# Genome 540 Discussion

Conor Camplisson

January 17th, 2023

Genome Sciences
UNIVERSITY OF WASHINGTON

## Outline

- Homework 1 wrap-up
- C/C++/Python programming tips
- Homework 2 overview

## Outline

- Homework 1 wrap-up
- C/C++/Python programming tips
- Homework 2 overview

# Homework #1 Wrap-up

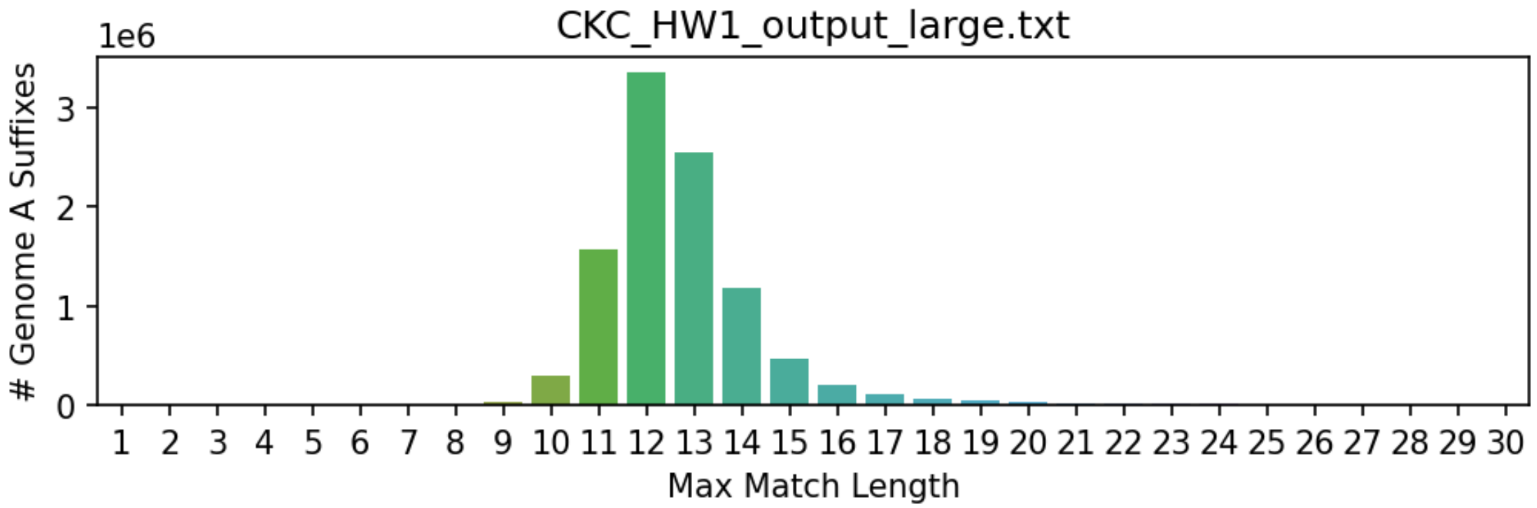
| | |
|---|---|
| $p_{10}$ | AAACCGTACACTGGGTTCAAGAGATTTCCC |
| $p_{11}$ | AACCGTACACTGGGTTCAAGAGATTTCCC |
| $p_{28}$ | AAGAGATTTCCC |
| $p_{17}$ | ACACTGGGTTCAAGAGATTTCCC |
| $p_{12}$ | ACCGTACACTGGGTTCAAGAGATTTCCC |
| $p_{1}$ | ACCTGCACTAAACCGTACACTGGGTTCAAGAGATTTCCC |
| $p_{7}$ | ACTAAACCGTACACTGGGTTCAAGAGATTTCCC |
| $p_{19}$ | ACTGGGTTCAAGAGATTTCCC |
| $p_{29}$ | AGAGATTTCCC |
| $p_{31}$ | AGATTTCCC |
| $p_{33}$ | ATTTCCC |
| $p_{27}$ | CAAGAGATTTCCC |
| | ⋮ |

## Common bugs

- Hist values slightly off
  - Sorting logic, sorted array processing logic
- Genomic coordinate 1-index

## Common debug scenarios

- Too much memory usage
  - Storing substrings (silent caching in python)
- Sort step is too slow
  - Slow comparison function
    - Many comparisons!

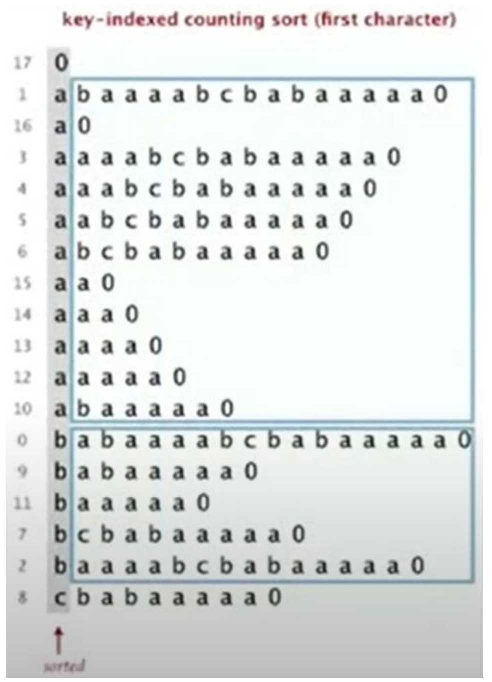


# Homework #1 Wrap-up

| | |
|---|---|
| $p_{10}$ | AAACCGTACACTGGGTTCAAGAGATTTCCC |
| $p_{11}$ | AACCGTACACTGGGTTCAAGAGATTTCCC |
| $p_{28}$ | AAGAGATTTCCC |
| $p_{17}$ | ACACTGGGTTCAAGAGATTTCCC |
| $p_{12}$ | ACCGTACACTGGGTTCAAGAGATTTCCC |
| $p_{1}$ | ACCTGCACTAAACCGTACACTGGGTTCAAGAGATTTCCC |
| $p_{7}$ | ACTAAACCGTACACTGGGTTCAAGAGATTTCCC |
| $p_{19}$ | ACTGGGTTCAAGAGATTTCCC |
| $p_{29}$ | AGAGATTTCCC |
| $p_{31}$ | AGATTTCCC |
| $p_{33}$ | ATTTCCC |
| $p_{27}$ | CAAGAGATTTCCC |
| | ⋮ |

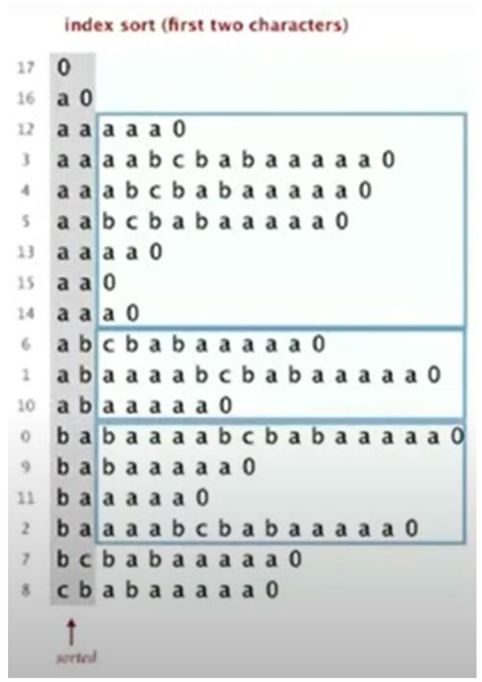
## Variant algorithm(s)

- Sort phases take log N because $2^k$
- Each phase can be linear
- Limited benefit if not rate-limiting (i.e. cache misses)



