- Previous Lecture (and lab):
  - Variables & assignment
  - Built-in functions, input & output
  - Good programming style (meaningful variable names; use comments)

- Today, Lecture 3:
  - Writing a program—systematic problem solving
  - Branching (conditional statements)

Announcements:

- Discussion sections in Upson 225 lab this week, not classroom listed on Student Center
- Project 1 (P1) to be posted after lecture; due Tue, Feb 4, at 11pm
  - Pay attention to Academic Integrity
- Matlab consultants at ACCEL Green Rm (Carpenter Hall 2nd floor computing facility) 4:30–9:30pm Sun.–Thurs.
- Piazza – "Q & A system" for all students in CS1112. Use it for clarification only—do not ask (or answer) homework questions and do not give hints on homework. Will be monitored by TAs.
- Reading from the textbook is important for your learning. Read the specified sections BEFORE lecture
  - Review material a little bit each day
  - Take notes during lecture
- Enroll in the optional AEWs (or sign up on the wait list)

Quick review

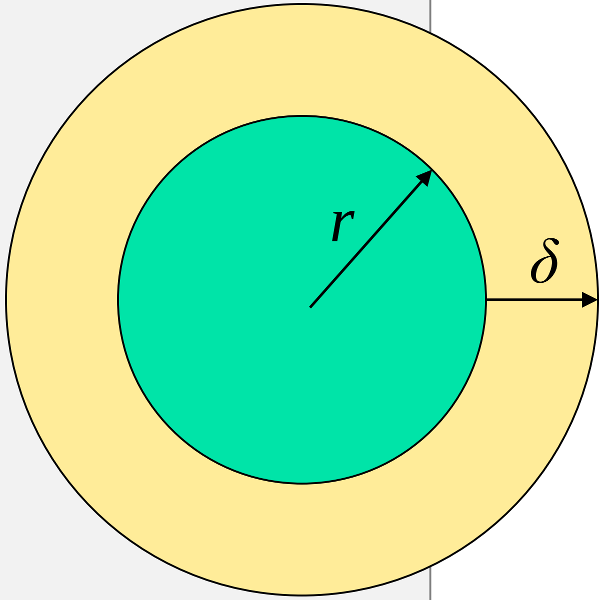
- Variable
  - A named memory space to store a value
- Assignment operator: =
  - Let x be a variable that has a value. To give variable y the same value as x, which statement below should you write?

    x = y   or   y = x
- Script (program)
  - A sequence of statements saved in an m-file
- ; (semi-colon)
  - Suppresses printing of the result of assignment statement

Tips for writing a program

- Check that you know what is given (or is input, or is assumed)
- Be *goal-oriented*: start by writing the last statement(s) for the program output
  - What is the program supposed to produce? *You know this from the problem statement*
  - Allows you to work backwards from the results
- Name as a variable what you don't know
  - Helps you break down the steps
  - Allows you to temporarily skip over any part that you don't know yet how to do

```matlab
% Compute surface area increase of a sphere in
% miles^2 given an increase in the radius in inches

r= input('Enter radius r in miles: ');
delta= input('Enter delta r in inches: ');

newr= r + (delta/12)/5280;  % mi

A= 4*pi*r^2;          % mi^2
newA= 4*pi*newr^2;    % mi^2

deltaA= newA - A;     % mi^2

fprintf('Increase in mile^2 is %f.\n', deltaA)
```



Beyond batching

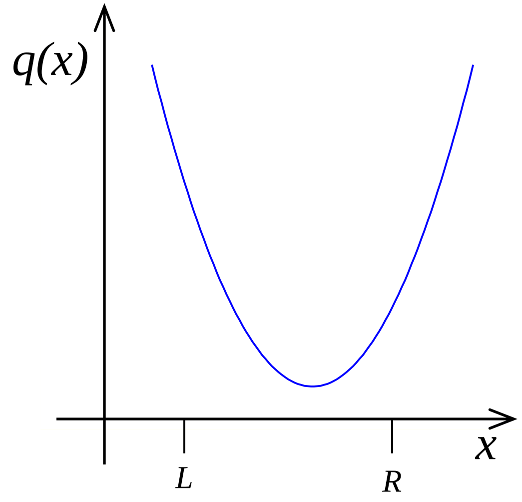
- So far, *all* the statements in our scripts are executed in order
- We do not have a way to specify that some statements should be executed only under some condition
  - Want to be able to make decisions
- We need a new language construct…

Motivation

Consider the quadratic function

$$q(x) = x^2 + bx + c$$

on the interval $[L\ ,\ R]$:



- Is the function strictly increasing in $[L\ ,\ R]$?
- Which is smaller, $q(L)$ or $q(R)$?
- What is the minimum value of $q(x)$ in $[L\ ,\ R]$?

