# 1-Your first steps

June 21, 2017

## 1 Table of Contents



## 2 The Aim: Replace fear with a hammer

Broadly this crash course will show you some quick methods to analyze your images to get data and do some rough analysis.

A more subtle effect will be the **Removal of Fear** by getting you to a point where you know what's possible and you know what you don't know. **Knowing whats possible**, with even simple programs, allows one to look at everyday activities and delegate parts to a computer. **Knowing what you don't know** gives you the power of words to google for; once you know what to search for the answers are often a few clicks away.

So, shall we jump?

A quick outline: We will first lean some basics before going on to play with images. I've tried to keep the examples short with enough details so that you can explore more.

By the end you will be able to take images of nuclei (below) and convert them into graphs

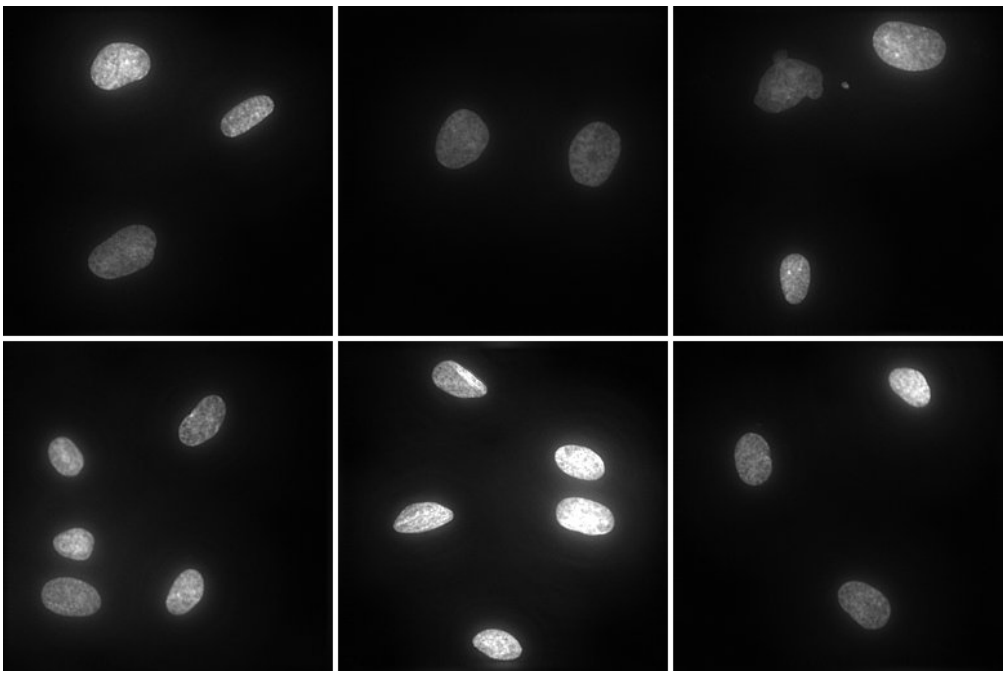
## 3 Say Hi!

If your setup has gone smoothly, you should have anaconda and Fiji Installed. Now create a folder (or directory) where you would like to store all the files related to this crash-course. Download the zip file from the following link, unzip the files into the directory which you just made.

