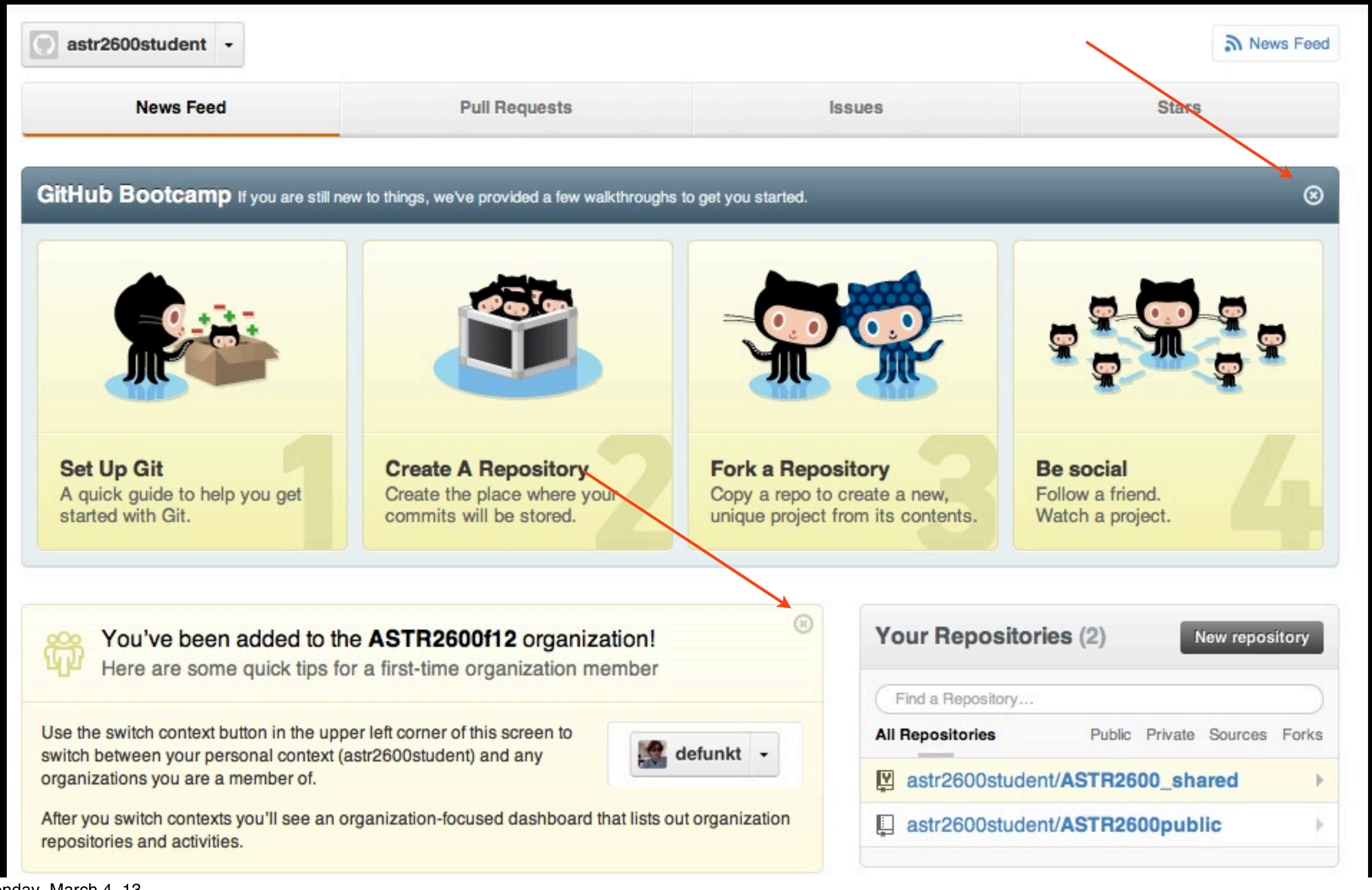
Does your homepage look like this?



Programs: Keywords

Keyword Variables

- Optional “named” arguments
- Optional means the user doesn’t have to set them, so your code needs to know how to deal with empty keywords
- Often used as boolean flags

Boolean Keyword

```
function square,x,verbose=verbose
    retvar = x^2
    if keyword_set(verbose) then print,retvar
    return,retvar
end
```

- print `retvar` only if `verbose` is set to non-zero
- keyword_set checks whether the keyword has been set
  - it recognizes *any non-zero* as True

Keyword

```
; take x to some power
; the power defaults to 1 if not set
function power,x,pow=pow
    if n_elements(pow) eq 0 then pow = 1
    return,x^pow
end
```

- pow is some number
- can’t use `keyword_set` because `0` is a valid power
- `if n_elements(keyword) eq 0 ...` sets a *default value* for the keyword

Keywords in Python

```
def power(x,pow=2):
    return x**pow

def square(x,verbose=False):
    retval = x**2
    if verbose:
        print retval
    return retval
```

- Declare defaults along with the keyword name

Keyword Oddities

```
function square,x,verbose=verbose
    retvar = x^2
    if keyword_set(verbose) then print,retvar
    return,retvar
end
```

```
function cube,x,verbose=verb
    retvar = x^3
    if keyword_set(verb) then print,retvar
    return,retvar
end
```

- The variable name is on the *right side* of the keyword declaration

Keyword Oddities

- The variable name is on the *right side* of the keyword declaration
- but the keyword you call with is on the *left side*

```
IDL> x=cube(2,/verbose)
       8
IDL> x=square(2,/verbose)
       4
```

```
IDL> x=cube(2,verbose=1)
       8
IDL> x=square(2,verbose=1)
       4
```

Debugging

Bugs and Errors

- Errors are things that IDL knows are wrong
  - syntax errors - caught at compile time
  - “Variable is undefined” errors - caught at run time

Bugs and Errors

- “Bugs” are things that cause undesired behavior, but aren’t wrong code
  - i.e., if the function `square(2)` returned 5, but the correct answer is 4, that is a bug
  - it “works” - it gives you output and doesn’t crash - but it works wrong.

Errors

- Syntax errors are caught when you "compile" (`.compile` or `.run`) your code
- IDL will try to compile the rest of the code even after a syntax error!
  - This can result in a LONG list of errors!
  - *ALWAYS address the first error first!*

```
if 1 then print,5

function test1
    x = 4
    print,x
end

function test2
    print,x
end

function test3,x
    x = 5
    return,x
end
```

Here's a `.pro` file. I'll try to compile it and show you the errors.

