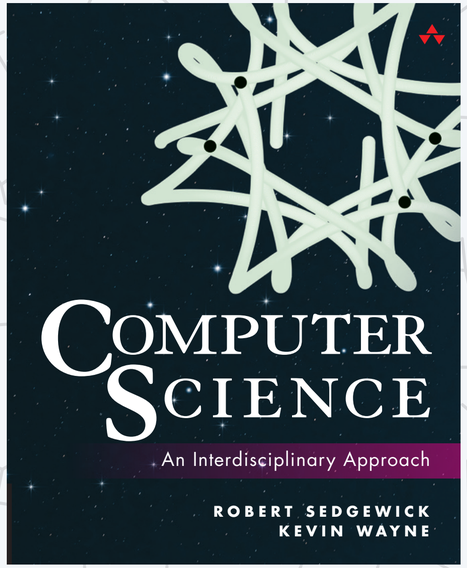


https://introcs.cs.princeton.edu

1.2 Built-In Data Types

- *strings*
- *integers*
- *floating-point numbers*
- *booleans*
- *type conversion*

Built-in data types

A data type (type) is a set of values and a set of operations on those values.

type	set of values	example values	examples of operations
`int`	*integers*	`17` `-12345`	*add, subtract, multiply, divide, compare, equality*
`double`	*floating-point numbers*	`2.5` `-0.125`	*add, subtract, multiply, divide, compare, equality*
`boolean`	*truth values*	`true` `false`	*and, or, not, equality*
`String`	*sequences of characters*	`"Hello, World"` `"COS 126 is fun!"`	*concatenate*

Java's built-in data types
(that we use regularly in this course)

Programming terminology

Program. Sequence of statements. ← *for now*

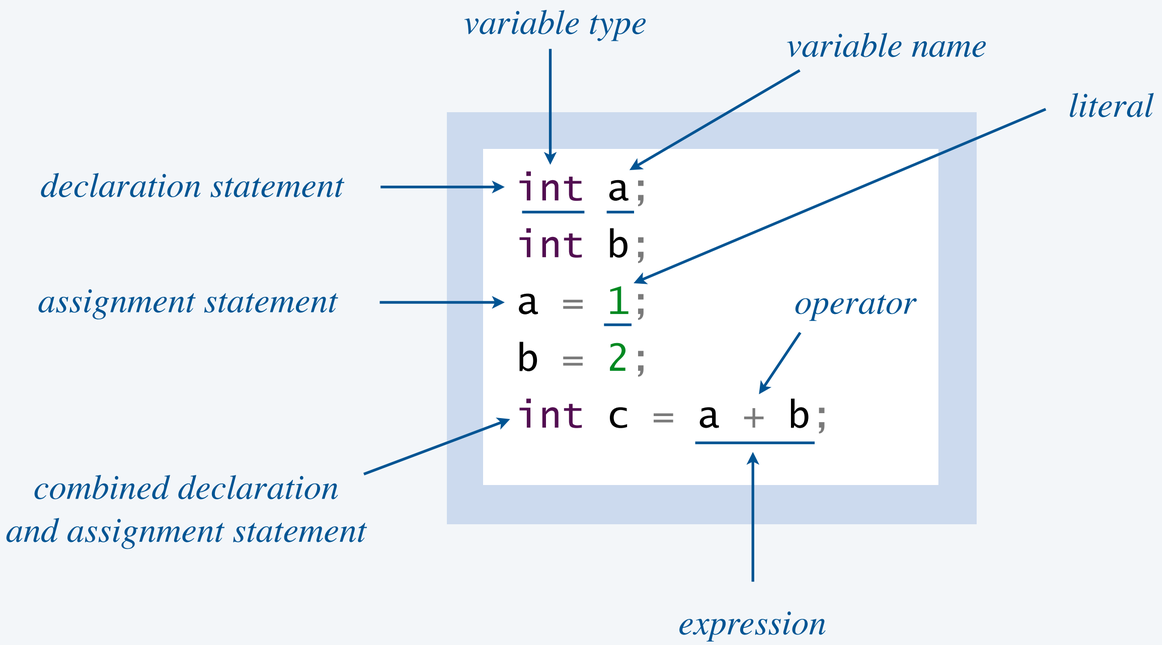
Declaration statement. Associates a variable with a name and type.

