Functions in Java

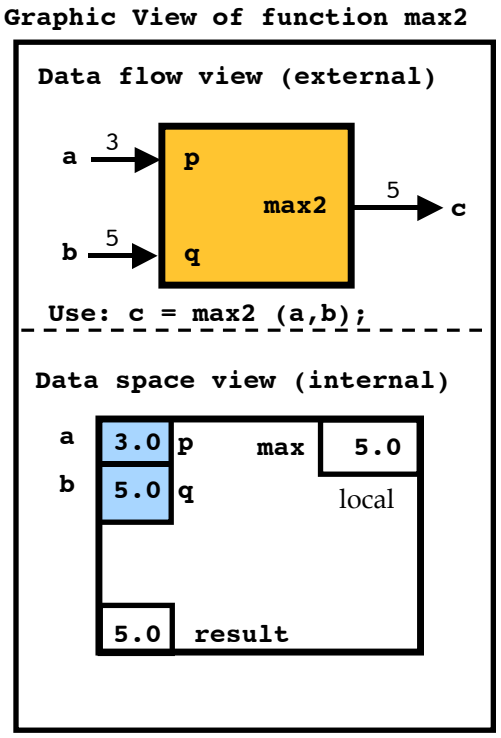
Functions are very important ways to "package" or encase (encapsulate) actions.

Functions are methods that return a value of a given type, so are called typed methods.

Within the definition there is a result, of a given type, which must be returned.

Functions are parts of other expressions; they do not stand alone.

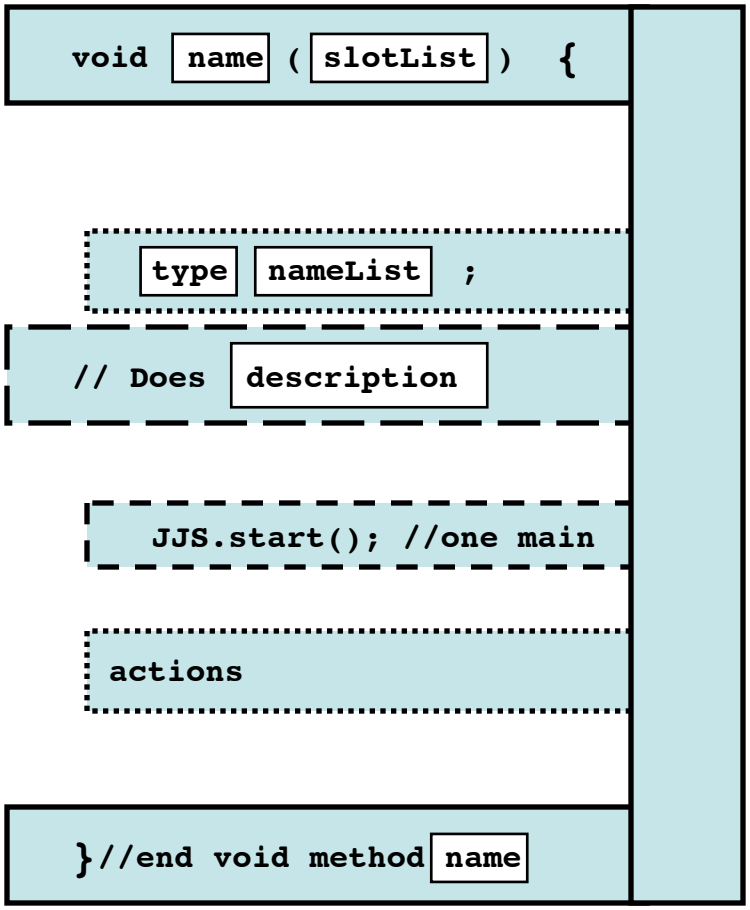
Max2 is shown below in a general form, with an external and internal view.

