

61A Lecture 6

Friday, September 12

Announcements

Announcements

- Homework 2 due Monday 9/15 @ 11:59pm

Announcements

- Homework 2 due Monday 9/15 @ 11:59pm
- Project 1 due Wednesday 9/17 @ 11:59pm

Announcements

- Homework 2 due Monday 9/15 @ 11:59pm
- Project 1 due Wednesday 9/17 @ 11:59pm
- Optional Guerrilla section Saturday 9/13 @ 12:30pm in 306 Soda about higher-order functions

Announcements

- Homework 2 due Monday 9/15 @ 11:59pm
- Project 1 due Wednesday 9/17 @ 11:59pm
- Optional Guerrilla section Saturday 9/13 @ 12:30pm in 306 Soda about higher-order functions
 - Organized by Andrew Huang and the readers

Announcements

- Homework 2 due Monday 9/15 @ 11:59pm
- Project 1 due Wednesday 9/17 @ 11:59pm
- Optional Guerrilla section Saturday 9/13 @ 12:30pm in 306 Soda about higher-order functions
 - Organized by Andrew Huang and the readers
 - Work in a group on a problem until everyone in the group understands the solution

Announcements

- Homework 2 due Monday 9/15 @ 11:59pm
- Project 1 due Wednesday 9/17 @ 11:59pm
- Optional Guerrilla section Saturday 9/13 @ 12:30pm in 306 Soda about higher-order functions
 - Organized by Andrew Huang and the readers
 - Work in a group on a problem until everyone in the group understands the solution
- Project party on Monday 9/15, 3pm–4pm in Wozniak Lounge and 6pm–8pm in 2050 VLSB

Announcements

- Homework 2 due Monday 9/15 @ 11:59pm
- Project 1 due Wednesday 9/17 @ 11:59pm
- Optional Guerrilla section Saturday 9/13 @ 12:30pm in 306 Soda about higher-order functions
 - Organized by Andrew Huang and the readers
 - Work in a group on a problem until everyone in the group understands the solution
- Project party on Monday 9/15, 3pm–4pm in Wozniak Lounge and 6pm–8pm in 2050 VLSB
- Midterm 1 on Monday 9/22 from 7pm to 9pm

Announcements

- Homework 2 due Monday 9/15 @ 11:59pm
- Project 1 due Wednesday 9/17 @ 11:59pm
- Optional Guerrilla section Saturday 9/13 @ 12:30pm in 306 Soda about higher-order functions
 - Organized by Andrew Huang and the readers
 - Work in a group on a problem until everyone in the group understands the solution
- Project party on Monday 9/15, 3pm–4pm in Wozniak Lounge and 6pm–8pm in 2050 VLSB
- Midterm 1 on Monday 9/22 from 7pm to 9pm
 - Details and review materials will be posted next week

Announcements

- Homework 2 due Monday 9/15 @ 11:59pm
- Project 1 due Wednesday 9/17 @ 11:59pm
- Optional Guerrilla section Saturday 9/13 @ 12:30pm in 306 Soda about higher-order functions
 - Organized by Andrew Huang and the readers
 - Work in a group on a problem until everyone in the group understands the solution
- Project party on Monday 9/15, 3pm–4pm in Wozniak Lounge and 6pm–8pm in 2050 VLSB
- Midterm 1 on Monday 9/22 from 7pm to 9pm
 - Details and review materials will be posted next week
 - There will be a web form for students who cannot attend due to a conflict

Announcements

- Homework 2 due Monday 9/15 @ 11:59pm
- Project 1 due Wednesday 9/17 @ 11:59pm
- Optional Guerrilla section Saturday 9/13 @ 12:30pm in 306 Soda about higher-order functions
 - Organized by Andrew Huang and the readers
 - Work in a group on a problem until everyone in the group understands the solution
- Project party on Monday 9/15, 3pm–4pm in Wozniak Lounge and 6pm–8pm in 2050 VLSB
- Midterm 1 on Monday 9/22 from 7pm to 9pm
 - Details and review materials will be posted next week
 - There will be a web form for students who cannot attend due to a conflict
- There's a pinned Piazza thread to find partners

Lambda Expressions

(Demo)

Lambda Expressions

Lambda Expressions

```
>>> x = 10
```

Lambda Expressions

```
>>> x = 10
```

```
>>> square = x * x
```


Lambda Expressions

```
>>> x = 10
```

An expression: this one evaluates to a number

```
>>> square = x * x
```

Lambda Expressions

```
>>> x = 10
```

An expression: this one evaluates to a number

```
>>> square = x * x
```

```
>>> square = lambda x: x * x
```

Lambda Expressions

```
>>> x = 10
```

An expression: this one evaluates to a number

```
>>> square = x * x
```

Also an expression: evaluates to a function

```
>>> square = lambda x: x * x
```

Lambda Expressions

```
>>> x = 10
```

An expression: this one evaluates to a number

```
>>> square = x * x
```

Also an expression: evaluates to a function

```
>>> square = lambda x: x * x
```

A function

Lambda Expressions

```
>>> x = 10
```

An expression: this one evaluates to a number

```
>>> square = x * x
```

Also an expression: evaluates to a function

```
>>> square = lambda x: x * x
```

A function

with formal parameter x

Lambda Expressions

```
>>> x = 10
```

An expression: this one evaluates to a number

```
>>> square = x * x
```

Also an expression: evaluates to a function

```
>>> square = lambda x: x * x
```

A function

with formal parameter `x`

that returns the value of `"x * x"`

Lambda Expressions

```
>>> x = 10
```

An expression: this one evaluates to a number

```
>>> square = x * x
```

Also an expression: evaluates to a function

```
>>> square = lambda x: x * x
```

Important: No "return" keyword!

A function

with formal parameter `x`

that returns the value of `"x * x"`

Lambda Expressions

```
>>> x = 10
```

An expression: this one evaluates to a number

```
>>> square = x * x
```

Also an expression: evaluates to a function

```
>>> square = lambda x: x * x
```

Important: No "return" keyword!

A function

with formal parameter `x`

that returns the value of `"x * x"`

Must be a single expression

Lambda Expressions

```
>>> x = 10
```

An expression: this one evaluates to a number

```
>>> square = x * x
```

Also an expression: evaluates to a function

```
>>> square = lambda x: x * x
```

Important: No "return" keyword!

A function

with formal parameter `x`

that returns the value of `"x * x"`

```
>>> square(4)  
16
```

Must be a single expression

Lambda Expressions

```
>>> x = 10
```

An expression: this one evaluates to a number

```
>>> square = x * x
```

Also an expression: evaluates to a function

```
>>> square = lambda x: x * x
```

Important: No "return" keyword!

A function
with formal parameter x
that returns the value of "x * x"

```
>>> square(4)
16
```

Must be a single expression

Lambda expressions are not common in Python, but important in general

Lambda Expressions

```
>>> x = 10
```

An expression: this one evaluates to a number

```
>>> square = x * x
```

Also an expression: evaluates to a function

```
>>> square = lambda x: x * x
```

Important: No "return" keyword!

A function

with formal parameter `x`

that returns the value of `'x * x'`

```
>>> square(4)
16
```

Must be a single expression

Lambda expressions are not common in Python, but important in general

Lambda expressions in Python cannot contain statements at all!

Lambda Expressions Versus Def Statements

Lambda Expressions Versus Def Statements

VS

Lambda Expressions Versus Def Statements



square = lambda x: x * x

VS

Lambda Expressions Versus Def Statements



```
square = lambda x: x * x
```

VS

```
def square(x):  
    return x * x
```



Lambda Expressions Versus Def Statements



```
square = lambda x: x * x
```

VS

```
def square(x):  
    return x * x
```



- Both create a function with the same domain, range, and behavior.

Lambda Expressions Versus Def Statements



```
square = lambda x: x * x
```

VS

```
def square(x):  
    return x * x
```



- Both create a function with the same domain, range, and behavior.
- Both functions have as their parent the frame in which they were defined.

Lambda Expressions Versus Def Statements



```
square = lambda x: x * x
```

VS

```
def square(x):  
    return x * x
```



- Both create a function with the same domain, range, and behavior.
- Both functions have as their parent the frame in which they were defined.
- Both bind that function to the name square.

Lambda Expressions Versus Def Statements



```
square = lambda x: x * x
```

VS

```
def square(x):  
    return x * x
```



- Both create a function with the same domain, range, and behavior.
- Both functions have as their parent the frame in which they were defined.
- Both bind that function to the name square.
- Only the def statement gives the function an intrinsic name.

Lambda Expressions Versus Def Statements



```
square = lambda x: x * x
```

VS

```
def square(x):  
    return x * x
```



- Both create a function with the same domain, range, and behavior.
- Both functions have as their parent the frame in which they were defined.
- Both bind that function to the name square.
- Only the def statement gives the function an intrinsic name.

```
Global frame
```

square	func λ(x) <line 1> [parent=Global]
--------	------------------------------------

```
f1: λ <line 1> [parent=Global]
```

x	4
Return value	16

Lambda Expressions Versus Def Statements



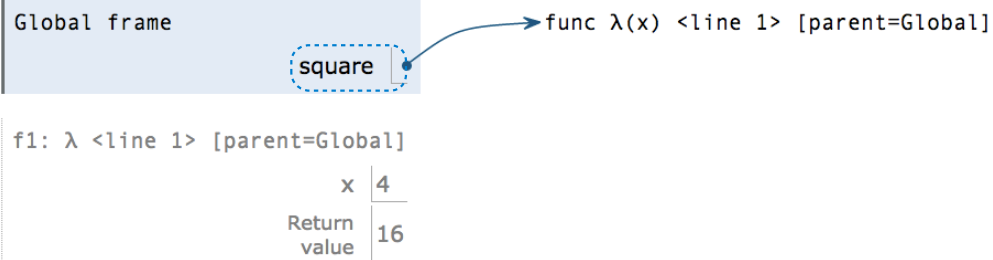
```
square = lambda x: x * x
```

VS

```
def square(x):  
    return x * x
```



- Both create a function with the same domain, range, and behavior.
- Both functions have as their parent the frame in which they were defined.
- Both bind that function to the name square.
- Only the def statement gives the function an intrinsic name.