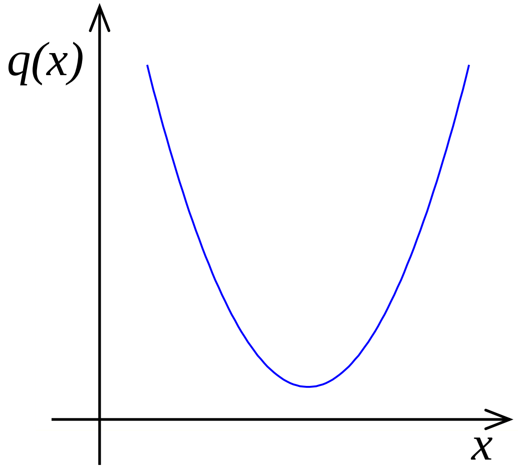
Problem 1

Write a code fragment that prints "Increasing" if $q(x)$ strictly increases across the interval and "Not increasing" if it does not.

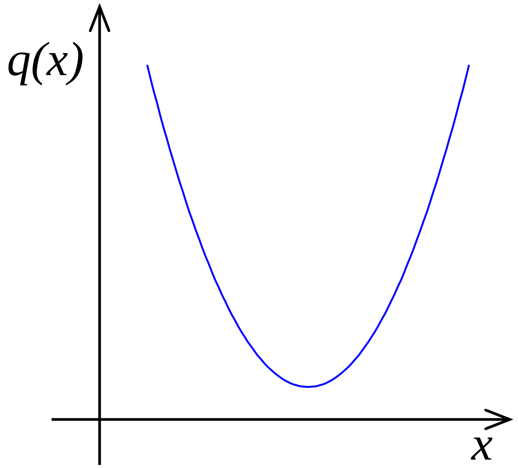
```
% Quadratic q(x) = x^2 + bx + c
b = input('Enter b: ');
c = input('Enter c: ');
L = input('Enter L: ');
R = input('Enter R, R>L: ');
```

```
% Determine whether q increases
% across [L,R]
```

- What are the critical points?
  - End points: $x = L$, $x = R$
  - $\{ x \mid q'(x) = 0 \}$



