Lecture 17: Errors and debugging

Errors

Errors are a common source of problems for a programmer. These can either be caused by a mistake of the program writer, or eventually more commonly, the mistake of the user (you or someone else). Luckily Python has a built-in mechanism to handle errors that allows you to handle mistakes on the fly. Let's study what happens when you make some common errors.

Error 1: Try to access a key in a dictionary that doesn't exist.

```
>>> my_dict = {}
>>> #Try to access a key that doesn't exist
...
>>> my_dict['New key']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'New key'
```

Error 2: Try to access an element in a list that doesn't exist.

```
>>> my_list = [1,2,3]
>>> my_list[3]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

Error 3: Try to add a string to an integer.

```
>>> '6' + 3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```

In all cases, Python throws what is known as an exception. Each error threw a different type of exception: KeyError, IndexError, and TypeError. All of the different types of exceptions can be found here: (https://docs.python.org/2/library/exceptions.html). Now the normal behavior is for Python to quit running the script when an exception occurs but you can override this by what's called "handling" the exception. We do this using try/except statements. Let's see how it's done using KeyErrors in dictionaries.

Dictionaries are useful ways to calculate histograms. Histograms count how many times something occurs. Let's say we have a string of DNA sequence ('ACGGCGAAGTGGCAGTGA') and we want to know how many times each nucleotide is found. We might try to do this as so:

```
>>> nuc_count = {}
>>> DNA = 'ACGGCGAAGTGGCAGTGA'
>>> for letter in DNA:
...     nuc_count[letter] += 1
...
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
KeyError: 'A'
```

The problem is that we first have to initialize key in the dictionary before we try to access it. We can fix the following code using try/except statements like so:

```
>>> nuc_count = {}
>>> DNA = 'ACGGCGAAGTGGCAGTGA'
```

```
>>> for letter in DNA:
...     try:
...         nuc_count[letter] += 1
...     except KeyError:
...         nuc_count[letter] = 0
...         nuc_count[letter] += 1
...
>>> nuc_count
{'G': 8, 'A': 5, 'C': 3, 'T': 2}
```

What we've done is added a try statement before attempting to increment the nuc_count[letter] dictionary. What this does, is allow us to handle any errors that arise from this line of code (actually any code within the try statement) using the except statement. We specifically handle issues arising from a KeyError (by adding KeyError after the except statement. If the KeyError arises, the code within the except statement then executes.

So in this case, we take the first letter of the DNA string ('A') and attempt to do: nuc_count['A'] += 1. Since 'A' doesn't exist in the dictionary yet, a KeyError is thrown. However, instead of quitting the script, the except statement 'catches' the KeyError and then executes nuc_count['A'] = 0 to initialize the element in the dictionary. Now the increment command can work.

Note, that the only error that gets handled is the KeyError. If another Error is thrown it will kill the program. For example, let's say we never created nuc_count as a dictionary and see what happens:

```
>>> DNA = 'ACGGCGAAGTGGCAGTGA'
>>> for letter in DNA:
...     try:
...         nuc_count[letter] += 1
...     except KeyError:
...         nuc_count[letter] = 0
...         nuc_count[letter] += 1
...
Traceback (most recent call last):
  File "<stdin>", line 3, in <module>
NameError: name 'nuc_count' is not defined

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "<stdin>", line 5, in <module>
NameError: name 'nuc_count' is not defined
```

Because nuc_count wasn't declared, we get a NameError (which is different than a KeyError). However, we could also add an additional except statement to handle this type of error:j

```
>>> for letter in DNA:
...     try:
...         nuc_count[letter] += 1
...     except KeyError:
...         nuc_count[letter] = 0
...         nuc_count[letter] += 1
...     except NameError:
...         nuc_count = {}
...         nuc_count[letter] = 0
...         nuc_count[letter] += 1
...
>>> nuc_count
```

You can also add an else statement at the end of the except statements. This will execute if and only if the try statement doesn't throw an error.

```
>>> for letter in DNA:
...     try:
...         nuc_count[letter] += 1
...     except KeyError:
...         nuc_count[letter] = 0
...         nuc_count[letter] += 1
...     except NameError:
...         nuc_count = {}
...         nuc_count[letter] = 0
...         nuc_count[letter] += 1
...     else:
...         print("No error occurred!")
...
```

You can also raise your own errors if you would like:

```
raise NameError("This wasn't declared", 'foo', 'bar')
```

Debugging

One of the most effective tools for debugging scripts is the pdb module. This is a very simple debugging library that allows you to enter the interpreter at specific points of your script to determine if your code is doing what you think it should be doing.

An example script is found that we will go through as a basic example.