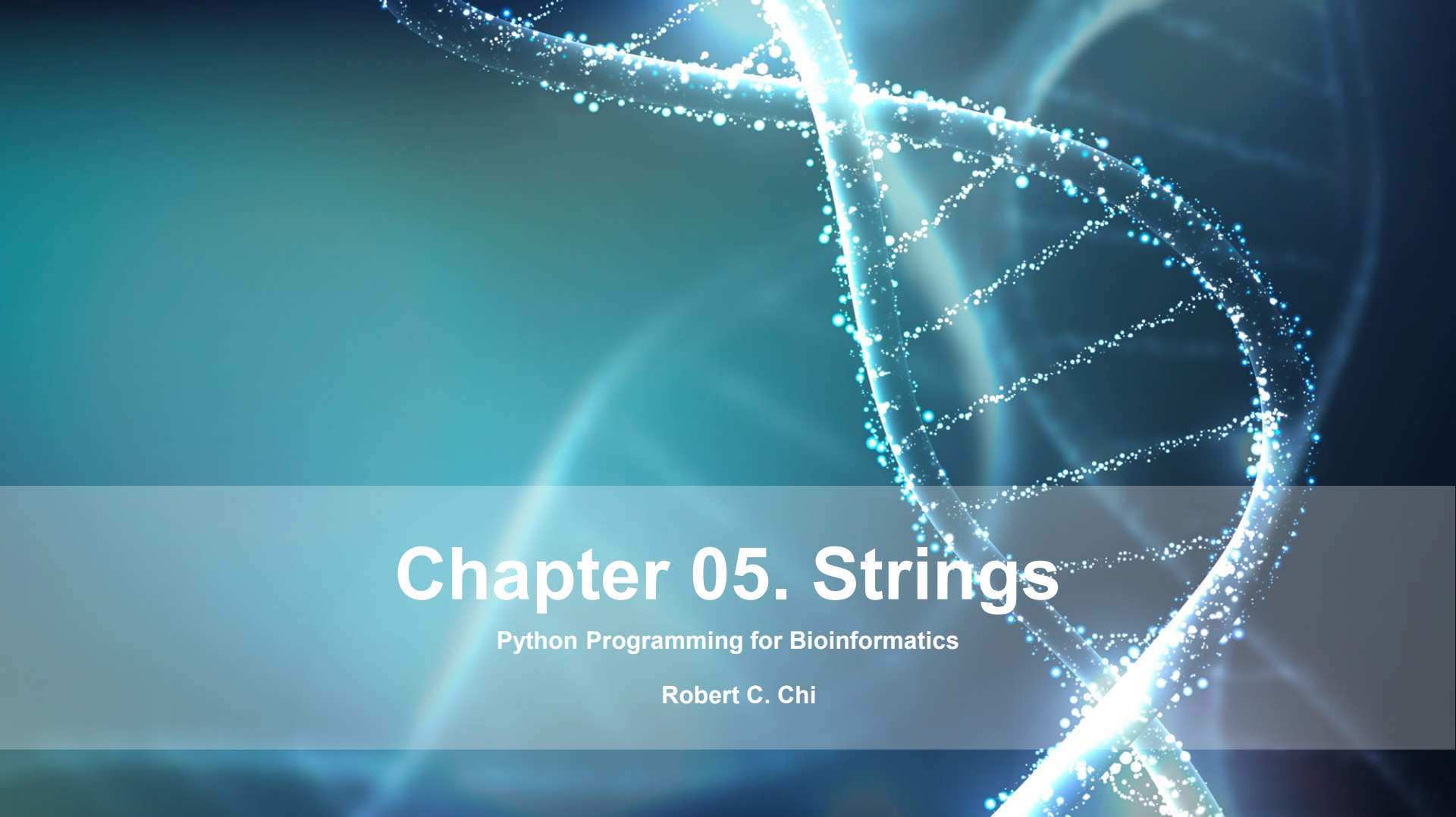


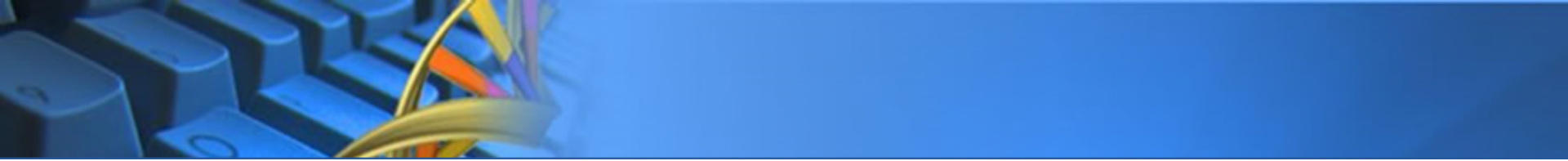
Chapter 05. Strings

Python Programming for Bioinformatics

Robert C. Chi

Agenda

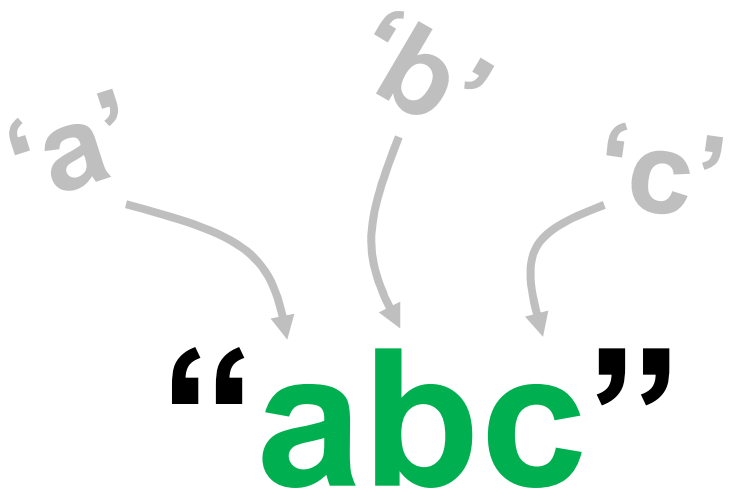
- Basic Operations
- Slicing
