

- [Intro to Programming](#)
 - [1. Elements of Programming](#)
 - [1.1 Your First Program](#)
 - [1.2 Built-in Types of Data](#)
 - [1.3 Conditionals and Loops](#)
 - [1.4 Arrays](#)
 - [1.5 Input and Output](#)
 - [1.6 Case Study: PageRank](#)
 - [2. Functions](#)
 - [2.1 Static Methods](#)
 - [2.2 Libraries and Clients](#)
 - [2.3 Recursion](#)
 - [2.4 Case Study: Percolation](#)
 - [3. OOP](#)
 - [3.1 Using Data Types](#)
 - [3.2 Creating Data Types](#)
 - [3.3 Designing Data Types](#)
 - [3.4 Case Study: N-Body](#)
 - [4. Data Structures](#)
 - [4.1 Performance](#)
 - [4.2 Sorting and Searching](#)
 - [4.3 Stacks and Queues](#)
 - [4.4 Symbol Tables](#)
 - [4.5 Case Study: Small World](#)
- [Computer Science](#)
 - [5. Theory of Computing](#)
 - [5.1 Formal Languages](#)
 - [5.2 Turing Machines](#)
 - [5.3 Universality](#)
 - [5.4 Computability](#)
 - [5.5 Intractability](#)
 - [9.9 Cryptography](#)
 - [6. A Computing Machine](#)
 - [6.1 Representing Info](#)
 - [6.2 TOY Machine](#)
 - [6.3 TOY Programming](#)
 - [6.4 TOY Virtual Machine](#)
 - [7. Building a Computer](#)
 - [7.1 Boolean Logic](#)
 - [7.2 Basic Circuit Model](#)
 - [7.3 Combinational Circuits](#)
 - [7.4 Sequential Circuits](#)
 - [7.5 Digital Devices](#)
- [Beyond](#)
 - [8. Systems](#)
 - [8.1 Library Programming](#)
 - [8.2 Compilers](#)
 - [8.3 Operating Systems](#)
 - [8.4 Networking](#)
 - [8.5 Applications Systems](#)
 - [9. Scientific Computation](#)
 - [9.1 Floating Point](#)
 - [9.2 Symbolic Methods](#)
 - [9.3 Numerical Integration](#)
 - [9.4 Differential Equations](#)
 - [9.5 Linear Algebra](#)
 - [9.6 Optimization](#)
 - [9.7 Data Analysis](#)
 - [9.8 Simulation](#)
- Related Booksites



- [Web Resources](#)
 - [FAQ](#)
 - [Data](#)
 - [Code](#)
 - [Errata](#)
 - [Lectures](#)
 - [Appendices](#)
 - [A. Operator Precedence](#)
 - [B. Writing Clear Code](#)
 - [C. Glossary](#)
 - [D. TOY Cheatsheet](#)
 - [E. Matlab](#)
 - [Online Course](#)
 - [Java Cheatsheet](#)
 - [Programming Assignments](#)

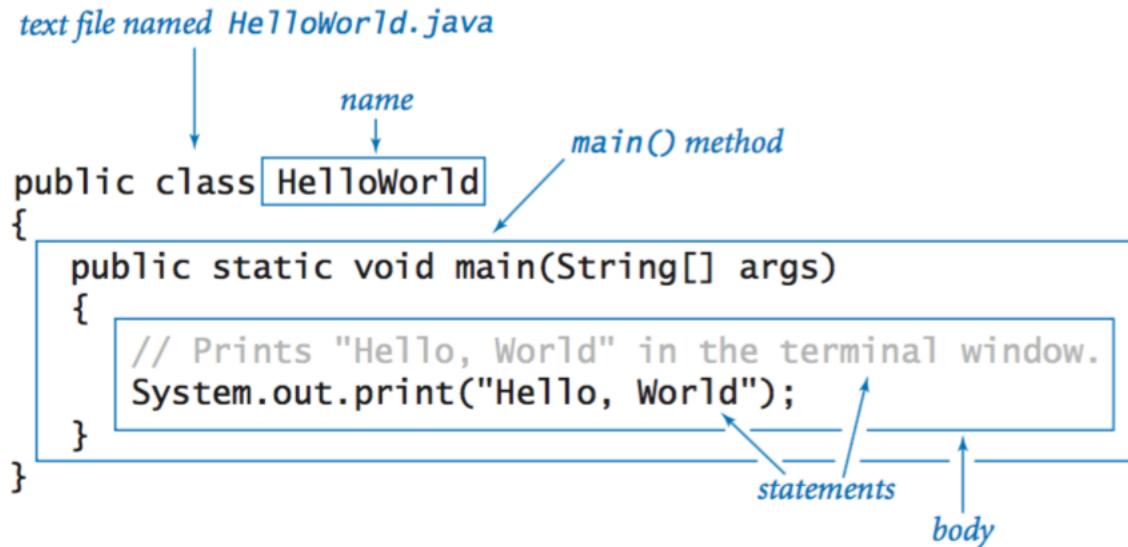
Custom Search



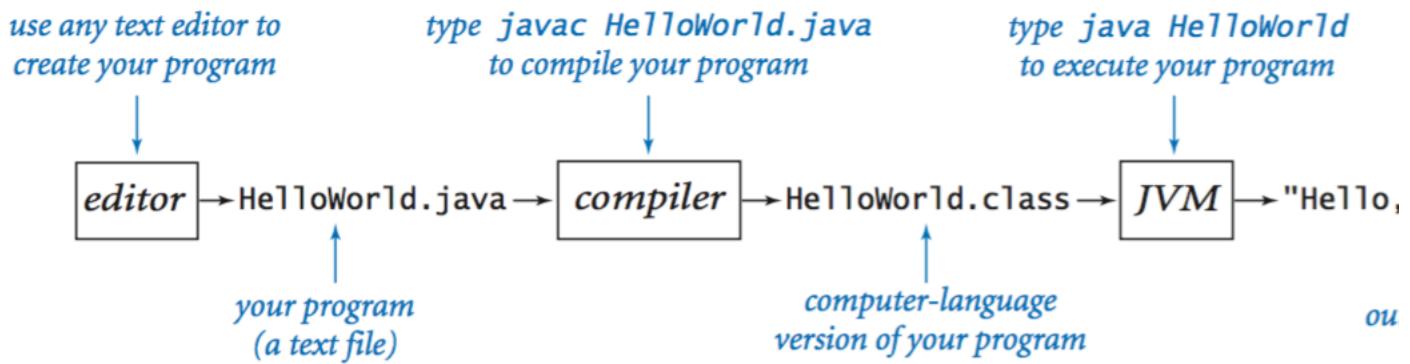
Java Programming Cheatsheet

We summarize the most commonly used Java language features and APIs in the textbook.

