Python Fu #4, Hello Warning

September 14, 2015November 30, 2015 | jacksonbates | gimp, gimp scripting, hello world, python, python-fu, tutorial
See the other tutorials in this series. (https://jacksonbates.wordpress.com/python-fu-gimp-scripting-tutorial-pages/)

In the last three tutorials we looked at the very basics of Python, just variables and functions, and then we added a little more Python knowledge with the basics of modules and why that is relevant to the GIMP and the Procedure Database. If all of that is nonsense to you, you better go back and check the last few tutorials, because this next one won't make much sense. The code explained in this tutorial can be downloaded here: registration template.py (https://gist.github.com/JacksonBates/cd98675d58403d295dee) and hello warning.py (https://gist.github.com/JacksonBates/4d73ac7c075516e3f427).

In this tutorial we are going to write a scipt that simply prints ‘Hello, World!’ to the error console. This is more or less pointless, but it will teach us three things: How to register a script, where to find the error console, and how to test some of you assumptions using the error console – this last one is very helpful as you start to develop you own plug-ins and need to do some detective work to get them working.

You will also notice that for this tutorial I will be using a particular text editor. You could use Notepad on Windows, but getting a text editor designed for programmers is much more helpful because they highlight key words and can show you where brackets are closed. These features, and many others, can make it much easier to spot errors and debug your code. On Windows you could use Notepad++, which is very easy to use. If you download Python, it comes with it’s own text editor, called IDLE. I’ll be using IDLE for the rest of these tutorials.

The first thing we will look at is registering the script. This has to be done in every plug-in we write, so it can be helpful to produce yourself a little template that you can always refer to, like this one:

```python
#!/usr/bin/env python

# Tutorial available at: https://www.youtube.com/watch?v=nmb-0KcgXzI
# Feedback welcome: jacksonbates@hotmail.com

from gimpfu import *

def NAME_OF_MAIN_FUNCTION(image, drawable):
    # function code goes here...


register(
    "python-fu-NAME-OF-MAIN-FUNCTION",
    "SHORT DESCRIPTION",
    "LONG DESCRIPTION",
    "Jackson Bates", "Jackson Bates", "2015",
    "NAME FOR MENU",
    "", # type of image it works on (*, RGB, RGB*, RGBA, GRAY etc...)
    [
        # basic parameters are: (UI_ELEMENT, "variable", "label", Default)
        (PF_IMAGE, "image", "takes current image", None),
        (PF_DRAWABLE, "drawable", "Input layer", None)
        # PF_SLIDER, SPINNER have an extra tuple (min, max, step)
        # PF_RADIO has an extra tuples within a tuple:
        # eg. (("radio_label", "radio_value), ...) for as many radio buttons
        # PF_OPTION has an extra tuple containing options in drop-down list
        # eg. ("opt1", "opt2", ...) for as many options
        # see ui_examples_1.py and ui_examples_2.py for live examples
    ],
    [],
    NAME_OF_MAIN_FUNCTION, menu="")  # second item is menu location

main()
```

It might look confusing to begin with, but there is nothing here, apart from the first line, that we shouldn’t already recognize.

```
#!/usr/bin/env python
```

The first line begins with a shebang, the !# symbol, and then a directory path. You don’t have to worry too much about this. It’s a UNIX thing that allows a script to run in the correct environment. You don’t always need it – but get in the habit of using it, in case you ever start programming more seriously. Then we have some commented out lines which we will ignore.

```
from gimpfu import *
```

The next line is an import statement that allows us access to the PDB, among other things.

```
def NAME_OF_MAIN_FUNCTION(image, drawable):
```

Next up we have our main function. The filename of the main function, and the later references to it should all match. More on that in a minute. You can see that I’ve already said this function has two arguments, or parameters that it needs: image and drawable. This type of function works on an existing image, and as such always needs to have these two arguments first. It can have others too, but these two go first and can be skipped. The image variable is obvious. The drawable refers to the active layer. GIMP thinks of images kind of like containers that can’t be directly ‘drawn’ on, that contain things that can be drawn on, drawables! In this case the drawable is the layer. Channels are drawables too, but we canignore that for now.