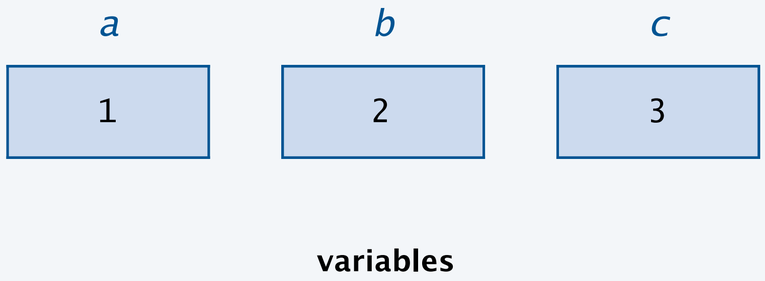
Variable. A place to store a data-type value.

Assignment statement. Stores a value in a variable.

Literal. Programming-language representation of a data-type value.

Expression. A combination of variable names, literals, operators, etc. that evaluates to a value.





Assignment statements

Q. How does an assignment statement work?

A. Java evaluates the expression on the RHS and assigns that value to the variable on the LHS.

expression type must be compatible with variable type

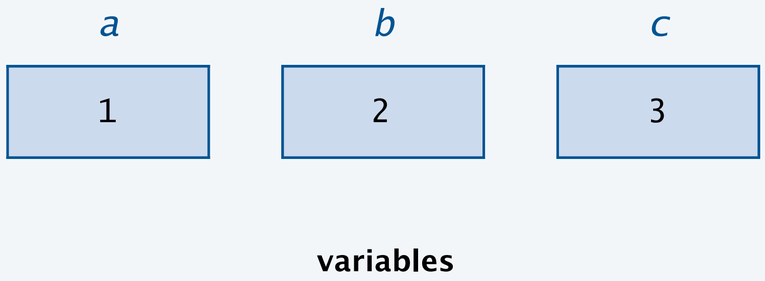
assignment operator = means assignment (not math equality!)

variable name evaluates to value stored in variable

```
int a;
int b;
a = 1;
b = 2;
int c = a + b;
```

variable of type `int`

expression of type `int`



Valid and invalid assignment statements

Q. Which of these independent code fragments are valid?

statements	compiles?	remark
`int a = 1;` `123 = a;`	☹️	*LHS is not a variable* *(= does not mean math equality)*
`double a = 2.5;` `int b = a;`	☹️	*RHS type is incompatible with LHS type*
`String s = 123;`	☹️	*RHS type is incompatible with LHS type*
`int b = 2;` `int a = 3 * b;`	😍	*RHS can be an expression*
`int a = 3;` `a = 2 * a;`	😍	*a variable can be reassigned* *(that's why it's called a variable!)*
`int a = 2 * a;`	☹️	*a variable must be assigned a value before it can be used in an expression*

Tracing the execution of a program

Q. What does this code fragment do?

A. Let's trace the variables during execution of the code. ← *table of variable values*

```
int a = 100;
int b = 126;
int temp = a;
a = b;
b = temp;
```

this idiom exchanges the values stored in the variables a *and* b

	a	*b*	*temp*
start of code fragment	*undeclared*	*undeclared*	*undeclared*
`int a = 100;`	100	*undeclared*	*undeclared*
`int b = 126;`	100	126	*undeclared*
`int temp = a;`	100	126	100
`a = b;`	126	126	100
`b = temp;`	126	100	100

trace of variables (after each statement)

Data types: quiz 1

What are the values stored in the variables *a* and *b* after the code fragment is executed?

A. 100 and 126.

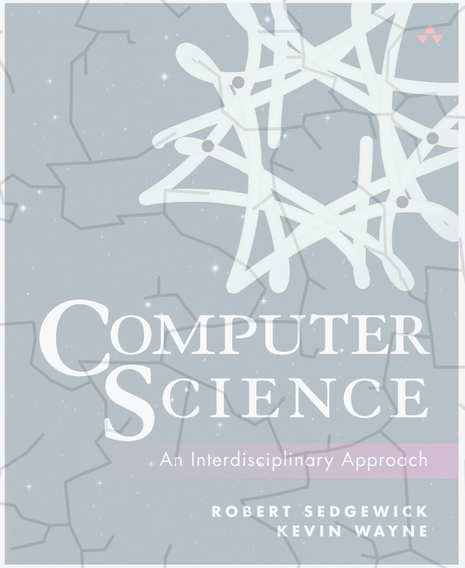
B. 126 and 100.

