# bio334_day2_part2

May 13, 2025

## 1 Bio334 Practical Bioinformatics

The 2nd module, 14-16, May, 2025

### 1.1 Masaomi Hatakeyama

- GitHab https://github.com/masaomi/bio334_2025
- TAs: Narjes Yousefi, Kenji Yip Tong

## 2 Day2 Part2

Function, Method

[ ]:
```
import IPython.display
IPython.display.Audio("voice/day2_part2.mp3")
```

Let me introduce another technique to improve reusability. It is function.

Function or method in computer programming is a block of code.

The function has input and output. The input value is called an argument, and the output value is called the return value.

Just please quickly review what you did yesterday and go to the explanation of the function.

## 3 Quick review

1. Sequence (pairwise) comparison
2. Combination list and double loop

- Triple loop for combination list + sequence comparison

3. Nucleotide diversity formula

## 4 Nucleotide Diversity

- $d$: number of nucleotide differences'))
- $n$: number of sequences
- $l$: length of sequence

## 5 Nucleotide Diversity, Mean pairwise difference

$\Pi = \frac{\sum_{i<j} d_{ij}}{{}_n\mathrm{C}_2}, \pi = \frac{\sum_{i<j} d_{ij}}{{}_n\mathrm{C}_2 l}$

- $\Pi$: Mean pairwise difference
- $\pi$: nucleotide diversity

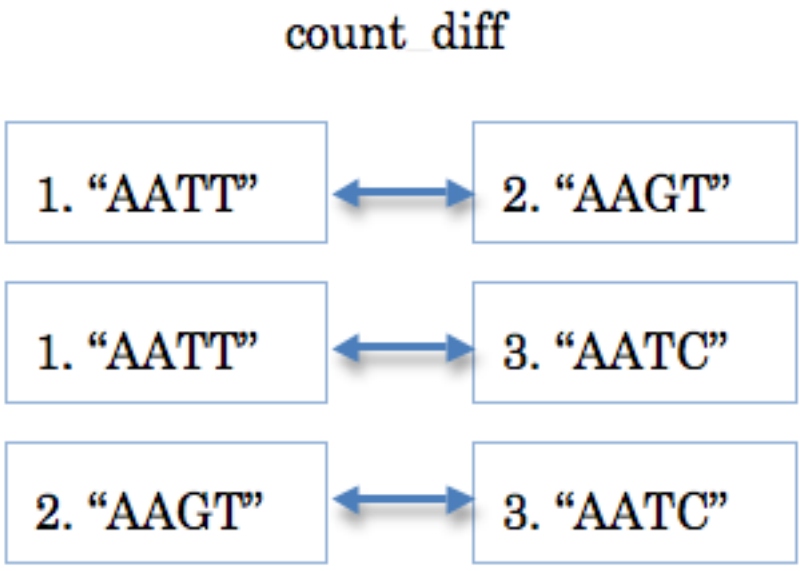
## 6 Algorithm

General Flow

1. load an aligned FASTA file
2. **pairwise comparison of sequences**
3. **count the number of different nucleotide**
4. sum them up
5. divide by the num of combination & seq length

## 7 Break it down

Pairwise comparison, double loop




## 8 Combination

Pairwise comparison, double loop

```
list = [seq1, seq2, seq3, seq4, seq5]
```

```
for i in range(0, len(list)):
    for j in range(i+1, len(list)):
            print(list[i], list[j])
```

- Remember the exercise yesterday

[ ]:
```
seq1 = "seq1"
seq2 = "seq2"
seq3 = "seq3"
seq4 = "seq4"
seq5 = "seq5"

lst = [seq1, seq2, seq3, seq4, seq5]
for i in range(0, len(lst)):
    for j in range(i+1, len(lst)):
        print(lst[i], lst[j])
```

