

Maybe exceptions are not so awesome after all

admin · Saturday, August 15th, 2009

Maybe I got a bit rusty on my 1 month vacation, but today I ran into a simple problem that I spent 2 hours solving. I Am working on a website made in Django and I decided to make some changes to the model in one of the apps in the project. I currently have 3 apps - game, content and members. One of the classes of the content model imports a class from the game model, but I decided to remove that particular class.

After syncing the DB I noticed that the database tables for the content app are missing. This was strange, because no errors were reported. I tried to do a "python manage.py sqlall content" and the shell threw an error that it couldn't find the content app on my PYTHONPATH.

Error: App with label content could not be found. Are you sure your INSTALLED_APPS setting is correct?

I tried all kinds of different stuff until i found a mailing list that said that if you have import errors in some module, you cannot import it, which is to be expected, but the error messages that come up are all but informative. The message that my PYTHONPATH is incomplete in no way suggests that I have an import error.

What does this have to do with exceptions ?

Well, I'll talk about another example first: While I was writing a screen scraper, I used a lot of regular expressions. The regular expressions return a match object if they successfully make a match and None if they don't. If you try and call the .group() method on a Match object - it does the job, but calling it on a None object throws and AttributeError. I used to catch this exception in the upper layers of my program in order to avoid crashing the script on invalid input. This turned out to be a bad idea since all kinds of other stuff throws an AttributeError and makes the debugging a living hell.

I suspect that the same thing is going on in Django. Failure to import inside a module throws an ImportError which Django interprets as a missing module - only the module isn't missing - its a "submodule" that is missing. But the exceptions are the same, no matter on which level they happen.

Using exceptions correctly requires defining your own classes for every particular error that you may run into. This is a lot of extra code and need's to be weighed against the old-fashioned error codes.

Posted in [Django](#), [Programming](#), [Python](#) | [No Comments](#) »

A recursive django template tag

admin · Friday, June 5th, 2009

Here is my attempt to create a "silver bullet" tag for printing tree structures with the Django templating language. It's far from a silver bullet, tho, but it can do basic stuff:

It's a modification of the standard "for" tag and i have kept the counter, counter0, first and last variables, only this time they are not attributes to forloop, but rather to recurseloop. Check the docstring for more info.

For example, if you need to print comments that have other comments as replies, you would want the comments to appear one below the other, but the replies to be indented a bit. Let's say 20 pixels per level. So the code would be:

```
{% load file_that_contains_recurse_tag %}
{% recurse comment in comments children="replies" indent=(0,20) %}
    {{ comment.text }}
{% endrecurse %}
```

This tag will expect a list of comments (top-level) that have a property named 'replies' which contain other comments.

indent is an argument that will start with the float value of 0 and get increased by 20 on each depth level of the recursion. You can pass as many variables like indent as you like. They must be in the form

```
name=(float,float)
```

or strings will also work but must be enclosed in quotes

```
name=("string","string")
```

Caveat: You must not leave blank spaces between the equal signs when assigning children and additional incremented arguments. I will fix this later.

A second scenario is when you need the parent element to **contain** the children, like in unordered/ordered lists. In that case you can use the `{% yield %}` tag inside the recurse block. This tag will output the HTML between the recurse and endrecurse tags if there are any children in the current iteration item, or it will output nothing if there are no children.

```
{% recurse comment in comments children="replies" indent=(0,20) %}
    {{ comment.text }}
    {% yield %}
{% endrecurse %}
```

The yield tag (as for now) can **only** be used directly inside the recurse block, much like the `{% else %}` tag can only be used directly inside the if-endif block. This means that you can't make code like

```
{% recurse comment in comments children="replies" indent=(0,20) %}
    ({{ comment.text }})
    {% if cond %}
        {% yield %}
    {% endif %}
{% endrecurse %}
```

This will fail with an invalid block tag exception. You can check the docstring of the `do_recurse` function inside the code for more info. Also, you may want to check this code for errors since I just wrote it today, and haven't had much time to test it.

[Download and comment on any errors you find :\)](#)

Posted in [Django](#), [Programming](#), [Python](#) | [4 Comments »](#)

Maze Drawer

admin · Wednesday, April 1st, 2009

This piece of python code is going to draw some mazes in a .PNG format. It's a side-product of the solution to the [Google Code Jam](#) practice problem B called "Always turn left". The script takes the input data sets and draws the mazes described with every test case. You will need Python with Python Imaging Library (PIL) installed. If you're working on OS X like me, check out [this link](#) for some tips on how to install PIL.

To run the script type "python left.py" and it will ask for a file path containing the test cases. It will create a png image for each test case in the directory from which the program is run.

[Download script + data set and make some mazes](#)

Posted in [Programming](#), [Python](#) | [2 Comments »](#)

Some problem solving and how it's easier with python

admin · Monday, March 23rd, 2009

There's a project I work on that required me to make an import utility for a CRM. The import should get a comma separated values file of clients and information about clients, and save it to the database. The database is split across several tables, so in the `clients` table I normally don't keep the name of the company, but just a foreign key. Now, our client is not very good with numbers and she needed to import files in which she could enter the name of the company instead of the database ID. A spreadsheet row representing a client looks like this:

```
FirstName | LastName | Email          | Company
John      | Doe      | johndoe@example.com | Coca Cola
```

But the database row in the `clients` table looks like this:

```
first_name | last_name | email          | company_id
John       | Doe      | john@example.com | 2
```

What I need to do is search for the company named 'Coca Cola' in the `companies` table and replace the name with it's ID. This is all fine except for one problem - typos. Moreover, the user could write "Apple Computer Inc." instead of "Apple Inc.". So I needed a way to compare the input strings with the ones in the database.

