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

Solution Code

bzr branch lp:eulerplay

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

Currently 25 worked examples

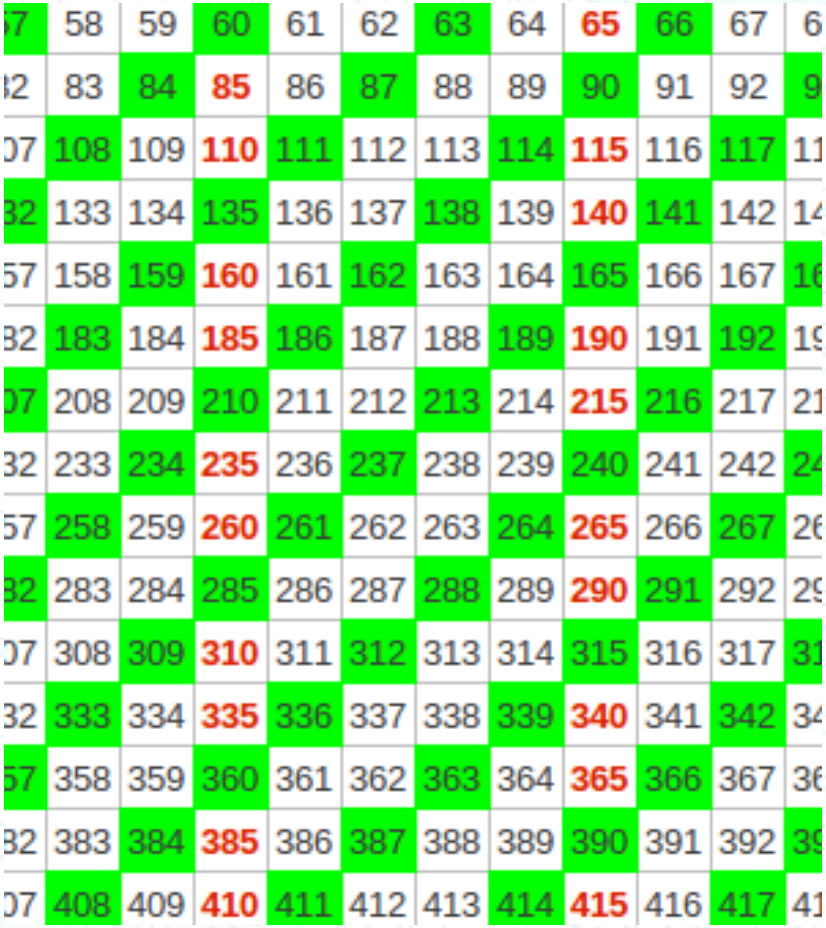
https://launchpad.net/pype

>50 worked examples

– Problem 1 –

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.

