Debugging Programs

Phil Mansfield

Debugging is as Important as Coding

Programmers spend as much time debugging incorrect code as they do programming.

Source: Evans Data Corporation (2012), Payscale (2012), RTI (2002), CVP Surveys (2012)

When you finish your code, you’re only half done: spending a lot of time on debugging is totally fine (and try to budget your time accordingly!).

Developing your debugging skills is as important as developing your programming skills.

Debugging is like a mystery where you are the criminal, the detective, the jury (and the victim).

(paraphrased from Filipe Fortes)

Being good at debugging means that you're a criminal who is very easy to catch, a detective who is persistent and efficient, and a jury who knows what to look at.

Debugging Stages

1. Writing your code
2. Finding out that there is a bug
3. Finding the bug.
4. Fixing the bug.

Debugging Stages

1. Writing your code
2. Finding out that there is a bug
3. Finding the bug.
4. Fixing the bug.

Being a bad criminal

Being an good detective

Being a smart jury

Being the Worst Criminal Humanly Possible: How to Write Code That's Easy to Debug

1. Always reread your code before running it.
2. Break your code into the smallest, most modular functions that you can.
3. Run/test your code after every ~20-30 lines of code, rather than once after you've written everything.
   a. Do this function by function
4. Write your code so that it isn't confusing.
5. Use assertions.

```python
def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) creates a histogram from the 1D array
    `data with `bins` bins that ranges from [`low`, `high`).

    Returns a tuple (`hist`, `edges`), where `hist` contains the number of
    elements in each histogram bin and `edges` contains every bin edge.
    """
    assert(bins > 0)
    assert(high > low)

    bin_width = (high - low) / bins
    scaled_data = (data - low) / bin_width
    bin_index = np.floor(scaled_data).astype(int)

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx < 0 or idx >= bins: continue
        hist[idx] += 1

    edges = np.linspace(low, high, bins+1)

    return hist, edges
```

Being the Worst Criminal Humanly Possible: How to Write Code That's Easy to Debug

1. Always reread your code before running it.
2. Break your code into the smallest, most modular functions that you can.
3. Run/test your code after every ~20-30 lines of code, rather than once after you've written everything.
   a. Do this function by function
4. **Write your code so that it isn't confusing.**
5. Use assertions.

```
def H(x,b,l,h):
    i = np.floor((x-l)*b/(h-l)).astype(int)
    H = np.zeros(b,dtype=int)
    for j in i:
        if b>j>=0: H[j]+=1
    return H,np.linspace(l,h,b+1)
```

```python
def H(x,b,l,h):
    i = np.floor((x-l)*b/(h-l)).astype(int)
    H = np.zeros(b,dtype=int)
    for j in i:
        if b>j>=0: H[j]+=1
    return H,np.linspace(l,h,b+1)
```

1. Just because you ***can*** fit something on one line doesn’t mean you ***should***.
    a. Don’t make any one line too complex: if you need to use “and” to describe what a line does, it should usually be broken up.

```python
def H(x,b,l,h):
    i = np.floor((x-l)*b/(h-l)).astype(int)
    H = np.zeros(b,dtype=int)
    for j in i:
        if b>j>=0: H[j]+=1
    return H,np.linspace(l,h,b+1)
```

1. Just because you ***can*** fit something on one line doesn't mean you ***should***.
   a. Don't make any one line too complex: if you need to use "and" to describe what a line does, it should usually be broken up.
2. Don't overuse one-letter variables. Make it easy to figure out what they are.
   a. "x" for generic data is okay, "i", "j", and "k" are okay for indices, ***well-known*** math constants are okay.

```python
def H(x,b,l,h):
    i = np.floor((x-l)*b/(h-l)).astype(int)
    H = np.zeros(b,dtype=int)
    for j in i:
        if b>j>=0: H[j]+=1
    return H,np.linspace(l,h,b+1)
```

1. Just because you ***can*** fit something on one line doesn't mean you ***should***.
    a. Don't make any one line too complex: if you need to use "and" to describe what a line does, it should usually be broken up.
2. Don't overuse one-letter variables. Make it easy to figure out what they are.
    a. "x" for generic data is okay, "i", "j", and "k" are okay for indices, ***well-known*** math constants are okay.
3. Comment your functions.
    a. Make comments long enough to describe what's happening, but short enough that it doesn't feel like a chore to write them.

```python
def H(x,b,l,h):
    i = np.floor((x-l)*b/(h-l)).astype(int)
    H = np.zeros(b,dtype=int)
    for j in i:
        if b>j>=0: H[j]+=1
    return H,np.linspace(l,h,b+1)
```

1. Just because you ***can*** fit something on one line doesn't mean you ***should***.
    a. Don't make any one line too complex: if you need to use "and" to describe what a line does, it should usually be broken up.
2. Don't overuse one-letter variables. Make it easy to figure out what they are.
    a. "x" for generic data is okay, "i", "j", and "k" are okay for indices, ***well-known*** math constants are okay.
3. Comment your functions.
    a. Make comments long enough to describe what's happening, but short enough that it doesn't feel like a chore to write them.
4. Split code up into related "paragraphs" with newlines.

```python
def H(x,b,l,h):
    i = np.floor((x-l)*b/(h-l)).astype(int)
    H = np.zeros(b,dtype=int)
    for j in i:
        if b>j>=0: H[j]+=1
    return H,np.linspace(l,h,b+1)
```

1. Just because you ***can*** fit something on one line doesn't mean you ***should***.
    a. Don't make any one line too complex: if you need to use "and" to describe what a line does, it should usually be broken up.
2. Don't overuse one-letter variables. Make it easy to figure out what they are.
    a. "x" for generic data is okay, "i", "j", and "k" are okay for indices, ***well-known*** math constants are okay.
3. Comment your functions.
    a. Make comments long enough to describe what's happening, but short enough that it doesn't feel like a chore to write them.
4. Split code up into related "paragraphs" with newlines.
5. Eventually, find a style guideline and follow it.
    a. I think this one is the best: http://google.github.io/styleguide/pyguide.html

Testing for Bugs

Before relying on the results of your code (whether it’s for a homework assignment, research or industry), ***always*** make sure that it’s doing what you expect it to do.

Start by making your expectations simple and testing them.

Then test “edge cases.”

It’s good to make your tests automated so you can rerun them whenever you make changes, but manually

Plot-based tests are great.

If you want to do the bare minimum:

```
levi-civita:scratch phil$ python
Python 2.7.5 (default, Mar  9 2014, 22:15:05)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import debugging_examples
>>> import numpy as np
>>> bins = 5
>>> low, high = -5, 5
>>> debugging_examples.histogram(np.array([]), bins, low, high)
(array([0, 0, 0, 0, 0]), array([-5., -3., -1.,  1.,  3.,  5.]))
>>> debugging_examples.histogram(np.array([0]), bins, low, high)
(array([0, 0, 1, 0, 0]), array([-5., -3., -1.,  1.,  3.,  5.]))
>>> debugging_examples.histogram(np.array([-4, -2, 0, 2, 4]), bins, low, high)
(array([1, 1, 1, 1, 1]), array([-5., -3., -1.,  1.,  3.,  5.]))
>>> debugging_examples.histogram(np.array([4, 5, 6]), bins, low, high)
(array([0, 0, 0, 0, 1]), array([-5., -3., -1.,  1.,  3.,  5.]))
>>> debugging_examples.histogram(np.array([-6, -5, -4]), bins, low, high)
(array([2, 0, 0, 0, 0]), array([-5., -3., -1.,  1.,  3.,  5.]))
>>>
```

If you want to do the bare minimum:

```
levi-civita:scratch phil$ python
Python 2.7.5 (default, Mar  9 2014, 22:15:05)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import debugging_examples
>>> import numpy as np
>>> bins = 5
>>> low, high = -5, 5
>>> debugging_examples.histogram(np.array([]), bins, low, high)
(array([0, 0, 0, 0, 0]), array([-5., -3., -1.,  1.,  3.,  5.]))
>>> debugging_examples.histogram(np.array([0]), bins, low, high)
(array([0, 0, 1, 0, 0]), array([-5., -3., -1.,  1.,  3.,  5.]))
>>> debugging_examples.histogram(np.array([-4, -2, 0, 2, 4]), bins, low, high)
(array([1, 1, 1, 1, 1]), array([-5., -3., -1.,  1.,  3.,  5.]))
>>> debugging_examples.histogram(np.array([4, 5, 6]), bins, low, high)
(array([0, 0, 0, 0, 1]), array([-5., -3., -1.,  1.,  3.,  5.]))
>>> debugging_examples.histogram(np.array([-6, -5, -4]), bins, low, high)
(array([2, 0, 0, 0, 0]), array([-5., -3., -1.,  1.,  3.,  5.]))
>>> 
```

Just open a shell or make a new cell in an ipython notebook.

If you want to do the bare minimum:

```
levi-civita:scratch phil$ python
Python 2.7.5 (default, Mar  9 2014, 22:15:05)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import debugging_examples
>>> import numpy as np
>>> bins = 5
>>> low, high = -5, 5
>>> debugging_examples.histogram(np.array([]), bins, low, high)
(array([0, 0, 0, 0, 0]), array([-5., -3., -1.,  1.,  3.,  5.]))
>>> debugging_examples.histogram(np.array([0]), bins, low, high)
(array([0, 0, 1, 0, 0]), array([-5., -3., -1.,  1.,  3.,  5.]))
>>> debugging_examples.histogram(np.array([-4, -2, 0, 2, 4]), bins, low, high)
(array([1, 1, 1, 1, 1]), array([-5., -3., -1.,  1.,  3.,  5.]))
>>> debugging_examples.histogram(np.array([4, 5, 6]), bins, low, high)
(array([0, 0, 0, 0, 1]), array([-5., -3., -1.,  1.,  3.,  5.]))
>>> debugging_examples.histogram(np.array([-6, -5, -4]), bins, low, high)
(array([2, 0, 0, 0, 0]), array([-5., -3., -1.,  1.,  3.,  5.]))
>>> 
```

Try the simplest possible input that you can first. If something goes wrong, it will go wrong in a simple way.

If you want to do the bare minimum:

```
levi-civita:scratch phil$ python
Python 2.7.5 (default, Mar  9 2014, 22:15:05)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import debugging_examples
>>> import numpy as np
>>> bins = 5
>>> low, high = -5, 5
>>> debugging_examples.histogram(np.array([]), bins, low, high)
(array([0, 0, 0, 0, 0]), array([-5., -3., -1.,  1.,  3.,  5.]))
>>> debugging_examples.histogram(np.array([0]), bins, low, high)
(array([0, 0, 1, 0, 0]), array([-5., -3., -1.,  1.,  3.,  5.]))
>>> debugging_examples.histogram(np.array([-4, -2, 0, 2, 4]), bins, low, high)
(array([1, 1, 1, 1, 1]), array([-5., -3., -1.,  1.,  3.,  5.]))
>>> debugging_examples.histogram(np.array([4, 5, 6]), bins, low, high)
(array([0, 0, 0, 0, 1]), array([-5., -3., -1.,  1.,  3.,  5.]))
>>> debugging_examples.histogram(np.array([-6, -5, -4]), bins, low, high)
(array([2, 0, 0, 0, 0]), array([-5., -3., -1.,  1.,  3.,  5.]))
>>>
```

Then try the next simplest thing.

If you want to do the bare minimum:

```
levi-civita:scratch phil$ python
Python 2.7.5 (default, Mar  9 2014, 22:15:05)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import debugging_examples
>>> import numpy as np
>>> bins = 5
>>> low, high = -5, 5
>>> debugging_examples.histogram(np.array([]), bins, low, high)
(array([0, 0, 0, 0, 0]), array([-5., -3., -1.,  1.,  3.,  5.]))
>>> debugging_examples.histogram(np.array([0]), bins, low, high)
(array([0, 0, 1, 0, 0]), array([-5., -3., -1.,  1.,  3.,  5.]))
>>> debugging_examples.histogram(np.array([-4, -2, 0, 2, 4]), bins, low, high)
(array([1, 1, 1, 1, 1]), array([-5., -3., -1.,  1.,  3.,  5.]))
>>> debugging_examples.histogram(np.array([4, 5, 6]), bins, low, high)
(array([0, 0, 0, 0, 1]), array([-5., -3., -1.,  1.,  3.,  5.]))
>>> debugging_examples.histogram(np.array([-6, -5, -4]), bins, low, high)
(array([2, 0, 0, 0, 0]), array([-5., -3., -1.,  1.,  3.,  5.]))
>>>
```

Try something closer to normal usage.

If you want to do the bare minimum:

```
levi-civita:scratch phil$ python
Python 2.7.5 (default, Mar  9 2014, 22:15:05)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import debugging_examples
>>> import numpy as np
>>> bins = 5
>>> low, high = -5, 5
>>> debugging_examples.histogram(np.array([]), bins, low, high)
(array([0, 0, 0, 0, 0]), array([-5., -3., -1.,  1.,  3.,  5.]))
>>> debugging_examples.histogram(np.array([0]), bins, low, high)
(array([0, 0, 1, 0, 0]), array([-5., -3., -1.,  1.,  3.,  5.]))
>>> debugging_examples.histogram(np.array([-4, -2, 0, 2, 4]), bins, low, high)
(array([1, 1, 1, 1, 1]), array([-5., -3., -1.,  1.,  3.,  5.]))
>>> debugging_examples.histogram(np.array([4, 5, 6]), bins, low, high)
(array([0, 0, 0, 0, 1]), array([-5., -3., -1.,  1.,  3.,  5.]))
>>> debugging_examples.histogram(np.array([-6, -5, -4]), bins, low, high)
(array([2, 0, 0, 0, 0]), array([-5., -3., -1.,  1.,  3.,  5.]))
>>> 
```

Try weird cases. What if a point is on the edge of a histogram? What if it's outside?

If you want to do the bare minimum:

```
levi-civita:scratch phil$ python
Python 2.7.5 (default, Mar  9 2014, 22:15:05)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import debugging_examples
>>> import numpy as np
>>> bins = 5
>>> low, high = -5, 5
>>> debugging_examples.histogram(np.array([]), bins, low, high)
(array([0, 0, 0, 0, 0]), array([-5., -3., -1.,  1.,  3.,  5.]))
>>> debugging_examples.histogram(np.array([0]), bins, low, high)
(array([0, 0, 1, 0, 0]), array([-5., -3., -1.,  1.,  3.,  5.]))
>>> debugging_examples.histogram(np.array([-4, -2, 0, 2, 4]), bins, low, high)
(array([1, 1, 1, 1, 1]), array([-5., -3., -1.,  1.,  3.,  5.]))
>>> debugging_examples.histogram(np.array([4, 5, 6]), bins, low, high)
(array([0, 0, 0, 0, 1]), array([-5., -3., -1.,  1.,  3.,  5.]))
>>> debugging_examples.histogram(np.array([-6, -5, -4]), bins, low, high)
(array([2, 0, 0, 0, 0]), array([-5., -3., -1.,  1.,  3.,  5.]))
>>>
```

Try other weird cases, even if they're similar.