Hello, World.



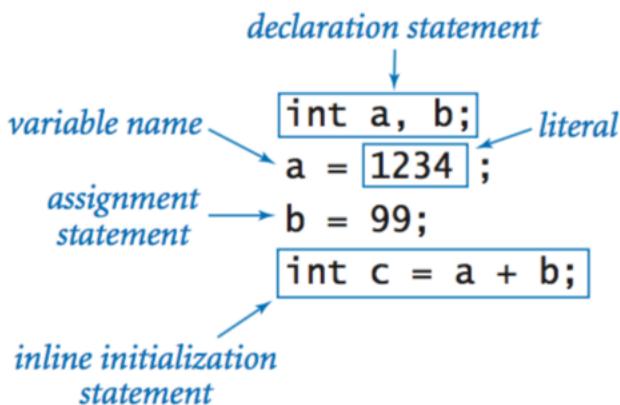
Editing, compiling, and executing.



Built-in data types.

<i>type</i>	<i>set of values</i>	<i>common operators</i>	<i>sample literal values</i>
<code>int</code>	integers	<code>+ - * / %</code>	99 12 21474836
<code>double</code>	floating-point numbers	<code>+ - * /</code>	3.14 2.5 6.022
<code>boolean</code>	boolean values	<code>&& !</code>	true false
<code>char</code>	characters		'A' '1' '%' '\
<code>String</code>	sequences of characters	<code>+</code>	"AB" "Hello" "2

Declaration and assignment statements.



Integers.

<i>values</i>	integers between -2^{31} and $+2^{31}-1$					
<i>typical literals</i>	1234 99 0 1000000					
<i>operations</i>	<i>sign</i>	<i>add</i>	<i>subtract</i>	<i>multiply</i>	<i>divide</i>	<i>remainder</i>
<i>operators</i>	<code>+</code> <code>-</code>	<code>+</code>	<code>-</code>	<code>*</code>	<code>/</code>	<code>%</code>

<i>expression</i>	<i>value</i>	<i>comment</i>
99	99	<i>integer literal</i>
+99	99	<i>positive sign</i>
-99	-99	<i>negative sign</i>
5 + 3	8	<i>addition</i>
5 - 3	2	<i>subtraction</i>
5 * 3	15	<i>multiplication</i>
5 / 3	1	<i>no fractional part</i>
5 % 3	2	<i>remainder</i>
1 / 0		<i>run-time error</i>
3 * 5 - 2	13	* has precedence
3 + 5 / 2	5	/ has precedence
3 - 5 - 2	-4	left associative
(3 - 5) - 2	-4	better style
3 - (5 - 2)	0	unambiguous

Floating-point numbers.

<i>values</i>	real numbers (specified by IEEE 754 standard)			
<i>typical literals</i>	3.14159 6.022e23 2.0 1.4142135623730951			
<i>operations</i>	<i>add</i>	<i>subtract</i>	<i>multiply</i>	<i>divide</i>
<i>operators</i>	+	-	*	/

<i>expression</i>	<i>value</i>
3.141 + 2.0	5.141
3.141 - 2.0	1.111
3.141 / 2.0	1.5705
5.0 / 3.0	1.6666666666666667
10.0 % 3.141	0.577
1.0 / 0.0	Infinity
Math.sqrt(2.0)	1.4142135623730951
Math.sqrt(-1.0)	NaN

Booleans.

<i>values</i>	<i>true or false</i>		
<i>literals</i>	<code>true</code> <code>false</code>		
<i>operations</i>	<code>and</code>	<code>or</code>	<code>not</code>
<i>operators</i>	<code>&&</code>	<code> </code>	<code>!</code>

<i>a</i>	<i>!a</i>	<i>a</i>	<i>b</i>	<i>a && b</i>	<i>a b</i>
<code>true</code>	<code>false</code>	<code>false</code>	<code>false</code>	<code>false</code>	<code>false</code>
<code>false</code>	<code>true</code>	<code>false</code>	<code>true</code>	<code>false</code>	<code>true</code>
		<code>true</code>	<code>false</code>	<code>false</code>	<code>true</code>
		<code>true</code>	<code>true</code>	<code>true</code>	<code>true</code>

Comparison operators.

<i>op</i>	<i>meaning</i>	<i>true</i>	<i>false</i>
<code>==</code>	<i>equal</i>	<code>2 == 2</code>	<code>2 == 3</code>
<code>!=</code>	<i>not equal</i>	<code>3 != 2</code>	<code>2 != 2</code>
<code><</code>	<i>less than</i>	<code>2 < 13</code>	<code>2 < 2</code>
<code><=</code>	<i>less than or equal</i>	<code>2 <= 2</code>	<code>3 <= 2</code>
<code>></code>	<i>greater than</i>	<code>13 > 2</code>	<code>2 > 13</code>
<code>>=</code>	<i>greater than or equal</i>	<code>3 >= 2</code>	<code>2 >= 3</code>

non-negative discriminant? $(b*b - 4.0*a*c) \geq 0.0$ *beginning of a century?* $(\text{year} \% 100) == 0$ *legal month?* $(\text{month} \geq 1) \&\& (\text{month} \leq 12)$ **Printing.**

<code>void System.out.print(String s)</code>	<i>prints</i>
<code>void System.out.println(String s)</code>	<i>prints s, followed by a newline</i>
<code>void System.out.println()</code>	<i>prints a newline</i>

Parsing command-line arguments.