Lambda Expressions Versus Def Statements



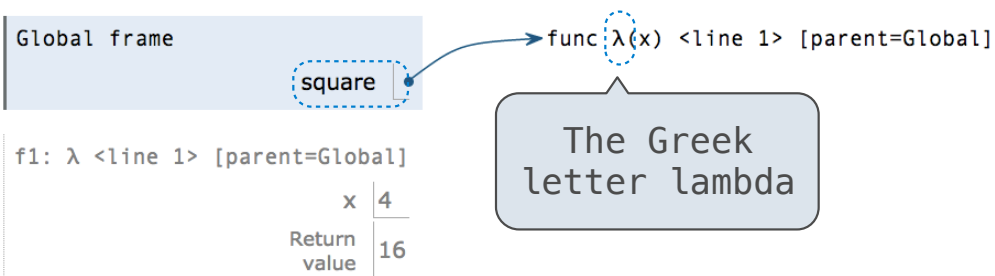
```
square = lambda x: x * x
```

VS

```
def square(x):  
    return x * x
```



- Both create a function with the same domain, range, and behavior.
- Both functions have as their parent the frame in which they were defined.
- Both bind that function to the name square.
- Only the def statement gives the function an intrinsic name.



Lambda Expressions Versus Def Statements



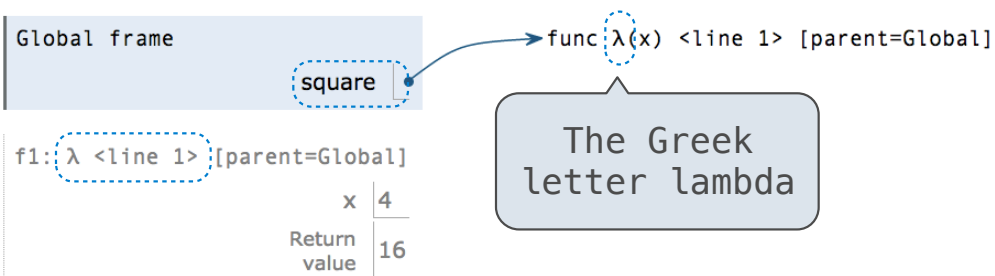
```
square = lambda x: x * x
```

VS

```
def square(x):  
    return x * x
```



- Both create a function with the same domain, range, and behavior.
- Both functions have as their parent the frame in which they were defined.
- Both bind that function to the name square.
- Only the def statement gives the function an intrinsic name.



Lambda Expressions Versus Def Statements



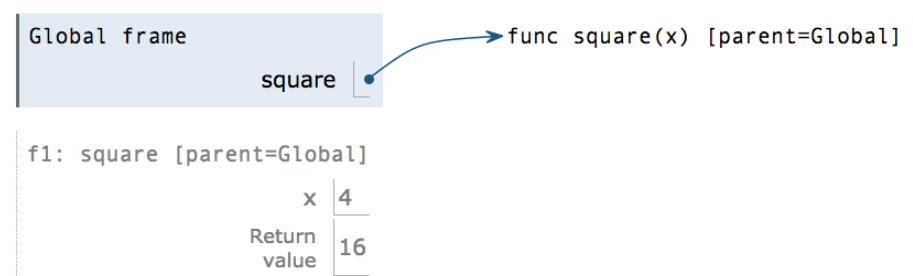
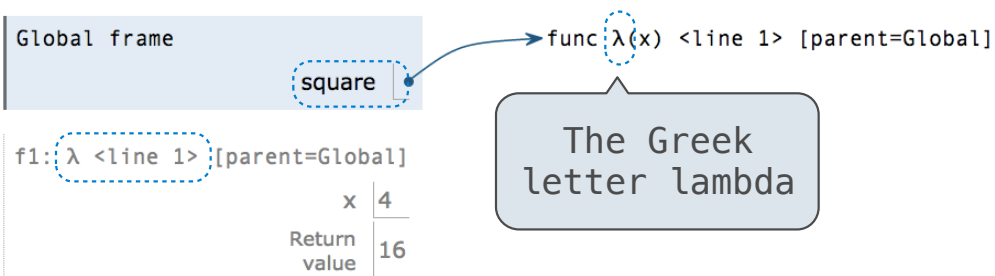
```
square = lambda x: x * x
```

VS



```
def square(x):  
    return x * x
```

- Both create a function with the same domain, range, and behavior.
- Both functions have as their parent the frame in which they were defined.
- Both bind that function to the name square.
- Only the def statement gives the function an intrinsic name.



Currying

Function Currying

Function Currying

```
def make_adder(n):  
    return lambda k: n + k
```

Function Currying

```
def make_adder(n):  
    return lambda k: n + k
```

```
>>> make_adder(2)(3)  
5  
>>> add(2, 3)  
5
```

Function Currying

```
def make_adder(n):  
    return lambda k: n + k
```

```
>>> make_adder(2)(3)  
5  
>>> add(2, 3)  
5
```

There's a general
relationship between
these functions

Function Currying

```
def make_adder(n):  
    return lambda k: n + k
```

```
>>> make_adder(2)(3)  
5  
>>> add(2, 3)  
5
```

There's a general
relationship between
these functions

(Demo)

Function Currying

```
def make_adder(n):  
    return lambda k: n + k
```

```
>>> make_adder(2)(3)  
5  
>>> add(2, 3)  
5
```

There's a general
relationship between
these functions

(Demo)

Curry: Transform a multi-argument function into a single-argument, higher-order function.

Function Currying

```
def make_adder(n):  
    return lambda k: n + k
```

```
>>> make_adder(2)(3)  
5  
>>> add(2, 3)  
5
```

There's a general
relationship between
these functions

(Demo)

Curry: Transform a multi-argument function into a single-argument, higher-order function.

Currying was discovered by Moses Schönfinkel and re-discovered by Haskell Curry.

Function Currying

```
def make_adder(n):  
    return lambda k: n + k
```

```
>>> make_adder(2)(3)  
5  
>>> add(2, 3)  
5
```

There's a general
relationship between
these functions

(Demo)

Curry: Transform a multi-argument function into a single-argument, higher-order function.

Currying was discovered by Moses Schönfinkel and re-discovered by Haskell Curry.

Schönfinkeling?

Newton's Method

Newton's Method Background

Quickly finds accurate approximations to zeroes of differentiable functions!

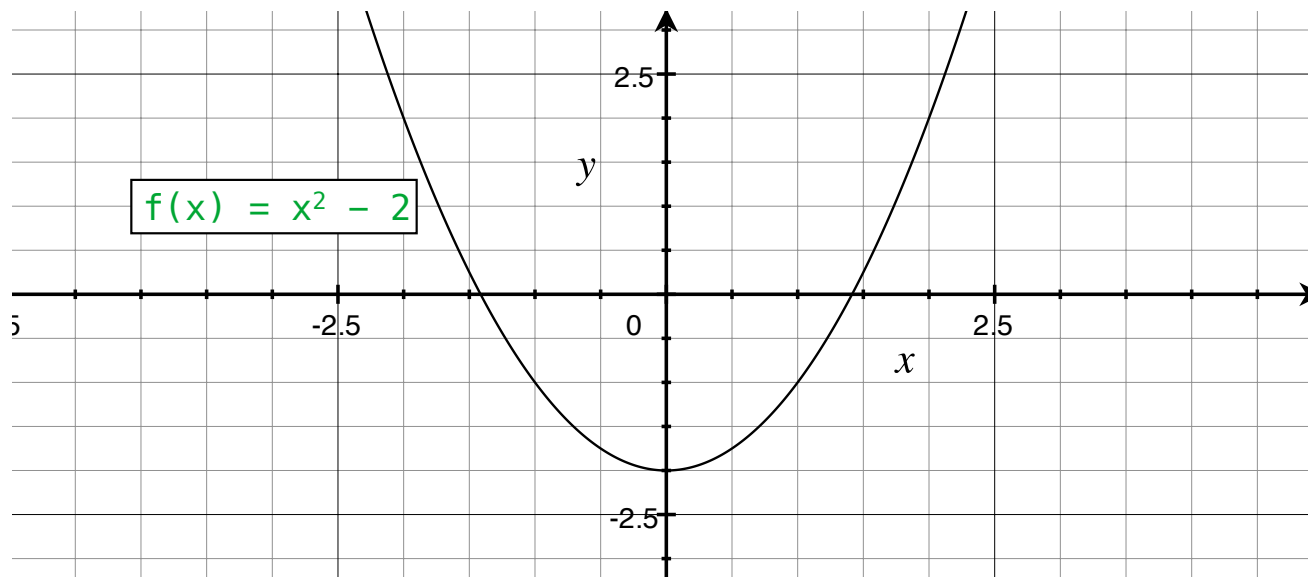
Newton's Method Background

Quickly finds accurate approximations to zeroes of differentiable functions!