- Iteration
- Commonly Used Functions
- Regular Expression
- Example: DNA Comparison



BASIC OPERATIONS

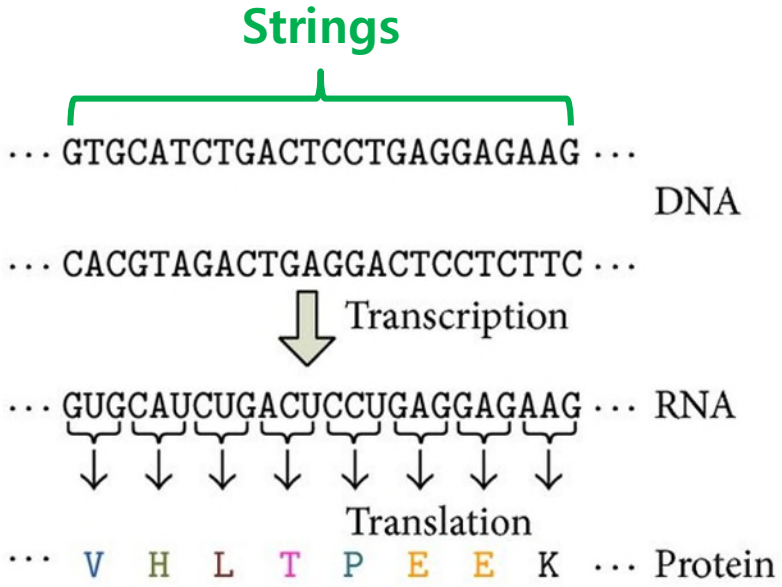
What is a String

- Data composed by a series of characters



Why Strings are Important?