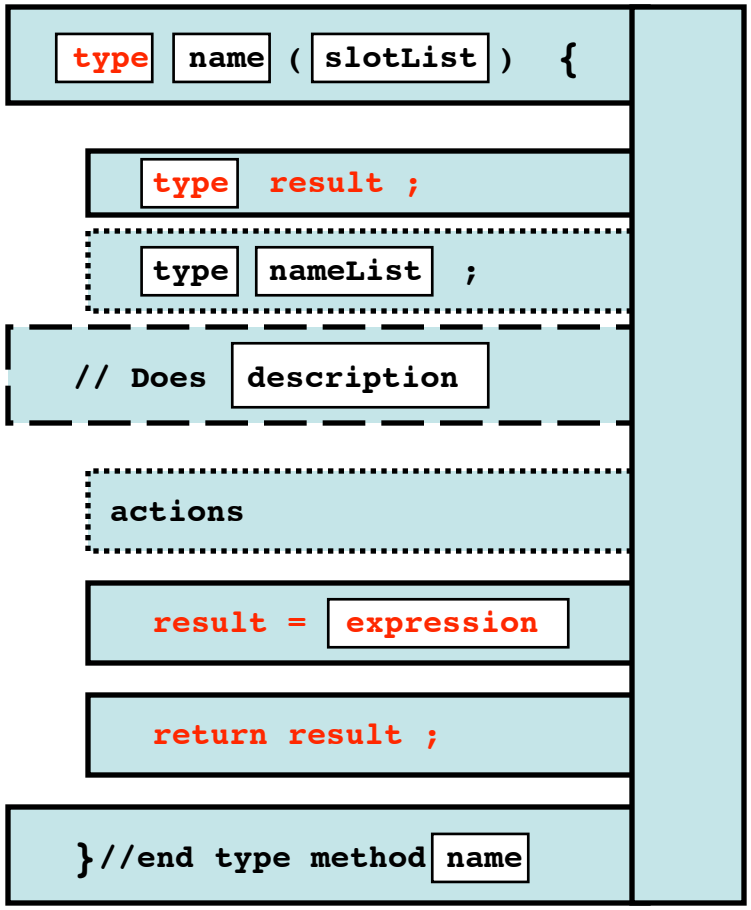


The max2 function can be used and reUsed
In various ways was shown;
The max2 is part of a larger expression such as

```
LargestOf2 = max2 (a, b);

largestOf3 = max2 ( max2 (a,b), c);

largestOf4 = max2 ( max2(a,b), Max2(c,d) );

biggestOf4 = max2 ( max2(a, max2(b, max2(c,d))));
```

Syntax of a type method is similar to that of a void method as shown below.
This kind of method has a type, that is returned, which is placed right before its name.
Also, the typed method involves a box called result, which is declared initially,
is assigned some value, and it is finally returned.
And the type method can have no **`JJS.start`** command; that is only in a routine.

