Python Introduction

UKZN SKA School 2013

1 What is python?

Python is a cross-platform (i.e., available on GNU/Linux, Mac, Windows etc.) scripting language. It can be used to write large(ish) applications or programs, or simply used as a 'glue' language to run other programs in sequence, automating tasks that would otherwise be tedious (e.g., to make processing pipelines). It is free software.

Python has extensive (and very good) documentation available on the web. See, e.g., `http://docs.python.org/2/tutorial/index.html` for the official tutorial, but of course there are many others.

2 Running a python program

Python is an interpreted language. To run a python program (e.g., say we have a file called `helloWorld.py`), at the terminal:

```
% python helloWorld.py
```

Running the `python` command without specifying the name of a python program will open the python interpreter in interactive mode, which is useful for testing things out or performing simple tasks. However, `ipython` (the 'i' stands for 'interactive') is a much friendlier interpreter if you simply wish to use python interactively rather than for running programs.

From the interpreter, it is easy to access built-in documentation for all python functions, modules etc. using the help command, e.g.,

```
% ipython
% import math
% help(math.cos)
```

This will print out a brief description of the function and show what parameters it needs, and returns.

3 Anatomy of a python program

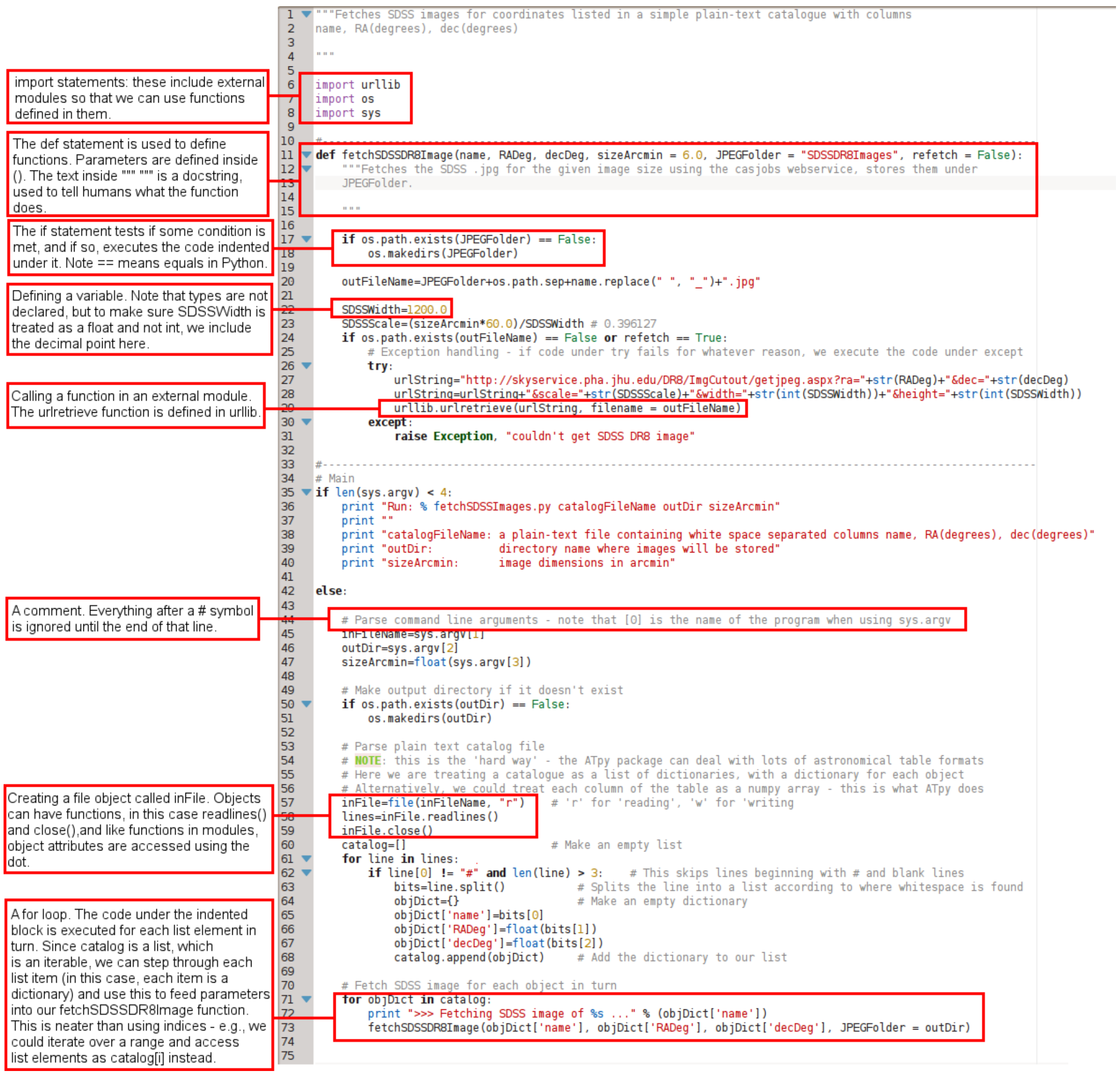
Fig. 1 overleaf shows a relatively simple python program that illustrates many of the features of the language. The code can be downloaded from `http://www.acru.ukzn.ac.za/~ska2013/exampleCode/fetchSDSSImages.py`. This program uses python's built-in `urllib` module to download colour images of astronomical objects from the Sloan Digital Sky Survey (`http://www.sdss.org/`).

Note that python reads programs from top to bottom, so all function definitions and module import statements appear at the top (python needs to know what something means before it can use it).

Indentation has meaning in python - it is used to group statements, e.g., under conditional **if** statements, inside **for** loops, or inside function definitions (using the **def** statement).

4 Variables and built-in types

Python is a dynamically typed language, which means that variable types do not have to be explicitly declared before they can be used (unlike C or Fortran). However, it is also strongly



```python
"""Fetches SDSS images for coordinates listed in a simple plain-text catalogue with columns
name, RA(degrees), dec(degrees)

"""

import urllib
import os
import sys

#------------------------------------------------------------------------------------------------------------
def fetchSDSSDR8Image(name, RADeg, decDeg, sizeArcmin = 6.0, JPEGFolder = "SDSSDR8Images", refetch = False):
    """Fetches the SDSS .jpg for the given image size using the casjobs webservice, stores them under
    JPEGFolder.

    """

    if os.path.exists(JPEGFolder) == False:
        os.makedirs(JPEGFolder)

    outFileName=JPEGFolder+os.path.sep+name.replace(" ", "_")+".jpg"

    SDSSWidth=1200.0
    SDSSScale=(sizeArcmin*60.0)/SDSSWidth # 0.396127
    if os.path.exists(outFileName) == False or refetch == True:
        # Exception handling - if code under try fails for whatever reason, we execute the code under except
        try:
            urlString="http://skyservice.pha.jhu.edu/DR8/ImgCutout/getjpeg.aspx?ra="+str(RADeg)+"&dec="+str(decDeg)
            urlString=urlString+"&scale="+str(SDSSScale)+"&width="+str(int(SDSSWidth))+"&height="+str(int(SDSSWidth))
            urllib.urlretrieve(urlString, filename = outFileName)
        except:
            raise Exception, "couldn't get SDSS DR8 image"

#------------------------------------------------------------------------------------------------------------
# Main
if len(sys.argv) < 4:
    print "Run: % fetchSDSSImages.py catalogFileName outDir sizeArcmin"
    print ""
    print "catalogFileName: a plain-text file containing white space separated columns name, RA(degrees), dec(degrees)"
    print "outDir:          directory name where images will be stored"
    print "sizeArcmin:      image dimensions in arcmin"

else:

    # Parse command line arguments - note that [0] is the name of the program when using sys.argv
    inFileName=sys.argv[1]
    outDir=sys.argv[2]
    sizeArcmin=float(sys.argv[3])

    # Make output directory if it doesn't exist
    if os.path.exists(outDir) == False:
        os.makedirs(outDir)

    # Parse plain text catalog file
    # NOTE: this is the 'hard way' - the ATpy package can deal with lots of astronomical table formats
    # Here we are treating a catalogue as a list of dictionaries, with a dictionary for each object
    # Alternatively, we could treat each column of the table as a numpy array - this is what ATpy does
    inFile=file(inFileName, "r")    # 'r' for 'reading', 'w' for 'writing
    lines=inFile.readlines()
    inFile.close()
    catalog=[]                      # Make an empty list
    for line in lines:
        if line[0] != "#" and len(line) > 3:    # This skips lines beginning with # and blank lines
            bits=line.split()           # Splits the line into a list according to where whitespace is found
            objDict={}                  # Make an empty dictionary
            objDict['name']=bits[0]
            objDict['RADeg']=float(bits[1])
            objDict['decDeg']=float(bits[2])
            catalog.append(objDict)     # Add the dictionary to our list

    # Fetch SDSS image for each object in turn
    for objDict in catalog:
        print ">>> Fetching SDSS image of %s ..." % (objDict['name'])
        fetchSDSSDR8Image(objDict['name'], objDict['RADeg'], objDict['decDeg'], JPEGFolder = outDir)
```

import statements: these include external modules so that we can use functions defined in them.

