What to do when a test fails

The next slides cover each of the steps below, via a concrete example.

1. *Avoid the temptation to just try things (fiddling with your code)!*
2. **Solve the test case that failed *by hand*.**
3. **Put *print* statements** that print the values of relevant variables at relevant places, in hopes of (per the following steps) discovering when the program first went wrong.
4. **Run the program. Examine the output, line by line.** Find the first line where ***what you expected to be printed is different from what actually was printed***.
   - If all the output is now what you expected:
     - If the code now passes the test case, you are done!
     - Otherwise, return to Step 3 and add additional ***print*** statements (and possibly remove some existing ones) to discover when the program first went wrong.
5. Figure out ***why*** the output is different than you expected. That is, identify the line(s) of code where the code does not do what you wanted it to do.
6. ***Correct the mistake(s)*** that the previous step uncovered. That is, make the code do what you want it to do.
7. **Go back to Step 4.**

What to do when a test fails: a solved example

```
MUTATEs the given list of integers to become partially sorted, as follows:
  -- Compares the first (beginning) and last items in the list.
     If the first item is greater than the last item, this function swaps those two items.
  -- Compares the second and second-to-last items in the list.  If the second item is greater
     than the second-to-last item, this function swaps those two items.
  -- And so forth.

For example, if the given list of integers is:
      [50, 77, 40, 3, 90, 10, 30, 80]
then after this function call that list of integers is mutated into:
      [50, 30, 10, 3, 90, 40, 77, 80]
because 50 is compared to 80,
then 77 is compared to 30 (swap them!),
then 40 is compared to 10 (swap them!),
then 3 is compared to 90.
```

Here (below) is a solution ***with multiple errors***. The next slides work through the debugging steps one might do to locate and fix those mistakes.

```
def problem1a(list_of_integers):
    left_index = 0
    right_index = len(list_of_integers)
    for k in range(len(list_of_integers)):
        if list_of_integers[left_index] > list_of_integers[right_index]:
            list_of_integers[left_index] = list_of_integers[right_index]
            list_of_integers[right_index] = list_of_integers[left_index]
```

What to do when a test fails: a solved example

```
def problem1a(list_of_integers):
    left_index = 0
    right_index = len(list_of_integers)
    for k in range(len(list_of_integers)):
        if list_of_integers[left_index] > list_of_integers[right_index]:
            list_of_integers[left_index] = list_of_integers[right_index]
            list_of_integers[right_index] = list_of_integers[left_index]
```

```
For example, if the given list of integers is:
      [50, 77, 40, 3, 90, 10, 30, 80]
then after this function call
that list of integers is mutated into:
      [50, 30, 10, 3, 90, 40, 77, 80]
because 50 is compared to 80,
then 77 is compared to 30 (swap them!),
then 40 is compared to 10 (swap them!),
then 3 is compared to 90.
```

Here (above) is a solution ***with multiple errors***. When we run the program for the first time, the code breaks, giving the message in red below.

```
File "C:\EclipseWorkspaces\csse120\Session13_Test2Practice_mutchler\src\problem1.py", line 98,
in problem1a
    if list_of_integers[left_index] > list_of_integers[right_index]:
IndexError: list index out of range
```

Per a previous video, the message is helpful: We know that ***left_index*** or ***right_index*** is incorrect. So we put a ***print*** statement that prints them, along with a print statement that prints the argument ***list_of_integers*** for our test case.

(Continues on next slide.)

What to do when a test fails: a solved example

```
def problem1a(list_of_integers):
    left_index = 0
    right_index = len(list_of_integers)
    print(list_of_integers)
    print(left_index, right_index)
    for k in range(len(list_of_integers)):
        if list_of_integers[left_index] > list_of_integers[right_index]:
            list_of_integers[left_index] = list_of_integers[right_index]
            list_of_integers[right_index] = list_of_integers[left_index]
```

For example, if the given list of integers is:
[50, 77, 40, 3, 90, 10, 30, 80]
then after this function call
that list of integers is mutated into:
[50, 30, 10, 3, 90, 40, 77, 80]
because 50 is compared to 80,
then 77 is compared to 30 (swap them!),
then 40 is compared to 10 (swap them!),
then 3 is compared to 90.