<code>int Integer.parseInt(String s)</code>	<i>convert s to an int value</i>
<code>double Double.parseDouble(String s)</code>	<i>convert s to a double value</i>
<code>long Long.parseLong(String s)</code>	<i>convert s to a long value</i>

Math library.

```
public class Math
```

<code>double abs(double a)</code>	<i>absolute value of a</i>
<code>double max(double a, double b)</code>	<i>maximum of a and b</i>
<code>double min(double a, double b)</code>	<i>minimum of a and b</i>
<code>double sin(double theta)</code>	<i>sine of theta</i>
<code>double cos(double theta)</code>	<i>cosine of theta</i>
<code>double tan(double theta)</code>	<i>tangent of theta</i>
<code>double toRadians(double degrees)</code>	<i>convert angle from degrees to radians</i>
<code>double toDegrees(double radians)</code>	<i>convert angle from radians to degrees</i>
<code>double exp(double a)</code>	<i>exponential (e^a)</i>
<code>double log(double a)</code>	<i>natural log ($\log_e a$, or $\ln a$)</i>
<code>double pow(double a, double b)</code>	<i>raise a to the bth power (a^b)</i>
<code>long round(double a)</code>	<i>round a to the nearest integer</i>
<code>double random()</code>	<i>random number in [0, 1)</i>
<code>double sqrt(double a)</code>	<i>square root of a</i>
<code>double E</code>	<i>value of e (constant)</i>
<code>double PI</code>	<i>value of π (constant)</i>

The full [java.lang.Math API](#).

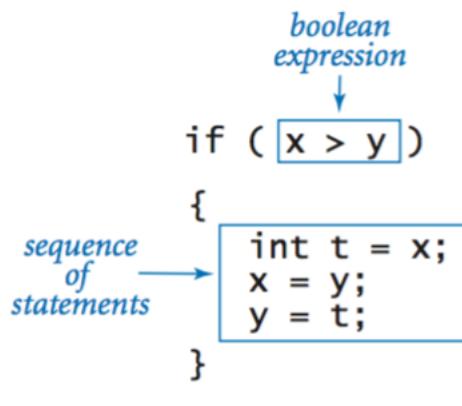
Java library calls.

<i>method call</i>	<i>library</i>	<i>return type</i>	<i>value</i>
<code>Integer.parseInt("123")</code>	<code>Integer</code>	<code>int</code>	123
<code>Double.parseDouble("1.5")</code>	<code>Double</code>	<code>double</code>	1.5
<code>Math.sqrt(5.0*5.0 - 4.0*4.0)</code>	<code>Math</code>	<code>double</code>	3.0
<code>Math.log(Math.E)</code>	<code>Math</code>	<code>double</code>	1.0
<code>Math.random()</code>	<code>Math</code>	<code>double</code>	<i>random in [0, 1)</i>
<code>Math.round(3.14159)</code>	<code>Math</code>	<code>long</code>	3
<code>Math.max(1.0, 9.0)</code>	<code>Math</code>	<code>double</code>	9.0

Type conversion.

<i>expression</i>	<i>expression type</i>	<i>expression value</i>
<code>(1 + 2 + 3 + 4) / 4.0</code>	<code>double</code>	2.5
<code>Math.sqrt(4)</code>	<code>double</code>	2.0
<code>"1234" + 99</code>	<code>String</code>	"123499"
<code>11 * 0.25</code>	<code>double</code>	2.75
<code>(int) 11 * 0.25</code>	<code>double</code>	2.75
<code>11 * (int) 0.25</code>	<code>int</code>	0
<code>(int) (11 * 0.25)</code>	<code>int</code>	2
<code>(int) 2.71828</code>	<code>int</code>	2
<code>Math.round(2.71828)</code>	<code>long</code>	3
<code>(int) Math.round(2.71828)</code>	<code>int</code>	3
<code>Integer.parseInt("1234")</code>	<code>int</code>	1234

Anatomy of an if statement.



If and if-else statements.

<i>absolute value</i>	<pre>if (x < 0) x = -x;</pre>
<i>put the smaller value in x and the larger value in y</i>	<pre>if (x > y) { int t = x; x = y; y = t; }</pre>
<i>maximum of x and y</i>	<pre>if (x > y) max = x; else max = y;</pre>
<i>error check for division operation</i>	<pre>if (den == 0) System.out.println("Division by zero"); else System.out.println("Quotient = " + num/de</pre>
<i>error check for quadratic formula</i>	<pre>double discriminant = b*b - 4.0*c; if (discriminant < 0.0) { System.out.println("No real roots"); } else { System.out.println((-b + Math.sqrt(discriminant))/2); System.out.println((-b - Math.sqrt(discriminant))/2); }</pre>

Nested if-else statement.

```

if      (income <      0) rate = 0.00;
else if (income <  8925) rate = 0.10;
else if (income < 36250) rate = 0.15;
else if (income < 87850) rate = 0.23;
else if (income < 183250) rate = 0.28;
else if (income < 398350) rate = 0.33;
else if (income < 400000) rate = 0.35;
else                                rate = 0.396;

```

Anatomy of a while loop.

```

int power = 1;    ← initialization is a separate statement
while (power <= n/2)    ← loop-continuation condition
{
    power = 2*power;    ← body
}

```

The diagram shows a while loop structure. An annotation 'initialization is a separate statement' points to the declaration 'int power = 1;'. Another annotation 'loop-continuation condition' points to the condition 'power <= n/2'. A brace on the right side of the loop body is labeled 'braces are optional when body is a single statement'. An annotation 'body' points to the line 'power = 2*power;'.

Anatomy of a for loop.

```

int power = 1;    ← initialize another variable in a separate statement
for (int i = 0; i <= n; i++)    ← declare and initialize a loop control variable
{
    System.out.println(i + " " + power);    ← body
    power = 2*power;    ← increment
}

```

The diagram shows a for loop structure. An annotation 'initialize another variable in a separate statement' points to the declaration 'int power = 1;'. Another annotation 'declare and initialize a loop control variable' points to the initialization part of the for loop. An annotation 'loop-continuation condition' points to the condition 'i <= n'. An annotation 'increment' points to the increment part 'i++'. An annotation 'body' points to the line 'System.out.println(i + " " + power);'.

Loops.