$$f(x) = x^2 - 2$$

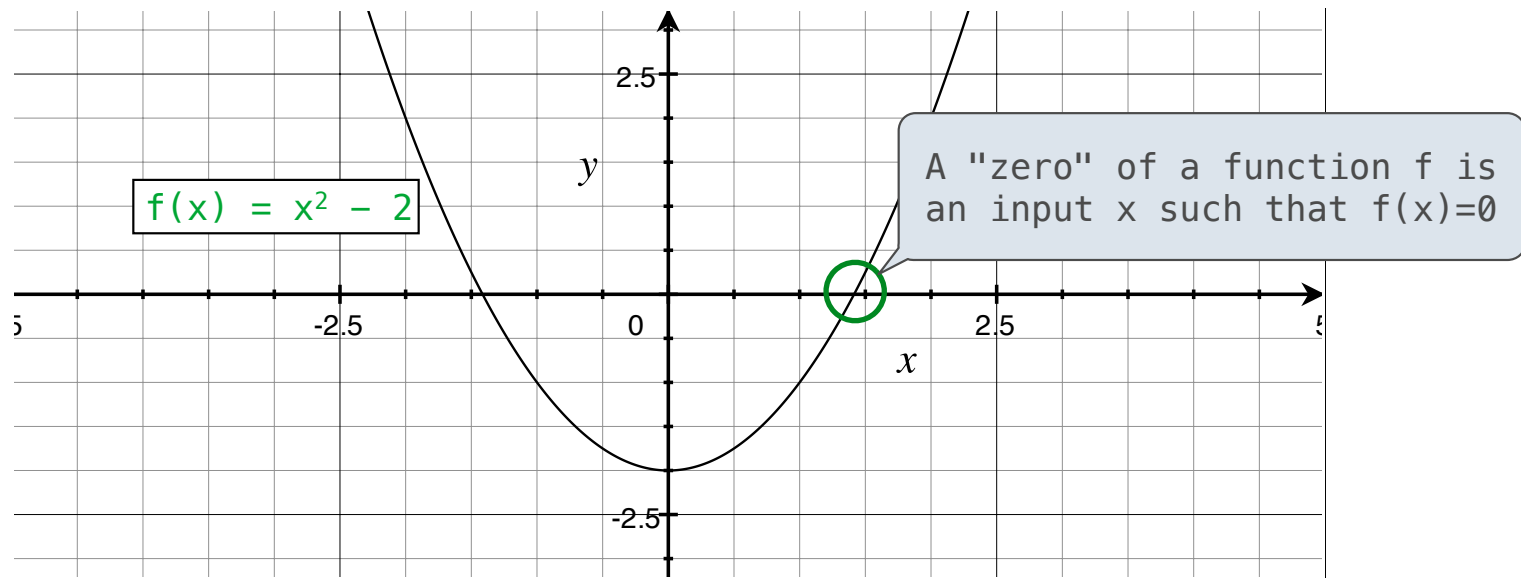
Newton's Method Background

Quickly finds accurate approximations to zeroes of differentiable functions!



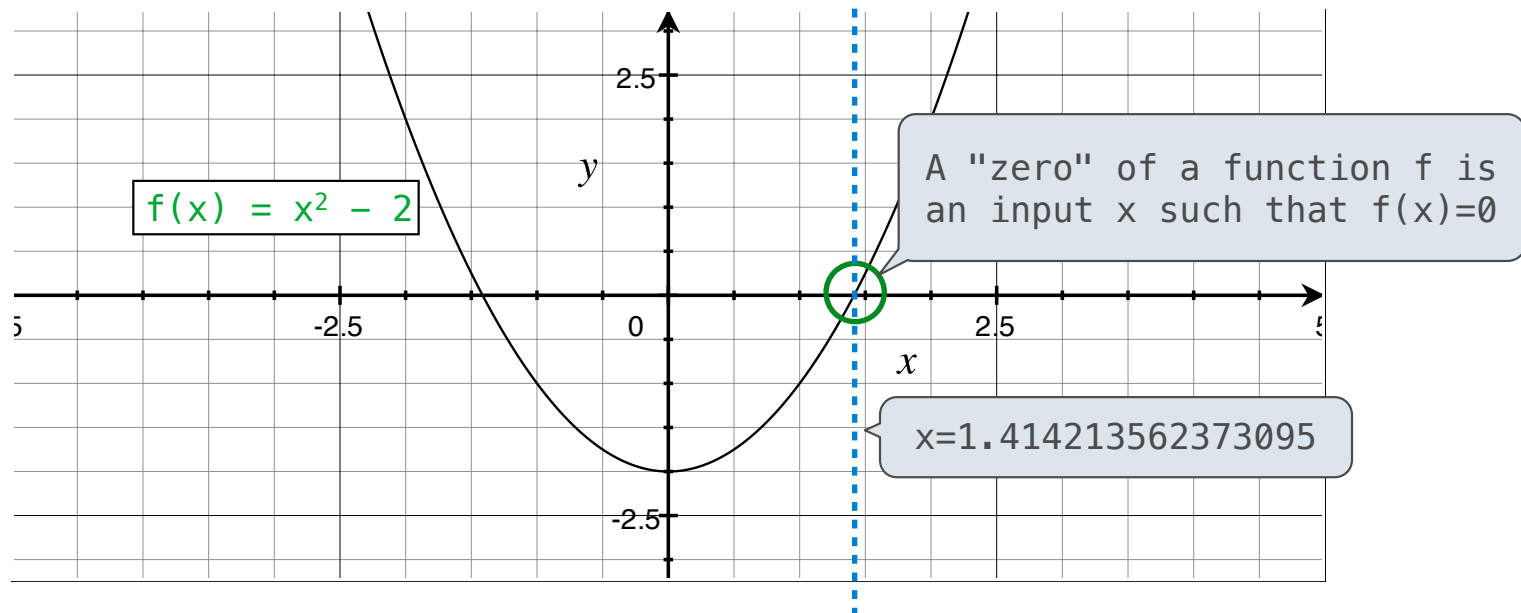
Newton's Method Background

Quickly finds accurate approximations to zeroes of differentiable functions!



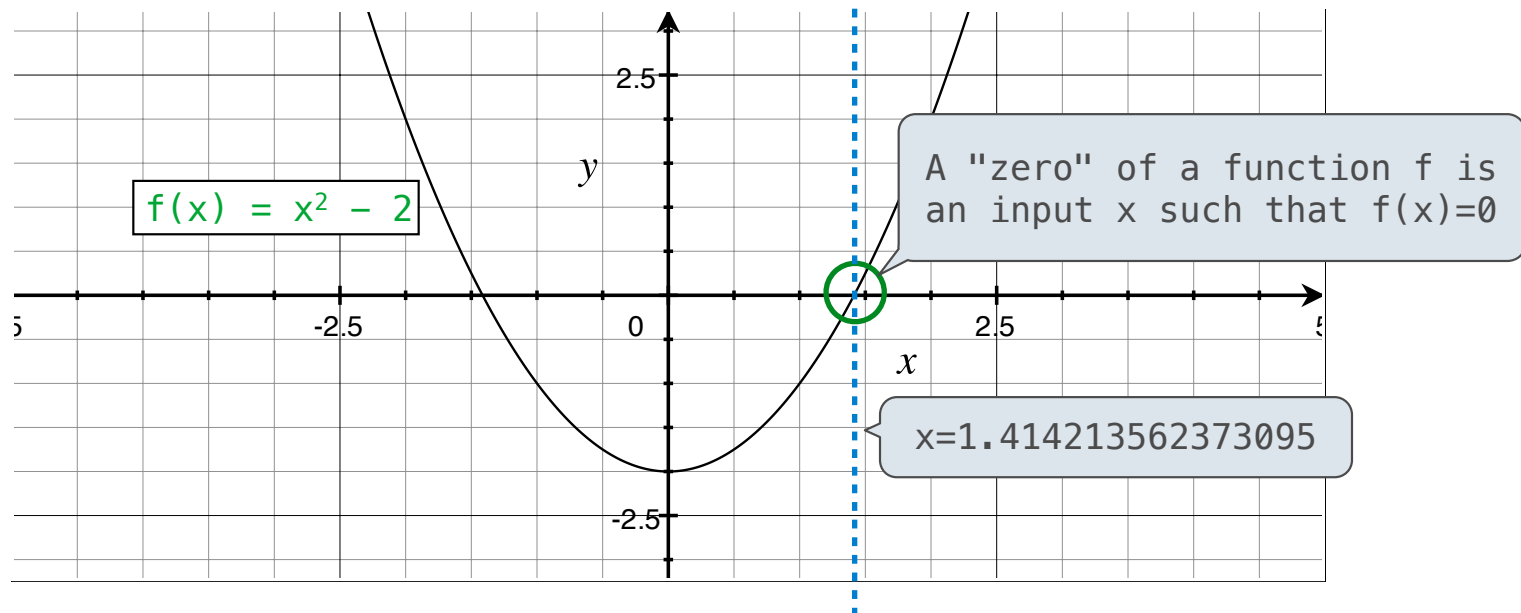
Newton's Method Background

Quickly finds accurate approximations to zeroes of differentiable functions!