But what if you want to change things?

```python
def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) creates a histogram from the 1D array
    `data with `bins` bins that ranges from [`low`, `high`).

    Returns a tuple (`hist`, `edges`), where `hist` contains the number of
    elements in each histogram bin and `edges` contains every bin edge.
    """
    assert(bins > 0)
    assert(high > low)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).astype(int)
    in_range = (bin_index >= 0) & (bin_index < bins)
    hist = np.bincount(bin_index[in_range])
    edges = np.linspace(low, high, bins+1)

    return hist, edges
```

Hey, I just remembered that there's this built in numpy function that counts things in bins! Maybe I can skip the slow for loop! ᕕ(ᐛ)ᕗ

Auto-running tests are better:

```
def test_histogram():
    bins = 5
    low, high = -5, 5

    hist, edges = histogram(np.array([]), bins, low, high)
    assert(np.array_equal([0, 0, 0, 0, 0], hist))
    assert(np.allclose([-5, -3, -1, 1, 3, 5], edges))

    hist, edges = histogram(np.array([0]), bins, low, high)
    assert(np.array_equal([0, 0, 1, 0, 0], hist))
    assert(np.allclose([-5, -3, -1, 1, 3, 5], edges))

    hist, edges = histogram(np.array([-4, -2, 0, 2, 4]), bins, low, high)
    assert(np.array_equal([1, 1, 1, 1, 1], hist))
    assert(np.allclose([-5, -3, -1, 1, 3, 5], edges))

    hist, edges = histogram(np.array([-6, -5, -4]), bins, low, high)
    assert(np.array_equal([2, 0, 0, 0, 0], hist))
    assert(np.allclose([-5, -3, -1, 1, 3, 5], edges))

    hist, edges = histogram(np.array([4, 5, 6]), bins, low, high)
    assert(np.array_equal([0, 0, 0, 0, 1], hist))
    assert(np.allclose([-5, -3, -1, 1, 3, 5], edges))
```

What it looks like when things go wrong:

```
levi-civita:scratch phil$ python debugging_examples.py
Traceback (most recent call last):
  File "debugging_examples.py", line 90, in <module>
    if __name__ == "__main__": main()
  File "debugging_examples.py", line 75, in main
    test_histogram()
  File "debugging_examples.py", line 60, in test_histogram
    assert(np.array_equal([0, 0, 0, 0, 0], hist))
AssertionError
```

Last line says “AssertionError”

Previous line tells you where things went wrong.

Plots are a great way to qualitatively test things

```
    data = random.randn(100000)*3 + 2
    bins = 100
    hist, edges = histogram(data, 100, low, high)
    centers = (edges[1:] + edges[:-1]) / 2

    dx = (high - low) / bins
    expected = gaussian(centers, 2, 3, len(data)) * dx

    plt.plot(centers, hist, "o", c="k")
    plt.plot(centers, expected, c="r")

def gaussian(x, mu, sigma, area):
    """ gaussian(x, mu, sigma, area) evaluates a Gaussian centered on `mu`
    with a standard deviation of `sigma`. It is normalized so the total integral
    is `area`.
    """
    amplitude = area / (sigma * np.sqrt(2 * np.pi))
    return np.exp(-0.5 * ((x - mu) / sigma)**2) * amplitude
```

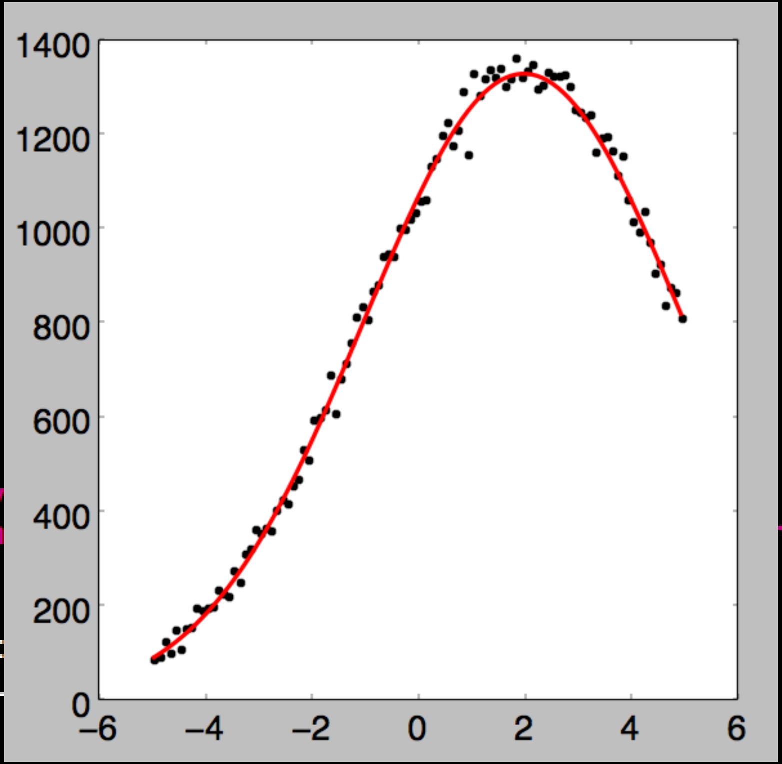
Plots are a great way to qualitatively test things

```
    data = random.randn(100000)*3 + 2
    bins = 100
    hist, edges = histogram(data, 100,
    centers = (edges[1:] + edges[:-1])

    dx = (high - low) / bins
    expected = gaussian(centers, 2, 3,

    plt.plot(centers, hist, "o", c="k")
    plt.plot(centers, expected, c="r")

def gaussian(x, mu, sigma, area):
    """ gaussian(x, mu, sigma, area) ev
    with a standard deviation of `sigma                    ral
    is `area`.
    """
    amplitude = area / (sigma * np.sqrt
    return np.exp(-0.5 * ((x - mu) / si
```



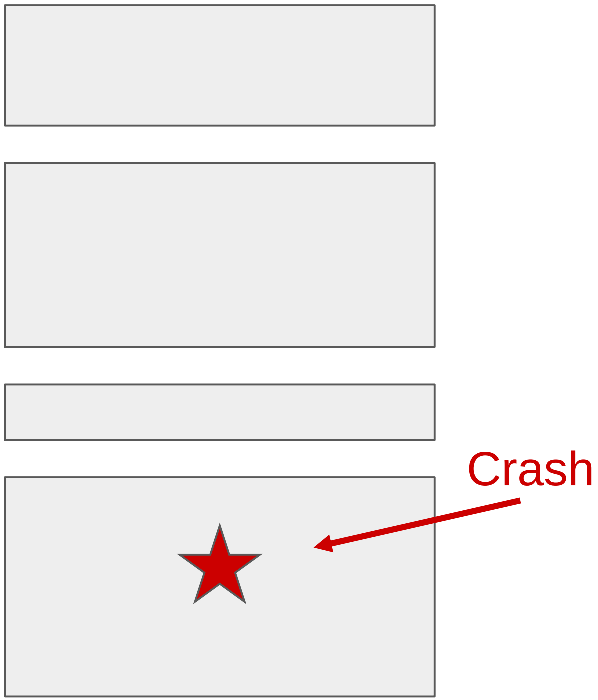
More Advanced Testing Tips:

1. Whenever testing code that generates random numbers or whenever you write tests that use random numbers, call random.seed(0) before testing to make sure that results are reproducible
2. After you're used to assertion-based testing, pick up a testing framework.
   a. pytest is probably the best one: https://docs.pytest.org/en/latest/
   b. The most important thing is getting into the habit of testing, no matter what your procedure is.
3. You shouldn't exactly compare floats in tests with "==". Use np.allclose().
4. If there are any conditionals that aren't activated by your tests, you aren't done yet.
5. If your code has to read some big messy file, make a small fake one for tests.
6. You can use "try" … "except" to make sure code crashes when it's supposed to.

Finding your bug:

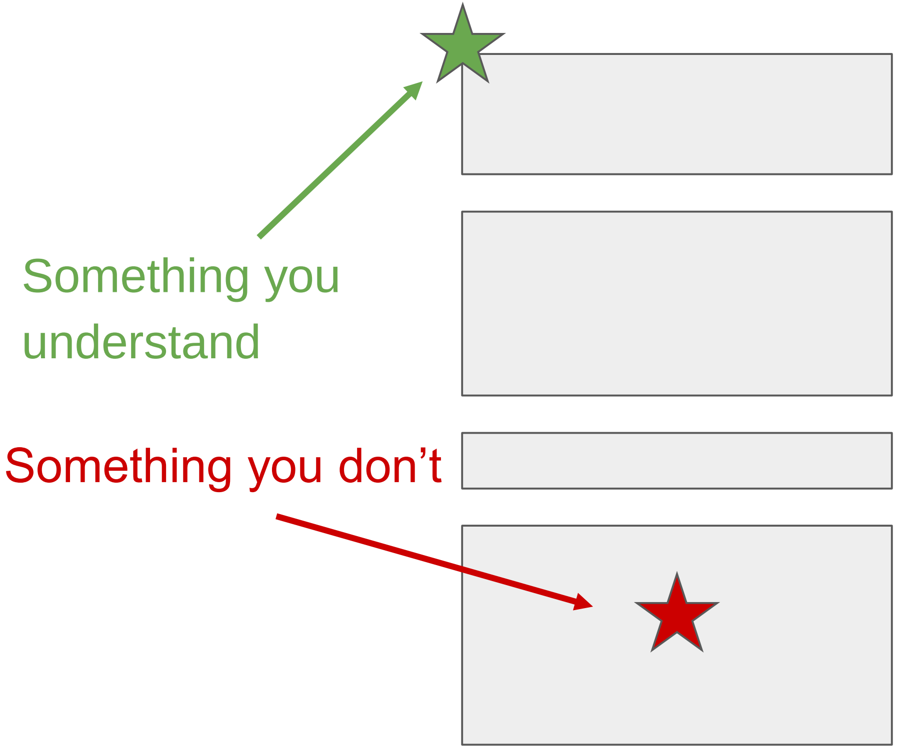
When you start debugging, you either failed a test or your code crashed. You do not understand what your code is doing at this point.

Cartoon version of your code.



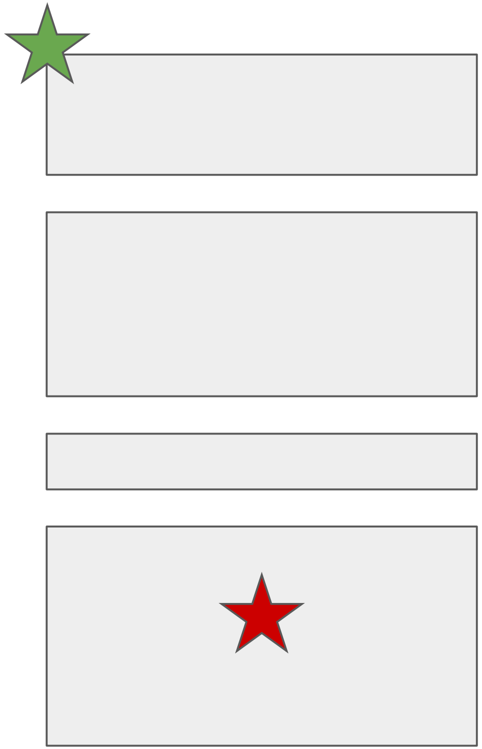
Finding your bug:

You gave this code some sort of input of input that you *did* understand.



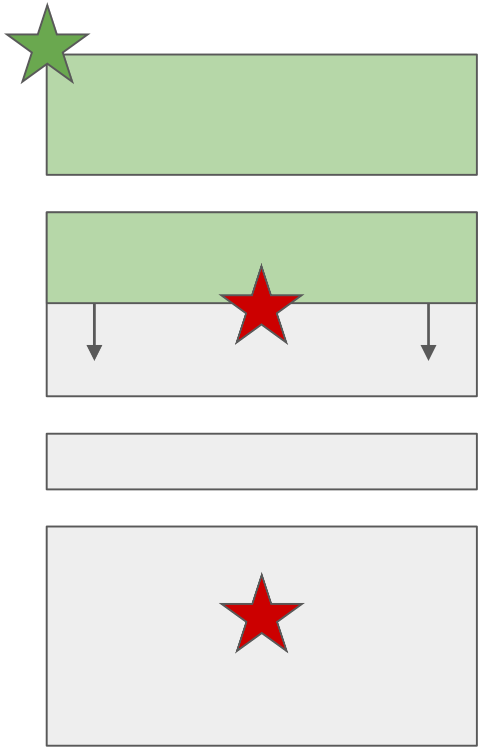
Finding your bug:

The goal is to find the line of code where things transition from making sense to not making sense.



Finding your bug:

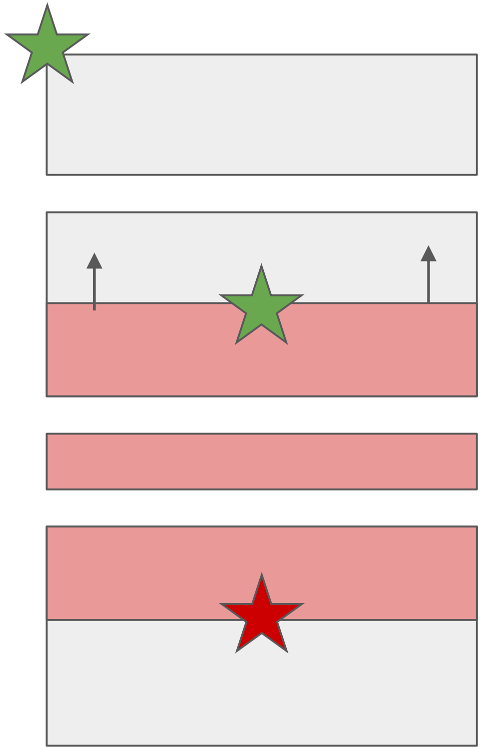
You do this either by working down and finding the first place where things don't make sense...



Finding your bug:

You do this either by working down and finding the first place where things are wrong or don't make sense…

… or by working up and finding the last place where they were right.

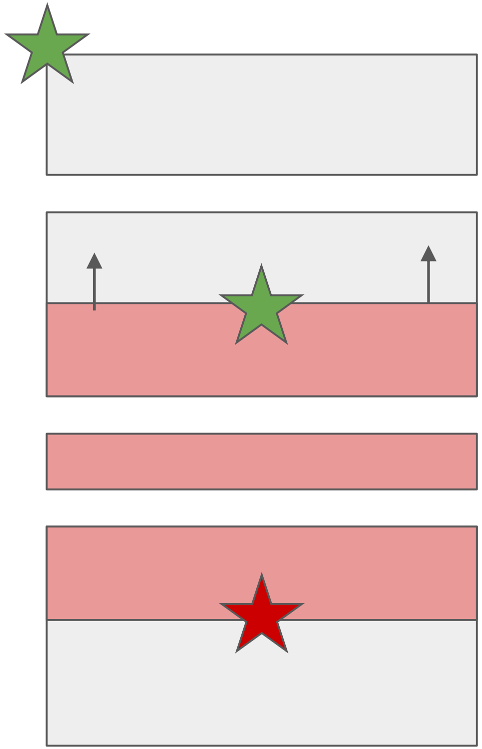


Finding your bug:

You do this either by working down and finding the first place where things are wrong or don't make sense…

… or by working up and finding the last place where they were right.

(This won't always be where the code crashed.)



Finding your bug:

The first step is making 100% sure that your inputs are right.

