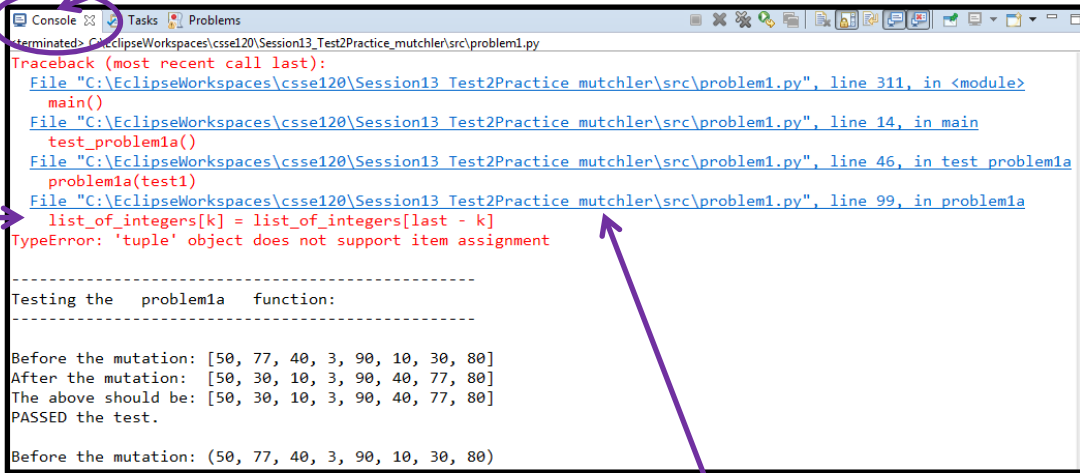


# *What to do when you get a run-time error message*

1. Look at the *console output*.
2. Find the *line that broke*.
3. Click on its *blue link*. More precisely, click on the lowest *blue link* that leads to *your* code.
4. Decipher the *red error message* that appeared at the bottom of the console message.
  - Oftentimes, you will see your mistake at this point. If so, fix and re-test.
  - If not, use the *Traceback (stack trace)* as needed to further track down your problem. You will probably also use *PRINT* as described in the next video.

The next slides cover each of these points.

## 1. Look at the *console output*.



```
terminated> C:\EclipseWorkspaces\csse120\Session13_Test2Practice_mutchler\src\problem1.py
Traceback (most recent call last):
  File "C:\EclipseWorkspaces\csse120\Session13_Test2Practice_mutchler\src\problem1.py", line 311, in <module>
    main()
  File "C:\EclipseWorkspaces\csse120\Session13_Test2Practice_mutchler\src\problem1.py", line 14, in main
    test_problem1a()
  File "C:\EclipseWorkspaces\csse120\Session13_Test2Practice_mutchler\src\problem1.py", line 46, in test_problem1a
    problem1a(test1)
  File "C:\EclipseWorkspaces\csse120\Session13_Test2Practice_mutchler\src\problem1.py", line 99, in problem1a
    list_of_integers[k] = list_of_integers[last - k]
TypeError: 'tuple' object does not support item assignment

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

Before the mutation: [50, 77, 40, 3, 90, 10, 30, 80]
After the mutation:  [50, 30, 10, 3, 90, 40, 77, 80]
The above should be: [50, 30, 10, 3, 90, 40, 77, 80]
PASSED the test.

Before the mutation: (50, 77, 40, 3, 90, 10, 30, 80)
```

## 2. Find the *line that broke*.

It will be in *red* just below the **BOTTOM** *blue link*.

Note: Sometimes the **output** from the problem (from *print* statements) gets intermixed with the **Traceback** and **error message**. The latter are always in **red**, and your output (from printing) will always be in **black**. **See the next slide for an example of the intermixing that may occur.**

In this example, the output appeared AFTER the Traceback and error message, but in other cases it may appear BEFORE the Traceback and error message or even intermingled with the Traceback!

