Errors

Your code will not always run correctly: in fact, your code may not always run at all. Usually when your code fails to execute, you get some sort of error message. The code below causes an error when executed:

In [ ]:
```
# Example E-1: Bad Code

print("A" + 2)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[29], line 3
      1 # Example E-1: Bad Code
----> 3 print("A" + 2)

TypeError: can only concatenate str (not "int") to str
```

Errors come in all sorts of formats but usually the error message has this format:

- Something saying there's a traceback
- The line Python was on when the error was raised (THIS IS NOT NECESSARILY THE SAME LINE THAT CAUSED THE ERROR)
- The error type and the description of the error

Another common error format you'll see is this:

```
Traceback (most recent call last):
  File "c:\Users\antho\Documents\ENGR 102\Lab 2\more_linear_interpolation.py", line 15, in <module>
    print("A" + 2)
TypeError: can only concatenate str (not "int") to str
```

Again, the same template applies though. There's the line that says "Traceback", the location of where the error was found (again, not necessarily where the error actually is), and the error type and description. Being able to read these error messages can save you a bunch of time and recognizing the format is critical as the error messge has two pieces of critical information:

- Where the error is (or around where): Usually the error is on the same line as listed but rarely, errors can cause the message to incorrect list on the next line, so always look a couple lines above and below the listed line to be sure
- What the error actually is: This info can be really helpful in figuring out what you need to do to fix the bug

In my case, the error is on the line `print("A" + 2)` and the error says that Python `can only concatenate str (not "int") to str`. This means that I tried adding an integer and a string together. To resolve this, I will need to ensure that I convert the 2 to a string. The solution does depend on context (maybe I should have changed the string to a number beforehand somehow).

Now this is the general way of dealing with errors. There is quite the list of common errors you will face, and I'll put them in some sort of organized fashion. Also note that these errors are well... common... so you can probably copy and paste the last line into Google and get a nice Stack Overflow link explaining the error and how to solve it.

String Errors

Unterminated string literal (SyntaxError)

This means that you did not close your string properly. Make sure that your text is enclosed by double quotes or single quotes (don't mix or you'll get the same error). If you have too many quotes, you'll get the same error as well (Example E-S2).

In [ ]:
```
# Example E-S1: Unterminated String Literal
print("a)
```

```
  Cell In[38], line 2
    print("a)
          ^
SyntaxError: unterminated string literal (detected at line 2)
```

In [ ]:
```
# Example E-S2: Extra quote
print("a"")
```

```
  Cell In[39], line 2
    print("a"")
             ^
SyntaxError: unterminated string literal (detected at line 2)
```

Concatenating with non-strings (TypeError)

When you are adding strings with plus signs (in other words, while performing string concatenation), the values left and right of the plus signs must be strings. If not, you get the error shown below, and Python will tell you what the unexpected value type is. Now it won't tell you which of the two is the wrong type, so you need to go find that out yourself.

```
In [ ]: # Example E-S3: Adding a string and a non-string
        print("A" + 3)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[40], line 2
      1 # Example E-S3: Adding a string and a non-string
----> 2 print("A" + 3)

TypeError: can only concatenate str (not "int") to str
```

String does not support item assignment (TypeError)

This comes back to a property of strings: they are immutable. You can never directly a change, but you can make a copy of a string with changes to it. The code below attempts to directly change the string itself with indexing. You can access a character with indices, but you may not change the contents in this fashion.

```
In [ ]: # Example E-S4: Immutable String meets Wrong Code
        letters = "ABCDE"
        letters[1] = "G"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[43], line 3
      1 # Example E-S4: Immutable String meets Wrong Code
      2 letters = "ABCDE"
----> 3 letters[1] = "G"

TypeError: 'str' object does not support item assignment
```

Accidental Special Characters

Be careful with using backwards slashes. By itself, it's used in conjunction with the letter next to it and it will negate that letter's special effect if possible. In E-S5, the backwards slash actually negates the double quote's ability to complete the string, which means Python think you forgot to close the string. This type of error can be rather tricky so pay careful attention whenever you're trying to use backwards slashes.

```
In [ ]: # Example E-S5: Special characters... or just wrong characters
        print("\")
```

```
  Cell In[42], line 2
    print("\")
          ^
SyntaxError: unterminated string literal (detected at line 2)
```

Typing Errors

These are errors because you accidently mistyped something (added some extra characters there and there or removed some there and there). Although the error messages tend to be straight forward, these historically have been pretty painful to deal with at times since they're not very obviously spotted.