All 3 of these functions are syntactically correct.

```
IDL> .r test

function test1
              ^
% Procedure header must appear first and only once: TEST1
  At: /Users/adam/Dropbox/astr2600/lectures/test.pro, Line 4
% 1 Compilation error(s) in module $MAIN$.

function test2
              ^
% Procedure header must appear first and only once: TEST2
  At: /Users/adam/Dropbox/astr2600/lectures/test.pro, Line 9
% 1 Compilation error(s) in module $MAIN$.

function test3,x
                ^
% Procedure header must appear first and only once: TEST3
  At: /Users/adam/Dropbox/astr2600/lectures/test.pro, Line 13

    return,x
             ^
% Return statement in procedures can't have values.
  At: /Users/adam/Dropbox/astr2600/lectures/test.pro, Line 15
% 2 Compilation error(s) in module $MAIN$.
```

```
if 1 then print,5

function test1
    x = 4
    print,x
end

function test2
    print,x
end

function test3,x
    x = 5
    return,x
end
```

This is the only real error (you're not allowed to have anything but functions and procedures in a function/ procedure file)

...unless the "something else" is at the end, followed by `end`

```
IDL> .r test

function test1
              ^
% Procedure header must appear first and only once: TEST1
  At: /Users/adam/Dropbox/astr2600/lectures/test.pro, Line 4
% 1 Compilation error(s) in module $MAIN$.
```

- The "compiler" tells you the closest location of an error, but does not make clear how to fix the error

- This is one of the challenges of programming - figuring out what obscure error messages mean

  - also a great reason to make YOUR error messages explicit!

```
print,"message"

function test1
    print,message
end
```

What error message will you get if you compile this file?

A) `% You compiled a main program while inside a procedure.  Returning.`

B) `% Procedure header must appear first and only once: TEST1`

C) `% End of file encountered before end of program.`

D) `% Syntax error.`

E) None of the above

```
function test1
    print,"message"
end

print,"message"
print,test1()

end
```

What error message will you get if you compile this file?

A) `% You compiled a main program while inside a procedure.  Returning.`

B) `% Procedure header must appear first and only once: TEST1`

C) `% End of file encountered before end of program.`

D) `% Syntax error.`

E) None of the above

```
print "Hi"

def f(x):
    return x*5

print f("Hi")
```

Will this code work? (hint: `f("Hi")` does)

A) Yes

B) No, can't have `prints` before and after the function definition

C) No, I don't believe your hint, you can't multiply a string by a number

D) Just no.

E) None of the above

```
print "Hi"

def f(x):
    return x*5

print f("Hi")
```

Will this code work? (hint: `f(“Hi”)` does)

A) Yes

```
In [111]: %run hi5.py
Hi
HiHiHiHiHi
```

Runtime Errors

```
function test2
    print,x
end
```

```
IDL> print,test2()
% PRINT: Variable is undefined: X.
% Execution halted at: TEST2                    10 /Users/adam/Dropbox/astr260
%                      $MAIN$
```

- `test2` is syntactically correct, but `x` is undefined
- Error message tells you:
  - `x` is undefined when you try to print it
  - the error is on line 10 of [filename] in function `test2`

```
In [118]: ??f
Type:        function
String Form:<function f at 0x10b030f50>
File:        /Users/adam/Dropbox/astr2600s13/lectures/lecture13notes.py
Definition: f(x)
Source:
def f(x):
    return y
In [119]: f(1)
Traceback (most recent call last):
  File "<ipython-input-119-90b61b657670>", line 1, in <module>
    f(1)
  File "/Users/adam/Dropbox/astr2600s13/lectures/lecture13notes.py", line 2, in f
    return y
NameError: global name 'y' is not defined
```

?? shows you source code

Can’t return an undefined variable:
‘y’ is not defined!

Problem is on line 2
of the function “f”

If your code doesn't work...

- You should ask first, "What error messages is it giving me?"
- "Does that tell me why it doesn't work?"
- Ask yourself these questions before you ask me or Cameron.

Runtime Errors

- In IDL, act just like `stop` statements
- They halt the code wherever the error was, allowing you to inspect *local* variables
- To get out of the halted code (the *debugger*), use `retall` (“return all”)

Runtime Errors

- Python doesn't kick you out into the debugger automatically, you need to enable it first:

```
In [120]: %pdb
Automatic pdb calling has been turned ON
In [121]: f(1)
Traceback (most recent call last):
  File "<ipython-input-121-90b61b657670>", line 1, in <module>
    f(1)
  File "/Users/adam/Dropbox/astr2600s13/lectures/lecture13notes.py", line 2, in f
    return y
NameError: global name 'y' is not defined

> /Users/adam/Dropbox/astr2600s13/lectures/lecture13notes.py(2)f()
      1 def f(x):
----> 2     return y
      3

ipdb>
```

Reading

- Read section 13.6. It is about debugging. Debugging is important, but there are too many words for me to repeat in class.

Debugging

1. Determine that there is a bug
2. Identify the bug
3. Treat the bug like a “whuduzitdo” - understand what the code is *actually* doing
4. Correct the bug

Debugging

1. Determine that there is a bug
2. Identify the bug
    A. Create a test that “fails” on the bug
3. Treat the bug like a “whuduzitdo” - understand what the code is *actually* doing
4. Correct the bug
    A. Check that the test now does NOT fail

```
def square(x):
    x = x**2
```

Will this function run?

A) Yes

B) No

```
def square(x):
    x = x**2
```

Will it set `x` to be its square?

A) Yes

B) No

```
def square(x):
    x = x**2
```

Will it return x^2?

A) Yes

B) No

```python
def square(x):
    x = x**2

def test_square():
    assert square(4) == 16
```

```
In [125]: %run square.py
In [126]: test_square()
Traceback (most recent call last):
  File "<ipython-input-126-108216126958>", line 1, in <module>
    test_square()
  File "/Users/adam/Dropbox/astr2600s13/lectures/square.py", line 5, in test_square
    assert square(4) == 16
AssertionError
```

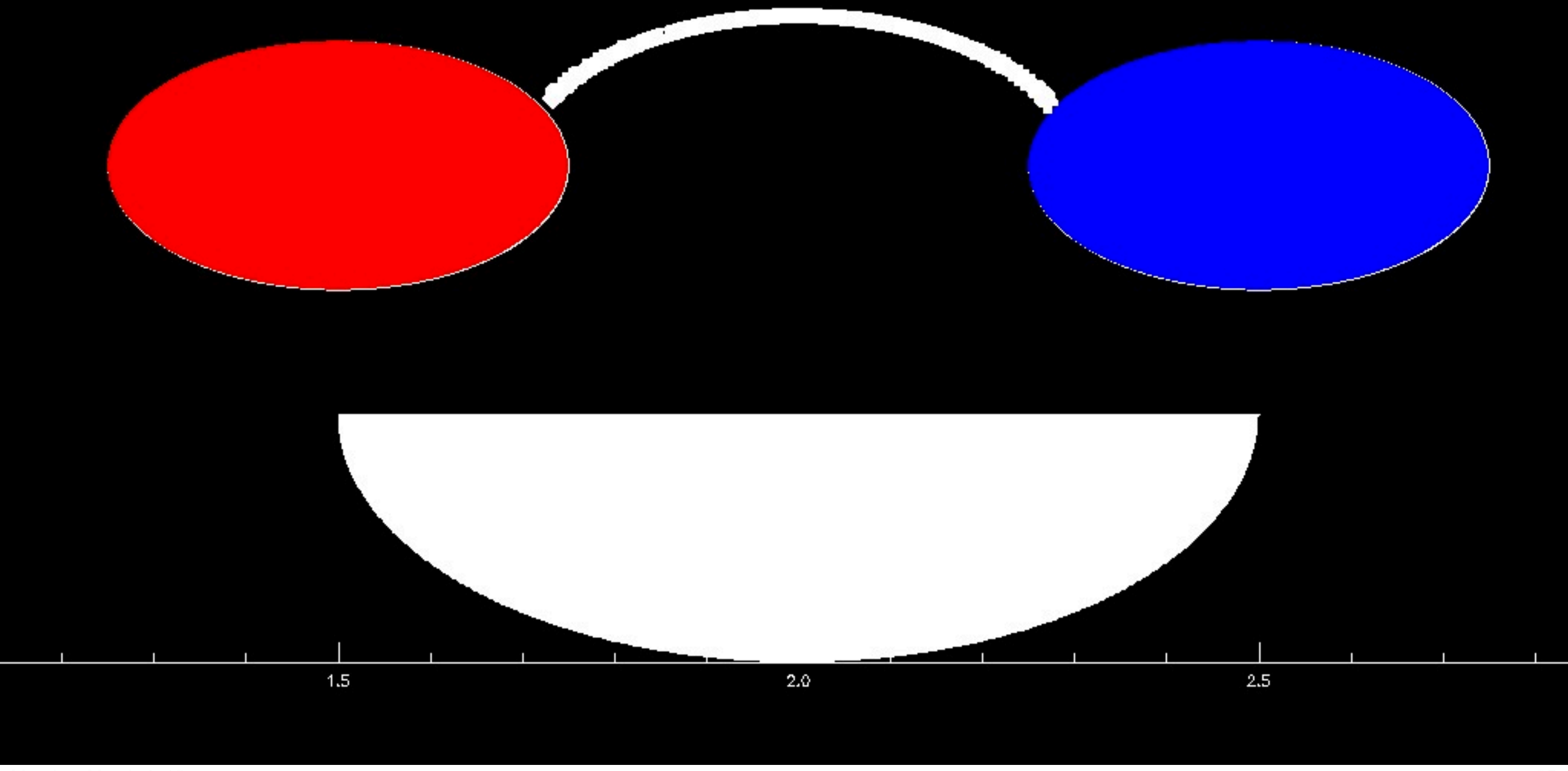
```python
def square(x):
    return x**2

def test_square():
    assert square(4) == 16
```