## 3. Click on its *blue link*.

More precisely, click on the lowest *blue link* that leads to *your* code. **See the example on a slide coming up.**

# An example showing the output (from *print*) intermixed with the *Traceback* and *error message*

```
Console <terminated> <terminated> <terminated>
-----
Testing the  problem1a  function:
-----
Before the mutation: [50, 77, 40, 3, 90, 10, 30, 80]
After the mutation: [50, 30, 10, 3, 90, 40, 77, 80]
The above should be: [50, 30, 10, 3, 90, 40, 77, 80]
PASSED the test.
Traceback (most recent call last):
  File "C:\EclipseWorkspaces\csse120\Session13 Test2Practice mutchler\src\problem1.py", line 316, in <mo
Before the mutation: [50, 77, 40, 3, 90, 10, 30, 80]
After the mutation: [50, 30, 10, 3, 90, 40, 77, 80]
The above should be: [50, 30, 10, 3, 90, 40, 77, 80]
PASSED the test.
-----
Testing the  problem1b  function:
-----
Test 1: calling problem1b 5 times.
Each x should be random in [ 71, 108]
Each y should be random in [101, 134]
Each r should be random in [1, 3]
main()
File "C:\EclipseWorkspaces\csse120\Session13 Test2Practice mutchler\src\p
test_problem1b()
File "C:\EclipseWorkspaces\csse120\Session13 Test2Practice mutchler\src\p
circle = problem1b(min_x, max_x, min_y, max_y, my_radius)
File "C:\EclipseWorkspaces\csse120\Session13 Test2Practice mutchler\src\p
circle = rg.Circle(point, radius)
File "C:\EclipseWorkspaces\csse120\Session13 Test2Practice mutchler\src\r
super().__init__(center, tkinter.Canvas.create_oval)
File "C:\EclipseWorkspaces\csse120\Session13 Test2Practice mutchler\src\r
self.center = center.clone()
AttributeError: 'tuple' object has no attribute 'clone'
```

*Traceback* starts here ...

But the output from *print* appears both before the *Traceback* ...

... and in the *middle* of the *Traceback*.

... and *Traceback* continues here.

In general, the output from *print* might appear anywhere: before the *Traceback*, inside it, after it, or some combination of all three.

Just look for the *red* and *blue* for the *Traceback* and *error message*.

# How to select the **blue link** upon which to click when your code breaks inside a library like rosegraphics

Testing the

Traceback (m

```
File "C:\EclipseWorkspaces\csse120\Session13 Test2Practice mutchler\src\problem4.py", line 14, in main
    main()
File "C:\EclipseWorkspaces\csse120\Session13 Test2Practice mutchler\src\problem4.py", line 51, in test_problem4
    test_problem4()
File "C:\EclipseWorkspaces\csse120\Session13 Test2Practice mutchler\src\problem4.py", line 51, in test_problem4
    st.SimpleTestCase.run_tests('problem4', [test1])
File "C:\EclipseWorkspaces\csse120\Session13 Test2Practice mutchler\src\simple testing.py", line 73, in run_tests
    result = tests[k].run_test()
File "C:\EclipseWorkspaces\csse120\Session13 Test2Practice mutchler\src\simple testing.py", line 47, in run_test
    your_answer = self.function_to_test(*(self.arguments_to_use))
File "C:\EclipseWorkspaces\csse120\Session13 Test2Practice mutchler\src\problem4.py", line 138, in problem4
    circle = rg.Circle((100, 50), 30)
File "C:\EclipseWorkspaces\csse120\Session13 Test2Practice mutchler\src\rosegraphics.py", line 786, in init
    super().__init__(center, tkinter.Canvas.create_oval)
File "C:\EclipseWorkspaces\csse120\Session13 Test2Practice mutchler\src\rosegraphics.py", line 634, in init
    self.center = center.clone()
AttributeError: 'tuple' object has no attribute 'clone'
```

2. Find the **line that broke**. In **red** just below the **BOTTOM blue link**.

### Important:

The line that broke will always be at the **BOTTOM** of the Traceback. The error message will always follow that line. **But sometimes, as in this example, that line is part of a library module like rosegraphics.**

In that case, you want to find the last place before the code broke **that was in YOUR code**. So you work your way **BACK** through the **Traceback** (that is, **UP** in the list) until you find a line that is in YOUR code. **I have circled that line in the above.**

3. Click on its **blue link**.

More precisely, click on the lowest **blue link** that leads to **your** code. See explanation in above box.

# What to do when you get a *run-time error* *message*

1. Look at the *console output*.
2. Find the *line that broke*.
3. Click on its *blue link*. More precisely, click on the lowest *blue link* that leads to *your* code.

4. Decipher the *red error message* that appeared at the bottom of the console message.
  - Oftentimes, you will see your mistake at this point. If so, fix and re-test.
  - If not, use the *Traceback (stack trace)* as needed to further track down your problem. You will probably also use *PRINT* as described in the next video.