Newton's Method Background

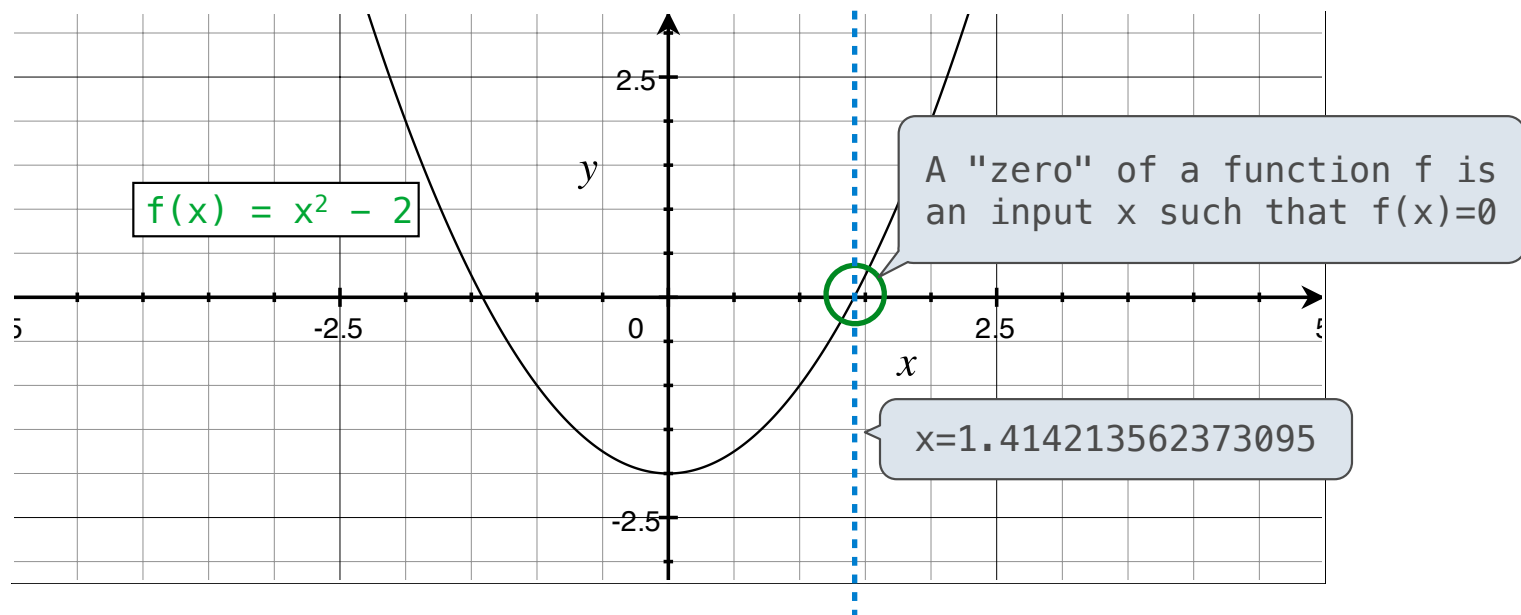
Quickly finds accurate approximations to zeroes of differentiable functions!



Application: a method for computing square roots, cube roots, etc.

Newton's Method Background

Quickly finds accurate approximations to zeroes of differentiable functions!



Application: a method for computing square roots, cube roots, etc.

The positive zero of $f(x) = x^2 - a$ is \sqrt{a} . (We're solving the equation $x^2 = a$.)

Newton's Method

Given a function f and initial guess x ,

Newton's Method

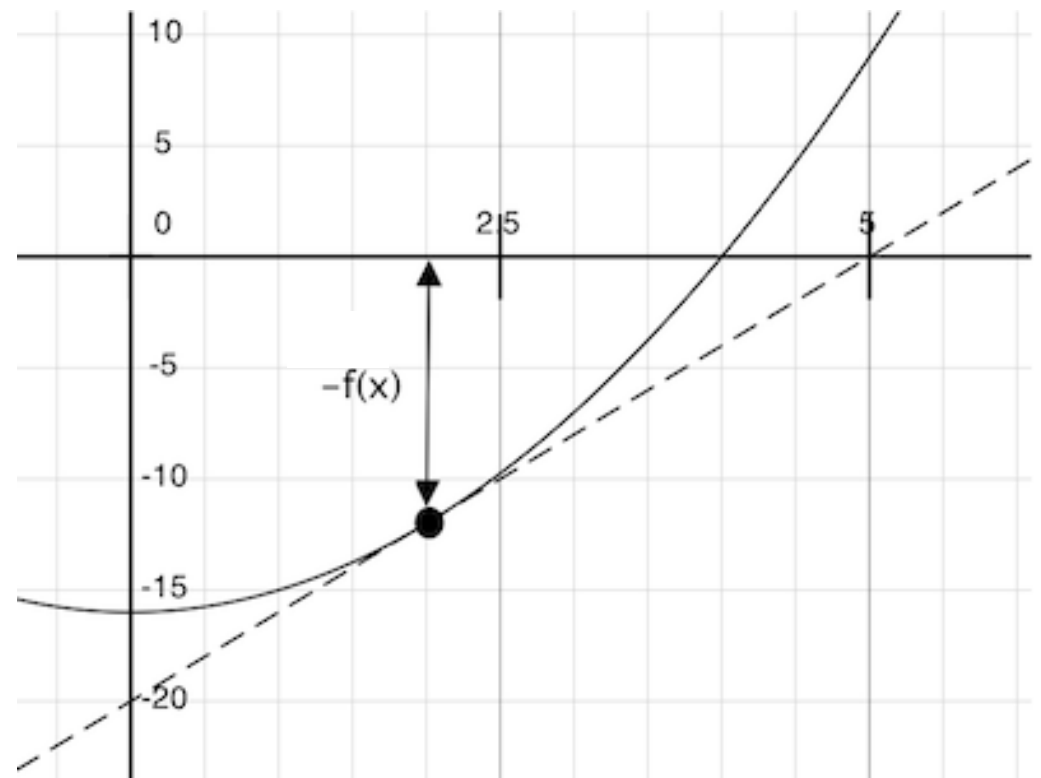
Given a function f and initial guess x ,

Repeatedly improve x :

Newton's Method

Given a function f and initial guess x ,

Repeatedly improve x :

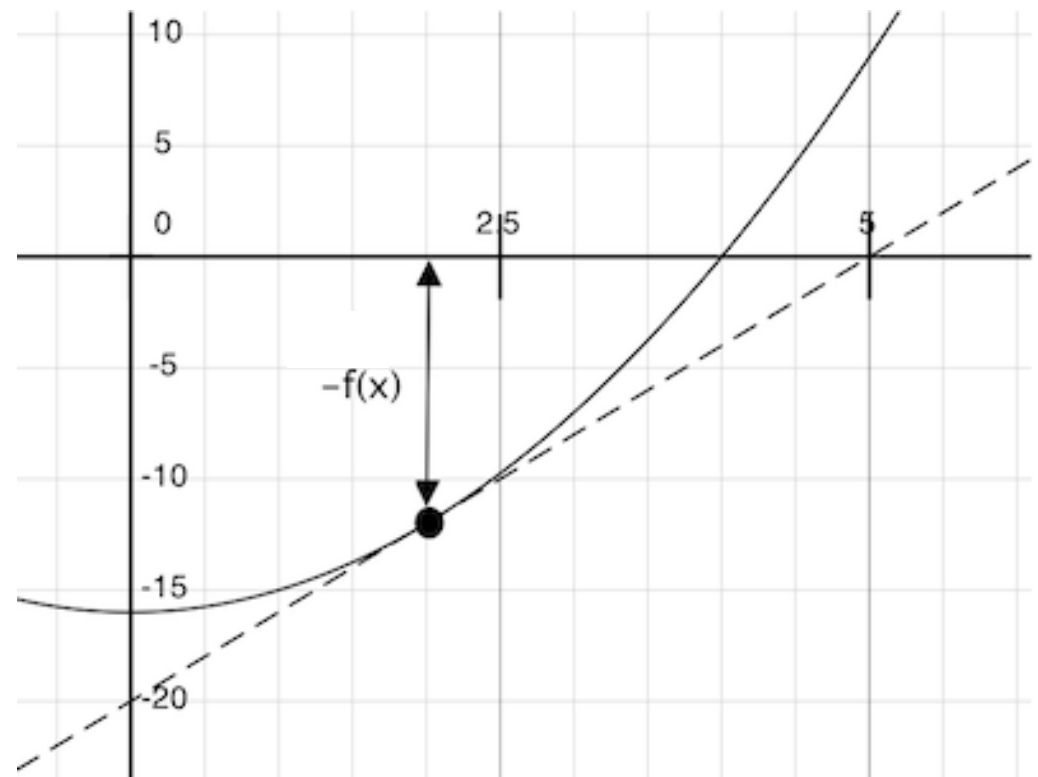


Newton's Method

Given a function f and initial guess x ,

Repeatedly improve x :

Compute the value of f
at the guess: $f(x)$



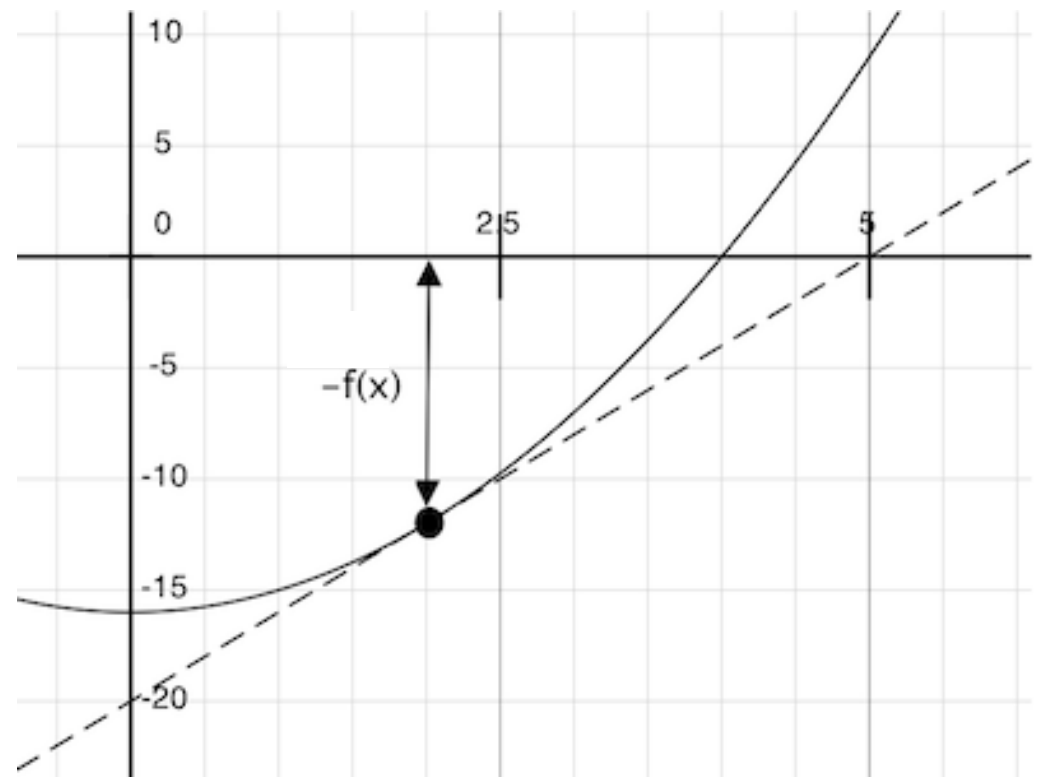
Newton's Method

Given a function f and initial guess x ,

Repeatedly improve x :