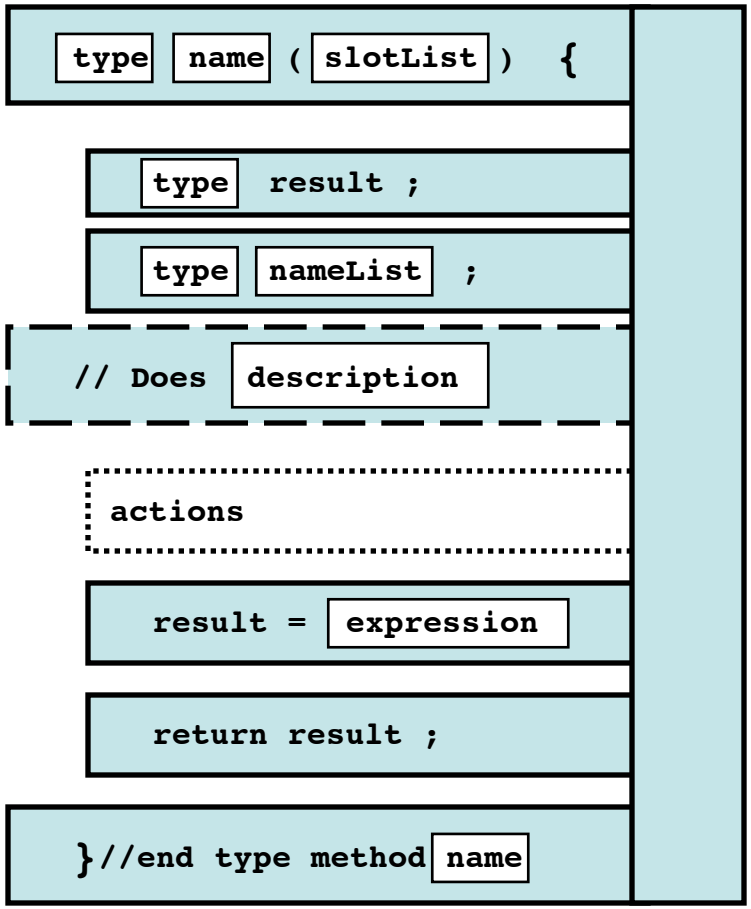




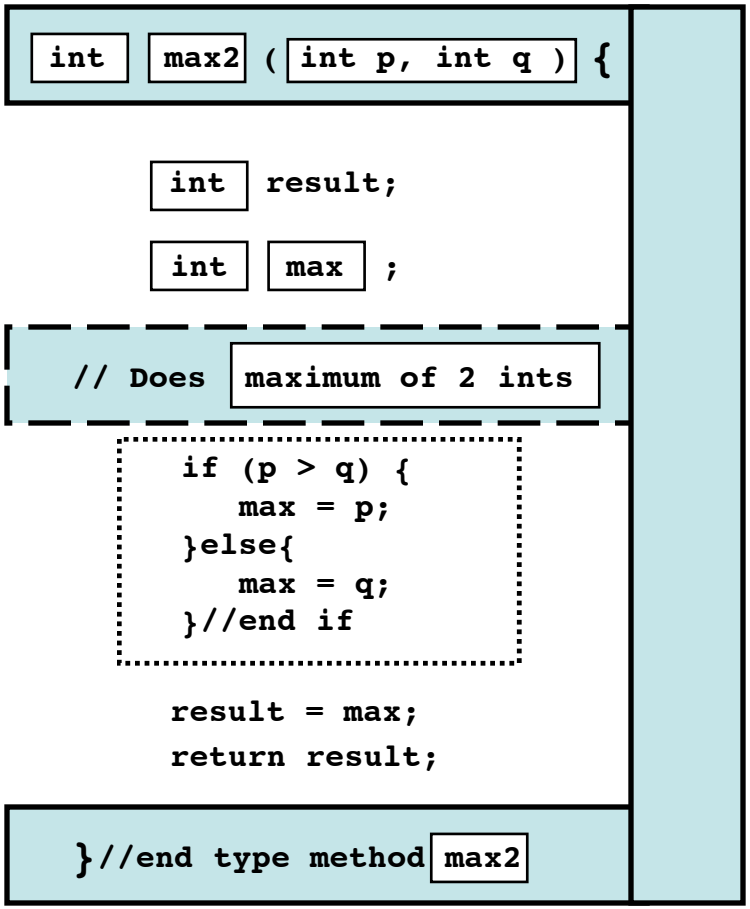
Note: similarities and differences

Max2 is shown defined below next to the syntax diagram.
The definition of max2 consists mainly of filling in the boxes of the general syntax.

General Syntax definition of Function



Definition of the max2 Function



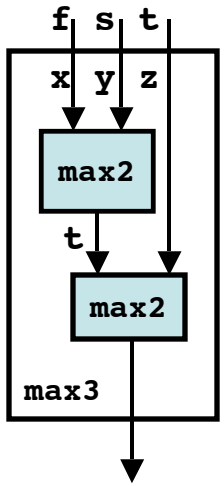
Maximum of Two is done again as follows.
It shows a main method which calls max2, and
it shows the function max2 defined in a different layout
(with the header spread over many linesrather than one), and
with a different structure (having no temporary box named max)
and involving a different type (double rather than int).

```
// Name An Onymous
void mainA() {
   double first, second, max;
// Does Maximum of 2 reals, another way
   JJS.start();
   JJS.outputlnString ("Enter 2 reals: ");
   first  = JJS.inputDouble ();
   second = JJS.inputDouble ();

   max =  max2 (first, second);

   JJS.outputlnString ("The maximum is ");
   JJS.outputlnDouble (max);
}//end void main
```

```
// Function max2 (p,q) of reals
   double
   max2 (
      double p,
      double q) {
    double result;
 // Does return max of 2 reals
    if (p > q) {
       result = p;
    }else{
       result = q;
    }// end if
    return result;
 }//end real function max2
```