After this, use print() functions to show what values variables hold. (You can also use debuggers.)

```python
def main():
    Fe_H = np.loadtxt("globular_cluster_metalicity.txt")
    bins = optimal_bins(Fe_H)
    globular_cluster_distribution(Fe_H, bins)

def optimal_bins(x):
    """ optimal_bins(fname) computes the optimal number of bins for the
    dataset `x` using Doane's rule.
    """
    assert(len(x) > 2)

    skew = stats.skew(x)
    n = len(x)
    sigma_g1 = np.sqrt(6*(n - 2) / (n + 1) / (n + 3))

    return int(1 + np.log2(n) + np.log2(1 + skew/sigma_g1))

def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) creates a histogram from the 1D array
    `data with `bins` bins that ranges from [`low`, `high`).

    Returns a tuple (`hist`, `edges`), where `hist` contains the number of
    elements in each histogram bin and `edges` contains every bin edge.
    """
    assert(bins > 0)
    assert(high > low)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).astype(int)

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx <= bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges

def globular_cluster_distribution(Fe_H, bins):
    """ globular_cluster_distirbution(Fe_H, bins) plots the distirbution of
    globular cluster metalicities, `Fe_H` using `bins` bins.
    """
    assert(bins > 0)

    plt.figure()

    hist, edges = histogram(Fe_H, bins, -3.0, 1.0)
    centers = (edges[1:] + edges[:-1]) / 2

    plt.plot(centers, hist, "o", c="k")
    plt.xlabel(r"${\rm [Fe/H]}$")
    plt.ylabel(r"$N$")
```

```python
def main():
    Fe_H = np.loadtxt("globular_cluster_metalicity.txt")
    bins = optimal_bins(Fe_H)
    globular_cluster_distribution(Fe_H, bins)

def optimal_bins(x):
    """ optimal_bins(fname) computes the optimal number of bins for the
    dataset `x` using Doane's rule.
    """
    assert(len(x) > 2)

    skew = stats.skew(x)
    n = len(x)
    sigma_g1 = np.sqrt(6*(n - 2) / (n + 1) / (n + 3))

    return int(1 + np.log2(n) + np.log2(1 + skew/sigma_g1))

def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) creates a histogram from the 1D array
    `data with `bins` bins that ranges from [`low`, `high`).

    Returns a tuple (`hist`, `edges`), where `hist` contains the number of
    elements in each histogram bin and `edges` contains every bin edge.
    """
    assert(bins > 0)
    assert(high > low)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).astype(int)

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx <= bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges

def globular_cluster_distribution(Fe_H, bins):
    """ globular_cluster_distirbution(Fe_H, bins) plots the distirbution of
    globular cluster metalicities, `Fe_H` using `bins` bins.
    """
    assert(bins > 0)

    plt.figure()

    hist, edges = histogram(Fe_H, bins, -3.0, 1.0)
    centers = (edges[1:] + edges[:-1]) / 2

    plt.plot(centers, hist, "o", c="k")
    plt.xlabel(r"${\rm [Fe/H]}$")
    plt.ylabel(r"$N$")
```


```python
def main():
    Fe_H = np.loadtxt("globular_cluster_metalicity.txt")
    bins = optimal_bins(Fe_H)
    globular_cluster_distribution(Fe_H, bins)
```

```python
def main():
    Fe_H = np.loadtxt("globular_cluster_metalicity.txt")
    bins = optimal_bins(Fe_H)
    globular_cluster_distribution(Fe_H, bins)

def optimal_bins(x):
    """ optimal_bins(fname) computes the optimal number of bins for the
    dataset `x` using Doane's rule.
    """
    assert(len(x) > 2)

    skew = stats.skew(x)
    n = len(x)
    sigma_g1 = np.sqrt(6*(n - 2) / (n + 1) / (n + 3))

    return int(1 + np.log2(n) + np.log2(1 + skew/sigma_g1))
def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) creates a histogram from
    `data with `bins` bins that ranges from [`low`, `high`).

    Returns a tuple (`hist`, `edges`), where `hist` contains the
    elements in each histogram bin and `edges` contains every bin
    """
    assert(bins > 0)
    assert(high > low)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).astype(int)

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx <= bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges

def globular_cluster_distribution(Fe_H, bins):
    """ globular_cluster_distirbution(Fe_H, bins) plots the distirbution of
    globular cluster metalicities, `Fe_H` using `bins` bins.
    """
    assert(bins > 0)

    plt.figure()

    hist, edges = histogram(Fe_H, bins, -3.0, 1.0)
    centers = (edges[1:] + edges[:-1]) / 2

    plt.plot(centers, hist, "o", c="k")
    plt.xlabel(r"${\rm [Fe/H]}$")
    plt.ylabel(r"$N$")
```

```python
def optimal_bins(x):
    """ optimal_bins(x) computes the optimal number of bins for the
    dataset `x` using Doane's rule.
    """
    assert(len(x) > 2)

    skew = stats.skew(x)
    n = len(x)
    sigma_g1 = np.sqrt(6*(n - 2) / (n + 1) / (n + 3))

    return int(1 + np.log2(n) + np.log2(1 + skew/sigma_g1))
```

```python
def main():
    Fe_H = np.loadtxt("globular_cluster_metalicity.txt")
    bins = optimal_bins(Fe_H)
    globular_cluster_distribution(Fe_H, bins)

def optimal_bins(x):
    """ optimal_bins(fname) computes the optimal number of bins for the
    dataset `x` using Doane's rule.
    """
    assert(len(x) > 2)

    skew = stats.skew(x)
    n = len(x)
    sigma_g1 = np.sqrt(6*(n - 2) / (n + 1) / (r

    return int(1 + np.log2(n) + np.log2(1 + ske

def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) create
    `data with `bins` bins that ranges from [`l

    Returns a tuple (`hist`, `edges`), where `h
    elements in each histogram bin and `edges`
    """
    assert(bins > 0)
    assert(high > low)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).astype(ir

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx <= bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges

def globular_cluster_distribution(Fe_H, bins):
    """ globular_cluster_distirbution(Fe_H, bir
    globular cluster metalicities, `Fe_H` using
    """
    assert(bins > 0)

    plt.figure()

    hist, edges = histogram(Fe_H, bins, -3.0, 1.0)
    centers = (edges[1:] + edges[:-1]) / 2

    plt.plot(centers, hist, "o", c="k")
    plt.xlabel(r"${\rm [Fe/H]}$")
    plt.ylabel(r"$N$")
```

```python
def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) creates a histogram from the 1D array
    `data with `bins` bins that ranges from [`low`, `high`).

    Returns a tuple (`hist`, `edges`), where `hist` contains the number of
    elements in each histogram bin and `edges` contains every bin edge.
    """
    assert(bins > 0)
    assert(high > low)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).astype(int)

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx <= bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges
```

```
def main():
    Fe_H = np.loadtxt("globular_cluster_metalicity.txt")
    bins = optimal_bins(Fe_H)
    globular_cluster_distribution(Fe_H, bins)

def optimal_bins(x):
    """ optimal_bins(fname) computes the optimal number of bins for the
    dataset `x` using Doane's rule.
    """
    assert(len(x) > 2)

    skew = stats.skew(x)
    n = len(x)
    sigma_g1 = np.sqrt(6*(n - 2) / (n + 1) / (n + 3))

    return int(1 + np.log2(n) + np.log2(1 + skew/sig

def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) creates a h
    `data with `bins` bins that ranges from [`low`,

    Returns a tuple (`hist`, `edges`), where `hist`
    elements in each histogram bin and `edges` conta
    """
    assert(bins > 0)
    assert(high > low)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).astype(int)

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx <= bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges

def globular_cluster_distribution(Fe_H, bins):
    """ globular_cluster_distirbution(Fe_H, bins) plots the distirbution of
    globular cluster metalicities, `Fe_H` using `bins` bins.
    """
    assert(bins > 0)

    plt.figure()

    hist, edges = histogram(Fe_H, bins, -3.0, 1.0)
    centers = (edges[1:] + edges[:-1]) / 2

    plt.plot(centers, hist, "o", c="k")
    plt.xlabel(r"${\rm [Fe/H]}$")
    plt.ylabel(r"$N$")
```


```
def globular_cluster_distribution(Fe_H, bins):
    """ globular_cluster_distirbution(Fe_H, bins) plots the distirbution of
    globular cluster metalicities, `Fe_H` using `bins` bins.
    """
    assert(bins > 0)

    plt.figure()

    hist, edges = histogram(Fe_H, bins, -3.0, 1.0)
    centers = (edges[1:] + edges[:-1]) / 2

    plt.plot(centers, hist, "o", c="k")
    plt.xlabel(r"${\rm [Fe/H]}$")
    plt.ylabel(r"$N$")
```

```python
def main():
    Fe_H = np.loadtxt("globular_cluster_metalicity.txt")
    bins = optimal_bins(Fe_H)
    globular_cluster_distribution(Fe_H, bins)

def optimal_bins(x):
    """ optimal_bins(fname) computes the optimal number of bins for the
    dataset `x` using Doane's rule.
    """
    assert(len(x) > 2)

    skew = stats.skew(x)
    n = len(x)
    sigma_g1 = np.sqrt(6*(n - 2) / (n + 1) / (n + 3))

    return int(1 + np.log2(n) + np.log2(1 + skew/sigma_g1))

def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) creates a histogram from the 1D array
    `data with `bins` bins that ranges from [`low`, `high`).

    Returns a tuple (`hist`, `edges`), where `hist` contains the number of
    elements in each histogram bin and `edges` contains every bin edge.
    """
    assert(bins > 0)
    assert(high > low)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).astype(int)

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx <= bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges

def globular_cluster_distribution(Fe_H, bins):
    """ globular_cluster_distirbution(Fe_H, bins) plots the distirbution of
    globular cluster metalicities, `Fe_H` using `bins` bins.
    """
    assert(bins > 0)

    plt.figure()

    hist, edges = histogram(Fe_H, bins, -3.0, 1.0)
    centers = (edges[1:] + edges[:-1]) / 2

    plt.plot(centers, hist, "o", c="k")
    plt.xlabel(r"${\rm [Fe/H]}$")
    plt.ylabel(r"$N$")
```

```python
def main():
    Fe_H = np.loadtxt("globular_cluster_metalicity.txt")
    bins = optimal_bins(Fe_H)
    globular_cluster_distribution(Fe_H, bins)

def optimal_bins(x):
    """ optimal_bins(fname) computes the optimal number of bins for the
    dataset `x` using Doane's rule.
    """
    assert(len(x) > 2)

    skew = stats.skew(x)
    n = len(x)
    sigma_g1 = np.sqrt(6*(n - 2) / (n + 1) / (n + 3))

    return int(1 + np.log2(n) + np.log2(1 + skew/sigma_g1))

def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) creates a histogram from the 1D array
    `data with `bins` bins that ranges from [`low`, `high`).

    Returns a tuple (`hist`, `edges`), where `hist` contains the number of
    elements in each histogram bin and `edges` contains every bin edge.
    """
    assert(bins > 0)
    assert(high > low)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).astype(int)

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx <= bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges

def globular_cluster_distribution(Fe_H, bins):
    """ globular_cluster_distirbution(Fe_H, bins) plots the distirbution of
    globular cluster metalicities, `Fe_H` using `bins` bins.
    """
    assert(bins > 0)

    plt.figure()

    hist, edges = histogram(Fe_H, bins, -3.0, 1.0)
    centers = (edges[1:] + edges[:-1]) / 2

    plt.plot(centers, hist, "o", c="k")
    plt.xlabel(r"${\rm [Fe/H]}$")
    plt.ylabel(r"$N$")
```

```
levi-civita:scratch phil$ python debugging_examples.py
Traceback (most recent call last):
  File "debugging_examples.py", line 166, in <module>
    if __name__ == "__main__": main()
  File "debugging_examples.py", line 114, in main
    globular_cluster_distribution(Fe_H, bins)
  File "debugging_examples.py", line 162, in globular_cluster_distribution
    plt.plot(centers, hist, "o", c="k")
  File "/Library/Python/2.7/site-packages/matplotlib-1.3.0-py2.7-macosx-10.
8-intel.egg/matplotlib/pyplot.py", line 2987, in plot
    ret = ax.plot(*args, **kwargs)
  File "/Library/Python/2.7/site-packages/matplotlib-1.3.0-py2.7-macosx-10.
8-intel.egg/matplotlib/axes.py", line 4137, in plot
    for line in self._get_lines(*args, **kwargs):
  File "/Library/Python/2.7/site-packages/matplotlib-1.3.0-py2.7-macosx-10.
8-intel.egg/matplotlib/axes.py", line 317, in _grab_next_args
    for seg in self._plot_args(remaining, kwargs):
  File "/Library/Python/2.7/site-packages/matplotlib-1.3.0-py2.7-macosx-10.
8-intel.egg/matplotlib/axes.py", line 295, in _plot_args
    x, y = self._xy_from_xy(x, y)
  File "/Library/Python/2.7/site-packages/matplotlib-1.3.0-py2.7-macosx-10.
8-intel.egg/matplotlib/axes.py", line 237, in _xy_from_xy
    raise ValueError("x and y must have same first dimension")
ValueError: x and y must have same first dimension
levi-civita:scratch phil$
```

```python
def main():
    Fe_H = np.loadtxt("globular_cluster_metalicity.txt")
    bins = optimal_bins(Fe_H)
    globular_cluster_distribution(Fe_H, bins)

def optimal_bins(x):
    """ optimal_bins(fname) computes the optimal number of bins for the
    dataset `x` using Doane's rule.
    """
    assert(len(x) > 2)

    skew = stats.skew(x)
    n = len(x)
    sigma_g1 = np.sqrt(6*(n - 2) / (n + 1) / (n + 3))

    return int(1 + np.log2(n) + np.log2(1 + skew/sigma_g1))

def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) creates a histogram from the 1D array
    `data with `bins` bins that ranges from [`low`, `high`).

    Returns a tuple (`hist`, `edges`), where `hist` contains the number of
    elements in each histogram bin and `edges` contains every bin edge.
    """
    assert(bins > 0)
    assert(high > low)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).astype(int)

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx <= bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges

def globular_cluster_distribution(Fe_H, bins):
    """ globular_cluster_distirbution(Fe_H, bins) plots the distirbution of
    globular cluster metalicities, `Fe_H` using `bins` bins.
    """
    assert(bins > 0)

    plt.figure()

    hist, edges = histogram(Fe_H, bins, -3.0, 1.0)
    centers = (edges[1:] + edges[:-1]) / 2

    plt.plot(centers, hist, "o", c="k")
    plt.xlabel(r"${\rm [Fe/H]}$")
    plt.ylabel(r"$N$")
```

```
levi-civita:scratch phil$ python debugging_examples.py
Traceback (most recent call last):
  File "debugging_examples.py", line 166, in <module>
    if __name__ == "__main__": main()
  File "debugging_examples.py", line 114, in main
    globular_cluster_distribution(Fe_H, bins)
  File "debugging_examples.py", line 162, in globular_cluster_distribution
    plt.plot(centers, hist, "o", c="k")
  File "/Library/Python/2.7/site-packages/matplotlib-1.3.0-py2.7-macosx-10.
8-intel.egg/matplotlib/pyplot.py", line 2987, in plot
    ret = ax.plot(*args, **kwargs)
  File "/Library/Python/2.7/site-packages/matplotlib-1.3.0-py2.7-macosx-10.
8-intel.egg/matplotlib/axes.py", line 4137, in plot
    for line in self._get_lines(*args, **kwargs):
  File "/Library/Python/2.7/site-packages/matplotlib-1.3.0-py2.7-macosx-10.
8-intel.egg/matplotlib/axes.py", line 317, in _grab_next_args
    for seg in self._plot_args(remaining, kwargs):
  File "/Library/Python/2.7/site-packages/matplotlib-1.3.0-py2.7-macosx-10.
8-intel.egg/matplotlib/axes.py", line 295, in _plot_args
    x, y = self._xy_from_xy(x, y)
  File "/Library/Python/2.7/site-packages/matplotlib-1.3.0-py2.7-macosx-10.
8-intel.egg/matplotlib/axes.py", line 237, in _xy_from_xy
    raise ValueError("x and y must have same first dimension")
ValueError: x and y must have same first dimension
levi-civita:scratch phil$
```

Read from the bottom up:

