BINF2111 – Introduction to Bioinformatics Computing

Dictionaries: The Merriam-Webster of python

Richard Allen White III, PhD
RAW Lab

Lecture 22 – Thursday Nov 18th, 2021

Learning Objectives

- Dictionaries

- Dictionaries vs. lists vs. tuples

- Uses

- Translation

- Coding a translator

- Quiz 22

Python Dictionary

In Python, a dictionary is mapping between a set of indices (keys) and a set of values.

The items in a dictionary are key-value pairs

Python Dictionary

In Python, a dictionary is mapping between a set of indices (keys) and a set of values.

Some Examples of Dictionaries:

Key	Value
trinucleotide	count
Name	protein sequence
name	restriction enzyme motif
codon	amino acid residue
name	email address
sample	coordinates
word	definition

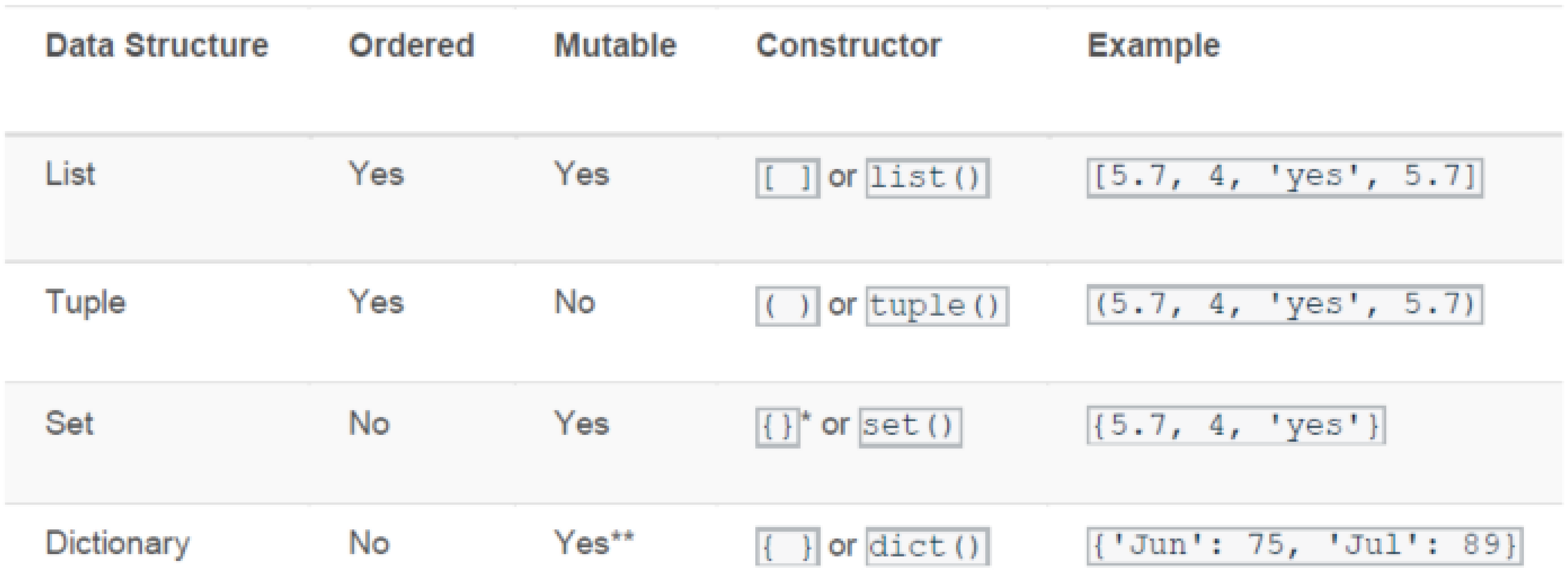
Python Dictionary

In Python, a dictionary is mapping between a set of indices (keys) and a set of values. Often called an associative array or a hash table.

