An introduction to python and its applications in bioinformatics

Nikhil George
Lab Master Class
Jan 2021

The print function and ‘Hello World’

```
(base) nageorge@hydra:~$ python
```

The print function and ‘Hello World’

```
(base) nageorge@hydra:~$ python
Python 3.7.7 (default, Mar 26 2020, 15:48:22)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
```

The print function and ‘Hello World’

```
(base) nageorge@hydra:~$ python
Python 3.7.7 (default, Mar 26 2020, 15:48:22)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.

>>> print('Hello World')
```

The print function and ‘Hello World’

```
(base) nageorge@hydra:~$ python
Python 3.7.7 (default, Mar 26 2020, 15:48:22)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.

>>> print('Hello World')
Hello World
```

Variable assignment

```
>>> lab_group = "Hug_lab"
```

Variable assignment

```
>>> lab_group = "Hug_lab"
>>> print(lab_group)
Hug_lab
```

Variable assignment

```
>>> lab_group = "Hug_lab"
>>> print(lab_group)
Hug_lab
>>> lab_group
'Hug_lab'
```

List generation and the len function

```
>>> lab_members = ['Laura', 'Fiona', 'Veronica', 'Dan', 'Panuya', 'Brian', 'Emilie', 'Grant', 'Rosalind', 'Nikhil', 'Lisa']
```

List generation and the len function

```
>>> lab_members = ['Laura', 'Fiona', 'Veronica', 'Dan', 'Panuya', 'Brian', 'Emilie', 'Grant', 'Rosalind', 'Nikhil', 'Lisa']
>>> lab_members
['Laura', 'Fiona', 'Veronica', 'Dan', 'Panuya', 'Brian', 'Emilie', 'Grant', 'Rosalind', 'Nikhil', 'Lisa']
```

List generation and the len function

```
>>> lab_members = ['Laura', 'Fiona', 'Veronica', 'Dan', 'Panuya', 'Brian', 'Emilie', 'Grant', 'Rosalind', 'Nikhil', 'Lisa']
>>> lab_members
['Laura', 'Fiona', 'Veronica', 'Dan', 'Panuya', 'Brian', 'Emilie', 'Grant', 'Rosalind', 'Nikhil', 'Lisa']
>>> len(lab_members)
11
```

List generation and the len function

```
>>> lab_members = ['Laura', 'Fiona', 'Veronica', 'Dan', 'Panuya', 'Brian', 'Emilie', 'Grant', 'Rosalind', 'Nikhil', 'Lisa']
>>> lab_members
['Laura', 'Fiona', 'Veronica', 'Dan', 'Panuya', 'Brian', 'Emilie', 'Grant', 'Rosalind', 'Nikhil', 'Lisa']
>>> len(lab_members)
11
>>> len('str')
3
```

Indexing in python

```
>>> lab_members
['Laura', 'Fiona', 'Veronica', 'Dan', 'Panuya', 'Brian', 'Emilie', 'Grant', 'Rosalind', 'Nikhil', 'Lisa']
```

Indexing in python

```
>>> lab_members
['Laura', 'Fiona', 'Veronica', 'Dan', 'Panuya', 'Brian', 'Emilie', 'Grant', 'Rosalind', 'Nikhil', 'Lisa']
>>> print("The principal investigator of this lab is: " + lab_members[1])
```

Indexing in python

```
>>> lab_members
['Laura', 'Fiona', 'Veronica', 'Dan', 'Panuya', 'Brian', 'Emilie', 'Grant', 'Rosalind', 'Nikhil', 'Lisa']
>>> print("The principal investigator of this lab is: " + lab_members[1])
The principal investigator of this lab is: Fiona
```

??????

Indexing in python

```
>>> lab_members
['Laura', 'Fiona', 'Veronica', 'Dan', 'Panuya', 'Brian', 'Emilie', 'Grant', 'Rosalind', 'Nikhil', 'Lisa']
```

0 1 2 3 4 5 6 7 8 9 10

Indexing in python

```
>>> print("The principal investigator of this lab is: " + lab_members[0])
```

Indexing in python

```
>>> print("The principal investigator of this lab is: " + lab_members[0])
The principal investigator of this lab is: Laura
```

Index ranges and non-inclusivity

```
>>> lab_members
['Laura', 'Fiona', 'Veronica', 'Dan', 'Panuya', 'Brian', 'Emilie', 'Grant', 'Rosalind', 'Nikhil', 'Lisa']
```

Index ranges and non-inclusivity

```
>>> lab_members
['Laura', 'Fiona', 'Veronica', 'Dan', 'Panuya', 'Brian', 'Emilie', 'Grant', 'Rosalind', 'Nikhil', 'Lisa']
>>> lab_members[0:4]
```

Index ranges and non-inclusivity

```
>>> lab_members
['Laura', 'Fiona', 'Veronica', 'Dan', 'Panuya', 'Brian', 'Emilie', 'Grant', 'Rosalind', 'Nikhil', 'Lisa']
>>> lab_members[0:4]
['Laura', 'Fiona', 'Veronica', 'Dan']
```

The for loop

```
>>> for lab_member in lab_members:
```

The for loop

```
>>> for lab_member in lab_members:
...     print(lab_member + " is a cool lab member")
```