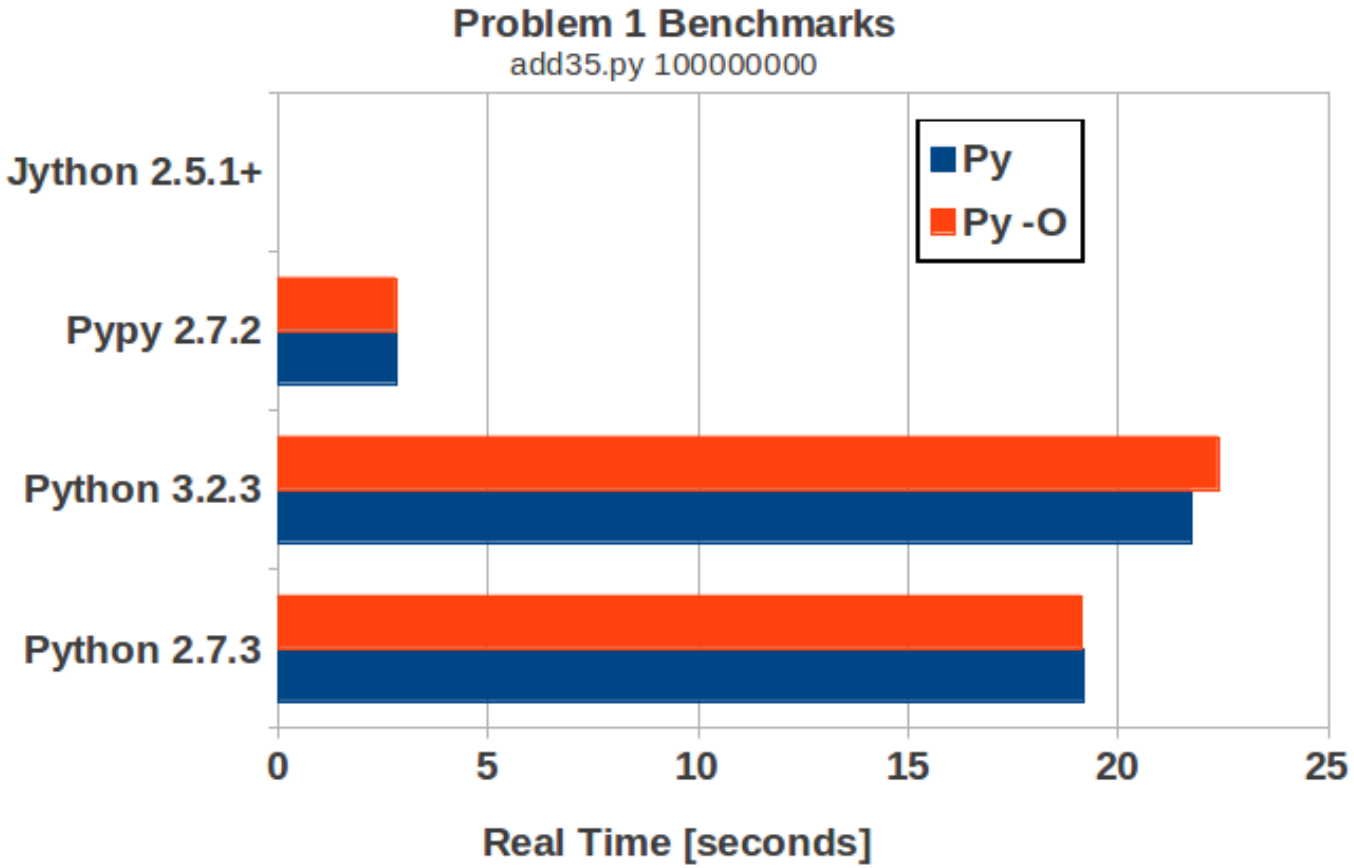


Problem 1 Solution

```
#!/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

– Problem 2 –

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, ...

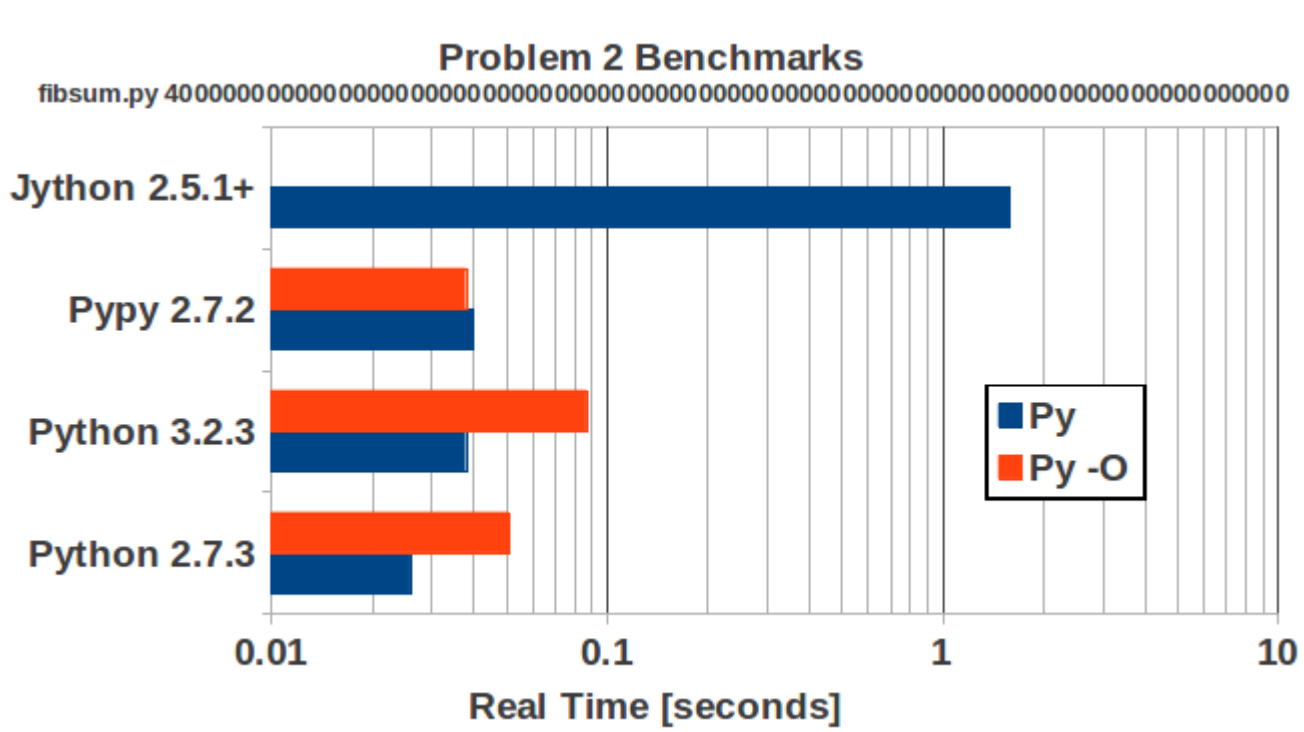
By considering the terms in the Fibonacci sequence whose values do not exceed four million, find the sum of the even-valued terms.

Problem 2 Solution

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

def nextfib(a,b): #compute the next Fibonachi 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

– Problem 3 –

The prime factors of 13195 are 5, 7, 13 and 29.

What is the largest prime factor of the number 600851475143 ?

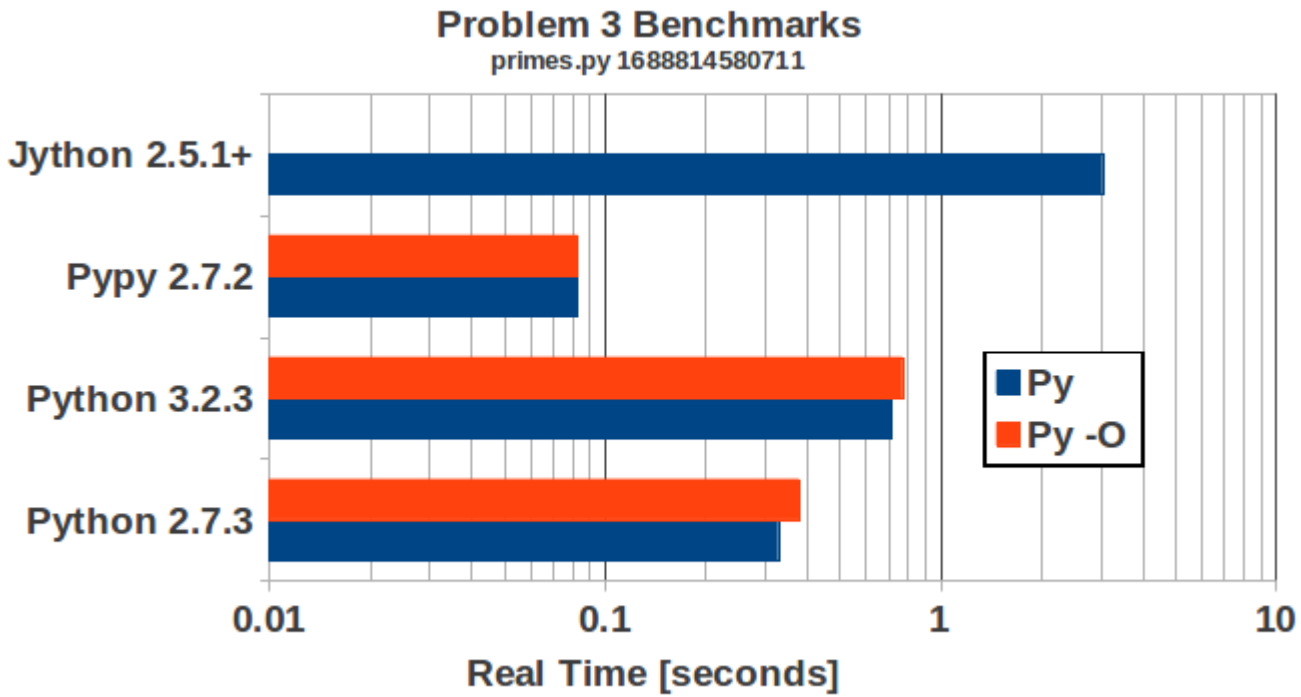
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46
47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92
93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115
116	117	118	119	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138
139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160	161
162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184
185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224	225	226	227	228	229	230
231	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253
254	255	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276
277	278	279	280	281	282	283	284	285	286	287	288	289	290	291	292	293	294	295	296	297	298	299
300	301	302	303	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319	320	321	322
323	324	325	326	327	328	329	330	331	332	333	334	335	336	337	338	339	340	341	342	343	344	345
346	347	348	349	350	351	352	353	354	355	356	357	358	359	360	361	362	363	364	365	366	367	368
369	370	371	372	373	374	375	376	377	378	379	380	381	382	383	384	385	386	387	388	389	390	391
392	393	394	395	396	397	398	399	400	401	402	403	404	405	406	407	408	409	410	411	412	413	414
415	416	417	418	419	420	421	422	423	424	425	426	427	428	429	430	431	432	433	434	435	436	437
438	439	440	441	442	443	444	445	446	447	448	449	450	451	452	453	454	455	456	457	458	459	460
461	462	463	464	465	466	467	468	469	470	471	472	473	474	475	476	477	478	479	480	481	482	483
484	485	486	487	488	489	490	491	492	493	494	495	496	497	498	499	500	501	502	503	504	505	506
507	508	509	510	511	512	513	514	515	516	517	518	519	520	521	522	523	524	525	526	527	528	529

Problem 3 Solution