When we know what our function will do, obviously we fill the rest in.

```
register(
    "python-fu-NAME-OF-MAIN-FUNCTION",
    "SHORT DESCRIPTION",
    "LONG DESCRIPTION",
    "Jackson Bates", "Jackson Bates", "2015",
    "NAME FOR MENU",
    "", # type of image it works on (*, RGB, RGB*, RGBA, GRAY etc...)
    [
        # basic parameters are: (UI_ELEMENT, "variable", "label", Default)
        (PF_IMAGE, "image", "takes current image", None),
        (PF_DRAWABLE, "drawable", "Input layer", None)
        # PF_SLIDER, SPINNER have an extra tuple (min, max, step)
        # PF_RADIO has an extra tuples within a tuple:
        # eg. (("radio_label", "radio_value), ...) for as many radio buttons
        # PF_OPTION has an extra tuple containing options in drop-down list
        # eg. ("opt1", "opt2", ...) for as many options
        # see ui_examples_1.py and ui_examples_2.py for live examples
    ],
    [],
    NAME_OF_MAIN_FUNCTION, menu="")  # second item is menu location
```

Next we have another function called register. It may not look like a function because of the strange layout, but if you ignore all the whitespace and linebreaks, you can see that this is a function with lots of arguments. In Python we use whitespace like this to make our code more readable. Python ignores this kind of white space, so it doesn’t effect the way it runs.

The register function takes the following arguments or parameters:

- The name of the script – which by convention begins python-fu, if written in Python (script-fu if written in Scheme)
- A short description of what it does.
- A longer description / help string.
- The author’s name
- Copyright info

- Copyright year
- The name to be printed in the menu
- The image type
- A list of parameters – I’ll explain a little more about lists below
- Results – allegedly, I haven’t found this being used in the wild yet, I’ve always just seen it blank!
- The name of the main plug in function
- The menu location

```
main()
```

The final thing in our script here is a call to the main() function. This simply tell GIMP to run the main function. We need to do this because even though we defined a function earlier, we didn’t call upon it to actually run. More complicated scripts also have multiple functions defined in them, so call the main one is what triggers the plug-in. The main function might call upon those other functions depending on input.

So finally we need to add something to the function to make it do something.

```
def hello_warning(image, drawable):
```

We will change the name of the function to hello_warning(), and need to replace that in three places in the register arguments.

Next we will fire up the PDB to see what the Error console message function is:
pdb.gimp_message(message)

```
pdb.gimp_message("Hello, world!")
```

So we change the argument to the string of text we want. and save it. Now we need to close and reopen GIMP, because the first time we register a plug-in, GIMP needs to restart. From then on though, every time we save the script the changes flow through automatically.

Now that we’ve restarted GIMP, we need to get the error console up, which can be pulled from any tools dialog – I like to use the layer / channels / undo / paths docking area for mine. Hit the little triangle, add a tab, and select error console at the bottom.

Now open an image, because this plug in supposedly runs on an image, and then go to the menu location that you registered the plug-in for. In this case, simply ‘File.’

So when we run the plug in we get the “Hello, world!” message in the console.

So that’s pretty pointless, but we can actually make this quite useful for us by feeding a different argument into the message. For example, when we imported everything from gimpfu, we loaded lots of extra information that is useful for us to know.

Just like we can access math.pi, that is a variable called pi that belongs to the math module, we can also access things like image.name, the name variable belonging to the image. So replace the hello world argument in the message function with image.name – notice this is not a string, so no speech marks, it’s a specific variable I’m calling.

The error console helpfully tells us the name of the image.