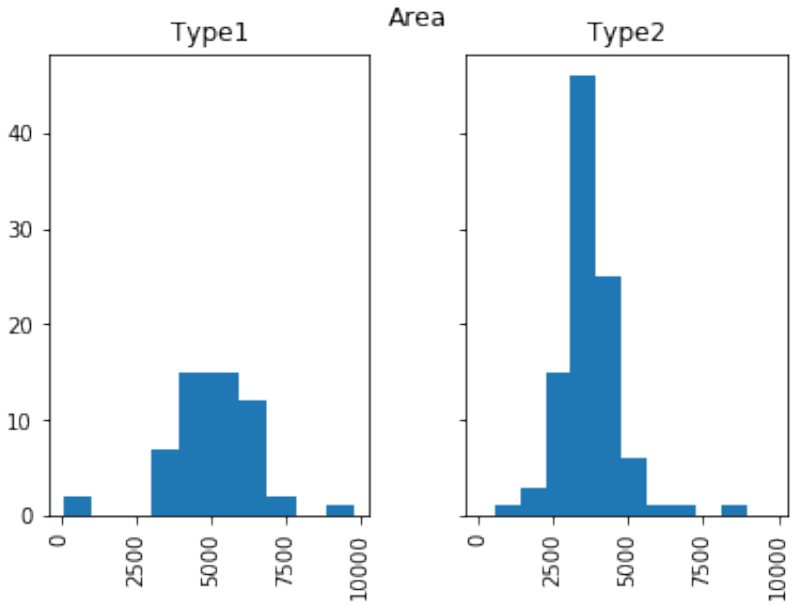
https://github.com/aneeshsathe/DataAndImageAnalysisForBiologists/archive/master.zip

Go ahead and explore the files, it contains the jupyter notebooks of the exercises which you will do, the images which you will analyze and some of the ideal outputs of your final program.

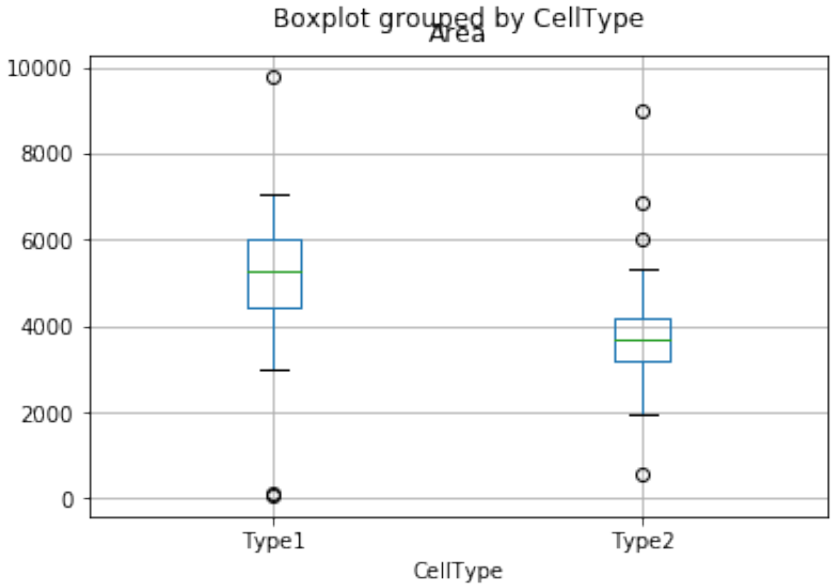
Good now your environment is all set up. It's time to say "Hi!".



Montage




HistogramArea




box_plot_area

In the directory you just created, create another directory, name it something like "MyPrograms" or something like that. You will put all the programs you will write into this directory, so give it a sensible name.

Ok? Now start your terminal or command line in this directory. If you don't know how to do this, google for "open terminal in directory" or "open cmd in folder" alternatively google for "navigate to directory from command line". This should give you commands like "cd /xyz/abc"

Once you are in your directory start a new jupyter notebook server by typing in "jupyter notebook" this should cause a browser window to open and you will see the Jupyter interface. You can explore that a bit if you want (google for notebook extensions if you would like to improve the usability).

On the right side of the interface, start a new notebook by clicking "New>Python" this might say Python3 or Python2 depending on which anaconda you installed.

A new notebook should start. This is where you will be spending most of your time. You should see a long grey box with `In [ ]:` on the left side.