Compute the value of f
at the guess: $f(x)$

Compute the derivative
of f at the guess: $f'(x)$



Newton's Method

Given a function f and initial guess x ,

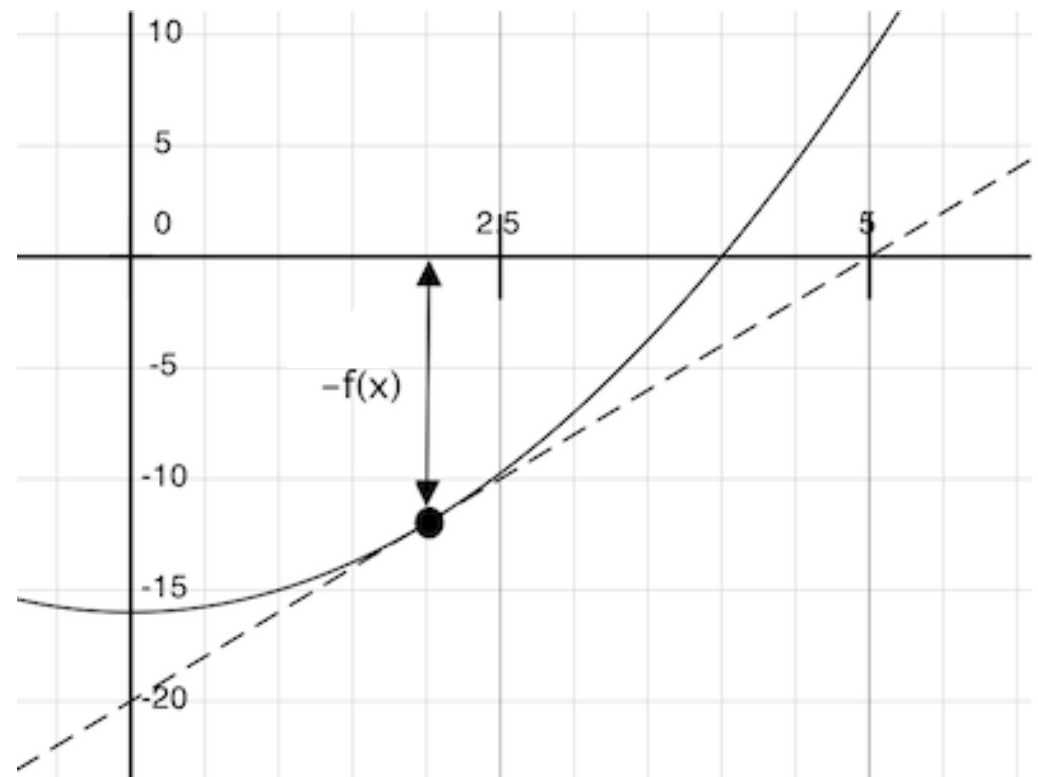
Repeatedly improve x :

Compute the value of f
at the guess: $f(x)$

Compute the derivative
of f at the guess: $f'(x)$

Update guess x to be:

$$x - \frac{f(x)}{f'(x)}$$



Newton's Method

Given a function f and initial guess x ,

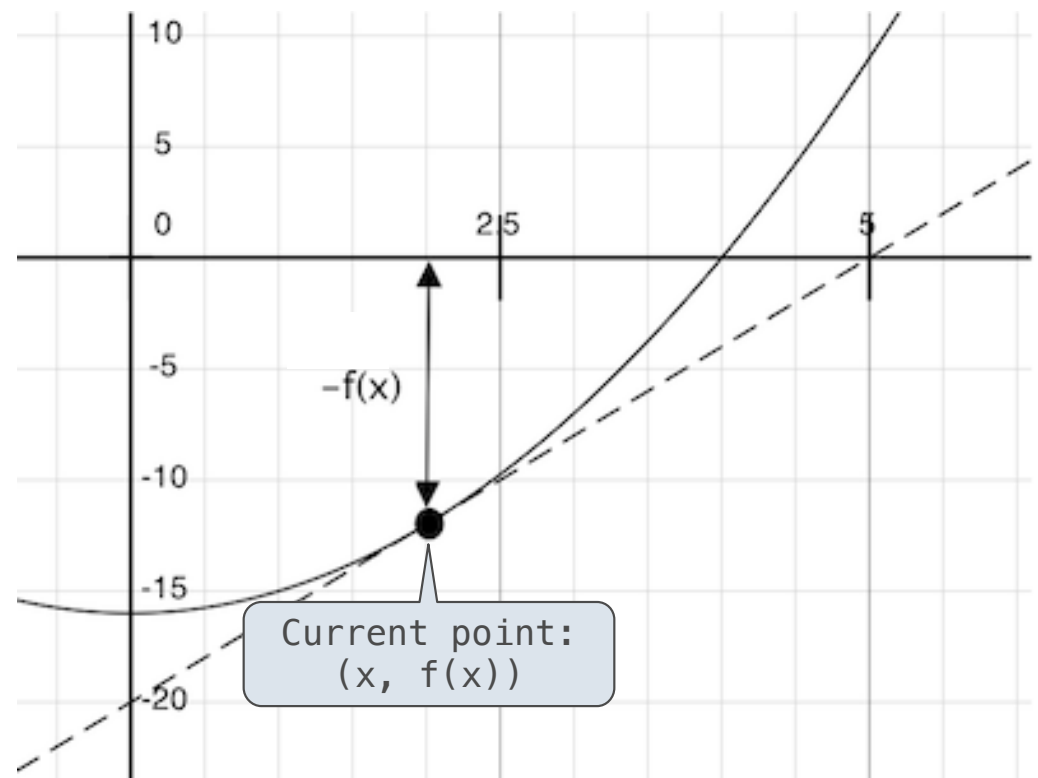
Repeatedly improve x :

Compute the value of f
at the guess: $f(x)$

Compute the derivative
of f at the guess: $f'(x)$

Update guess x to be:

$$x - \frac{f(x)}{f'(x)}$$



Newton's Method

Given a function f and initial guess x ,

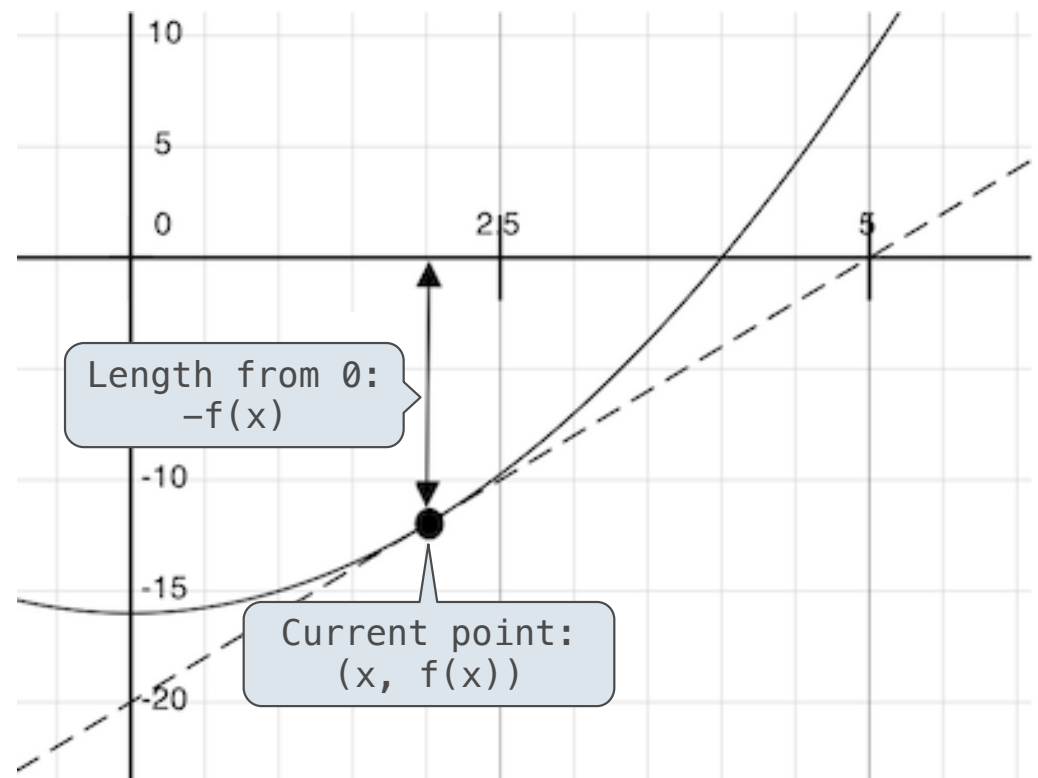
Repeatedly improve x :