C. 226 and 126.

D. −26 and −26.

E. Compile-time error.

```
int a = 100;
int b = 126;
a = a + b;
b = a - b;
a = a - b;
```



https://introcs.cs.princeton.edu

Built-In Data Types

- *strings*
- *integers*
- *floating-point numbers*
- *booleans*
- *type conversion*

The *String* data type

Typical usage. Program input and output; text processing.

values	*sequences of characters*
example literals	`"Hi"` `"1234"` `"Nǐ hǎo"` `"💩💩💩"`
operation	*concatenation*
operator	`+`

expression	value	remark
`"My " + "Precious"`	`"My Precious"`	*spaces within a string literal matter*
`"1234" + "99"`	`"123499"`	*strings are not integers*
`"A" + "B" + "C"`	`"ABC"`	*can concatenate several strings together, in one expression*
`"ሰላም " + "ልዑል!"`	`"ሰላም ልዑል!"`	*Unicode supported*

Command-line arguments are strings

Command-line arguments. The variables *args*[0], *args*[1], *args*[2], … are of type *String*.
Java initializes them automatically to corresponding values.

we'll revisit notation in Section 1.4 (arrays)

```java
public class CommandLineArguments {
   public static void main(String[] args) {
      String a = args[0];
      String b = args[1];
      String c = args[2];
      String result = a + "-" + b + "-" + c;
      System.out.println(result);
   }
}
```

```
~/cos126/datatypes> java CommandLineArguments A B C
A-B-C
```

args[0]

```
~/cos126/datatypes> java CommandLineArguments do re mi
do-re-mi
```

arguments delimited by whitespace

```
~/cos126/datatypes> java CommandLineArguments
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException:
Index 0 out of bounds for length 0 at
CommandLineArguments.main(CommandLineArguments.java:3)
```

line number of error

Ruler function

```java
public class Ruler {
   public static void main(String[] args) {
      String ruler0 = "0";
      String ruler1 = ruler0 + " 1 " + ruler0;
      String ruler2 = ruler1 + " 2 " + ruler1;
      String ruler3 = ruler2 + " 3 " + ruler2;
      System.out.println(ruler3);
   }
}
```

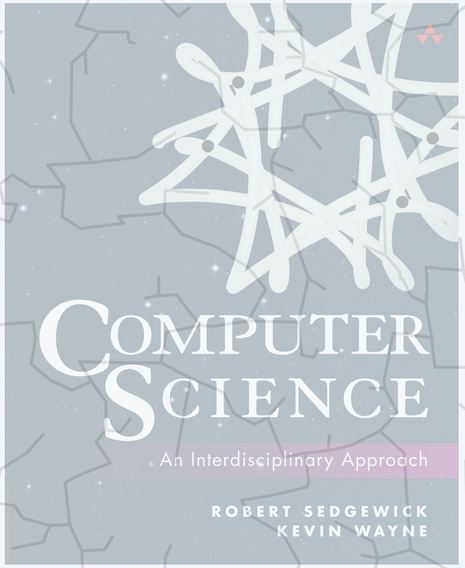
string concatenation

0 1 0 2 0 1 0 3 0 1 0 2 0 1 0

```
~/cos126/datatypes> java Ruler
0 1 0 2 0 1 0 3 0 1 0 2 0 1 0
```

ruler0	*ruler1*	*ruler2*	*ruler3*
undeclared	*undeclared*	*undeclared*	*undeclared*
"0"	*undeclared*	*undeclared*	*undeclared*
"0"	"0 1 0"	*undeclared*	*undeclared*
"0"	"0 1 0"	"0 1 0 2 0 1 0"	*undeclared*
"0"	"0 1 0"	"0 1 0 2 0 1 0"	"0 1 0 2 0 1 0 3 0 1 0 2 0 1 0"

trace of variables (after each statement)



https://introcs.cs.princeton.edu

Built-In Data Types

- *strings*
- *integers*
- *floating-point numbers*
- *booleans*
- *type conversion*

The *int* data type

Typical usage: math calculations involving integers; program control flow.

values	*integers between* -2^{31} *and* $2^{31} - 1$				
example literals	1234 99 0 1000000 -3				
operations	*add*	*subtract*	*multiply*	*divide*	*remainder*
operators	+	-	*	/	%

only 2^{32} *different* `int` *values*
(not quite the same as integers)

expression	value	remark
20 + 3	23	
20 - 3	17	
20 * 3	60	
20 / 3	6	*drop fractional part*
20 % 3	2	*remainder*
20 / 0	–	*division-by-zero error*
2147483647 + 1	-2147483648	*integer overflow*