Click on the grey box and type `print("Hello World")` followed by Ctrl+Enter or press the button on top that looks like the play symbol. YOu should see something like this:

```
In [2]: print("Hello World!")

Hello World!
```

Congrats! you have written your first little program. So what are you doing you ask? well the `print()` part of that statement is a function which tells the computer to print something on the screen. The *something* is enclosed in the quotation marks. Try printing something else, like your name maybe?

```
In [4]: print("Captain, Captain Jack Sparrow!")
```

```
Captain, Captain Jack Sparrow!
```

```
In [5]: print("Print this)
```

```
          File "<ipython-input-5-cab8052285d6>", line 1
        print("Print this)
                          ^
    SyntaxError: EOL while scanning string literal
```

If you played around a little bit, you might have encountered some scary red text. These are called errors, they are normally caused by the computer not understanding what you want it to do. For example I have created an error above by not typing the second quotation mark. This throws out a `SyntaxError` . *Syntax* is just a fancy way of saying the agreed upon rules of a language. The computer expects all quotes and brackets to be completed before running the code. The little ^ symbol tried to tell us where the problem might be. Can you fix the problem?

## 4 Can the silicon do mathematics?

Now that you've got a little feel for the ability of the computer to do your bidding let's step things up a tiny bit. Lets make the computer do Maths, afterall, it *is* a computer! Make it do `2+2`. Same as before, type `2+2` followed by Ctrl+Enter or the play button on top.

```
In [6]: 2+2
```

```
Out[6]: 4
```

It seems to do that ok... can you make it do other operations like divide? multiply?

This is all well and good but one can't be expected to always type out numbers that you want to do operations on. There is a way to make the computer store numbers you want (or really store anything you want) and make it do operations on that.

## 5 Variables

Variables are basically bags for your items. There isn't much of a point in trying to explain variables with words let's just play with them!

```
In [ ]: a = 3
        b = 4
        a+b
```

What would you expect the result of the above cell to be? Can you make the computer do other operations? Can you assign more than 2 variables and do operations with them?

Here's a tricky one:

```
In [3]: a = 'hello'
        b = 4
        a*b

Out[3]: 'hellohellohellohello'
```

As you can see, python interprets `a*b` as "repeat `a` `b` number of times. What happens if you try to add a and b? Can you imagine why?

As variables go, `a` and `b` are terrible names for variables, because if you read them tomorrow you will forget what they were for. So as a rule you should always give sensible names where possible.

Suppose we have the following problem: The average squirrel weighs 0.5 kg whats the weight of a thousand squirrels?

We should assign sensible names for the calculations as below:

```
In [5]: # Average Weight of Squirrels
        avg_squirl_wt = 0.5
        # Number of Squirrels
        no_of_squirl = 1000
        # Calculate and print total weight
        tot_weight = avg_squirl_wt*no_of_squirl
        print(tot_weight)

500.0
```

If you noticed above we have normal English mixed in with the code. Any letters following the # symbol is interpreted by Python as a **comment**. Comments are meant for human readers of the code and are completely ignored by the computer. Use comments generously, your future self will love you for leaving helpful notes along with your code :-)

Continuing the problem above.. suppose you had 3 different kinds of squirrels with three different average weights: TypeA: 0.3 kg, TypeB:0.1 kg and TypeC:50 kg. The three types are also present in 3 differnt numbers: TypeA: 10, TypeB: 20 and TypeC: 1 How would you find the total weight of a thousand squirrels of the different types?

```
In [9]: # write your answer
```

## 6 Boo! lean

The computer is also able to make comparisons and tell you if something is bigger, smaller, same. These type of operations are called boolean operations. Lets try out some

```
In [11]: big_number = 100
         small_number = 2
         print(big_number<small_number)
         print(big_number>small_number)
         print(big_number==small_number)
         print(big_number>=small_number)
```

```
False
True
False
True
```

If you noticed above we use the == symbol to ask the computer if the numbers are equal instead of the = symbol. This is because the = symbol is used for assigning variables their values. Can you guess what the >= symbol measured?