First is the error - a basic description of the issue

```python
def main():
    Fe_H = np.loadtxt("globular_cluster_metalicity.txt")
    bins = optimal_bins(Fe_H)
    globular_cluster_distribution(Fe_H, bins)

def optimal_bins(x):
    """ optimal_bins(fname) computes the optimal number of bins for the
    dataset `x` using Doane's rule.
    """
    assert(len(x) > 2)

    skew = stats.skew(x)
    n = len(x)
    sigma_g1 = np.sqrt(6*(n - 2) / (n + 1) / (n + 3))

    return int(1 + np.log2(n) + np.log2(1 + skew/sigma_g1))

def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) creates a histogram from the 1D array
    `data with `bins` bins that ranges from [`low`, `high`).

    Returns a tuple (`hist`, `edges`), where `hist` contains the number of
    elements in each histogram bin and `edges` contains every bin edge.
    """
    assert(bins > 0)
    assert(high > low)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).astype(int)

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx <= bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges

def globular_cluster_distribution(Fe_H, bins):
    """ globular_cluster_distirbution(Fe_H, bins) plots the distirbution of
    globular cluster metalicities, `Fe_H` using `bins` bins.
    """
    assert(bins > 0)

    plt.figure()

    hist, edges = histogram(Fe_H, bins, -3.0, 1.0)
    centers = (edges[1:] + edges[:-1]) / 2

    plt.plot(centers, hist, "o", c="k")
    plt.xlabel(r"${\rm [Fe/H]}$")
    plt.ylabel(r"$N$")
```

```
levi-civita:scratch phil$ python debugging_examples.py
Traceback (most recent call last):
  File "debugging_examples.py", line 166, in <module>
    if __name__ == "__main__": main()
  File "debugging_examples.py", line 114, in main
    globular_cluster_distribution(Fe_H, bins)
  File "debugging_examples.py", line 162, in globular_cluster_distribution
    plt.plot(centers, hist, "o", c="k")
  File "/Library/Python/2.7/site-packages/matplotlib-1.3.0-py2.7-macosx-10.
8-intel.egg/matplotlib/pyplot.py", line 2987, in plot
    ret = ax.plot(*args, **kwargs)
  File "/Library/Python/2.7/site-packages/matplotlib-1.3.0-py2.7-macosx-10.
8-intel.egg/matplotlib/axes.py", line 4137, in plot
    for line in self._get_lines(*args, **kwargs):
  File "/Library/Python/2.7/site-packages/matplotlib-1.3.0-py2.7-macosx-10.
8-intel.egg/matplotlib/axes.py", line 317, in _grab_next_args
    for seg in self._plot_args(remaining, kwargs):
  File "/Library/Python/2.7/site-packages/matplotlib-1.3.0-py2.7-macosx-10.
8-intel.egg/matplotlib/axes.py", line 295, in _plot_args
    x, y = self._xy_from_xy(x, y)
  File "/Library/Python/2.7/site-packages/matplotlib-1.3.0-py2.7-macosx-10.
8-intel.egg/matplotlib/axes.py", line 237, in _xy_from_xy
    raise ValueError("x and y must have same first dimension")
ValueError: x and y must have same first dimension
levi-civita:scratch phil$
```

Next could be your code, but in this case is a bunch of other peoples' code (look at the file names). Ignore it.

```python
def main():
    Fe_H = np.loadtxt("globular_cluster_metalicity.txt")
    bins = optimal_bins(Fe_H)
    globular_cluster_distribution(Fe_H, bins)

def optimal_bins(x):
    """ optimal_bins(fname) computes the optimal number of bins for the
    dataset `x` using Doane's rule.
    """
    assert(len(x) > 2)

    skew = stats.skew(x)
    n = len(x)
    sigma_g1 = np.sqrt(6*(n - 2) / (n + 1) / (n + 3))

    return int(1 + np.log2(n) + np.log2(1 + skew/sigma_g1))

def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) creates a histogram from the 1D array
    `data with `bins` bins that ranges from [`low`, `high`).

    Returns a tuple (`hist`, `edges`), where `hist` contains the number of
    elements in each histogram bin and `edges` contains every bin edge.
    """
    assert(bins > 0)
    assert(high > low)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).astype(int)

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx <= bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges

def globular_cluster_distribution(Fe_H, bins):
    """ globular_cluster_distirbution(Fe_H, bins) plots the distirbution of
    globular cluster metalicities, `Fe_H` using `bins` bins.
    """
    assert(bins > 0)

    plt.figure()

    hist, edges = histogram(Fe_H, bins, -3.0, 1.0)
    centers = (edges[1:] + edges[:-1]) / 2

    plt.plot(centers, hist, "o", c="k")
    plt.xlabel(r"${\rm [Fe/H]}$")
    plt.ylabel(r"$N$")
```

```
levi-civita:scratch phil$ python debugging_examples.py
Traceback (most recent call last):
  File "debugging_examples.py", line 166, in <module>
    if __name__ == "__main__": main()
  File "debugging_examples.py", line 114, in main
    globular_cluster_distribution(Fe_H, bins)
  File "debugging_examples.py", line 162, in globular_cluster_distribution
    plt.plot(centers, hist, "o", c="k")

    raise ValueError("x and y must have same first dimension")
ValueError: x and y must have same first dimension
levi-civita:scratch phil$
```

Next is the place in your code where things went wrong.

```python
def main():
    Fe_H = np.loadtxt("globular_cluster_metalicity.txt")
    bins = optimal_bins(Fe_H)
    globular_cluster_distribution(Fe_H, bins)

def optimal_bins(x):
    """ optimal_bins(fname) computes the optimal number of bins for the
    dataset `x` using Doane's rule.
    """
    assert(len(x) > 2)

    skew = stats.skew(x)
    n = len(x)
    sigma_g1 = np.sqrt(6*(n - 2) / (n + 1) / (n + 3))

    return int(1 + np.log2(n) + np.log2(1 + skew/sigma_g1))

def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) creates a histogram from the 1D array
    `data with `bins` bins that ranges from [`low`, `high`).

    Returns a tuple (`hist`, `edges`), where `hist` contains the number of
    elements in each histogram bin and `edges` contains every bin edge.
    """
    assert(bins > 0)
    assert(high > low)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).astype(int)

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx <= bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges

def globular_cluster_distribution(Fe_H, bins):
    """ globular_cluster_distirbution(Fe_H, bins) plots the distirbution of
    globular cluster metalicities, `Fe_H` using `bins` bins.
    """
    assert(bins > 0)

    plt.figure()

    hist, edges = histogram(Fe_H, bins, -3.0, 1.0)
    centers = (edges[1:] + edges[:-1]) / 2

    plt.plot(centers, hist, "o", c="k")
    plt.xlabel(r"${\rm [Fe/H]}$")
    plt.ylabel(r"$N$")
```

```
levi-civita:scratch phil$ python debugging_examples.py
Traceback (most recent call last):
  File "debugging_examples.py", line 166, in <module>
    if __name__ == "__main__": main()
  File "debugging_examples.py", line 114, in main
    globular_cluster_distribution(Fe_H, bins)
  File "debugging_examples.py", line 162, in globular_cluster_distribution
    plt.plot(centers, hist, "o", c="k")

    raise ValueError("x and y must have same first dimension")
ValueError: x and y must have same first dimension
levi-civita:scratch phil$
```

Next is the function that called that code (there may be several lines of this).

```
def main():
    Fe_H = np.loadtxt("globular_cluster_metalicity.txt")
    bins = optimal_bins(Fe_H)
    globular_cluster_distribution(Fe_H, bins)

def optimal_bins(x):
    """ optimal_bins(fname) computes the optimal number of bins for the
    dataset `x` using Doane's rule.
    """
    assert(len(x) > 2)

    skew = stats.skew(x)
    n = len(x)
    sigma_g1 = np.sqrt(6*(n - 2) / (n + 1) / (n + 3))

    return int(1 + np.log2(n) + np.log2(1 + skew/sigma_g1))

def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) creates a histogram from the 1D array
    `data with `bins` bins that ranges from [`low`, `high`).

    Returns a tuple (`hist`, `edges`), where `hist` contains the number of
    elements in each histogram bin and `edges` contains every bin edge.
    """
    assert(bins > 0)
    assert(high > low)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).astype(int)

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx <= bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges

def globular_cluster_distribution(Fe_H, bins):
    """ globular_cluster_distirbution(Fe_H, bins) plots the distirbution of
    globular cluster metalicities, `Fe_H` using `bins` bins.
    """
    assert(bins > 0)

    plt.figure()

    hist, edges = histogram(Fe_H, bins, -3.0, 1.0)
    centers = (edges[1:] + edges[:-1]) / 2

    plt.plot(centers, hist, "o", c="k")
    plt.xlabel(r"${\rm [Fe/H]}$")
    plt.ylabel(r"$N$")
```

We know things were correct here.

We crashed here.

```python
def main():
    Fe_H = np.loadtxt("globular_cluster_metalicity.txt")
    bins = optimal_bins(Fe_H)
    globular_cluster_distribution(Fe_H, bins)

def optimal_bins(x):
    """ optimal_bins(fname) computes the optimal number of bins for the
    dataset `x` using Doane's rule.
    """
    assert(len(x) > 2)

    skew = stats.skew(x)
    n = len(x)
    sigma_g1 = np.sqrt(6*(n - 2) / (n + 1) / (n + 3))

    return int(1 + np.log2(n) + np.log2(1 + skew/sigma_g1))

def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) creates a histogram from the 1D array
    `data with `bins` bins that ranges from [`low`, `high`).

    Returns a tuple (`hist`, `edges`), where `hist` contains the number of
    elements in each histogram bin and `edges` contains every bin edge.
    """
    assert(bins > 0)
    assert(high > low)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).astype(int)

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx <= bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges

def globular_cluster_distribution(Fe_H, bins):
    """ globular_cluster_distirbution(Fe_H, bins) plots the distirbution of
    globular cluster metalicities, `Fe_H` using `bins` bins.
    """
    assert(bins > 0)

    plt.figure()

    hist, edges = histogram(Fe_H, bins, -3.0, 1.0)
    centers = (edges[1:] + edges[:-1]) / 2

    plt.plot(centers, hist, "o", c="k")
    plt.xlabel(r"${\rm [Fe/H]}$")
    plt.ylabel(r"$N$")
```

We know things were correct here.

We need to check this code.

We crashed here.

```python
def main():
    Fe_H = np.loadtxt("globular_cluster_metalicity.txt")
    bins = optimal_bins(Fe_H)
    globular_cluster_distribution(Fe_H, bins)

def optimal_bins(x):
    """ optimal_bins(fname) computes the optimal number of bins for the
    dataset `x` using Doane's rule.
    """
    assert(len(x) > 2)

    skew = stats.skew(x)
    n = len(x)
    sigma_g1 = np.sqrt(6*(n - 2) / (n + 1) / (n + 3))

    return int(1 + np.log2(n) + np.log2(1 + skew/sigma_g1))

def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) creates a histogram from the 1D array
    `data with `bins` bins that ranges from [`low`, `high`).

    Returns a tuple (`hist`, `edges`), where `hist` contains the number of
    elements in each histogram bin and `edges` contains every bin edge.
    """
    assert(bins > 0)
    assert(high > low)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).astype(int)

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx <= bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges

def globular_cluster_distribution(Fe_H, bins):
    """ globular_cluster_distirbution(Fe_H, bins) plots the distirbution of
    globular cluster metalicities, `Fe_H` using `bins` bins.
    """
    assert(bins > 0)

    plt.figure()

    hist, edges = histogram(Fe_H, bins, -3.0, 1.0)
    centers = (edges[1:] + edges[:-1]) / 2

    plt.plot(centers, hist, "o", c="k")
    plt.xlabel(r"$[\rm [Fe/H]]$")
    plt.ylabel(r"$N$")
```

We know things were correct here.

...and the functions that code calls.

We need to check this code.

We crashed here.

Stop!

That’s a lot of lines of code to search! We need to cut down on our search region.

1. The most likely cause of the bug is that there’s some mistake on the line that crashed.
2. The next most likely is that the function inputs were bad.
3. The next is that something in the function (or the functions it calls) is wrong.

Always use this pattern: (1) check the crashing line, (2) check the function inputs, (3) check the code in between. If steps 2 or 3 take you to another function, do the same there.

```python
def main():
    Fe_H = np.loadtxt("globular_cluster_metalicity.txt")
    bins = optimal_bins(Fe_H)
    globular_cluster_distribution(Fe_H, bins)

def optimal_bins(x):
    """ optimal_bins(fname) computes the optimal number of bins for the
    dataset `x` using Doane's rule.
    """
    assert(len(x) > 2)

    skew = stats.skew(x)
    n = len(x)
    sigma_g1 = np.sqrt(6*(n - 2) / (n + 1) / (n + 3))

    return int(1 + np.log2(n) + np.log2(1 + skew/si

def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) creates a
    `data with `bins` bins that ranges from [`low`,

    Returns a tuple (`hist`, `edges`), where `hist`
    elements in each histogram bin and `edges` cont
    """
    assert(bins > 0)
    assert(high > low)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).astype(int)

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx <= bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges

def globular_cluster_distribution(Fe_H, bins):
    """ globular_cluster_distirbution(Fe_H, bins) plots the distirbution of
    globular cluster metalicities, `Fe_H` using `bins` bins.
    """
    assert(bins > 0)

    plt.figure()

    hist, edges = histogram(Fe_H, bins, -3.0, 1.0)
    centers = (edges[1:] + edges[:-1]) / 2

    plt.plot(centers, hist, "o", c="k")
    plt.xlabel(r"${\rm [Fe/H]}$")
    plt.ylabel(r"$N$")
```


```python
def globular_cluster_distribution(Fe_H, bins):
    """ globular_cluster_distirbution(Fe_H, bins) plots the distirbution of
    globular cluster metalicities, `Fe_H` using `bins` bins.
    """
    assert(bins > 0)

    plt.figure()

    hist, edges = histogram(Fe_H, bins, -3.0, 1.0)
    centers = (edges[1:] + edges[:-1]) / 2

    plt.plot(centers, hist, "o", c="k")
    plt.xlabel(r"${\rm [Fe/H]}$")
    plt.ylabel(r"$N$")
```

```python
def main():
    Fe_H = np.loadtxt("globular_cluster_metalicity.txt")
    bins = optimal_bins(Fe_H)
    globular_cluster_distribution(Fe_H, bins)

def optimal_bins(x):
    """ optimal_bins(fname) computes the optimal number of bins for the
    dataset `x` using Doane's rule.
    """
    assert(len(x) > 2)

    skew = stats.skew(x)
    n = len(x)
    sigma_g1 = np.sqrt(6*(n - 2) / (n + 1) / (n + 3))

    return int(1 + np.log2(n) + np.log2(1 + skew/sig

def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) creates a
    `data with `bins` bins that ranges from [`low`,

    Returns a tuple (`hist`, `edges`), where `hist`
    elements in each histogram bin and `edges` cont
    """
    assert(bins > 0)
    assert(high > low)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).astype(int)

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx <= bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges

def globular_cluster_distribution(Fe_H, bins):
    """ globular_cluster_distirbution(Fe_H, bins) plots the distirbution of
    globular cluster metalicities, `Fe_H` using `bins` bins
    """
    assert(bins > 0)

    plt.figure()

    hist, edges = histogram(Fe_H, bins, -3.0, 1.0)
    centers = (edges[1:] + edges[:-1]) / 2

    plt.plot(centers, hist, "o", c="k")

    plt.ylabel(r"$N$")
```

```python
plt.plot(centers, hist, "o", c="k")
```

