Functional-style scripting in Python (2.7)

Philip Dießner

Computing session – 18.03.15

What a Phenomenologist does

Physics

- SUSY model (MRSSM) $\longrightarrow$ Lagrangian
- Parameter space: ≈ 10 important
- predictions for masses and couplings
- predictions for the TeV scale, many constrains from previous experiment
- $\longrightarrow$ need many checks
- not all points give a physical spectrum
- need generate lots of points
- compare to other models (MSSM)

Technicalities

- Mass spectrum and coupling output in a standard format
- Tools to study different observables exist
- everything takes text file input and gives text file output as standard

Different codes exist

All in different languages

SPheno	Fortran
FlexibleSUSY	C++
HiggsBounds/HiggsSignals	Fortran
MicroMegas	C
Madgraph	Python
Herwig++	C++
Delphes/CheckMATE	C++/Python wrapper

Python standard libary

- subprocess
- multiprocessing
- sqlite3

subprocess

Calling external routines

```
import subprocess as spr
spr.call(['ls','-l'])
...
cwd = '/path/to/input/files/'
out = spr.check_output(['HiggsBounds', 'LandH', 'effC',
                        str(neutralhiggs), str(chargedhiggs),
                        name], stderr=spr.STDOUT, cwd=cwd)
```

multiprocessing

As the name says

```
import multiprocessing as mp

def square(x):
    return x*x

ncore = 4
pool = mp.Pool(ncore)
for i in range(10):
    print pool.apply_async(square,(i,))
print pool.map(square, range(10))
```

sqlite3–Database management

```
import sqlite3
conn = sqlite3.connect('example.db')
c = conn.cursor()
c.execute('''CREATE TABLE points
             (tanb real, lambda real, mu233 real)''')
c.execute('INSERT INTO points VALUES
           ({0},{1},{2})'.format(10.0,0.1,1000**2))

c.execute('''CREATE TABLE result
             (tanb real, mh real, pval real)''')
c.execute('INSERT INTO result VALUES
           ({0},{1},{2})'.format(10.0,125.09, 0.3))
conn.commit()
c.execute('SELECT points.lambda, result.mh FROM points
           JOIN results ON points.tanb=results.tanb')

for result in c:
    print result  # (0.1,125.09)

conn.close()
```

Functional programming style

Computation as the evaluation of mathematical functions, trying to avoid changing-state and mutable data. Eliminating side effects, i.e. changes in state that do not depend on the function inputs, same input leads to same output can make it much easier to understand and predict the behavior of a program

Buzz words

- Higher order functions
- Anonymous function / λ function
- Recursion
- List vs. generator (more python-specific)
- List comprehension (more python-specific)

Higher order functions

First order functions take data types as input/return them. Higher order functions take functions as argument and/or return function.
(compare to operators and functionals in math)

Usual suspects – `map`, `filter`, `reduce`

```
print map(square, range(10))
print filter(lambda x: x > 4, range(10))
print reduce(lambda x, y: x + y, range(10))
```

`lambda` is keyword for anonymous function, so no name only for one-time use

```
def run_generator(execute):
    """
    return function that executes toexecute with info on
    runner
    """
    def runner(path):
            out = spr.check_output(execute,
                                   stderr=spr.STDOUT,
                                   cwd=path)
            return out
    return runner

run_HB = run_generator(['HiggsBounds', 'LandH', 'effC',
                        str(neutralhiggs), str(chargedhiggs),
                        name])
# this is a function
run_HB('parameterpoints/point1/')
run_HB('parameterpoints/point2/')
map(run_HB,list_of_point_dirs)
```

Generators

Difference between `range` and `xrange`

`range` returns list, `xrange` returns generator object (comparable with iterator). This knows only acutal value and how to get to next element. Needs `yield` keyword.

```
# Fibonacci numbers, imperative style
# https://docs.python.org/2.7/tutorial/modules.html
def fibonacci(n, first = 0, second = 1):
    for i in range(n):
        yield first  # Return current iteration
        first, second = second, first + second

result = fibonacci(9)
print result #<generator object fibonacci at 0x7f6732287af0>
print [ x for x in result ] # [0, 1, 1, 2, 3, 5, 8, 13, 21]
```