## 7 A list of lists of things that mean 3

What if you could store more than one thing in a variable? What if you could store more than one type of thing in a variable? Well you can! Python has a type of variable called a list which can do that. There are a few python data types which you should read about, broadly you have numeric, and strings, i.e. numbers and letters. But there can also be different types of numbers and different kinds of strings. Let's see how strings work:

```
In [16]: # assign the value 3 to 3 variables
         goat = guitar = eskimo = 3
         # declare and print list
         my_name_is_three = [[3,3,3], ['three', 'teen', 'tres', 'san'], [goat, guitar, eskimo ]]
         my_name_is_three

Out[16]: [[3, 3, 3], ['three', 'teen', 'tres', 'san'], [3, 3, 3]]
```

The `my_name_is_three` list above is actually a list of lists... wait what?? You read that right.. that list actually contains other lists and can also be constructed like this:

```
In [ ]: three_numbers = [3,3,3]
        three_langs =  ['three', 'teen', 'tres', 'san']
        goat = guitar = eskimo = 3
        three_variables = [goat, guitar, eskimo ]
        my_name_is_three = [three_numbers, three_langs, three_variables]
```

What if after storing, you wanted to only look at the first list in the list of lists? This can be done using `indexing`. Indexing means telling the computer to pull out a particular value at the coordinates we specify. Python uses 0-indexing. 0-indexing starts counting from 0 instead of 1. So to access the first element you have to type:

```
In [17]: my_name_is_three[0]

Out[17]: [3, 3, 3]
```

We will explore this property of lists as we go on. For now lets actually jump into accessing our images!

## 8 Read my files

If you explore the files you unzipped in the beginning, you will notice that the `/Images/Source/` path contains two folders: `Type1` and `Type2` each of these directories contains images taken from two different types of cells that we will use for our analysis.

We know where these files are.. but how do we tell the computer? File paths are the addresses the computer uses to track where different files are stored on the hard-disk. So all that we have to do is tell the computer the path of every file. For example this is where the files are stored on my computer:

```
In [ ]: path1 = '/home/aneesh/Images/Source/Type1/Type1_1.tif'
        path2= '/home/aneesh/Images/Source/Type1/Type1_2.tif'
        path3= '/home/aneesh/Images/Source/Type1/Type1_3tif'
```

Um... but doesn't that sound like a LOT of work. It is and luckily for us we don't have to this! all that we have to do is to tell the computer where the folder of our images is. This can be stored in a string variable, which I like to call `root_root` but you can call it whatever you want (as long as you can make sense of it later). Replace the string below with the path where your Images are stored:

```
In [30]: root_root = '/home/aneesh/Images/Source/'
```

To make Python automatically read our image names we have to import a *package* called `os`. A package is a set of mini programs that do small jobs. Each of these mini programs is called a function. Just like the `print()` function which we used we will use a few other functions to help us get the list of files.

Until we "import" a package Python doesn't know that a set of functions exists or that they should be used. To import a package we use the format

`import packageXYZ`

For our first example we will import the os package and use the the `listdir()` function to get a list of the items in the `root_root` directory.

We can also import only `listdir()` by typing: `from os import listdir()` but we will need a few other functions that are in os so we will import all of os so that we have access to the other functions using the `os.abc` notation.

```
In [29]: import os
         dir_of_root = os.listdir(root_root)
         dir_of_root

Out[29]: ['Type1', 'Type2']
```

Great! now we know that `root_root` has two directories where our images are stored, but thats not the full path to the images is it? Lets see how we can get that for images in the `Type1` directory.

**Note** If you got an error, make sure that your 'root_root' variable has a forward slash at the end. This signifies it as a directory.

```
In [33]: img_file_list = os.listdir(root_root+dir_of_root[0])
         img_file_list
```

```
Out[33]: ['Type1_11.tif',
          'Type1_13.tif',
          'Type1_1.tif',
          'Type1_19.tif',
          'Type1_16.tif',
          'Type1_9.tif',
          'Type1_6.tif',
          'Type1_14.tif',
          'Type1_7.tif',
          'Type1_3.tif',
          'Type1_10.tif',
          'Type1_4.tif',
          'Type1_12.tif',
          'Type1_8.tif',
          'Type1_5.tif',
          'Type1_18.tif',
          'Type1_17.tif',
          'Type1_20.tif',
          'Type1_15.tif',
          'Type1_2.tif']
```

Let's break down what happened there. To do that we must go to the innermost part of the statement, the part where we are using the plus sign. We are adding the string at the first position in `dir_of_root` to the string in `root_root`. If you don't understand how that works, try doing it:

```
In [31]: root_root+dir_of_root[0]

Out[31]: '/home/aneesh/Images/Source/Type1'
```