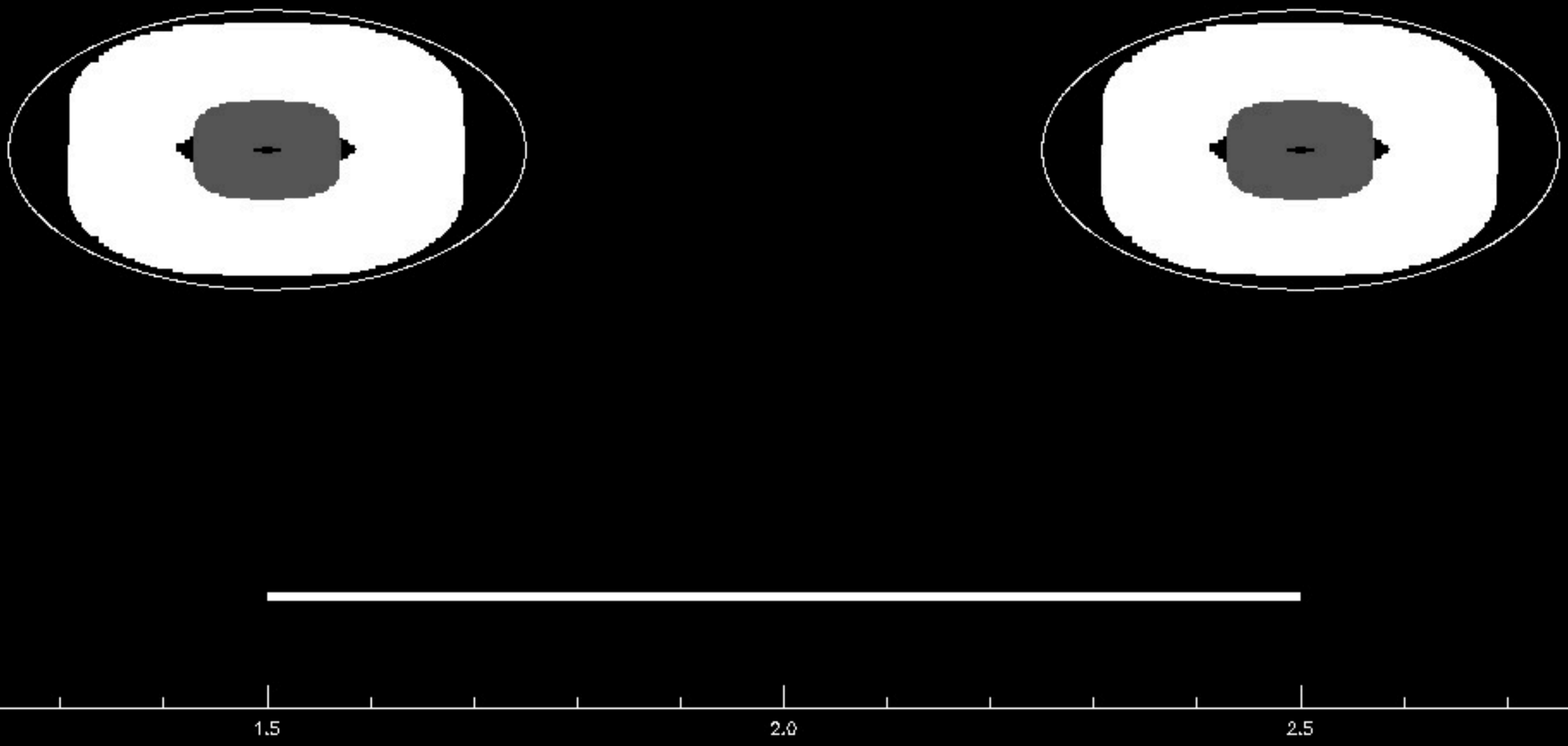
```
In [127]: %run square.py
In [128]: test_square()
```