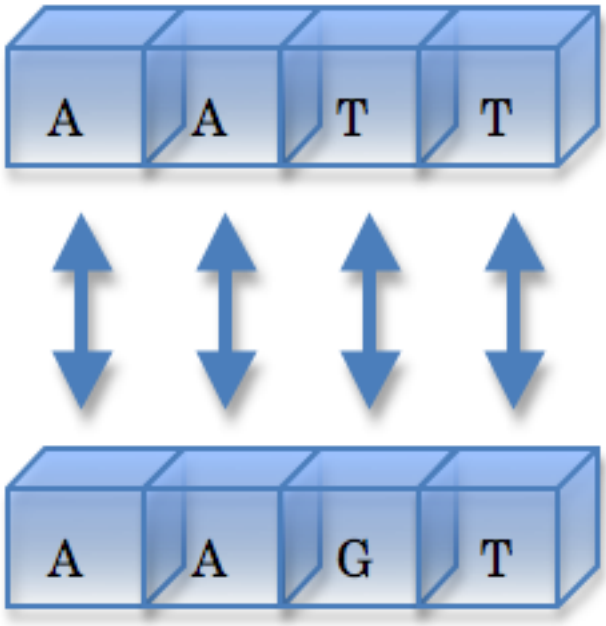
## 9 Break it down

Count the number of different nucleotide at the same position

1. Focusing on only two sequences
2. Compare nucleotide in each position
3. If they are different, count it up

## 10 Compare two sequences

for + if




## 11 Compare two sequences (for + if)

```
seq1 = "ATGC"
seq2 = "ATAT"
same = 0; diff = 0;
for i in range(0, len(seq1)):
    if seq1[i] == seq2[i]:
        same += 1
    else:
        diff += 1
```

[ ]:
```
seq1 = "ATGC"
seq2 = "ATAT"
same = 0; diff = 0;
for i in range(0, len(seq1)):
    if seq1[i] == seq2[i]:
        same += 1
    else:
        diff += 1
print(diff)
```

[ ]:
```
# Compare two sequences
# without loop

seq1 = "ATGC"
seq2 = "ATAT"
```

```
same = 0
if seq1[0] == seq2[0]:
    same += 1
if seq1[1] == seq2[1]:
    same += 1
if seq1[2] == seq2[2]:
    same += 1
if seq1[3] == seq2[3]:
    same += 1

print(same)
```

## 12 Core part, Nucleotide diversity

```
seqs = [seq1, seq2, seq3]
total_diff = 0
for i in range(0, len(seqs)):
    for j in range(i+1, len(seqs)):
        for k in range(0, len(seqs[i])):
            if not seqs[i][k] == seqs[j][k]:
                total_diff += 1
```

[ ]:
```
seq1 = "ATGC"
seq2 = "ATAT"
seq3 = "ATGT"

seqs = [seq1, seq2, seq3]
total_diff = 0
for i in range(0, len(seqs)):
    for j in range(i+1, len(seqs)):
        for k in range(0, len(seqs[i])):
            if not seqs[i][k] == seqs[j][k]:
                total_diff += 1
print(total_diff)
```

## 13 based on these things...

## 14 Define a function

```
def <function name>(<arguments>):
    # process
    return <return value>
```

- **Function** is a small block of processes
  - You can generalize a process
  - You can separate *data (variable)* and *process*
- **Argument** is input and **return value** is output of function

[ ]:
```
IPython.display.Audio("voice/function.mp3")
```

When you use a function, you have to define the function before calling it.

When you define a function, you need the *def* keyword and its function name. If you need some arguments, you can put it as a variable in round brackets.

The process needs indentation and the *return* keyword sets the result of the process, which becomes the return value. You can omit the return value.

You can see three examples below. The first example has no argument and no return value.

The second example has an argument but no return value.

The final example has both an argument and a return value.

Please look at carefully both fucntion definion and function calling, how to call the function.

[ ]:
```
# Example1
# Without an argument and return value

def hello():
    print("hello, world!!")

hello()
```

[ ]:
```
# Example2
# With an argument and without return value

def hello(word):
    print("hello,", word)

hello("python!!")
```

[ ]:
```
# Example3
# With an argument and return value

def hello(word):
    sentence = "hello, %s!!" % word
    return sentence


x = hello("bio334")
print(x)
```

## 15 Core part, Nucleotide diversity

```
seqs = [seq1, seq2, seq3]
total_diff = 0
for i in range(0, len(seqs)):
    for j in range(i+1, len(seqs)):
```

```
            for k in range(0, len(seqs[i])):
                if not seqs[i][k] == seqs[j][k]:
                    total_diff += 1
```

Q: How do you design a function?

## 16 Mini-Summary

- Function is a block of code
- Define a function before calling the function
- A funciton can have **argument(s)** and **return value**
- A function makes the code more strucutred and human readable

[ ]:
```
IPython.display.Audio("voice/summary_day2_part2.mp3")
```

Now let's think about how you can define a function, a block of code.

It is actually the generalization of the process.

I just give you some hints here.

There are some solutions. You can define any block of code as a function, but there are some tips or criteria to define a function.

For example.

- If you do copy and paste more than twice, that it can be a function.
- If you think of a lot of comments, it must be a function.
- If you see more than twice nested loop, it can be a function.
- If your code becomes longer than 100 lines or more than one screen, it would be better to define a function even if the function is called only once.

These are just my recommendations, and you do not have to do so, but just please keep in mind as a hint, and let's go on to the exercise.

## 17 How should we design a function?

generalization, *it is a sign to make a function if you notice the followings…*

- **Reusable**: If you do copy&paste more than twice
- **Break down**: If you need many comments
- **Nested loops**: If you need more than triple nested loops
- **Long spaghetti**: If your code becomes more than 100 line (one screen)