- **Almost all Bioinformatical Sequences are represented by Strings**



Define a String

Empty Strings

s = ‘’

s = “”

Regular Strings

s = ‘abc’

s = “abc”

Multi-line Strings

s = '''ATM Machine
1. Withdraw
2. Deposit
3. Balance
4. Quit'''

Un-packing

- **Decompose a String into Characters**

```
x, y, z = “abc”
```

Specify respectively

```
x, _, z = “abc”
```

Only head and tail

```
x, *b, z = “abcde”
```

Get all the Rest

Concatenation

- “String1” + “String2” → “String1String2”

```
>>> s = "abc" + "xyz"
>>> s
'abcxyz'
```

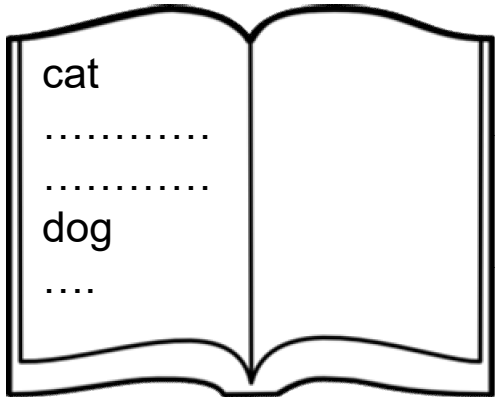
Repeats

- “String1” * 3 → “String1String1String1”

```
>>> s = "abc" * 3
>>> s
'abcabcabc'
```

Comparison

- **“cat” < “dog” → True**
- **“cat” <= “dog” → True**
- **“cat” == “dog” → False**
- **“cat” != “dog” → True**
- **“cat” >= “dog” → False**
- **“cat” > “dog” → False**



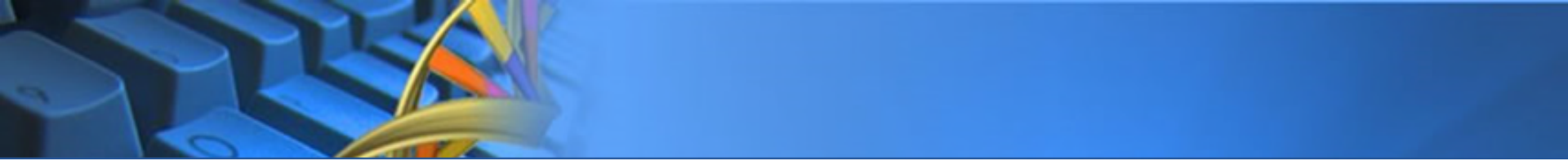
- “cat” < “Cat” → False
- “cat” == “Cat” → False
- “cat” > “Cat” → True