Here (to the left) is a solution ***with multiple errors***.
When we run the program for the first time, the code breaks, giving the message on the previous slide (in red).

```
-------------------------------------------------
Testing the   problem1a   function:
-------------------------------------------------

Before the mutation: [50, 77, 40, 3, 90, 10, 30, 80]
[50, 77, 40, 3, 90, 10, 30, 80]
0 8
```

In response, (per the previous slide) we put ***print*** statements (shown in *purple*) that prints those variables along with the ***list_of_integers*** (which is our test case).

The output from the ***print*** statements is shown above.

I expected the test list to be just as what is printed – good!

I expected ***left_index*** to be ***0***, and ***0*** is printed – good!

I expected ***right_index*** to be ***7*** (for the test list), but ***8*** is printed – looks like I have identified the place where my code did not work as I expected, good! (Continues on next slide.)

What to do when a test fails: a solved example

```
def problem1a(list_of_integers):
    left_index = 0
    right_index = len(list_of_integers) - 1
    print(list_of_integers)
    print(left_index, right_index)
    for k in range(len(list_of_integers)):
        if list_of_integers[left_index] > list_of_integers[right_index]:
            list_of_integers[left_index] = list_of_integers[right_index]
            list_of_integers[right_index] = list_of_integers[left_index]
```

For example, if the given list of integers is:
[50, 77, 40, 3, 90, 10, 30, 80]
then after this function call
that list of integers is mutated into:
[50, 30, 10, 3, 90, 40, 77, 80]
because 50 is compared to 80,
then 77 is compared to 30 (swap them!),
then 40 is compared to 10 (swap them!),
then 3 is compared to 90.

Here (to the left) is a solution ***with multiple errors***. On the previous slide, we determined that the initial value for ***right_index*** was NOT what I expected: it was ***8***, but I expected ***7*** on the test list.

That woke me up to my first error – ***right_index*** should start at ***len(list_of_integers) - 1***, not ***len(list_of_integers)***.

So I made the correction and have shown the corrected line in ***purple*** in the code above.

I run the program again and this time I get the output shown below.

```
Before the mutation: [50, 77, 40, 3, 90, 10, 30, 80]
[50, 77, 40, 3, 90, 10, 30, 80]
0 7
After the mutation:  [50, 77, 40, 3, 90, 10, 30, 80]
The above should be: [50, 30, 10, 3, 90, 40, 77, 80]
FAILED the test!
```

This time the code does NOT break (good!). It prints the expected value for ***right_index*** (***7***) – good!

However, the code now ***fails the test***, as shown to the left. So now I continue the debugging, looking for a second mistake.
(Continues on next slide.)

What to do when a test fails: a solved example

```
For example, if the given list of integers is:
      [50, 77, 40, 3, 90, 10, 30, 80]
then after this function call
that list of integers is mutated into:
      [50, 30, 10, 3, 90, 40, 77, 80]
because 50 is compared to 80,
then 77 is compared to 30 (swap them!),
then 40 is compared to 10 (swap them!),
then 3 is compared to 90.
```

```
def problem1a(list_of_integers):
    left_index = 0
    right_index = len(list_of_integers) - 1
    print(list_of_integers)
    print(left_index, right_index)
    for k in range(len(list_of_integers)):
        if list_of_integers[left_index] > list_of_integers[right_index]:
            list_of_integers[left_index] = list_of_integers[right_index]
            list_of_integers[left_index] = list_of_integers[right_index]
```

Here (to the left) is a solution ***with multiple errors***. I have fixed one error (the correction is shown in ***purple***), but now the code fails the test, as shown in the output below.

```
Before the mutation: [50, 77, 40, 3, 90, 10, 30, 80]
[50, 77, 40, 3, 90, 10, 30, 80]
0 7
After the mutation:  [50, 77, 40, 3, 90, 10, 30, 80]
The above should be: [50, 30, 10, 3, 90, 40, 77, 80]
FAILED the test!
```

Now I need to put additional ***print*** statements, this time INSIDE the loop. Once again, I am trying to locate the first place in the code's execution where what is printed is NOT what I expected to be printed.