Some Examples of Dictionaries:

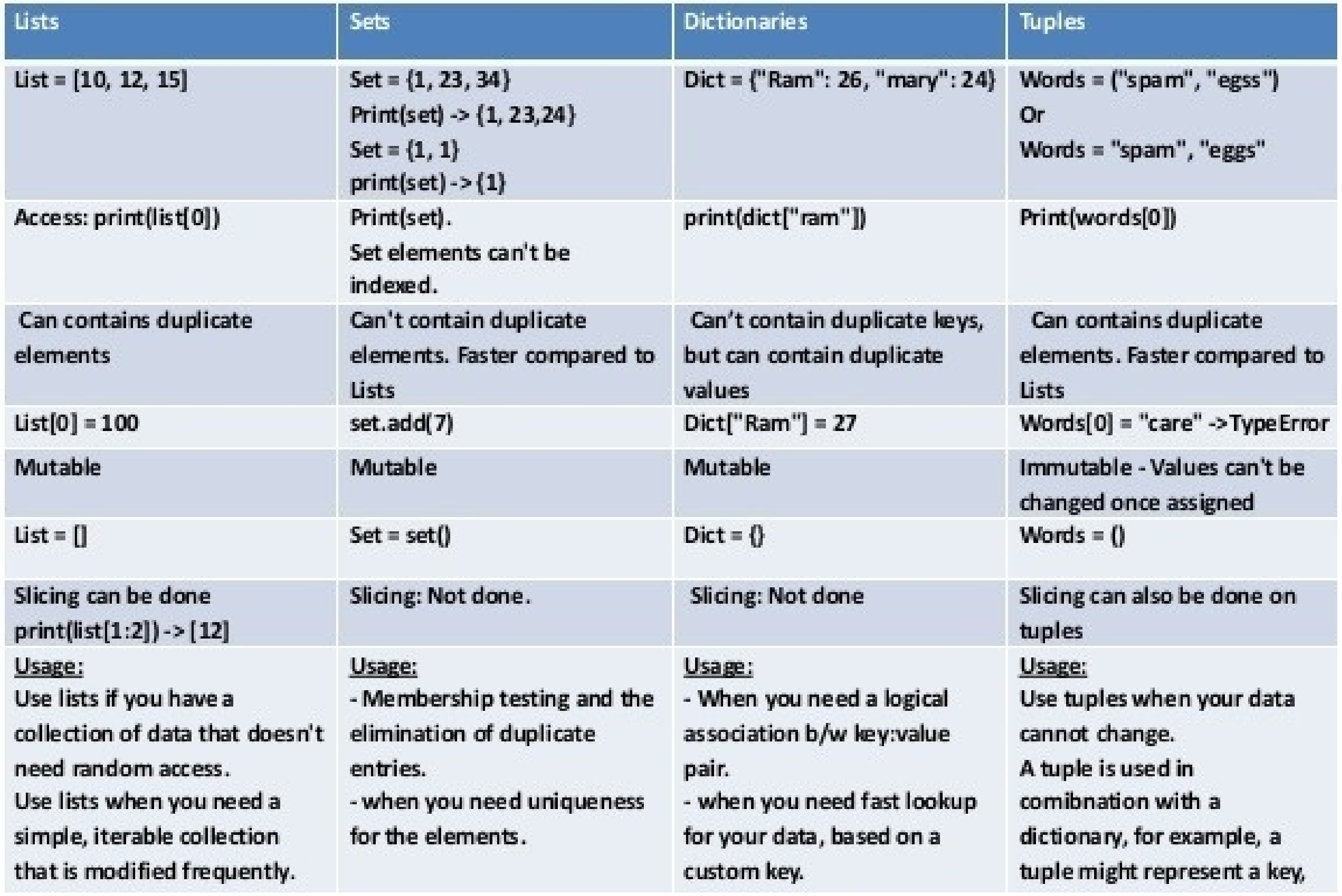
Key	Value
trinucleotide	count
Name	protein sequence
name	restriction enzyme motif
codon	amino acid residue
name	email address
sample	coordinates
word	**definition**

Python Dictionary vs. others



Data Structure	Ordered	Mutable	Constructor	Example
List	Yes	Yes	`[ ]` or `list()`	`[5.7, 4, 'yes', 5.7]`
Tuple	Yes	No	`( )` or `tuple()`	`(5.7, 4, 'yes', 5.7)`
Set	No	Yes	`{}`* or `set()`	`{5.7, 4, 'yes'}`
Dictionary	No	Yes**	`{ }` or `dict()`	`{'Jun': 75, 'Jul': 89}`

Python Dictionary vs. others detailed



Lists	Sets	Dictionaries	Tuples

List = [10, 12, 15]	Set = {1, 23, 34}
	Print(set) -> {1, 23,24}
	Set = {1, 1}
	print(set) -> {1}
		Dict = {"Ram": 26, "mary": 24}	Words = ("spam", "egss")
			Or
			Words = "spam", "eggs"