Compute the value of f
at the guess: $f(x)$

Compute the derivative
of f at the guess: $f'(x)$

Update guess x to be:

$$x - \frac{f(x)}{f'(x)}$$



Newton's Method

Given a function f and initial guess x ,

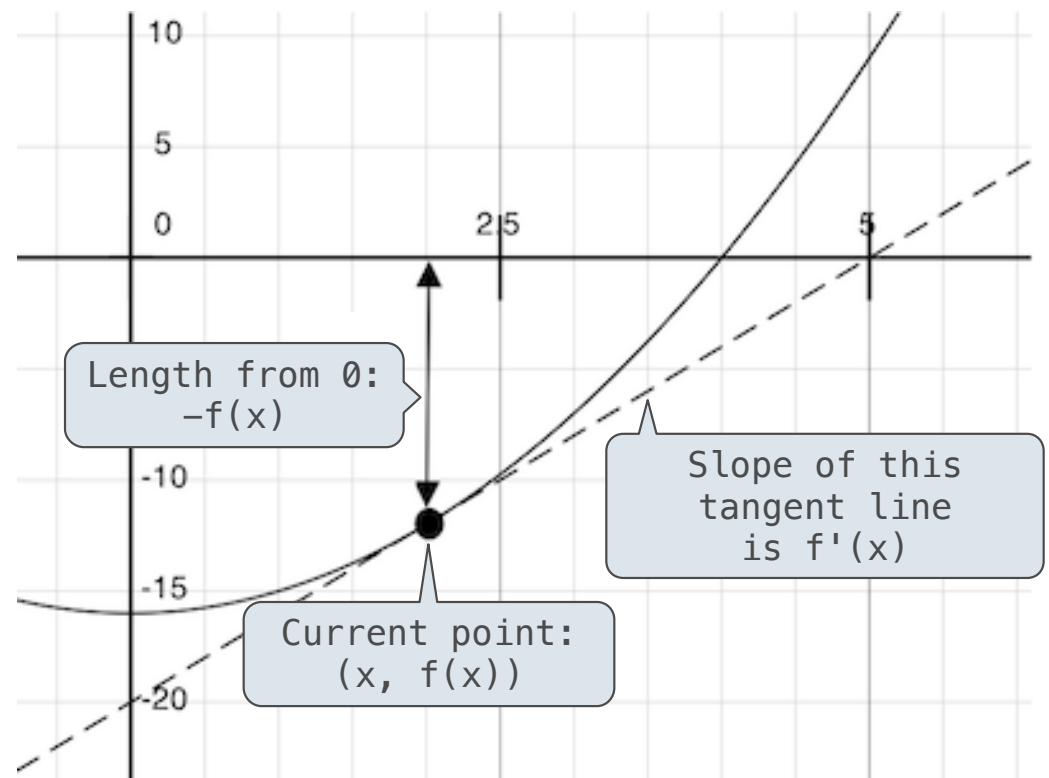
Repeatedly improve x :

Compute the value of f
at the guess: $f(x)$

Compute the derivative
of f at the guess: $f'(x)$

Update guess x to be:

$$x - \frac{f(x)}{f'(x)}$$



Newton's Method

Given a function f and initial guess x ,

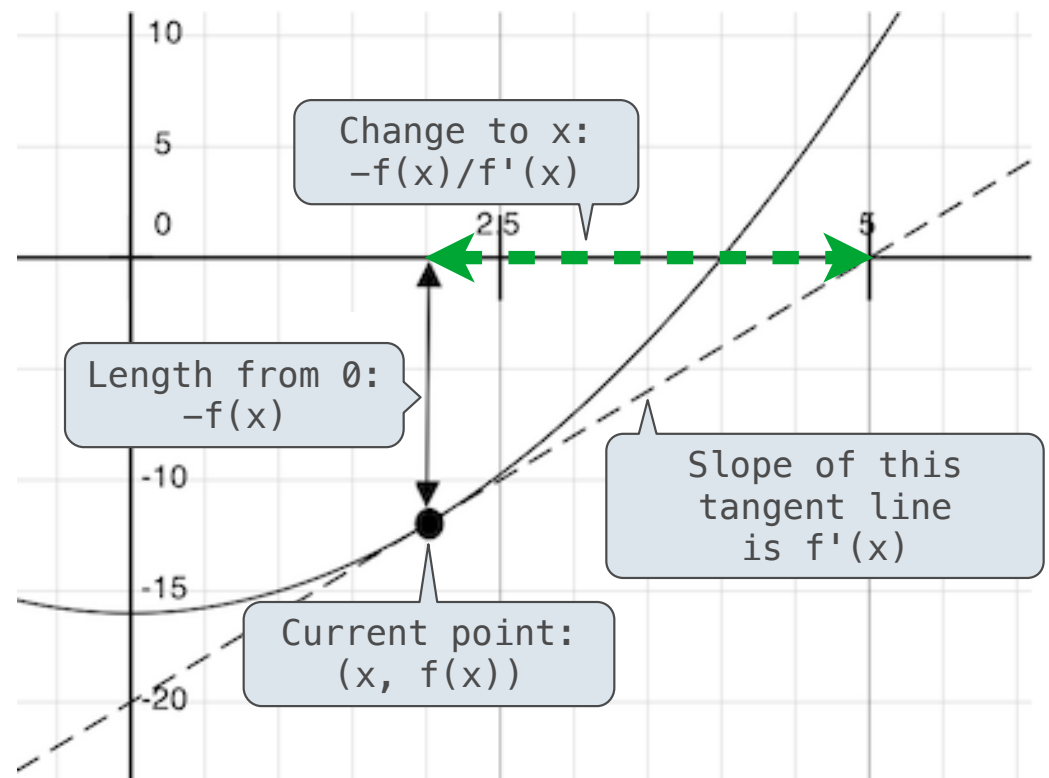
Repeatedly improve x :

Compute the value of f
at the guess: $f(x)$

Compute the derivative
of f at the guess: $f'(x)$

Update guess x to be:

$$x - \frac{f(x)}{f'(x)}$$



Newton's Method

Given a function f and initial guess x ,

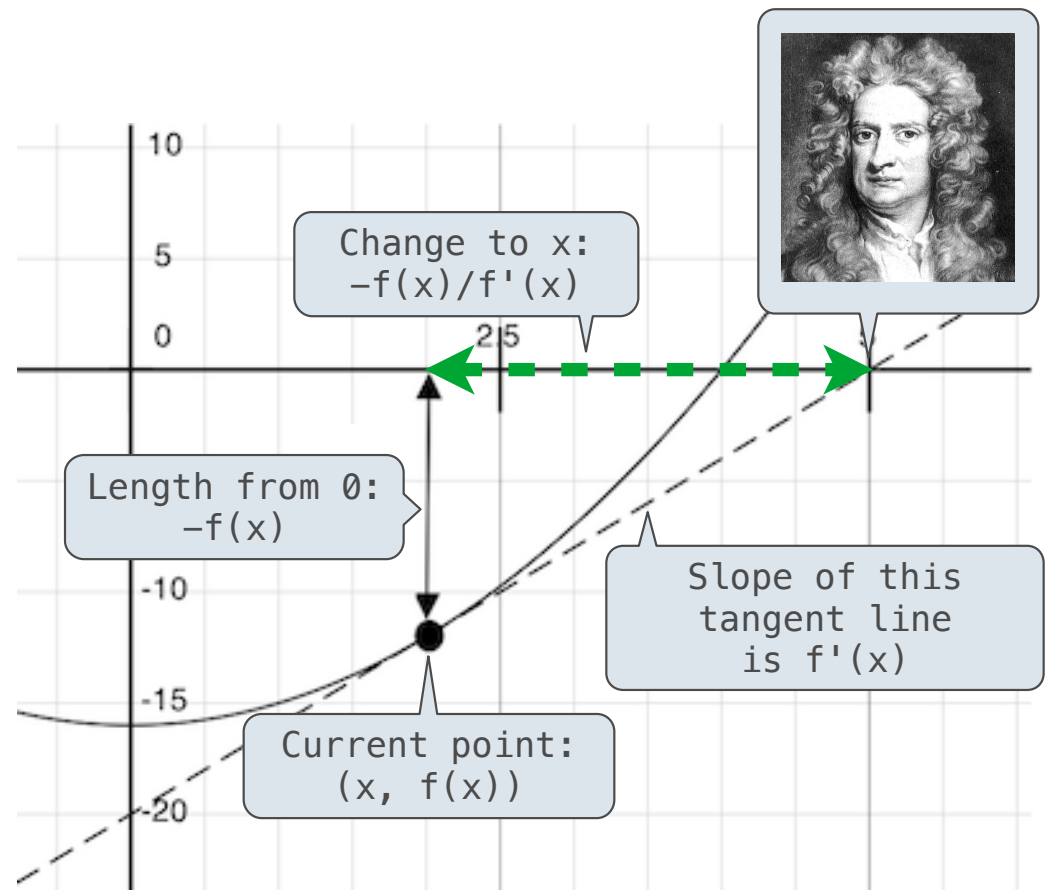
Repeatedly improve x :