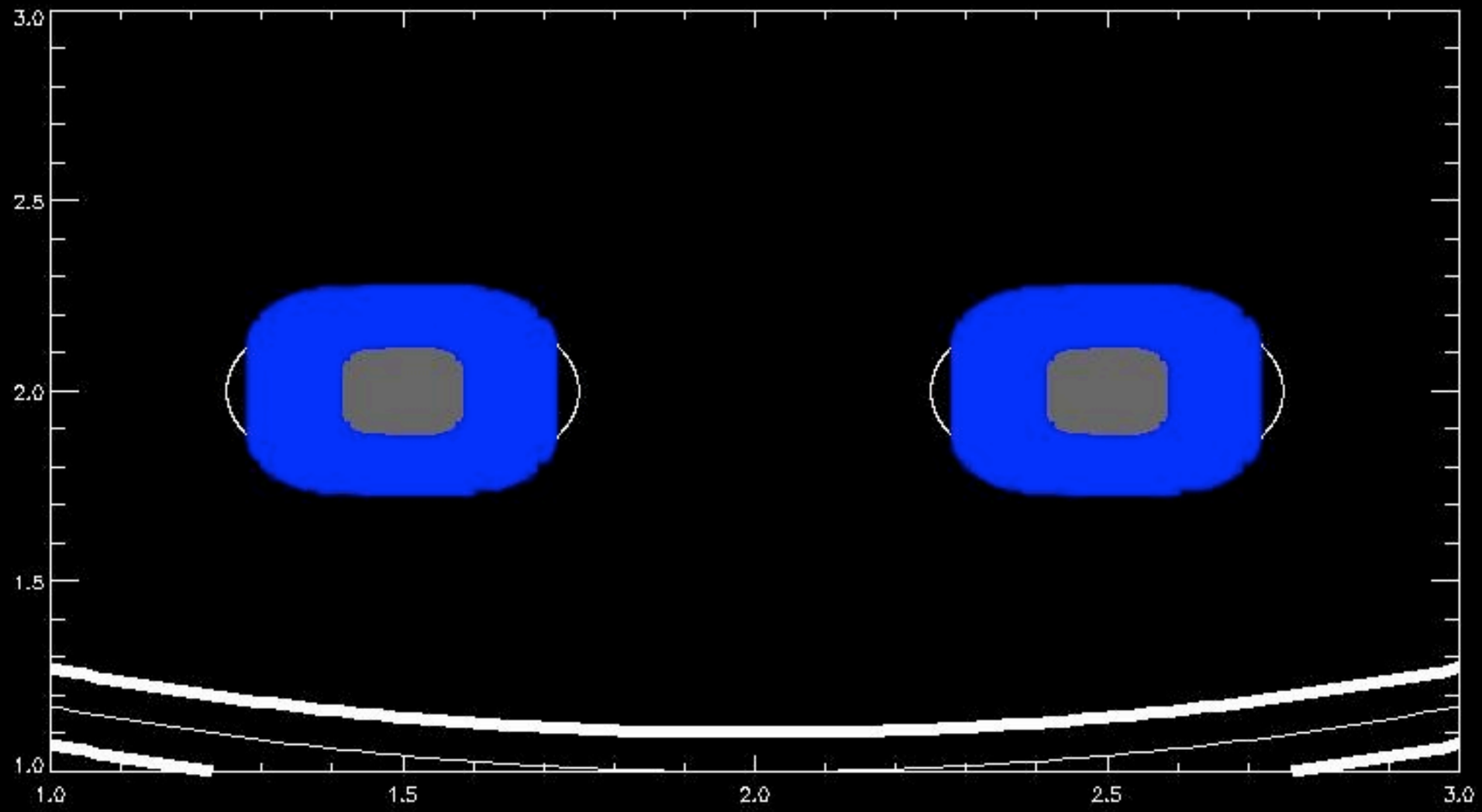
Debugging

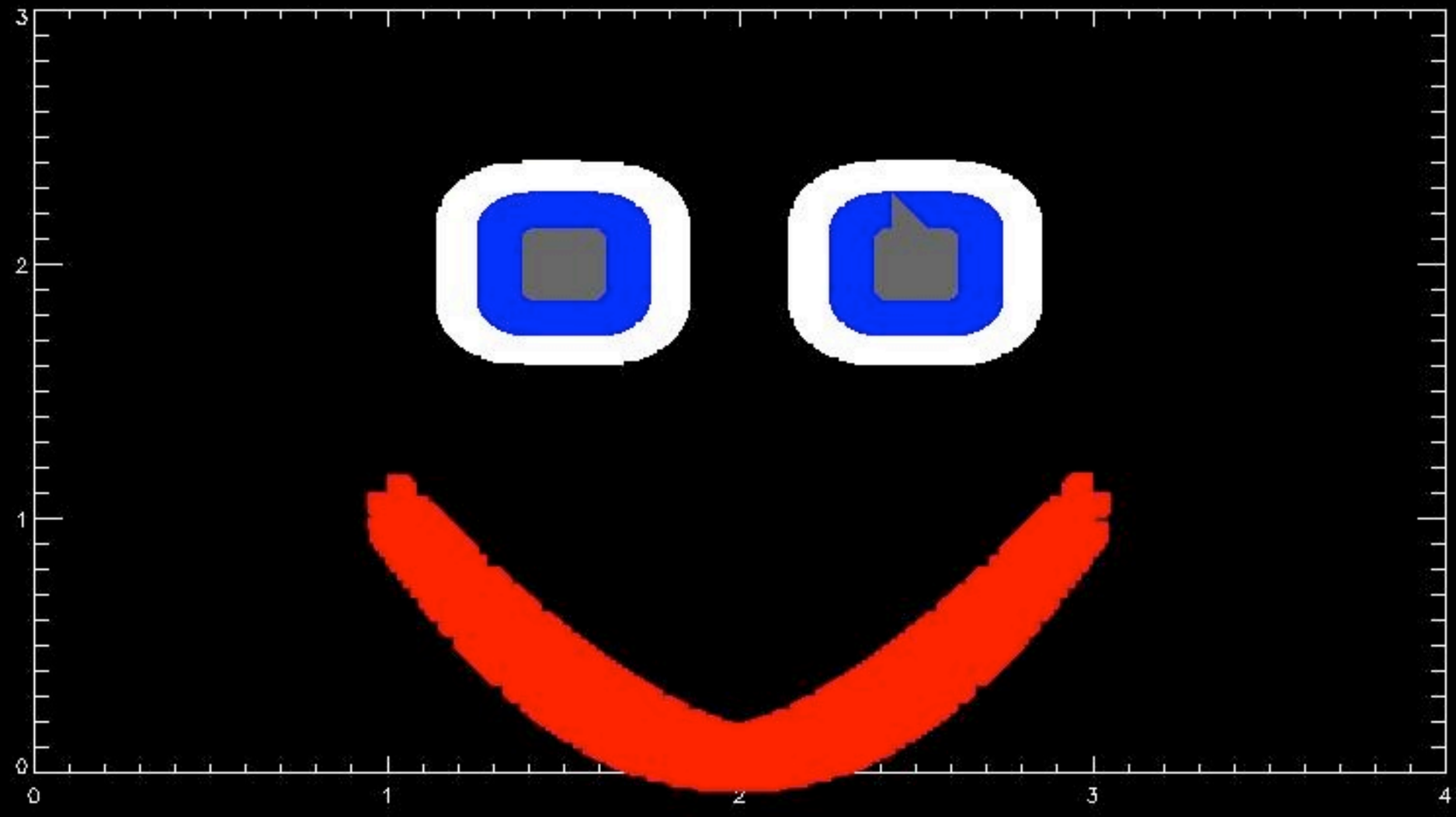
- Debugging is a skill developed over time
- whuduzitdo's are designed to give you that experience
- debugging = you are the compiler