Access: print(list[0])	Print(set). Set elements can't be indexed.	print(dict["ram"])	Print(words[0])
Can contains duplicate elements	Can't contain duplicate elements. Faster compared to Lists	Can't contain duplicate keys, but can contain duplicate values	Can contains duplicate elements. Faster compared to Lists
List[0] = 100	set.add(7)	Dict["Ram"] = 27	Words[0] = "care" ->TypeError
Mutable	Mutable	Mutable	Immutable - Values can't be changed once assigned
List = []	Set = set()	Dict = {}	Words = ()
Slicing can be done print(list[1:2]) -> [12]	Slicing: Not done.	Slicing: Not done	Slicing can also be done on tuples
<u>Usage:</u> Use lists if you have a collection of data that doesn't need random access. Use lists when you need a simple, iterable collection that is modified frequently.	<u>Usage:</u> - Membership testing and the elimination of duplicate entries. - when you need uniqueness for the elements.	<u>Usage:</u> - When you need a logical association b/w key:value pair. - when you need fast lookup for your data, based on a custom key.	<u>Usage:</u> Use tuples when your data cannot change. A tuple is used in comibnation with a dictionary, for example, a tuple might represent a key,

Creating a Python Dictionary

The syntax for creating a dictionary is similar to that for creating a list, but we use curly brackets rather than square ones.

Each pair of data, consisting of a key and a value, is called an item.

When storing items in a dictionary, we separate them with commas.

Within an individual item, we separate the key and the value with a colon.

```
eng2sp = {'one': 'uno', 'two': 'dos', 'three': 'tres'}
```

Creating a Python Dictionary

The syntax for creating a dictionary is similar to that for creating a list, but we use curly brackets rather than square ones.

Each pair of data, consisting of a key and a value, is called an item.

When storing items in a dictionary, we separate them with commas.

Within an individual item, we separate the key and the value with a colon.

eng2sp = **{**'one': 'uno', 'two': 'dos', 'three': 'tres'**}**

The dictionary is called eng2sp (English to Spanish)

The keys are one, two, and three.

The values are uno, dos, and tres.

Creating a Python Dictionary

To lookup the values for a particular key in a dictionary, we use the following syntax.

```
eng2sp = {'one': 'uno', 'two': 'dos', 'three': 'tres'}

Print(eng2sp[‘one’])
```

This looks very similar to a list, but instead of giving the index of the element that we want, we’re using the key for the value that we want to retrieve.

Note that the dictionary is created using {}, but [] is used to lookup the value.

Creating a Python Dictionary

To lookup the values for a particular key in a dictionary, we use the following syntax.

```
eng2sp = {'one': 'uno', 'two': 'dos', 'three': 'tres'}

Print(eng2sp[‘one’])
```

This looks very similar to a list, but instead of giving the index of the element that we want, we’re using the key for the value that we want to retrieve.

Note that the dictionary is created using {}, but [] is used to lookup the value.

Restrictions on Python Dictionary

Restrictions on dictionaries:

The only types of data we are allowed to use as keys are strings and numbers. These must be **unique**!

