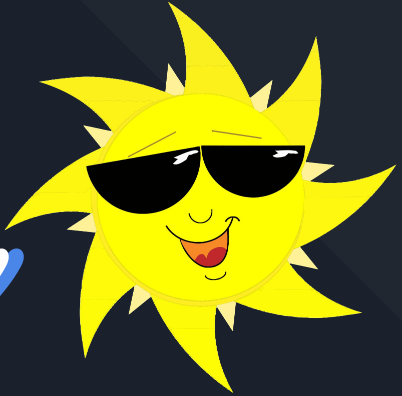
Riemann Summers



Group members:

- Gun Woo Kim
- Leonardo Alves Nunes
- Nicole Moreno Gonzalez
- Slok Rajbhandari

PROJECT IDEA:
- RIEMANN SUMS-

WHAT'S A RIEMANN SUM?

It is an approximation of area under the curve with n rectangles of same width. The height of each rectangles should be the height of the curve.

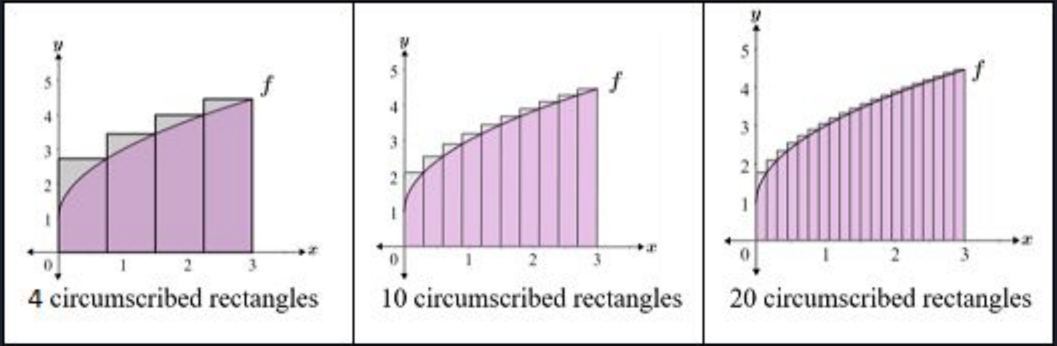
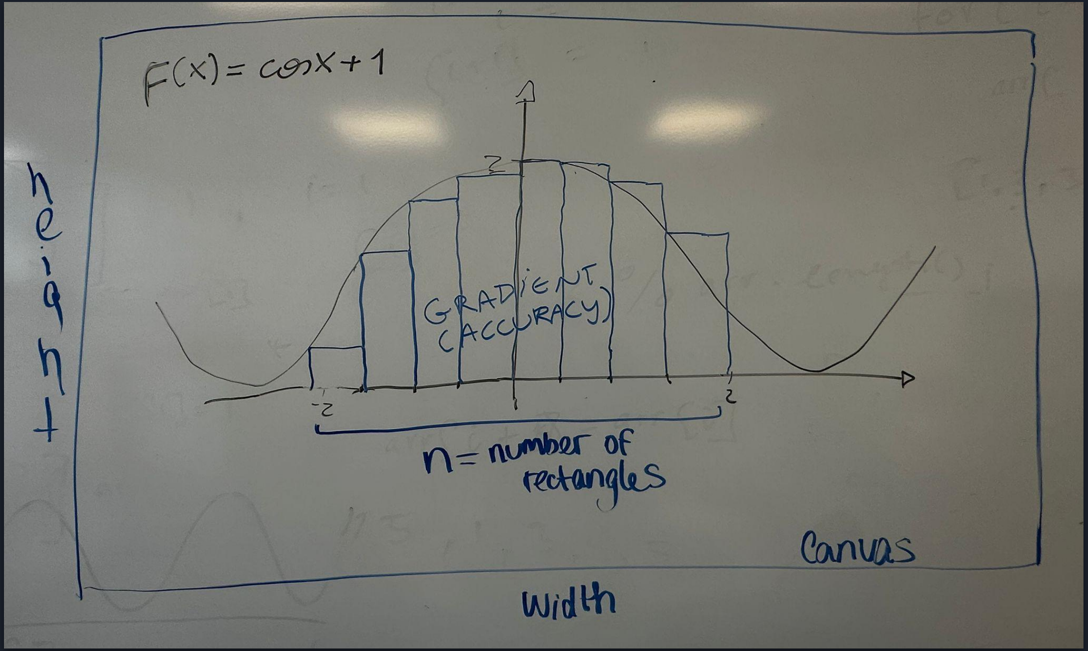


Image source: https://www.math.net/img/a/calculus/integrals/riemann-sum/riemann-sum.png

PROJECT SKETCH/GOALS



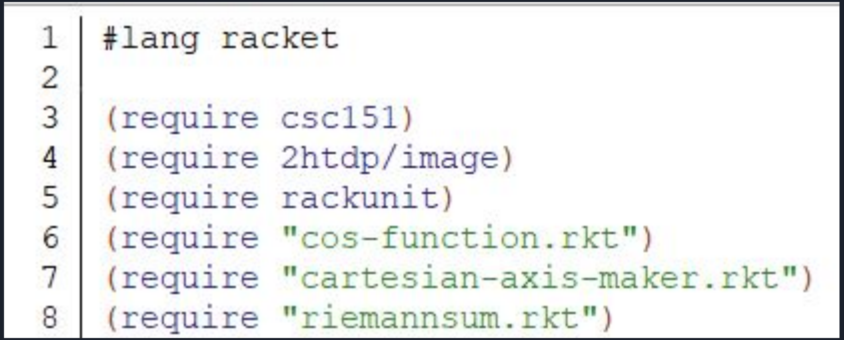
Features and limitations:

- User can input height, width and number of rectangles
- Accuracy gradient
- Fixed function
- Fixed interval

Curve: cos(x) + 1

Interval: $-\pi \rightarrow \pi$

HOW DID WE WORK?



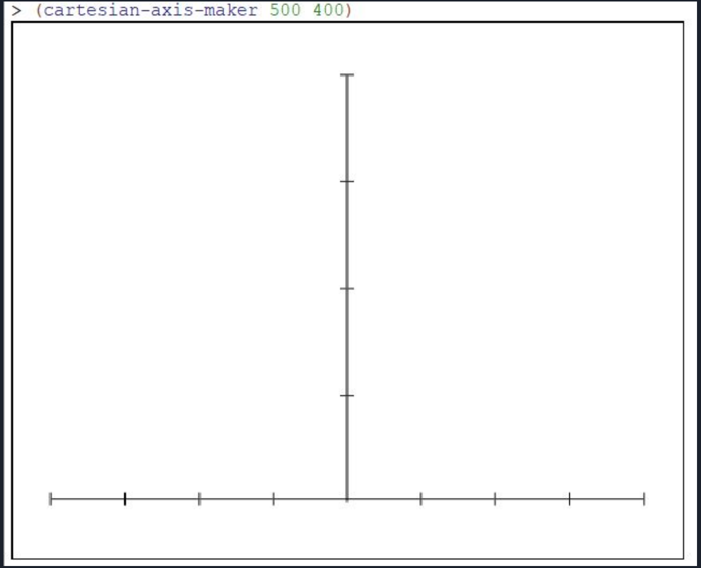
```
#lang racket

(require csc151)
(require 2htdp/image)
(require rackunit)
(require "cos-function.rkt")
(require "cartesian-axis-maker.rkt")
(require "riemannsum.rkt")
```

- Code was organized into separate files
- Main file (images-series.rkt) requires these files to build the final image

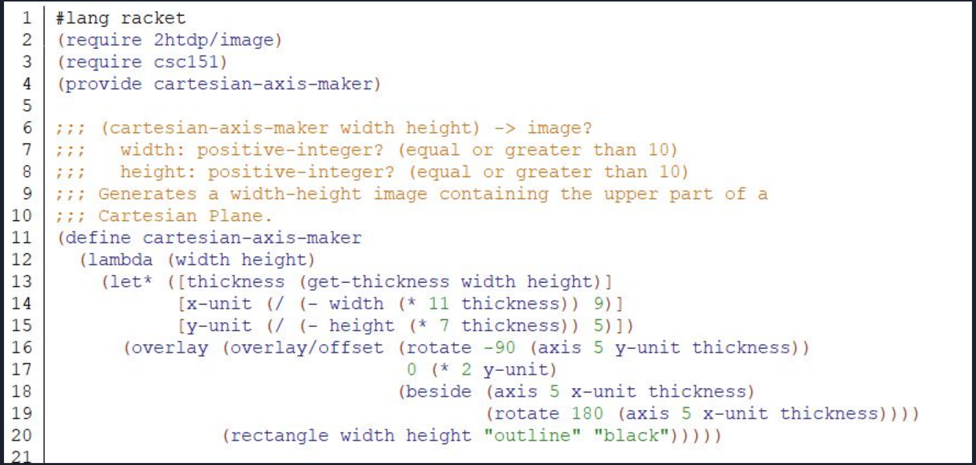
- We used GitHub to facilitate collaborative teamwork.

Decomposing the Solution

Axis creation image



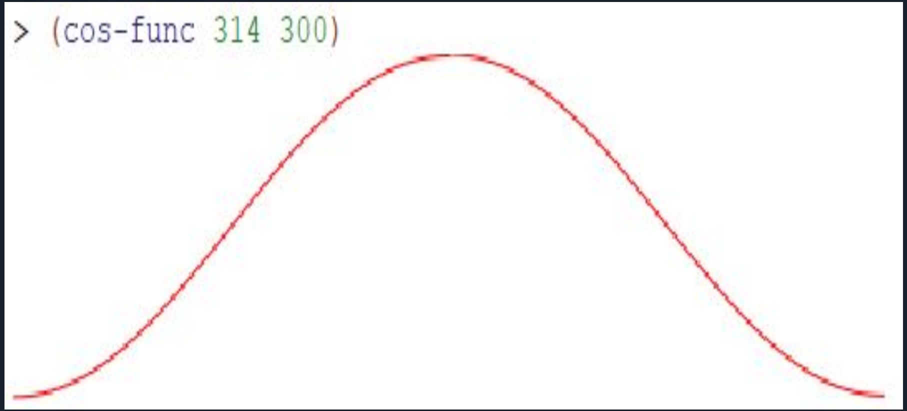
```
> (cartesian-axis-maker 500 400)
```



```
#lang racket
(require 2htdp/image)
(require csc151)
(provide cartesian-axis-maker)

;;; (cartesian-axis-maker width height) -> image?
;;;   width: positive-integer? (equal or greater than 10)
;;;   height: positive-integer? (equal or greater than 10)
;;; Generates a width-height image containing the upper part of a
;;; Cartesian Plane.
(define cartesian-axis-maker
  (lambda (width height)
    (let* ([thickness (get-thickness width height)]
           [x-unit (/ (- width (* 11 thickness)) 9)]
           [y-unit (/ (- height (* 7 thickness)) 5)])
      (overlay (overlay/offset (rotate -90 (axis 5 y-unit thickness))
                               0 (* 2 y-unit)
                              (beside (axis 5 x-unit thickness)
                                      (rotate 180 (axis 5 x-unit thickness))))
               (rectangle width height "outline" "black")))))
```

Axis creation outstanding code

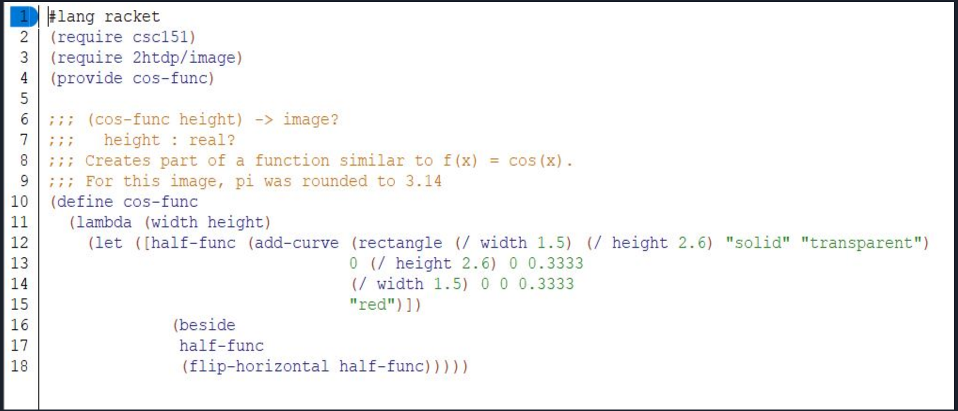
Decomposing the Solution



```
> (cos-func 314 300)
```

Curve creation

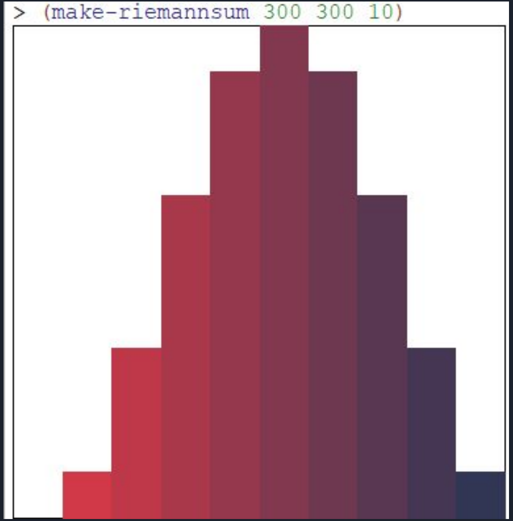
Curve creation outstanding code



```
#lang racket
(require csc151)
(require 2htdp/image)
(provide cos-func)

;;; (cos-func height) -> image?
;;;   height : real?
;;; Creates part of a function similar to f(x) = cos(x).
;;; For this image, pi was rounded to 3.14
(define cos-func
  (lambda (width height)
    (let ([half-func (add-curve (rectangle (/ width 1.5) (/ height 2.6) "solid" "transparent")
                                0 (/ height 2.6) 0 0.3333
                                (/ width 1.5) 0 0 0.3333
                                "red")])
           (beside
            half-func
            (flip-horizontal half-func)))))
```

