Test

Intro to digital Filters

02: C Interfacing

Engr 315: Hardware / Software Codesign
Andrew Lukefahr
Indiana University

Some material taken from:
https://github.com/trekhleb/homemade-machine-learning/tree/master/homemade/neural_network
http://cs231n.github.io/neural-networks-1/

Slack

- Thanks Joe!
- https://join.slack.com/t/engr-315/shared_invite/zt-21z61r228-J6gqPdrlBwnt_0M8IKFQrw

Announcements

- Slack – See Website
- Office Hours – See Website / Syllabus
- P1: Due ~~next~~ Friday
- P2: Ready when you are….

Course Website

engr315.github.io

Write that down!

Project 2: Accelerate Exp. Moving Avg.

- Need Pynq board for ~~P2~~ P3.

- Hope to have those ready by Monday.

deque => Double Ended Queue

Array vs. Linked List: Random Access

```
lst = collections.deque(nums)
arr = np.array(nums)
print (lst)
print (arr)
```

```
deque([5, 1, 9, 0, 3, 2, 6, 4, 8, 7])
[5 1 9 0 3 2 6 4 8 7]
```

```
def traverse( thing, times):
    idx = 0
    for i in range(times):
        nidx = thing[idx]
        print (i, ':', idx, '->', nidx)
        idx = nidx
```

```
trips = 10
traverse(lst, trips)
```

```
0 : 0 -> 5
1 : 5 -> 2
2 : 2 -> 9
3 : 9 -> 7
4 : 7 -> 4
5 : 4 -> 3
6 : 3 -> 0
7 : 0 -> 5
8 : 5 -> 2
9 : 2 -> 9
```

Array vs. Linked List: Random Access

```
def traverse( thing, times):
    idx = 0
    for i in range(times):
        idx = thing[idx]

random.seed(1)
sz = 1000000
nums = [x for x in range(sz)]
random.shuffle(nums)
random.shuffle(nums)
lst = collections.deque(nums)
arr = np.array(nums)
trips = 1000

start_time = time.time()
traverse(lst, trips)
end_time = time.time()
print("True List: %f seconds" % (end_time - start_time))

start_time = time.time()
traverse(arr, trips)
end_time = time.time()
print("Array: %f seconds" % (end_time - start_time))

start_time = time.time()
traverse(nums, trips)
end_time = time.time()
print("Python List: %f seconds" % (end_time - start_time))
```

```
True List: 0.037878 seconds
Array: 0.000312 seconds
Python List: 0.000410 seconds
```

Array vs. Linked List: Sequential Insert

```python
def insert(thing, idx, values):
    print (thing)
    for value in values:
        thing.insert(idx, value)
    print (thing)

random.seed(1)
sz = 10
nums = [x for x in range(sz)]
random.shuffle(nums)
random.shuffle(nums)
lst = collections.deque(nums)
arr = np.array(nums)

idxs = int(sz/2)
insert(nums, idxs, [-1,-2,-3,-4])
```

```
[5, 1, 9, 0, 3, 2, 6, 4, 8, 7]
[5, 1, 9, 0, 3, -4, -3, -2, -1, 2, 6, 4, 8, 7]
```

Array vs. Linked List: Sequential Insert

```
Insert at:  0

True List: 0.000085 seconds
Array: 0.335853 seconds
Python List: 0.115629 seconds

Insert at:  750000
```

~3/4

```
True List: 0.054327 seconds
Array: 0.336377 seconds
Python List: 0.022257 seconds
```

Let’s plot that.

X-axis:

Insert Times 1,10,100,1000(1e3), 1e4, 1e5,1e6,1e7

Y-axis:

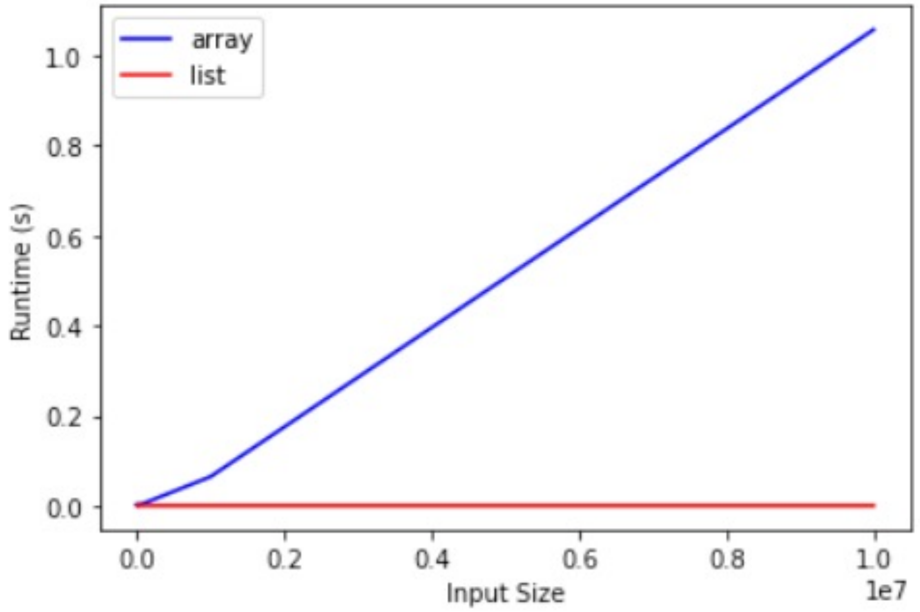
Total runtime for list insert

Total runtime for array insert

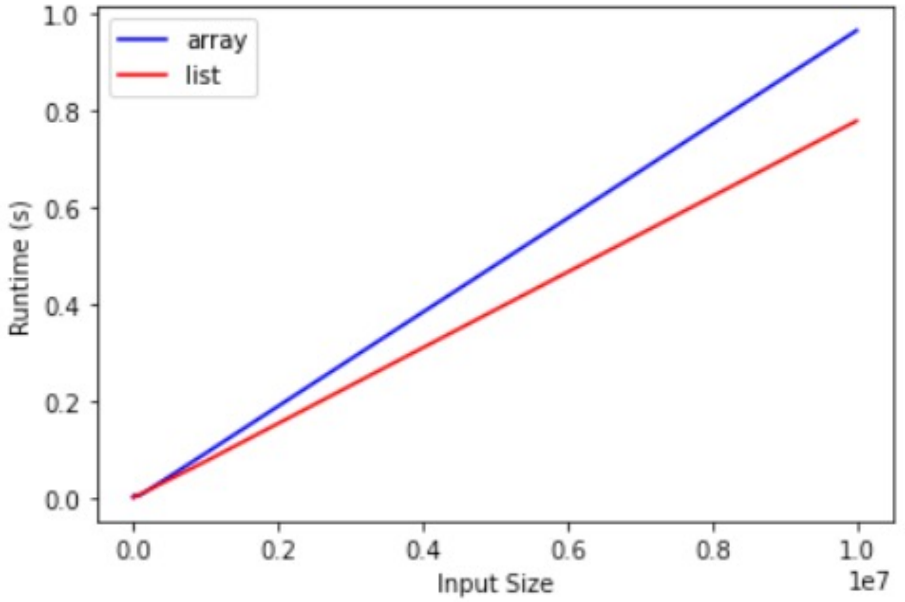
A Runtime Plot

Switch

- Insert at beginning?



- Insert in middle?



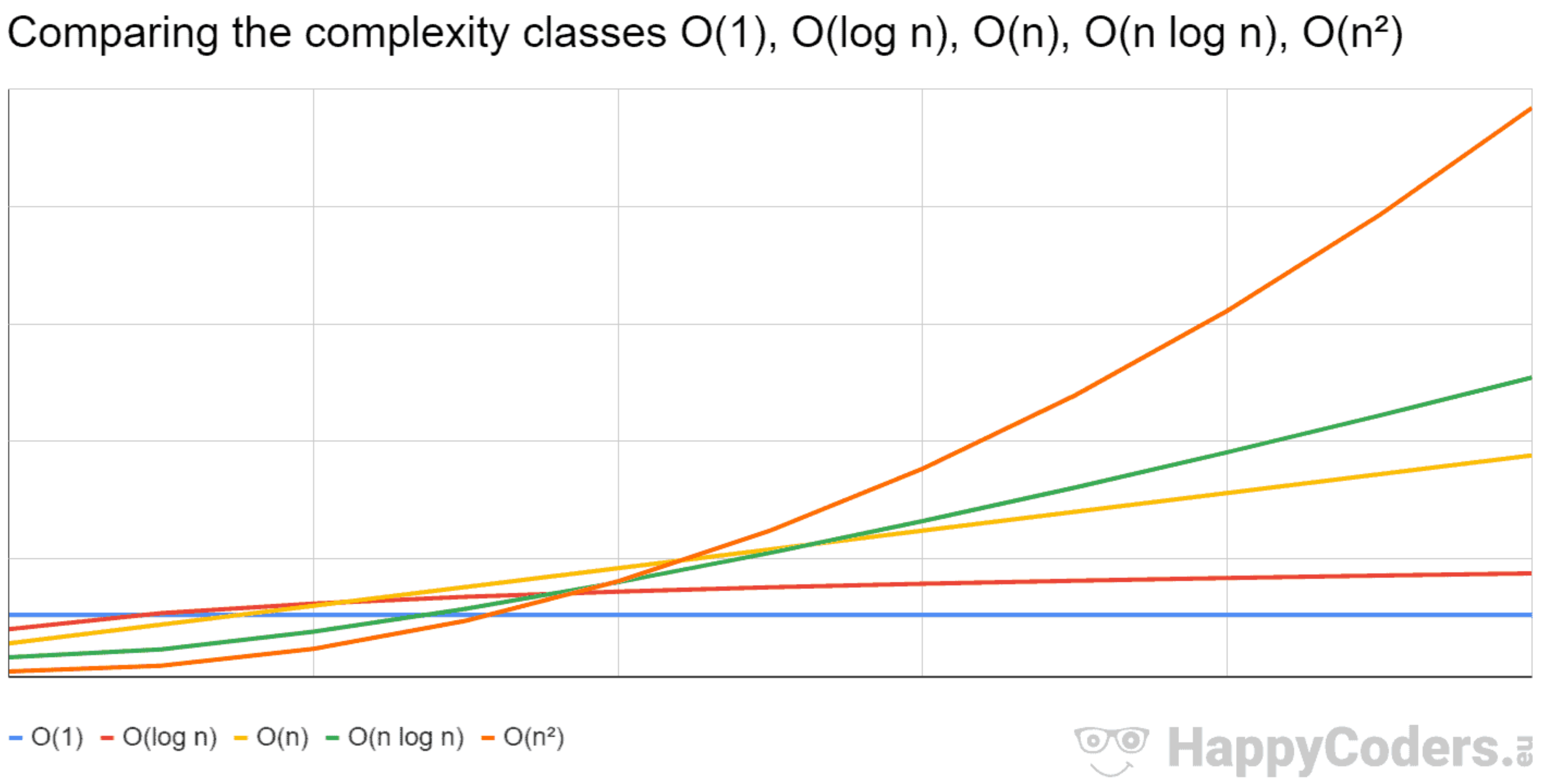
Big O Complexity

- Computational time complexity describes the change in the runtime of an algorithm, depending on the change in the input data's size.

- "How much does an algorithm’s performance change when the amount of input data changes?"

Material taken from: https://www.happycoders.eu/algorithms/big-o-notation-time-complexity/

O() Complexities



Material taken from: https://www.happycoders.eu/algorithms/big-o-notation-time-complexity/

Conclusion #3: Think about your data structure!

- How will you be accessing your data?
  - Randomly? Sequentially?
- How will you up updating your data?

- Pick a data structure to minimize overheads for your access patterns

Find: The needle in the haystack.

```
def find_ignore_case( needle, haystack):
    results = []
    for hi in range(len(haystack)):
        match = True
        for ni in range(len(needle)):
            h = haystack[hi + ni].lower()
            n = needle[ni].lower()
            if h != n:
                match=False
                break
        if match:
            results.append(hi)
    return results
```

```
sz=20
haystack = random_str(sz)
needle = haystack[int(sz/2):int(sz/2)+2]
results = find_ignore_case(needle, haystack)

print (needle)
print (haystack)
print (results)
```

```
sk
eiPPzDAnWiskaumnqYpl
[10]
```

```python
def find_ignore_case4( needle, haystack):
    results = []
    needle = needle.lower() # new
    haystack = haystack.lower() # new
    r = range(len(needle)-1) # new

    for hi in range(len(haystack)-len(needle)):
        #match = False

        if haystack[hi] == needle[0]:
            for ni in r: # update
                h = haystack[hi + ni]#.lower()
                n = needle[ni]#.lower()
                if h == n: # new
                    #match=False
                    results.append(hi) # new
                    break # new
            #if match:
                #results.append(hi)
    return results
```

```
Find: 0.259516 seconds
Find2: 0.057128 seconds
Find3: 0.053053 seconds
Find4: 0.048197 seconds
```

Using built-in libraries is usually the fastest…

```
def find_ignore_case5( needle, haystack):
    return [haystack.find(needle)]
```

```
Find: 0.259516 seconds
Find2: 0.057128 seconds
Find3: 0.053053 seconds
Find4: 0.048197 seconds
Find5: 0.000172 seconds
```

Q: How is this so much faster?

```
def find_ignore_case5( needle, haystack):
    return [haystack.find(needle)]
```

Q: How is this so much faster?

```
def find_ignore_case5( needle, haystack):
    return [haystack.find(needle)]
```

A: It uses a built-in Python library.