The for loop

```
>>> for lab_member in lab_members:
...     print(lab_member + " is a cool lab member")
...
Laura is a cool lab member
Fiona is a cool lab member
Veronica is a cool lab member
Dan is a cool lab member
Panuya is a cool lab member
Brian is a cool lab member
Emilie is a cool lab member
Grant is a cool lab member
Rosalind is a cool lab member
Nikhil is a cool lab member
Lisa is a cool lab member
```

The for loop’s iteration variable

```
>>> for chicken13 in lab_members:
...     print(chicken13 + " is a cool lab member")
```

The for loop's iteration variable

```
>>> for chicken13 in lab_members:
...     print(chicken13 + " is a cool lab member")
...
Laura is a cool lab member
Fiona is a cool lab member
Veronica is a cool lab member
Dan is a cool lab member
Panuya is a cool lab member
Brian is a cool lab member
Emilie is a cool lab member
Grant is a cool lab member
Rosalind is a cool lab member
Nikhil is a cool lab member
Lisa is a cool lab member
```

The if else conditional statements and the “>” comparison operator

```
>>> if len(lab_members) > 10:
```

The if else conditional statements and the “>” comparison operator

```
>>> if len(lab_members) > 10:
...     print("Since when did our lab get so big?")
```

The if else conditional statements and the “>” comparison operator

```
>>> if len(lab_members) > 10:
...     print("Since when did our lab get so big?")
... else:
...     print("Is our lab considered big?")
```

The if else conditional statements and the “>” comparison operator

```
>>> if len(lab_members) > 10:
...     print("Since when did our lab get so big?")
... else:
...     print("Is our lab considered big?")
...
Since when did our lab get so big?
```

Counters, another comparison operator “==” and... basically everything else we’ve discussed

```
>>> name = 'LAURA'
>>> letters = ['L','A','U','R','A']
>>> counter = 0
```

Counters, another comparison operator “==” and... basically everything else we’ve discussed

```
>>> name = 'LAURA'
>>> letters = ['L','A','U','R','A']
>>> counter = 0
>>> for character in name:
...     if character == letters[1]:
...             counter = counter + 1
...
```

Counters, another comparison operator “==” and... basically everything else we’ve discussed

```
>>> name = 'LAURA'
>>> letters = ['L','A','U','R','A']
>>> counter = 0
>>> for character in name:
...     if character == letters[1]:
...             counter = counter + 1
...
>>> print(counter)
2
```

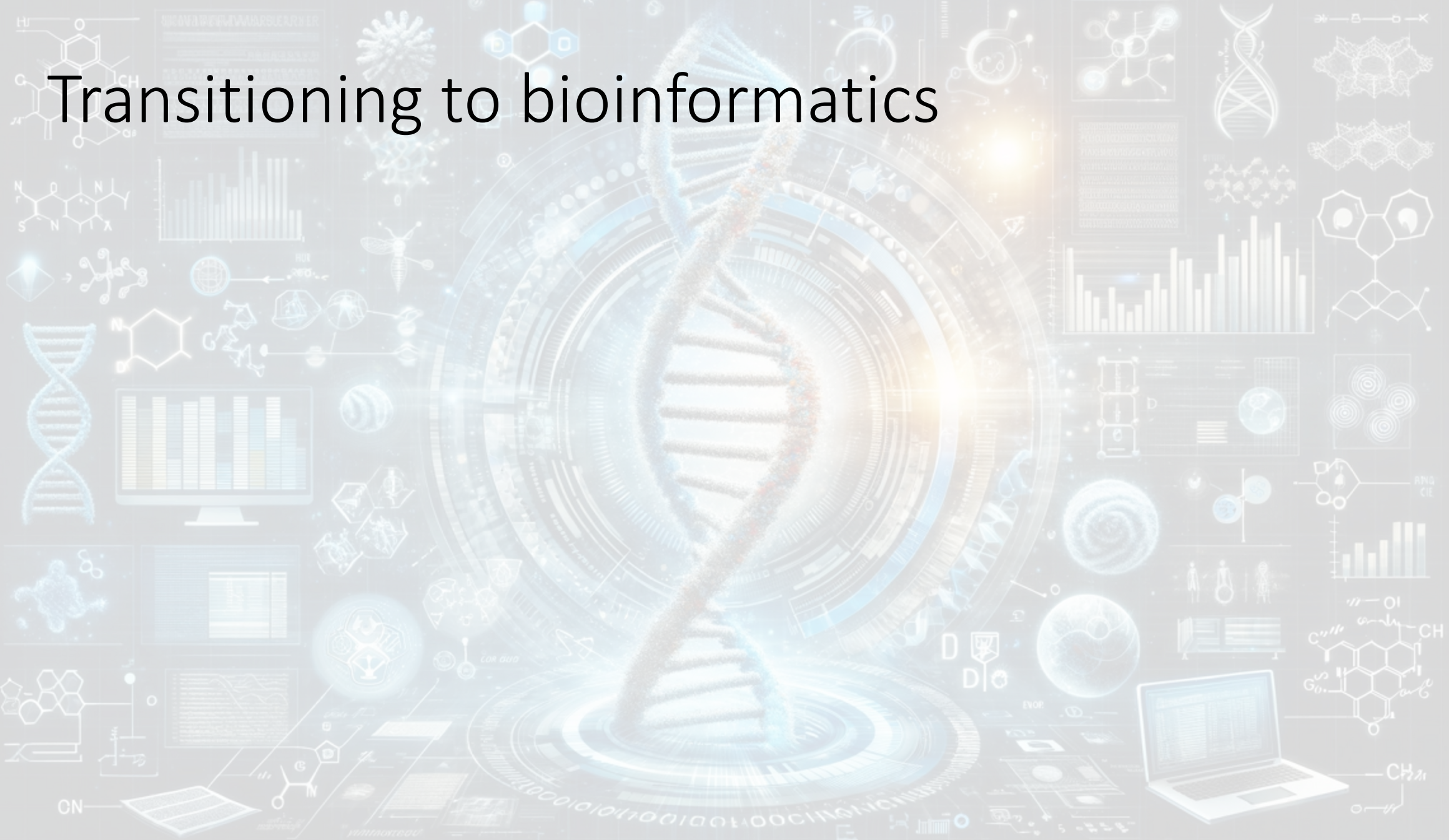
The range function with a for loop

```
>>> for i in range(17):
...     print(i)
...
```