We can also access things like image.width. Make the changes to you code and see what that does. This particular variable, and image.height is very useful – you’ll likely use it lots when you begin making your own plug-ins.

There is also a slightly more complicated thing we can access, image.layers. Now, my original image only has one layer, but let’s make another and see what happens when I ask for this information. So I’ll make a blank layer and call it Booyah! and then call for the message image.layers. Now the variables we have seen so far have all had a single value. But if we ask for this variable, it will have two values.

Python deals with this by creating a list. A list is like a variable in the way it is set up, but obviously it contains a list of items. We can access the whole list or just one at a time. The parameters we set up in the register earlier were in a list, and we can tell because they were put in square-brackets – this is how we set a list in Python. I’ll quickly show you that in the Python console:

```
>>>a = [1, 2, 3, 4, 5]
```

Here we create a list of integers, 1 through 5.

I can print the whole list:

```
>>>print a
[1, 2, 3, 4, 5]
```

Or I can print elements from the list:

```
>>>print a[0]
1
```

Notice that I ask for the **Zeroth** item to get what we think of as the first item! This is how computers count – **they start at zero**. If I want to retrieve the number 2 from the list I:

```
>>>print a[1]
2
```

It's confusing to begin with, but you get used to it.

This principle is useful to know though, because if you have to work on a specific layer in the image list, you need to know how to access it and how to refer to it's position.

Here is what the final hello warning script looks like:

```
#!/usr/bin/env python

# Tutorial available at: https://www.youtube.com/watch?v=nmb-0KcgXzI
# Feedback welcome: jacksonbates@hotmail.com

from gimpfu import *

def hello_warning(image, drawable):
    # function code goes here...
    pdb.gimp_message("Hello, world!")


register(
    "python-fu-hello-warning",
    "Hello world warning",
    "Prints 'Hello, world!' to the error console",
    "Jackson Bates", "Jackson Bates", "2015",
    "Hello warning",
    "", # type of image it works on (*, RGB, RGB*, RGBA, GRAY etc...)
    [
        (PF_IMAGE, "image", "takes current image", None),
        (PF_DRAWABLE, "drawable", "Input layer", None)
    ],
    [],
    hello_warning, menu="/File")  # second item is menu location

main()
```

Anyway, that’s a lot of detail for a simple hello world tutorial.

In the next tutorial we’ll make a more functional, but just as simple plug-in that actually edits a photo, which is what GIMP is for after all.

Related posts: Python Fu GIMP tutorials (https://jacksonbates.wordpress.com/python-fu-gimp-scripting-tutorial-pages/)

6 thoughts on "Python Fu #4, Hello Warning"

NAP0 SAYS: Thanks for the tutorials
In the final warning script I had to change the last parameter of the register function to
menu="/File"
in stead of
menu="/File"
1. to make it work.
March 17, 2016 at 7:47 am • Reply »
    1. **NAP0 SAYS:** apparently wordpress removes the part between angular brackets automatically thats wy my previous comment also doesn't make sense
    March 17, 2016 at 7:49 am • Reply »
    NAP0 SAYS: so the part that is missing is
    Image
        1. between angular brackets
        March 17, 2016 at 7:51 am
        JACKSONBATES SAYS: That's interesting – I remember I originally had trouble registering scripts following someone else's tutorial…the way I did it was the fix that worked for me at the time, but maybe there's some version issue that means the register function takes a slightly different form? I can't imagine why that would be the case, but I'll test my script later and amend the post of necessary. Thanks for picking up the potential bug.
        March 17, 2016 at 7:55 am
        2. **NAP0 SAYS:** ok thanks
        March 17, 2016 at 10:17 am
2. **PYTHON EN GIMP | NAP0'S BLOG SAYS:** […] Bijhorende Blog post: https://jacksonbates.wordpress.com/2015/09/14/python-fu-4-hello-warning/ […]
March 19, 2016 at 9:57 am • Reply »