According: Dictionary Order

Inclusion

“abc” in “abcd” → True

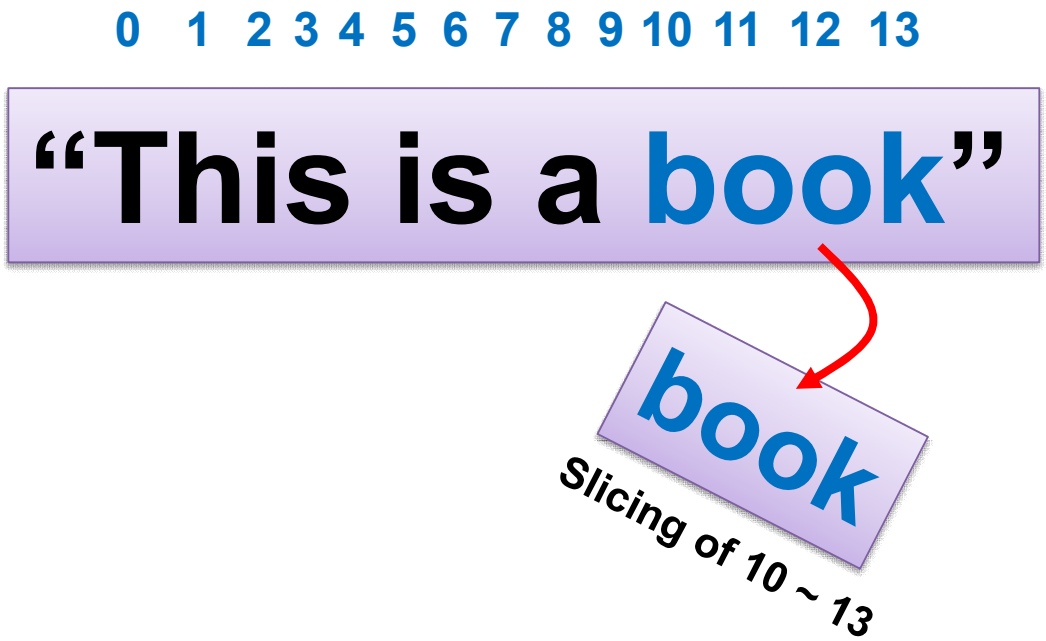
“abc” in “abs” → False



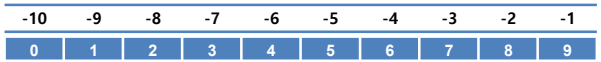
SLICING

What is “Slicing”?

- The Way to Get “Sub-strings”