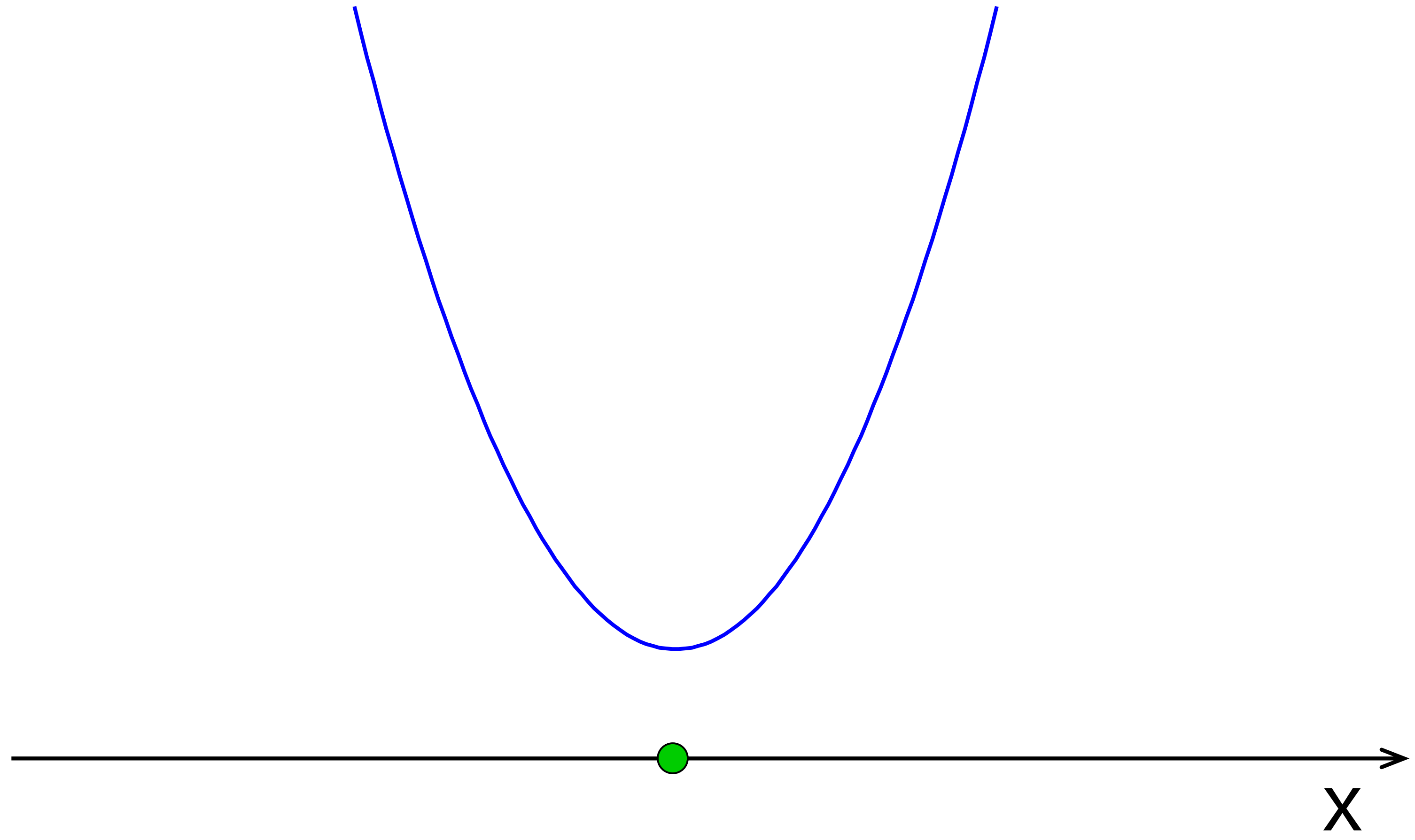
- What are the critical points?
  - End points: $x = L$, $x = R$
  - $\{ x \mid q'(x) = 0 \}$