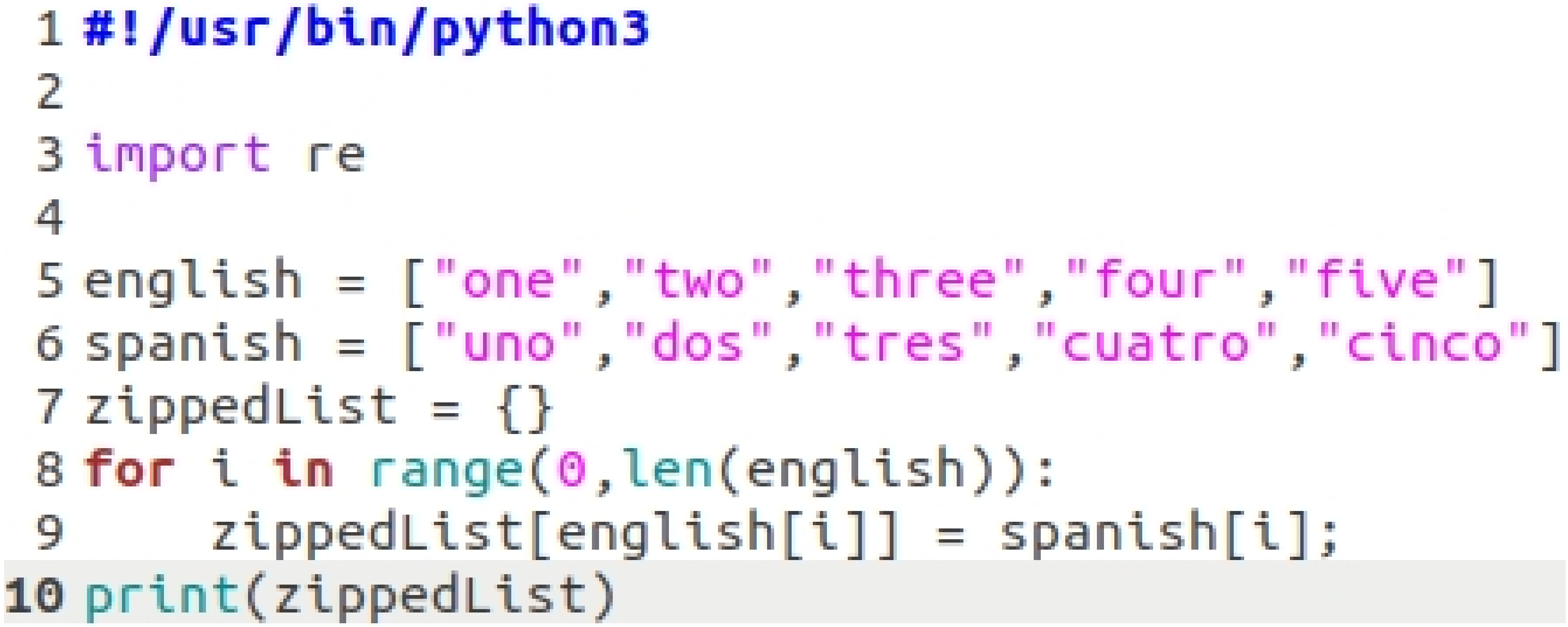
Values can be whatever type of data we like.

Most commonly, we create a dictionary, then add keys and values to it.

```
eng2sp = {}
eng2sp['one‘] = ‘uno‘
eng2sp[‘two‘] = ‘dos‘
eng2sp[‘three‘] = ‘tres‘
```

Making Python Dictionary from a list

You can zip two lists together using a zip (think of a zipper)

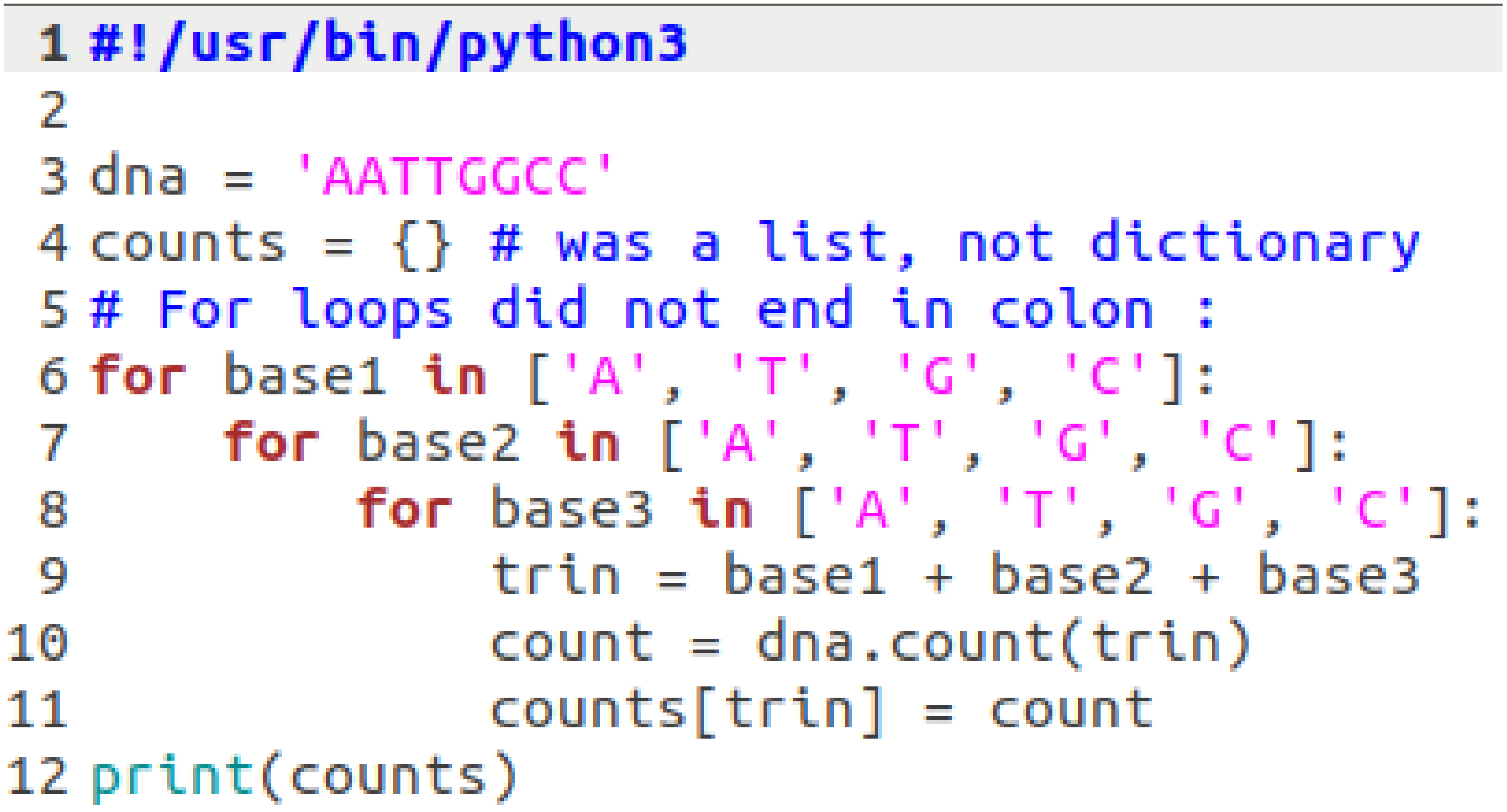


```python
#!/usr/bin/python3

import re

english = ["one","two","three","four","five"]
spanish = ["uno","dos","tres","cuatro","cinco"]
zippedList = {}
for i in range(0,len(english)):
    zippedList[english[i]] = spanish[i];
print(zippedList)
```

