The cell above loads the visual style of the notebook when run.

In [1]:
```
from IPython.core.display import HTML
css_file = '../styles.css'
HTML(open(css_file, "r").read())
```

Out[1]:

Errors and Exceptions

Learning Objectives

- To be able to read a traceback, and determine the following relevant pieces of information:
  - The file, function, and line number on which the error occurred
  - The type of the error
  - The error message
- To be able to describe the types of situations in which the following errors occur:
  - `SyntaxError` and `IndentationError`
  - `NameError`
  - `IndexError`
  - `IOError`

Every programmer encounters errors, both those who are just beginning, and those who have been programming for years. Encountering errors and exceptions can be very frustrating at times, and can make coding feel like a hopeless endeavour.

However, understanding what the different types of errors are and when you are likely to encounter them can help a lot. Once you know *why* you get certain types of errors, they become much easier to fix.

Errors in Python have a very specific form, called a traceback (reference.html#traceback). Let's examine one:

In [2]:
```
'''the next two lines add the 'code' directory
   to the list of places Python looks for libraries'''
import sys
sys.path.append('code')

# now we import the errors01 library, which is in the code directory
import errors01

errors01.favorite_ice_cream()
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-2-e1fe763e3862> in <module>
      7 import errors01
      8
----> 9 errors01.favorite_ice_cream()

~\OneDrive\UF\Obs Tech 1\StuartLittlefair-python-bootcamp-74e4fc9\pybootcamp-pa
rt2\code\errors01.py in favorite_ice_cream()
      5         "strawberry"
      6     ]
----> 7     print (ice_creams[3])

IndexError: list index out of range
```

This particular traceback has two levels. You can determine the number of levels by looking for the number of arrows on the left hand side.

In this case:

1. The first shows code from the cell above, with an arrow pointing to Line 2 (which is `favorite_ice_cream()`).
2. The second shows some code in another function (`favorite_ice_cream` , located in the file `errors_01.py`), with an arrow pointing to Line 7 (which is `print ice_creams[3]`).

The last level is the actual place where the error occurred. The other level(s) show what function the program executed to get to the next level down. So, in this case, the program first performed a function call (reference.html#function-call) to the function `favorite_ice_cream` . Inside this function, the program encountered an error on Line 7, when it tried to run the code `print ice_creams[3]` .

📌 Long tracebacks

> Sometimes, you might see a traceback that is very long -- sometimes they might even be 20 levels deep! This can make it seem like something horrible happened, but really it just means that your program called many functions before it ran into the error. Most of the time, you can just pay attention to the bottom-most level, which is the actual place where the error occurred.

So what error did the program actually encounter?

In the last line of the traceback, Python helpfully tells us the category or type of error (in this case, it is an `IndexError`) and a more detailed error message (in this case, it says "list index out of range").

If you encounter an error and don't know what it means, it is still important to read the traceback closely. That way, if you fix the error, but encounter a new one, you can tell that the error changed. Additionally, sometimes just knowing *where* the error occurred is enough to fix it, even if you don't entirely understand the message.

If you do encounter an error you don't recognize, try looking at the official documentation on errors (http://docs.python.org/2/library/exceptions.html). However, note that you may not always be able to find the error there, as it is possible to create custom errors. In that case, hopefully the custom error message is informative enough to help you figure out what went wrong.

Syntax Errors

When you forget a colon at the end of a line, accidentally add one space too many when indenting under an `if` statement, or forget a parentheses, you will encounter a syntax error (reference.html#syntax-error). This means that Python couldn't figure out how to read your program. This is similar to forgetting punctuation in English:

> this text is difficult to read there is no punctuation there is also no capitalization why is this hard because you have to figure out where each sentence ends you also have to figure out where each sentence begins to some extent it might be ambiguous if there should be a sentence break or not

People can typically figure out what is meant by text with no punctuation, but people are much smarter than computers. If Python doesn't know how to read the program, it will just give up and inform you with an error. For example:

In [ ]:
```
def some_function()
    msg = "hello, world!"
    print (msg)
     return msg
```

Here, Python tells us that there is a `SyntaxError` on line 1, and even puts a little arrow in the place where there is an issue. In this case the problem is that the function definition is missing a colon at the end.

Actually, the function above has *two* issues with syntax. If we fix the problem with the colon, we see that there is *also* an `IndentationError`, which means that the lines in the function definition do not all have the same indentation:

In [ ]:
```
def some_function():
    msg = "hello, world!"
    print (msg)
     return msg
```

Both `SyntaxError` and `IndentationError` indicate a problem with the syntax of your program, but an `IndentationError` is more specific: it *always* means that there is a problem with how your code is indented.

📌 Long tracebacks

> A quick note on indentation errors: they can sometimes be insidious, especially if you are mixing spaces and tabs. Because they are both whitespace (reference.html#whitespace), it is difficult to visually tell the difference. The IPython notebook actually gives us a bit of a hint, but not all Python editors will do that. In general, avoid mixing tabs and spaces!

Variable Name Errors

Another very common type of error is called a `NameError`, and occurs when you try to use a variable that does not exist. For example:

In [ ]:
```
print (a)
```

Variable name errors come with some of the most informative error messages, which are usually of the form "name 'the_variable_name' is not defined".