```
#!/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

– Problem 6 –

The sum of the squares of the first ten natural numbers is,

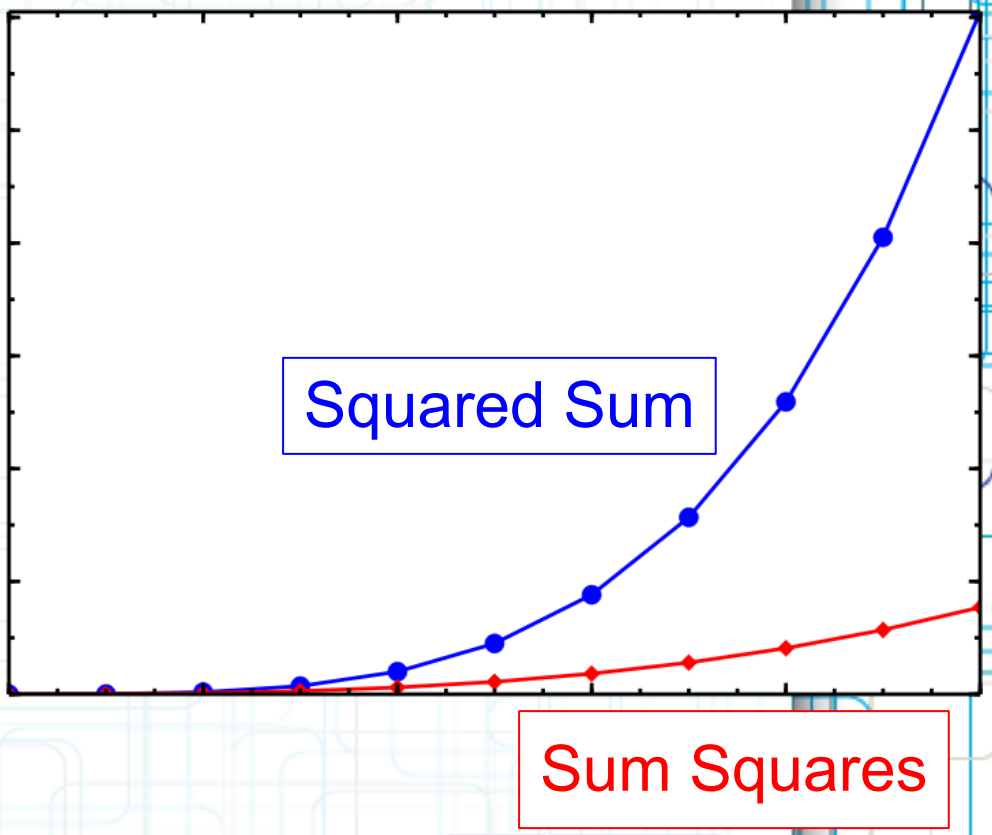
$$1^2 + 2^2 + ... + 10^2 = 385$$

The square of the sum of the first ten natural numbers is,

$$(1 + 2 + ... + 10)^2 = 55^2 = 3025$$

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.

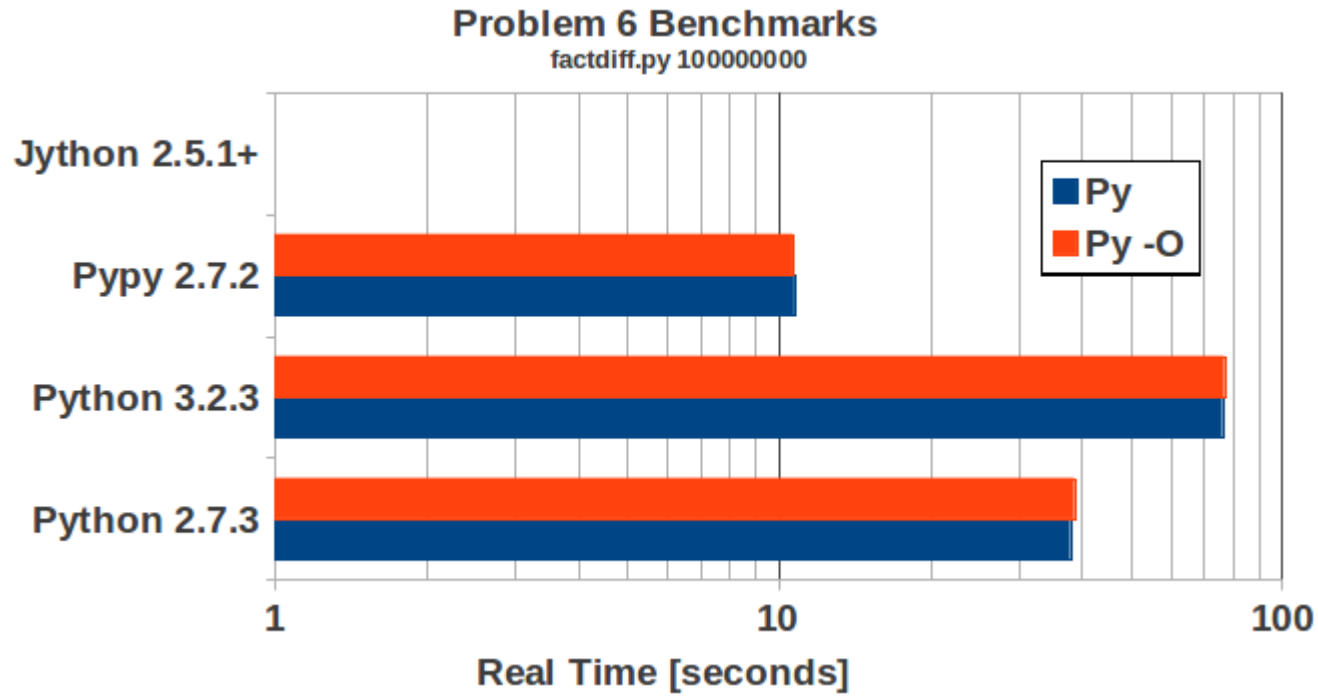


Problem 6 Solution

```
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))
```

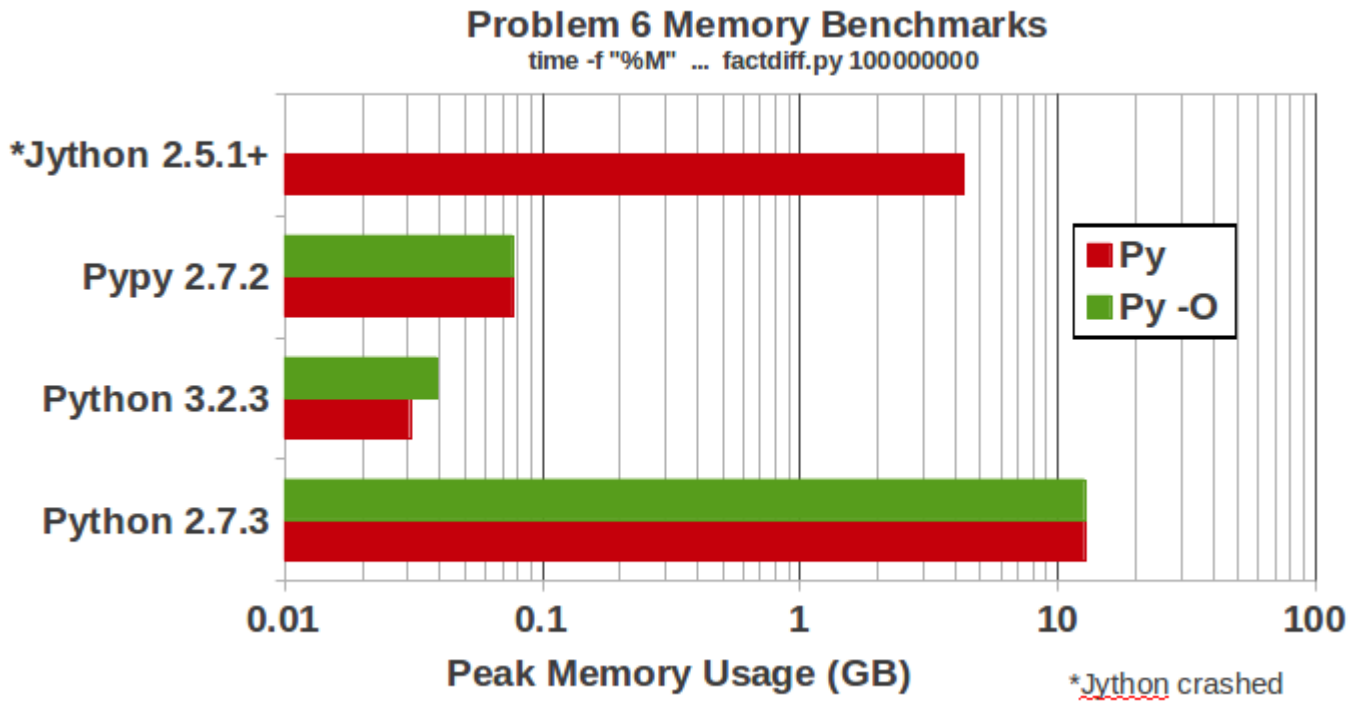
25164150

Problem 6 CPU Benchmarks



- Python took longest
- Lots of Memory Usage

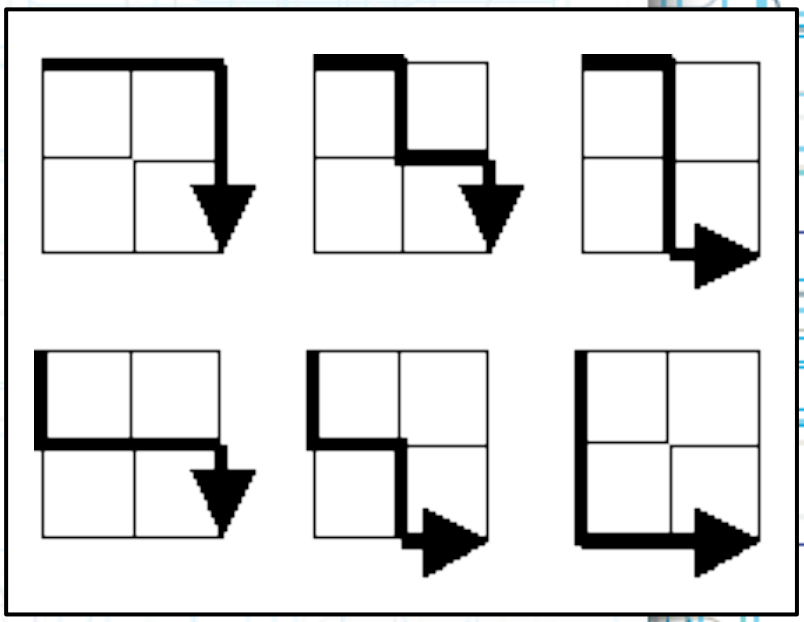
Problem 6 Memory Usage



- Python 3 Uses the least memory
- Jython used just over 4 GB then crashed

– Problem 15 –

Starting in the top left corner of a 2x2 grid, there are 6 routes (without backtracking) to the bottom right corner.



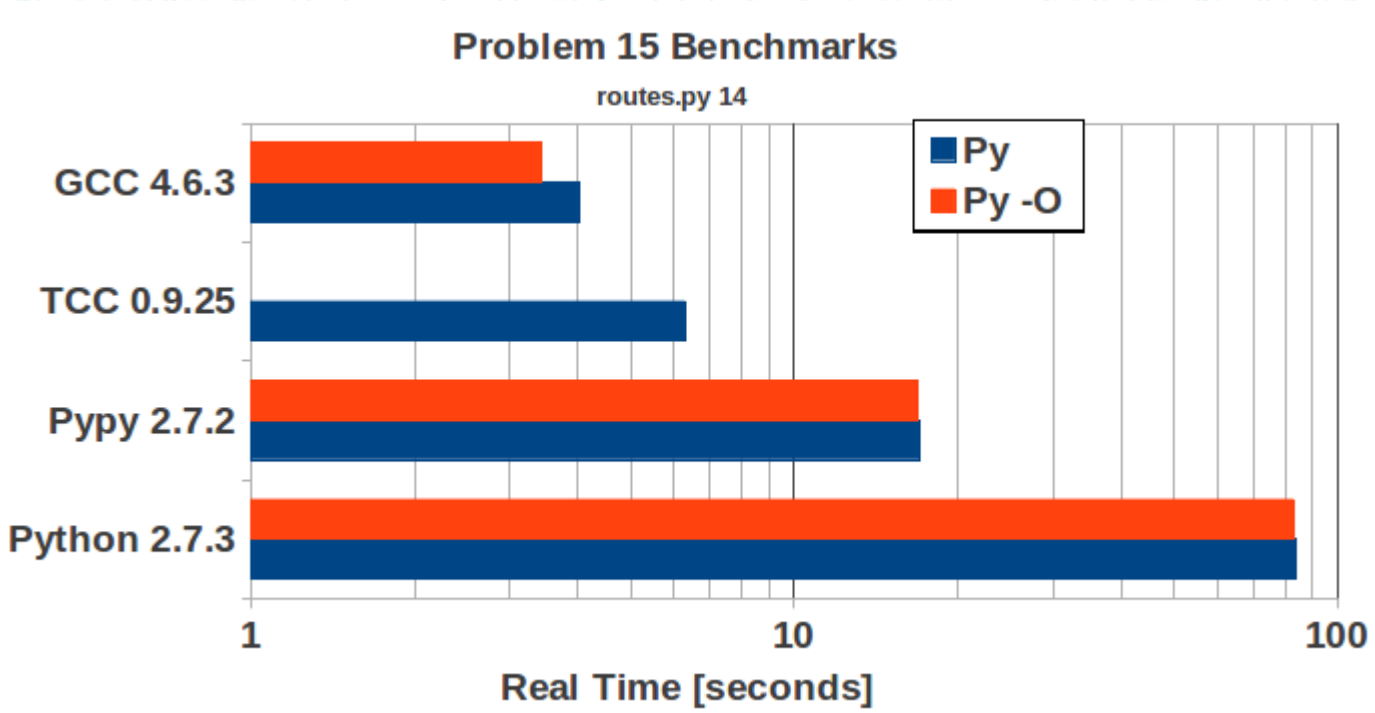
How many routes are there through a 20x20 grid?