$2^{31} - 1$

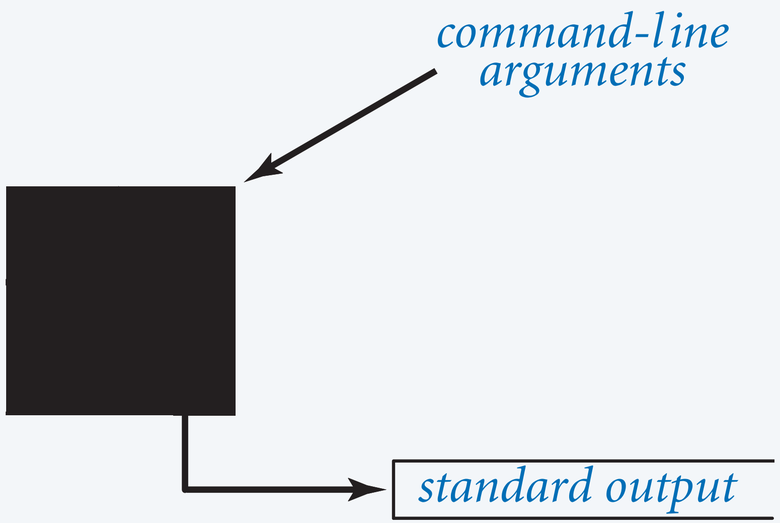
applying an `int` *operator*
to two `int` *operands*
always results in an `int`
(or division-by-zero error)

don't use `int` *with very large integers*

Input and output

Java I/O model. [for now]

- Read strings from the command line.
- Print strings to standard output.



Q. How to read integers from the command line?

A. The system method `Integer.parseInt()` converts from a `String` to an `int`.

Q. How to print integers to standard output?

A. When a `String` is concatenated with an `int`, Java converts the `int` to a `String`.

Input and output with integers

```java
public class IntOps {
   public static void main(String[] args) {
      int a = Integer.parseInt(args[0]);
      int b = Integer.parseInt(args[1]);
      int sum  = a + b;
      int prod = a * b;
      int quot = a / b;
      int rem  = a % b;
      System.out.println(a + " + " + b + " = " + sum);
      System.out.println(a + " * " + b + " = " + prod);
      System.out.println(a + " / " + b + " = " + quot);
      System.out.println(a + " % " + b + " = " + rem);
   }
}
```

converts from String *to* int

converts from int *to* String

```
~/cos126/datatypes> java IntOps 20 3
20 + 3 = 23
20 * 3 = 60
20 / 3 = 6
20 % 3 = 2
```

$20 = 6 \times 3 + 2$

```
~/cos126/datatypes> java IntOps 1234 10
1234 + 10 = 1244
1234 * 10 = 12340
1234 / 10 = 123
1234 % 10 = 4
```

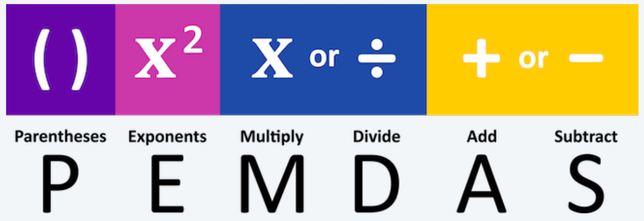
$1234 = 123 \times 10 + 4$

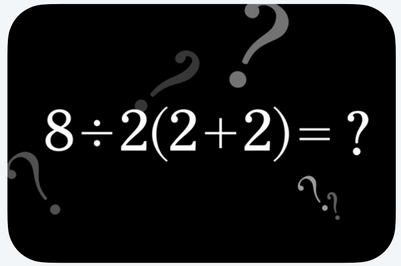
```
~/cos126/datatypes> java IntOps 1234 Hello
Exception in thread "main"
java.lang.NumberFormatException:
For input string: "Hello"
...
at IntOps.main(IntOps.java:4)
```

line number of run-time error

Order of operations

PEMDAS. Rules for evaluating an arithmetic expression.





internet meme

Operator precedence. Priority for grouping operands with operators in an expression.

Operator associativity. Rule for when two operators in an expression have same priority.

expression	equivalent to	value	remark
`3 * 5 - 2`	`(3 * 5) - 2`	13	** has higher precedence than -*
`3 + 5 / 2`	`3 + (5 / 2)`	5	*/ has higher precedence than +*
`3 - 5 - 2`	`(3 - 5) - 2`	-4	*left-to-right associative*
`(3 - 5) - 2`	*itself*	-4	*better style*
`8 / 2 * (2 + 2)`	`(8 / 2) * (2 + 2)`	16	*left-to-right associative* (*multiplication and division have same precedence*)

Data types: quiz 2

What value does the following expression evaluate to?

```
1 + 2 + "ABC" + 3 + 4
```

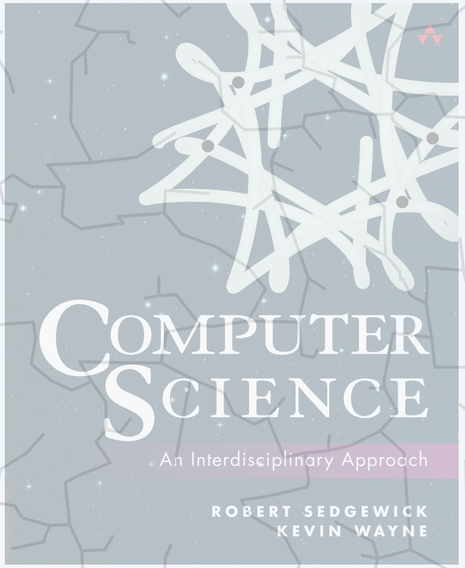
A. "12ABC34"