<i>compute the largest power of 2 less than or equal to n</i>	<pre>int power = 1; while (power <= n/2) power = 2*power; System.out.println(power);</pre>
<i>compute a finite sum ($1 + 2 + \dots + n$)</i>	<pre>int sum = 0; for (int i = 1; i <= n; i++) sum += i; System.out.println(sum);</pre>
<i>compute a finite product ($n! = 1 \times 2 \times \dots \times n$)</i>	<pre>int product = 1; for (int i = 1; i <= n; i++) product *= i; System.out.println(product);</pre>
<i>print a table of function values</i>	<pre>for (int i = 0; i <= n; i++) System.out.println(i + " " + 2*Math.PI*i/n);</pre>
<i>compute the ruler function (see PROGRAM 1.2.1)</i>	<pre>String ruler = "1"; for (int i = 2; i <= n; i++) ruler = ruler + " " + i + " " + ruler; System.out.println(ruler);</pre>

Break statement.

```
int factor;
for (factor = 2; factor <= n/factor; factor++)
    if (n % factor == 0) break;

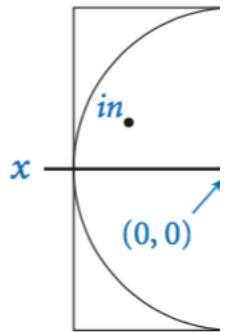
if (factor > n/factor)
    System.out.println(n + " is prime");
```

Do-while loop.

```

do
{ // Scale x and y to be random in (-1, 1).
    x = 2.0*Math.random() - 1.0;
    y = 2.0*Math.random() - 1.0;
} while (Math.sqrt(x*x + y*y) > 1.0);

```



Switch statement.

```

switch (day) {
    case 0: System.out.println("Sun"); break;
    case 1: System.out.println("Mon"); break;
    case 2: System.out.println("Tue"); break;
    case 3: System.out.println("Wed"); break;
    case 4: System.out.println("Thu"); break;
    case 5: System.out.println("Fri"); break;
    case 6: System.out.println("Sat"); break;
}

```

Arrays.

a	a[0]
	a[1]
	a[2]
	a[3]
	a[4]
	a[5]
	a[6]
	a[7]

Inline array initialization.

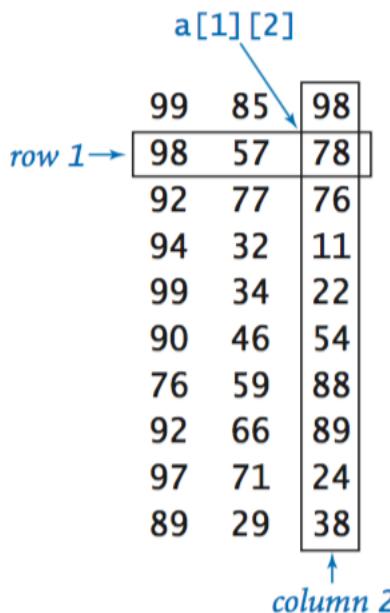
```
String[] SUITS = { "Clubs", "Diamonds", "Hearts", "Spades" };

String[] RANKS = {
    "2", "3", "4", "5", "6", "7", "8", "9", "10",
    "Jack", "Queen", "King", "Ace"
};
```

Typical array-processing code.

<i>create an array with random values</i>	<pre>double[] a = new double[n]; for (int i = 0; i < n; i++) a[i] = Math.random();</pre>
<i>print the array values, one per line</i>	<pre>for (int i = 0; i < n; i++) System.out.println(a[i]);</pre>
<i>find the maximum of the array values</i>	<pre>double max = Double.NEGATIVE_INFINITY; for (int i = 0; i < n; i++) if (a[i] > max) max = a[i];</pre>
<i>compute the average of the array values</i>	<pre>double sum = 0.0; for (int i = 0; i < n; i++) sum += a[i]; double average = sum / n;</pre>
<i>reverse the values within an array</i>	<pre>for (int i = 0; i < n/2; i++) { double temp = a[i]; a[i] = a[n-1-i]; a[n-i-1] = temp; }</pre>
<i>copy sequence of values to another array</i>	<pre>double[] b = new double[n]; for (int i = 0; i < n; i++) b[i] = a[i];</pre>

Two-dimensional arrays.



Inline initialization.

```
double [][] a =
{
    { 99.0, 85.0, 98.0, 0.0 },
    { 98.0, 57.0, 79.0, 0.0 },
    { 92.0, 77.0, 74.0, 0.0 },
    { 94.0, 62.0, 81.0, 0.0 },
    { 99.0, 94.0, 92.0, 0.0 },
    { 80.0, 76.5, 67.0, 0.0 },
    { 76.0, 58.5, 90.5, 0.0 },
    { 92.0, 66.0, 91.0, 0.0 },
    { 97.0, 70.5, 66.5, 0.0 },
    { 89.0, 89.5, 81.0, 0.0 },
    { 0.0, 0.0, 0.0, 0.0 }
};
```

Our standard output library.

```
public class StdOut

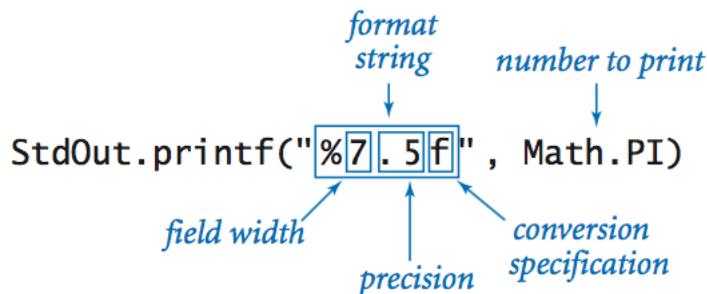
    void print(String s)
    void println(String s)
    void println()
    void printf(String format, ... )
```

print s to standard output

print s and a newline to standard output

print a newline to standard output

print the arguments to standard output as specified by the format string format



<i>type</i>	<i>code</i>	<i>typical literal</i>	<i>sample format strings</i>	<i>converted string values for output</i>	
int	d	512	"%14d" "%-14d"	"	512"
double	f	1595.1680010754388	%14.2f %.7f %14.4e	1595.17	
	e			"1595.1680011" "1.5952e+03"	
String	s	"Hello, World"	"%14s" "%-14s" "%-14.5s"	Hello, World	
boolean	b	true	%b	true	

Our standard input library.

public class StdIn

methods for reading individual tokens from standard input

<code>boolean isEmpty()</code>	<i>is standard input empty (or only whitespace)?</i>
<code>int readInt()</code>	<i>read a token, convert it to an int, and return it</i>
<code>double readDouble()</code>	<i>read a token, convert it to a double, and return it</i>
<code>boolean readBoolean()</code>	<i>read a token, convert it to a boolean, and return it</i>
<code>String readString()</code>	<i>read a token and return it as a String</i>

methods for reading characters from standard input

<code>boolean hasNextChar()</code>	<i>does standard input have any remaining characters?</i>
<code>char readChar()</code>	<i>read a character from standard input and return it</i>

methods for reading lines from standard input

<code>boolean hasNextLine()</code>	<i>does standard input have a next line?</i>
<code>String readLine()</code>	<i>read the rest of the line and return it as a String</i>

methods for reading the rest of standard input

<code>int[] readAllInts()</code>	<i>read all remaining tokens and return them as an array of ints</i>
<code>double[] readAllDoubles()</code>	<i>read all remaining tokens and return them as an array of doubles</i>
<code>boolean[] readAllBooleans()</code>	<i>read all remaining tokens and return them as an array of booleans</i>
<code>String[] readAllStrings()</code>	<i>read all remaining tokens and return them as an array of strings</i>
<code>String[] readAllLines()</code>	<i>read all remaining lines and return them as a String array</i>
<code>String readAll()</code>	<i>read the rest of the input and return it as a String</i>

The full [StdIn API](#).