After poking around I found out about the [Levenshtein distance](#) between strings, but that solved only half of my problems - the typo part. The distance would be very small between "Apple" and "Aple" but very big between "ACME International Inc." and "International ACME Inc.", and the latter two are obviously the same.

I devised the following method to compare entries:

- Split up the terms by words and eliminate blanks
- Get the Levenshtein distance between each word from the first term and each word from the second term. Comparing "Apple Computer Inc." with "Apple Inc." for example, will give a matrix of 6 distances. ☒
- Get the shortest term (one with less words, not the one with less characters). It has 2 words in this case. Then choose the smallest values from each **row**. When you pick the smallest **row** value, you cannot pick anymore values from that **column**. This means that the word in the **column** is the best match for some word in the **rows**.
- Add these values up and add the difference between the word count of the 2 terms - and you have a score for the similarity of the terms. If the score is zero, they are the same. We are adding +1 for each extra word, but this can be weighted if needed. The point is that we don't care much for extra words since company names can have many words in them, but they are often called by one or two words.

But there is a problem with step 3. If, for example, a column has the lowest values for more than one row, we always choose the first, and this practice is not always the best answer. For instance, matching "Fast Cats" with "Fats Cats" (notice the typo) gets a total score of 3 - matching **Cats** to **Fats** and **Fast** to **Cats**, which is wrong - it will be 2 if we match **Fast** to **Fats** and **Cats** to **Cats**, which is the intended solution.



So to be sure we have the best match, we need to always have the lowest sum that is unique across rows and columns. One solution is to make all permutations of the words in the **columns** and join them to a single permutation of the words **in the rows** then see which one has the lowest score. If the words in the rows are fewer then we need to get all permutations **P(n,k)** of the words in **the columns**, where **n** is the number of columns and **k** is the number of rows. This is a $O(n!)$ algorithm but it's the best that I could think of - practically the same problem as finding every possible way to place 8 rooks on a chess table without making them attack each other.

And finally, here is the part where we get to write some code. I need a function that can calculate all permutations consisted of **k** elements out of a larger set consisted of **n** elements (**k** <= **n**).

I decided first to write the algorithm in Python because it's cleaner and easier to think, and then to rewrite it in PHP. The first attempt was really, really sucky and I won't talk about it because I'm a bit embarrassed. But I wasn't aware of a neat thing that Python has: the **yield** statement. The darn thing can be written in 6 lines with it:

```
def permutations(the_set, n):
    if n==0:
        yield []
    else:
        for i in xrange( len( the_set ) ):
            for x in permutations
            ( the_set[0:i] + the_set[i+1:], n-1 ):
```

```
yield [the_set[i]]+x
```

I will go into the yield statement later, maybe I will extend this post, but for now, I'll say that it allows you to make a function that will calculate the combinations on the fly, without storing them in a huge list and then returning the list. It sort of lazy-loads the list of combinations when needed. There is no such thing in PHP (as far as I know). So here's my best shot at the function in PHP:

```
function permutations( $array, $size )
{
    $result = array();
    $x = count($array);
    for( $i=0; $i<$x; $i++ ) {
        $copy = $array; // copy: array_splice gets the arg by reference
        $item = array_splice( $copy, $i, 1 );
        if( $size == 1 )
            $result[] = $item;
        else {
            $rest = permutations( $copy , $size - 1 );
            foreach( $rest as $r )
                $result[] = array_merge($item, $r);
        }
    }
    return $result;
}
```

There really are excessive parts of the PHP code like storing the final result, but more importantly copying the array each time because `array_splice` takes the array argument by reference and modifies it (talking about orthogonality), plus its twice as long as the python code and half as readable.

Anyway, to get back at my original problem - the solution worked in terms of accuracy (at least for the first few test cases), but I fear it's going to be slow for large datasets. I have around 7 fields to compare with each respective table of the database, each table having 100 records on average; each record is 3 words long on average which gives 6 permutations per comparison. Importing a list of 1000 clients would require $1000*7*100*6 = 4,200,000$ comparisons, plus 700,000 calls to the permutations function (not counting the recursive calls :). I still think that it's better than hammering the database with 7000 fulltext search queries, not to mention moving the database tables to MyISAM and indexing a bunch of fields. After all, it's an import. I could put one of those useless progress indicators like when you're starting Windows.

Posted in [PHP](#), [Programming](#), [Python](#) | [No Comments](#) »

Get your copy of bronze framework

admin · Wednesday, February 25th, 2009

Bronze framework is a **PHP MVC** framework for building web applications. It was inspired by many other frameworks that can be found around. It uses a system of internal redirects and a recursive front controller to allow the developer to reuse code as much as possible, and avoid repetition.

For now, the only thing you can get is the source and consult :) I need some time to write proper documentation.

Anyway, download [here](#), and please comment on it [here](#)

Posted in [Programming](#) | [No Comments](#) »