The def statement is used to define functions. Parameters are defined inside (). The text inside """ """ is a docstring, used to tell humans what the function does.

The if statement tests if some condition is met, and if so, executes the code indented under it. Note == means equals in Python.

Defining a variable. Note that types are not declared, but to make sure SDSSWidth is treated as a float and not int, we include the decimal point here.

Calling a function in an external module. The urlretrieve function is defined in urllib.

A comment. Everything after a # symbol is ignored until the end of that line.

Creating a file object called inFile. Objects can have functions, in this case readlines() and close(),and like functions in modules, object attributes are accessed using the dot.

A for loop. The code under the indented block is executed for each list element in turn. Since catalog is a list, which is an iterable, we can step through each list item (in this case, each item is a dictionary) and use this to feed parameters into our fetchSDSSDR8Image function. This is neater than using indices - e.g., we could iterate over a range and access list elements as catalog[i] instead.

Figure 1: An example python program, with elements labelled and briefly explained.

typed, so this means that the following will give an error:

```
% a=6
% print 'Cheese' + a
TypeError:  cannot concatenate 'str' and 'int' objects
```

Here is a brief description of the main built-in types, look at the example programs for how to use them (or read the documentation):

- **int** - an integer, e.g., 2
- **float** - a float, e.g, 3.141592654
- **str** - a string, e.g., 'cheese'

You can convert between types by using `int()`, `float()`, `str()` etc., e.g.,

```
% a=6
% float(a)
6.0
```

See also the example `http://www.acru.ukzn.ac.za/~ska2013/exampleCode/fromJon/python_variables.py`.

Also built-in are **lists** and **dictionaries**. These are used in the example code shown in Fig. 1.

Lists can contain any python object, and they can be mixed, i.e., they can contain floats, ints, strings etc. - i.e., `a=['Cheese', 3.141592654, 7]` is a valid list. List items can be accessed using square brackets `[]`, e.g., for the above list,

```
% a[1]
3.141592654
```

since the first item in the list is at index 0.

Lists can be *sliced* using `:`. For example, to return only the last two items of the above list:

```
% a[1:]
[3.141592654, 7]
```

To access items in reverse order we can use negative indices, e.g.,

```
% a[-2]
3.141592654
```

To reverse the order of a list, use

```
% a[::-1]
[7, 3.141592654, 'cheese']
```

Dictionaries are unordered lists where the elements are accessed using *keys*. They are defined using curly brackets {}. In the below example, 'bread' and 'filling' are the keys:

```
% sandwich={'bread':  'white', 'filling':  'cheese'}
% sandwich
{'bread':  'white', 'filling':  'cheese'}
```

To iterate over all keys in a dictionary, use e.g.,

```
% for key in sandwich.keys():
%   print sandwich[key]
cheese
white
```

5 Classes and objects

Python is an object oriented language, like C++ or Java. Unlike C++, classes have no private variables or functions. For an example of classes and objects in action, see `http://www.acru.ukzn.ac.za/~ska2013/exampleCode/classExample.py`.

6 The standard library

Python has an extensive standard library with very good documentation, available at `http://docs.python.org/2/library/`.

7 Installing additional modules from source

It is relatively straightforward to install additional python modules from source, even without root access. Almost all modules or packages come with an install script called `setup.py`. This will build the code using the compilers available on your system if necessary, or just copy it to the appropriate directories in the case of a module written purely in python.

If you don't have root access, the way to make modules visible to your python installation is something like the following (exact paths may vary depending on python version):

```
% python setup.py install --prefix=$HOME/local
```

and add something like the following to your `.bashrc` file (or equivalent):

```
export PYTHONPATH=$PYTHONPATH:$HOME/local/lib/python2.7/site-packages
```

8 Numpy, scipy and matplotlib

Numpy provides a fast implementation of numerical arrays in python. To get the best out of it, perform operations using built-in numpy routines and *avoid using for loops*, which are slow for very large arrays. Numpy arrays can be sliced and indexed like lists.

Scipy is a package which provides many modules useful for scientific computing - e.g., it includes modules for interpolation (`scipy.interpolate`), statistics (`scipy.stats`), and some image processing (`scipy.ndimage`).

Matplotlib is a package which can be used to make a huge variety of publication quality plots.

Documentation for all of these is available and easily accessible on the web (as well as through python's built-in help function).

For a simple example of using all of these, see `http://www.acru.ukzn.ac.za/~ska2013/exampleCode/numpyScipyMatplotlibExample.py`.

9 Final remarks

The only way to master coding is through practice. Try to make something, read documentation, and look at other people's code. IPython is particularly good for playing with things to see how they work. Perhaps try this with the numpy/scipy/matplotlib example. Or take the classes/objects example and try to make a zoo.