Maximum of Three is done by using max3 which call max2 twice.

```
// Name An Onymous
void mainB();
   int first, second, third, max;
// Does Max3: functions calling functions
   JJS.start();
   JJS.outputlnString ("Enter 3 values : ");
   first  = JJS.inputInt ();  // do also
   second = JJS.inputInt ();  // an echo
   third  = JJS.inputInt ();  // of these

   max = max3 (first, second, third);

   JJS.outputlnString ("The maximum is ");
   JJS.outputlnDouble (max);
}//end void mainB
```




```
Enter 3 values :
3
6
5
The maximum is
6
```

```
// Function max2 (p,q) of ints
   int max2 (int p, int q) {
      int result;
   // Does return max of 2 ints
      if (p > q) {
         result = p;
      }else{
         result = q;
      }// end if
      return result;
   }//end int function max2
```

```
// Function max3 (x,y,z)
   int
   max3 (int x, int y, int z) {
      int result;
      int temp;
   // Does return max of 3 ints
      temp   = max2 (x,y);
      result = max2 (temp, z);
      return result;
   }//end int function max3
```

Some convenient typed methods

```
int round (double x) {
   int result;
// Does round a real to nearest int
   result = JJS.doubleToInt (x + 0.5);
   return result;
}//end integer method round
```

```
double degreesToRadians (double d) {
   double result;
// Does do angle conversion
   result = d * Math.PI / 180.0);
   return result;
}//end real method degrees2radians
```

```
int abs (int x) {
   int result;
// Does return positive value
   if (x < 0) {
      result = 0 - x;
   }else{
      result = x;
   }//end if
   return result
}//end int method abs
```

```
double celsiusToFahrenheit (double c) {
   double result;
// Does convert tempertures
   result = (9.0/5.0)*c + 32.0;
   return result;
}//end double method c2f
```

Do define some other convenient typed methods such as:
radiansToDegrees, fahrenheitToCelsius, sine (degrees), log10(x).

```
// Name An Onymous

void mainC() {
   double first, second, third, area;
// Does area of a triangle
   JJS.start() {
   JJS.outputlnString ("Enter 3 sides : ");
   first  = JJS.inputDouble ();  //do also
   second = JJS.inputDouble ();  //an echo
   third  = JJS.inputDouble ();  //of these

   area = triArea (first, second, third);

   JJS.outputlnString ("The area is ");
   JJS.outputlnDouble (area);
}//end void mainC
```

```
  double
  triArea (double a, double b, double c ) {
      double result;
      double s, t;
   // Does area of a triangle
   // Need positive lengths of sides a,b,c
   // using Hero's formula
      s = (a + b + c) / 2.0;
      t = s*(s-a)*(s-b)*(s-c);
      result = Math.sqrt (t);
      return result;
   }//end double function triArea
```

```
Enter 3 sides :
3.0
4.0
5.0
The area is 6.0

Enter 3 sides :
3.0
4.0
7.0
The area is 0.0

Enter 3 sides :
3.0
3.0
3.0
The area is 3.89711
```

```
// Name An Onymous
void mainD() {
   double pay, hours, rate;
// Does First Weekly Gross Pay: uses Choices
   JJS.start() {
   JJS.outputlnString ("Enter hours : ");
   hours = JJS.inputDouble ();

   JJS.outputlnString ("Enter  rate : ");
   rate = JJS.inputDouble ();

   pay  = grossPay (hours, rate);

   JJS.outputString ("The gross pay is ");
   JJS.outputDouble ( pay);
}//end void mainD
```

```
   double
   grossPay (double hrs, double rate) {
      double result;
   // Does return real gross pay
   // Need real positive (hrs < 7*24) & rate
      if (hrs < 40.0) {
         result = hrs * rate;
      }else{
         result = rate * 40.0 +
                  rate * 1.50 * (hrs - 40.0);
      }// end if
      return result;
   }//end function grossPay
```

```
Enter hours :
50.0
Enter rate :
10.00
The gross pay is 550
```