The range function with a for loop

```
>>> for i in range(17):
...     print(i)
...
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
```

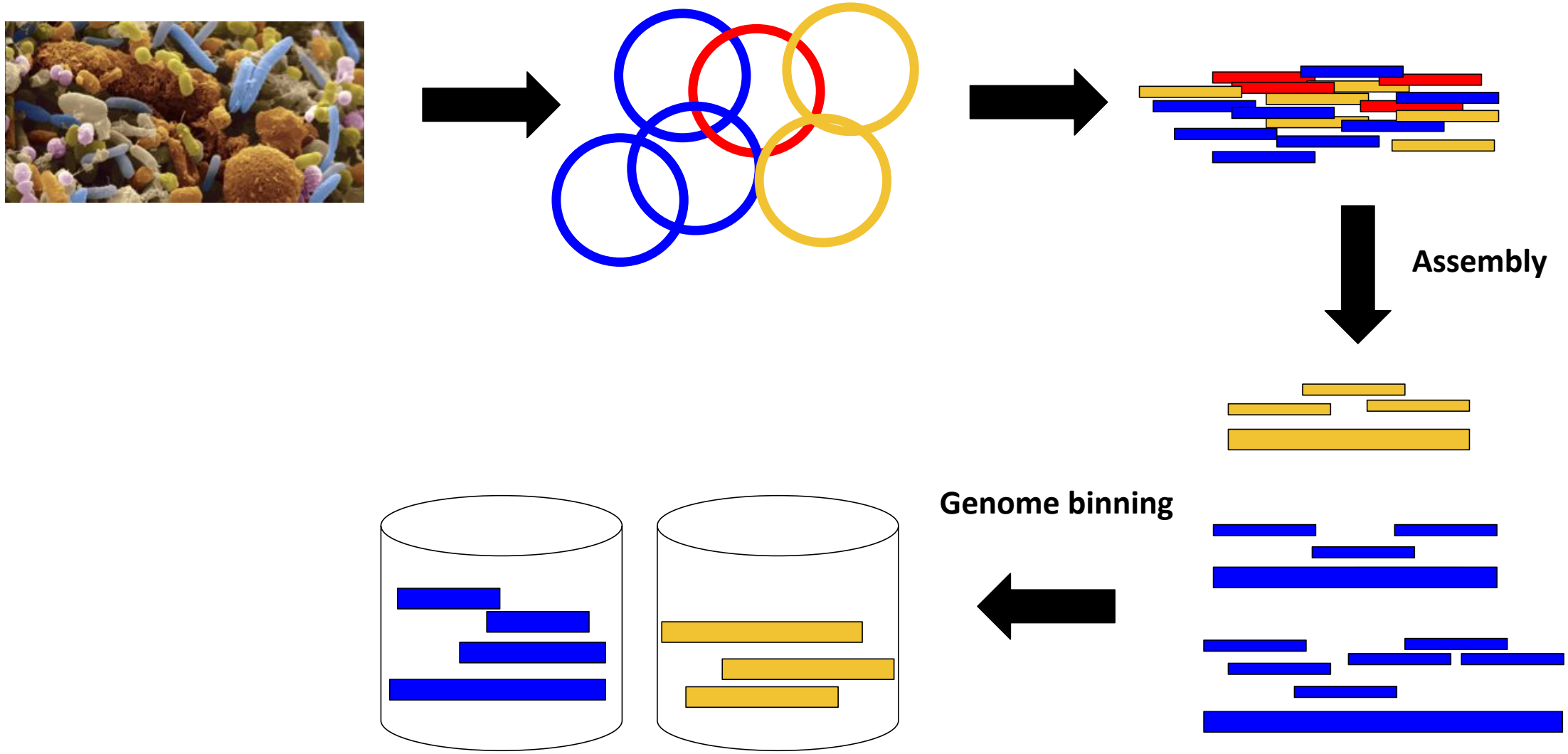
Transitioning to bioinformatics



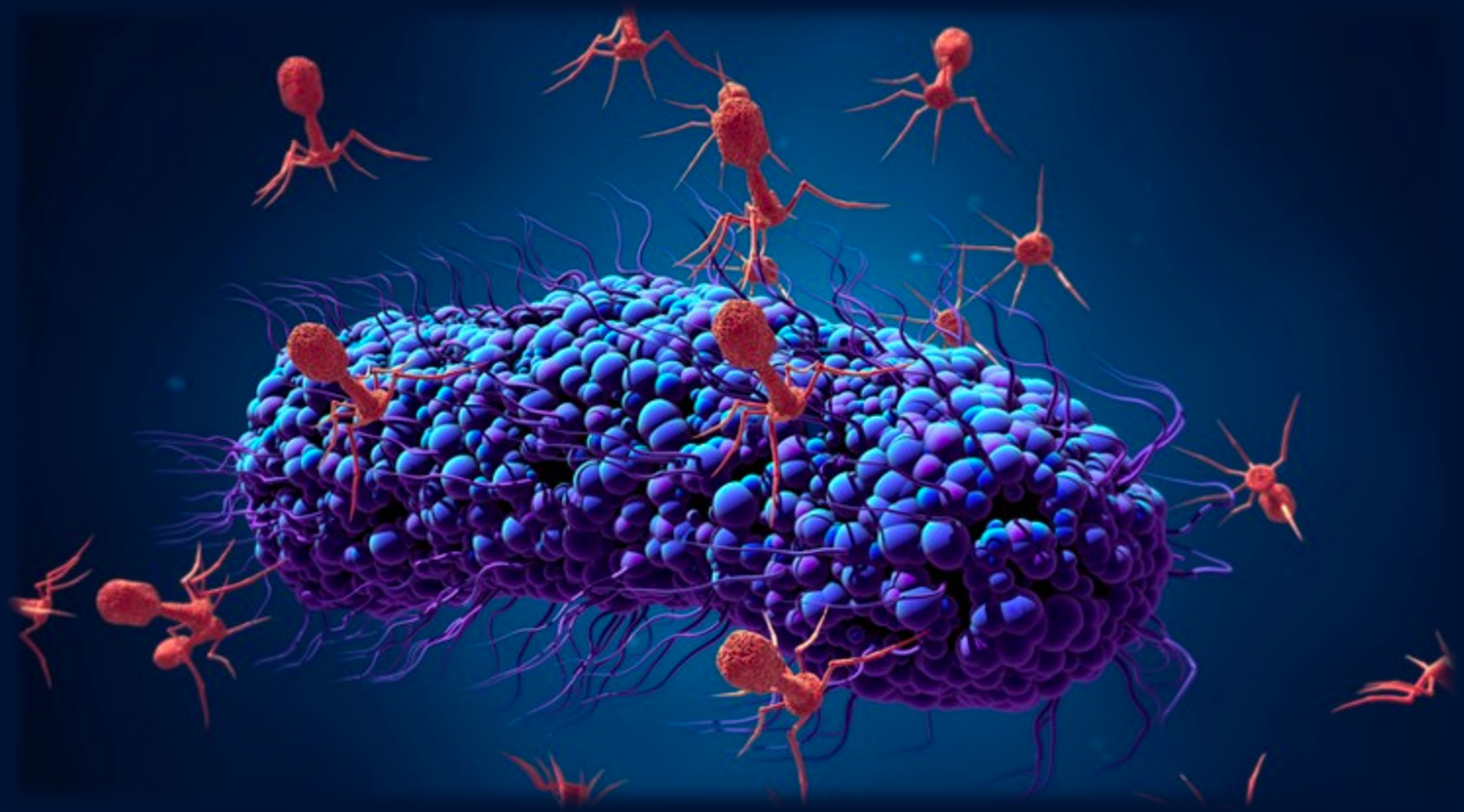
K-mers

- A k-mer is pattern (or part of a sequence) of length k