The Situation

$$q(x) = x^2 + bx + c$$

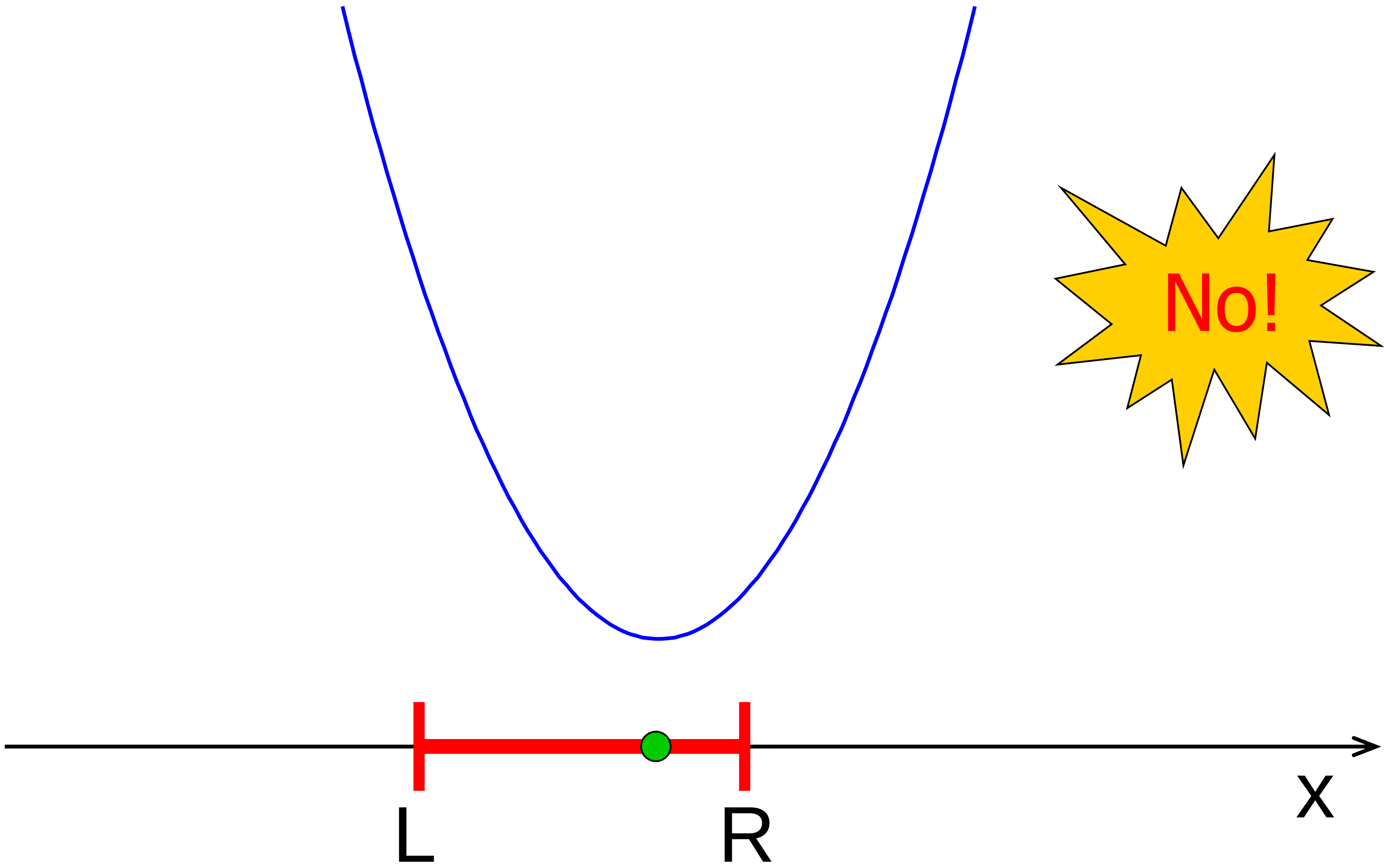
$$x_c = -b/2$$



Does q(x) increase across [L,R]?

$$q(x) = x^2 + bx + c$$

$$x_c = -b/2$$



So what is the requirement?

```
% Determine whether q increases
% across [L,R]
xc = -b/2;

if ____________________

   fprintf('Increasing\n')
else   % otherwise
   fprintf('Not increasing\n')
end
```

Relational Operators

<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
~=	Not equal to

So what is the requirement?

```
% Determine whether q increases
% across [L,R]
xc = -b/2;

if ________________

    fprintf('Increasing\n')
else
    fprintf('Not increasing\n')
end
```

A: R >= xc

B: xc <= R

C: xc <= L

D: L <= xc

Final code

```
% Determine whether q increases
% across [L,R]
xc = -b/2;

if xc <= L

    fprintf('Increasing\n')
else
    fprintf('Not increasing\n')
end
```

Problem 2

Write a code fragment that prints

"qleft is smaller"

if q(L) is smaller than q(R).

If q(R) is smaller print

"qright is smaller."

Algorithm v0

Calculate q(L)

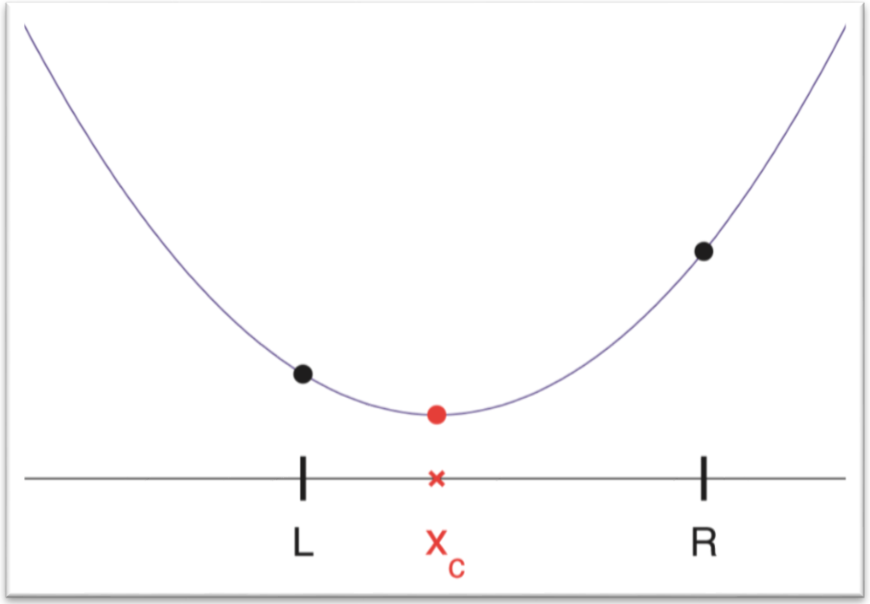
Calculate q(R)