First step: check the line. Does anything look wrong here?

```python
def main():
    Fe_H = np.loadtxt("globular_cluster_metalicity.txt")
    bins = optimal_bins(Fe_H)
    globular_cluster_distribution(Fe_H, bins)

def optimal_bins(x):
    """ optimal_bins(fname) computes the optimal number of bins for the
    dataset `x` using Doane's rule.
    """
    assert(len(x) > 2)

    skew = stats.skew(x)
    n = len(x)
    sigma_g1 = np.sqrt(6*(n - 2) / (n + 1) / (n + 3))

    return int(1 + np.log2(n) + np.log2(1 + skew/si

def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) creates a 
    `data with `bins` bins that ranges from [`low`,

    Returns a tuple (`hist`, `edges`), where `hist`
    elements in each histogram bin and `edges` cont
    """
    assert(bins > 0)
    assert(high > low)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).astype(int)

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx <= bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges

def globular_cluster_distribution(Fe_H, bins):
    """ globular_cluster_distirbution(Fe_H, bins) plots the distirbution of
    globular cluster metalicities, `Fe_H` using `bins` bins.
    """
    assert(bins > 0)

    plt.figure()

    hist, edges = histogram(Fe_H, bins, -3.0, 1.0)
    centers = (edges[1:] + edges[:-1]) / 2

    plt.plot(centers, hist, "o", c="k")
    plt.xlabel(r"${\rm [Fe/H]}$")
    plt.ylabel(r"$N$")
```

```python
def main():
    Fe_H = np.loadtxt("globular_cluster_metalicity.txt")
    bins = optimal_bins(Fe_H)
    globular_cluster_distribution(Fe_H, bins)

def optimal_bins(x):
    """ optimal_bins(fname) computes the optimal number of bins for the
    dataset `x` using Doane's rule.
    """
    assert(len(x) > 2)

    skew = stats.skew(x)
    n = len(x)
    sigma_g1 = np.sqrt(6*(n - 2) / (n + 1) / (n + 3)

    return int(1 + np.log2(n) + np.log2(1 + skew/sic

def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) creates a h
    `data with `bins` bins that ranges from [`low`,

    Returns a tuple (`hist`, `edges`), where `hist`
    elements in each histogram bin and `edges` conta
    """
    assert(bins > 0)
    assert(high > low)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).astype(int)

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx <= bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges

def globular_cluster_distribution(Fe_H, bins):
    """ globular_cluster_distirbution(Fe_H, bins) plots the distirbution of
    globular cluster metalicities, `Fe_H` using `bins` bins.
    """
    assert(bins > 0)

    plt.figure()

    hist, edges = histogram(Fe_H, bins, -3.0, 1.0)
    centers = (edges[1:] + edges[:-1]) / 2

    plt.plot(centers, hist, "o", c="k")
    plt.ylabel(r"$N$")
```

```python
def globular_cluster_distribution(Fe_H, bins):
    """ globular_cluster_distirbution(Fe_H, bins) plots the distirbution of
    globular cluster metalicities, `Fe_H` using `bins` bins.
    """
    assert(bins > 0)
    print("Fe_H:", Fe_H)
    print("bins:", bins)
```

Second step: are the function inputs reasonable?

```python
def main():
    Fe_H = np.loadtxt("globular_cluster_metalicity.txt")
    bins = optimal_bins(Fe_H)
    globular_cluster_distribution(Fe_H, bins)

def optimal_bins(x):
    """ optimal_bins(fname) computes the optimal number of bins for the
    dataset `x` using Doane's rule.
    """
    assert(len(x) > 2)

    skew = stats.skew(x)
    n = len(x)
    sigma_g1 = np.sqrt(6*(n - 2) / (n + 1) / (n + 3)

    return int(1 + np.log2(n) + np.log2(1 + skew/sig

def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) creates a h
    `data with `bins` bins that ranges from [`low`,

    Returns a tuple (`hist`, `edges`), where `hist`
    elements in each histogram bin and `edges` conta
    """
    assert(bins > 0)
    assert(high > low)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).astype(int)

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx <= bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges

def globular_cluster_distribution(Fe_H, bins):
    """ globular_cluster_distirbution(Fe_H, bins)
    globular cluster metalicities, `Fe_H` using `

    assert(bins > 0)

    plt.figure()

    hist, edges = histogram(Fe_H, bins, -3.0, 1.0)
    centers = (edges[1:] + edges[:-1]) / 2

    plt.plot(centers, hist, "o", c="k")
    plt.xlabel(r"$[\rm [Fe/H]]$")
    plt.ylabel(r"$N$")
```

```python
def globular_cluster_distribution(Fe_H, bins):
    """" globular_cluster_distirbution(Fe_H, bins) plots the distirbution of
    globular cluster metalicities, `Fe_H` using `bins` bins.
    """"
    assert(bins > 0)
    print("Fe_H:", Fe_H)
    print("bins:", bins)
```

```
levi-civita:scratch phil$ python debugging_examples.py
Fe_H: [-3.6401761  -2.62104217 -4.59689928 -4.50583312 -4.62252371  3.1086572
 -2.76378391 -2.79293035 -1.38716174  0.82455294 -7.85408701 -2.29753269
 -3.31201677 -3.75630165 -4.01417183 -5.23256656 -2.45843828 -2.1861628
 -2.78822755 -1.69699626 -1.63649125 -4.16953641 -2.89774636 -0.08939127
 -2.04364833 -0.86384391 -1.62295559  1.92094566 -0.08021028 -0.47251922
 -4.51679014 -4.11206368 -2.09035686 -0.83837957  2.43439993 -1.89542316
 -1.42353272 -4.79614936 -2.85842067  0.41540516 -2.79320033 -1.13293636
 -3.34792346 -2.65280793 -3.43679299 -0.61708158  2.93588617 -3.65491541
 -1.94661883 -4.19282036]
bins: 7
```

Yep! Those are valid inputs. Error is in this function or its children.

```
def main():
    Fe_H = np.loadtxt("globular_cluster_metalicity.txt")
    bins = optimal_bins(Fe_H)
    globular_cluster_distribution(Fe_H, bins)

def optimal_bins(x):
    """ optimal_bins(fname) computes the optimal number of bins for the
    dataset `x` using Doane's rule.
    """
    assert(len(x) > 2)

    skew = stats.skew(x)
    n = len(x)
    sigma_g1 = np.sqrt(6*(n - 2) / (n + 1) / (n + 3))

    return int(1 + np.log2(n) + np.log2(1 + skew/sigma_g1))

def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) creates a histogram from the 1D array
    `data with `bins` bins that ranges from [`low`, `high`).

    Returns a tuple (`hist`, `edges`), where `hist` contains the number of
    elements in each histogram bin and `edges` contains every bin edge.
    """
    assert(bins > 0)
    assert(high > low)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).astype(int)

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx <= bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges

def globular_cluster_distribution(Fe_H, bins):
    """ globular_cluster_distirbution(Fe_H, bins) plots the distirbution of
    globular cluster metalicities, `Fe_H` using `bins` bins.
    """
    assert(bins > 0)

    plt.figure()

    hist, edges = histogram(Fe_H, bins, -3.0, 1.0)
    centers = (edges[1:] + edges[:-1]) / 2

    plt.plot(centers, hist, "o", c="k")
    plt.xlabel(r"${\rm [Fe/H]}$")
    plt.ylabel(r"$N$")
```

This code up here is safe.

Last safe line of code is here.

```python
def main():
    Fe_H = np.loadtxt("globular_cluster_metalicity.txt")
    bins = optimal_bins(Fe_H)
    globular_cluster_distribution(Fe_H, bins)

def optimal_bins(x):
    """ optimal_bins(fname) computes the optimal number of bins for the
    dataset `x` using Doane's rule.
    """
    assert(len(x) > 2)

    skew = stats.skew(x)
    n = len(x)
    sigma_g1 = np.sqrt(6*(n - 2) / (n + 1) / (n + 3))

    return int(1 + np.log2(n) + np.log2(1 + skew/sigma_g1))

def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) creates a histogram from the 1D array
    `data with `bins` bins that ranges from [`low`, `high`).

    Returns a tuple (`hist`, `edges`), where `hist` contains the number of
    elements in each histogram bin and `edges` contains every bin edge.
    """
    assert(bins > 0)
    assert(high > low)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).astype(int)

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx <= bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges

def globular_cluster_distribution(Fe_H, bins):
    """ globular_cluster_distirbution(Fe_H, bins) plots the distirbution of
    globular cluster metalicities, `Fe_H` using `bins` bins.
    """
    assert(bins > 0)

    plt.figure()

    hist, edges = histogram(Fe_H, bins, -3.0, 1.0)
    centers = (edges[1:] + edges[:-1]) / 2

    plt.plot(centers, hist, "o", c="k")
    plt.xlabel(r"${\rm [Fe/H]}$")
    plt.ylabel(r"$N$")
```

This code up here is safe.

(If those inputs were invalid, we would have ruled out the bottom two functions instead and repeated the procedure with the calling function.)

Last safe line of code is here.

```python
def main():
    Fe_H = np.loadtxt("globular_cluster_metalicity.txt")
    bins = optimal_bins(Fe_H)
    globular_cluster_distribution(Fe_H, bins)

def optimal_bins(x):
    """ optimal_bins(fname) computes the optimal number of bins for the
    dataset `x` using Doane's rule.
    """
    assert(len(x) > 2)

    skew = stats.skew(x)
    n = len(x)
    sigma_g1 = np.sqrt(6*(n - 2) / (n + 1) / (n + 3)

    return int(1 + np.log2(n) + np.log2(1 + skew/sig

def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) creates a h
    `data with `bins` bins that ranges from [`low`,

    Returns a tuple (`hist`, `edges`), where `hist`
    elements in each histogram bin and `edges` conta
    """
    assert(bins > 0)
    assert(high > low)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).astype(int)

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx <= bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges

def globular_cluster_distribution(Fe_H, bins):
    """ globular_cluster_distirbution(Fe_H, bins) plots the distirbution of
    globular cluster metalicities, `Fe_H` using `bins` bins.
    """
    assert(bins > 0)

    plt.figure()

    hist, edges = histogram(Fe_H, bins, -3.0, 1.0)
    centers = (edges[1:] + edges[:-1]) / 2

    plt.plot(centers, hist, "o", c="k")
    plt.xlabel(r"${\rm [Fe/H]}$")
    plt.ylabel(r"$N$")
```

```python
def globular_cluster_distribution(Fe_H, bins):
    """ globular_cluster_distirbution(Fe_H, bins) plots the distirbution of
    globular cluster metalicities, `Fe_H` using `bins` bins.
    """
    assert(bins > 0)
    print("Fe_H:", Fe_H)
    print("bins:", bins)

    plt.figure()

    hist, edges = histogram(Fe_H, bins, -3.0, 1.0)
    centers = (edges[1:] + edges[:-1]) / 2

    plt.plot(centers, hist, "o", c="k")
    plt.xlabel(r"${\rm [Fe/H]}$")
    plt.ylabel(r"$N$")
```

We could go either way, but let's start at the crashing line and work our way back.

```python
def main():
    Fe_H = np.loadtxt("globular_cluster_metalicity.txt")
    bins = optimal_bins(Fe_H)
    globular_cluster_distribution(Fe_H, bins)

def optimal_bins(x):
    """ optimal_bins(fname) computes the optim
    dataset `x` using Doane's rule.
    """
    assert(len(x) > 2)

    skew = stats.skew(x)
    n = len(x)
    sigma_g1 = np.sqrt(6*(n - 2) / (n + 1) / (

    return int(1 + np.log2(n) + np.log2(1 + sk

def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) cre
    `data with `bins` bins that ranges from [

    Returns a tuple (`hist`, `edges`), where `
    elements in each histogram bin and `edges`
    """
    assert(bins > 0)
    assert(high > low)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).astype(i

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx <= bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges

def globular_cluster_distribution(Fe_H, bins):
    """ globular_cluster_distirbution(Fe_H, bins) plots the distirbution of
    globular cluster metalicities, `Fe_H` using `bins` bins.
    """
    assert(bins > 0)

    plt.figure()

    hist, edges = histogram(Fe_H, bins, -3.0, 1.0)
    centers = (edges[1:] + edges[:-1]) / 2

    plt.plot(centers, hist, "o", c="k")
    plt.xlabel(r"${\rm [Fe/H]}$")
    plt.ylabel(r"$N$")
```

```python
def globular_cluster_distribution(Fe_H, bins):
    """ globular_cluster_distirbution(Fe_H, bins) plots the distirbution of
    globular cluster metalicities, `Fe_H` using `bins` bins.
    """
    assert(bins > 0)
    print("Fe_H:", Fe_H)
    print("bins:", bins)

    plt.figure()

    hist, edges = histogram(Fe_H, bins, -3.0, 1.0)
    centers = (edges[1:] + edges[:-1]) / 2

    print("centers:", centers)
    print("hist:   ", hist)
    print("len(centers):", len(centers), "len(hist):", len(hist))
    plt.plot(centers, hist, "o", c="k")
    plt.xlabel(r"${\rm [Fe/H]}$")
    plt.ylabel(r"$N$")
```

We could go either way, but let's start at the crashing line and work our way back.

```python
def main():
    Fe_H = np.loadtxt("globular_cluster_metalicity.txt")
    bins = optimal_bins(Fe_H)
    globular_cluster_distribution(Fe_H, bins)

def optimal_bins(x):
    """ optimal_bins(fname) computes the optim
    dataset `x` using Doane's rule.
    """
    assert(len(x) > 2)

    skew = stats.skew(x)
    n = len(x)
    sigma_g1 = np.sqrt(6*(n - 2) / (n + 1) / (

    return int(1 + np.log2(n) + np.log2(1 + sk

def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) cre
    `data with `bins` bins that ranges from [

    Returns a tuple (`hist`, `edges`), where `
    elements in each histogram bin and `edges`
    """
    assert(bins > 0)
    assert(high > low)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).astype(i

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx <= bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges

def globular_cluster_distribution(Fe_H, bins):
    """ globular_cluster_distirbution(Fe_H, bins) plots the distirbution of
    globular cluster metalicities, `Fe_H` using `bins` bins.
    """
    assert(bins > 0)

    plt.figure()

    hist, edges = histogram(Fe_H, bins, -3.0, 1.0)
    centers = (edges[1:] + edges[:-1]) / 2

    plt.plot(centers, hist, "o", c="k")
    plt.xlabel(r"${\rm [Fe/H]}$")
    plt.ylabel(r"$N$")
```

```python
def globular_cluster_distribution(Fe_H, bins):
    """ globular_cluster_distirbution(Fe_H, bins) plots the distirbution of
    globular cluster metalicities, `Fe_H` using `bins` bins.
    """
    assert(bins > 0)
    print("Fe_H:", Fe_H)
    print("bins:", bins)

    plt.figure()

    hist, edges = histogram(Fe_H, bins, -3.0, 1.0)
    centers = (edges[1:] + edges[:-1]) / 2

    print("centers:", centers)
    print("hist:   ", hist)
    print("len(centers):", len(centers), "len(hist):", len(hist))
    plt.plot(centers, hist, "o", c="k")
    plt.xlabel(r"${\rm [Fe/H]}$")
    plt.ylabel(r"$N$")
```