- E.g. ATA is a 3-mer

Work in our lab involving k-mers



Work in our lab involving k-mers



K-mer detection

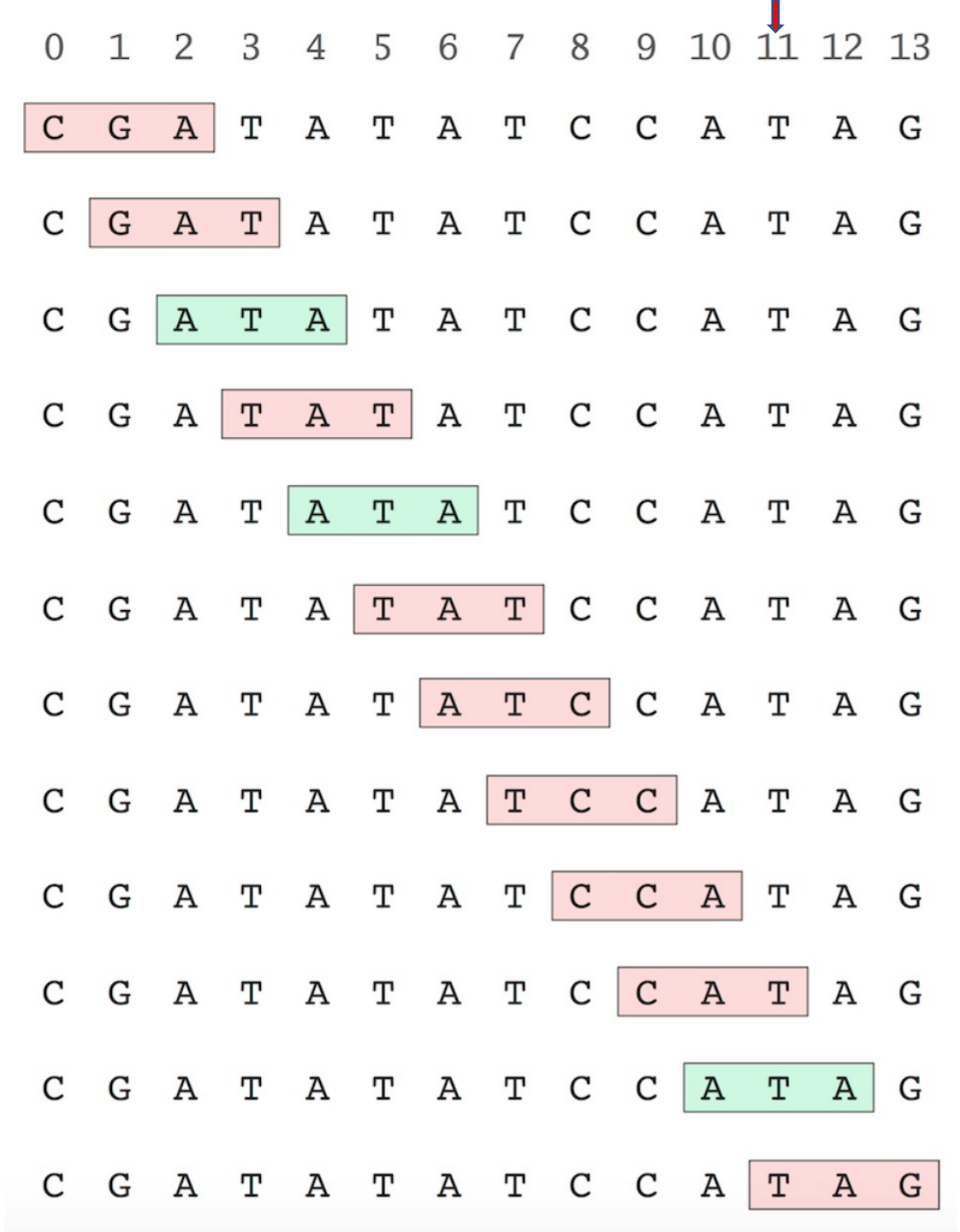
How many times does the 3-mer “ATA” exist in the string below?

CGATATATCCATAG

K-mer detection

How many times does the 3-mer “ATA” exist in the string below?