Missing or Extra Parentheses

This can get a little tricky sometimes. Generally, if you're missing the first parentheses rather than the last one, you get an error similar to E-T2 and E-T3 (mismatching parenteheses). However, parentheses are also vital for functions, so you may get an error similar to E-T1 where your missing parentheses (generally end) looks like invalid function syntax. Check your code very carefully to ensure all your parentheses are properly paired with one another.

```
In [ ]: # Example E-T1: Missing parentheses
        print(((1 + 2) + 4)
```

```
  Cell In[41], line 2
    print(((1 + 2) + 4)
                       ^
SyntaxError: incomplete input
```

```
In [ ]: # Example E-T2: Still missing parentheses
        x = (1 + 2) + 4)
```

```
  Cell In[44], line 2
    x = (1 + 2) + 4)
                   ^
SyntaxError: unmatched ')'
```

```
In [ ]: # Example E-T3: One extra parentheses
        print(1+ 2))
```

```
 Cell In[45], line 2
   print(1+ 2))
              ^
SyntaxError: unmatched ')'
```

No module named "blah blah blah" (ModuleNotFoundError)

You either forgot to install it with `pip` (I believe it's `python -m pip install {module_here}` for Windows and `python3 -m pip install {module_here}` for MacOS) or you misspelled the module name (without fail, someone will misspell `matplotlib` when we get to that unit). Ensure that the module is spelled correctly and that it is installed onto the version of Python you are using as the interpretter.

```
In [ ]: # Example E-T4: An attempt at importing matplotlib
        import mathplotlib
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[46], line 2
      1 # Example E-T4: An attempt at importing matplotlib
----> 2 import mathplotlib

ModuleNotFoundError: No module named 'mathplotlib'
```

Missing Comma (usually SyntaxError)

So missing commas can break a lot of code, especially since commas are required to separate arguments from one another. Sometimes you'll get a nice error message like the one in E-T5 that's extremely straight forward, other times, Python is gonna tell you that it can't make sense of your values.

```
In [ ]: # Example E-T5: Missing comma
        print("YO" 2)
```

```
 Cell In[15], line 2
   print("YO" 2)
         ^
SyntaxError: invalid syntax. Perhaps you forgot a comma?
```

```
In [ ]: # Example E-T6: Where the comma at
        from math import sin, radians, power

        print(power(sin(radians) 3))
```

```
 Cell In[47], line 4
   print(power(sin(radians) 3))
               ^
SyntaxError: invalid syntax. Perhaps you forgot a comma?
```

ImportError

This generally happens when you try to import something that doesn't exist. It's different from `ModuleNotFoundError` because this occurs when you try to import something from a module that does exist but the thing you're trying to import does not. This time, ensure that you spelled the name of the thing you're importing correctly and that you are importing from the correct module.

```
In [ ]: # Example E-T7: Ah yes, sin/cos = tang
        from math import tang

        print(tang(3))
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
Cell In[48], line 2
      1 # Example E-T7: Ah yes, sin/cos = tang
----> 2 from math import tang
      4 print(tang(3))

ImportError: cannot import name 'tang' from 'math' (unknown location)
```

```
In [ ]: # Example E-T8: Right function, wrong module
        from turtle import cos
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
Cell In[49], line 2
      1 # Example E-T8: Right function, wrong module
----> 2 from turtle import cos

ImportError: cannot import name 'cos' from 'turtle' (c:\Users\antho\AppData\Local\Programs\Python\Python310\lib\
turtle.py)
```

NameError

As the name suggests, there's an error with the name of something. Usually this happens because you misspelled a variable or function name. Ultimately, you will get this error if you reference something (like a variable or function) by name and Python can't find it in your code. Some other times, you will get the error if you forget to import the module it comes in or forget to prefix the function or constant with `module_name.` (examples E-T11 and E-T12).

In [ ]:
```python
# Example E-T9: Misspelled function

def howdy():
    print("Howdy!")

howy()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[50], line 6
      3 def howdy():
      4     print("Howdy!")
----> 6 howy()

NameError: name 'howy' is not defined
```

In [ ]:
```python
# Example E-T10: Variable never defined

print(my_rizz_level)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[51], line 3
      1 # Example E-T10: Variable never defined
----> 3 print(my_rizz_level)

NameError: name 'my_rizz_level' is not defined
```

None Errors

Surprisingly yes, there's actually a lot of errors you get when you try to do stuff with `None`. 99% of the time, this is unintentional: you try to do something but it unexpectedly returns `None` (the default behavior of a function with no return type or the general behavior of a function that's supposed to return nothing back to you).