Our standard drawing library.

public class StdDraw

drawing commands

```
void line(double x0, double y0, double x1, double y1)
void point(double x, double y)
void circle(double x, double y, double radius)
void filledCircle(double x, double y, double radius)
void square(double x, double y, double radius)
void filledSquare(double x, double y, double radius)
void rectangle(double x, double y, double r1, double r2)
void filledRectangle(double x, double y, double r1, double r2)
void polygon(double[] x, double[] y)
void filledPolygon(double[] x, double[] y)
void text(double x, double y, String s)
```

control commands

void setXscale(double x0, double x1)	<i>reset x-scale to (x0, x1)</i>
void setYscale(double y0, double y1)	<i>reset y-scale to (y0, y1)</i>
void setPenRadius(double radius)	<i>set pen radius to radius</i>
void setPenColor(Color color)	<i>set pen color to color</i>
void setFont(Font font)	<i>set text font to font</i>
void setCanvasSize(int w, int h)	<i>set canvas size to w-by-h</i>
void enableDoubleBuffering()	<i>enable double buffering</i>
void disableDoubleBuffering()	<i>disable double buffering</i>
void show()	<i>copy the offscreen canvas to the onscreen canvas</i>
void clear(Color color)	<i>clear the canvas to color color</i>
void pause(int dt)	<i>pause dt milliseconds</i>
void save(String filename)	<i>save to a .jpg or .png file</i>

The full [StdDraw API](#).

Our standard audio library.

```
public class StdAudio
```

void play(String filename)	<i>play the given .wav file</i>
void play(double[] a)	<i>play the given sound wave</i>
void play(double x)	<i>play sample for 1/44100 s</i>
void save(String filename, double[] a)	<i>save to a .wav file</i>
double[] read(String filename)	<i>read from a .wav file</i>

The full [StdAudio API](#).

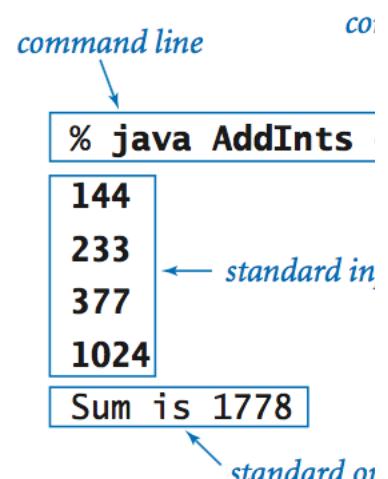
Command line.

```
public class AddInts
{
    public static void main(String[] args)
    {
        int n = Integer.parseInt(args[0]);
        int sum = 0;
        for (int i = 0; i < n; i++) parse command-
        { line argument
            int value = StdIn.readInt();
            sum += value; read from
        } standard input stream
        StdOut.println("Sum is " + sum); print to
    }
}
```

parse command-line argument

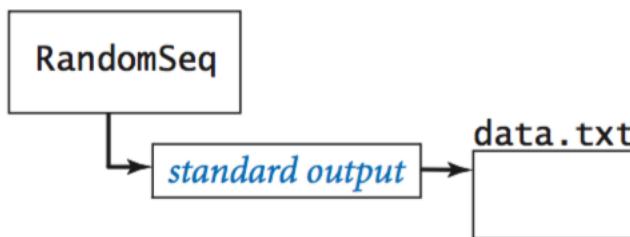
read from standard input stream

print to standard output stream

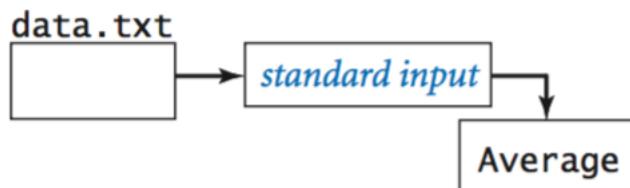


Redirection and piping.

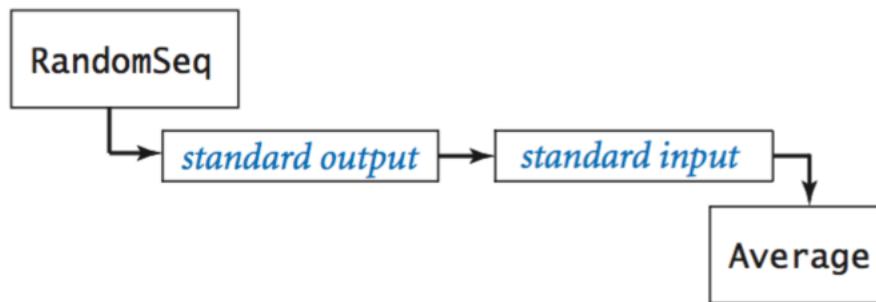
```
% java RandomSeq 1000 > data.txt
```