B. "3ABC7"

C. "3ABC34"

D. "12ABC7"

E. Compile-time error.



https://introcs.cs.princeton.edu

Built-In Data Types

- *strings*
- *integers*
- *floating-point numbers*
- *booleans*
- *type conversion*

The *double* data type

Typical usage: scientific calculations involving real numbers.

values	*IEEE floating-point numbers*				
example literals	18.25 -2.0 1.4142135623730951 6.022E23				
operations	*add*	*subtract*	*multiply*	*divide*	*remainder*
operators	+	-	*	/	%

only 2^{64} *different* `double` *values*
(not quite the same as real numbers)

6.022×10^{23}
(scientific notation)

expression	value	remark
1.5 + 0.25	1.75	
1.5 - 0.25	1.25	
1.5 * 2.0	3.0	
5.0 / 3.0	1.6666666666666667	*not exactly* $\frac{5}{3}$
-1.0 / 0.0	-Infinity	*not an error*
0.0 / 0.0	NaN	*"not a number"*

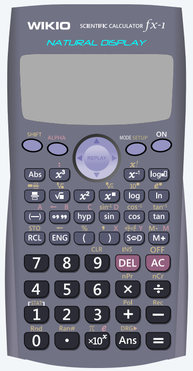
applying a `double` *operator*
to two `double` *operands*
always results in a `double`
(never results in an error)

only binary fractional values
can be represented exactly, such as
$\frac{1}{4} + \frac{1}{16} + \frac{1}{128} = 0.3203125$
(but not $\frac{5}{3}$, $\frac{1}{10}$, *or* π*)*

Excepts from Java's *Math* library

Math library function	description
`static double abs(double a)`	*absolute value of* a
`static double max(double a, double b)`	*maximum of* a *and* b
`static double min(double a, double b)`	*minimum of* a *and* b
`static double sin(double theta)`	*sine* ($\sin \theta$)
`static double cos(double theta)`	*cosine* ($\cos \theta$)
`static double tan(double theta)`	*tangent* ($\tan \theta$)
`static double exp(double a)`	*exponential* (e^a)
`static double log(double a)`	*natural logarithm* ($\log_e a$)
`static double sqrt(double a)`	*positive square root* ($\sqrt{a}$)
`static double pow(double a, double b)`	*power* (a^b)
`static long round(double a)`	*round to the nearest integer*
`static double random()`	*pseudorandom number in* $[0, 1)$
`static double E`	*value of e (constant)*
`static double PI`	*value of* π *(constant)*

← *also defined for* `int`

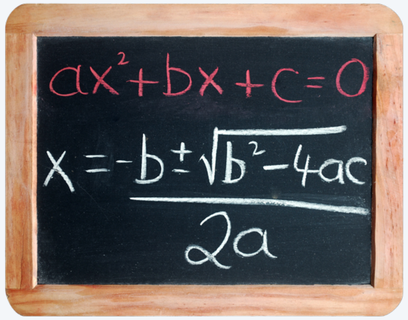


You can discard your calculator now (please).

expression	value
`Math.max(1.0, 2.5)`	2.5
`Math.cos(0.0)`	1.0
`Math.sqrt(2.0)`	1.4142135623730951
`Math.random()`	0.7707780210347349
`Math.PI`	3.141592653589793

Quadratic equation

Goal. Print the solutions to the equation $ax^2 + bx + c = 0$, assuming $a \neq 0$.




```java
public class Quadratic {
   public static void main(String[] args) {

      // Parse coefficients from command-line.
      double a = Double.parseDouble(args[0]);
      double b = Double.parseDouble(args[1]);
      double c = Double.parseDouble(args[2]);

      // Calculate roots of ax^2 + bx + c = 0.
      double discriminant = b*b - 4.0*a*c;
      double d = Math.sqrt(discriminant);
      double root1 = (-b + d) / (2.0*a);
      double root2 = (-b - d) / (2.0*a);

      // Print the two roots.
      System.out.println(root1);
      System.out.println(root2);
   }
}
```

```
~/cos126/datatypes> java Quadratic 1.0 -3.0 2.0
2.0
1.0

~/cos126/datatypes> java Quadratic 1.0 -1.0 -1.0
1.618033988749895
-0.6180339887498949

~/cos126/datatypes> java Quadratic 1.0 1.0 1.0
NaN
NaN
```