A "Monolithic" program (at the left) broken into "Sub" programs (at the right)

```
// Name An Onymous
void mainE() {
   int fact, num;
// Does compute factorial: uses While

   JJS.start() {

   JJS.outputlnString ("Enter an int ");
   num = JJS.inputInt ();
   JJS.outputlnInt (num);

   fact = 1;

   while (num > 0) {
      fact = fact * num;
      num = num - 1;
   }//end while

   JJS.outputString ("Factorial is  ");
   JJS.outputlnInt (fact);
}//end void mainE
```

```
Enter an int
10
Factorial is 3628800
```

```
// Name An Onymous
void mainF() {
   int fact, num;
// Does call factorial
   JJS.start();
   JJS.outputlnString ("Enter an int ");
   num  = JJS.inputInt ();
   JJS.outputlnInt (num);

   JJS.outputlnInt ( factorial (num) );
//end void mainF
```

```
int factorial (int num) {
   int result;
   int fact;
// Does factorial with decreasing index
// Need an integer (num >= 0)
   fact = 1;
   while (num > 0) {
      fact = fact * num;
      num = num - 1;
   }//end while
   result = fact;
   return result;
}//end int function factorial
```

Combinations computes the number of ways that m things can be chosen n at a time.

For example 10 things can be arranged 2 at a time in 45 ways given by the formula:

c = n!/((n - r)!/r!)

The following main code shows how factorial is reused 3 times.

```
void myMain () {
   int c,n,r;
// Does compute combinations
// Shows reUse of factorial function
   JJS.start();

   n = 10;
   r = 2;

   c = factorial (n) /
      (factorial(n-r)*factorial(r));

   JJS.outputString ("Combinations = ");
   JJS.outputlnInt ( c );
   JJS.outputString ("\n"); // gap

}//end void method myMain
```

```
int factorial (int num) {
   int result;
   int fact, i;
// Does factorial of num
// Need positive integer num
   fact = 1;
   i = 1;
   while (i <= num) {
      fact = fact * i;
      i = i + 1;
   }//end while

   result = fact;
   return result;
}//end int function factorial
```

```
Combinations = 45
```

Many more ways (3) to do sub-programs: Factorial again

```
// Name An Onymous
void mainG() {
   int fact, num;
// Does compute factorial
   JJS.start();
   JJS.outputlnString ("Enter an int ");
   num  = JJS.inputInt ();

   JJS.outputlnInt ( factorial (num) );
}//end void mainG
```

```
Enter an int
10
Factorial is 3628800
```

```
int factorial (int num) {
   int result;
// Does factorial without temp fact box
// Need (num >= 0)
   result = 1;
   while (num > 0) {
      result = result * num;
      num = num - 1;
   }//end while

   return result;
}//end int function factorial
```

```
int factorial (int num) {
   int result;
   int fact;
// Does factorial: the original way
// Need (num >= 0)
   fact = 1;
   while (num > 0) {
      fact = fact * num;
      num = num - 1;
   }//end while

   result = fact;
   return result;
}//end int function factorial
```

```
int factorial (int num) {
   int result;
   int i;
// Does factorial: using a for loop
// Need (num >= 0)
   result = 1;
   for (i=1; i<num; i++) {
        result = i * result;
   }//end for loop

   return result;
}//end int function factorial
```

Yet another way (#4) to do sub-programs: recursively (methods calling themselves)

```
// Name An Onymous
void mainH() {
   int fact, num;
// Does call recursive factorial
   JJS.start();
   JJS.outputlnString ("Enter an int ");
   num  = JJS.inputInt ();

   JJS.outputlnInt ( factorial (num) );
}//end void mainH
```

```
int factorial (int num) {
   int result;
// Does factorial recursively; calls itself
// Need (num >= 0)
   if (num > 0) {
      result = num * factorial (num - 1);
   }else{
      result = 1;
   }//end if
   return result;
}//end int function factorial
```

```
Enter an int
5
Factorial is 120
```