Syntax of Slicing

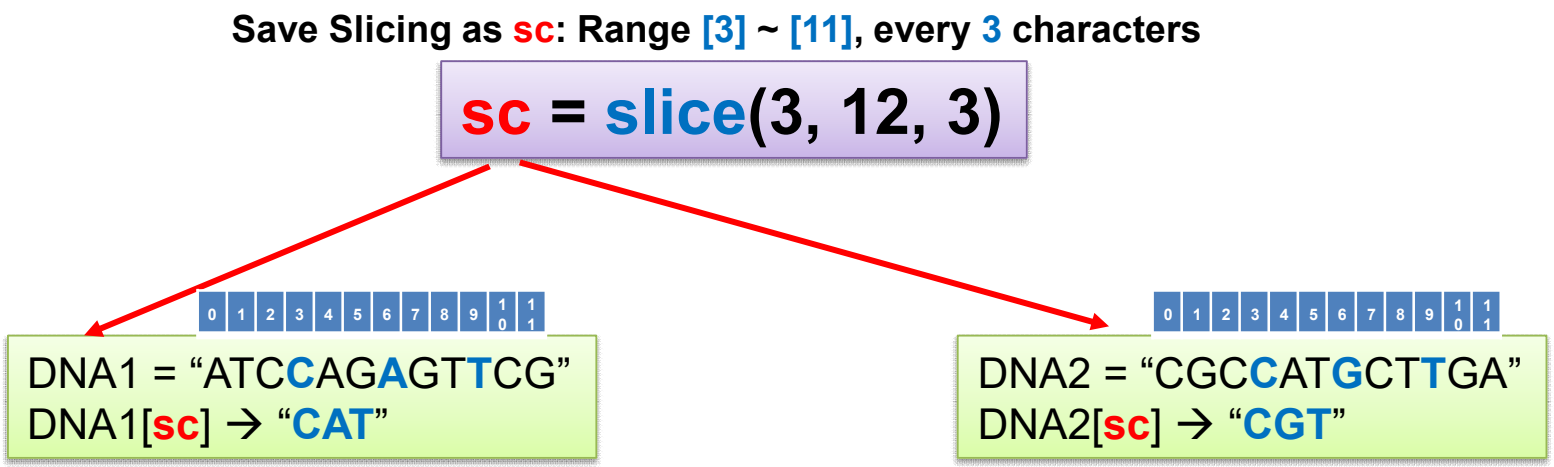


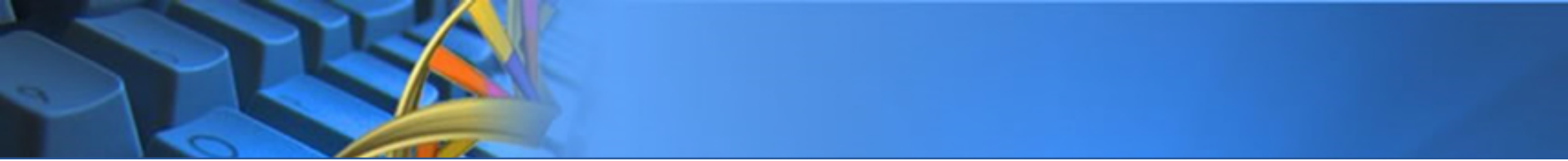
- s =" a b c d e f g h i j "
- s[0] → "a" # Get a Single Character
- s[2:7] → "cdefg" # Get Characters with Index [2], [3], [4], … [6]
- s[-6:-3] → "efg" # Get Characters with Negative Index [-6] ~ [-4]
- s[2:] → "cdefghij" # No Upper-bound = the End of String
- s[:7] → "abcdefg" # No Lower-bound = the Beginning of String
- s[2:7:2] → "ceg" # Get every 2 Character from [2] ~ [6] → [2], [4], [6]

[start : end : step] → Get string from [start] ~ [end-1], every step characters

Reuse the Slicing by Templates

- **Save the Slicing as a Template then Reuse:**

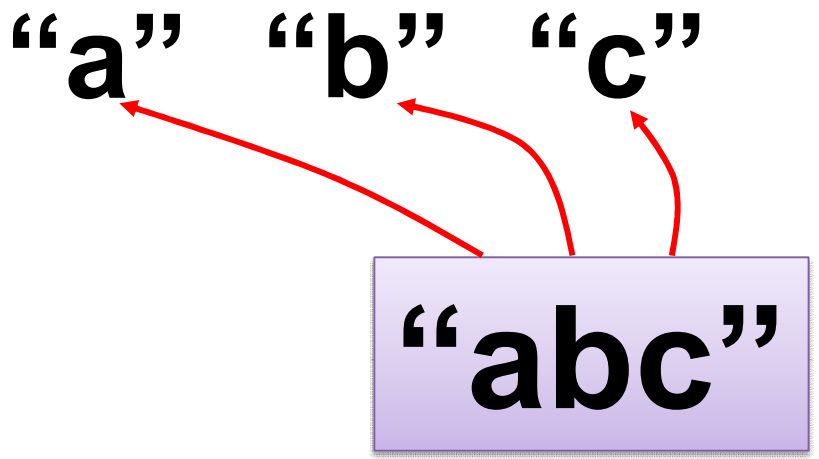




ITERATION

What is "Iteration"?

- **Extract** one **character** a time from a **string**.



How to Iterate a String?

- for

```
>>> for c in "abcdefg":
        print(c, end=" ")


a b c d e f g
```

- iter()

```
>>> iterator = iter("abcdefg")
>>> for i in iterator:
        print(i)


a
b
c
d
e
f
g
```

- enumerate()

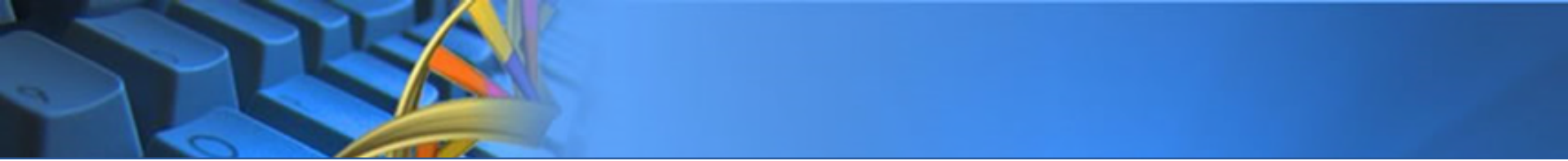
```
>>> iterator = enumerate("abcdefg")
>>> for i, j in iterator:
        print(i, j)


0 a
1 b
2 c
3 d
4 e
5 f
6 g
```

Easy to Use

Customize the Order

Return both Index and Character



COMMONLY USED FUNCTIONS

Length, Maximum, Minimum

- **Length**
  - **len(“abcdefg”) → 7**
- **Maximum & Minimum**
  - **min(“abcdefg”) → “a”**
  - **max(“abcdefg”) → “g”**

Case Conversion

- **All lowercase**
  - “**T**his is a book”.**lower**() → “**t**his is a book”
- **All uppercase**
  - “**T**his is a book”.**upper**() → “**THIS IS A BOOK**”