Compute the value of f
at the guess: $f(x)$

Compute the derivative
of f at the guess: $f'(x)$

Update guess x to be:

$$x - \frac{f(x)}{f'(x)}$$



Newton's Method

Given a function f and initial guess x ,

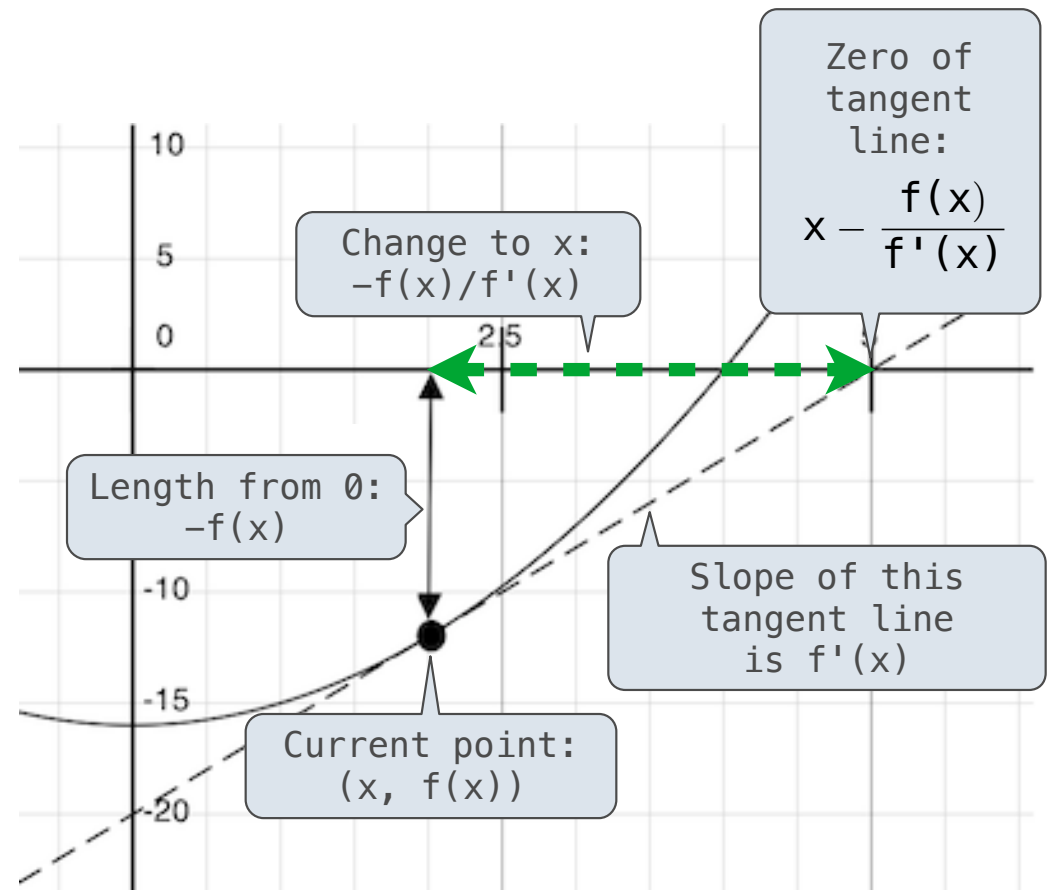
Repeatedly improve x :

Compute the value of f
at the guess: $f(x)$

Compute the derivative
of f at the guess: $f'(x)$

Update guess x to be:

$$x - \frac{f(x)}{f'(x)}$$



Newton's Method

Given a function f and initial guess x ,

Repeatedly improve x :

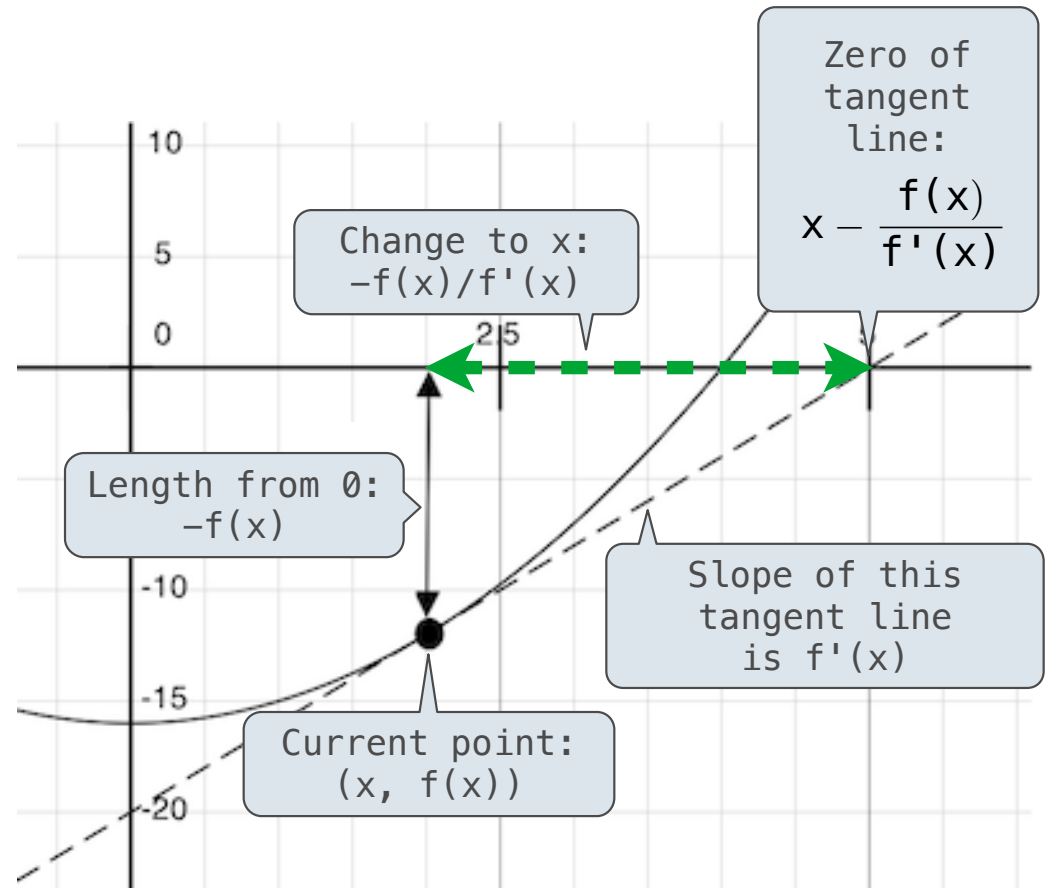
Compute the value of f
at the guess: $f(x)$

Compute the derivative
of f at the guess: $f'(x)$

Update guess x to be:

$$x - \frac{f(x)}{f'(x)}$$

Finish when $f(x) = 0$ (or close enough)



Newton's Method

Given a function f and initial guess x ,

Repeatedly improve x :

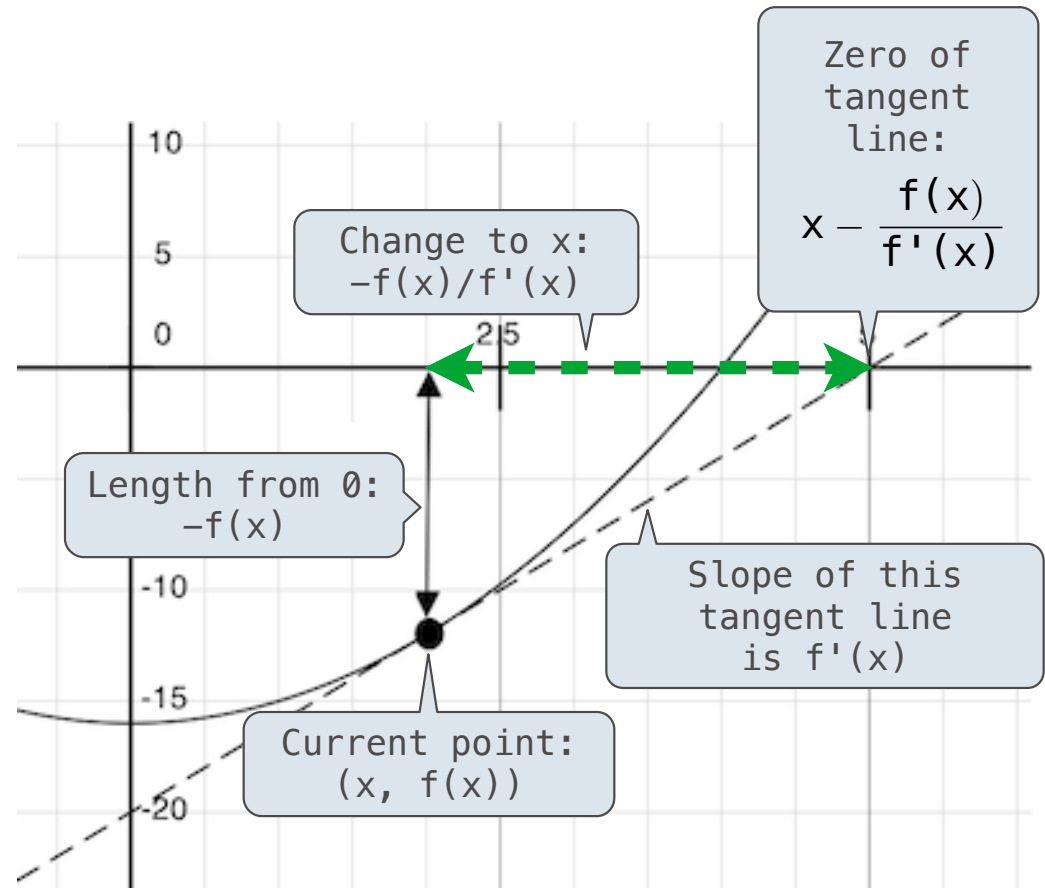
Compute the value of f
at the guess: $f(x)$

Compute the derivative
of f at the guess: $f'(x)$

Update guess x to be:

$$x - \frac{f(x)}{f'(x)}$$

Finish when $f(x) = 0$ (or close enough)



Using Newton's Method

Using Newton's Method

How to find the square root of 2?