We could go either way, but let's start at the crashing line and work our way back.

```python
def main():
    Fe_H = np.loadtxt("globular_cluster_metalicity.txt")
    bins = optimal_bins(Fe_H)
    globular_cluster_distribution(Fe_H, bins)

def optimal_bins(x):
    """ optimal_bins(fname) computes the optim
    dataset `x` using Doane's rule.
    """
    assert(len(x) > 2)

    skew = stats.skew(x)
    n = len(x)
    sigma_g1 = np.sqrt(6*(n - 2) / (n + 1) / (

    return int(1 + np.log2(n) + np.log2(1 + sk

def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) creat
    `data with `bins` bins that ranges from [`

    Returns a tuple (`hist`, `edges`), where `
    elements in each histogram bin and `edges`
    """
    assert(bins > 0)
    assert(high > low)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).astype(i

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx <= bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges

def globular_cluster_distribution(Fe_H, bins):
    """ globular_cluster_distirbution(Fe_H, bi
    globular cluster metalicities, `Fe_H` usin
    """
    assert(bins > 0)

    plt.figure()

    hist, edges = histogram(Fe_H, bins, -3.0, 1.0)
    centers = (edges[1:] + edges[:-1]) / 2

    plt.plot(centers, hist, "o", c="k")
    plt.ylabel(r"$N$")
```

```python
def globular_cluster_distribution(Fe_H, bins):
    """ globular_cluster_distirbution(Fe_H, bins) plots the distirbution of
    globular cluster metalicities, `Fe_H` using `bins` bins.
    """

    assert(bins > 0)
    print("Fe_H:", Fe_H)
```

```
levi-civita:scratch phil$ python debugging_examples.py
Fe_H: [-3.6401761  -2.62104217 -4.59689928 -4.50583312 -4.62252371  3.10865721
 -2.76378391 -2.79293035 -1.38716174  0.82455294 -7.85408701 -2.29753269
 -3.31201677 -3.75630165 -4.01417183 -5.23256656 -2.45843828 -2.1861628
 -2.78822755 -1.69699626 -1.63649125 -4.16953641 -2.89774636 -0.08939127
 -2.04364833 -0.86384391 -1.62295559  1.92094566 -0.08021028 -0.47251922
 -4.51679014 -4.11206368 -2.09035686 -0.83837957  2.43439993 -1.89542316
 -1.42353272 -4.79614936 -2.85842067  0.41540516 -2.79320033 -1.13293636
 -3.34792346 -2.65280793 -3.43679299 -0.61708158  2.93588617 -3.65491541
 -1.94661883 -4.19282036]
bins: 7
centers: [ -2.66666667e+00  -2.00000000e+00  -1.33333333e+00  -6.66666667e-01
  -2.22044605e-16   6.66666667e-01]
hist:    [9 6 5 3 2 3 1]
len(centers): 6 len(hist): 7
```

hist looks fine, but centers has one less element than the input bin number!

```python
def main():
    Fe_H = np.loadtxt("globular_cluster_metalicity.txt")
    bins = optimal_bins(Fe_H)
    globular_cluster_distribution(Fe_H, bins)

def optimal_bins(x):
    """ optimal_bins(fname) computes the optim
    dataset `x` using Doane's rule.
    """
    assert(len(x) > 2)

    skew = stats.skew(x)
    n = len(x)
    sigma_g1 = np.sqrt(6*(n - 2) / (n + 1) / (

    return int(1 + np.log2(n) + np.log2(1 + sk

def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) cre
    `data with `bins` bins that ranges from [

    Returns a tuple (`hist`, `edges`), where `
    elements in each histogram bin and `edges`
    """
    assert(bins > 0)
    assert(high > low)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).astype(i

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx <= bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges

def globular_cluster_distribution(Fe_H, bins):
    """ globular_cluster_distirbution(Fe_H, bins) plots the distirbution of
    globular cluster metalicities, `Fe_H` using `bins` bins.
    """
    assert(bins > 0)

    plt.figure()

    hist, edges = histogram(Fe_H, bins, -3.0, 1.0)
    centers = (edges[1:] + edges[:-1]) / 2

    plt.plot(centers, hist, "o", c="k")
    plt.xlabel(r"${\rm [Fe/H]}$")
    plt.ylabel(r"$N$")
```

```python
def globular_cluster_distribution(Fe_H, bins):
    """ globular_cluster_distirbution(Fe_H, bins) plots the distirbution of
    globular cluster metalicities, `Fe_H` using `bins` bins.
    """
    assert(bins > 0)
    print("Fe_H:", Fe_H)
    print("bins:", bins)

    plt.figure()

    hist, edges = histogram(Fe_H, bins, -3.0, 1.0)
    centers = (edges[1:] + edges[:-1]) / 2

    print("centers:", centers)
    print("hist:   ", hist)
    print("len(centers):", len(centers), "len(hist):", len(hist))
    plt.plot(centers, hist, "o", c="k")
    plt.xlabel(r"${\rm [Fe/H]}$")
    plt.ylabel(r"$N$")
```

centers comes directly from edges, so edges must have length 7 instead of 8. The problem must have happened before histogram returns.

```
def main():
    Fe_H = np.loadtxt("globular_cluster_metalicity.txt")
    bins = optimal_bins(Fe_H)
    globular_cluster_distribution(Fe_H, bins)

def optimal_bins(x):
    """ optimal_bins(fname) computes the optim
    dataset `x` using Doane's rule.
    """
    assert(len(x) > 2)

    skew = stats.skew(x)
    n = len(x)
    sigma_g1 = np.sqrt(6*(n - 2) / (n + 1) / (

    return int(1 + np.log2(n) + np.log2(1 + sk

def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) cre
    `data with `bins` bins that ranges from [

    Returns a tuple (`hist`, `edges`), where
    elements in each histogram bin and `edges
    """
    assert(bins > 0)
    assert(high > low)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).astype(i

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx <= bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges
```

```
def globular_cluster_distribution(Fe_H, bins):
    """ globular_cluster_distirbution(Fe_H, bins) plots the distirbution of
    globular cluster metalicities, `Fe_H` using `bins` bins.
    """
    assert(bins > 0)
    print("Fe_H:", Fe_H)
    print("bins:", bins)

    plt.figure()

    hist, edges = histogram(Fe_H, bins, -3.0, 1.0)
    centers = (edges[1:] + edges[:-1]) / 2

    print("centers:", centers)
    print("hist:   ", hist)
    print("len(centers):", len(centers), "len(hist):", len(hist))
    plt.plot(centers, hist, "o", c="k")
    plt.xlabel(r"${\rm [Fe/H]}$")
    plt.ylabel(r"$N$")
```

```
def globular_cluster_distribution(Fe_H, bins):
    """ globular_cluster_distirbution(Fe_H, bins) plots the distirbution of
    globular cluster metalicities, `Fe_H` using `bins` bins.
    """
    assert(bins > 0)

    plt.figure()

    hist, edges = histogram(Fe_H, bins, -3.0, 1.0)
    centers = (edges[1:] + edges[:-1]) / 2

    plt.plot(centers, hist, "o", c="k")
    plt.xlabel(r"${\rm [Fe/H]}$")
    plt.ylabel(r"$N$")
```

If we know what the issue is at this point, we can just go fix it, but if not we'll jump into histogram and try to figure it out.

```python
def main():
    Fe_H = np.loadtxt("globular_cluster_metalicity.txt")
    bins = optimal_bins(Fe_H)
    globular_cluster_distribution(Fe_H, bi

def optimal_bins(x):
    """ optimal_bins(fname) computes the o
    dataset `x` using Doane's rule.
    """
    assert(len(x) > 2)

    skew = stats.skew(x)
    n = len(x)
    sigma_g1 = np.sqrt(6*(n - 2) / (n + 1)

    return int(1 + np.log2(n) + np.log2(1

def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) c
    `data with `bins` bins that ranges fro

    Returns a tuple (`hist`, `edges`), whe
    elements in each histogram bin and `ed
    """
    assert(bins > 0)
    assert(high > low)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).asty

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx <= bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges

def globular_cluster_distribution(Fe_H, bi
    """ globular_cluster_distirbution(Fe_H
    globular cluster metalicities, `Fe_H`
    """
    assert(bins > 0)

    plt.figure()

    hist, edges = histogram(Fe_H, bins, -3.0, 1.0)
    centers = (edges[1:] + edges[:-1]) / 2

    plt.plot(centers, hist, "o", c="k")
    plt.xlabel(r"${\rm [Fe/H]}$")
    plt.ylabel(r"$N$")
```

```python
def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) creates a histogram from the 1D array
    `data with `bins` bins that ranges from [`low`, `high`).

    Returns a tuple (`hist`, `edges`), where `hist` contains the number of
    elements in each histogram bin and `edges` contains every bin edge.
    """
    assert(bins > 0)
    assert(high > low)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).astype(int)

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx < bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges
```

First, check the inputs.

```python
def main():
    Fe_H = np.loadtxt("globular_cluster_meta
    bins = optimal_bins(Fe_H)
    globular_cluster_distribution(Fe_H, bins

def optimal_bins(x):
    """ optimal_bins(fname) computes the opt
    dataset `x` using Doane's rule.
    """
    assert(len(x) > 2)

    skew = stats.skew(x)
    n = len(x)
    sigma_g1 = np.sqrt(6*(n - 2) / (n + 1) /

    return int(1 + np.log2(n) + np.log2(1 +

def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) cre
    `data with `bins` bins that ranges from

    Returns a tuple (`hist`, `edges`), where
    elements in each histogram bin and `edge
    """
    assert(bins > 0)
    assert(high > low)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).astype

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx <= bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges

def globular_cluster_distribution(Fe_H, bins
    """ globular_cluster_distirbution(Fe_H,
    globular cluster metalicities, `Fe_H` us
    """
    assert(bins > 0)

    plt.figure()

    hist, edges = histogram(Fe_H, bins, -3.0
    centers = (edges[1:] + edges[:-1]) / 2

    plt.plot(centers, hist, "o", c="k")
    plt.xlabel(r"${\rm [Fe/H]}$")
    plt.ylabel(r"$N$")
```

```python
def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) creates a histogram from the 1D array
    `data with `bins` bins that ranges from [`low`, `high`).

    Returns a tuple (`hist`, `edges`), where `hist` contains the number of
    elements in each histogram bin and `edges` contains every bin edge.
    """
    assert(bins > 0)
    assert(high > low)

    print("data:", data)
    print("bins:", bins)
    print("low: ", low)
    print("high:", high)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).astype(int)

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx < bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges
```

```
def histogram(data, bins, low, high):
```

```
levi-civita:scratch phil$ python debugging_examples.py
Fe_H: [-3.6401761  -2.62104217 -4.59689928 -4.50583312 -4.62252371  3.10865721
 -2.76378391 -2.79293035 -1.38716174  0.82455294 -7.85408701 -2.29753269
 -3.31201677 -3.75630165 -4.01417183 -5.23256656 -2.45843828 -2.1861628
 -2.78822755 -1.69699626 -1.63649125 -4.16953641 -2.89774636 -0.08939127
 -2.04364833 -0.86384391 -1.62295559  1.92094566 -0.08021028 -0.47251922
 -4.51679014 -4.11206368 -2.09035686 -0.83837957  2.43439993 -1.89542316
 -1.42353272 -4.79614936 -2.85842067  0.41540516 -2.79320033 -1.13293636
 -3.34792346 -2.65280793 -3.43679299 -0.61708158  2.93588617 -3.65491541
 -1.94661883 -4.19282036]
bins: 7
data: [-3.6401761  -2.62104217 -4.59689928 -4.50583312 -4.62252371  3.10865721
 -2.76378391 -2.79293035 -1.38716174  0.82455294 -7.85408701 -2.29753269
 -3.31201677 -3.75630165 -4.01417183 -5.23256656 -2.45843828 -2.1861628
 -2.78822755 -1.69699626 -1.63649125 -4.16953641 -2.89774636 -0.08939127
 -2.04364833 -0.86384391 -1.62295559  1.92094566 -0.08021028 -0.47251922
 -4.51679014 -4.11206368 -2.09035686 -0.83837957  2.43439993 -1.89542316
 -1.42353272 -4.79614936 -2.85842067  0.41540516 -2.79320033 -1.13293636
 -3.34792346 -2.65280793 -3.43679299 -0.61708158  2.93588617 -3.65491541
 -1.94661883 -4.19282036]
bins: 7
low:  -3.0
high: 1.0
centers: [ -2.66666667e+00  -2.00000000e+00  -1.33333333e+00  -6.66666667e-01
  -2.22044605e-16   6.66666667e-01]
hist:    [9 6 5 3 2 3 1]
len(centers): 6 len(hist): 7
```

(Whew, these outputs are starting to get big: good thing we annotated each value!)

```
def main():
    Fe_H = np.loadtxt("globular_cluster_meta
    bins = optimal_bins(Fe_H)
    globular_clu

def optimal_bins
    """ optimal_
    dataset `x`
    """
    assert(len(x

    skew = stats
    n = len(x)
    sigma_g1 = n

    return int(1

def histogram(da
    """ histogra
    `data with `

    Returns a tu
    elements in
    """
    assert(bins
    assert(high

    bin_width =
    scaled_data
    bin_index =

    hist = np.ze
    for idx in b
        if idx >
            hist

    edges = np.l

    return hist,

def globular_clu
    """ globular
    globular clu
    """
    assert(bins

    plt.figure()

    hist, edges
    centers = (e

    plt.plot(cen
    plt.xlabel(r
    plt.ylabel(r"$N$")
```

```
def histogram(data, bins, low, high):
    m the 1D array

    number of
    n edge.
```

```
levi-civita:scratch phil$ python debugging_examples.py
Fe_H: [-3.6401761  -2.62104217 -4.59689928 -4.50583312 -4.62252371  3.10865721
 -2.76378391 -2.79293035 -1.38716174  0.82455294 -7.85408701 -2.29753269
 -3.31201677 -3.75630165 -4.01417183 -5.23256656 -2.45843828 -2.1861628
 -2.78822755 -1.69699626 -1.63649125 -4.16953641 -2.89774636 -0.08939127
 -2.04364833 -0.86384391 -1.62295559  1.92094566 -0.08021028 -0.47251922
 -4.51679014 -4.11206368 -2.09035686 -0.83837957  2.43439993 -1.89542316
 -1.42353272 -4.79614936 -2.85842067  0.41540516 -2.79320033 -1.13293636
 -3.34792346 -2.65280793 -3.43679299 -0.61708158  2.93588617 -3.65491541
 -1.94661883 -4.19282036]
bins: 7
data: [-3.6401761  -2.62104217 -4.59689928 -4.50583312 -4.62252371  3.10865721
 -2.76378391 -2.79293035 -1.38716174  0.82455294 -7.85408701 -2.29753269
 -3.31201677 -3.75630165 -4.01417183 -5.23256656 -2.45843828 -2.1861628
 -2.78822755 -1.69699626 -1.63649125 -4.16953641 -2.89774636 -0.08939127
 -2.04364833 -0.86384391 -1.62295559  1.92094566 -0.08021028 -0.47251922
 -4.51679014 -4.11206368 -2.09035686 -0.83837957  2.43439993 -1.89542316
 -1.42353272 -4.79614936 -2.85842067  0.41540516 -2.79320033 -1.13293636
 -3.34792346 -2.65280793 -3.43679299 -0.61708158  2.93588617 -3.65491541
 -1.94661883 -4.19282036]
bins: 7
low:  -3.0
high: 1.0
centers: [ -2.66666667e+00  -2.00000000e+00  -1.33333333e+00  -6.66666667e-01
  -2.22044605e-16   6.66666667e-01]
hist:    [9 6 5 3 2 3 1]
len(centers): 6 len(hist): 7
```