Most of the time, these errors can be resolved by fixing the code before it that returned a `None` in the first place rather than the list or dictionary or string or whatever it is you were actually expecting. But I will show some examples of `None` being youtr not-so-best friend.

Forgot to Return

You have a function that's supposed to return something. Keywords here being "supposed to". If you read the function `getHour` I made below for Example E-N1, you can see that I even type hinted that the function is supposed to return an integer (with `-> int`). Obviously, you can see that I didn't put any sort of `return` statement, which means this function returns `None`.

I use the function as if I made it correctly and I use the value that was supposed to be return to compare to another number to see if the time is for the morning or the afternoon. I get an error that I cannot compare `NoneType` and `int`. 12 is obviously an integer, so `getHour(time)` must be `None` and thus I must have forgot a `return` statement somewhere.

This is especially important for programs with any sort of nesting (nested conditionals and nested loops) as if your function is supposed to return something, you must ensure that every possible path through the branches/nests leads to a `return` statement. This problem is demonstrated in Example E-N2.

(If you are curious what the solution is for E-N1, I just need to add `return hour` to the end of the function. For E-N2, I need to add `return tax` in the `else` block.)

In [ ]:
```python
# Example E-N1: The classic forgot to return

def getHour(time: str) -> int:
    """ Gets the hour from a time given in 24HR, HH:MM format """
    hour = int(time.split(":")[0])

time = input("Enter the time: ")
if getHour(time) < 12:
    print("It's the morning!")
else:
    print("It's the afternoon!")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[53], line 8
      5     hour = time.split(":")[0]
      7 time = input("Enter the time: ")
----> 8 if getHour(time) < 12:
      9     print("It's the morning!")
     10 else:

TypeError: '<' not supported between instances of 'NoneType' and 'int'
```

```
In [ ]: # Example E-N2: Nesting inside a function
def getTax(cost: float) -> float:
    if cost <= 0:
        return 0
    else:
        tax = cost * 1.0825

iceCreamCost = 2.00
if iceCreamCost + getTax(iceCreamCost) > 2.05:
    print("Too expensive!")
else:
    print("Let's get ice cream!")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[54], line 9
      6         tax = cost * 1.0825
      8 iceCreamCost = 2.00
----> 9 if iceCreamCost + getTax(iceCreamCost) > 2.05:
     10     print("Too expensive!")
     11 else:

TypeError: unsupported operand type(s) for +: 'float' and 'NoneType'
```

Invalid Data

This tends to be a more advanced error since this is reliant on you using code that returns unexpected results. There's also the occasional slip up of using `None` as a variable's default value and forgetting to update it later. Example E-N3 shows the default value scenario, and E-N4 should should show the unexpected results scenario. It won't be as straight-forward usually, but I want to highlight again that `NoneType` errors typically occur from edge cases you didn't consider or just code that's not working as intended.

```
In [ ]: # Example E-N3: Default None value

ice_cream_flavor = None

""""
Bunch of code here...
"""

if ice_cream_flavor != "chocolate":
    print("I'm a chocolate fan, but I would support " + ice_cream_flavor + "!")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[57], line 10
      5 """"
      6 Bunch of code here...
      7 """
      9 if ice_cream_flavor != "chocolate":
---> 10     print("I'm a chocolate fan, but I would support " + ice_cream_flavor + "!")

TypeError: can only concatenate str (not "NoneType") to str
```

```
In [ ]: # Example E_N4: Printed instead of returned
def is_chocolate(flavor: str) -> str:
    if flavor == "chocolate":
        return "Yay!"
    else:
        print("Aww.")

favorite_flavor = input("Enter your favorite flavor: ")
print("Anthony replied with: " + is_chocolate(favorite_flavor))
```

```
Aww.
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[58], line 9
      6         print("Aww.")
      8 favorite_flavor = input("Enter your favorite flavor: ")
----> 9 print("Anthony replied with: " + is_chocolate(favorite_flavor))

TypeError: can only concatenate str (not "NoneType") to str
```

These examples are non-realistic and pretty forced, but again, they just serve to demonstrate that techncially correct code can lead to logic errors and undesired results that will cause errors in other places of your code.

Collections (Lists, Dictionaries, etc.)

Structures such as lists and dictionaries and strings (already mentioned) have their fair share of common errors. Most of these errors end up being `IndexError`, but there are some others that occasionally get run into.

Index out of range (IndexError)

If you try to access outside the range of the collection, you will get this error. It's as simple as that. Ensure you are aware that Python uses zero-indexing and that your collection is not missing any values. Of course, also ensure you are using the correct index.