Python Dictionary Methods

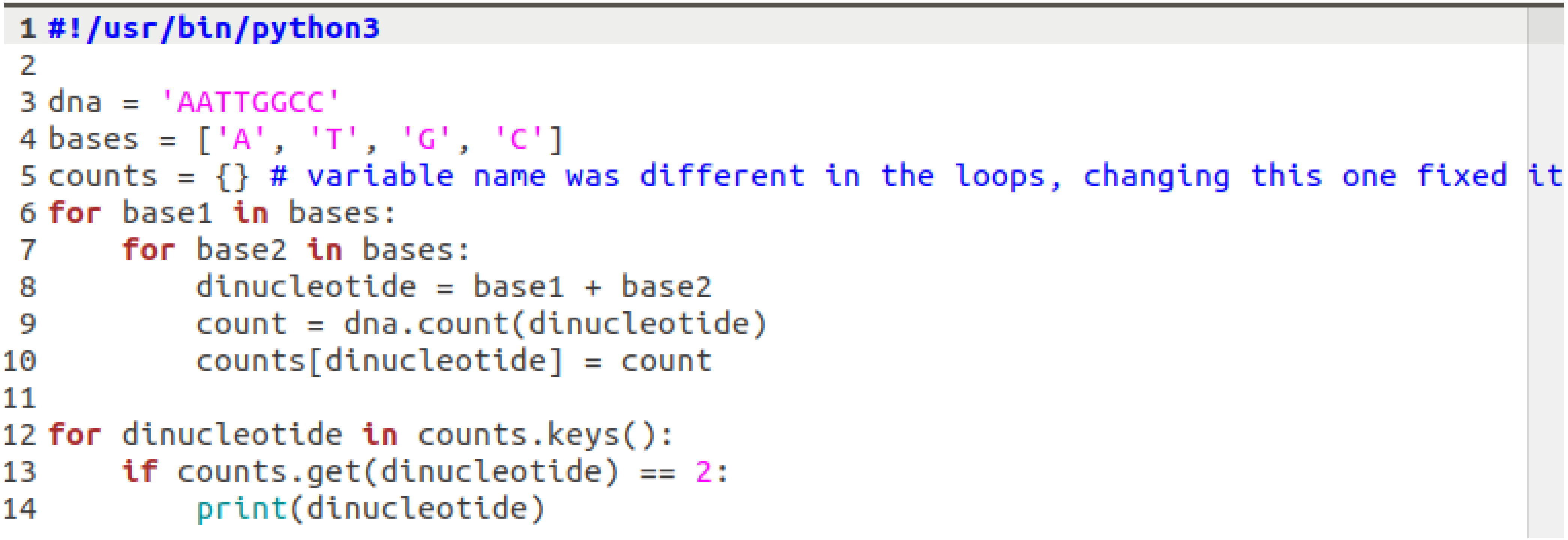
dict.get(key) – get a value
dict.clear() – removes everything in a dictionary
dict.items() – returns the key-value pairs from a dictionary
dict.keys() – returns a list of keys
dict.values()