data look valid. bins looks valid. low looks valid. high looks valid. Error is in this function!

```python
def main():
    Fe_H = np.loadtxt("globular_cluster_metalicity.txt")
    bins = optimal_bins(Fe_H)
    globular_cluster_distribution(Fe_H, bins)

def optimal_bins(x):
    """ optimal_bins(fname) computes the optimal number of bins for the
    dataset `x` using Doane's rule.
    """
    assert(len(x) > 2)

    skew = stats.skew(x)
    n = len(x)
    sigma_g1 = np.sqrt(6*(n - 2) / (n + 1) / (n + 3))

    return int(1 + np.log2(n) + np.log2(1 + skew/sigma_g1))

def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) creates a histogram from the 1D array
    `data with `bins` bins that ranges from [`low`, `high`).

    Returns a tuple (`hist`, `edges`), where `hist` contains the number of
    elements in each histogram bin and `edges` contains every bin edge.
    """
    assert(bins > 0)
    assert(high > low)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).astype(int)

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx <= bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges

def globular_cluster_distribution(Fe_H, bins):
    """ globular_cluster_distirbution(Fe_H, bins) plots the distirbution of
    globular cluster metalicities, `Fe_H` using `bins` bins.
    """
    assert(bins > 0)

    plt.figure()

    hist, edges = histogram(Fe_H, bins, -3.0, 1.0)
    centers = (edges[1:] + edges[:-1]) / 2

    plt.plot(centers, hist, "o", c="k")
    plt.xlabel(r"${\rm [Fe/H]}$")
    plt.ylabel(r"$N$")
```

```
def main():
    Fe_H = np.loadtxt("globular_cluster_meta
    bins = optimal_bins(Fe_H)
    globular_cluster_distribution(Fe_H, bins

def optimal_bins(x):
    """ optimal_bins(fname) computes the opt
    dataset `x` using Doane's rule.
    """
    assert(len(x) > 2)

    skew = stats.skew(x)
    n = len(x)
    sigma_g1 = np.sqrt(6*(n - 2) / (n + 1) /

    return int(1 + np.log2(n) + np.log2(1 +

def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) cre
    `data with `bins` bins that ranges from

    Returns a tuple (`hist`, `edges`), where
    elements in each histogram bin and `edge
    """
    assert(bins > 0)
    assert(high > low)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).asty

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx <= bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges

def globular_cluster_distribution(Fe_H, bins
    """ globular_cluster_distirbution(Fe_H,
    globular cluster metalicities, `Fe_H` us
    """
    assert(bins > 0)

    plt.figure()

    hist, edges = histogram(Fe_H, bins, -3.0
    centers = (edges[1:] + edges[:-1]) / 2

    plt.plot(centers, hist, "o", c="k")
    plt.xlabel(r"${\rm [Fe/H]}$")
    plt.ylabel(r"$N$")
```

```
def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) creates a histogram from the 1D array
    `data with `bins` bins that ranges from [`low`, `high`).

    Returns a tuple (`hist`, `edges`), where `hist` contains the number of
    elements in each histogram bin and `edges` contains every bin edge.
    """
    assert(bins > 0)
    assert(high > low)

    print("data:", data)
    print("bins:", bins)
    print("low: ", low)
    print("high:", high)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).astype(int)

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx < bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges
```

```python
def main():
    Fe_H = np.loadtxt("globular_cluster_meta
    bins = optimal_bins(Fe_H)
    globular_cluster_distribution(Fe_H, bins

def optimal_bins(x):
    """ optimal_bins(fname) computes the opt
    dataset `x` using Doane's rule.
    """
    assert(len(x) > 2)

    skew = stats.skew(x)
    n = len(x)
    sigma_g1 = np.sqrt(6*(n - 2) / (n + 1) /

    return int(1 + np.log2(n) + np.log2(1 +

def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) cre
    `data with `bins` bins that ranges from

    Returns a tuple (`hist`, `edges`), where
    elements in each histogram bin and `edge
    """
    assert(bins > 0)
    assert(high > low)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).asty

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx <= bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges

def globular_cluster_distribution(Fe_H, bins
    """ globular_cluster_distirbution(Fe_H,
    globular cluster metalicities, `Fe_H` us
    """
    assert(bins > 0)

    plt.figure()

    hist, edges = histogram(Fe_H, bins, -3.0
    centers = (edges[1:] + edges[:-1]) / 2

    plt.plot(centers, hist, "o", c="k")
    plt.xlabel(r"${\rm [Fe/H]}$")
    plt.ylabel(r"$N$")
```

```python
def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) creates a histogram from the 1D array
    `data with `bins` bins that ranges from [`low`, `high`).

    Returns a tuple (`hist
    elements in each histo
    """
    assert(bins > 0)
    assert(high > low)

    print("data:", data)
    print("bins:", bins)
    print("low: ", low)
    print("high:", high)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).astype(int)

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx < bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges
```

We know the issue is with edges. First, find where edges is created.

```
def main():
    Fe_H = np.loadtxt("globular_cluster_meta
    bins = optimal_bins(Fe_H)
    globular_cluster_distribution(Fe_H, bins

def optimal_bins(x):
    """ optimal_bins(fname) computes the opt
    dataset `x` using Doane's rule.
    """
    assert(len(x) > 2)

    skew = stats.skew(x)
    n = len(x)
    sigma_g1 = np.sqrt(6*(n - 2) / (n + 1) /

    return int(1 + np.log2(n) + np.log2(1 +

def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) cre
    `data with `bins` bins that ranges from

    Returns a tuple (`hist`, `edges`), where
    elements in each histogram bin and `edge
    """
    assert(bins > 0)
    assert(high > low)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).asty

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx <= bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges

def globular_cluster_distribution(Fe_H, bins
    """ globular_cluster_distirbution(Fe_H,
    globular cluster metalicities, `Fe_H` us
    """
    assert(bins > 0)

    plt.figure()

    hist, edges = histogram(Fe_H, bins, -3.0
    centers = (edges[1:] + edges[:-1]) / 2

    plt.plot(centers, hist, "o", c="k")
    plt.xlabel(r"${\rm [Fe/H]}$")
    plt.ylabel(r"$N$")
```

```
def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) creates a histogram from the 1D array
    `data with `bins` bins that ranges from [`low`, `high`).

    Returns a tuple (`hist
    elements in each histo
    """
    assert(bins > 0)
    assert(high > low)

    print("data:", data)
    print("bins:", bins)
    print("low: ", low)
    print("high:", high)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).astype(int)

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx < bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges
```

Next, look at its dependencies: high, low, and bins. We already know those are valid.

```python
def main():
    Fe_H = np.loadtxt("globular_cluster_meta
    bins = optimal_bins(Fe_H)
    globular_cluster_distribution(Fe_H, bins

def optimal_bins(x):
    """ optimal_bins(fname) computes the opt
    dataset `x` using Doane's rule.
    """
    assert(len(x) > 2)

    skew = stats.skew(x)
    n = len(x)
    sigma_g1 = np.sqrt(6*(n - 2) / (n + 1) /

    return int(1 + np.log2(n) + np.log2(1 +

def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) cre
    `data with `bins` bins that ranges from

    Returns a tuple (`hist`, `edges`), where
    elements in each histogram bin and `edge
    """
    assert(bins > 0)
    assert(high > low)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).astype

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx <= bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges

def globular_cluster_distribution(Fe_H, bins
    """ globular_cluster_distirbution(Fe_H,
    globular cluster metalicities, `Fe_H` us
    """
    assert(bins > 0)

    plt.figure()

    hist, edges = histogram(Fe_H, bins, -3.0
    centers = (edges[1:] + edges[:-1]) / 2

    plt.plot(centers, hist, "o", c="k")
    plt.xlabel(r"${\rm [Fe/H]}$")
    plt.ylabel(r"$N$")
```

```python
def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) creates a histogram from the 1D array
    `data with `bins` bins that ranges from [`low`, `high`).

    Returns a tuple (`hist`, `edges`), where `hist` contains the number of
    elements in each histogram bin and `edges` contains every bin edge.
    """
    assert(bins > 0)
    assert(high > low)

    print("data:", data)
    print("bins:", bins)
    print("low: ", low)
    print("high:", high)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).astype(int)

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx < bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges
```

So the error is on this line.

```
def main():
    Fe_H = np.loadtxt("globular_cluster_meta
    bins = optimal_bins(Fe_H)
    globular_cluster_distribution(Fe_H, bins

def optimal_bins(x):
    """ optimal_bins(fname) computes the opt
    dataset `x` using Doane's rule.
    """
    assert(len(x) > 2)

    skew = stats.skew(x)
    n = len(x)
    sigma_g1 = np.sqrt(6*(n - 2) / (n + 1) /

    return int(1 + np.log2(n) + np.log2(1 +

def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) cre
    `data with `bins` bins that ranges from

    Returns a tuple (`hist`, `edges`), where
    elements in each histogram bin and `edge
    """
    assert(bins > 0)
    assert(high > low)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).astype

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx <= bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges

def globular_cluster_distribution(Fe_H, bins
    """ globular_cluster_distirbution(Fe_H,
    globular cluster metalicities, `Fe_H` us
    """
    assert(bins > 0)

    plt.figure()

    hist, edges = histogram(Fe_H, bins, -3.0
    centers = (edges[1:] + edges[:-1]) / 2

    plt.plot(centers, hist, "o", c="k")
    plt.xlabel(r"${\rm [Fe/H]}$")
    plt.ylabel(r"$N$")
```

```
def histogram(data, bins, low, high):
    """ histogram(data, bins, low, high) creates a histogram from the 1D array
    `data with `bins` bins that ranges from [`low`, `high`).

    Returns a tuple (`hist`, `edges`), where `hist` contains the number of
    elements in each histogram bin and `edges` contains every bin edge.
    """
    assert(bins > 0)
    assert(high > low)

    print("data:", data)
    print("bins:", bins)
    print("low: ", low)
    print("high:", high)

    bin_width = (high - low)/bins
    scaled_data = (data - low)/bin_width
    bin_index = np.floor(scaled_data).astype(int)

    hist = np.zeros(bins, dtype=int)
    for idx in bin_index:
        if idx >= 0 and idx < bins:
            hist[idx] += 1

    edges = np.linspace(low, high, bins)

    return hist, edges
```

Oh, I see! I wrote bins instead of bins+1!

Let’s review the bug-finding process

1. Your goal is always to shrink the number of lines of possibly wrong code.
2. Obey the holy debugging order:
    a. Check the line that crashed or the test that failed.
    b. Check that the function inputs are valid.
    c. Check the code between the points.
    d. Do the same for any functions this takes you do.
3. You can disobey the holy debugging order if you’re ever pretty sure you know what the error is: just go there.
4. Print data out and annotate what that data is.
5. There ~~might~~ will be multiple bugs at the same time.
    a. (There was a second one in that example code. We’d have to fix that next.)
6. Delete all your print statements afterwards.

Advanced bug-finding tips: printing

- To make printed output less confusing, learn how to use Python's "string formatting."
  - There are two versions of this. One uses the % operator and works like C.
  - The newer version uses the string.format() method.
  - Look at np.array2string() and np.set_printoptions() for numpy arrays.
- You can put debugging output directly into a text file from the command line with "$ python my_program.py > my_text_file.txt".
  - This makes it easier to search and it won't go away when you close your terminal.
- If you're printing in a giant for loop, use the integer modulo operation to print things out every thousand/million/etc. loops.
- Delete all your debugging statements before going to sleep.
  - They didn't work for you today and you'll forget what they mean anyway: start from scratch tomorrow.

Advanced bug-finding tips: debuggers

- An alternative to print statements are programs called “debuggers” which let you navigate the code “directly” without print statements.
- The most popular two are pdb and pudb
  - https://docs.python.org/3/library/pdb.html
  - https://pypi.org/project/pudb/
- Your favorite IDE may also have one built in.

I do not like debuggers and will not teach anyone to use them. But some programmers prefer using them over print statements, or use them as an emergency measure.

Advanced bug-finding tips: miscellaneous

- If your code used to work but doesn't work now, use "git diff" to find the changed lines of code and focus on those.
  - This is a good reason to commit frequently.
  - If you aren't using version control yet, now is a good time to start building that skill!
  - This is a good tutorial: https://www.atlassian.com/git/tutorials/what-is-version-control
- If an error only shows up on a complicated test with lots of data, try to make a smaller test which fails in the same way (more on this later).
- Make sure other tests don't fail after you've made your debugging fix.
- After you've fixed something, ask yourself if you did it anywhere else instead of just going on your merry way.

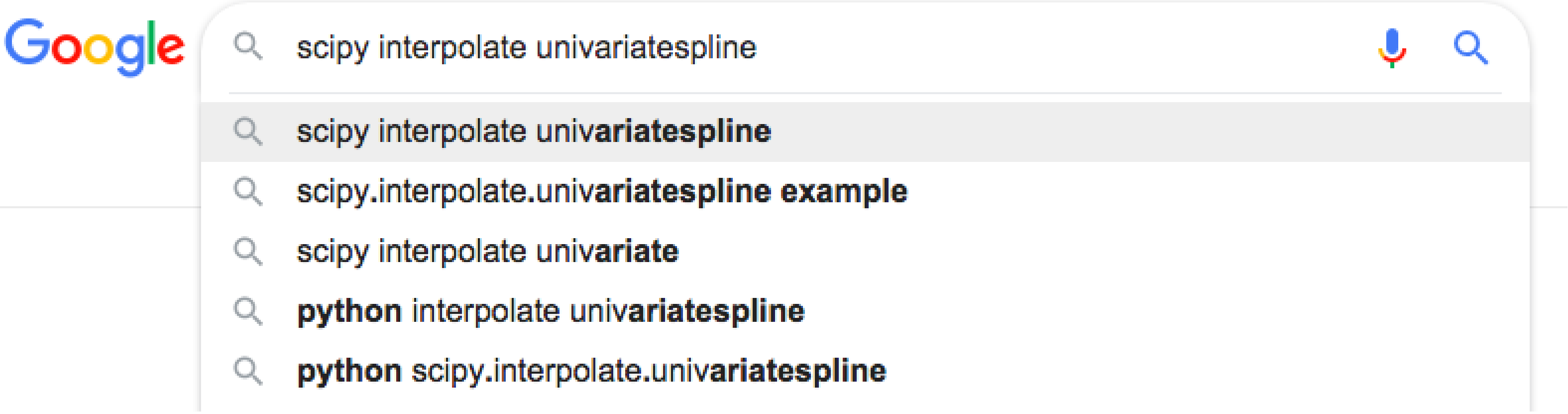
Telling Right from Wrong

Deciding whether a block of code is wrong or whether a particular variable is valid can be hard. There are a four stages to this:

1. Careful line-reading
2. Research
3. Minimal failing examples
4. Asking for help

Research

1. Always, always read the documentation for the function first.

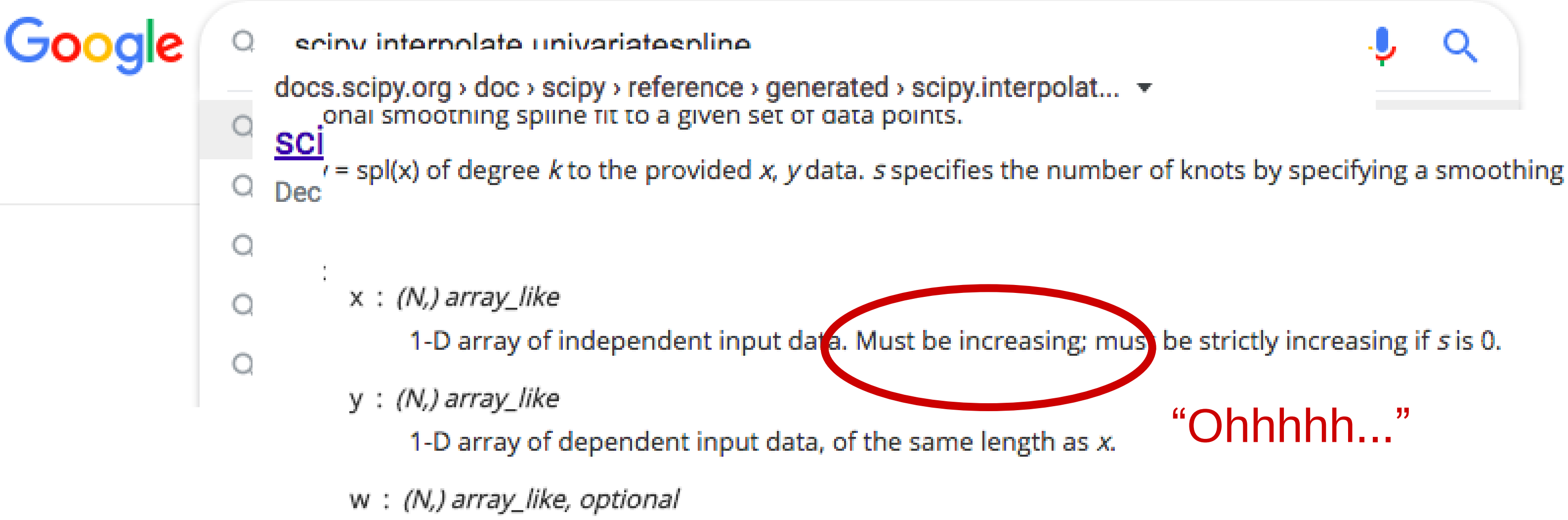


Research

1. Always, always read the documentation for the function first.

Research

1. Always, always read the documentation for the function first.



“Ohhhhh...”