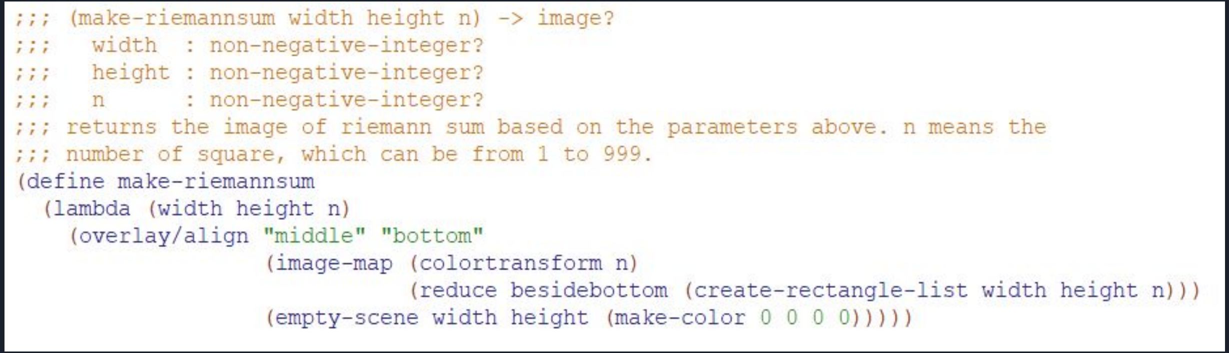
Decomposing the solution

Riemann Sum creation



```
> (make-riemannsum 300 300 10)
```

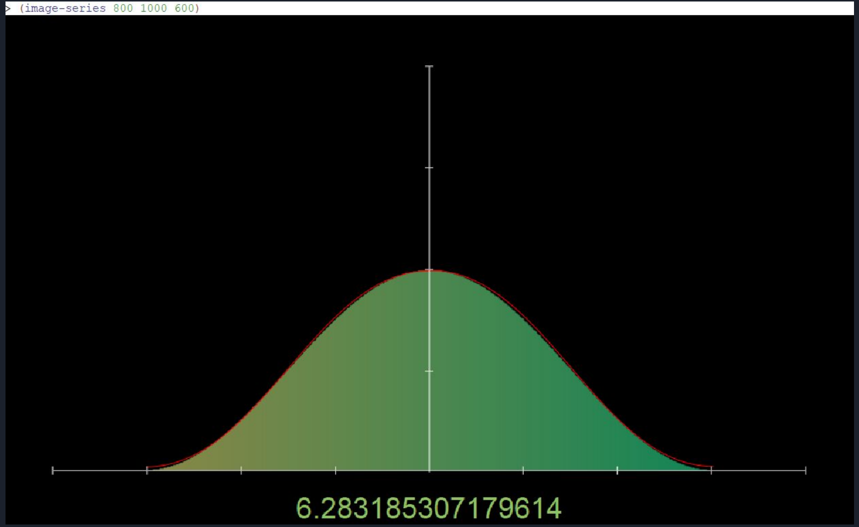
Riemann Sum creation outstanding code



```
;;; (make-riemannsum width height n) -> image?
;;;   width  : non-negative-integer?
;;;   height : non-negative-integer?
;;;   n      : non-negative-integer?
;;; returns the image of riemann sum based on the parameters above. n means the
;;; number of square, which can be from 1 to 999.
(define make-riemannsum
  (lambda (width height n)
    (overlay/align "middle" "bottom"
                   (image-map (colortransform n)
                              (reduce besidebottom (create-rectangle-list width height n)))
                   (empty-scene width height (make-color 0 0 0 0)))))
```