Can we even faster?

```
def find_ignore_case5( needle, haystack):
    return [haystack.find(needle)]
```

move title

Numpy is optimized C/Assembly. It's faster.

```
import numpy
def find_ignore_case6(needle, haystack):
    return np.where(haystack==needle)
```

assembly?

```
Find: 0.270210 seconds
Find2: 0.061821 seconds
Find3: 0.054265 seconds
Find4: 0.051191 seconds
Find5: 0.000265 seconds
Find6: 0.000052 seconds
```

E315 Performance Conclusions

1. Optimize algorithm if possible
2. Function calls are not free!
3. Preallocation (Bulk Allocation) is *usually* faster
4. Think about your data structure!
5. Use optimized libraries if possible

Way more optimizations
possible in “Data Structures” class

Popcount

- Count the number of binary 1's in a number
- 01000100101000010010000100000000
- 7 total 1's

Popcount

```
def popcount(num):
    return bin(num).count('1’)
```

5 → "0101"

```
value: 0 bin: 0b0 popcount: 0
value: 1 bin: 0b1 popcount: 1
value: 2 bin: 0b10 popcount: 1
value: 3 bin: 0b11 popcount: 2
value: 4 bin: 0b100 popcount: 1
value: 5 bin: 0b101 popcount: 2
value: 6 bin: 0b110 popcount: 2
value: 7 bin: 0b111 popcount: 3
value: 8 bin: 0b1000 popcount: 1
value: 9 bin: 0b1001 popcount: 2
```

popcount (slower, but no external calls)

dec
11

bin
1011

1011

```
def popcount2(num):
    w = 0
    while (num):
        w += 1
        num &= num - 1
    return w
```

num= num & (num-1)

num	w	num-1
1011	0	1010
1010	1	1001
1000	2	0111
0000	3	—

$$\begin{array}{r} 1011 \\ \&\ 1010 \\ \hline 1010 \end{array} \qquad \begin{array}{r} 1010 \\ \&\ 1001 \\ \hline 1000 \end{array} \qquad \begin{array}{r} 1000 \\ \&\ 0111 \\ \hline 0000 \end{array}$$

`Popcount_all` is a helper function to run larger blocks of inputs

```
def popcount_all(buf):
    return sum(map(popcount,buf))

def popcount2_all(buf):
    return sum(map(popcount2,buf))
```

Big Bitcount

```
np.random.seed(1)
buf = np.random.randint(0,1E9,int(1E6))

start_time = time.time()
sum_1s = popcount_all(buf)
end_time = time.time()
print("popcount: %f seconds (w/libs)"
      % (end_time - start_time))

start_time = time.time()
sum_1s = popcount2_all(buf)
end_time = time.time()
print("popcount2: %f seconds (w/o libs)"
      % (end_time - start_time))
```

```
popcount: 0.307169 seconds (w/libs)
popcount2: 1.853192 seconds (w/o libs)
```

How did the library go so much faster?

- Python called C.
- The computations happened in C. It's faster.
- Can we do that?

Let's find out.

Popcount in Python vs. C

Python

```python
def popcount2(num):

    w = 0
    while (num):
        w += 1
        num &= num - 1

    return w
```

C

```c
int popcount(uint64_t num)
{
    int w=0;
    while (num) {
        w +=1;
        num &= (num -1);
    }
    return w;
}
```

Popcount test?

```
#include <stdio.h>
#include "popcount.h"

int main()
{
    int res;
    for (int i = 0; i < 20; ++i){
        res = popcount(i);
        printf ("i:%d i:0x%x res: %d\n", i, i, res);
    }
    return 0;
}
```

```
i:0 i:0x0 res: 0
i:1 i:0x1 res: 1
i:2 i:0x2 res: 1
i:3 i:0x3 res: 2
i:4 i:0x4 res: 1
i:5 i:0x5 res: 2
i:6 i:0x6 res: 2
i:7 i:0x7 res: 3
i:8 i:0x8 res: 1
i:9 i:0x9 res: 2
i:10 i:0xa res: 2
i:11 i:0xb res: 3
i:12 i:0xc res: 2
i:13 i:0xd res: 3
i:14 i:0xe res: 3
i:15 i:0xf res: 4
i:16 i:0x10 res: 1
i:17 i:0x11 res: 2
i:18 i:0x12 res: 2
i:19 i:0x13 res: 3
```