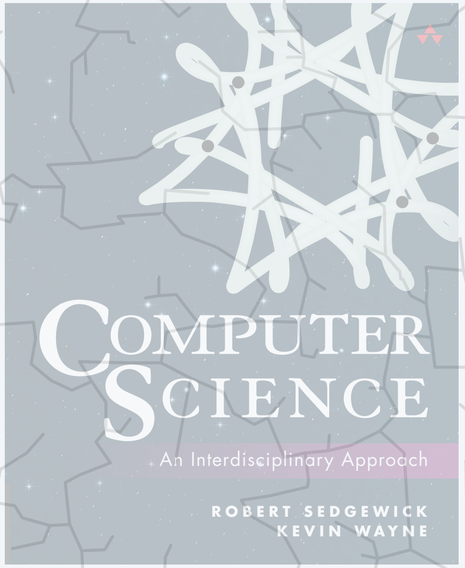
$\longleftarrow \dfrac{1 \pm \sqrt{5}}{2}$

$\longleftarrow \dfrac{-1 \pm 3i}{2}$

$x^2 - 3x + 2$

$x^2 - x - 1$

$x^2 + x + 1$



https://introcs.cs.princeton.edu

Built-In Data Types

- *strings*
- *integers*
- *floating-point numbers*
- *booleans*
- *type conversion*

The *boolean* data type

Typical usage: decision making in a program. ← *stay tuned for conditionals and loops*

values	*true and false*		
literals	`true` `false`		
operations	*not*	*and*	*or*
operators	`!`	`&&`	`||`

← *logical operators*

expression	value
`!false`	`true`
`!true`	`false`

truth table for NOT

expression	value
`false && false`	`false`
`false && true`	`false`
`true && false`	`false`
`true && true`	`true`

truth table for AND

expression	value
`false || false`	`false`
`false || true`	`true`
`true || false`	`true`
`true || true`	`true`

truth table for OR

Boolean meme

Equality and comparison operators

Equality and comparison operators. To compare numeric values.

- Operands: two numeric expressions. ← *can be literals, variable, or arbitrary expressions*
- Evaluates to: a value of type *boolean*.

operator	meaning	true	false
==	*equal*	2 == 2	2 == 3
!=	*not equal*	3 != 2	2 != 2
<	*less than*	2 < 13	13 < 2
<=	*less than or equal*	2 <= 2	3 <= 2
>	*greater than*	13 > 2	2 > 13
>=	*greater than or equal*	2 >= 2	2 >= 3

equality and comparison operators in Java

Equality and comparison operators: examples

zero denominator?	`denominator == 0`
non-negative discriminant?	`(b*b - 4.0*a*c) >= 0.0`
divisible by 60?	`(minutes % 60) == 0`
RGB color is not black?	`(red > 0) || (green > 0) || (blue > 0)`
valid month?	`(month >= 1) && (month <= 12)`
invalid month?	`!((month >= 1) && (month <= 12))`
floating-point roundoff error	`(0.1 * 3.0) == 0.3`
string equality	`args[0] == "Hello"`

parentheses for clarity: arithmetic operators have higher precedence than equality/comparison operators

compound boolean expressions

don't do this! (*evaluates to* `false`)

or this! (*always evaluates to* `false`)

Example of computing with booleans: leap year test

Q. Is a given year a leap year? ← *Gregorian calendar*

A. Yes if **either**: (Case A:) divisible by 400 or (Case B:) divisible by 4 but not 100.

```java
public class LeapYear {
   public static void main(String[] args) {

      int year = Integer.parseInt(args[0]);
      boolean isLeapYear;

      // Case B: divisible by 4 but not 100
      isLeapYear = (year % 4 == 0) && (year % 100 != 0);

      // ...or Case A: divisible by 400
      isLeapYear = isLeapYear || (year % 400 == 0);

      System.out.println(isLeapYear);

   }
}
```

if argument to `System.out.println()` *is of type* `boolean`*,*
it prints either `true` *or* `false`

```
~/cos126/datatypes> java LeapYear 2024
true

~/cos126/datatypes> java LeapYear 2023
false

~/cos126/datatypes> java LeapYear 1900
false

~/cos126/datatypes> java LeapYear 2000
true
```