Why does this error message occur? That's harder question to answer, because it depends on what your code is supposed to do. However, there are a few very common reasons why you might have an undefined variable. The first is that you meant to use a string (reference.html#string), but forgot to put quotes around it:

In [ ]:
```
print (hello)
```

The second is that you just forgot to create the variable before using it. In the following example, `count` should have been defined (e.g., with `count = 0`) before the for loop:

In [ ]:
```
for number in range(10):
    count = count + number
print ("The count is: " + str(count))
```

Finally, the third possibility is that you made a typo when you were writing your code. Let's say we fixed the error above by adding the line `Count = 0` before the for loop. Frustratingly, this actually does not fix the error. Remember that variables are case-sensitive (reference.html#case-sensitive), so the variable `count` is different from `Count`. We still get the same error, because we still have not defined `count`:

In [ ]:
```
Count = 0
for number in range(10):
    count = count + number
print ("The count is: " + str(count))
```

Item Errors

Next up are errors having to do with containers (like lists and dictionaries) and the items within them. If you try to access an item in a list or a dictionary that does not exist, then you will get an error. This makes sense: if you asked someone what day they would like to get coffee, and they answered "caturday", you might be a bit annoyed. Python gets similarly annoyed if you try to ask it for an item that doesn't exist:

In [ ]:
```
letters = ['a', 'b', 'c']
print ("Letter #1 is " + letters[0])
print ("Letter #2 is " + letters[1])
print ("Letter #3 is " + letters[2])
print ("Letter #4 is " + letters[3])
```

Here, Python is telling us that there is an `IndexError` in our code, meaning we tried to access a list index that did not exist.

File Errors

The last type of error we'll cover today are those associated with reading and writing files: `IOError`. The "IO" in `IOError` stands for "input/output", which is just a fancy way of saying "reading/writing". If you try to read a file that does not exist, you will recieve an `IOError` telling you so. This is the most common reason why you would receive `IOError`, and if the error messages says "no such file or directory", then you know you have just tried to access a file that does not exist:

In [ ]:
```
file_handle = open('myfile.txt', 'r')
```

One reason for receiving this error is that you specified an incorrect path to the file. For example, if I am currently in a folder called `myproject`, and I have a file in `myproject/writing/myfile.txt`, but I try to just open `myfile.txt`, this will fail. The correct path would be `writing/myfile.txt`. It is also possible (like with `NameError`) that you just made a typo.

Reading error messages

Read the traceback below, and identify the following pieces of information about it:

Write your answers here

1. The trace back has three levels
2. The error occurd with the file errore_02.py
3. The error occurd with the function 'print_friday_message()'
4. The error occurd at line 11 where 'print messages[day]' is
5. The error is a Key error meaning that we attempted to access an item that doesn't exist with our indexing syntax
6. The error message is 'KeyError: 'Friday'

Reading error messages

1. Read the code below, and (without running it) try to identify what the errors are.
2. Run the code, and read the error message. Is it a `SyntaxError` or an `IndentationError`?
3. Fix the error.
4. Repeat steps 2 and 3, until you have fixed all the errors.

```
    def another_function
      print ("Syntax errors are annoying.")
       print ("But at least python tells us about them!")
      print "So they are usually not too hard to fix."
```

In [14]:
```
# Run the code here

def another_function():
    print ("Syntax errors are annoying.")
    print ("But at least python tells us about them!")
    print ("So they are usually not too hard to fix.")
```

1. The first error is that there is not a ':' after the function and no ()s after the name after the function. Then, there will be an error that the print statements aren't properly indexed. The last error will be that there are no ()s around the last print statement.
2. The first is a syntax error . The second is an indication error. The third is a syntax error
3. To fix the first error, add '():' after the function name. To fix the second error, line up the print statements with proper indentation. To fix the third, add ()' around the print statemen

Finding variable name errors

1. Read the code below, and (without running it) try to identify what the errors are.
2. Run the code, and read the error message. What type of `NameError` do you think this is? In other words, is it a string with no quotes, a misspelled variable, or a variable that should have been defined but was not?
3. Fix the error.
4. Repeat steps 2 and 3, until you have fixed all the errors.

```
for number in range(10):
    # use a if the number is a multiple of 3, otherwise use b
    if (Number % 3) == 0:
        message = message + a
    else:
        message = message + "b"
print message
```

In [24]:
```
# Run the code here

message = 0
a = 0
b = 0

for number in range(10):
    # use a if the number is a multiple of 3, otherwise use b
    if (number % 3) == 0:
        message = message + a
    else:
        message = message + b
print(message)
```

```
0
```

1. a) There will be an error because there are no ()s around the print statement.

b)'number' is capitalized in the if statement.

c) There will be an error because message was never defined and we are trying to use it to define its self.

d) There will be an error because a is not defined

e) There will be an error because b is called as a string

f) There is an error because b isn't defined

2. a) The first error message is a syntax error fixed by adding ()s.

b) The next error is due too the mispelled varible and is a name error.

c)The third error is a namme error because the varible wasn't defined.

d) This is a name error due to a not being defined

e) This is a type error because b is being treated like a string and we are working with integers

f) name error because b isn't defined

In [ ]: