bio334_day2_part1

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 Quieck Review

1. Double loop for the combination of Lists
2. *for* + *if* for the comparison of two sequences
3. Nucleotide diversity (Advanced exercise yesterday, a milestone today)

3 Tips

1. **Deduction**: Drawing specific conclusions logically from general principles or laws.
2. **Induction**: Observing multiple specific instances to identify patterns and generalize conclusions.
3. **Abduction**: Formulating the most hypothesis based on observed data, often involving intuitive leaps or "Aha!" moments.

4 Nucleotide Diversity

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

5 Nucleotide Diversity, Mean pairwise difference

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

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

- $ $:Mean pairwise difference
- $ $: nucleotide diversity

6 Today

1. Part1(9-10) File I/O
2. **Part2(10-11) Method, PI** (*Milestone*)
3. Part3(11-12) Segregating site
4. **Part4(13-15) Tajima's D** (*Milestone*)
5. Part5(15-17) Batch process, module

6.1 Lectures (*plan*)

1. 9:00- (10min)
2. 10:00- (10min)
3. 11:00- (10min)
4. 13:00- (10min)
5. 15:00- (10min)

[ ]:
```
import IPython.display
IPython.display.Audio("voice/day2_part1.mp3")
```

Today, we will focus on more details and applications.

The code will become more complicated, but just keep in mind one thing.

If you get confused with the code, please break it down into smaller parts, elements, and think the process logically step by step, and integrate the piece of blocks as a whole.

Breaking and integration. That's the most important thing in both science and programming.

7 Why programming?

1. **Reusability**: Reproducibility, you can calculate with just one command type again
2. **Batch process**: Automation, computer can work while you are sleeping
3. **Understanding**: It helps you to learn algorithms behind a problem, and it improves your logical thinking ability

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

First of all, I have a question.

Why are you learning computer programming?

In my opinion, there are three good points you should keep in mind in computer programming regardless of programming language.

Reusability, batch processing, and understanding.

The programmining code is reusable. You can use the source code many times once you make it, you can use the source code again in another time, situation, repeatedly. This will maintain the reproducibility of the calculation result.

The programming is batch process. A computer program processes many things at once, in other words, the many complicated processes are automized.

Making a programming code improves your understanding. By implementing an algorithm, calculation steps, you understand it more concretely. implementing the code means that you understand the process well.

This will be proven after you finish this module, and compare your knowledge before and after the course. Hopefully, you will feel that you get a more clear idea about nucleotide diversity than before.

8 Reusability

How can you improve the reusability? (besides copy&paste)

- Generalization of **Process**
  - $ $ Separation of **Data** and **Process**

What does this mean?

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

How can you improve the reusability?

The key concept is the genelization.

In order to generalize a process in computer programming, you need to clearly separate the data and process.

What does this mean?

Let's look at the example below.

9 Command line argument

Warming up exercise

```
import sys
print(sys.argv)

# Result
# $ python day2_1_example1.py 123 abc
# ['day2_1_example1.py', '123', 'abc']
```

- *sys.argv* is a List object
- An argument becomes a String object

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

The first example shows the separation of command-line argument and script.

It is difficult to show you an example in the Jupyter notebook, so just please try to execute the example in the terminal.

When you execute a Python script, you can put additional input data to the Python script. It is called a command-line argument.

The external data can be used in the Python script.

The next example is another separation of data and process.

10 File Input

```
import sys
file = open("input.fa")
for line in file:
    print(line.rstrip())
file.close()
```

- *for* statement reads line by line assigning line data to the variable
- *rstrip()* removes line break from the line
- *open()* and *close()* are needed

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

This example shows the file loading.

Loading a text file and just showing the file contents by *print()* function.

This is also the separation of data and process. The python script is the process, and the fasta file is the input data.

By separating data and process, you can use this Python code many times for other fasta files without changing the code.

In other words, the reusability is improved by separating data and process.

[ ]:
```
import sys
file = open("input.fa") # input.fa is expected to be uploaded in jupyterhub␣
 ↪folder
for line in file:
    print(line.rstrip())
file.close()
```

[ ]:
```
# Another example
import sys
with open("input.fa") as file: # input.fa is expected to be uploaded in␣
 ↪jupyterhub folder
    for line in file:
        print(line.rstrip())
```

11 Mini-Summary

Command line argument + File input

- We can separate *Data* from *Process*
  - **Data**: input file
  - **Process**: script file

$ $ **Reusability up!!** You do not have to update the source code again.

12 Command line argument + File input

```
import sys
file = open(sys.argv[1])
for line in file:
    print(line.rstrip())
file.close()

# command line
# $ python day2_1_example2.py input.fa
```

[ ]:
```
import sys
# Assuming that the script runs as follows:
# $ python day2_1_example2.py input.fa
sys.argv[1] = "input.fa" # This is needed only for jupyter notebook

file = open(sys.argv[1])
for line in file:
    print(line.rstrip())
file.close()
```

We used sequences coded in a Python script yesterday

```
seq1 = "ATGC"   # First sequence
seq2 = "ATAT"   # Second sequence
seq3 = "ATGC"   # Third sequence

sequences = [seq1, seq2, seq3]
```

It is called hard-coding

13 Day2 Part1 Exercise1

Loading a fasta file and count the genome size of *Arabidopsis thaliana*

```
# command example
$ python day2_1_exercise1.py athal_genome.fa
genome size = 119667750 bp
```

- The script is reusable without any changes
- bp: base pair, Kb, Mb, Gb
- There is *athal_genome.fa* in data folder (compressed by gzip)

[ ]:
```
IPython.display.Audio("voice/day2_part1_exercise.mp3")
```

The first exercise today is just loading a FASTA file.

The FASTA file contains a lot of nucleotide sequences, actually more than two.

If you do not know what the FASTA file is. Please look at the explanation below.

14 FASTA format

Nucleotides (*Arabidopsis thaliana*)

```
>AT1G51370.2 | F-box/RNI-like/FBD-like domains-containing protein
ATGGTGGGTGGCAAGAAGAAAACCAAGATATGTGACAAAGTGTCACATGAGGAAGATAGGATAAGCCAGTTTTTGATATCTGAAATACTTTTTCA
```

1. > Annotation information
2. Sequence

15 FASTQ Format

```
@SEQ_ID
GATTTGGGGTTCAAAGCAGTATCGATCAAATAGTAAATCCATTTGTTCAACTCACAGTTT
+
!''*((((***+))%%%++)(%%%%).1***-+*''))**55CCF>>>>>>CCCCCCC65
```

1. ID
2. Sequence
3. Nothing
4. Quatliy (ASCII code')