Content Detection

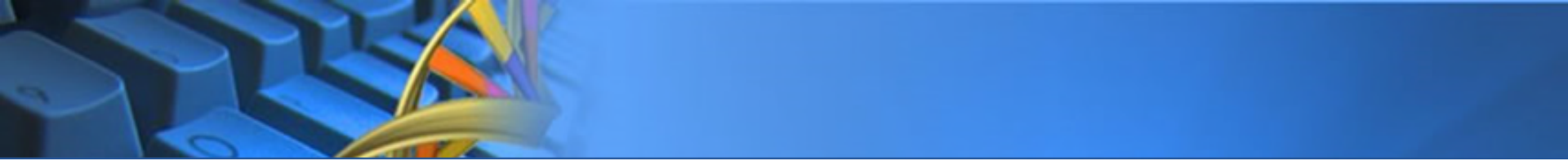
- **All lowercase or not**
  - “book”.**islower**() → True
- **All uppercase or not**
  - “BOOK”.**isupper**() → True
- **All blank or not**
  - “ “.**isspace**() → True
- **Contain only alphabets or not**
  - “book”.**isalpha**() → True
- **Contain only digits or not**
  - “123”.**isdigit**() → True
- **Contain only alphabets and digits or not**
  - “13books”.**isalnum**() → True

Search and Counting

- **Find the index of the first occurrence forwardly:**
  - "Th**is** is a book".**find**("**is**") → 2
  - **Not Found**: Return -1
- **Find the index of the first occurrence backwardly:**
  - "This **is** a book".**rfind**("**is**") → 5
  - **Not Found**: Return -1
- **If starts with a specific text:**
  - "This is a book" .**startswith**("This") → True
- **If ends with a specific text:**
  - "This is a book" .**endswith**("book") → True
- **Calculate the number of occurrence:**
  - "This is a book".**count**("is") → 2

Replace, Strip, Split, Join

- **Replace**
  - “This is a book”.**replace**(“book”, “cat”) → “This is a cat”
- **Strip**
  - “ This is a book “.**strip**() → “This is a book”
  - Remove the spaces on left- and right-hand sides of a string.
- **Split**
  - “This is a book”.**split**(“ “) → ['This', 'is', 'a', 'book’]
- **Join**
  - s = “This is a book”.split(“ “)
  - “-”.**join**(s) → “This-is-a-book”
  - “ “.**join**(s) → “This is a book”



EXAMPLE: DNA COMPARISON

Description

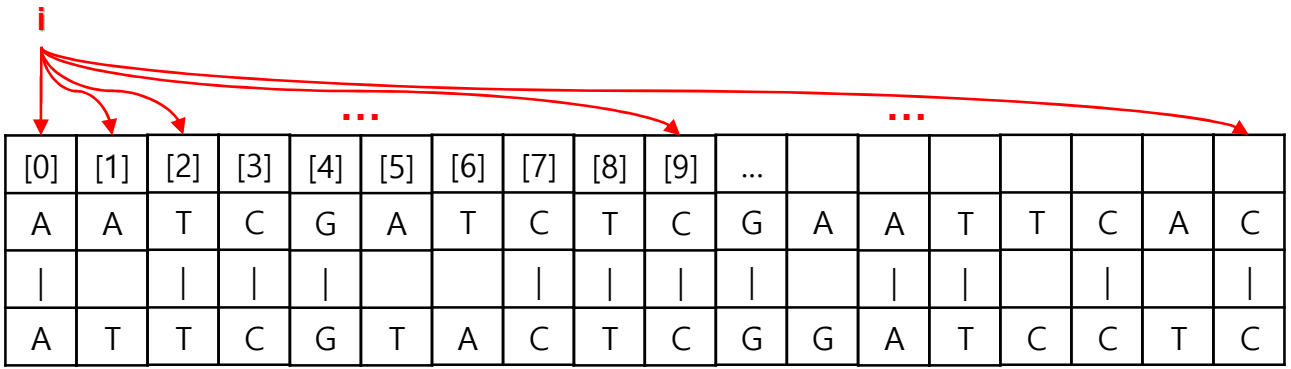
- **Identify the same "bases" from two DNA sequences**
  - DNA will consist of only four symbols: A, T, G, C
    - Adenine (A), Thymine (T), Guanine (G), Cytosine (C)
  - Suppose you got two sequences of DNA:
    - DNA1 : AATCGATCTCGAATTCAC
    - DNA2 : ATTCGTACTCGGATCCTC
  - Please write a program that connects the same symbols as below:

```
A A T C G A T C T C G A A T T C A C
|   | | |     | | | |   | |   | |
A T T C G T A C T C G G A T C C T C
```