CGATATATCCATAGTGCTAATTTAATAGCTATGCCCTAAGTATAAGGGCTACGA

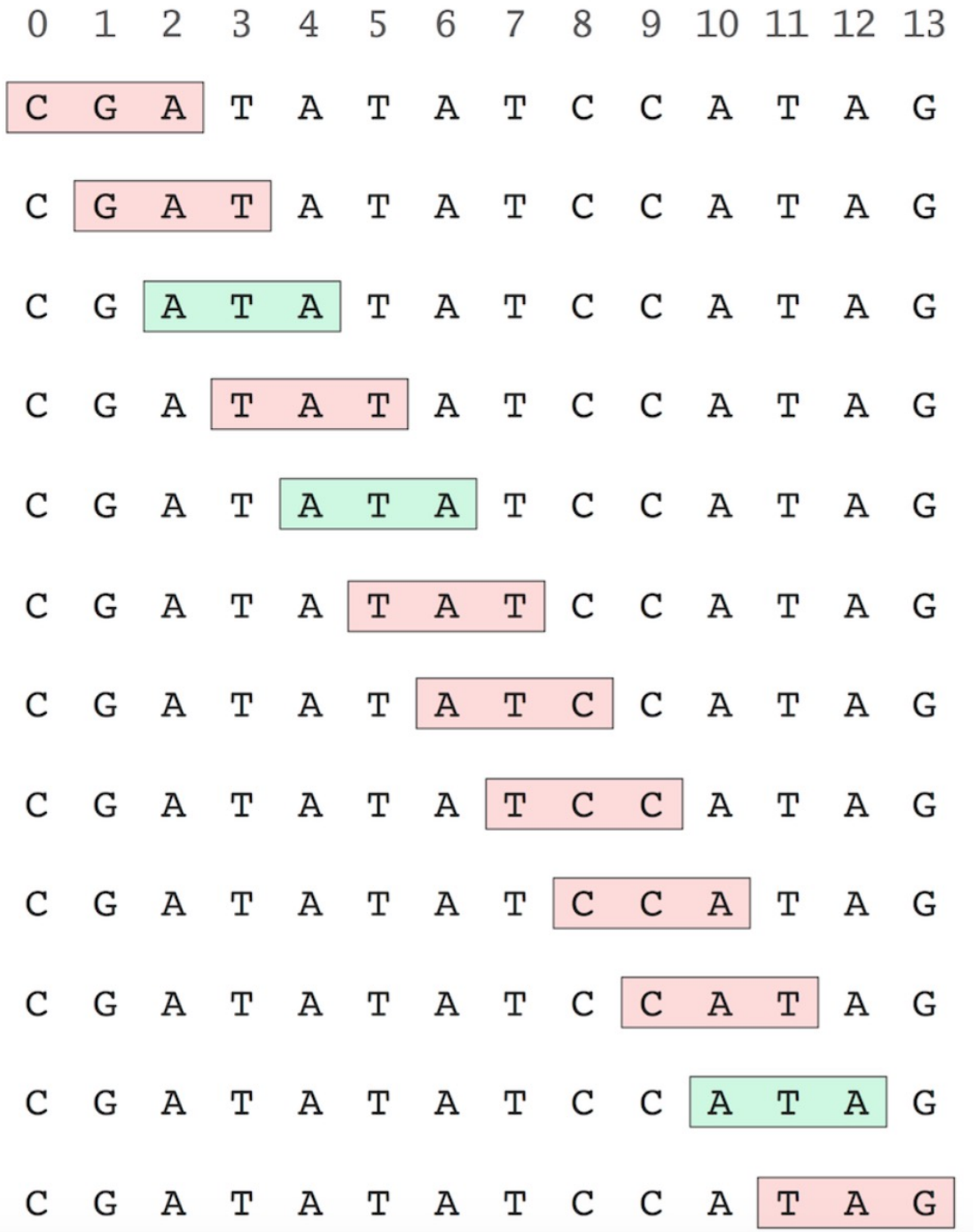


Sequence: CGATATATCCATAG
(14 characters)

3-mer: ATA

Note that the 11th position in the sequence is last position at which a 3-mer can be captured

I.e., the last position a k-mer can be called in a sequence = the length of the sequence – the length of the k-mer



Sequence: CGATATATCCATAG

3-mer: ATA

- We need a way of iterating across the sequence
- We need to compare pieces of the sequence to the k-mer
- We need a counter to keep track of matches to the k-mer

K-mer detection

```
#!/opt/anaconda3/bin/python

kmer = 'ATA'
sequence = 'CGATATATCCATAG'
```

K-mer detection

```
#!/opt/anaconda3/bin/python