A trace of recursive factorial

```
factorial (5) = 5* factorial (4)
              = 5* 4*factorial (3)
              = 5*4* 3*factorial (2)
              = 5*4*3* 2*factorial (1)
              = 5*4*3*2* 1*factorial (0)
              = 5*4*3*2*1
              = 120
```

```
Enter an int
10
Factorial is 3628800

Enter an int
16
Factorial is 2004189184
```

```
// Name An Onymous

void mainI() {
   int first, second, num, sum, rolls, odds;
   double prob, percent;
// Does Count Odd Dice sums
   JJS.start();
   JJS.outputlnString ("Enter rolls :");
   rolls = JJS.inputInt ();

   odds = 0; // Reset number of odd sums

   num = 0;
   while (num < rolls) {

      sum = sum2();

      if ( (sum % 2) == 1) {
         odds = odds + 1;
      }//end if

      num = num + 1;
   }//end while

   percent = 100.0 * JJS.intToDouble (odds) /
                     JJS.intToDouble (rolls);

   JJS.outputString  ("Percent odds = ");
   JJS.outputlnDouble (percent);

}//end void mainI
```

```
int roll () {
   int result;
   double prob, side;
// Does roll one die
   prob = Math.random ();
   side = 6.0 * prob;
   result = JJS.doubleToInt(side) + 1;
   return result;
}//end int function roll
```

```
int sum2 () {
   int result;
   int first, second;
// Does return sum 2 to 12 of 2 rolls
   first  = roll();
   second = roll();
   result = first + second;
   return result;
}//end int function sum2
```

```
Enter rolls :
1000
Percent odds = 50.1
```

Some type functions involving Strings

```
String firstChar (String str) {
   String result;
// Does give first char
   result = str.substring (1,1);
   return result;
}//end String method firstChar
```

```
boolean isChar (String str) {
   boolean result;
// Does tell if str is short
   result = (str.length() == 1);
   return result;
}//end String method is a Char
```

```
String joined
  (String str1, String str2) {
   String result;
// Does join 2 strs with space
   result = str1 + " " + sdtr2;
   return result;
}//end String join Strings
```

```
Void mainString() {
   String first, mid, last;
   String initial, fullName;
// Does manipulate names
   JJS.Start();

   first = "John";
   mid   = "Michael";
   last  = "Gates";

   initial = firstChar (mid);

   fullname = joined (first, initial);

   fullname = joined (fullname, last);

   JJS.outputlnString (fullname);
//end void method mainString
```

```
John M Gates
```

Boolean type methods: many ways to get **even**

```
boolean isEven (int n) {
   boolean result;
// Needs (n >= 0)
// Does tell if n is even
   if ( (n % 2) == 0) {
      result = true;
   }else{
      result = false;
   }//end if
   return result;
}//end boolean method isEven
```

```
boolean isEven (int n) {
   boolean result;
// Does tell if n is even
   result = false;
   if ( (n % 2) == 0) {
      result = true;
   }//end if 2 divides n
   return result;
}//end boolean method isEven
```

```
boolean isEven (int n) {
   boolean result;
// Needs (n >= 0)
// Does tell if n is even
   result = n;
   while (result > 1) {
      result -= 2;
   }//end while
   return (result == 0);
}//end boolean method isEven
```

```
boolean isEven (int n) {
   boolean result;
// Does tell if n is even
   result = ( (n%2) == 0);
   return result;
}//end boolean method isEven
```

```
boolean isEven (int n) {
   return ( (n%2) == 0);
}//end boolean method isEven
```