Inside the loop, I want to print just about everything, since I do not know what is going wrong. So inside the loop, before the IF statement, I put:

```
print(k, left_index, right_index, list_of_integers[left_index],
                                  list_of_integers[right_index])
```

(Continues on next slide.)

What to do when a test fails: a solved example

```
def problem1a(list_of_integers):
    left_index = 0
    right_index = len(list_of_integers) - 1
    print(list_of_integers)
    print(left_index, right_index)
    for k in range(len(list_of_integers)):
        print(k, left_index, right_index, list_of_integers[left_index],
                                          list_of_integers[right_index])
        if list_of_integers[left_index] > list_of_integers[right_index]:
            list_of_integers[left_index] = list_of_integers[right_index]
            list_of_integers[right_index] = list_of_integers[left_index]
```

```
For example, if the given list of integers is:
      [50, 77, 40, 3, 90, 10, 30, 80]
then after this function call
that list of integers is mutated into:
      [50, 30, 10, 3, 90, 40, 77, 80]
because 50 is compared to 80,
then 77 is compared to 30 (swap them!),
then 40 is compared to 10 (swap them!),
then 3 is compared
```

Here (to the left) is a solution ***with multiple errors***. I have fixed one error, but the code now fails the test. In response, I added another ***print*** statement, shown in ***purple***.

Note that I used a ***single*** *print* statement rather than multiple ones. That makes the output easier to read when it is in a loop – I get one line of output for each iteration of the loop.

I run the program again. The relevant output is shown below.

```
0 7
0  0 7 50 80
1  0 7 50 80
2  0 7 50 80
3  0 7 50 80
4  0 7 50 80
5  0 7 50 80
6  0 7 50 80
7  0 7 50 80
```

Yikes! Although ***k*** is changing as expected (it is the left column of numbers), the other variables are ***not changing at all*** as the loop continues!

Again, I have found a place where ***what is printed is NOT what I expected to be printed***. Again, I ask myself: Why did my code behave in this unexpected way? (Continues on next slide.)

What to do when a test fails: a solved example

```
For example, if the given list of integers is:
      [50, 77, 40, 3, 90, 10, 30, 80]
then after this function call
that list of integers is mutated into:
      [50, 30, 10, 3, 90, 40, 77, 80]
because 50 is compared to 80,
then 77 is compared to 30 (swap them!),
then 40 is compared to 10 (swap them!),
then 3 is compared to 90
```

```
def problem1a(list_of_integers):
    left_index = 0
    right_index = len(list_of_integers) - 1
    print(list_of_integers)
    print(left_index, right_index)
    for k in range(len(list_of_integers)):
        print(k, left_index, right_index, list_of_integers[left_index],
                                          list_of_integers[right_index])
        if list_of_integers[left_index] > list_of_integers[right_index]:
            list_of_integers[left_index] = list_of_integers[right_index]
            list_of_integers[right_index] = list_of_integers[left_index]
```

Here (to the left) is a solution ***with multiple errors***. I have fixed one error, but the code now fails the test. In response, I added another ***print*** statement, shown in ***purple***.

The relevant output when I ran with that new ***print*** statement is shown below.

```
0 7
0  0 7 50 80
1  0 7 50 80
2  0 7 50 80
3  0 7 50 80
4  0 7 50 80
5  0 7 50 80
6  0 7 50 80
7  0 7 50 80
```

Per the previous slide, I am surprised to see that although ***k*** is changing as expected, the other variables are ***not changing at all*** as the loop continues!

Again, I have found a place where ***what is printed is NOT what I expected to be printed***. Again, I ask myself: Why did my code behave in this unexpected way?

Ah! I forgot to make the index variables change! I meant to include:

```
left_index = left_index + 1
right_index = right_index - 1
```

at the end of the loop. So I add those statements and run the program again.

(Continues on next slide.)