Find Trinucleotides



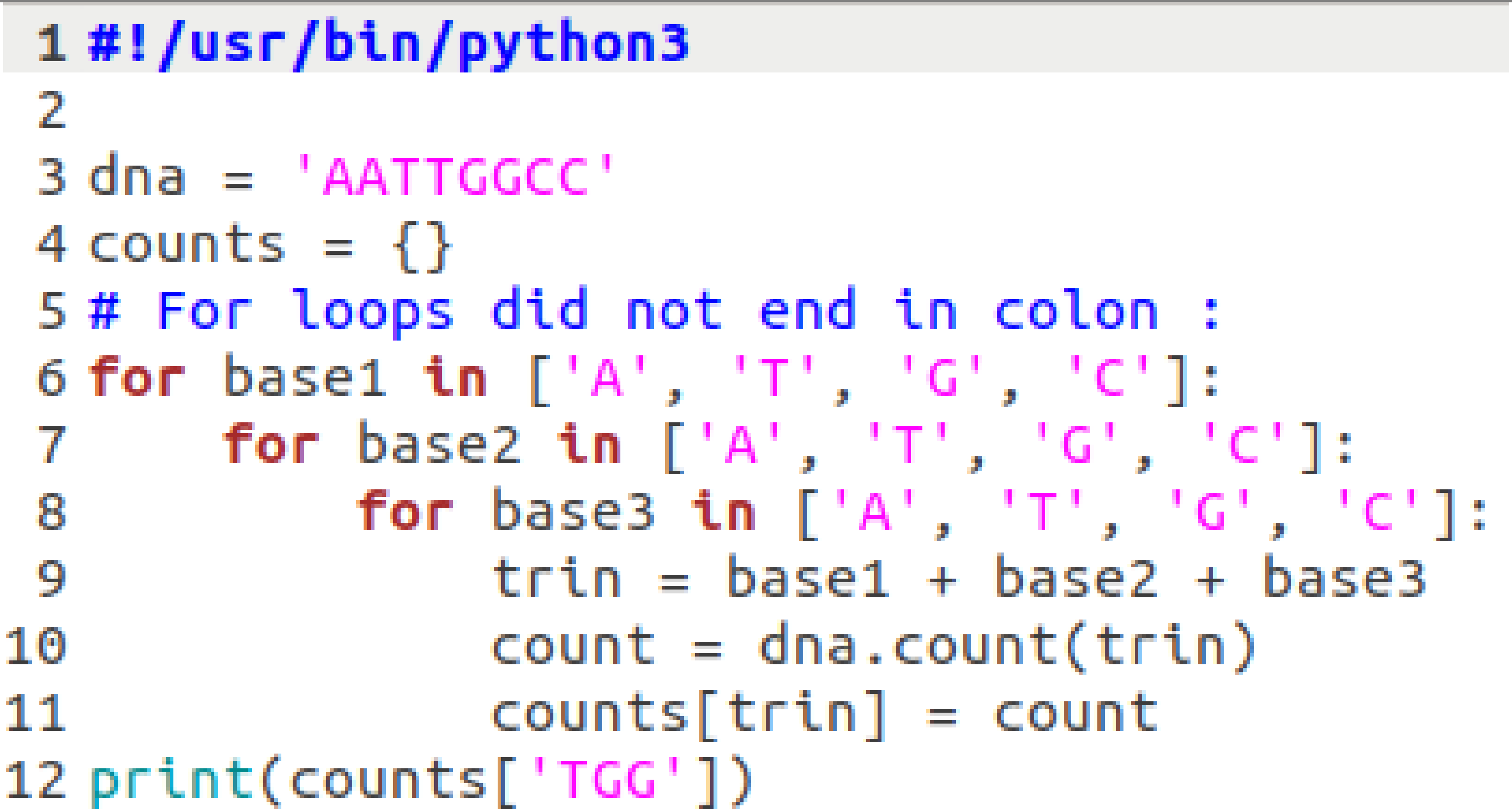
```python
#!/usr/bin/python3

dna = 'AATTGGCC'
counts = {} # was a list, not dictionary
# For loops did not end in colon :
for base1 in ['A', 'T', 'G', 'C']:
    for base2 in ['A', 'T', 'G', 'C']:
        for base3 in ['A', 'T', 'G', 'C']:
            trin = base1 + base2 + base3
            count = dna.count(trin)
            counts[trin] = count
print(counts)
```