kmer = 'ATA'
sequence = 'CGATATATCCATAG'

counter = 0
```

K-mer detection

```
#!/opt/anaconda3/bin/python

kmer = 'ATA'
sequence = 'CGATATATCCATAG'

counter = 0
for i in                                :
```

K-mer detection

```
#!/opt/anaconda3/bin/python

kmer = 'ATA'
sequence = 'CGATATATCCATAG'

counter = 0
for i in [redacted]:
    if [redacted] == kmer:
        counter = counter + 1
```

K-mer detection

```
#!/opt/anaconda3/bin/python

kmer = 'ATA'
sequence = 'CGATATATCCATAG'

counter = 0
for i in range((len(sequence))-(len(kmer))+1):
```

The range function

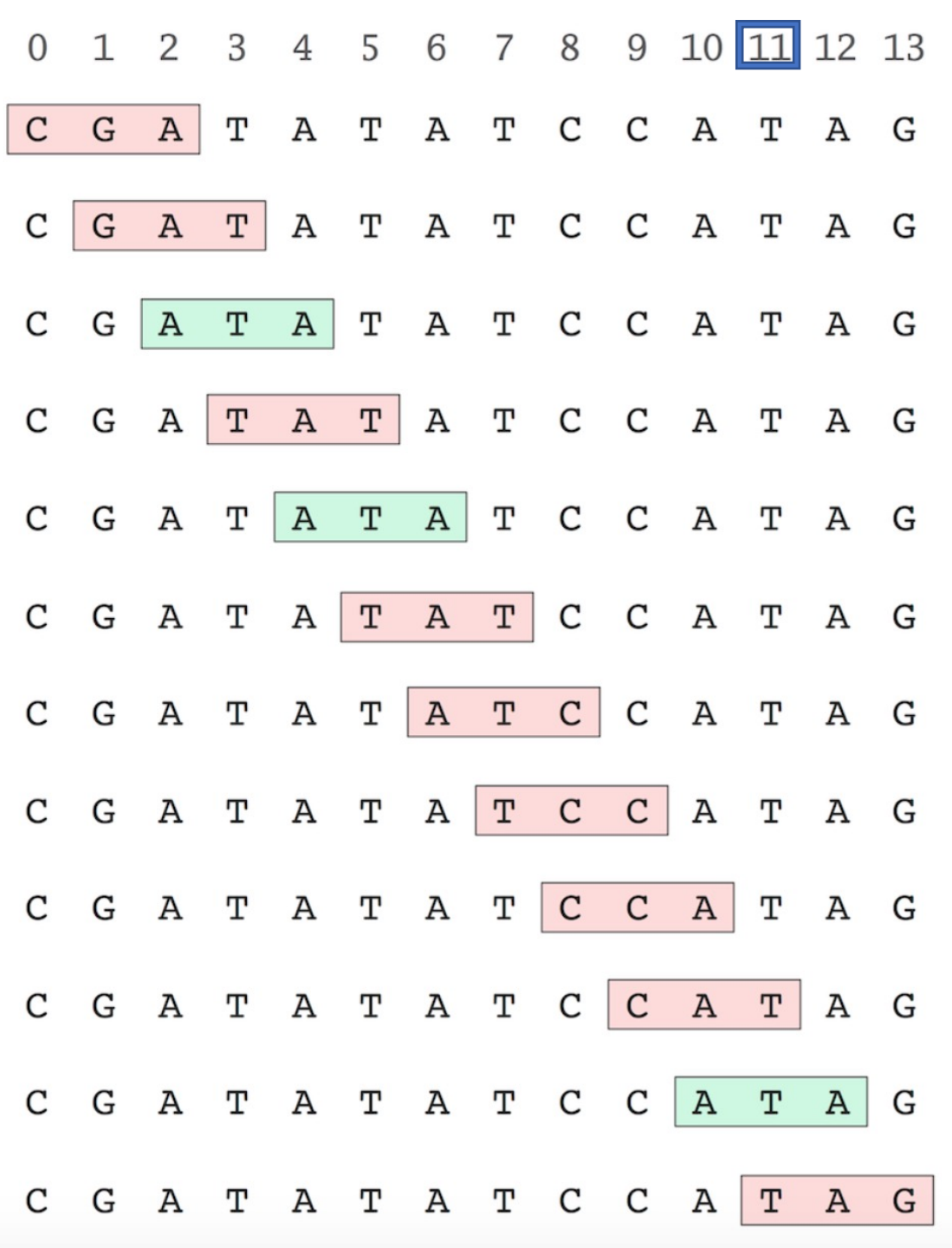
```
>>> for i in range(17):
...     print(i)
...
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
```