Let’s see if we can wrap C `popcount` with Python

- https://realpython.com/build-python-c-extension-module/#packaging-your-python-c-extension-module

- https://docs.python.org/3/extending/extending.html

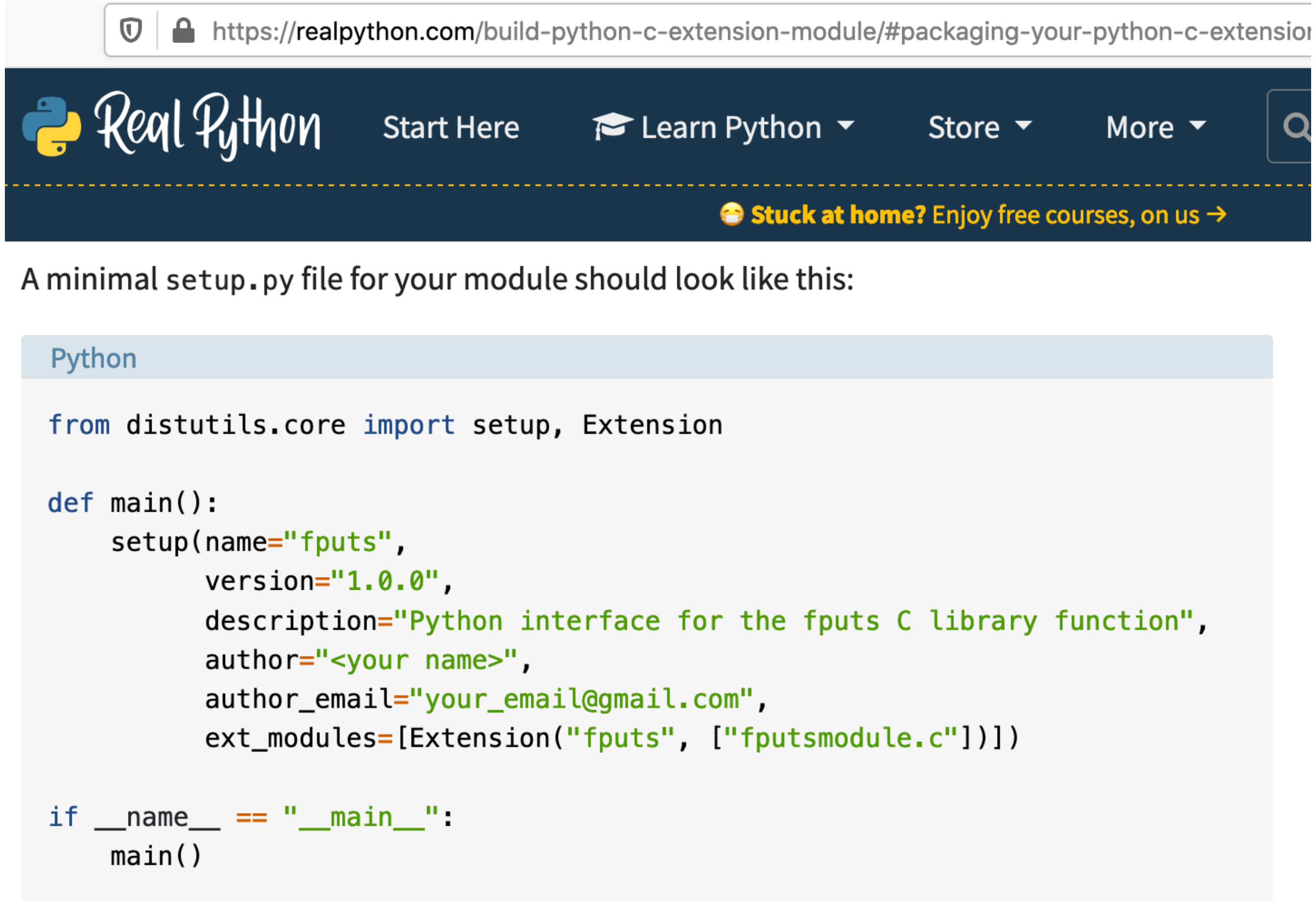
Step 1: RTFM

https://docs.python.org/3/extending/extending.html

```c
static PyObject *
spam_system(PyObject *self, PyObject *args)
{
    const char *command;
    int sts;

    if (!PyArg_ParseTuple(args, "s", &command))
        return NULL;
    sts = system(command);
    return PyLong_FromLong(sts);
}
```