For working with generators the package `itertools` is interesting

list comprehension

do stuff with iterable object and get a list (or generator) out:

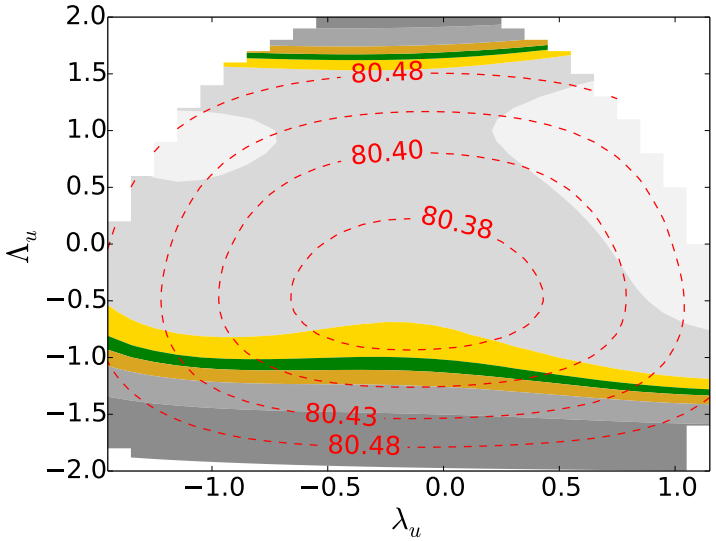
```
result = range(5) # [0, 1, 2, 3, 4]
print [square(x) for x in result] # [0, 1, 4, 9, 16]
print [x for x in result if x == 4] # [4]
print (square(x) for x in result) # round brackets
#  <generator object <genexpr> at 0x7f6732287af0>

alternative =  'result'
[x for x in alternative] # ['r', 'e', 's', 'u', 'l', 't']
```

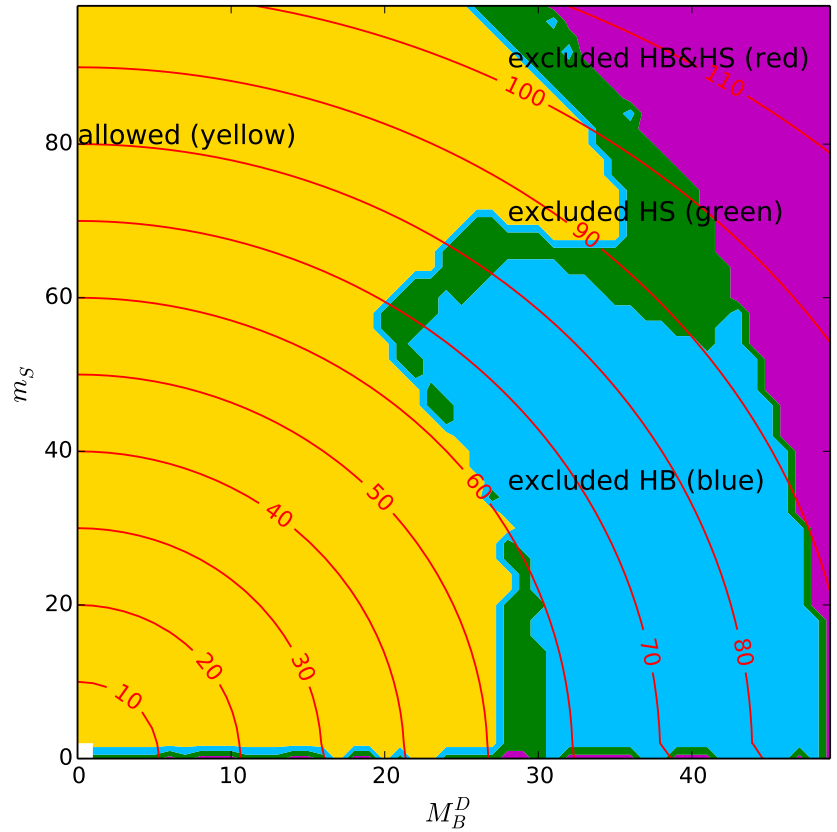
What is missing? – making plots

Using `mathplotlib`

Exclusion by `HiggsBounds` and `HiggsSignals`:

No tree-level reduction for SM-like Higgs





And then your supervisor tells you to change the color scheme...