K-mer detection

```
#!/opt/anaconda3/bin/python

kmer = 'ATA'
sequence = 'CGATATATCCATAG'

counter = 0
for i in range((len(sequence))-(len(kmer))+1):
```

K-mer detection

```
#!/opt/anaconda3/bin/python

kmer = 'ATA'
sequence = 'CGATATATCCATAG'

counter = 0
for i in range((len(sequence))-(len(kmer))+1):
```



K-mer detection

```
#!/opt/anaconda3/bin/python

kmer = 'ATA'
sequence = 'CGATATATCCATAG'

counter = 0
for i in range((len(sequence))-(len(kmer))+1):
    if sequence[i:i+(len(kmer))] == kmer:
        counter = counter + 1
```

K-mer detection

```
#!/opt/anaconda3/bin/python

kmer = 'ATA'
sequence =
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13
C	G	A	T	A	T	A	T	C	C	A	T	A	G

```
counter = 0
for i in range((len(sequence))-(len(kmer))+1):
    if sequence[i:i+(len(kmer))] == kmer:
        counter = counter + 1
```

if sequence[0:3] == kmer:

K-mer detection

```
#!/opt/anaconda3/bin/python

kmer = 'ATA'
sequence = 'CGATATATCCATAG'

counter = 0
for i in range((len(sequence))-(len(kmer))+1):
    if sequence[i:i+(len(kmer))] == kmer:
        counter = counter + 1
print(counter)
```

K-mer detection (as a python script)

```
➜ PhD_student_thoroughly_confuses_lab_with_poor_explanation_of_python ./kmer_counting2.py
3
```

K-mer detection (with friendly user output)

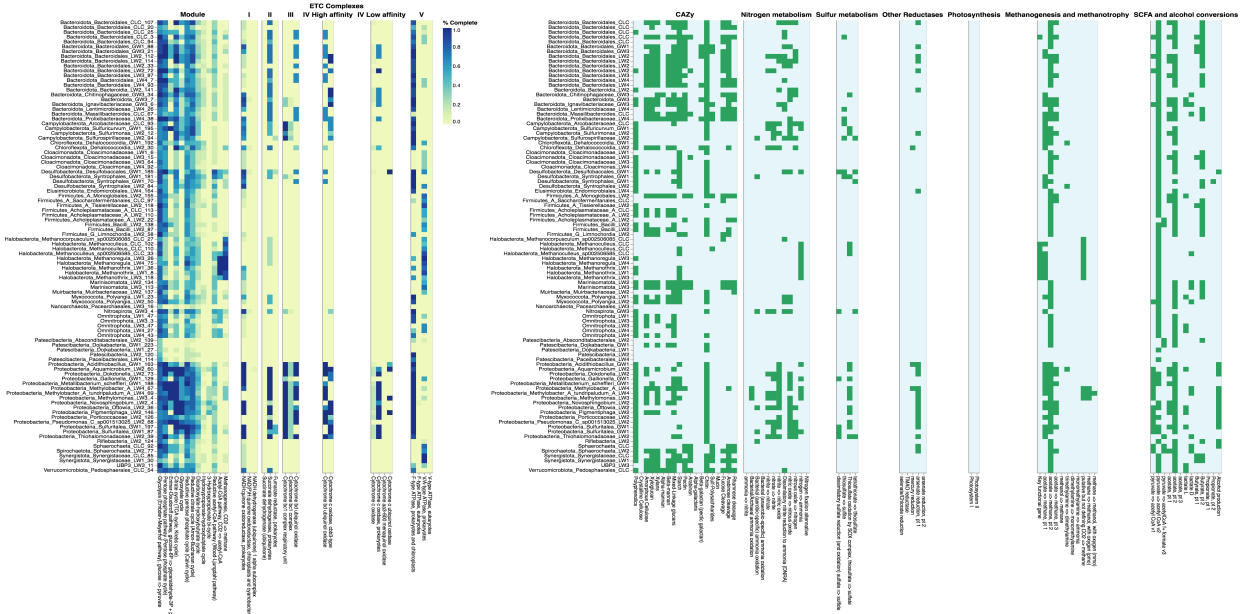
```
➜ PhD_student_thoroughly_confuses_lab_with_poor_explanation_of_python ./kmer_counting2.py
The kmer ATA occurs 3 times in the sequence CGATATATCCATAG
```

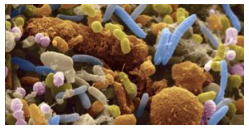
K-mer detection (with arguments passed from the command line)

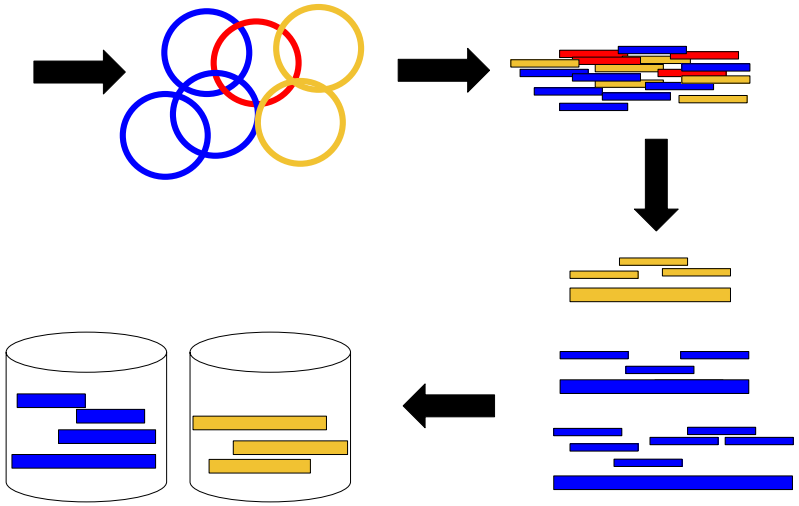
```
➜ PhD_student_thoroughly_confuses_lab_with_poor_explanation_of_python ./kmer_counting.py ATA CGATATATCCATAG
The kmer ATA occurs 3 times in the sequence CGATATATCCATAG
```

K-mer detection (with arguments passed from the command line)