Step 2: RTFM 2



A minimal `setup.py` file for your module should look like this:

Python

```python
from distutils.core import setup, Extension

def main():
    setup(name="fputs",
          version="1.0.0",
          description="Python interface for the fputs C library function",
          author="<your name>",
          author_email="your_email@gmail.com",
          ext_modules=[Extension("fputs", ["fputsmodule.c"])])

if __name__ == "__main__":
    main()
```

```c
static PyObject *
cPopcount(PyObject *self, PyObject *args)
{
    uint64_t num;

    if (!PyArg_ParseTuple(args, "l", &num))
        return NULL;

    //popcount!!!
    uint64_t res = popcount(num);

    return PyLong_FromLong(res);
}
```

```c
int popcount(uint64_t num)
{
    int w=0;
    while (num) {
        w +=1;
        num &= (num -1);
    }
    return w;
}
```

```python
import cPopcount
cPopcount.cPopcount(0xffff)
```

```
16
```

```python
np.random.seed(1)
buf = np.random.randint(0,1E9,int(1E6))
buf = buf.tolist()

start_time = time.time()
sum_1s = popcount_all(buf)
end_time = time.time()
print("popcount: %f seconds (w/calls)"
      % (end_time - start_time))

start_time = time.time()
sum_1s = popcount2_all(buf)
end_time = time.time()
print("popcount2: %f seconds (w/o calls)"
      % (end_time - start_time))

start_time = time.time()
sum_1s = sum(map(cPopcount.cPopcount,buf))
end_time = time.time()
print("c_popcount: %f seconds (64-bits in C)"
      % (end_time - start_time))
```

```
popcount: 0.261108 seconds (w/calls)
popcount2: 0.881429 seconds (w/o calls)
c_popcount: 0.027510 seconds (64-bits in C)
```

Same algo
different lang

Can we do cPopcount_all in C?

- Send an entire list to C?

```c
static PyObject *
cPopcount_all(PyObject *self, PyObject *args)
{
    PyObject *obj;
    int64_t res = 0;

    //parse the list argument
    if (!PyArg_ParseTuple(args, "O", &obj)) {
        return NULL;
    }

    //hope it's iteratable
    PyObject *iter = PyObject_GetIter(obj);
    if (!iter) {
        return NULL;// error not iterator
    }

    //loop over all elements in list
    while (1) {
        PyObject *next = PyIter_Next(iter);

        if (!next) {
            // nothing left in the iterator
            break;
        }
```

```c
        // conver to int64_t
        int64_t num = 0;
        if (PyLong_Check(next)) {
            num = PyLong_AsLong(next);
        } else {
            printf ("unsupported type\n");
            return NULL;
        }

        //now do popcount!
        res += popcount(num);// do something with foo

        /* release reference when done */
        Py_DECREF(next);
    }
    Py_DECREF(iter);

    return PyLong_FromLong(res);
}
```

Two ways to handle lists:

- Iterators (previous slide)
- https://stackoverflow.com/questions/22458298/extending-python-with-c-pass-a-list-to-pyarg-parsetuple
- Array indices (not shown)
- https://stackoverflow.com/questions/39063112/passing-a-python-list-to-c-function-using-the-python-c-api

```python
start_time = time.time()
sum_1s = sum(map(cPopcount.cPopcount,buf))
end_time = time.time()
print("c_popcount: %f seconds (64-bits in C)"
      % (end_time - start_time))

start_time = time.time()
sum_1s = cPopcount.cPopcount_all(buf)
end_time = time.time()
print("c_popcount: %f seconds (List in C)"
      % (end_time - start_time))
```

```
popcount: 0.261108 seconds (w/calls)
popcount2: 0.881429 seconds (w/o calls)
c_popcount: 0.027510 seconds (64-bits in C)
c_popcount: 0.007329 seconds (List in C)
```

Same algorithm. C vs. Python.

```
popcount: 0.261108 seconds (w/calls)
popcount2: 0.881429 seconds (w/o calls)
c_popcount: 0.027510 seconds (64-bits in C)
c_popcount: 0.007329 seconds (List in C)
```

When performance matters, use C.
When it doesn't, use Python.