If q(L) < q(R)

  print "qleft is smaller"

Otherwise

  print "qright is smaller"



Algorithm v0.1

Calculate x_c

If distance x_cL is smaller than distance x_cR

print "qleft is smaller"

Otherwise

print "qright is smaller"

Do these two fragments do the same thing?

```
% given x, y
if  x>y
    disp('alpha')
else
    disp('beta')
end
```

```
% given x, y
if  y>x
    disp('beta')
else
    disp('alpha')
end
```

A: yes

B: no

Algorithm v1.1

Calculate x_c

If distance x_cL is smaller than distance x_cR

print "qleft is smaller"

Otherwise

print "qright is smaller or equals qleft"

Algorithm v2.1

Calculate x_c

If distance x_cL is same as distance x_cR

print "qleft and qright are equal"

Otherwise, if x_cL is shorter than x_cR

print "qleft is smaller"

Otherwise

print "qright is smaller"

```
% Which is smaller, q(L) or q(R)?

xc= -b/2;   % x at minimum
if (abs(xc-L) == abs(xc-R))
   disp('qleft and qright are equal')
elseif (abs(xc-L) < abs(xc-R))
   disp('qleft is smaller')
else
   disp('qright is smaller')
end
```

Algorithm v2

Calculate q(L)

Calculate q(R)

If q(L) equals q(R)

    print "qleft and qright are equal"

Otherwise, if q(L) < q(R)

    print "qleft is smaller"

Otherwise

    print "qright is smaller"

```
% Which is smaller, q(L) or q(R)?

qL= L*L + b*L + c;   % q(L)
qR= R*R + b*R + c;   % q(R)
if (qL == qR)
   disp('qleft and qright are equal')
elseif (qL < qR)
   disp('qleft is smaller')
else
   disp('qright is smaller')
end
```

```
% Which is smaller, q(L) or q(R)?

qL= L*L + b*L + c;   % q(L)
qR= R*R + b*R + c;   % q(R)
if (qL == qR)
   disp('qleft and qright are equal')
   fprintf('q value is %f\n', qL)
elseif (qL < qR)
   disp('qleft is smaller')
else
   disp('qright is smaller')
end
```

Consider the quadratic function

$$q(x) = x^2 + bx + c$$

on the interval $[L , R]$:

What if you only want to know if $q(L)$ is close to $q(R)$?

```
% Is q(L) close to q(R)?

tol= 1e-4;   % tolerance
qL= L*L + b*L + c
qR= R*R + b*R + c
if (abs(qL-qR) < tol)
  disp('qleft and qright similar')
end
```

Name an important parameter and define it with a comment!

`else` is optional in an if-statement. This if-statement without `else` is correct.

The **if** construct

```
if  boolean expression1
    statements to execute if expression1 is true
elseif  boolean expression2
    statements to execute if expression1 is false
    but expression2 is true
:
else
    statements to execute if all previous conditions
    are false
end
```

Can have any number of elseif branches
but at most one else branch

Things to know about the `if` construct

- At most one branch of statements is executed
- There can be any number of `elseif` clauses
- There can be at most one `else` clause
- The `else` clause must be the last clause in the construct
- The `else` clause does not have a condition (boolean expression)