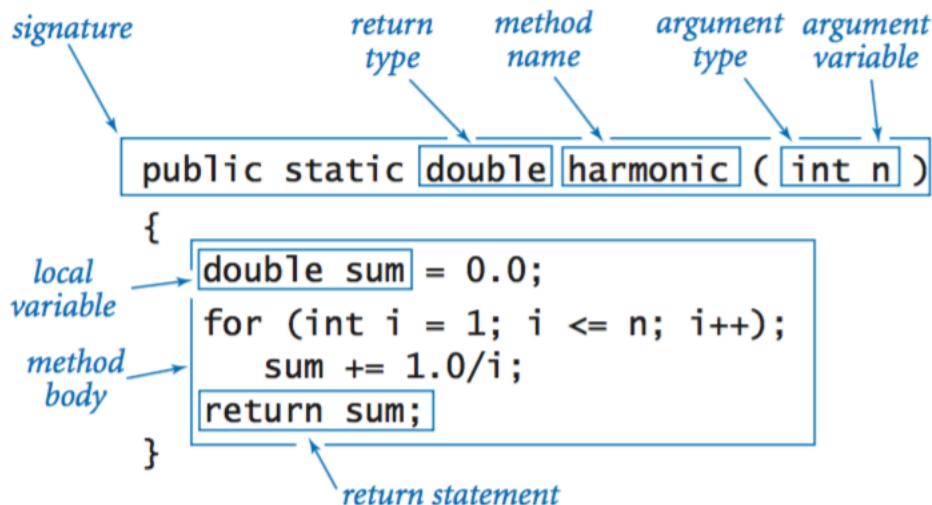
```
% java Average < data.txt
```



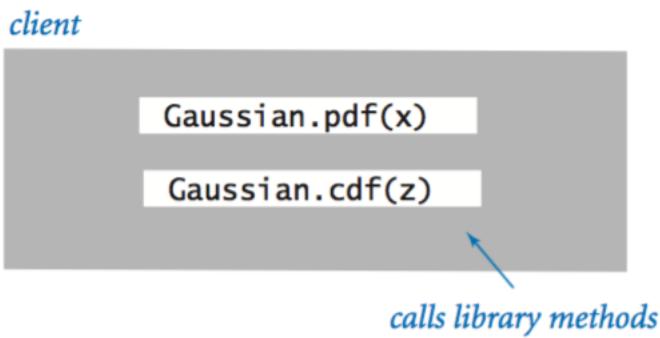
```
% java RandomSeq 1000 | java Average
```



Functions.



<i>absolute value of an int value</i>	<pre>public static int abs(int x) { if (x < 0) return -x; else return x; }</pre>
<i>absolute value of a double value</i>	<pre>public static double abs(double x) { if (x < 0.0) return -x; else return x; }</pre>
<i>primality test</i>	<pre>public static boolean isPrime(int n) { if (n < 2) return false; for (int i = 2; i <= n/i; i++) if (n % i == 0) return false; return true; }</pre>
<i>hypotenuse of a right triangle</i>	<pre>public static double hypotenuse(double a, double b) { return Math.sqrt(a*a + b*b); }</pre>
<i>harmonic number</i>	<pre>public static double harmonic(int n) { double sum = 0.0; for (int i = 1; i <= n; i++) sum += 1.0 / i; return sum; }</pre>
<i>uniform random integer in [0, n)</i>	<pre>public static int uniform(int n) { return (int) (Math.random() * n); }</pre>
<i>draw a triangle</i>	<pre>public static void drawTriangle(double x0, double y0, double x1, double y1, double x2, double y2) { StdDraw.line(x0, y0, x1, y1); StdDraw.line(x1, y1, x2, y2); StdDraw.line(x2, y2, x0, y0); }</pre>

Libraries of functions.*API*

```

public class Gaussian
{
    double pdf(double x)   φ(x)
    double cdf(double z)   Φ(z)
}

```

defines signatures and describes library methods

implementation

```

public class Gaussian
{
    ...
    public static double pdf(double x)
    {
        ...
    }

    public static double cdf(double z)
    {
        ...
    }
}


```

Java code that implements library methods

Our standard random library.

public class StdRandom

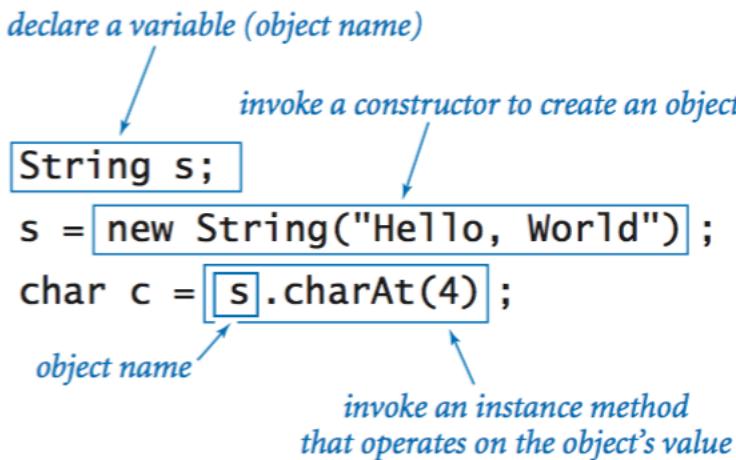
<code>void setSeed(long seed)</code>	<i>set the seed for reproducible results</i>
<code>int uniform(int n)</code>	<i>integer between 0 and n - 1</i>
<code>double uniform(double lo, double hi)</code>	<i>real between lo and hi</i>
<code>boolean bernoulli(double p)</code>	<i>true with probability p</i>
<code>double gaussian()</code>	<i>normal, mean 0, standard deviation 1</i>
<code>double gaussian(double mu, double sigma)</code>	<i>normal, mean mu, standard deviation sigma</i>
<code>int discrete(double[] probabilities)</code>	<i>i with probability probabilities[i]</i>
<code>void shuffle(double[] a)</code>	<i>randomly shuffle the array a</i>

Our standard statistics library.

public class StdStats

<code>double max(double[] a)</code>	<i>largest value</i>
<code>double min(double[] a)</code>	<i>smallest value</i>
<code>double mean(double[] a)</code>	<i>average</i>
<code>double var(double[] a)</code>	<i>sample variance</i>
<code>double stddev(double[] a)</code>	<i>sample standard deviation</i>
<code>double median(double[] a)</code>	<i>median</i>
<code>void plotPoints(double[] a)</code>	<i>plot points at (i, a[i])</i>
<code>void plotLines(double[] a)</code>	<i>plot lines connecting (i, a[i])</i>
<code>void plotBars(double[] a)</code>	<i>plot bars to points at (i, a[i])</i>

Using an object.



