Project Euler (Mostly) in Python

Bryan Harris
Research Engineer
University of Dayton
Research Institute
BSME, PE
Python, LabView, C tinkerer
GO Bucks!
The Euler Challenge

What?

391 mathematical programming challenges
- Integers
- Require Programming
- Should be solvable in under an hour
  - Some are
  - some aren't

Why?

- I Love logic puzzles
- Brush up on Python(3?)
- Experiment with multi-threading and various interpreters
Benchmark System

Ubuntu 12.04 x86_64
4 core i7 @2.3 GHz+
16 GB Ram
120 GB SSD

Python 2.7.3
Pypy 2.7.2
Python 3.2.3
Jython 2.5.1
on OpenJDK Java 1.6.24
bzr branch lp:eulerplay

http://bazaar.launchpad.net/~brywilharris/eulerplay/trunk/files

Currently 25 worked examples

https://launchpad.net/pype

>50 worked examples
If we list all the natural numbers below 10 that are multiples of 3 or 5, we get 3, 5, 6 and 9. The sum of these multiples is 23.

Find the sum of all the multiples of 3 or 5 below 1000.
#!/usr/bin/env python
#This program adds all the integer numbers divisible by 3 and 5
#below a selected maximum number. It is a straight forward,
brute-force approach.
if __name__ == '__main__':  #boilerplate
    import sys  #for command line input
    sum = 0;
    for i in range(0,int(sys.argv[1])):  #int arg required
        if (i%3==0) or (i%5==0):  #divisible by 3 or 5?
            sum = sum + i  #add to the sum
    print(sum)  #spit it out at the end

233168
Problem 1 Benchmarks

- n=100000000
- Pypy wins
- Jython – 4.2 GB of usage then FAIL
  - java.lang.OutOfMemoryError: Java heap space
Each new term in the Fibonacci sequence is generated by adding the previous two terms. By starting with 1 and 2, the first 10 terms will be:
1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

By considering the terms in the Fibonacci sequence whose values do not exceed four million, find the sum of the even-valued terms.
```python
#!/usr/bin/env python

def nextfib(a, b):
    # compute the next Fibonacci term
    c = a + b
    return (b, c) # keep the previous term

if __name__ == '__main__':
    import sys
    a = 1
    b = 2
    mysum = 0
    val = int(sys.argv[1]) # Bad code, assumes input is an int
    while(b < val):
        if b%2==0: mysum += b # add it up if it's even
        (a, b) = nextfib(a, b)
    print(mysum) # print the answer at the end

4613732
```
Problem 2 Benchmarks

- Jython doesn't *always* fail..
- Note the log scale
The prime factors of 13195 are 5, 7, 13 and 29.

What is the largest prime factor of the number 600851475143?
#!/usr/bin/python

def isprime(val):
    # Checks to see if a number is prime, brute force
    for x in range(2, int(val**0.5)+1):
        if val % x == 0:
            return False
    return True

def factor(val):
    # Get the prime factors of a number
    if isprime(val):
        return [val]
    i = 2
    thesefactors = []
    tempval = val
    while (i <= tempval):
        if tempval % i == 0:
            thesefactors += [i]
            tempval = tempval / i
            i = 2
        else:
            i = i + 1
    if len(thesefactors) <= 1:
        return [val]
    else:
        return thesefactors

if __name__ == '__main__':
    import sys
    factors = factor(int(sys.argv[1]))
    print(factors)

[71, 839, 1471, 6857]
Problem 3 Benchmarks

- Factoring Primes
  - Used in crypto systems
The sum of the squares of the first ten natural numbers is,
\[ 12 + 22 + \ldots + 102 = 385 \]
The square of the sum of the first ten natural numbers is,
\[ (1 + 2 + \ldots + 10)^2 = 55^2 = 3025 \]
Hence the difference between the sum of the squares of the first ten natural numbers and the square of the sum is 3025 - 385 = 2640.
Find the difference between the sum of the squares of the first one hundred natural numbers and the square of the sum.
def sumsquare(val): #Returns the sum of the squares.
    ss = 1
    for each in range(2,val+1): ss = ss + each**2
    return ss

def squaresum(val): #Returns the squares of the sums.
    ss = 1
    for each in range(2,val+1): ss = ss + each
    return ss**2

if __name__ == '__main__':
    import sys
    val = int(sys.argv[1]) # Bad Code, assumes input is an integer
    print (sumsquare(val))
    print (squaresum(val))
    print (squaresum(val)-sumsquare(val))

25164150
Problem 6 CPU Benchmarks

- Python took longest
- Lots of Memory Usage
Problem 6 Memory Usage

- Python 3 Uses the least memory
- Jython used just over 4 GB then crashed
Starting in the top left corner of a 2x2 grid, there are 6 routes (without backtracking) to the bottom right corner.

How many routes are there through a 20x20 grid?
Problem 6 Solution

#Computes the number of routes through an nxn square. Complexity increases
#by ~4n

def valid((x,y),(w,h)):
    validmoves = []
    #print p
    if x < w:  #right
        validmoves += [(x+1,y)]
    if y < h:  #down
        validmoves += [(x, y+1)]
    return validmoves

def walk(p,s):
    mycount = 0
    for each in valid(p,s):
        if each == s:
            #print each
            return 1
        else:
            #print each,
            mycount += walk(each,s)
    return mycount

import sys
p=(0,0)
size = int(sys.argv[1])  # Bad code, assumes first argument is an int
print (walk(p,(size,size))))

137846528820
Problem 15 Benchmarks

- GCC 20x faster than Python
- 4x faster than pypy
- Compile time is small
- tcc 1.5x slower than gcc
- time tcc -run routes.c 14
Multithreading in Python

- import Threading
  - Single Core
  - May speed up I/O bound code

- import Processing
  - Syntax same as above
  - multithread -> multiprocess
  - Higher overhead

- greenlets

- IPC
  - pipes, SHM, sockets, etc.
import threading as t

t.Thread(group=None, target=aFunction, name=None, args=(), kwargs={})

t.start(), t.run(), t.join()
  start() calls run(), which should be defined in your code
  join() wait soe a thread to complete

Basic multithreading primitives:
  t.Lock()
  t.Semaphore()
  t.Timer()
  t.is_alive()
Processing

- `import multiprocessing as p`
- `p.Process(group=None, target=aFunction, name=None, args=(), kwargs={})`
- `p.start(), p.run(), p.join()`
  - `start()` calls `run()`, which should be defined in your code
  - `join()` wait soe a thread to complete
- **Basic multithreading primitives:**
  - `p.Lock()`
  - `p.Semaphore()`
  - `p.Timer()`
  - `p.is_alive()`
Processing Benchmarks

Real Time vs. nxn Cube Size
8 processes used for multithreaded runs

- blue: routes-fast n
- orange: pypy routes.py n
- yellow: pypy routes-mt.py n
- green: ./routes.py n
- red: ./routes-mt.py n
Greenlets

- `from greenlet import greenlet`
  
  ```python
  def test1():
      print 12
      gr2.switch()
      print 34
  
  def test2():
      print 56
      gr1.switch()
      print 78
  
  gr1 = greenlet(test1)
  gr2 = greenlet(test2)
  gr1.switch()
  ```

- Looks like could be used same way as threading or multiprocessing
IPC

- A bit less pythonic
- Portable - maybe
- Sockets - Multi-machine
Questions?