Analysis

DNA1 = "AATCGATCTCGAATTCAC "
similar = " "
DNA2 = "ATTCGTACTCGGATCCTC"

$$similar[i] = \begin{cases} \text{“ | ”} & if\ DNA1[i] == DNA2[i] \\ (space) & if\ DNA1[i]\ != DNA2[i] \end{cases} \quad \text{i} = 0, 1, 2, \ldots$$



Source Code

```
DNA1 = "AATCGATCTCGAATTCAC"
similar = ""
DNA2 = "ATTCGTACTCGGATCCTC"

for i in range(0, len(DNA1)):
    if DNA1[i] == DNA2[i]:
        similar += "|"
    else:
        similar += " "

print(DNA1)
print(similar)
print(DNA2)
```

```
AATCGATCTCGAATTCAC
|  |||  ||||  ||  |  |
ATTCGTACTCGGATCCTC
```

Improvement (1)

- **Calculate Similarity**
  - **Requirement**:
    - Please calculate the similarity of the two DNA sequences.
  - **Hint**:
    - similarity = similar.count("|") / len(similar) x 100%

Source Code

```
DNA1 = "AATCGATCTCGAATTCAC"
similar = ""
DNA2 = "ATTCGTACTCGGATCCTC"

for i in range(0, len(DNA1)):
    if DNA1[i] == DNA2[i]:
        similar += "|"
    else:
        similar += " "

print(DNA1)
print(similar)
print(DNA2)

similarity = similar.count("|") / len(similar)
print("Similarity: {:.2%}".format(similarity))
```

```
AATCGATCTCGAATTCAC
|  |||   ||||  ||  |  |
ATTCGTACTCGGATCCTC
Similarity: 66.67%
```

Improvement (2)

- **Allows users to enter their own DNA sequences**
  - DNA1 = input("Enter DNA1: ")
  - DNA2 = input("Enter DNA2: ")
- **The input DNA symbols are case insensitive**
  - DNA1 = input("Enter DNA1: ").upper()
  - DNA2 = input("Enter DNA2: ").upper()
- **DNA sequences can be different length**

```
1  DNALength = min(len(DNA1), len(DNA2))
2  for i in range(0, DNALength):
3      ...
```

Improvement (2)

- **Check the symbols entered by users are only in A, T, C, G**

```
DNA1 = input("Enter DNA1: ").upper()
similar = ""
DNA2 = input("Enter DNA2: ").upper()

DNALength = min(len(DNA1), len(DNA2))

isLegal = True
for i in range(0, DNALength):
    if (DNA1[i] in "ATCG") and (DNA2[i] in "ATCG"):
        if DNA1[i] == DNA2[i]:
            similar += "|"
        else:
            similar += " "
    else:
        isLegal = False
        break

if isLegal:
    print(DNA1)
    print(similar)
    print(DNA2)

    similarity = similar.count("|") / len(similar)
    print("Similarity: {:.2%}".format(similarity))
else:
    print("Illegal Symbols of DNA Sequences!")
```

```
Enter DNA1: ATCGGCGATGACA
Enter DNA2: acggCCGATCCAAGCATG
ATCGGCGATGACA
|  | ||||   |
ACGGCCGATCCAAGCATG
Similarity: 53.85%
```

```
Enter DNA1: ppp
Enter DNA2: atcg
Illegal Symbols of DNA Sequences!
```