Deciphering those messages is very perplexing at first, but very easy once you “learn the lingo.” The next slides decipher some example error messages that you may encounter. Refer back to the following slides if you get an error message that you cannot decipher!

Exception: Could not place the shape on the given window.

Did you **accidentally render a closed window?**

**Example error message, with a key phrase circled:**

raise Exception(message) from None

Exception: Could not place the shape on the given window.

Did you **accidentally render a closed window?**

**Example code  
that produced  
the error message:**

```
oval = rg.Ellipse(rectangle.corner_1,
                  rectangle.corner_2)
oval.attach_to(window)
window.render()
window.close_on_mouse_click()
```

The above error message was generated by a statement in rosegraphics. The line in the student's code that led to the error message is the line written in red.

**Explanation:**

- “render a closed window” means that the window in `window.render()` is currently closed. Rosegraphics does not allow you to draw on a window once it is closed (reasonably enough).
- So you look for a statement that might have (erroneously) closed the window. That statement is obvious in this example: it is the line that immediately follows the `window.render()` line:  
`window.close_on_mouse_click()`
- It so happens that in this (and many other) problems, the green specification of the function did NOT ask you to close the window. So it is *wrong* to close the window in the function – functions should do no “side effects” beyond those demanded by the specification. Instead, the window should be closed in the **testing** code at the appropriate place.

The error message was generated when the function was called a second time on the window. The student's code above wrongly closed the window, so the code broke on that second call to the function. Note that the mistake did NOT occur on the line that broke – sometimes, like here, you have to do detective work to find the actual source of the error.

AttributeError: 'Blah' object has no attribute 'foo'

*Example error message, with a key phrase circled:*

AttributeError: 'int' object has no attribute 'x'

*Example code that produced the error message:*

```
super().__init__(Point((corner_2.x+ corner_1.x) / 2,
```

The error message (in red) was generated by a statement in rosegraphics, shown here (also in red).

```
oval = rg.Ellipse(rectangle.corner_1.x, rectangle.corner_2.y)
```

The line in the student's code that led to the error message is the line shown here, in black.

*Explanation:*

- **This message has an unambiguous and very helpful meaning, namely:**  
There is some object that is of type `int` and that object has a `.x` after it.
- You don't necessarily know why the object is of type `int`, nor whether it should or shouldn't have a `.x` after it, but you definitely know that it is of type `int` and it has a `.x` after it and that that combination is no good. (Integers do not have an "x" attribute.)
- Usually that is enough of a hint to spot the error. In this case, the red code makes it clear that the object is either `corner_2.x` or `corner_1.x`, since they are the only two names with a `.x` after them. Presumably a corner should be an `rg.Point` (so having a `.x` attribute makes sense), but apparently one or both of these corners are `int` objects, not `rg.Point` objects.
- So you look at your code and see if perhaps you gave the `rg.Ellipse` constructor arguments of the wrong type. Yep! An `rg.Ellipse` needs two `rg.Point` objects, but this code gives an `x` (which is an `int`) and a `y` (also an `int`). The author of the code perhaps intended to write:

```
oval = rg.Ellipse(rectangle.corner_1, rectangle.corner_2)
```

## TypeError:

*BLAH* object does not support

*item assignment*

*Example error message, with a key phrase circled:*

TypeError: 'tuple' object does not support

*item assignment*

*Example code that produced the error message:*

```
list_of_integers[k] = list_of_integers[last - k]
```

*Explanation:*

- “Assignment” means an `=` sign.
- “Item assignment” means that the assignment is to an item in a sequence, as in `blah[k] = ...`
- “Tuple object” means an object that is a tuple, as in `(3, 29, 1, 4)`.
- So this message is saying that you cannot assign a value to an item in a TUPLE. Hopefully that triggers your memory (or you look up in the videos et al) that **tuples are immutable**. So the problem is that you are trying to mutate a tuple.

So in the code above, it looks like `LIST_of_integers` is a TUPLE (despite its name) when it should be a LIST. You can verify this by putting a PRINT statement just before the line that broke and re-running:

```
print(list_of_integers)
```

Ultimately, this might be an error in the testing code (inadvertently sending a TUPLE where the function demands a LIST).



- I will be adding more examples of error messages (and what they mean) over the next couple of days.