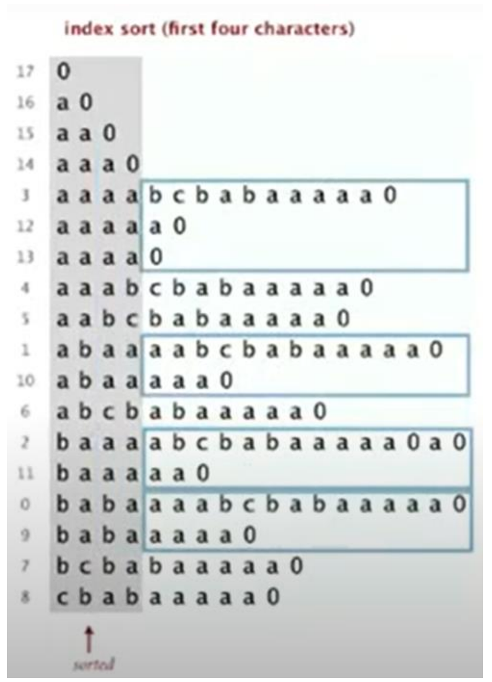
key-indexed counting sort (first character)

```
17  0
 1  a b a a a a b c b a b a a a a a 0
16  a 0
 3  a a a a b c b a b a a a a a 0
 4  a a a b c b a b a a a a a 0
 5  a a b c b a b a a a a a 0
 6  a b c b a b a a a a a 0
15  a a 0
14  a a a 0
13  a a a a 0
12  a a a a a 0
10  a b a a a a a 0
 0  b a b a a a a b c b a b a a a a a 0
 9  b a b a a a a a 0
11  b a a a a a 0
 7  b c b a b a a a a a 0
 2  b a a a a b c b a b a a a a a 0
 8  c b a b a a a a a 0
```



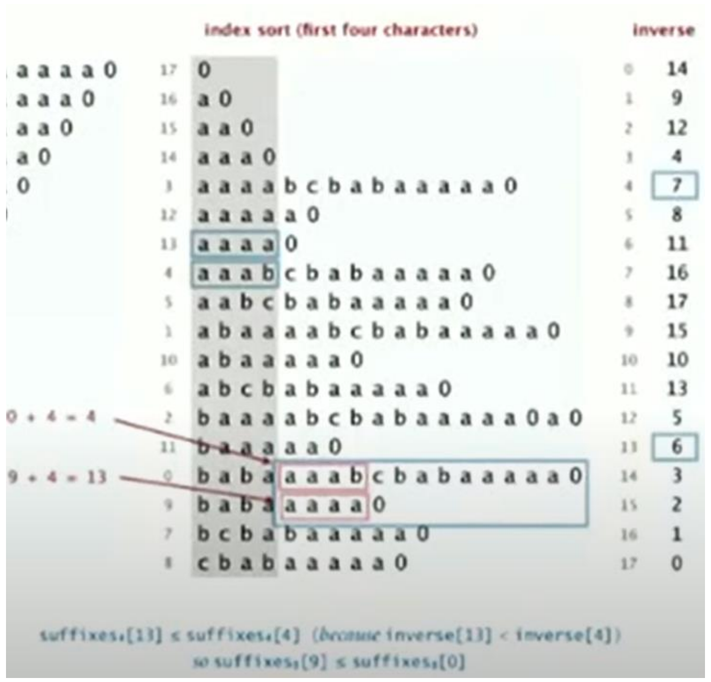
index sort (first two characters)

```
17  0
16  a 0
12  a a a a a 0
 3  a a a a b c b a b a a a a a 0
 4  a a a b c b a b a a a a a 0
 5  a a b c b a b a a a a a 0
13  a a a a 0
15  a a 0
14  a a a 0
 6  a b c b a b a a a a a 0
 1  a b a a a a b c b a b a a a a a 0
10  a b a a a a a 0
 0  b a b a a a a b c b a b a a a a a 0
 9  b a b a a a a a 0
11  b a a a a a 0
 2  b a a a a b c b a b a a a a a 0
 7  b c b a b a a a a a 0
 8  c b a b a a a a a 0
```



index sort (first four characters)

```
17  0
16  a 0
15  a a 0
14  a a a 0
 3  a a a a b c b a b a a a a a 0
12  a a a a a 0
13  a a a a 0
 4  a a a b c b a b a a a a a 0
 5  a a b c b a b a a a a a 0
 1  a b a a a a b c b a b a a a a a 0
10  a b a a a a a 0
 6  a b c b a b a a a a a 0
 2  b a a a a b c b a b a a a a a 0 a 0
11  b a a a a a 0
 0  b a b a a a a b c b a b a a a a a 0
 9  b a b a a a a a 0
 7  b c b a b a a a a a 0
 8  c b a b a a a a a 0
```



index sort (first four characters)

```
a a a a 0    17  0                                     0   14
a a a 0      16  a 0                                   1    9
a a 0        15  a a 0                                 2   12
a 0          14  a a a 0                               3    4
0             3  a a a a b c b a b a a a a a 0         4    7
             12  a a a a a 0                           5    8
             13  a a a a 0                             6   11
              4  a a a b c b a b a a a a a 0           7   16
              5  a a b c b a b a a a a a 0             8   17
              1  a b a a a a b c b a b a a a a a 0     9   15
             10  a b a a a a a 0                      10   10
              6  a b c b a b a a a a a 0              11   13
0 + 4 = 4     2  b a a a a b c b a b a a a a a 0 a 0  12    5
             11  b a a a a a 0                        13    6
9 + 4 = 13    0  b a b a a a a b c b a b a a a a a 0  14    3
              9  b a b a a a a a 0                    15    2
              7  b c b a b a a a a a 0                16    1
              8  c b a b a a a a a 0                  17    0
```

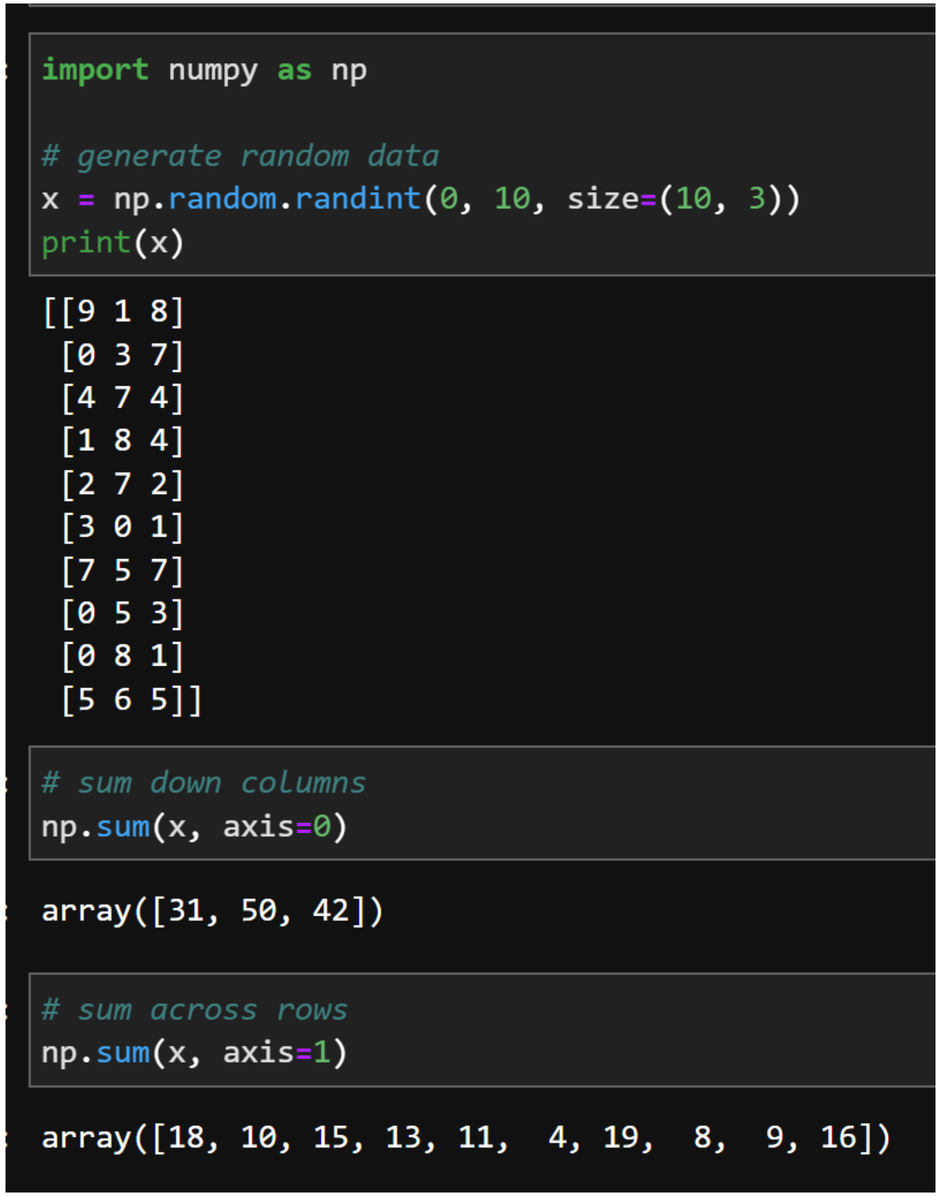
$\text{suffixes}_4[13] \le \text{suffixes}_4[4]$ (*because* inverse[13] < inverse[4])
*so* $\text{suffixes}_8[9] \le \text{suffixes}_8[0]$

http://web.stanford.edu/class/cs97si/suffix-array.pdf
https://www.youtube.com/watch?v=f8S05ZS-8KY&t=703s 11:40 - end

## Outline

- Homework 1 wrap-up
- C/C++/Python programming tips
- Homework 2 overview

# Vectorized array operations in python with numpy



```
import numpy as np

# generate random data
x = np.random.randint(0, 10, size=(10, 3))
print(x)
```

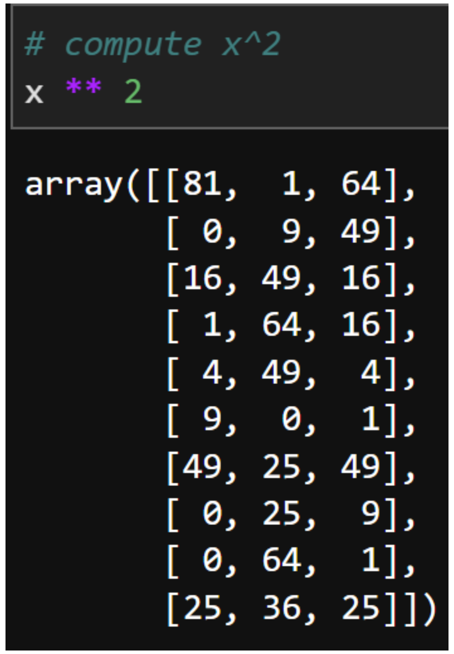
```
[[9 1 8]
 [0 3 7]
 [4 7 4]
 [1 8 4]
 [2 7 2]
 [3 0 1]
 [7 5 7]
 [0 5 3]
 [0 8 1]
 [5 6 5]]
```

```
# sum down columns
np.sum(x, axis=0)
```

```
array([31, 50, 42])
```

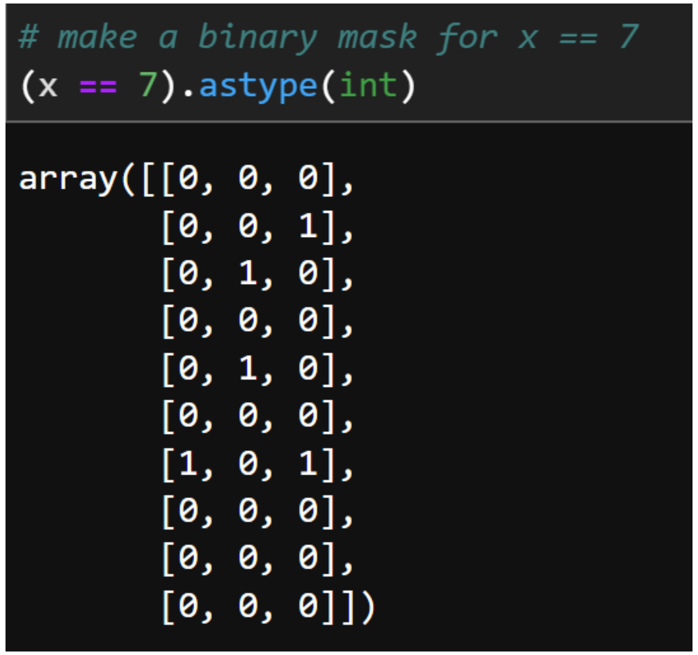
```
# sum across rows
np.sum(x, axis=1)
```

```
array([18, 10, 15, 13, 11,  4, 19,  8,  9, 16])
```



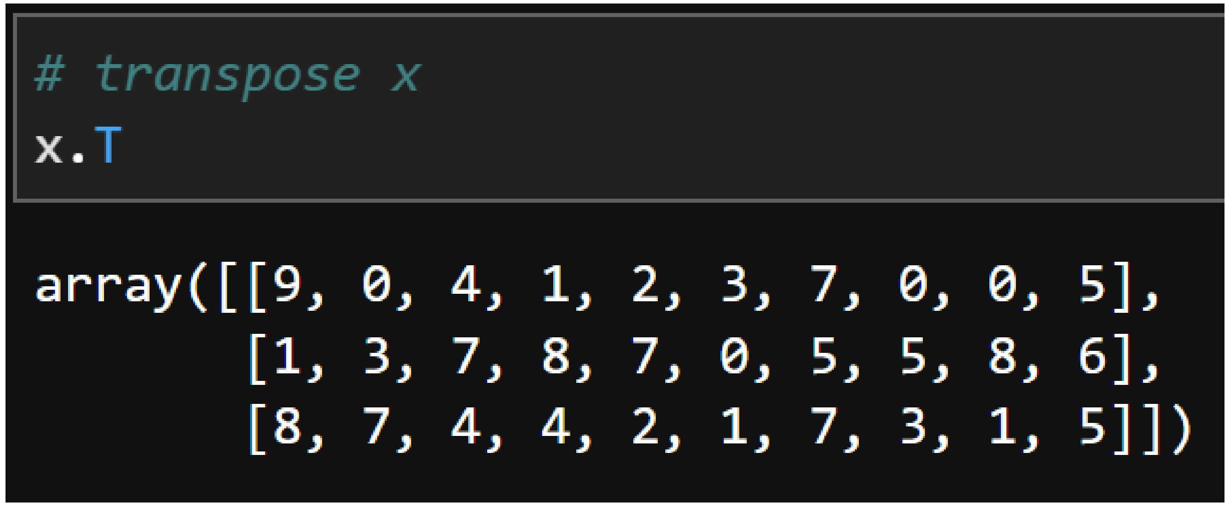
```
# compute x^2
x ** 2
```

```
array([[81,  1, 64],
       [ 0,  9, 49],
       [16, 49, 16],
       [ 1, 64, 16],
       [ 4, 49,  4],
       [ 9,  0,  1],
       [49, 25, 49],
       [ 0, 25,  9],
       [ 0, 64,  1],
       [25, 36, 25]])
```



```
# make a binary mask for x == 7
(x == 7).astype(int)
```

```
array([[0, 0, 0],
       [0, 0, 1],
       [0, 1, 0],
       [0, 0, 0],
       [0, 1, 0],
       [0, 0, 0],
       [1, 0, 1],
       [0, 0, 0],
       [0, 0, 0],
       [0, 0, 0]])
```