Some Logical type methods: Or, Not, Nor

```
boolean not (boolean p) {
   boolean result;
// Does logical Not
   result = ! p ;
   return result;
}//end boolean method Not
```

```
boolean or
   (boolean p, boolean q) {
   boolean result;
// Does logical Or in
   result = p || q;
   return result;
}//end boolean method Or
```

```
boolean nor
   (boolean p, boolean q) {
   boolean result;
// Does operate as a Nor
   result = not( or(p,q) );
   return result;
}//end boolean method Nor
```

```
void NorTest() {
   boolean a, b, c;
//Does show Nor operator
   JJS.start();
   JJS.outputlnString ("Enter truths");
   a= JJS.inputBoolean();
   JJS.outputlnBoolean (a);
   b= JJS.inputBoolean();
   JJS.outputlnBoolean (b);
// c = nor(a,b);
   JJS.outputlnString ("The Nor is ");
   JJS.outputlnBoolean (nor(a,b));

}// end void method logicTest
```

```
Enter truths
true
false
The Nor is
false
```

```
play = (tied or early) and not rain
     = and ( or(tied,early), not(rain) )
```

Fallacies, are improper arguments that are not true.
For example consider the following truth, which may look like a proper form of argument:

```
 (p -> q)      -- if p implies q
 ! P           -- and not p
---------      -- then (therefore)
 ! q           -- q is not true
```

Using the following example for symbols p, q shows the wrong reasoning:
If the fire is lit then there is light
And the fire is not lit
Therefore there is no light
But the light need not come only from the fire (perhaps from a bulb, sun, etc)

This fallacy can be written as a single expression:

```
( (p -> q) && ! p ) -> ! q
```

It can be disproved by a truth table. Actually the entire table is not required;
only one case is sufficient, and this is the second case (when p is false and q is true).

p	q	((p -> q)	&&	! p)	->	! q
F	F	T	T	T	T	T
F	T	T	T	T	F	F
T	F					
T	T					
1	2	3	5	4	7	6

More Boolean type methods: disproving a fallacy **((p --> q) && !p) --> !q**

```
boolean not (boolean p) {
   boolean result;
// Does logical Not in prefix form
   result = ! p ;
   return result;
}//end boolean method Not
```

```
boolean implies
   (boolean p, boolean q) {
   boolean result;
// Does "if p then q" = ! p | q
   result = or (not(p), q);
   return result;
}//end boolean method implies
```

```
boolean or
   (boolean p, boolean q) {
   boolean result;
// Does logical Or in prefix form
   result = p || q;
   return result;
}//end boolean method Or
```

```
void logicTest() {
   boolean a,b,c,d, p, q;
//Does show a fallacy
   p = false;
   q = true;

   a = implies(p,q);  // premise
   b = not(p);        // premise
   c = not(q);        // conclude
   d = a && b;        // both
   e = implies (d,c); // falacy

   JJS.outputlnBoolean(e); //false!
}// end void method logicTest
```

```
boolean and
   (boolean p, boolean q) {
   boolean result;
// Does logical And in prefix form
   result = p && q;
   return result;
}//end boolean method And
```

Problems on typed methods

`Trunc(r)` which returns the truncated value of real value r

`kilometersToMiles (km)` converts metric distances

`milesToKilometers (mi)` converts distances

`Mean3(a,b,c)` returns the average of 3 values

`areaRectangle (wid, length)` computer the area of a rectangle

`Hypot (x, y)` returns the hypotenuse given sides of a right triangle

`withinBounds (val, lo ,hi)` returns truth of value val
being within a given lower bound and a higher one.

`sizeInt(i)` returns the number of decimals in integer i

`lsd(i)` returns the least significant digit of an int i.
`msd(i)` returns the most significantr digit of an int i.

`Mod(num, den)` returns the remainder of num / den
`Div (num, den)` returns the quotient of num / den

Finally, you provide one.

More Problems on typed methods

Define the following arithmetic methods acting on ints:

Sum(a,b) adding the given two integers.
Dif (a,b) subtracting these ints

Prod(a,b) multiplying the ints
Quot(a,b) dividing the ints

Pow(a,b) taking a to power b
Rem (a,b) remainder when a is divided by b

lsd (n)   least significant digit of n (rightmost)
msd (n)   most significant digit of n (leftmost)

Some boolean methods:

evenlyDivides(a,b)
isMultipleOf(a,b)
isTwiceSize(a,b)
isfactorOf(a,b)
isHalfOf(a,b)