We then passed that joined string to the `os.listdir()` function which gave us a list of the images in the Type1 directory. The directory names can also be joint using the `os.path.join()` function. This can be useful when you are dealing with many variables contributing different sources like this:

```
In [34]: full_path = os.path.join(root_root,dir_of_root[0], img_file_list[0])

Out[34]: '/home/aneesh/Images/Source/Type1/Type1_11.tif'
```

Ok, now we have the path to one image file.. what about the others? Our lazy butts can't seriously be expected to change the 0 to a 1 then to a 2 etc just to access every file... Again we don't have to! We can use LOOPS!

**Sidenote:** you might have noticed above that the filenames don't seem to be in the right order. This seems to be a property of the way python assigns priority to file reading sequence. If you'd like to know more or if you need the files to be in order you should google how to do that.

## 9 Loop! Cause I'm lazy and smart :D

Loops are what you apply once you have figured out how to do something once, but need to do it again and again by making some minor changes. In our case we need to change the file name

every time. This idea of again-and-again is packaged into the word *iterate*. So when someone says "iterate over X" all that it means is that you apply the same things that are in X. We will be iterating over the file names stored in `img_file_list`.

In case you come from other languages, you can't normally iterate over objects in a container (like a list) however in python you can. Let's get our file list again:

```
In [10]: import os
         root_root = '/home/aneesh/Images/Source/'
         dir_of_root = os.listdir(root_root)
         img_file_list = os.listdir(root_root+dir_of_root[0])
```

The variable `img_file_list` contains the names of all the files in the Type1 directory. For loops are used to iterate over items. The term *for* comes from the understanding: "for x items in y do z". This is also how we write for loops in python:

```
In [11]: for file_name in img_file_list:
             print(file_name)

Type1_15.tif
Type1_4.tif
Type1_1.tif
Type1_9.tif
Type1_18.tif
Type1_14.tif
Type1_5.tif
Type1_11.tif
Type1_16.tif
Type1_2.tif
Type1_10.tif
Type1_12.tif
Type1_7.tif
Type1_17.tif
Type1_8.tif
Type1_13.tif
Type1_6.tif
Type1_19.tif
Type1_20.tif
Type1_3.tif
```

As you can see above what we are doing is printing every filename in the `img_file_list` variable. There is another function called `enumerate()` which counts the number of items in your object and returns both the count of the objects and the objects themselves. We use it below to print both the number of the file and the file name itself.

```
In [8]: for file_num, file_name in enumerate(img_file_list):
            print(file_num)
            print(file_name)
```

```
0
Type1_15.tif
1
Type1_4.tif
2
Type1_1.tif
3
Type1_9.tif
4
Type1_18.tif
5
Type1_14.tif
6
Type1_5.tif
7
Type1_11.tif
8
Type1_16.tif
9
Type1_2.tif
10
Type1_10.tif
11
Type1_12.tif
12
Type1_7.tif
13
Type1_17.tif
14
Type1_8.tif
15
Type1_13.tif
16
Type1_6.tif
17
Type1_19.tif
18
Type1_20.tif
19
Type1_3.tif
```

Now we have all the filenames being accessed, but how do we make that into a full file path? Remember `the os.path.join()` function? That's going to help a bit

```
In [12]: for file_name in img_file_list:
             full_path = os.path.join(root_root,dir_of_root[0], file_name)
             print(full_path)
```

```
/home/aneesh/Images/Source/Type1/Type1_15.tif
/home/aneesh/Images/Source/Type1/Type1_4.tif
/home/aneesh/Images/Source/Type1/Type1_1.tif
/home/aneesh/Images/Source/Type1/Type1_9.tif
/home/aneesh/Images/Source/Type1/Type1_18.tif
/home/aneesh/Images/Source/Type1/Type1_14.tif
/home/aneesh/Images/Source/Type1/Type1_5.tif
/home/aneesh/Images/Source/Type1/Type1_11.tif
/home/aneesh/Images/Source/Type1/Type1_16.tif
/home/aneesh/Images/Source/Type1/Type1_2.tif
/home/aneesh/Images/Source/Type1/Type1_10.tif
/home/aneesh/Images/Source/Type1/Type1_12.tif
/home/aneesh/Images/Source/Type1/Type1_7.tif
/home/aneesh/Images/Source/Type1/Type1_17.tif
/home/aneesh/Images/Source/Type1/Type1_8.tif
/home/aneesh/Images/Source/Type1/Type1_13.tif
/home/aneesh/Images/Source/Type1/Type1_6.tif
/home/aneesh/Images/Source/Type1/Type1_19.tif
/home/aneesh/Images/Source/Type1/Type1_20.tif
/home/aneesh/Images/Source/Type1/Type1_3.tif
```