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

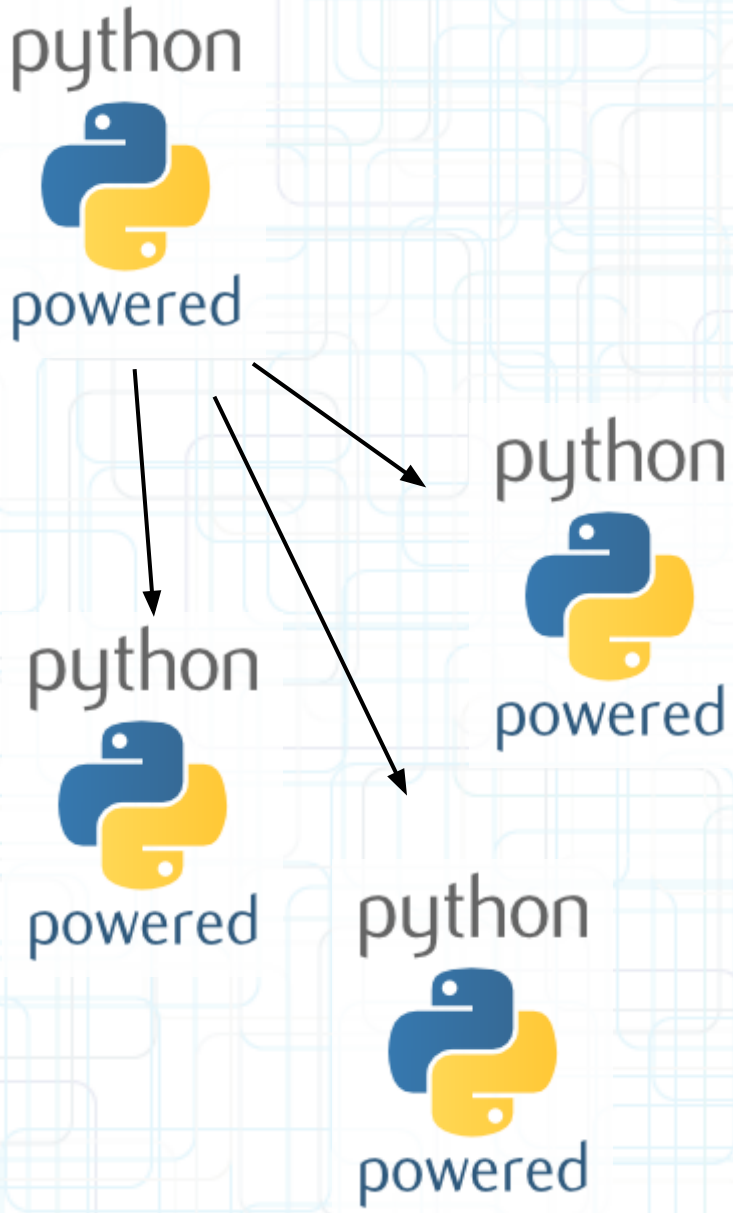
Problem 15 Benchmarks



- 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.