What to do when a test fails: a solved example

```
For example, if the given list of integers is:
      [50, 77, 40, 3, 90, 10, 30, 80]
then after this function call
that list of integers is mutated into:
      [50, 30, 10, 3, 90, 40, 77, 80]
because 50 is compared to 80,
then 77 is compared to 30 (swap them!),
then 40 is compared to 10 (swap them!),
then 3 is compared to 90.
```

```
def problem1a(list_of_integers):
    left_index = 0
    right_index = len(list_of_integers) - 1
    print(list_of_integers)
    print(left_index, right_index)
    for k in range(len(list_of_integers)):
        print(k, left_index, right_index, list_of_integers[left_index],
                                          list_of_integers[right_index])
        if list_of_integers[left_index] > list_of_integers[right_index]:
            list_of_integers[left_index] = list_of_integers[right_index]
            list_of_integers[right_index] = list_of_integers[left_index]
        left_index = left_index + 1
        right_index = right_index - 1
```

Here (to the left) is a solution ***with multiple errors***. I have fixed two errors, but the code still fails the test. Here (below and to the left) is the relevant output when I run after adding the two lines shown in *purple*.

I re-examine the output. Now both ***left_index*** and ***right_index*** are behaving as I expected them to do, as shown by the *purple* circle in the output. Also, I see that I am comparing the right items for the first several iterations: first 50 and 80, then 77 and 30, and so forth, as shown by the ***green*** circle.

```
0 7
0 0 7  50 80
1 1 6  77 30
2 2 5  40 10
3 3 4  3 90
4 4 3  90 3
5 5 2  10 10
6 6 1  30 30
7 7 0  80 50
After the mutation:  [50, 30, 10, 3, 3, 10, 30, 50]
The above should be: [50, 30, 10, 3, 90, 40, 77, 80]  FAILED the test!
```

But the last three iterations (circled in ***red***) are NOT what I expected. And the test failed – the final state of the list has bogus numbers in it (again circled in ***red***). So I again add a ***print*** statement, this time inside the IF statement, to try to figure out what is going on.

(Continues on next slide.)

What to do when a test fails: a solved example

```
For example, if the given list of integers is:
      [50, 77, 40, 3, 90, 10, 30, 80]
then after this function call
that list of integers is mutated into:
      [50, 30, 10, 3, 90, 40, 77, 80]
```

```
def problem1a(list_of_integers):
    left_index = 0
    right_index = len(list_of_integers) - 1
    print(list_of_integers)
    print(left_index, right_index)
    for k in range(len(list_of_integers)):
        print(k, left_index, right_index, list_of_integers[left_index],
                                          list_of_integers[right_index])
        if list_of_integers[left_index] > list_of_integers[right_index]:
            print(k, list_of_integers)
            list_of_integers[left_index] = list_of_integers[right_index]
            list_of_integers[right_index] = list_of_integers[left_index]
            print(k, list_of_integers)
        left_index = left_index + 1
        right_index = right_index - 1
```

I put the new ***print*** statements (shown in ***purple***) both **before** and **after** the "swap" code, and I printed ***k*** (so that I know the iteration) as well as the entire list (because something is going wrong with it).

Here (to the left) is a solution ***with multiple errors***. I have fixed two errors, but the code still fails the test. Here (below and to the left) is the relevant output when I run after adding the two lines shown in ***purple***. (The full output is shown directly below, in smaller print.)

When ***k*** is ***0***, the code correctly compares ***50*** and ***80*** and correctly determines NOT to swap them. When ***k*** is ***1***, the code correctly compares ***77*** and ***30*** and ***correctly determines to swap them***.

```
0 7
0 0 7 50 80
1 1 6 77 30
1 [50, 77, 40, 3, 90, 10, 30, 80]
1 [50, 30, 40, 3, 90, 10, 30, 80]
 ...
```

(Continues on next slide.)

```
0 7
0 0 7 50 80
1 1 6 77 30
1 [50, 77, 40, 3, 90, 10, 30, 80]
1 [50, 30, 40, 3, 90, 10, 30, 80]
2 2 5 40 10
2 [50, 30, 40, 3, 90, 10, 30, 80]
2 [50, 30, 10, 3, 90, 10, 30, 80]
3 3 4 3 90
4 4 3 90 3
4 [50, 30, 10, 3, 90, 10, 30, 80]
4 [50, 30, 10, 3, 3, 10, 30, 80]
5 5 2 10 10
6 6 1 30 30
7 7 0 80 50
7 [50, 30, 10, 3, 3, 10, 30, 80]
7 [50, 30, 10, 3, 3, 10, 30, 50]
After the mutation:  [50, 30, 10, 3, 3, 10, 30, 50]
The above should be: [50, 30, 10, 3, 90, 40, 77,
80]
FAILED the test!
```