Find Trinucleotides (Loops)



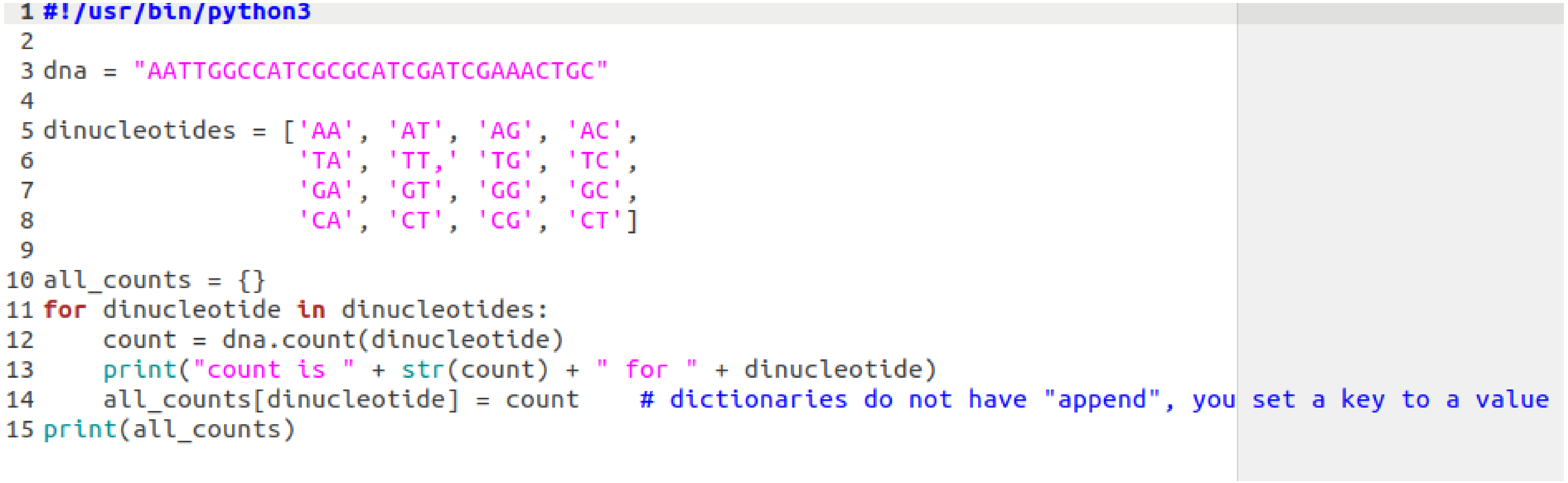
```python
#!/usr/bin/python3

dna = 'AATTGGCC'
bases = ['A', 'T', 'G', 'C']
counts = {} # variable name was different in the loops, changing this one fixed it
for base1 in bases:
    for base2 in bases:
        dinucleotide = base1 + base2
        count = dna.count(dinucleotide)
        counts[dinucleotide] = count

for dinucleotide in counts.keys():
    if counts.get(dinucleotide) == 2:
        print(dinucleotide)
```

Find Trinucleotides - single



```python
#!/usr/bin/python3

dna = 'AATTGGCC'
counts = {}
# For loops did not end in colon :
for base1 in ['A', 'T', 'G', 'C']:
    for base2 in ['A', 'T', 'G', 'C']:
        for base3 in ['A', 'T', 'G', 'C']:
            trin = base1 + base2 + base3
            count = dna.count(trin)
            counts[trin] = count
print(counts['TGG'])
```

Find dinucleotides



```python
#!/usr/bin/python3

dna = "AATTGGCCATCGCGCATCGATCGAAACTGC"

dinucleotides = ['AA', 'AT', 'AG', 'AC',
                 'TA', 'TT,' 'TG', 'TC',
                 'GA', 'GT', 'GG', 'GC',
                 'CA', 'CT', 'CG', 'CT']

all_counts = {}
for dinucleotide in dinucleotides:
    count = dna.count(dinucleotide)
    print("count is " + str(count) + " for " + dinucleotide)
    all_counts[dinucleotide] = count    # dictionaries do not have "append", you set a key to a value
print(all_counts)
```

Why Use a Dict over a List?