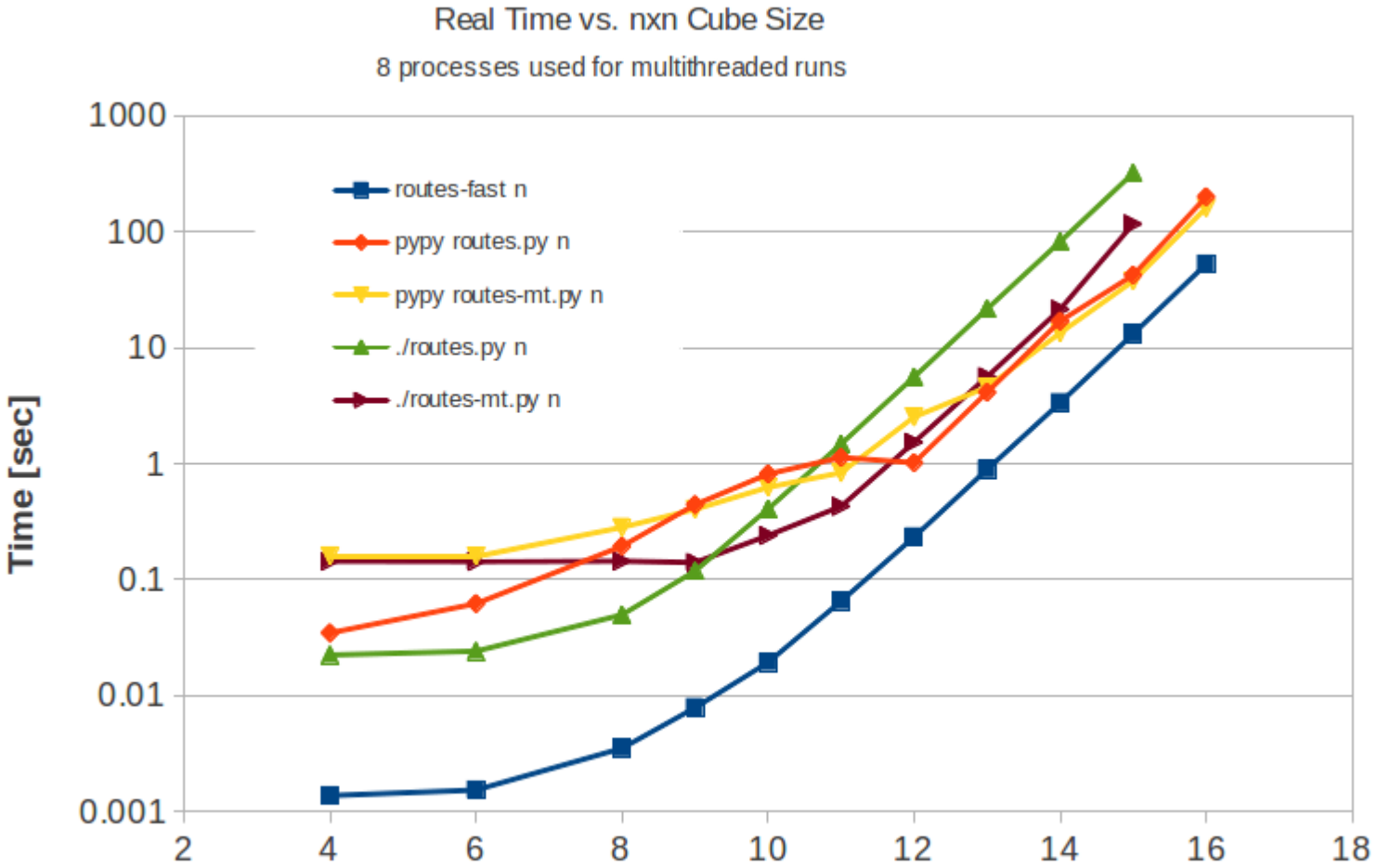
Threading

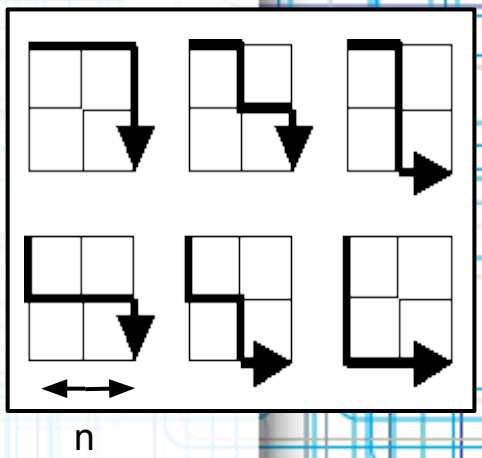
- import threading as t
- t.Thread(group=None, target=aFunction, name=None, args=(), kwargs={})
- t.start(), t.run(), t.join()
  - start() calls run(), which should be defined in your code
  - join() wait soe a thread to complete
- Basic multithreading primitives:
  - t.Lock()
  - t.Semaphore()
  - t.Timer()
  - 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





Greenlets

```
from greenlet import greenlet
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?