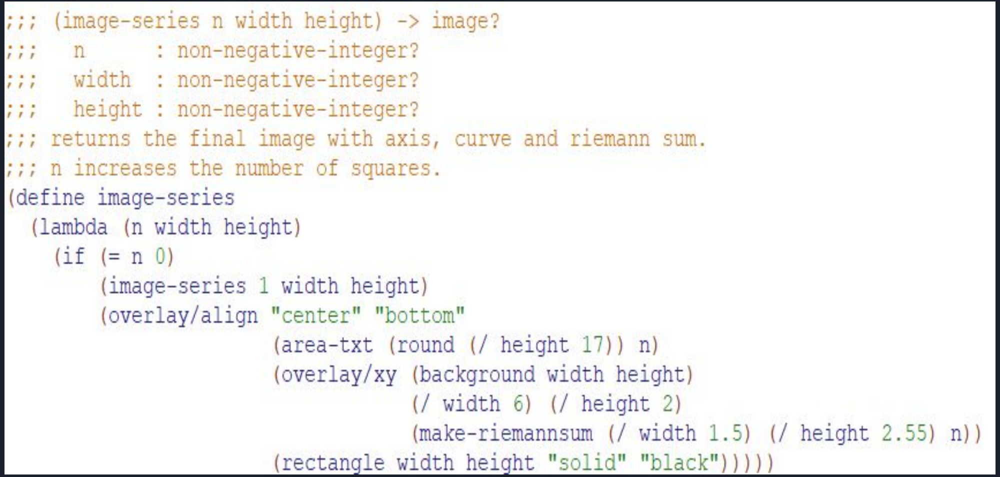
Decomposing the solution

Image-series output



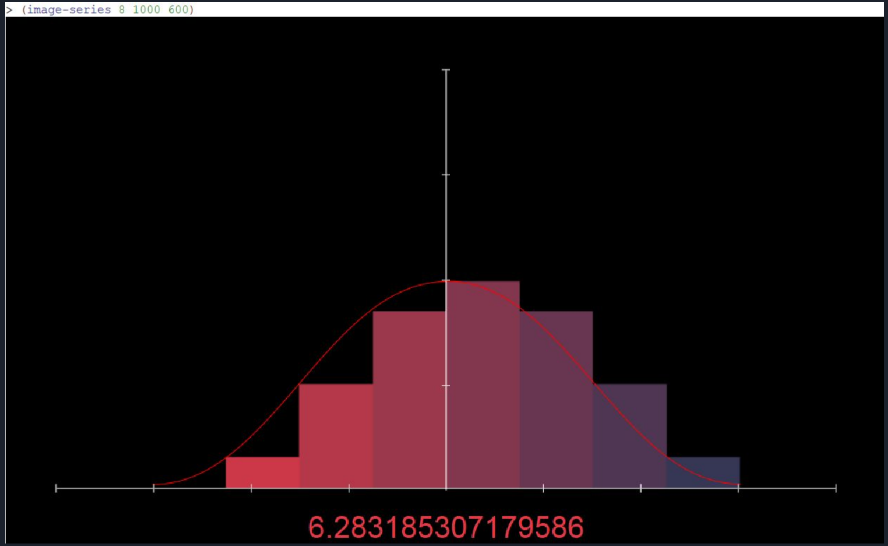
```
> (image-series 800 1000 600)
```

Image-series code

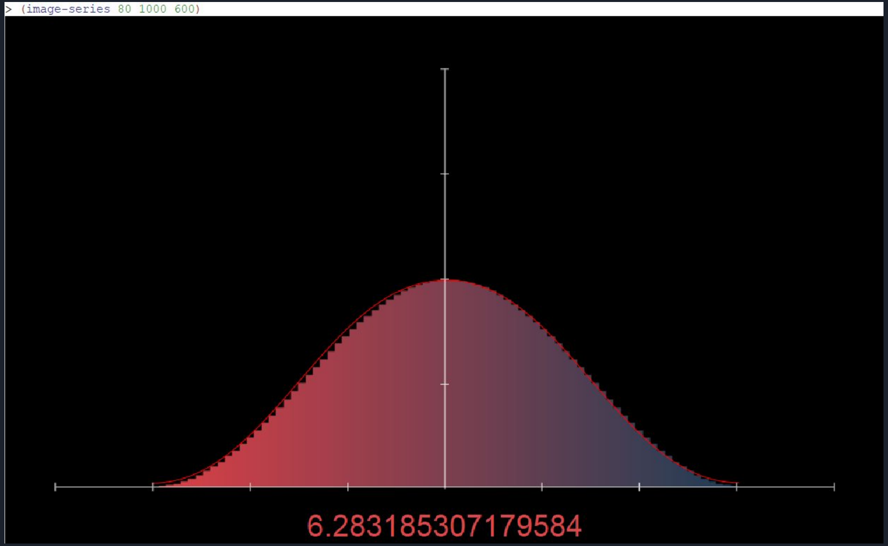


```
;;; (image-series n width height) -> image?
;;;   n      : non-negative-integer?
;;;   width  : non-negative-integer?
;;;   height : non-negative-integer?
;;; returns the final image with axis, curve and riemann sum.
;;; n increases the number of squares.
(define image-series
  (lambda (n width height)
    (if (= n 0)
        (image-series 1 width height)
        (overlay/align "center" "bottom"
                       (area-txt (round (/ height 17)) n)
                       (overlay/xy (background width height)
                                   (/ width 6) (/ height 2)
                                   (make-riemannsum (/ width 1.5) (/ height 2.55) n))
                       (rectangle width height "solid" "black")))))
```