What to do when a test fails: a solved example

```
For example, if the given list of integers is:
      [50, 77, 40, 3, 90, 10, 30, 80]
then after this function call
that list of integers is mutated into:
      [50, 30, 10, 3, 90, 40, 77, 80]
because 50 is compared to 80,
then 77 is compared to 30 (swap them!),
then 40 is compared to 10 (swap them!),
then 3 is compared to 90.
```

```
def problem1a(list_of_integers):
    left_index = 0
    right_index = len(list_of_integers) - 1
    print(list_of_integers)
    print(left_index, right_index)
    for k in range(len(list_of_integers)):
        print(k, left_index, right_index, list_of_integers[left_index],
                                          list_of_integers[right_index])
        if list_of_integers[left_index] > list_of_integers[right_index]:
            print(k, list_of_integers)
            list_of_integers[left_index] = list_of_integers[right_index]
            list_of_integers[right_index] = list_of_integers[left_index]
            print(k, list_of_integers)
        left_index = left_index + 1
        right_index = right_index - 1
```

Here (to the left) is a solution ***with multiple errors***. I have fixed two errors, but the code still fails the test. Here (below) is the relevant output when I run after adding the two lines shown in ***purple***. (The full output is shown on the previous slide.)

```
0 7
0 0 7 50 80
1 1 6 77 30
1 [50, 77, 40, 3, 90, 10, 30, 80]
1 [50, 30, 40, 3, 90, 10, 30, 80]
 ...
```

When ***k*** is ***0***, the code correctly compares ***50*** and ***80*** and correctly determines NOT to swap them. When ***k*** is ***1***, the code correctly compares ***77*** and ***30*** and ***correctly determines to swap them***.

But when the "swap" of ***77*** and ***30*** occurs., the list at index ***1*** becomes ***30*** (good!) but I expected the list at index ***6*** to become ***77***. The print statement says that it remains ***30*** (circled in red in the output). So I run the code "by hand":

`list_of_integers[left_index] = list_of_integers[right_index]` list at 1 becomes value of list at 6, which is 30. Good!

`list_of_integers[right_index] = list_of_integers[left_index]` list at 6 becomes value of list at 1, which is now ***30***. Oops!

What to do when a test fails: a solved example

```
def problem1a(list_of_integers):
    left_index = 0
    right_index = len(list_of_integers) - 1
    print(list_of_integers)
    print(left_index, right_index)
    for k in range(len(list_of_integers)):
        print(k, left_index, right_index, list_of_integers[left_index],
                                          list_of_integers[right_index])
        if list_of_integers[left_index] > list_of_integers[right_index]:
            print(k, list_of_integers)
            list_of_integers[left_index] = list_of_integers[right_index]
            list_of_integers[right_index] = list_of_integers[left_index]
            print(k, list_of_integers)
        left_index = left_index + 1
        right_index = right_index - 1
```

```
For example, if the given list of integers is:
      [50, 77, 40, 3, 90, 10, 30, 80]
then after this function call
that list of integers is mutated into:
      [50, 30, 10, 3, 90, 40, 77, 80]
because 50 is compared to 80,
then 77 is compared to 30 (swap them!),
then 40 is compared to 10 (swap them!),
then 3 is compared to 90.
```

Here (to the left) is a solution ***with multiple errors***. I have fixed two errors, but the code still fails the test. Here (below) is the relevant output when I run after adding the two lines shown in ***purple***. (The full output is shown on a previous slide.)

```
 0 7
 0 0 7 50 80
 1 1 6 77 30
 1 [50, 77, 40, 3, 90, 10, 30, 80]
 1 [50, 30, 40, 3, 90, 10, 30, 80]
  ...
```

Per the analysis on the previous slide, I see that my "swap" technique does not work. So I google for "swap variables" and learn that the right way to swap variables A and B is per this pattern:

```
temp = A
A = B
B = temp
```

I then correct my code to do the swap correctly and run the program again… (Continues on next slide.)