Voila! We have all the filepaths without having to type them out... BUT we have all the file paths in the Type1 directory... What about the Type2 directory? Remember how the `dir_of_root` variable contains two values. Do you think we can loop over those too?

Why would I bring it up if we couldn't we can this is called loop nesting.

```
In [15]: for dor in dir_of_root:
             img_file_list = os.listdir(root_root+dor)
             for file_name in img_file_list:
                 full_path = os.path.join(root_root,dor, file_name)
                 print(full_path)
```

```
/home/aneesh/Images/Source/Type1/Type1_15.tif
/home/aneesh/Images/Source/Type1/Type1_4.tif
/home/aneesh/Images/Source/Type1/Type1_1.tif
/home/aneesh/Images/Source/Type1/Type1_9.tif
/home/aneesh/Images/Source/Type1/Type1_18.tif
/home/aneesh/Images/Source/Type1/Type1_14.tif
/home/aneesh/Images/Source/Type1/Type1_5.tif
/home/aneesh/Images/Source/Type1/Type1_11.tif
/home/aneesh/Images/Source/Type1/Type1_16.tif
/home/aneesh/Images/Source/Type1/Type1_2.tif
/home/aneesh/Images/Source/Type1/Type1_10.tif
/home/aneesh/Images/Source/Type1/Type1_12.tif
/home/aneesh/Images/Source/Type1/Type1_7.tif
/home/aneesh/Images/Source/Type1/Type1_17.tif
/home/aneesh/Images/Source/Type1/Type1_8.tif
```

```
/home/aneesh/Images/Source/Type1/Type1_13.tif
/home/aneesh/Images/Source/Type1/Type1_6.tif
/home/aneesh/Images/Source/Type1/Type1_19.tif
/home/aneesh/Images/Source/Type1/Type1_20.tif
/home/aneesh/Images/Source/Type1/Type1_3.tif
/home/aneesh/Images/Source/Type2/Type2_5.tif
/home/aneesh/Images/Source/Type2/Type2_32.tif
/home/aneesh/Images/Source/Type2/Type2_3.tif
/home/aneesh/Images/Source/Type2/Type2_6.tif
/home/aneesh/Images/Source/Type2/Type2_13.tif
/home/aneesh/Images/Source/Type2/Type2_4.tif
/home/aneesh/Images/Source/Type2/Type2_9.tif
/home/aneesh/Images/Source/Type2/Type2_22.tif
/home/aneesh/Images/Source/Type2/Type2_15.tif
/home/aneesh/Images/Source/Type2/Type2_16.tif
/home/aneesh/Images/Source/Type2/Type2_12.tif
/home/aneesh/Images/Source/Type2/Type2_18.tif
/home/aneesh/Images/Source/Type2/Type2_11.tif
/home/aneesh/Images/Source/Type2/Type2_14.tif
/home/aneesh/Images/Source/Type2/Type2_26.tif
/home/aneesh/Images/Source/Type2/Type2_19.tif
/home/aneesh/Images/Source/Type2/Type2_24.tif
/home/aneesh/Images/Source/Type2/Type2_25.tif
/home/aneesh/Images/Source/Type2/Type2_8.tif
/home/aneesh/Images/Source/Type2/Type2_30.tif
/home/aneesh/Images/Source/Type2/Type2_31.tif
/home/aneesh/Images/Source/Type2/Type2_7.tif
/home/aneesh/Images/Source/Type2/Type2_21.tif
/home/aneesh/Images/Source/Type2/Type2_28.tif
/home/aneesh/Images/Source/Type2/Type2_20.tif
/home/aneesh/Images/Source/Type2/Type2_17.tif
/home/aneesh/Images/Source/Type2/Type2_2.tif
/home/aneesh/Images/Source/Type2/Type2_1.tif
/home/aneesh/Images/Source/Type2/Type2_23.tif
/home/aneesh/Images/Source/Type2/Type2_29.tif
/home/aneesh/Images/Source/Type2/Type2_10.tif
/home/aneesh/Images/Source/Type2/Type2_27.tif
```