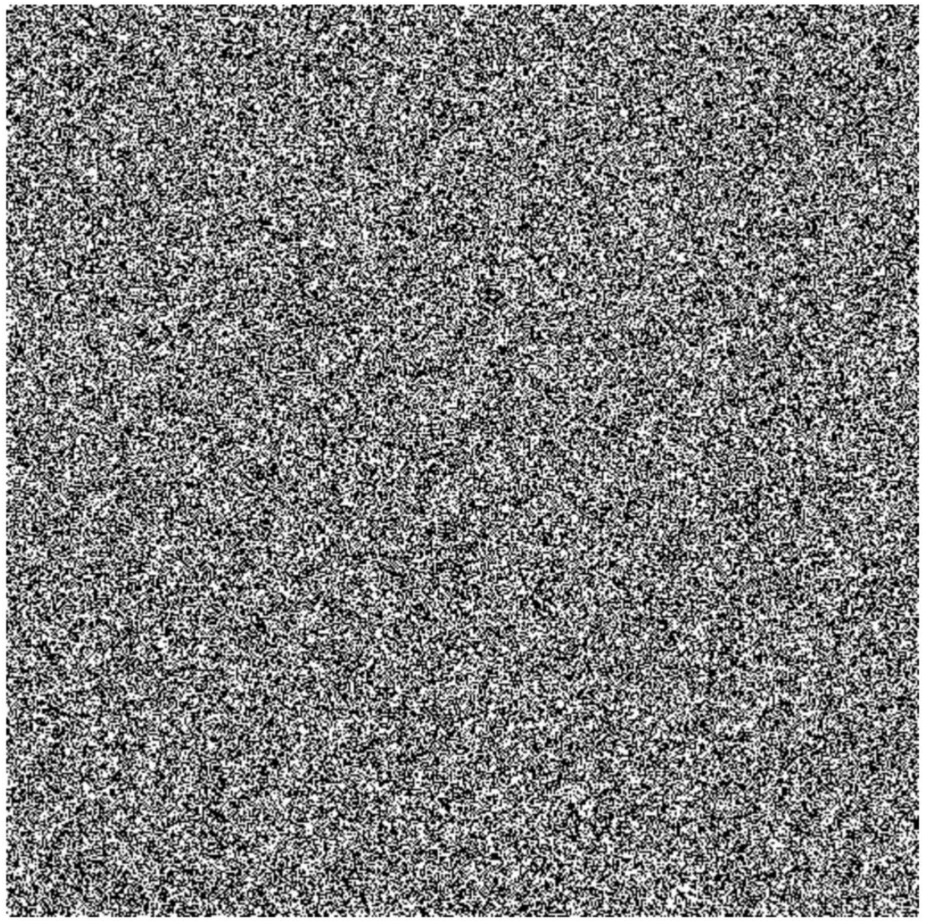
```
# transpose x
x.T
```

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

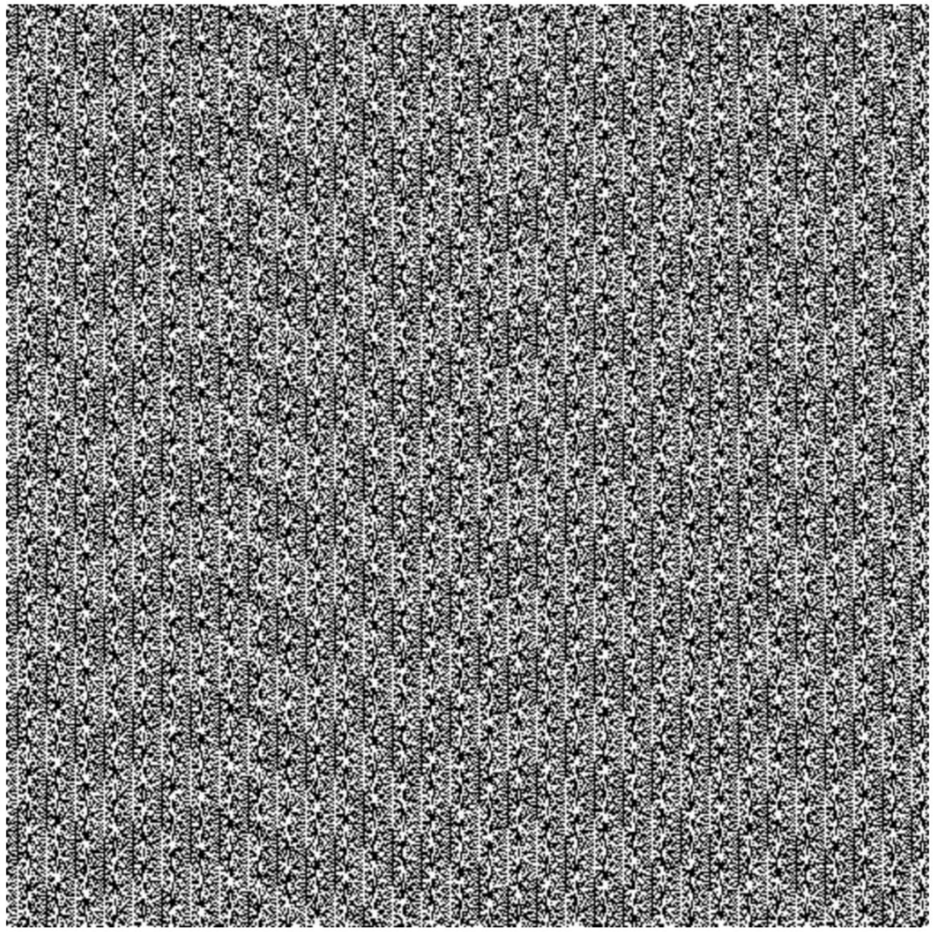
https://numpy.org/install/

## Random number generation

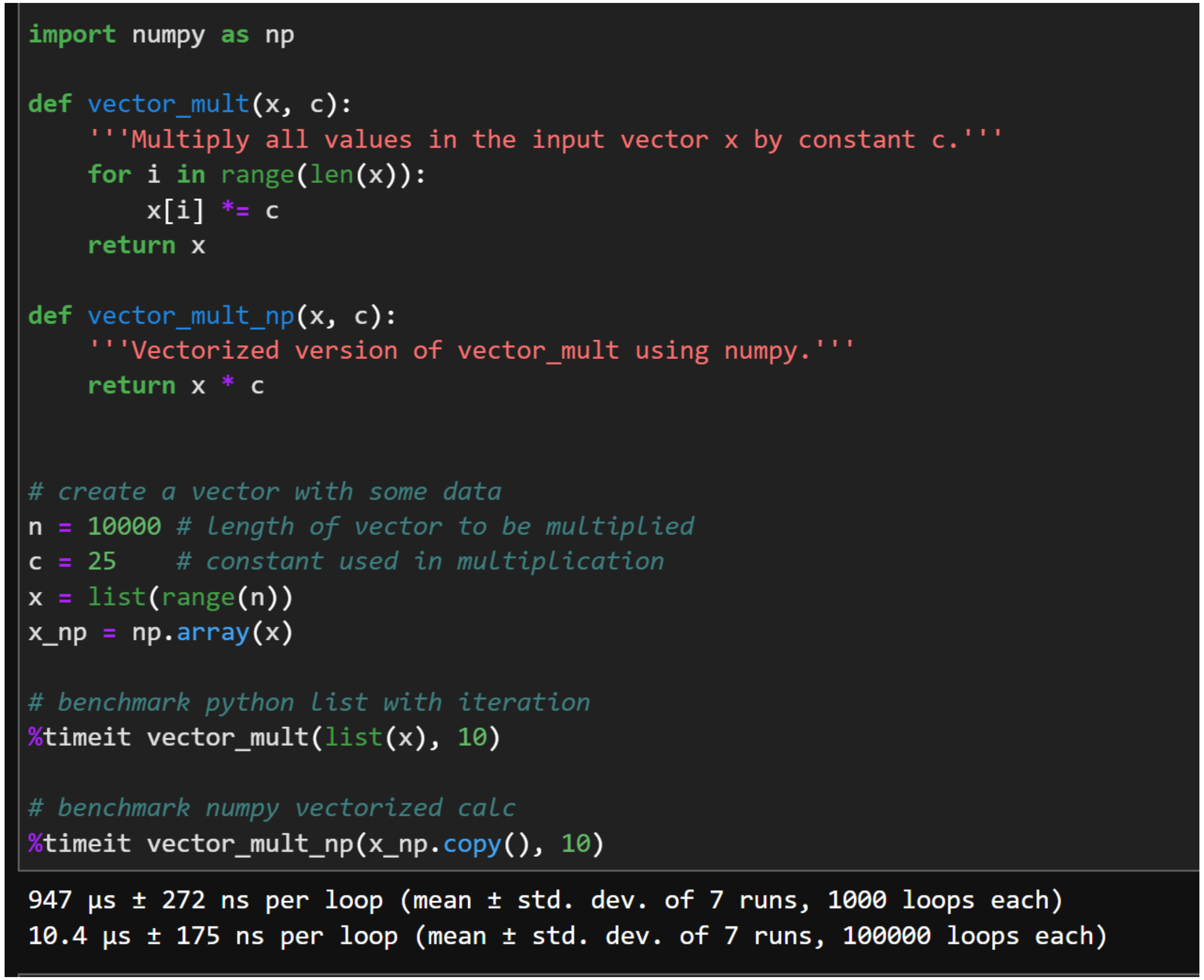
C# System.Random



php rand()

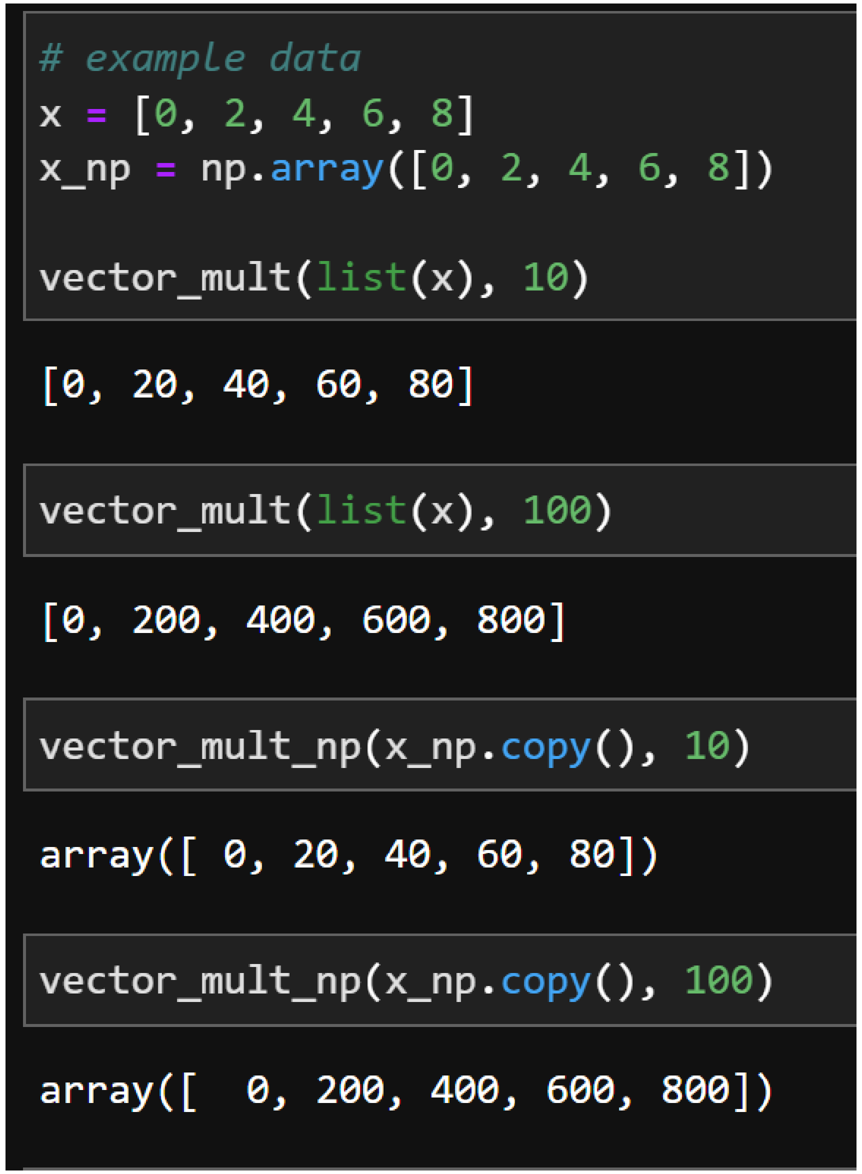


# Vectorized array operations in python with numpy



```
import numpy as np

def vector_mult(x, c):
    '''Multiply all values in the input vector x by constant c.'''
    for i in range(len(x)):
        x[i] *= c
    return x

def vector_mult_np(x, c):
    '''Vectorized version of vector_mult using numpy.'''
    return x * c


# create a vector with some data
n = 10000 # length of vector to be multiplied