What to do when a test fails: a solved example

```
For example, if the given list of integers is:
      [50, 77, 40, 3, 90, 10, 30, 80]
then after this function call
that list of integers is mutated into:
      [50, 30, 10, 3, 90, 40, 77, 80]
because 50 is compared to 80,
then 77 is compared to 30 (swap them!),
then 40 is compared to 10 (swap them!),
then 3 is compared to 90.
```

The correct swap (per the previous slide) is shown in *purple* in the code below.

When I run the program with the correct swap, I get the output shown below. (I have shown the output only to the relevant point.)

```
def problem1a(list_of_integers):
    left_index = 0
    right_index = len(list_of_integers) - 1
    print(list_of_integers)
    print(left_index, right_index)
    for k in range(len(list_of_integers)):
        print(k, left_index, right_index, list_of_integers[left_index],
                                          list_of_integers[right_index])
        if list_of_integers[left_index] > list_of_integers[right_index]:
            print(k, list_of_integers)
            temp = list_of_integers[left_index]
            list_of_integers[left_index] = list_of_integers[ri
            list_of_integers[right_index] = temp
            print(k, list_of_integers)
        left_index = left_index + 1
        right_index = right_index - 1
```

```
0 7
0 0 7 50 80
1 1 6 77 30
1 [50, 77, 40, 3, 90, 10, 30, 80]
1 [50, 30, 40, 3, 90, 10, 77, 80]
2 2 5 40 10
2 [50, 30, 40, 3, 90, 10, 77, 80]
2 [50, 30, 10, 3, 90, 40, 77, 80]
3 3 4 3 90
4 4 3 90 3
4 [50, 30, 10, 3, 90, 40, 77, 80]
4 [50, 30, 10, 90, 3, 40, 77, 80]
```

When ***k*** is ***0*** (see first blue circle in output), the code correctly does NOT swap ***50*** and ***80***. When ***k*** is ***1***, the code correctly and successfully swaps the ***77*** and ***30*** (see the first set of green and purple circles). When ***k*** is ***2***, the code correctly and successfully swaps ***40*** and ***10*** (next set of green/purple circles). When ***k*** is ***3***, the code correctly does NOT swap ***3*** and ***90*** (see second blue circle). But … (Continues on next slide.)

What to do when a test fails: a solved example

```
For example, if the given list of integers is:
      [50, 77, 40, 3, 90, 10, 30, 80]
then after this function call
that list of integers is mutated into:
      [50, 30, 10, 3, 90, 40, 77, 80]
because 50 is compared to 80,
then 77 is compared to 30 (swap them!),
then 40 is compared to 10 (swap them!),
then 3 is compared to 90.
```

The correct swap (per the previous slide) is shown in *purple* in the code below.

When I run the program with the correct swap, I get the output shown below. (I have shown the output only to the relevant point.)

```
def problem1a(list_of_integers):
    left_index = 0
    right_index = len(list_of_integers) - 1
    print(list_of_integers)
    print(left_index, right_index)
    for k in range(len(list_of_integers)):
        print(k, left_index, right_index, list_of_integers[left_index],
                                          list_of_integers[right_index])
        if list_of_integers[left_index] > list_of_integers[right_index]:
            print(k, list_of_integers)
            temp = list_of_integers[left_index]
            list_of_integers[left_index] = list_of_integers[ri
            list_of_integers[right_index] = temp
            print(k, list_of_integers)
        left_index = left_index + 1
        right_index = right_index - 1
```

```
0 7
0 0 7 50 80
1 1 6 77 30
1 [50, 77, 40, 3, 90, 10, 30, 80]
1 [50, 30, 40, 3, 90, 10, 77, 80]
2 2 5 40 10
2 [50, 30, 40, 3, 90, 10, 77, 80]
2 [50, 30, 10, 3, 90, 40, 77, 80]
3 3 4 3 90
4 4 3 90 3
4 [50, 30, 10, 3, 90, 40, 77, 80]
4 [50, 30, 10, 90, 3, 40, 77, 80]
```

(Continued from the previous slide.) When ***k*** is ***4***, the code looks at ***90*** and ***3***, sees that they are in the wrong order, and ***re-swaps them back to their original positions.*** (See the set of green and red circles.) ***Oops!***

My loop is going too far. After it reaches the middle, I have to STOP the loop at that point. I make that correction, run again and ***pass the tests!***