You will notice above that we added an extra line where we have to re-assign the value of `img_file_list` on every iteration over `dir_of_root` because we need to get the contents of different directories.

While the above method works well for our purposes, it is a little lengthy to type, there is *again* an easier solution. This comes in the form of the funtion `glob()`. Glob goes through your directory and directly finds all the files matching a particular pattern. In our case all our files end in the '.tif' extension. So, we can use `*.tif` as the pattern. Here the `*` is what is called as a wild-card. Python interprets it as "anything" and '*.tif' is interpreted as "anything that ends in .tif"

Lets try out glob on the Type1 folder below, first we need to import the glob package.

```
In [18]: import glob
         glob.glob( os.path.join(root_root,dir_of_root[0], '*.tif'))
```

```
Out[18]: ['/home/aneesh/Images/Source/Type1/Type1_15.tif',
          '/home/aneesh/Images/Source/Type1/Type1_4.tif',
          '/home/aneesh/Images/Source/Type1/Type1_1.tif',
          '/home/aneesh/Images/Source/Type1/Type1_9.tif',
          '/home/aneesh/Images/Source/Type1/Type1_18.tif',
          '/home/aneesh/Images/Source/Type1/Type1_14.tif',
          '/home/aneesh/Images/Source/Type1/Type1_5.tif',
          '/home/aneesh/Images/Source/Type1/Type1_11.tif',
          '/home/aneesh/Images/Source/Type1/Type1_16.tif',
          '/home/aneesh/Images/Source/Type1/Type1_2.tif',
          '/home/aneesh/Images/Source/Type1/Type1_10.tif',
          '/home/aneesh/Images/Source/Type1/Type1_12.tif',
          '/home/aneesh/Images/Source/Type1/Type1_7.tif',
          '/home/aneesh/Images/Source/Type1/Type1_17.tif',
          '/home/aneesh/Images/Source/Type1/Type1_8.tif',
          '/home/aneesh/Images/Source/Type1/Type1_13.tif',
          '/home/aneesh/Images/Source/Type1/Type1_6.tif',
          '/home/aneesh/Images/Source/Type1/Type1_19.tif',
          '/home/aneesh/Images/Source/Type1/Type1_20.tif',
          '/home/aneesh/Images/Source/Type1/Type1_3.tif']
```

So now we can replace the inner loop of with glob:

```
In [19]: for dor in dir_of_root:
             img_file_list = os.listdir(root_root+dor)
             full_paths = glob.glob(os.path.join(root_root,dor, '*.tif'))
             print(full_paths)
```

```
['/home/aneesh/Images/Source/Type1/Type1_15.tif', '/home/aneesh/Images/Source/Type1/Type1_4.tif'
['/home/aneesh/Images/Source/Type2/Type2_5.tif', '/home/aneesh/Images/Source/Type2/Type2_32.tif'
```