In [ ]:
```python
# Example E-C1: Index out of range
names = ["Anthony", "Sophie"]
print(names[2])
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[59], line 3
      1 # Example E-C1: Index out of range
      2 names = ["Anthony", "Sophie"]
----> 3 print(names[2])

IndexError: list index out of range
```

The infamous adding a list to a non-list

This happens a lot in ENGR 102. And it's fine because it's usually easy to fix. Usually you're looping through a list, and instead of accessing the value at an index or using the value from the list, you use the actual list itself and do stuff to it. A lot of times, this makes Python unhappy such as with Example E-C2. Though this example is for lists, you will see this a lot with any sort of loopable collection.

In [ ]:
```python
# Example E-C2: Adding a list (this always happens)

letters = ["A", "B", "C", "D", "E", "F"]

for letter in letters:
    print("The current letter is: " + letters)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[60], line 6
      3 letters = ["A", "B", "C", "D", "E", "F"]
      5 for letter in letters:
----> 6     print("The current letter is: " + letters)

TypeError: can only concatenate str (not "list") to str
```

{insert some type here} is not subscriptable

It means whatever you just tried to access like a list, is indeed not a list (or similar type such as string). This usually gets complicated when you use functions that return something you can loop over but not necessarily are able to subscript. Ensure that you cast the object to `list` (if legal) before trying to subscript it.

In [ ]:
```python
# Example E-C3: A list but not actually a list
def get_squares(n: int):
    for i in range(n+1):
        yield i ** 2

squares = get_squares(5)
print(squares[1])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[64], line 7
      4         yield i ** 2
      6 squares = get_squares(5)
----> 7 print(squares[1])

TypeError: 'generator' object is not subscriptable
```

List Indices must be integers or slices (TypeError)

This is a self-explanatory error: you're using something under numeric indices or list slicing to access a list. Sometimes, you may be confusing data structures (such as with Example E-C4) or you mistyped the variable name (meant to use `scores` instead of `names` for example if still using E-C4). This is also a popular error that happens, so make sure you're accessing the list correctly and you're actually wanting to access a list in the first place.

In [ ]:
```
# Example E-C4: Wrong list slice

scores = {"Anthony": 1}
names = ["Anthony"]

print(names["Anthony"])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[67], line 6
      3 scores = {"Anthony": 1}
      4 names = ["Anthony"]
----> 6 print(names["Anthony"])

TypeError: list indices must be integers or slices, not str
```

KeyError

Something is wrong with your key. In this case, it simply doesn't exist in the dictionary. Remember that you add a key-value pair into a dictionary using assignment (`dict[key] = value`) even though the key doesn't exist since Python is smart enough to add it in there for you. However, if you're purely accessing a value at a key that doesn't exist, you will get an error. Ensure that your dictionary has the key you want to access.

In [ ]:
```
# Example E-C5: Missing dictionary key
scores = {"Anthony": 1}

print(scores["TAO"])
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[68], line 4
      1 # Example E-C5: Missing dictionary key
      2 scores = {"Anthony": 1}
----> 4 print(scores["TAO"])

KeyError: 'TAO'
```

'tuple' object doesn't support item assignment (TypeError)

You can't cannot change the structure or the size of a tuple. Note how I didn't say you can't change the values of a tuple. This is because, you can change mutable items within the tuple. This distinction is shown in Example E-C7: I can access the inner list and change the values within it (without changing the size), but I cannot do anything that affects the tuple structure wise or size wise.

The output shows that the inner list was indeed modified but attempting to change the structure (swapping out the string for another one) is unacceptable. Imagine a tuple as a max-security, max-capacity prison: you can't exchange prisoners, take prisoners out, or even put prisoners in. However, you're free to edit each prisoner individually: maybe give them different clothes, a new haircut, whatever. Structurally, the prison is the same and the size of the prison is the same as well, so you have not violated any rules.

In [ ]:
```
# Example E-C6: Trying to change a tuple
data = (1, 2, 3)

data[1] = 4
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[69], line 4
      1 # Example E-C6: Trying to change a tuple
      2 data = (1, 2, 3)
----> 4 data[1] = 4

TypeError: 'tuple' object does not support item assignment
```

In [ ]:

```python
# Example E-C7: Changing mutable item

data = ([1, 2], "1")

data[0][0] = 0
print(data)
data[1] = "2"
print(data)
```

```
([0, 2], '1')
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[70], line 7
      5 data[0][0] = 0
      6 print(data)
----> 7 data[1] = "2"
      8 print(data)

TypeError: 'tuple' object does not support item assignment
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js