Instance variables.

```

public class Charge
{
    private final double rx, ry;
    private final double q;
    ...
}

```

instance variable declarations

access modifiers

Constructors.

```

public Charge ( double x0 , double y0 , double q0 )
{
    rx = x0;
    ry = y0;
    q = q0;
}

```

access modifier

no return type

constructor name (same as class name)

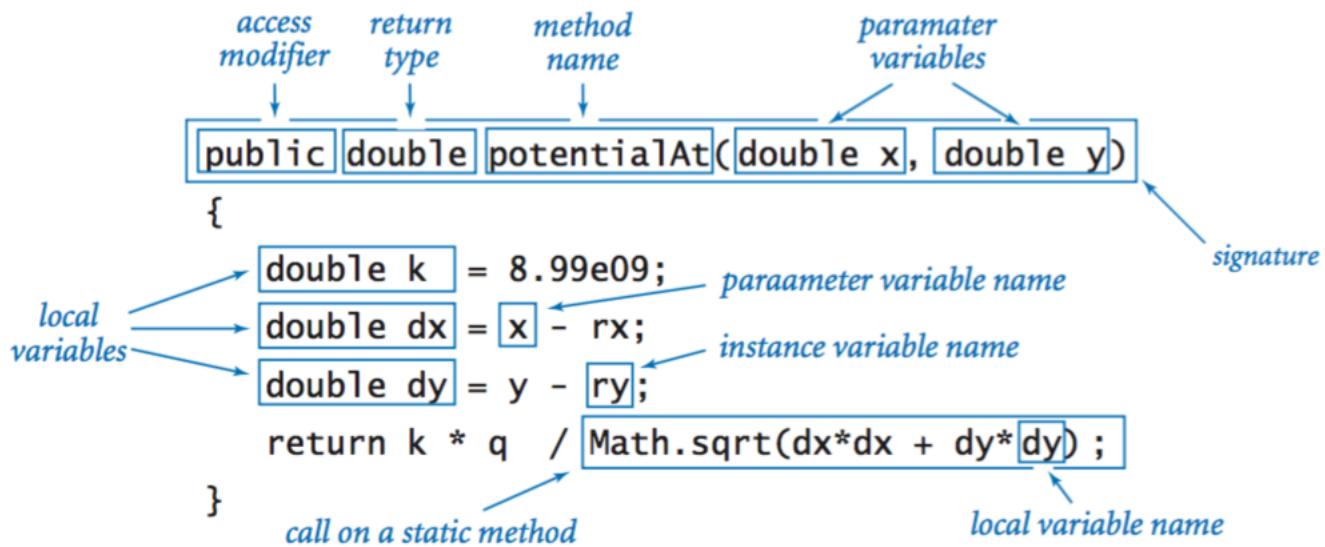
parameter variables

body of constructor

signature

instance variable names

Instance methods.



Classes.

```

public class Charge {
    private final double rx, ry;
    private final double q;

    public Charge(double x0, double y0, double q0)
    {   rx = x0; ry = y0; q = q0; }

    public double potentialAt(double x, double y)
    {
        double k = 8.99e09;
        double dx = x - rx;
        double dy = y - ry;
        return k * q / Math.sqrt(dx*dx + dy*dy);
    }

    public String toString()
    {   return q + " at " + "(" + rx + ", " + ry + ")"; }

    public static void main(String[] args)
    {
        double x = Double.parseDouble(args[0]);
        double y = Double.parseDouble(args[1]);
        Charge c1 = new Charge(0.51, 0.63, 21.3);
        Charge c2 = new Charge(0.13, 0.94, 81.9);
        double v1 = c1.potentialAt(x, y);
        double v2 = c2.potentialAt(x, y);
        StdOut.printf("%.2e\n", (v1 + v2));
    }
}

```

Annotations:

- instance variables**: Points to the declaration of `rx`, `ry`, and `q`.
- constructor**: Points to the constructor `Charge(double x0, double y0, double q0)`.
- instance methods**: Points to the method `potentialAt(double x, double y)`.
- test client**: Points to the `main` method.
- create and initialize object**: Points to the creation of objects `c1` and `c2` using the `new` keyword.
- object name**: Points to the variable `c1` and `c2`.
- class name**: Points to the class name `Charge`.
- instance variable names**: Points to the variable names `rx`, `ry`, and `q` within the `potentialAt` method.
- invoke constructor**: Points to the call to the constructor `new Charge(0.51, 0.63, 21.3)`.
- invoke method**: Points to the calls to the `potentialAt` method on `c1` and `c2`.

Object-oriented libraries.

client

```
Charge c1 = new Charge(0.51, 0.63, 21.3);
c1.potentialAt(x, y)
```

*creates objects
and invokes methods*

API

```
public class Charge
{
    Charge(double x0, double y0, double q0)
    double potentialAt(double x, double y) potential at (x, y)  
due to charge
    String toString() string  
representation
}
```

*defines signatures
and describes methods*

implementation

```
public class Charge
{
    private final double rx, ry;
    private final double q;

    public Charge(double x0, double y0, double q0)
    {
        ...
    }

    public double potentialAt(double x, double y)
    {
        ...
    }

    public String toString()
    {
        ...
    }
}
```

*defines instance variables
and implements methods*

Java's String data type.

public class String

<code>String(String s)</code>	<i>create a string with the same value as s</i>
<code>String(char[] a)</code>	<i>create a string that represents the sequence of characters as in a[]</i>
<code>int length()</code>	<i>number of characters</i>
<code>char charAt(int i)</code>	<i>the character at index i</i>
<code>String substring(int i, int j)</code>	<i>characters at indices i through j-1</i>
<code>boolean contains(String substring)</code>	<i>does this string contain substring?</i>
<code>boolean startsWith(String prefix)</code>	<i>does this string start with prefix?</i>
<code>boolean endsWith(String postfix)</code>	<i>does this string end with postfix?</i>
<code>int indexOf(String pattern)</code>	<i>index of first occurrence of pattern</i>
<code>int indexOf(String pattern, int i)</code>	<i>index of first occurrence of pattern, starting at index i</i>
<code>String concat(String t)</code>	<i>this string, with t appended</i>
<code>int compareTo(String t)</code>	<i>string comparison</i>
<code>String toLowerCase()</code>	<i>this string, with lowercase letters</i>
<code>String toUpperCase()</code>	<i>this string, with uppercase letters</i>
<code>String replace(String a, String b)</code>	<i>this string, with a replaced by b</i>
<code>String trim()</code>	<i>this string, with leading and trailing whitespace removed</i>
<code>boolean matches(String regexp)</code>	<i>is this string matched by the regular expression?</i>
<code>String[] split(String delimiter)</code>	<i>strings between occurrences of delimiter</i>
<code>boolean equals(Object t)</code>	<i>is this string's value the same as t?</i>
<code>int hashCode()</code>	<i>an integer hash code</i>

The full [java.lang.String API](#).

```
String a = new String("now is");
String b = new String("the time");
String c = new String(" the");
```

<i>instance method call</i>	<i>return type</i>	<i>return value</i>
a.length()	int	6
a.charAt(4)	char	'i'
a.substring(2, 5)	String	"w i"
b.startsWith("the")	boolean	true
a.indexOf("is")	int	4
a.concat(c)	String	"now is the"
b.replace("t", "T")	String	"The Time"
a.split(" ")	String[]	{ "now", "is" }
b.equals(c)	boolean	false

Java's Color data type.

```
public class java.awt.Color
```

Color(int r, int g, int b)	
int getRed()	<i>red intensity</i>
int getGreen()	<i>green intensity</i>
int getBlue()	<i>blue intensity</i>
Color brighter()	<i>brighter version of this color</i>
Color darker()	<i>darker version of this color</i>
String toString()	<i>string representation of this color</i>
String equals(Object c)	<i>is this color's value the same as c ?</i>

The full [java.awt.Color API](#).