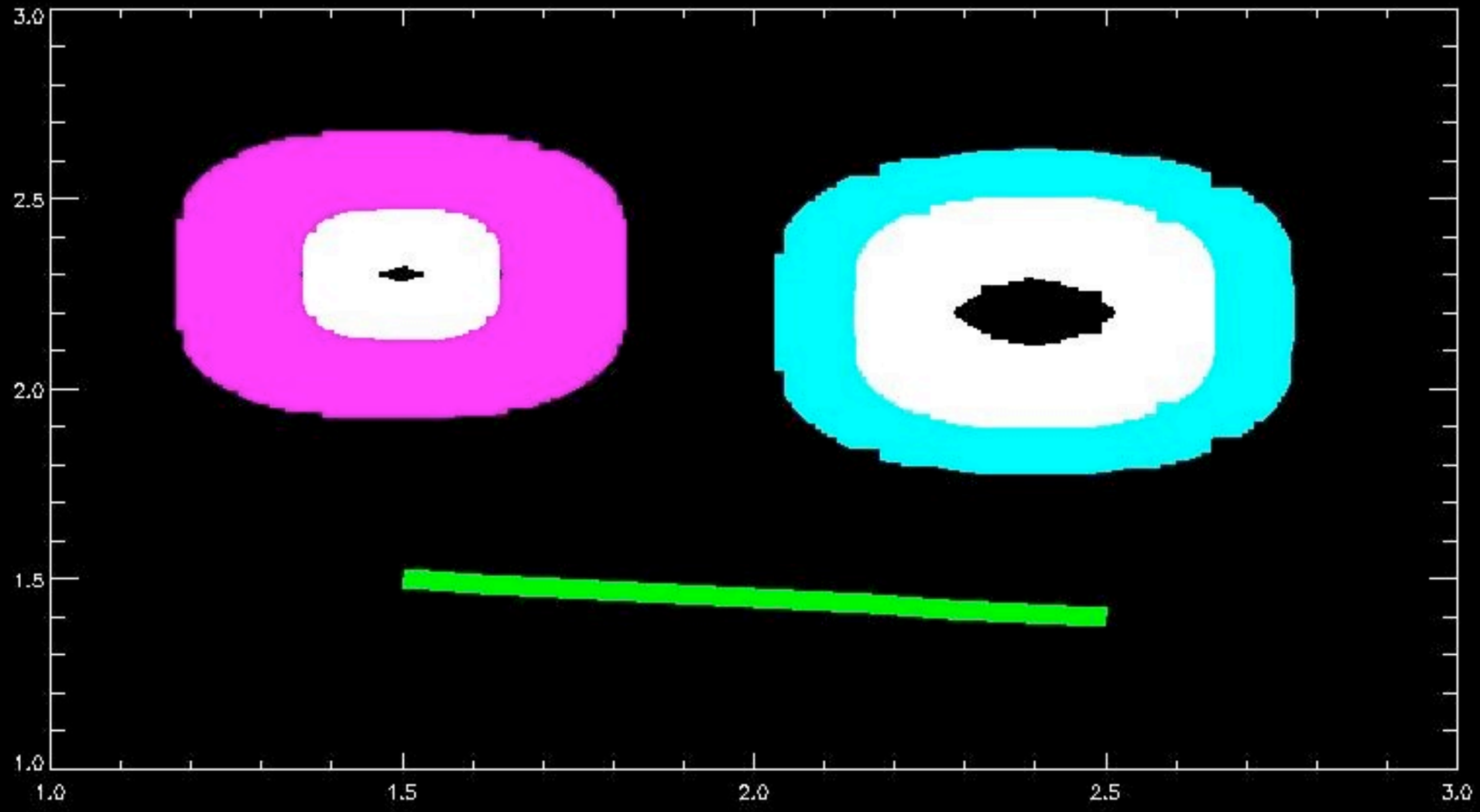
Now for a gallery of eyeballs...