```
➜ PhD_student_thoroughly_confuses_lab_with_poor_explanation_of_python ./kmer_counting.py ATA CGATATATCCATAGTGCTAATTTAATAGCTATGCCCTAAGTATAAGGGCTACGA

The kmer ATA occurs 5 times in the sequence CGATATATCCATAGTGCTAATTTAATAGCTATGCCCTAAGTATAAGGGCTACGA
```







max	34.000000	27386.000000	3906.000000	31292.000000	42588.000000	73880.000000

```
In [11]: data.columns

Out[11]: Index(['Unnamed: 0', 'date', 'state', 'hospitalized_with_symptoms',
       'intensive_care', 'total_hospitalized', 'home_quarantine',
       'total_confirmed_cases', 'new_confirmed_cases', 'recovered', 'deaths',
       'total_cases', 'swabs_made', 'note_it', 'note_en'],
      dtype='object')
```

relating the variables with scatterplots

```
In [15]: sns.relplot(x="total_confirmed_cases", y="hospitalize", data=data)

Out[15]: <seaborn.axisgrid.FacetGrid at 0x180efd13e80>
```

Python » English 3.9.1 Documentation »

Download

Download these documents

Docs by version

Python 3.9.1 documentation

Welcome! This is the documentation for Python 3.9.1.

codecademy

codewars by Qualified

For Companies For

Achieve mastery through challenge

Improve your skills by training with others on real code challenges

SIGN UP

To join you must first prove your skills.
Choose your language to begin...

Clojure	CoffeeScript	C	Coq
C#	Crystal	Dart	Elixir
Go	Groovy	Haskell	Java
Kotlin	Lean	Lua	NASM
Python	Racket	Ruby	Rust
Shell	SQL	Swift	TypeScript

ROSALIND About Problems Statistics Glossary search Log in Register

Locations

Rosalind is a platform for learning bioinformatics and programming through problem solving. Take a tour to get the hang of how Rosalind works.

If you don't know anything about programming, you can start at the Python Village. For a collection of exercises to accompany Bioinformatics Algorithms book, go to the Textbook Track. Otherwise you can try to storm the Bioinformatics Stronghold right now.

Python Village

If you are completely new to programming, try these initial problems to learn a few basics about the Python programming language. You'll get familiar with the operations needed to start solving bioinformatics challenges in the Stronghold.

Bioinformatics Stronghold

Discover the algorithms underlying a variety of bioinformatics topics: computational mass spectrometry, alignment, dynamic programming, genome assembly, genome rearrangements, phylogeny, probability, string algorithms and others.

Bioinformatics Armory

Ready-to-use software tools abound for bioinformatics analysis. Whereas in the Stronghold you implement algorithms on your own, in the Armory you solve similar problems by using existing tools.

Bioinformatics Textbook Track

A collection of exercises to accompany Bioinformatics Algorithms: An Active-Learning Approach by Phillip Compeau & Pavel Pevzner. A full version of this text is hosted on stepic.org

Algorithmic Heights

A collection of exercises in introductory algorithms to accompany "Algorithms", the popular textbook by Dasgupta, Papadimitriou, and Vazirani.


JUST DO IT