Research

1. Always, always read the documentation for the function first.
2. If the documentation is confusing, search for an example instead trying to fight through it.

Ack, WTF

numpy.choose(*a, choices, out=None, mode='raise'*) [source]

Construct an array from an index array and a set of arrays to choose from.

First of all, if confused or uncertain, definitely look at the Examples - in its full generality, this function is less simple than it might seem from the following code description (below ndi = numpy.lib.index_tricks):

`np.choose(a,c) == np.array([c[a[I]][I] for I in ndi.ndindex(a.shape)])`.

But this omits some subtleties. Here is a fully general summary:

Given an "index" array (*a*) of integers and a sequence of *n* arrays (*choices*), *a* and each choice array are first broadcast, as necessary, to arrays of a common shape; calling these *Ba* and *Bchoices[i], i = 0,...,n-1* we have that, necessarily, `Ba.shape == Bchoices[i].shape` for each *i*. Then, a new array with shape `Ba.shape` is created as follows:

- if `mode=raise` (the default), then, first of all, each element of *a* (and thus *Ba*) must be in the range *[0, n-1]*; now, suppose that *i* (in that range) is the value at the *(j0, j1, ..., jm)* position in *Ba* - then the value at the same position in the new array is the value in *Bchoices[i]* at that same position;
- if `mode=wrap`, values in *a* (and thus *Ba*) may be any (signed) integer; modular arithmetic is used to map integers out-

Research

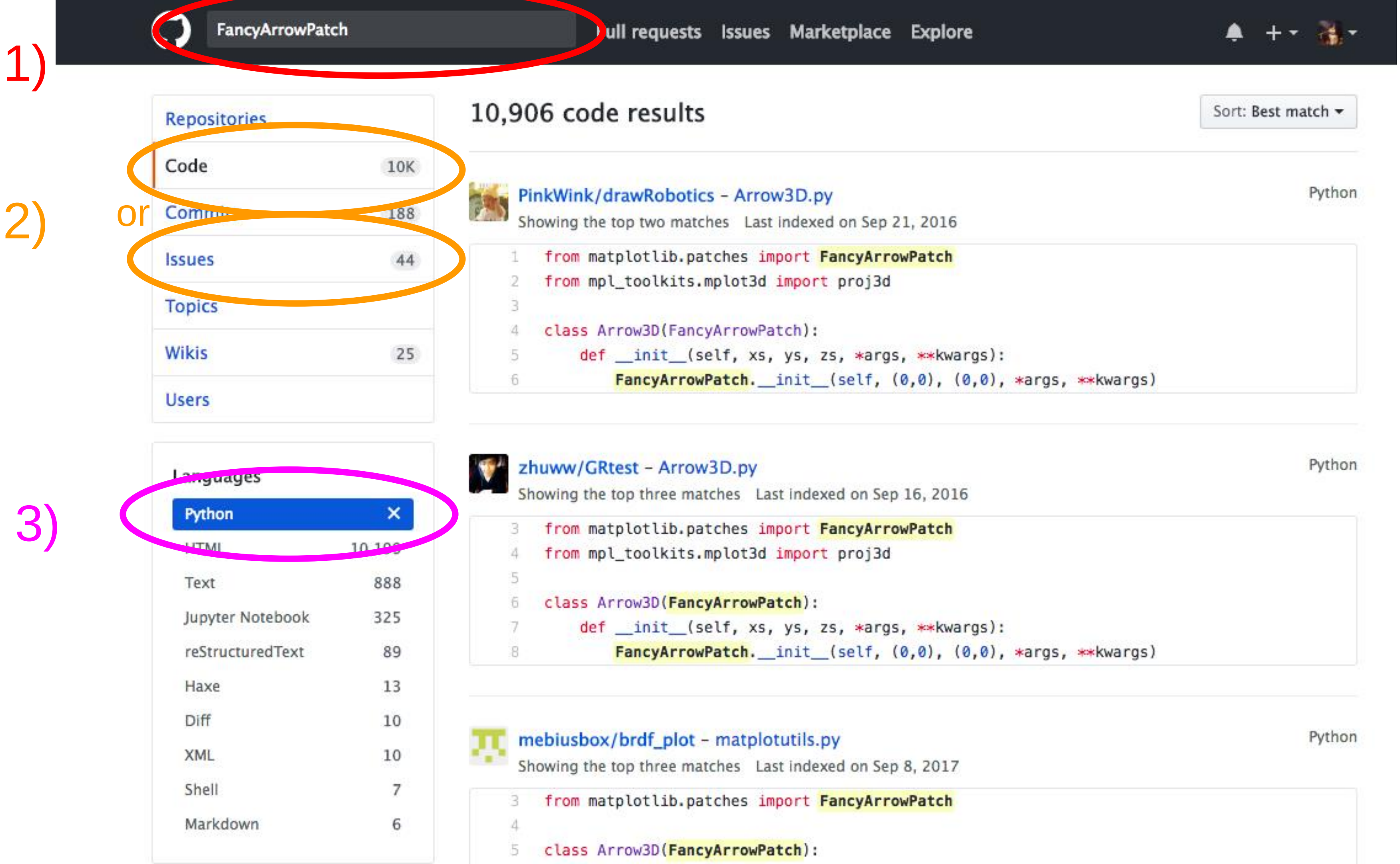
1. Always, always read the documentation for the function first.
2. If the documentation is confusing, search for an example instead trying to fight through it.

Ahhh...

```
>>> a = [[1, 0, 1], [0, 1, 0], [1, 0, 1]]
>>> choices = [-10, 10]
>>> np.choose(a, choices)
array([[ 10, -10,  10],
       [-10,  10, -10],
       [ 10, -10,  10]])
```

Research

1. Always, always read the documentation for the function first.
2. If the documentation is confusing, search for an example instead trying to fight through it.
    a. You can find examples on the official website, but searching “library function_name example” on Google, “library function_name stack overflow,” or by searching github.


1)
FancyArrowPatch
Pull requests
Issues
Marketplace
Explore
10,906 code results
Sort: Best match
Repositories
Code 10K
2)
or
Commits 188
Issues 44
Topics
Wikis 25
Users
Languages
3)
Python
HTML 10,109
Text 888
Jupyter Notebook 325
reStructuredText 89
Haxe 13
Diff 10
XML 10
Shell 7
Markdown 6
PinkWink/drawRobotics – Arrow3D.py
Python
Showing the top two matches Last indexed on Sep 21, 2016
from matplotlib.patches import FancyArrowPatch
from mpl_toolkits.mplot3d import proj3d
class Arrow3D(FancyArrowPatch):
def __init__(self, xs, ys, zs, *args, **kwargs):
FancyArrowPatch.__init__(self, (0,0), (0,0), *args, **kwargs)
zhuww/GRtest – Arrow3D.py
Python
Showing the top three matches Last indexed on Sep 16, 2016
from matplotlib.patches import FancyArrowPatch
from mpl_toolkits.mplot3d import proj3d
class Arrow3D(FancyArrowPatch):
def __init__(self, xs, ys, zs, *args, **kwargs):
FancyArrowPatch.__init__(self, (0,0), (0,0), *args, **kwargs)
mebiusbox/brdf_plot – matplotutils.py
Python
Showing the top three matches Last indexed on Sep 8, 2017
from matplotlib.patches import FancyArrowPatch
class Arrow3D(FancyArrowPatch):

Research

1. Always, always read the documentation for the function first.
2. If the documentation is confusing, search for an example instead trying to fight through it.
    a. You can find examples on the official website, but searching “library function_name example” on Google, “library function_name stack overflow,” or by searching github.
    b. Search for other examples in your own codebase. Use ‘$grep “function_name” file1.pt file2.py …’ to search quickly across many files (or use an IDE).

Research

1. Always, always read the documentation for the function first.
2. If the documentation is confusing, search for an example instead trying to fight through it.
   a. You can find examples on the official website, but searching “library function_name example” on Google, “library function_name stack overflow,” or by searching github.
   b. Search for other examples in your own codebase. Use ‘$grep “function_name” file1.pt file2.py …’ to search quickly across many files (or use an IDE).
3. Make sure you understand the numerical details of what you’re doing

(Use this guy’s notebooks)

Research

1. Always, always read the documentation for the function first.
2. If the documentation is confusing, search for an example instead trying to fight through it.
    a. You can find examples on the official website, but searching “library function_name example” on Google, “library function_name stack overflow,” or by searching github.
    b. Search for other examples in your own codebase. Use ‘$grep “function_name” file1.pt file2.py …’ to search quickly across many files (or use an IDE).
3. Make sure you understand the numerical details of what you’re doing
4. Google the error message.
    a. Remove any numbers/file names/etc. specific to your code.

Minimal Failing Example

This is the most powerful debugging technique that exists. But it takes time.

I’ve used this to find bugs in major numerical libraries, standard unix utilities, several compilers, and countless of my own programs.

1. Make a copy (or git commit) of your code.
2. Slowly remove code while preserving the error.
3. Create the simplest possible block of code that still fails in the same way.
4. Analyze the simple code.

Asking for Help

Generally, you should ask for help *after* you’ve made a minimal failing example.

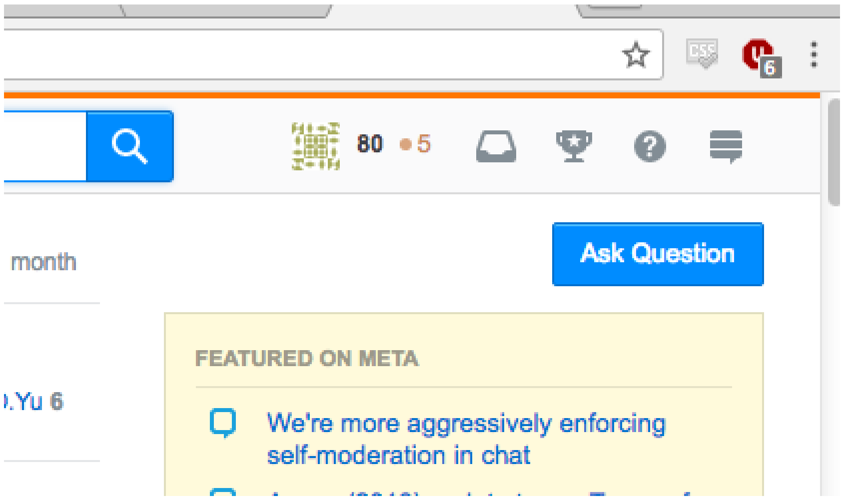
Beyond people that you know/work for/are taught by, good sources are:

- Stack overflow
- github

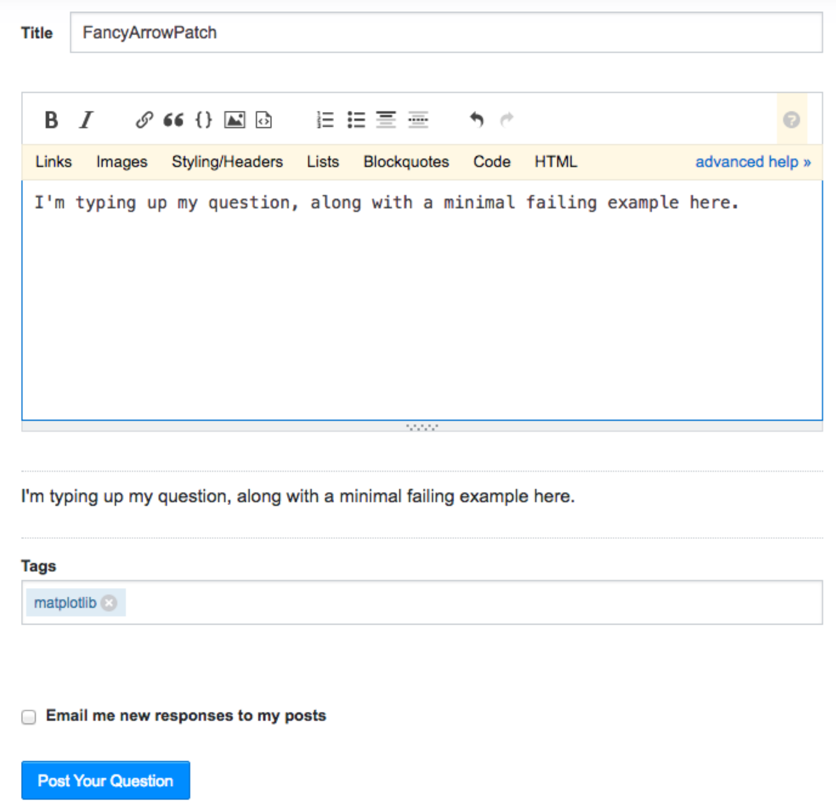
How to ask questions on Stack Overflow

1. Make an account today.

2.



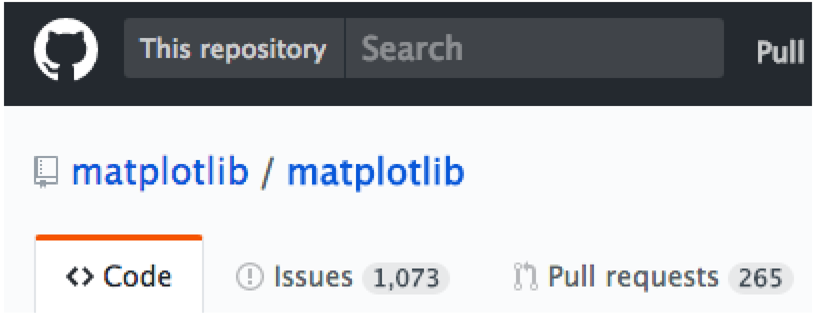
3.



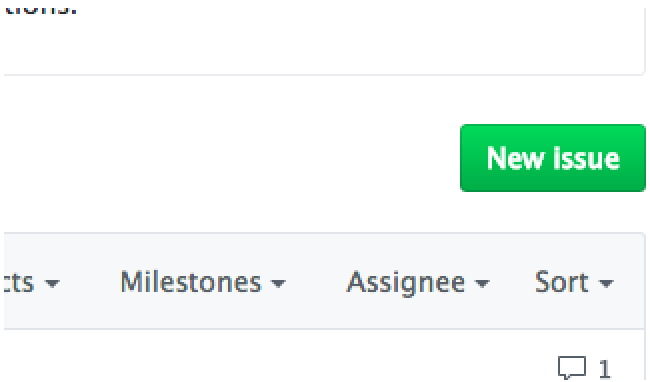
How to ask questions on github

1. Make an account today.

2.



3.



4.

How to write questions

Essential parts of a bug questions:

1. 1-2 sentence bug summary
2. Code
3. What you expected to happen
4. What actually happened
5. Version and system information

To get an answer, the code in your question needs to be:

- A minimal failing example
- Self-contained (don’t rely on data files, write everything inline)
- Compilable (MUST copy and paste into a window before submitting)

Summary:

- Think of debugging as a core part of programming, not some annoying quick thing you do after you finish programming.

- Write your code so it’s easy to debug.
- Test your code.
- Obey the sacred bug searching order.
- Google it.
- Make minimal failing examples and ask for help.