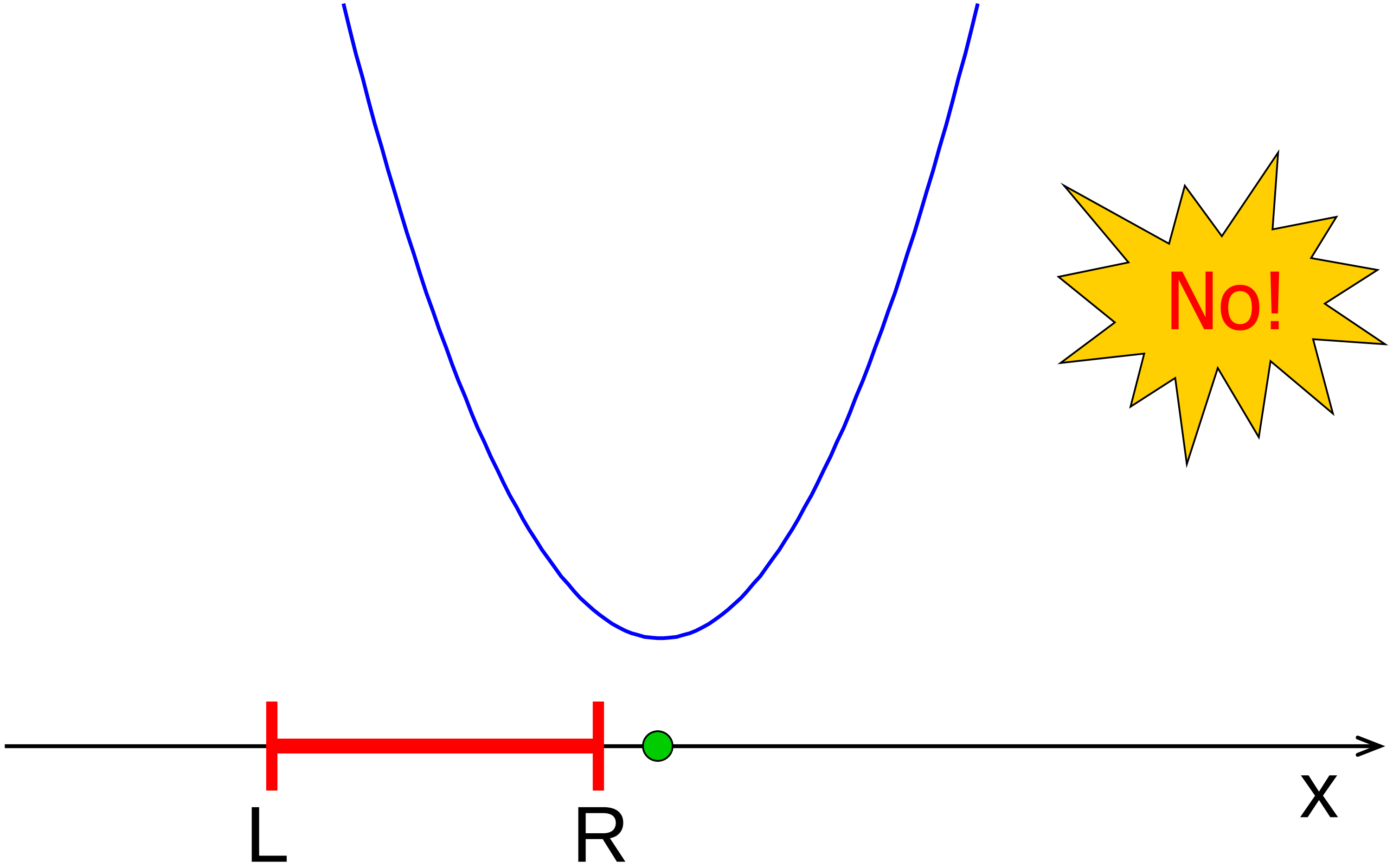
Problem 3

Write a code fragment that prints "Inside" if xc is in the interval and "Outside" if it is not.

Is `xc` in the interval [`L`,`R`]?

$$q(x) = x^2 + bx + c$$

$$x_c = -b/2$$



Logical operators

`&&` logical <u>and</u>: Are both conditions true?

E.g., we ask "is $L \le x_c$ and $x_c \le R$?"

In our code: `L<=xc && xc<=R`

`||` logical <u>or</u>: Is at least one condition true?

E.g., we can ask if x_c is outside of $[L,R]$,

i.e., "is $x_c \le L$ or $R \le x_c$?"

In code: `xc<L || R<xc`

`~` logical <u>not</u>: Negation

E.g., we can ask if x_c is not outside $[L,R]$.

In code: `~(xc<L || R<xc)`