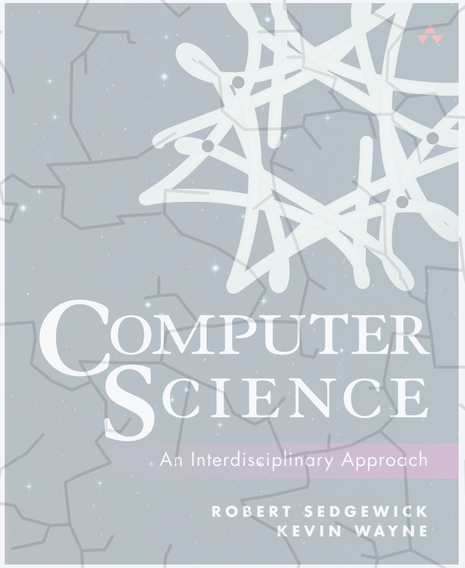
Data types: quiz 3

What does the following expression evaluate to?

```
1 <= month <= 12
```

A. Works: equivalent to `(month >= 1) && (month <= 12)`.

B. Compile-time error: equivalent to `(1 <= month) <= 12`.



https://introcs.cs.princeton.edu

Built-In Data Types

- *strings*
- *integers*
- *floating-point numbers*
- *booleans*
- *type conversion*

Data types

Types limit the allowable operations on values and determine the meaning of those operations.

```java
public class StringMultiply {
   public static void main(String[] args) {
      String s = "123" * "456";
   }
}
```

```
~/cos126/datatypes> javac StringMultiply.java
StringMultiply.java:3: error: bad operand types
for binary operator '*'
      String s = "123" * "456";
                         ^
  first type:  String
  second type: String
1 error
```

Java compiler. The compiler checks for type mismatch errors in your code.

Data types

Types limit the allowable operations on values and determine the meaning of those operations.

operator	int	double	boolean	String
+	*addition*	*addition*	*no*	*concatenation*
-	*subtraction*	*subtraction*	*no*	*no*
*	*multiplication*	*multiplication*	*no*	*no*
/	*integer division*	*division*	*no*	*no*
&&	*no*	*no*	*logical AND*	*no*
||	*no*	*no*	*logical OR*	*no*
!	*no*	*no*	*logical NOT*	*no*
<	*less than*	*less than*	*no*	*no*
⋮	⋮	⋮	⋮	⋮

can't subtract, multiply, or divide two `String` *or* `boolean` *values (compile-time errors)*

Static typing. Every Java variable and expression has a type that is known at compile time.

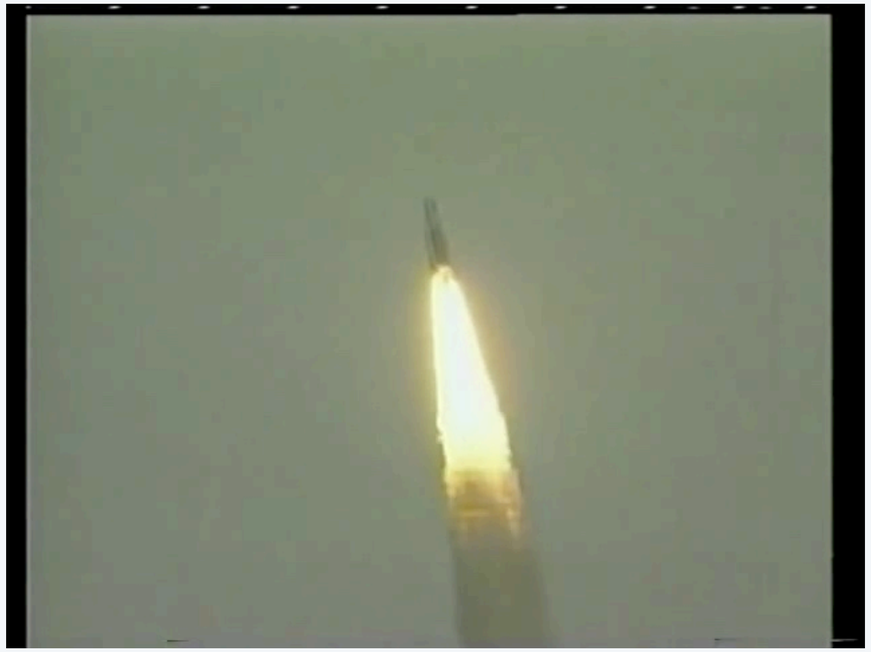
- Benefit: compiler catches entire class of programming errors automatically.
- Drawback: extra boilerplate code.

Type-conversion catastrophe

Ariane 5 rocket.

- European Space Agency spent a decade and $7 billion in research and development.
- Rocket self-destructed 39 seconds after first launch.
- Source of bug: unsafe type conversion of 64-bit floating-point number to 16-bit integer.

code worked fine in Ariane 4
(but Ariane 5 velocity was much higher)



https://www.youtube.com/watch?v=PK_yguLapgA

Type conversions with built-in types

Type conversion is an essential aspect of programming.