c = 25    # constant used in multiplication
x = list(range(n))
x_np = np.array(x)

# benchmark python list with iteration
%timeit vector_mult(list(x), 10)

# benchmark numpy vectorized calc
%timeit vector_mult_np(x_np.copy(), 10)
```

```
947 µs ± 272 ns per loop (mean ± std. dev. of 7 runs, 1000 loops each)
10.4 µs ± 175 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```



```
# example data
x = [0, 2, 4, 6, 8]
x_np = np.array([0, 2, 4, 6, 8])

vector_mult(list(x), 10)
```

```
[0, 20, 40, 60, 80]
```

```
vector_mult(list(x), 100)
```

```
[0, 200, 400, 600, 800]
```

```
vector_mult_np(x_np.copy(), 10)
```

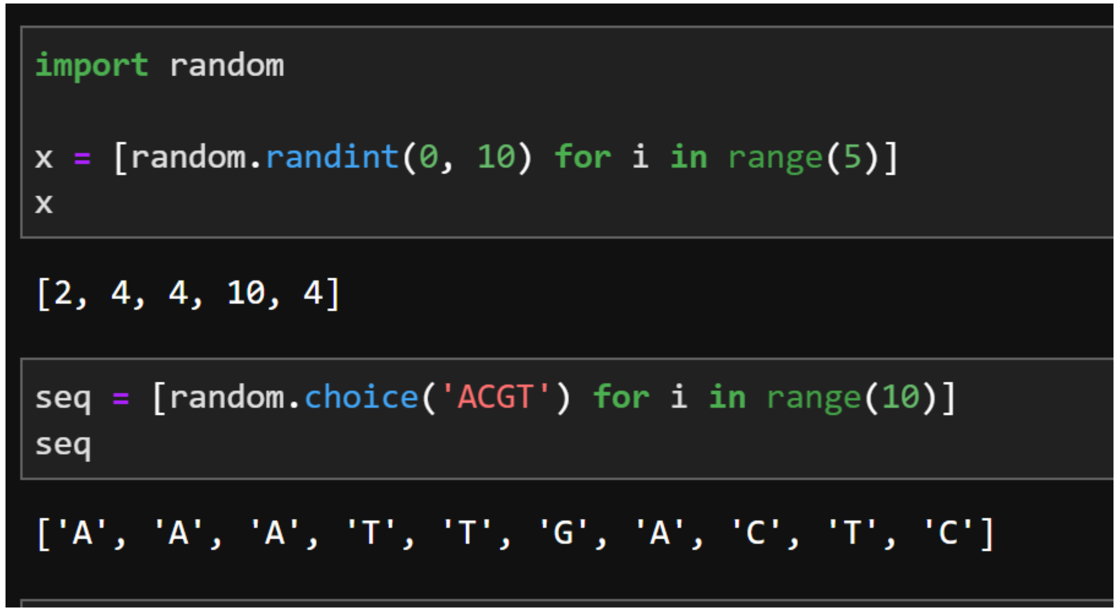
```
array([ 0, 20, 40, 60, 80])
```

```
vector_mult_np(x_np.copy(), 100)
```

```
array([  0, 200, 400, 600, 800])
```

https://numpy.org/install/

# Random number generation in python

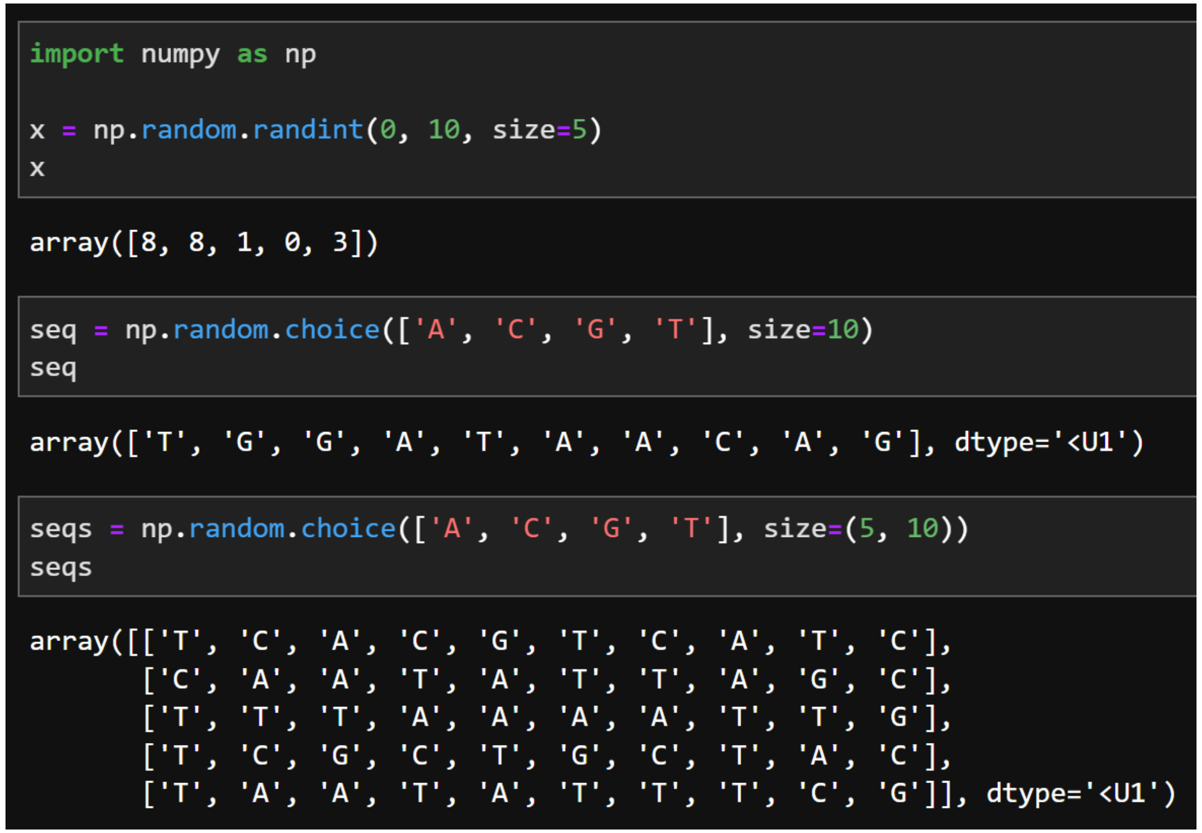


```
import random

x = [random.randint(0, 10) for i in range(5)]
x
```

```
[2, 4, 4, 10, 4]
```

```
seq = [random.choice('ACGT') for i in range(10)]
seq
```

```
['A', 'A', 'A', 'T', 'T', 'G', 'A', 'C', 'T', 'C']
```



```
import numpy as np

x = np.random.randint(0, 10, size=5)
x
```

```
array([8, 8, 1, 0, 3])
```

```
seq = np.random.choice(['A', 'C', 'G', 'T'], size=10)
seq
```

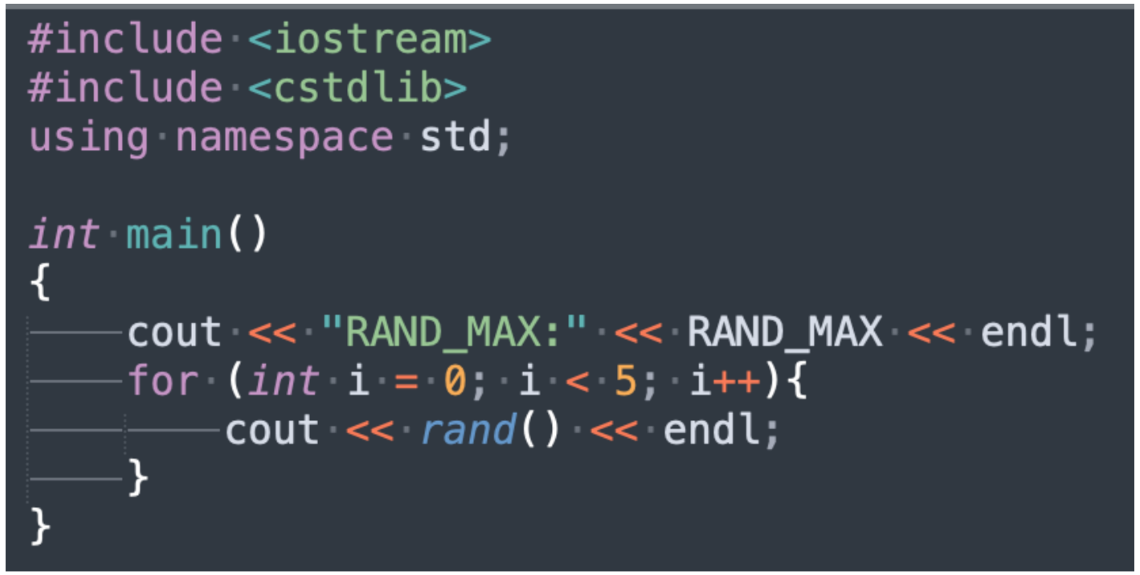
```
array(['T', 'G', 'G', 'A', 'T', 'A', 'A', 'C', 'A', 'G'], dtype='<U1')
```

```
seqs = np.random.choice(['A', 'C', 'G', 'T'], size=(5, 10))
seqs
```

```
array([['T', 'C', 'A', 'C', 'G', 'T', 'C', 'A', 'T', 'C'],
       ['C', 'A', 'A', 'T', 'A', 'T', 'T', 'A', 'G', 'C'],
       ['T', 'T', 'T', 'A', 'A', 'A', 'A', 'T', 'T', 'G'],
       ['T', 'C', 'G', 'C', 'T', 'G', 'C', 'T', 'A', 'C'],
       ['T', 'A', 'A', 'T', 'A', 'T', 'T', 'T', 'C', 'G']], dtype='<U1')
```

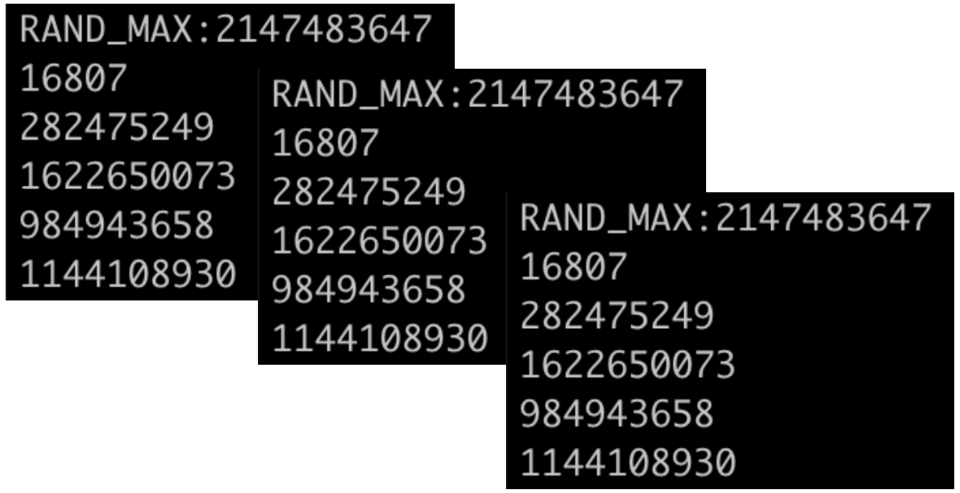
https://numpy.org/install/

# Random number generation in C++



```cpp
#include <iostream>
#include <cstdlib>
using namespace std;

int main()
{
    cout << "RAND_MAX:" << RAND_MAX << endl;
    for (int i = 0; i < 5; i++){
        cout << rand() << endl;
    }
}
```

rand() will return a random number between 0 and RNAD_MAX



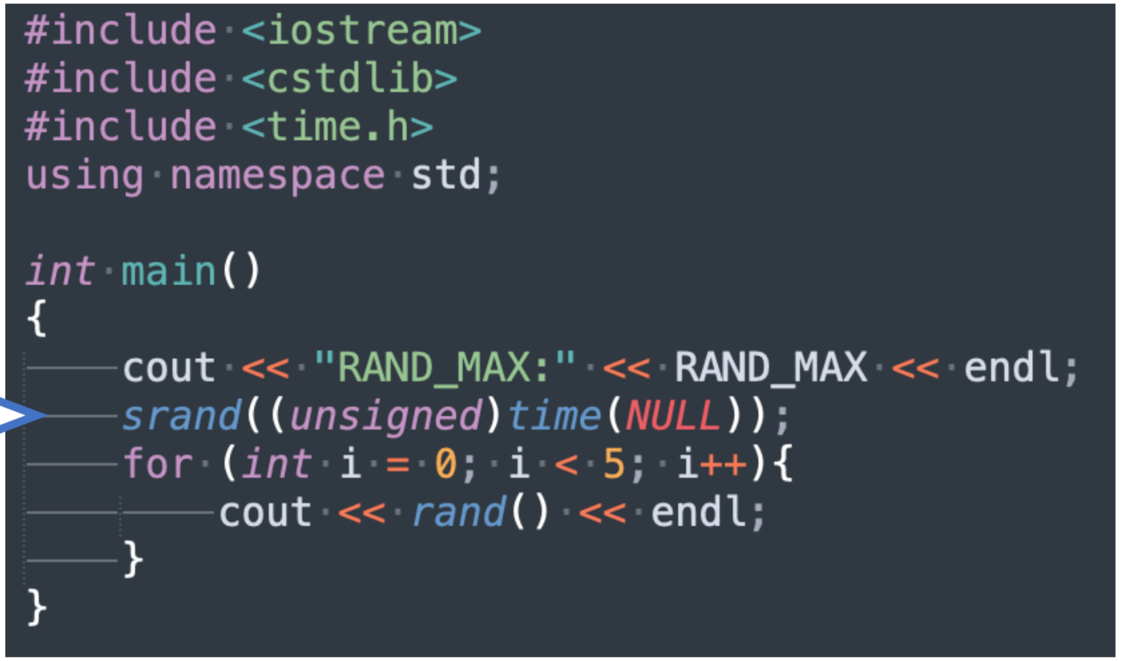
```
RAND_MAX:2147483647
16807
282475249
1622650073
984943658
1144108930
```

```
RAND_MAX:2147483647
16807
282475249
1622650073
984943658
1144108930
```

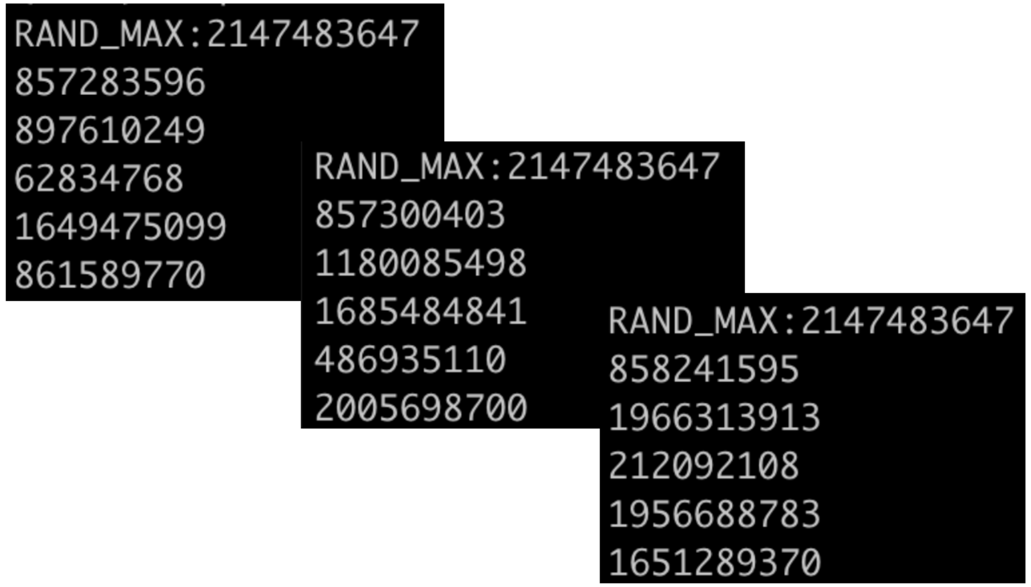
```
RAND_MAX:2147483647
16807
282475249
1622650073
984943658
1144108930
```

Pseudo random number

# Random number generation in C++



```cpp
#include <iostream>
#include <cstdlib>
#include <time.h>
using namespace std;

int main()
{
    cout << "RAND_MAX:" << RAND_MAX << endl;
    srand((unsigned)time(NULL));
    for (int i = 0; i < 5; i++){
        cout << rand() << endl;
    }
}
```



```
RAND_MAX:2147483647
857283596
897610249
62834768
1649475099
861589770
```

```
RAND_MAX:2147483647
857300403
1180085498
1685484841
486935110
2005698700
```

```
RAND_MAX:2147483647
858241595
1966313913
212092108
1956688783
1651289370
```

```cpp
cout << rand()/double(RAND_MAX) << endl;
```

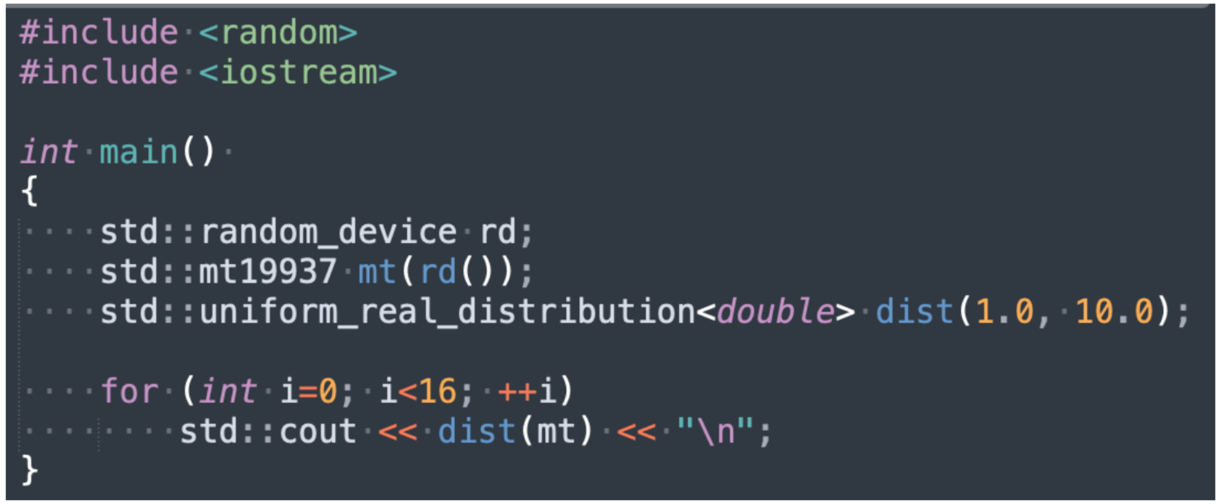
[0, 1]

```cpp
cout << (rand() % (b-a+1))+a << endl;
```

[a, b]

## Random number generation in C++

## Using random() instead of rand()



```cpp
#include <random>
#include <iostream>

int main()
{
    std::random_device rd;
    std::mt19937 mt(rd());
    std::uniform_real_distribution<double> dist(1.0, 10.0);

    for (int i=0; i<16; ++i)
        std::cout << dist(mt) << "\n";
}
```

rand() is typically a low quality pRNG. <random> provides a variety of engines with different characteristics suitable for many different use cases.

% generally biases the data and floating point division still produces non-uniform distributions. <random> distributions are higher quality as well as more readable.

srand() only permits a limited range of seeds. Engines in <random> can be initialized using seed sequences which permit the maximum possible seed data.

## Outline

- Homework 1 wrap-up
- C/C++/Python programming tips
- Homework 2 overview

# Homework 2 Overview

Part one: write a new program

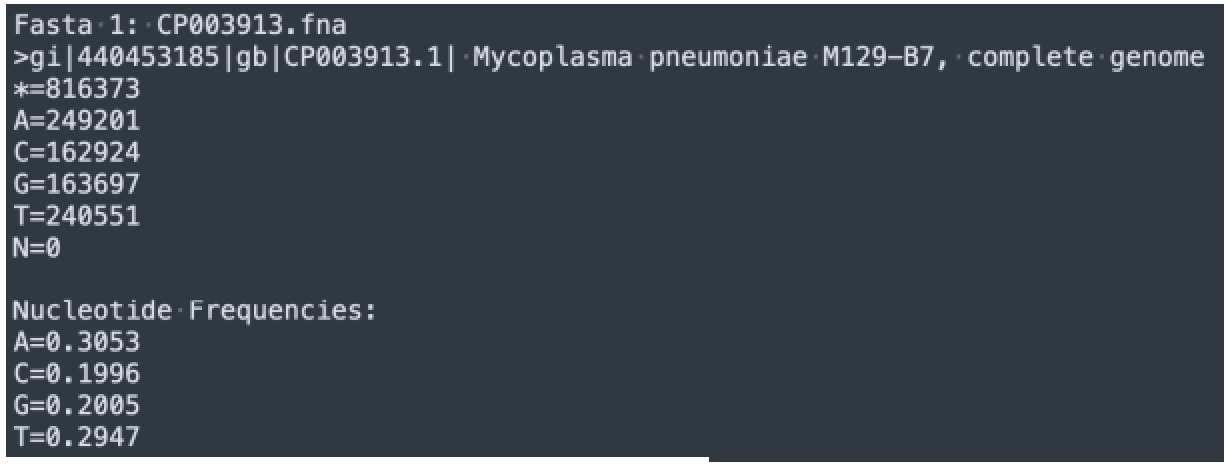
- read in a file in FASTA format
- determine the frequencies of the nucleotides and dinucleotides (based on the forward strand) and the length of the sequence
- produce three simulated sequences based on the length and nucleotide or dinucleotide frequency of the original sequence
    - ‘Equal frequency’ model
    - Order 0 Markov model
    - Order 1 Markov model
- output three files in FASTA format containing the simulated sequence

# Homework 2 Overview

## order-0 Markov

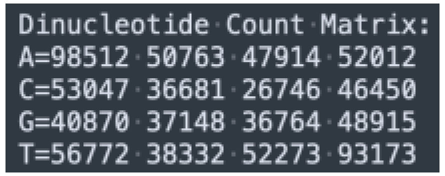
## “Equal frequency” model

```
Nucleotide Frequencies:
A=0.2500
C=0.2500
G=0.2500
T=0.2500
```



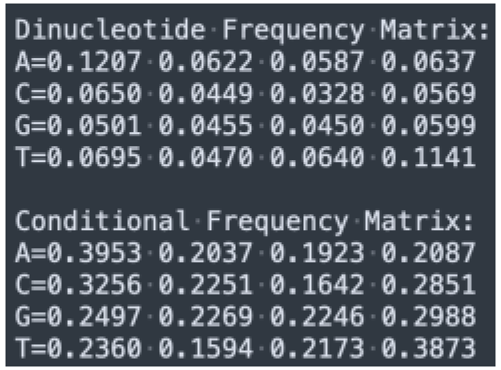
```
Fasta 1: CP003913.fna
>gi|440453185|gb|CP003913.1| Mycoplasma pneumoniae M129-B7, complete genome
*=816373
A=249201
C=162924
G=163697
T=240551
N=0

Nucleotide Frequencies:
A=0.3053
C=0.1996
G=0.2005
T=0.2947
```

## order-1 Markov



```
         A     C     G     T
  Dinucleotide Count Matrix:
A A=98512 50763 47914 52012
C C=53047 36681 26746 46450
G G=40870 37148 36764 48915
T T=56772 38332 52273 93173
```

CGACTA



```
Dinucleotide Frequency Matrix:
A=0.1207 0.0622 0.0587 0.0637
C=0.0650 0.0449 0.0328 0.0569      = 1
G=0.0501 0.0455 0.0450 0.0599
T=0.0695 0.0470 0.0640 0.1141

Conditional Frequency Matrix:
A=0.3953 0.2037 0.1923 0.2087      = 1
C=0.3256 0.2251 0.1642 0.2851      = 1
G=0.2497 0.2269 0.2246 0.2988      = 1
T=0.2360 0.1594 0.2173 0.3873      = 1
```

# Homework 2 Overview

Part two: run your HW1 program on three simulated genomes

- Run your HW1 program three times, using as input:
  - Human 10-Mb segment + simulated ‘equal frequency’ genome
  - Human 10-Mb segment + simulated Mouse Markov-0
  - Human 10-Mb segment + simulated Mouse Markov-1

- Given observed matches between the Human and simulated genomes, what can you conclude about the statistical significance of matches between the orthologous mouse and human regions in homework 1?

## Reminders

- Homework 2 due this Sunday Jan. 22, 11:59 pm
  - Single text file, compressed with `gzip`
  - name in the file: `camplisson_hw2.txt.gz`

- Homework 3 will be posted tomorrow (Jan. 18)