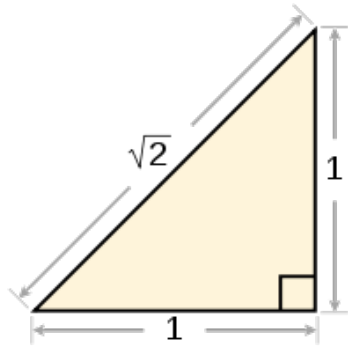
Using Newton's Method

How to find the square root of 2?

```
>>> f = lambda x: x*x - 2
>>> df = lambda x: 2*x
>>> find_zero(f, df)
1.4142135623730951
```

Using Newton's Method

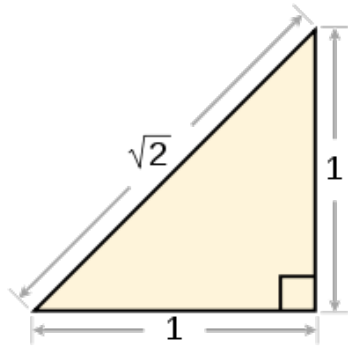
How to find the square root of 2?



```
>>> f = lambda x: x*x - 2
>>> df = lambda x: 2*x
>>> find_zero(f, df)
1.4142135623730951
```

Using Newton's Method

How to find the square root of 2?

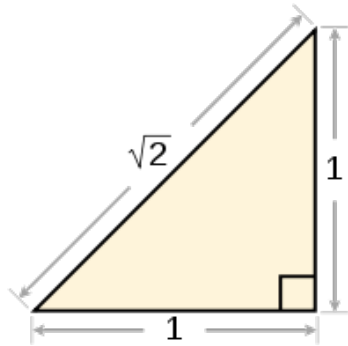


```
>>> f = lambda x: x*x - 2
>>> df = lambda x: 2*x
>>> find_zero(f, df)
1.4142135623730951
```

$$f(x) = x^2 - 2$$
$$f'(x) = 2x$$

Using Newton's Method

How to find the square root of 2?



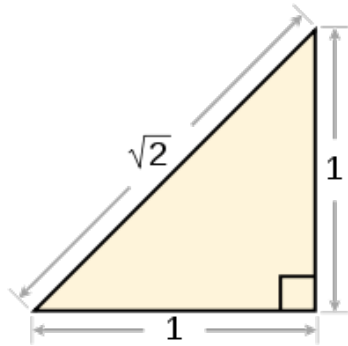
```
>>> f = lambda x: x*x - 2
>>> df = lambda x: 2*x
>>> find_zero(f, df)
1.4142135623730951
```

$$f(x) = x^2 - 2$$
$$f'(x) = 2x$$

Applies Newton's method

Using Newton's Method

How to find the square root of 2?



```
>>> f = lambda x: x*x - 2
>>> df = lambda x: 2*x
>>> find_zero(f, df)
1.4142135623730951
```

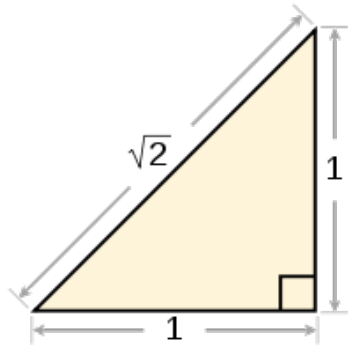
$$f(x) = x^2 - 2$$
$$f'(x) = 2x$$

Applies Newton's method

How to find the cube root of 729?

Using Newton's Method

How to find the square root of 2?

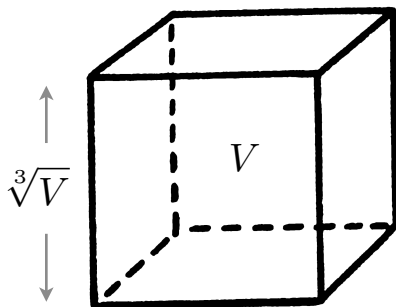


```
>>> f = lambda x: x*x - 2  
>>> df = lambda x: 2*x  
>>> find_zero(f, df)  
1.4142135623730951
```

$$f(x) = x^2 - 2$$
$$f'(x) = 2x$$

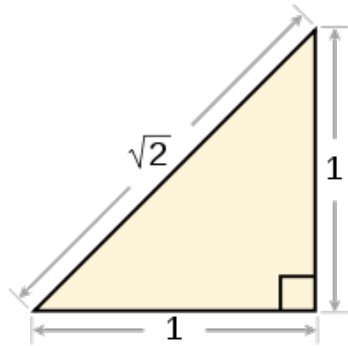
Applies Newton's method

How to find the cube root of 729?



Using Newton's Method

How to find the square root of 2?

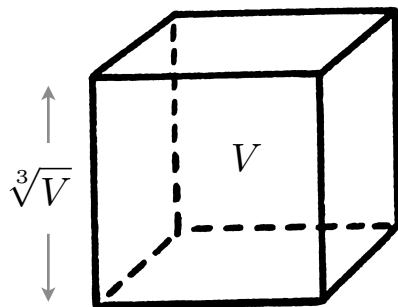


```
>>> f = lambda x: x*x - 2
>>> df = lambda x: 2*x
>>> find_zero(f, df)
1.4142135623730951
```

$$f(x) = x^2 - 2$$
$$f'(x) = 2x$$

Applies Newton's method

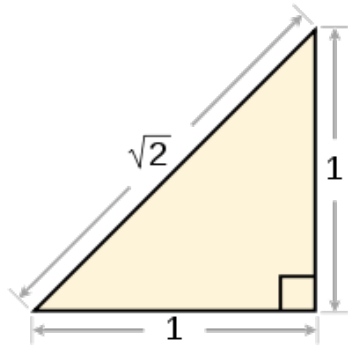
How to find the cube root of 729?



```
>>> g = lambda x: x*x*x - 729
>>> dg = lambda x: 3*x*x
>>> find_zero(g, dg)
9.0
```


Using Newton's Method

How to find the square root of 2?

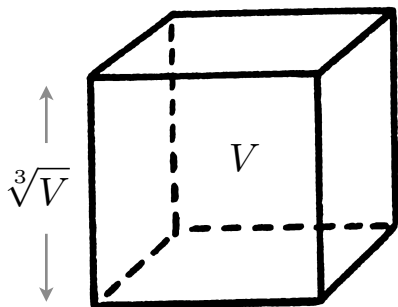


```
>>> f = lambda x: x*x - 2
>>> df = lambda x: 2*x
>>> find_zero(f, df)
1.4142135623730951
```

$$f(x) = x^2 - 2$$
$$f'(x) = 2x$$

Applies Newton's method

How to find the cube root of 729?



```
>>> g = lambda x: x*x*x - 729
>>> dg = lambda x: 3*x*x
>>> find_zero(g, dg)
9.0
```

$$g(x) = x^3 - 729$$
$$g'(x) = 3x^2$$

Iterative Improvement

Special Case: Square Roots

Special Case: Square Roots

How to compute `square_root(a)`

Idea: Iteratively refine a guess x about the square root of a

Special Case: Square Roots

How to compute `square_root(a)`

Idea: Iteratively refine a guess x about the square root of a

Update:

Special Case: Square Roots

How to compute `square_root(a)`

Idea: Iteratively refine a guess x about the square root of a

Update:
$$x = \frac{x + \frac{a}{x}}{2}$$

Special Case: Square Roots

How to compute `square_root(a)`

Idea: Iteratively refine a guess x about the square root of a

Update:
$$x = \frac{x + \frac{a}{x}}{2}$$

Babylonian Method

Special Case: Square Roots

How to compute `square_root(a)`

Idea: Iteratively refine a guess x about the square root of a

Update:
$$x = \frac{x + \frac{a}{x}}{2}$$

Babylonian Method

Implementation questions:

Special Case: Square Roots

How to compute `square_root(a)`

Idea: Iteratively refine a guess x about the square root of a

Update:
$$x = \frac{x + \frac{a}{x}}{2}$$

Babylonian Method

Implementation questions:

What guess should start the computation?

Special Case: Square Roots

How to compute `square_root(a)`

Idea: Iteratively refine a guess x about the square root of a

Update:
$$x = \frac{x + \frac{a}{x}}{2}$$

Babylonian Method

Implementation questions:

What guess should start the computation?

How do we know when we are finished?

Special Case: Cube Roots

Special Case: Cube Roots

How to compute `cube_root(a)`

Idea: Iteratively refine a guess x about the cube root of a

Special Case: Cube Roots

How to compute `cube_root(a)`

Idea: Iteratively refine a guess x about the cube root of a

Update:

Special Case: Cube Roots

How to compute `cube_root(a)`

Idea: Iteratively refine a guess x about the cube root of a

Update:
$$x = \frac{2 \cdot x + \frac{a}{x^2}}{3}$$

Special Case: Cube Roots

How to compute `cube_root(a)`

Idea: Iteratively refine a guess x about the cube root of a

Update:
$$x = \frac{2 \cdot x + \frac{a}{x^2}}{3}$$

Implementation questions:

Special Case: Cube Roots

How to compute `cube_root(a)`

Idea: Iteratively refine a guess x about the cube root of a

Update:
$$x = \frac{2 \cdot x + \frac{a}{x^2}}{3}$$

Implementation questions:

What guess should start the computation?

Special Case: Cube Roots

How to compute `cube_root(a)`

Idea: Iteratively refine a guess `x` about the cube root of `a`

Update:
$$x = \frac{2 \cdot x + \frac{a}{x^2}}{3}$$

Implementation questions:

What guess should start the computation?

How do we know when we are finished?

Implementing Newton's Method

(Demo)