Automatic type conversions.

- String conversion: from any type to *String* (via string concatenation).
- Numeric promotion: from *int* to *double* (when a *double* is expected).

every `int` *can be exactly represented as a* `double`

expression	type	value
`"x = " + 99`	`String`	`"x = 99"`
`11 * 0.25`	`double`	`2.75`

System methods.

- *Integer.parseInt()* from *String* to *int*.
- *Double.parseDouble()* from *String* to *double*.

expression	type	value
`Integer.parseInt("126")`	`int`	`126`
`Double.parseDouble("2.5")`	`double`	`2.5`

Explicit casts from one type to another.

- Cast from *double* to *int*. ← *discards fractional part*
- Cast from *int* to *double*.

expression	type	value
`(int) 2.71828;`	`int`	`2`
`(double) sum / n;`	`double`	*average*

cast operator

cast has higher precedence

two `int` *variables*

Example of type conversion

Q. What is type and value of each expression on the left?

expression	type	value	remark
`(7 / 2) * 2.0`	`double`	`6.0`	*integer division; then promotion to* `double`
`(7 / 2.0) * 2`	`double`	`7.0`	*promotion to* `double`*; then floating-point division*
`"12" + 6`	`String`	`"126"`	*conversion to* `String`
`0 == false`	*compile-time error*		*can't compare* `int` *to* `boolean`

Simulate the rolling of a fair die

Goal. Given an integer $n > 0$, generate a uniformly random integer between 1 and n. ← *each possible integer is equally likely*

n = 6

n = 10

n = 100

Generate pseudo-random integers

Problem. Given an integer $n > 0$, generate a uniformly random integer between 0 and $n - 1$.

Useful system method. *Math.random()* returns a pseudorandom *double* value in $[0, 1)$.

can return `0.0`
can't return `1.0`

not truly random, but close enough for most applications

Approach. Scale to desired range, round down to nearest integer.

`String` *to* `int` (*system method*)

```
public class RandomInt {
   public static void main(String[] args) {

      int n = Integer.parseInt(args[0]);
      double r = Math.random();
      int result = (int) (r * n);

      System.out.println(result);
   }
}
```

`double` *to* `int` (*cast*)

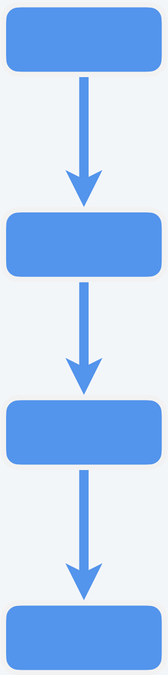
`int` *to* `double` (*automatic*)

```
~/cos126/datatypes> java RandomInt 6
3

~/cos126/datatypes> java RandomInt 6
0

~/cos126/datatypes> java RandomInt 6
5

~/cos126/datatypes> java RandomInt 10000
3184
```

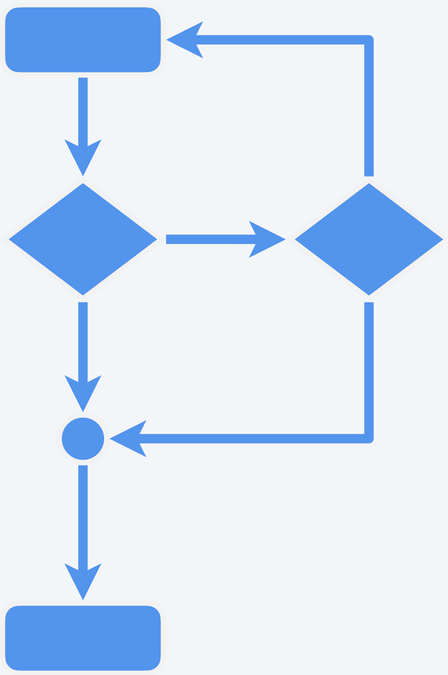
Overview

This lecture. Write programs with declaration, assignment, and print statements.

Next week. Write programs with conditionals and loops.



straight-line control flow



control flow with conditionals and loops

Credits

media	source	license
PEMDAS	Mometrix	
PEMDAS meme	New York Times	
Scientific Calculator	Wikimedia	CC BY-SA 3.0
Solving Quadratic Equations	Adobe Stock	education license
Patriot Missile Launcher	Raytheon	
Incorrectly Calculated Range Gate	GAO	public domain
Scud Missile Hits a U.S. Barracks	New York Times	
!FALSE	redbuble.com	
Ariane 5 Rocket Launch	European Space Agency	
Two Red Dice	Wikimedia	public domain
Ten-Sided Die	Wikimedia	CC BY-SA 3.0
Hundred-Sided Die	gameoutonline.com	