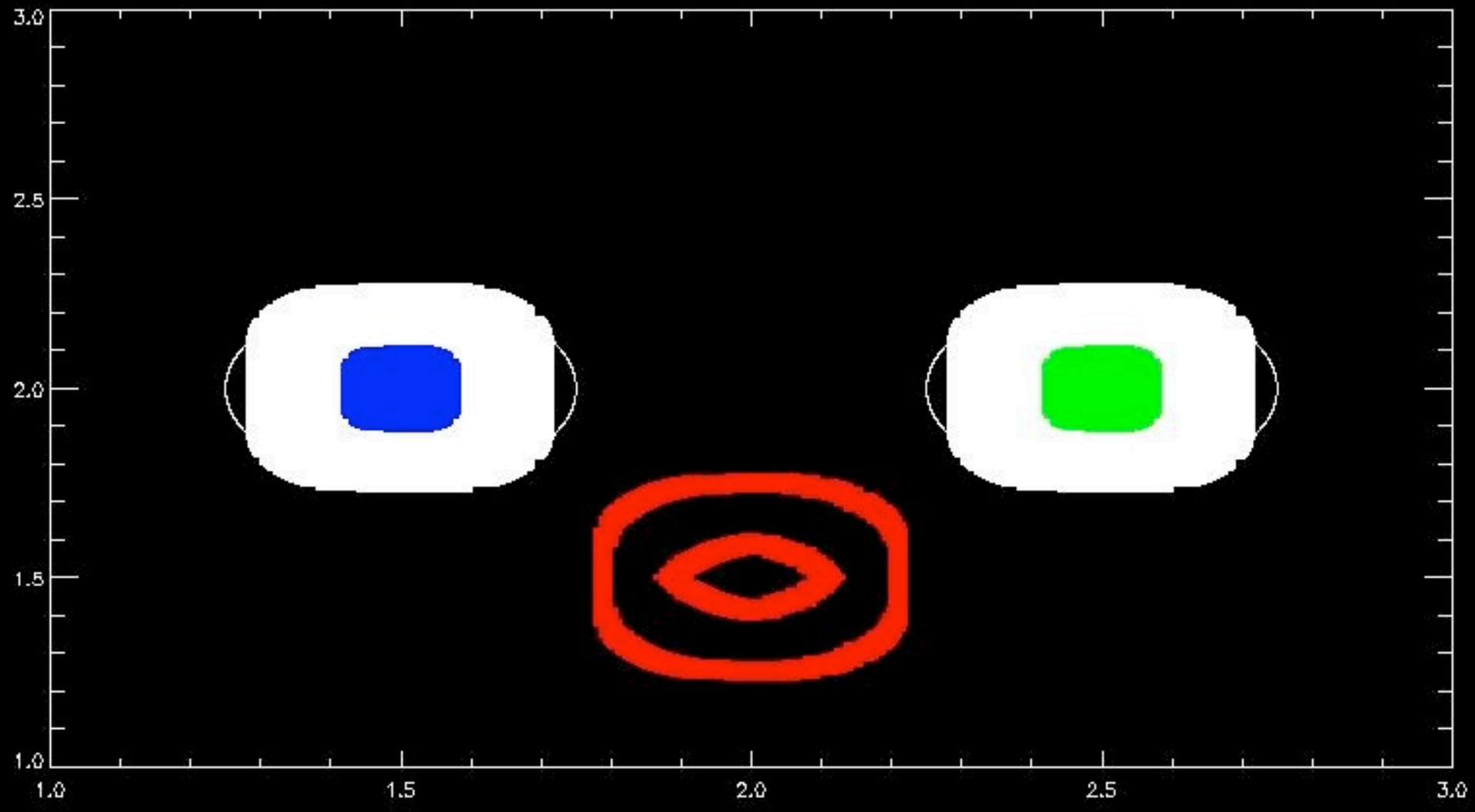


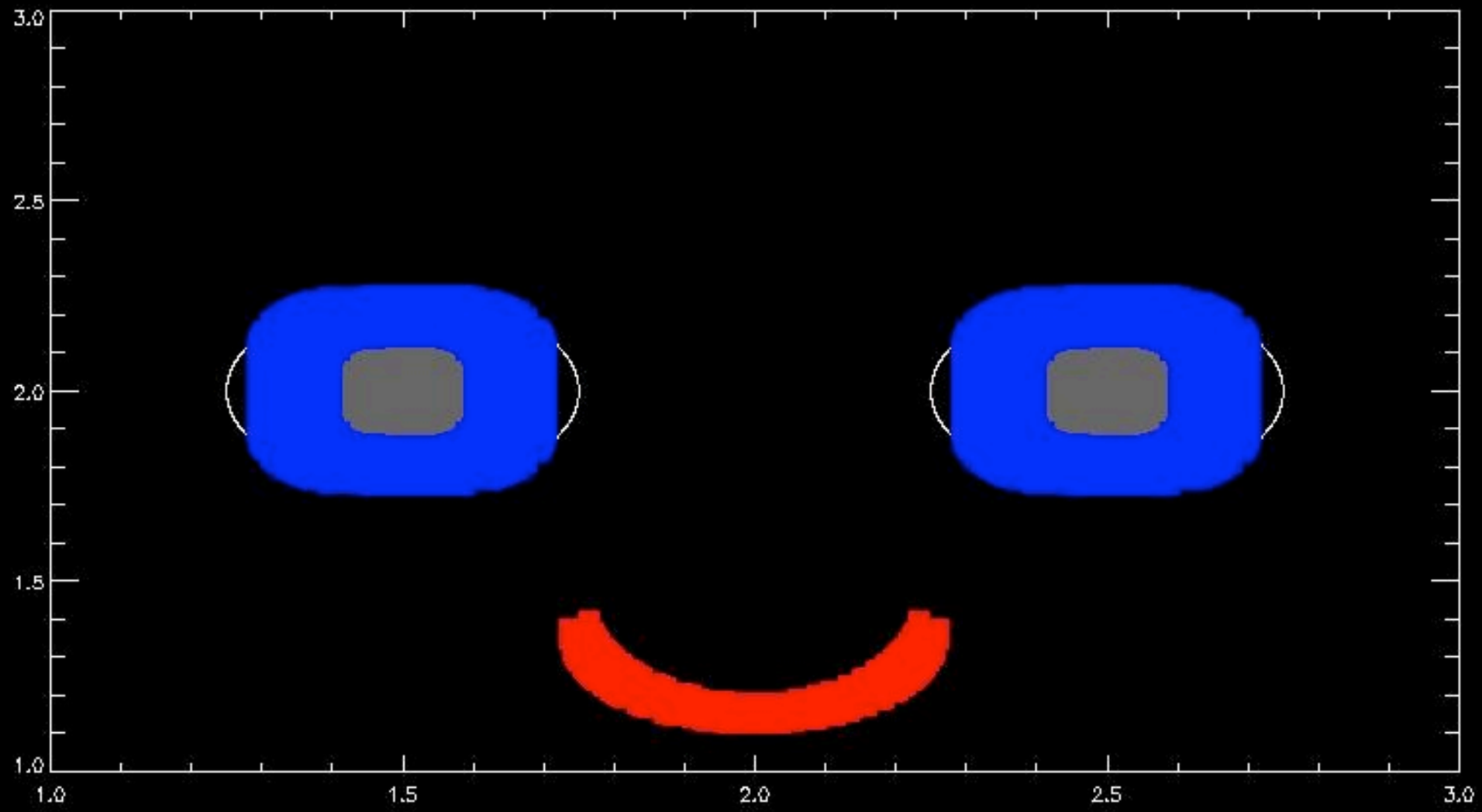

1.5
2.0
2.5

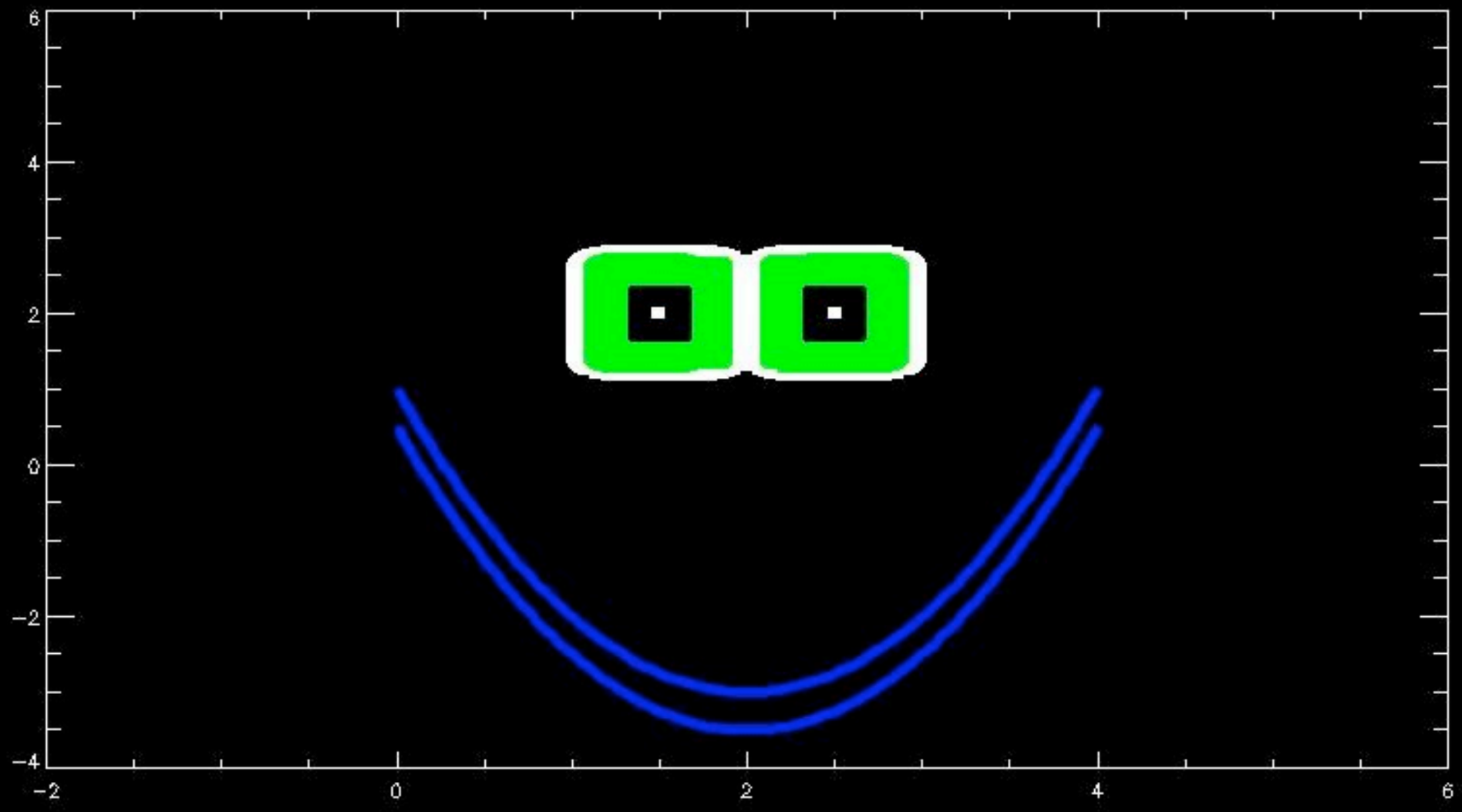

3.0
2.5
2.0
1.5
1.0
1.0
1.5
2.0
2.5
3.0

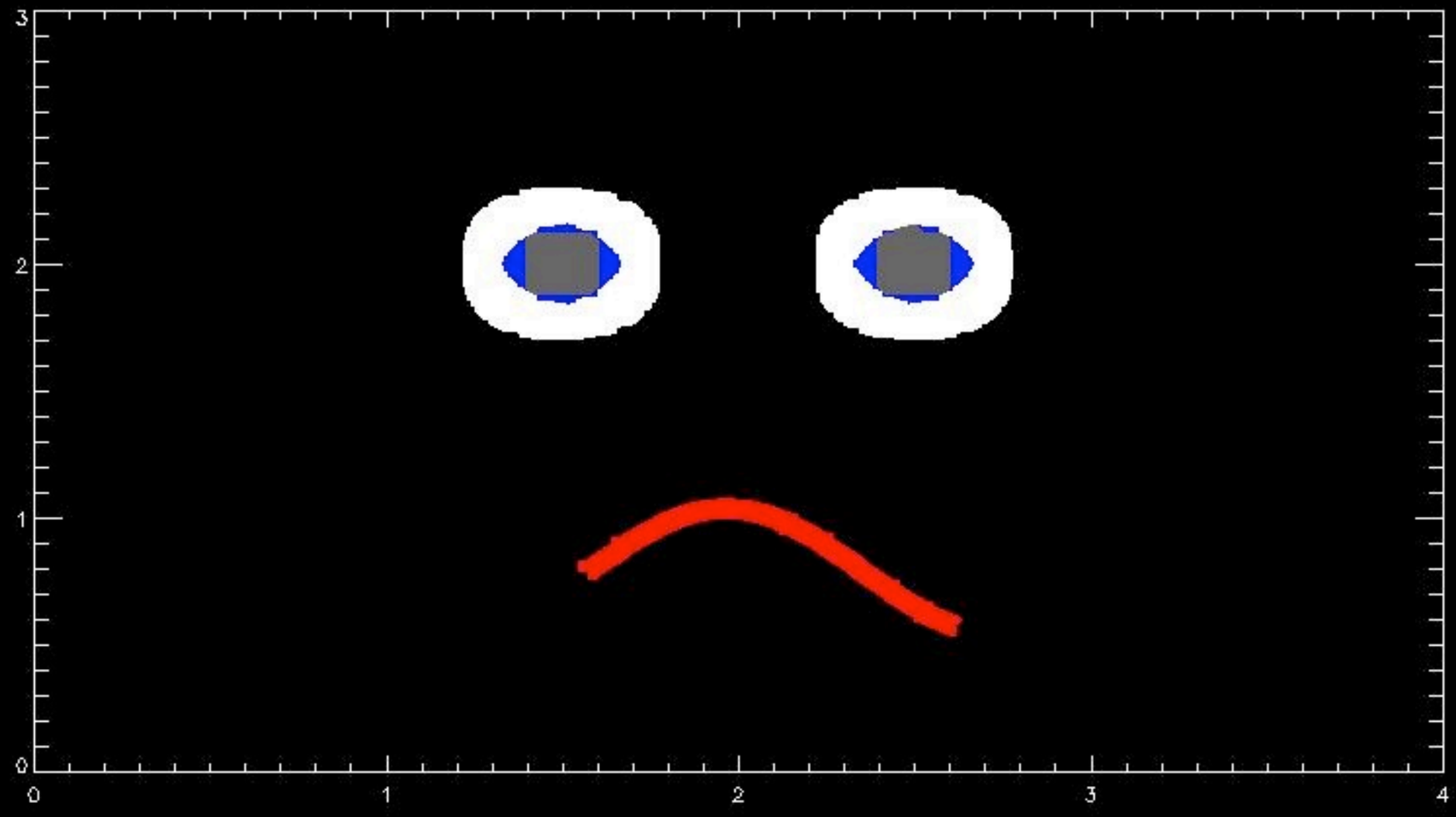

3
2
1
0
0
1
2
3
4

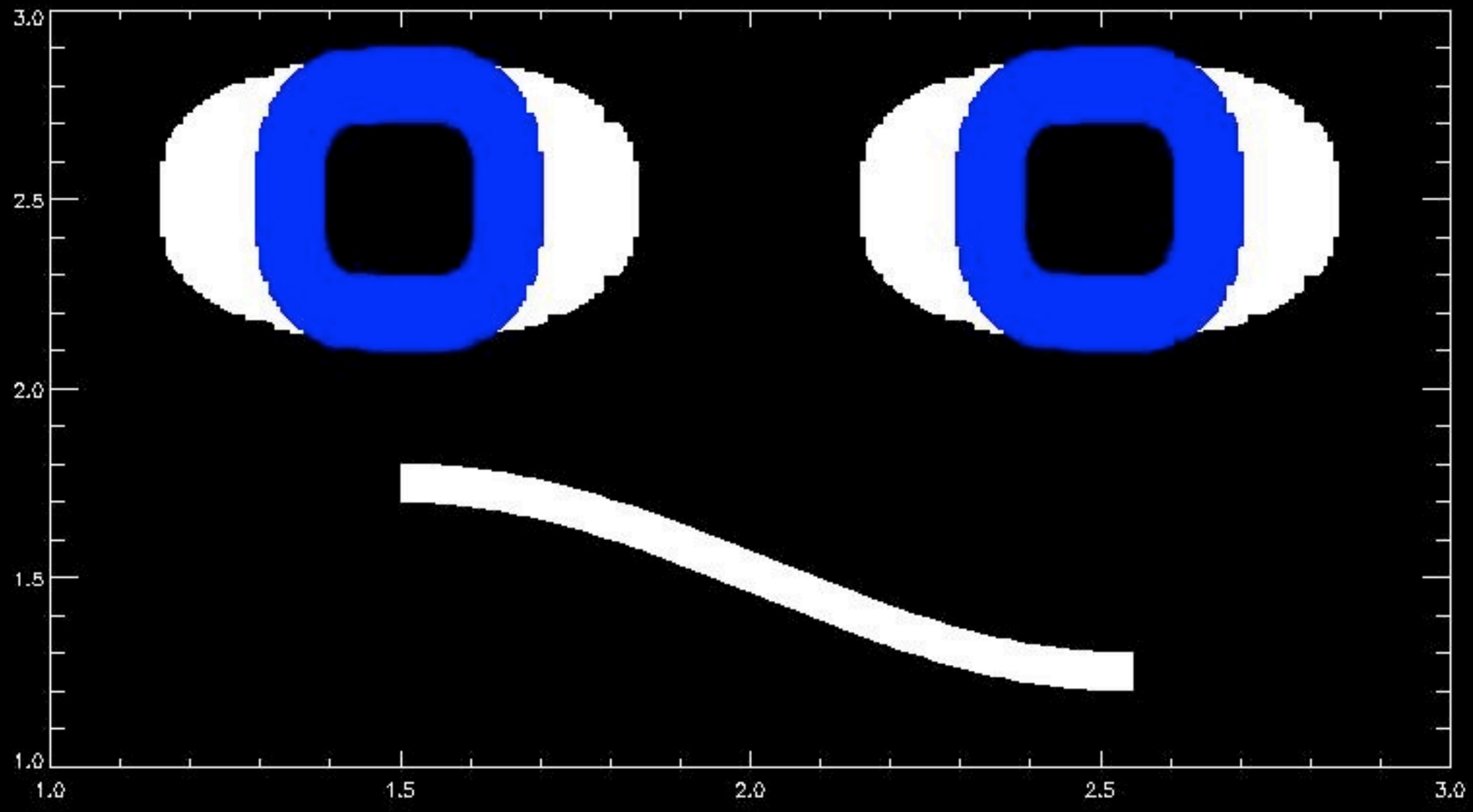

3.0
2.5
2.0
1.5
1.0
1.0
1.5
2.0
2.5
3.0

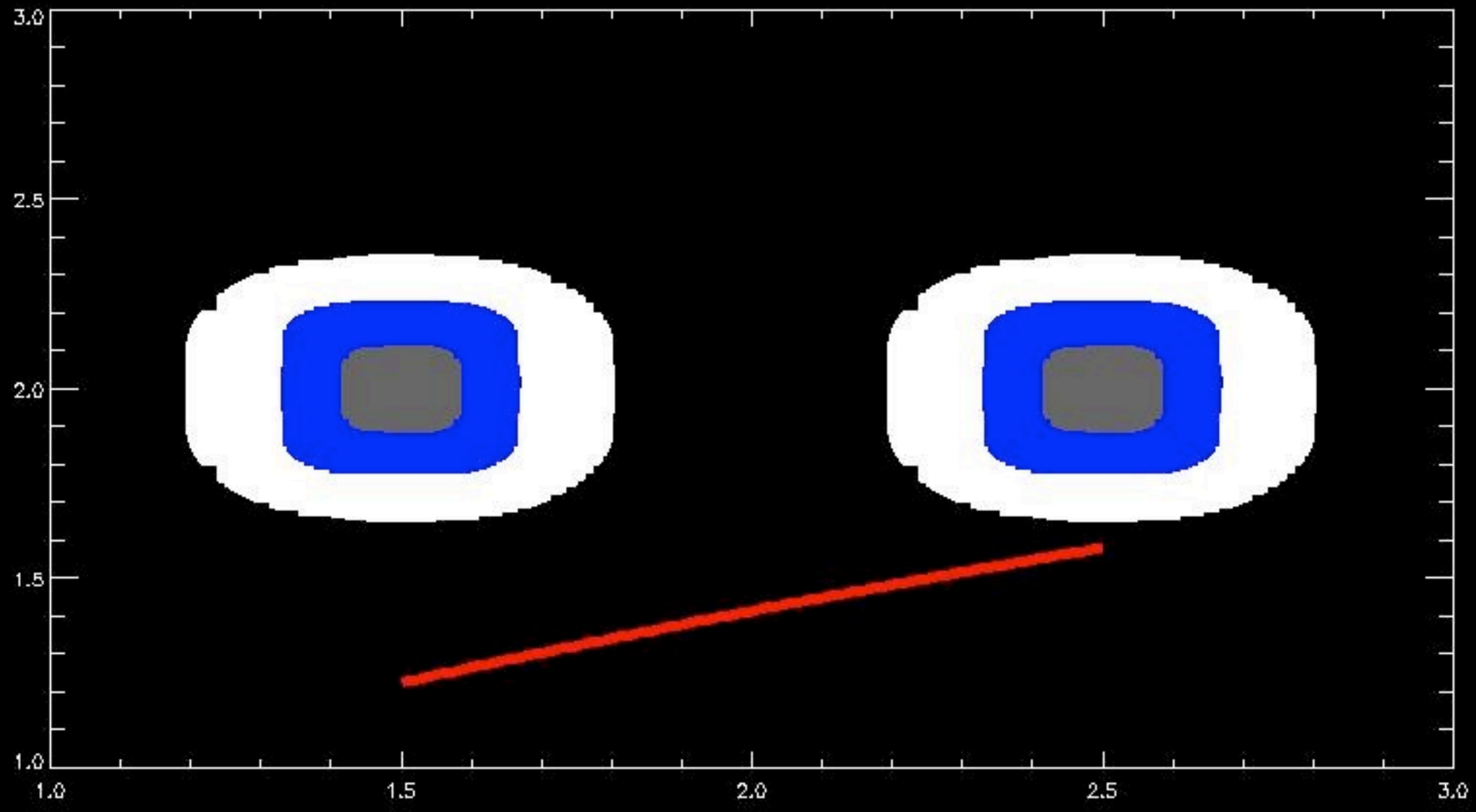

3.0
2.5
2.0
1.5
1.0
1.0
1.5
2.0
2.5
3.0


3.0
2.5
2.0
1.5
1.0
1.0
1.5
2.0
2.5
3.0


6
4
2
0
−2
−4
−2
0
2
4
6


3
2
1
0
0
1
2
3
4


3.0
2.5
2.0
1.5
1.0
1.0
1.5
2.0
2.5
3.0


3.0
2.5
2.0
1.5
1.0
1.0
1.5
2.0
2.5
3.0

Chapter 14: Program Design and Development

General Rules

- Start small: Make testable, individual pieces of code
- debugging is much slower than code writing

Top Down Design

- Look at the big picture first (start with a vague outline, with the big bullets written first)

Example

1. Get input data
2. Do computation
3. Output results

Example

1. Get input data `get_data,x,y`
2. Do computation `z = compute_z(x,y)`
3. Output results `output_results,z`

Example

- Fill in individual functions with test stubs:

```
; get_data