Our input library.

public class In

In() *create an input stream from standard input*

In(String name) *create an input stream from a file or website*

instance methods that read individual tokens from the input stream

boolean isEmpty() *is standard input empty (or only whitespace)?*

int readInt() *read a token, convert it to an int, and return it*

double readDouble() *read a token, convert it to a double, and return it*

...

instance methods that read characters from the input stream

boolean hasNextChar() *does standard input have any remaining characters?*

char readChar() *read a character from standard input and return it*

instance methods that read lines from the input stream

boolean hasNextLine() *does standard input have a next line?*

String readLine() *read the rest of the line and return it as a String*

instance methods that read the rest of the input stream

int[] readAllInts() *read all remaining tokens; return as array of ints*

double[] readAllDoubles() *read all remaining tokens; return as array of doubles*

...

The full [In API](#).

Our output library.

public class Out

<code>Out()</code>	<i>create an output stream to standard out</i>
<code>Out(String name)</code>	<i>create an output stream to a file</i>
<code>void print(String s)</code>	<i>print s to the output stream</i>
<code>void println(String s)</code>	<i>print s and a newline to the output stream</i>
<code>void println()</code>	<i>print a newline to the output stream</i>
<code>void printf(String format, ...)</code>	<i>print the arguments to the output stream as specified by the format string format</i>

The full [Out API](#).

Our picture library.

public class Picture

<code>Picture(String filename)</code>	<i>create a picture from a file</i>
<code>Picture(int w, int h)</code>	<i>create a blank w-by-h picture</i>
<code>int width()</code>	<i>return the width of the picture</i>
<code>int height()</code>	<i>return the height of the picture</i>
<code>Color get(int col, int row)</code>	<i>return the color of pixel (col, row)</i>
<code>void set(int col, int row, Color color)</code>	<i>set the color of pixel (col, row) to color</i>
<code>void show()</code>	<i>display the picture in a window</i>
<code>void save(String filename)</code>	<i>save the picture to a file</i>

The full [Picture API](#).

Our stack data type.

```
public class Stack<Item> implements Iterable<Item>
```

<code>Stack()</code>	<i>create an empty stack</i>
<code>boolean isEmpty()</code>	<i>is the stack empty?</i>
<code>void push(Item item)</code>	<i>push an item onto the stack</i>
<code>Item pop()</code>	<i>return and remove the item that was inserted most recently</i>
<code>int size()</code>	<i>number of items on stack</i>

The full [Stack API](#).

Our queue data type.

```
public class Queue<Item> implements Iterable<Item>
```

<code>Queue()</code>	<i>create an empty queue</i>
<code>boolean isEmpty()</code>	<i>is the queue empty?</i>
<code>void enqueue(Item item)</code>	<i>insert an item onto queue</i>
<code>Item dequeue()</code>	<i>return and remove the item that was inserted least recently</i>
<code>int size()</code>	<i>number of items on queue</i>

The full [Queue API](#).

Iterable.

```

import java.util.Iterator; ← Iterator  
not in language

public class Queue<Item>
    implements Iterable<Item> ← promise to  
implement  
iterator()
{
    private Node first;
    private Node last;
    private class Node
    {
        Item item;
        Node next;
    }
    public void enqueue(Item item)
    ...
    public Item dequeue()
    ...

    public Iterator<Item> iterator()
    { return new ListIterator(); } ← FIFO  
queue  
code
}

private class ListIterator
    implements Iterator<Item> ← implementation  
for Iterable  
interface
{
    Node current = first; ← additional  
code to  
make the  
class iterable
    public boolean hasNext()
    { return current != null; } ← promise to implement  
hasNext(), next(),  
and remove()

    public Item next()
    {
        Item item = current.item;
        current = current.next;
        return item;
    } ← implementations  
for Iterator  
interface

    public void remove()
    { }

    public static void main(String[] args)
    {
        Queue<Integer> queue = new Queue<Integer>();
        while (!StdIn.isEmpty())
            queue.enqueue(StdIn.readInt());
        for (int s : queue) ← foreach  
statement
            StdOut.println(s);
    }
}

```

Our symbol table data type.

```
public class ST<Key extends Comparable<Key>, Value>
```

ST()	<i>create an empty symbol table</i>
void put(Key key, Value val)	<i>associate val with key</i>
Value get(Key key)	<i>value associated with key</i>
void remove(Key key)	<i>remove key (and its associated value)</i>
boolean contains(Key key)	<i>is there a value associated with key?</i>
int size()	<i>number of key-value pairs</i>
Iterable<Key> keys()	<i>all keys in the symbol table</i>

The full [ST API](#).

Our set data type.

```
public class SET<Key extends Comparable<Key>> implements Iter
```

SET()	<i>create an empty set</i>
boolean isEmpty()	<i>is the set empty?</i>
void add(Key key)	<i>add key to the set</i>
void remove(Key key)	<i>remove key from set</i>
boolean contains(Key key)	<i>is key in the set?</i>
int size()	<i>number of elements in set</i>

The full [SET API](#).

Our graph data type.

public class Graph

```

    Graph()                               cr
    Graph(String filename, String delimiter)  cr
    void addEdge(String v, String w)        aa
    int V()                                ni
    int E()                                ni
Iterable<String> vertices()           ve
Iterable<String> adjacentTo(String v)   ne
    int degree(String v)                  ni
    boolean hasVertex(String v)          is
    boolean hasEdge(String v, String w)   is

```

The full [Graph API](#).

Compile-time and run-time errors.

Here's a [list of errors](#) compiled by Mordechai Ben-Ari. It includes a list of common error message and typical mistakes that give rise to them.

Last modified on October 30, 2019.

Copyright © 2000–2019 [Robert Sedgewick](#) and [Kevin Wayne](#). All rights reserved.