The print above looks a little different, this is because two lists which are the output of the function glob are being printed. However, when it comes to loops there is another little trick that can make our code even shorter. This is called a **list comprehension**. A list comprehension is basically a for loop compressed into one line. It has some limitations compared to a for loop but its the most convinient way to iterate over loops. Can you figure out how we use the list comprehension to generate out list of lists of full paths to the images in the Type1 and Type2 directories?

```
In [20]: file_paths = [glob.glob(os.path.join(root_root,dor, '*.tif')) for dor in dir_of_root]
         file_paths
```

```
Out[20]: [['/home/aneesh/Images/Source/Type1/Type1_15.tif',
           '/home/aneesh/Images/Source/Type1/Type1_4.tif',
           '/home/aneesh/Images/Source/Type1/Type1_1.tif',
```

```
 '/home/aneesh/Images/Source/Type1/Type1_9.tif',
 '/home/aneesh/Images/Source/Type1/Type1_18.tif',
 '/home/aneesh/Images/Source/Type1/Type1_14.tif',
 '/home/aneesh/Images/Source/Type1/Type1_5.tif',
 '/home/aneesh/Images/Source/Type1/Type1_11.tif',
 '/home/aneesh/Images/Source/Type1/Type1_16.tif',
 '/home/aneesh/Images/Source/Type1/Type1_2.tif',
 '/home/aneesh/Images/Source/Type1/Type1_10.tif',
 '/home/aneesh/Images/Source/Type1/Type1_12.tif',
 '/home/aneesh/Images/Source/Type1/Type1_7.tif',
 '/home/aneesh/Images/Source/Type1/Type1_17.tif',
 '/home/aneesh/Images/Source/Type1/Type1_8.tif',
 '/home/aneesh/Images/Source/Type1/Type1_13.tif',
 '/home/aneesh/Images/Source/Type1/Type1_6.tif',
 '/home/aneesh/Images/Source/Type1/Type1_19.tif',
 '/home/aneesh/Images/Source/Type1/Type1_20.tif',
 '/home/aneesh/Images/Source/Type1/Type1_3.tif'],
['/home/aneesh/Images/Source/Type2/Type2_5.tif',
 '/home/aneesh/Images/Source/Type2/Type2_32.tif',
 '/home/aneesh/Images/Source/Type2/Type2_3.tif',
 '/home/aneesh/Images/Source/Type2/Type2_6.tif',
 '/home/aneesh/Images/Source/Type2/Type2_13.tif',
 '/home/aneesh/Images/Source/Type2/Type2_4.tif',
 '/home/aneesh/Images/Source/Type2/Type2_9.tif',
 '/home/aneesh/Images/Source/Type2/Type2_22.tif',
 '/home/aneesh/Images/Source/Type2/Type2_15.tif',
 '/home/aneesh/Images/Source/Type2/Type2_16.tif',
 '/home/aneesh/Images/Source/Type2/Type2_12.tif',
 '/home/aneesh/Images/Source/Type2/Type2_18.tif',
 '/home/aneesh/Images/Source/Type2/Type2_11.tif',
 '/home/aneesh/Images/Source/Type2/Type2_14.tif',
 '/home/aneesh/Images/Source/Type2/Type2_26.tif',
 '/home/aneesh/Images/Source/Type2/Type2_19.tif',
 '/home/aneesh/Images/Source/Type2/Type2_24.tif',
 '/home/aneesh/Images/Source/Type2/Type2_25.tif',
 '/home/aneesh/Images/Source/Type2/Type2_8.tif',
 '/home/aneesh/Images/Source/Type2/Type2_30.tif',
 '/home/aneesh/Images/Source/Type2/Type2_31.tif',
 '/home/aneesh/Images/Source/Type2/Type2_7.tif',
 '/home/aneesh/Images/Source/Type2/Type2_21.tif',
 '/home/aneesh/Images/Source/Type2/Type2_28.tif',
 '/home/aneesh/Images/Source/Type2/Type2_20.tif',
 '/home/aneesh/Images/Source/Type2/Type2_17.tif',
 '/home/aneesh/Images/Source/Type2/Type2_2.tif',
 '/home/aneesh/Images/Source/Type2/Type2_1.tif',
 '/home/aneesh/Images/Source/Type2/Type2_23.tif',
 '/home/aneesh/Images/Source/Type2/Type2_29.tif',
 '/home/aneesh/Images/Source/Type2/Type2_10.tif',
```

```
'/home/aneesh/Images/Source/Type2/Type2_27.tif']]
```

We will stop here for this chapter. In the next chapter we will read and display our first image.