;  get x and y values
;  **TEST STUB**
;   TO DO: Fix to get user data from a specified file
; OUTPUT:
;   X & Y arrays
pro get_data,x,y
    x=findgen(5)
    y=randomu(seed,5)
end ; get_data
```

- Doesn't actually *read* anything, but outputs data in the right format
  - i.e., does enough for the next step

Example

- `get_data` does just enough for `compute_z` to work

Example

- Overly simplified output

```
; output_results
;   output the results of the computations
; **TEST STUB**
;   TO DO: Fix to create plot & write jpg
; INPUT:
;   z - the result
pro output_results,z
    print,z
end ; output_results
```

- Just enough to see if the prior steps worked (not “pretty” yet)

Example - Outline revisited

1. Get input data `get_data,x,y`
   - Reads reasonable test data
2. Do computation `z = compute_z(x,y)`
   - still not implemented
3. Output results `output_results,z`
   - May be ugly, but does enough to see if compute_z works

Example

- Create `compute_z` now
  - you already have a “testing framework” built around it
- Then, iterate: maybe you need more complicated data for `compute_z`? change `get_data`

Another approach: Top-Down

- Advantages:
  - Big Picture is set up early
  - Can break down into small chunks
    - Good for team development
  - Visualize as “flow charts”

Top-Down

- Frequently utilizes “pseudo-code”
  - pseudo-code is “almost code” that reads more like English
    - it’s code, but you don’t care about syntax - you aim for readability
    - pseudocode is usually easy to translate into real code, but easier to read (as a human) than code

Pseudo-Code example

- Open my flux and wavelength files
- Plot flux versus wavelength
- Highlight the spectral line at 1020A
- Fit a gaussian to that line

Bottom-Up approach

- Start with small details
- Test each small step along the way
  - (for small projects, this will end up looking the same as “Top Down”, except you won’t write an outline at the start)

Did you find info about a programming or research job?

A) Yes, and I will need to use IDL

B) Yes, and I will need to use python

C) Yes, and I will need to use something else

D) No, but I have an idea where to look

E) No, and I have no idea where to look