EXAMPLE IMAGES



```
> (image-series 8 1000 600)
```

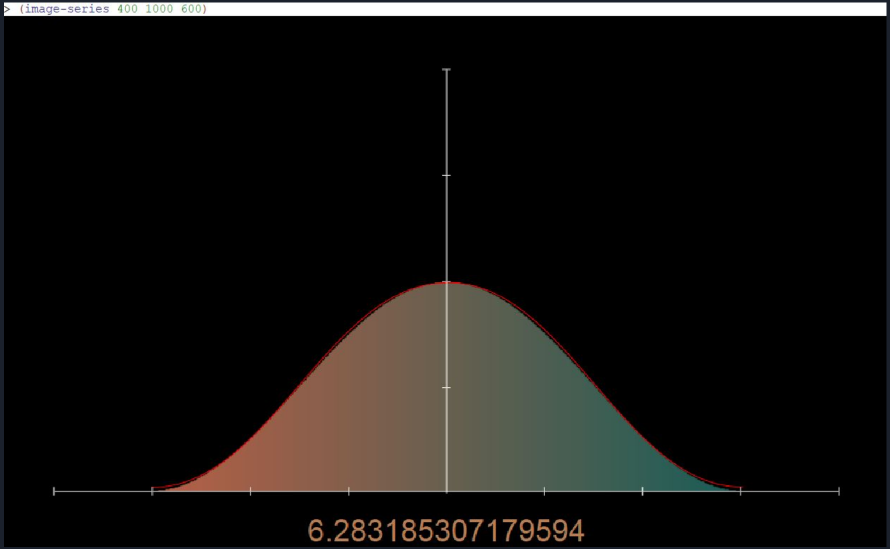


```
> (image-series 80 1000 600)
```

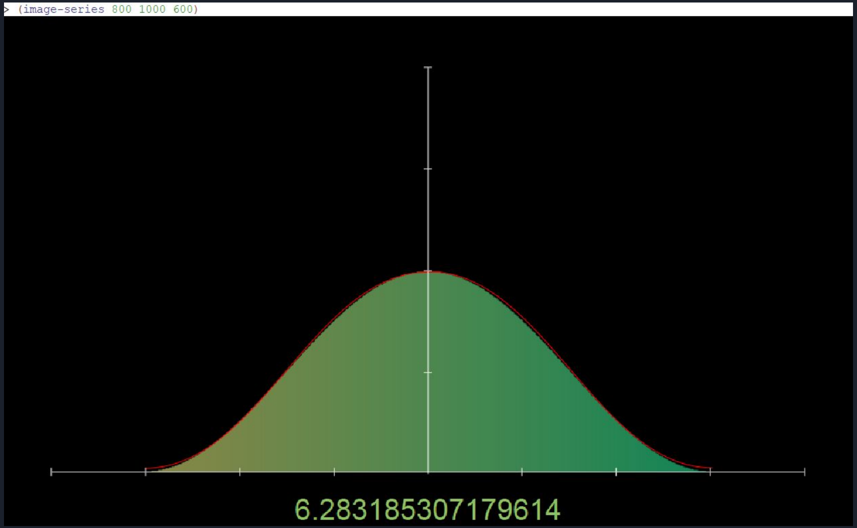
1. n=8 , Low Area Accuracy

2. n=80 , Low-medium Area Accuracy

EXAMPLE IMAGES



```
> (image-series 400 1000 600)
```



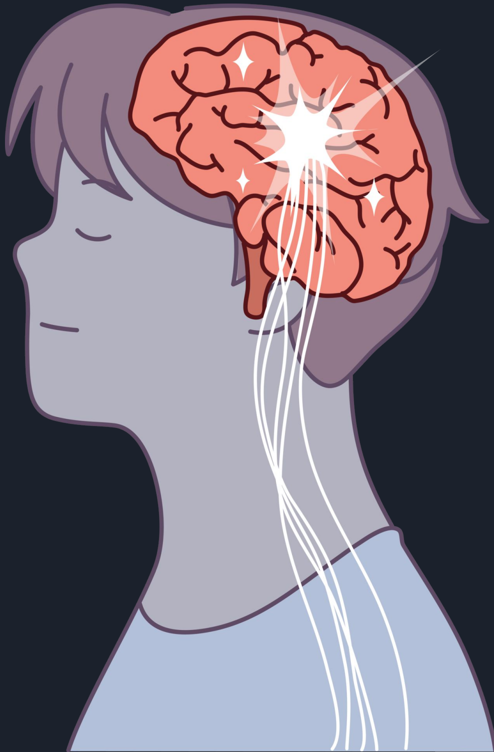
```
> (image-series 800 1000 600)
```

3. n=400 , Medium-high Area Accuracy

4. n=800 , High Area Accuracy

WHAT DID WE LEARN?

- Group work in the GitHub environment is efficient.
- Add-curve function
- Data abstraction is used a lot in collaboratively work.
- It is important to write clean code and always document it.



THANK YOU FOR LISTENING!!!