A list keeps order, dict doesn’t.
When you care about order, you must use list.

List contains just values
Dict associates with each key a value

```
>seq1
AAAAAACCCCACAGATACAT
>seq2
ACATTAGATATATATTACATAT
```

When we parsed this using a list, we had to create a sequence list and a header list. With a dictionary, we can assign the sequence to the header.

Dictionary

In Python, a dictionary is mapping between a set of indices (keys) and a set of values.

```
>>> dict([(1,'a'), (2,'e'), (3,'i'), (4,'o'), (5,'u')])
    {1: 'a', 2: 'e', 3: 'i', 4: 'o', 5: 'u'}
```

Remember, the great thing about dictionaries is we can find a value instantly, without needing to search through the whole dictionary manually.

You can use the form

```
    value = my_dict['key']
    value = my_dict.get('key').
```

Dictionary for translation

Translation
Write a program that will translate a DNA sequence into protein.

Your program should use the standard genetic code (64 types):

TTT F Phe	**TCT S Ser**	**TAT Y Tyr**	**TGT C Cys**
TTC F Phe	**TCC S Ser**	**TAC Y Tyr**	**TGC C Cys**
TTA L Leu	**TCA S Ser**	**TAA * Ter**	**TGA * Ter**
TTG L Leu	**TCG S Ser**	**TAG * Ter**	**TGG W Trp**
CTT L Leu	**CCT P Pro**	**CAT H His**	**CGT R Arg**
CTC L Leu	**CCC P Pro**	**CAC H His**	**CGC R Arg**
CTA L Leu	**CCA P Pro**	**CAA Q Gln**	**CGA R Arg**
CTG L Leu	**CCG P Pro**	**CAG Q Gln**	**CGG R Arg**
ATT I Ile	**ACT T Thr**	**AAT N Asn**	**AGT S Ser**
ATC I Ile	**ACC T Thr**	**AAC N Asn**	**AGC S Ser**
ATA I Ile	**ACA T Thr**	**AAA K Lys**	**AGA R Arg**
ATG M Met	**ACG T Thr**	**AAG K Lys**	**AGG R Arg**
GTT V Val	**GCT A Ala**	**GAT D Asp**	**GGT G Gly**
GTC V Val	**GCC A Ala**	**GAC D Asp**	**GGC G Gly**
GTA V Val	**GCA A Ala**	**GAA E Glu**	**GGA G Gly**

Dictionary for translation

Here is an example gene: starts with the amino acid M and ends in *

<u>ATGTCGAGATTACCATGGGACAATATACAGTGA</u>

Each triplet in the DNA when translated becomes one of these amino acids

ATGTCGAGATTACCATGGGACAATATACAGTGA
M S R L P W D N I Q *****

TTT F Phe	**TCT S Ser**	**TAT Y Tyr**	**TGT C Cys**
TTC F Phe	**TCC S Ser**	**TAC Y Tyr**	**TGC C Cys**
TTA L Leu	**TCA S Ser**	**TAA * Ter**	**TGA * Ter**
TTG L Leu	**TCG S Ser**	**TAG * Ter**	**TGG W Trp**
CTT L Leu	**CCT P Pro**	**CAT H His**	**CGT R Arg**
CTC L Leu	**CCC P Pro**	**CAC H His**	**CGC R Arg**
CTA L Leu	**CCA P Pro**	**CAA Q Gln**	**CGA R Arg**
CTG L Leu	**CCG P Pro**	**CAG Q Gln**	**CGG R Arg**
ATT I Ile	**ACT T Thr**	**AAT N Asn**	**AGT S Ser**
ATC I Ile	**ACC T Thr**	**AAC N Asn**	**AGC S Ser**
ATA I Ile	**ACA T Thr**	**AAA K Lys**	**AGA R Arg**
ATG M Met	**ACG T Thr**	**AAG K Lys**	**AGG R Arg**
GTT V Val	**GCT A Ala**	**GAT D Asp**	**GGT G Gly**
GTC V Val	**GCC A Ala**	**GAC D Asp**	**GGC G Gly**
GTA V Val	**GCA A Ala**	**GAA E Glu**	**GGA G Gly**

Dictionary for translation

Most DNA is double stranded, so for every strand of DNA there is a **reverse complement.**

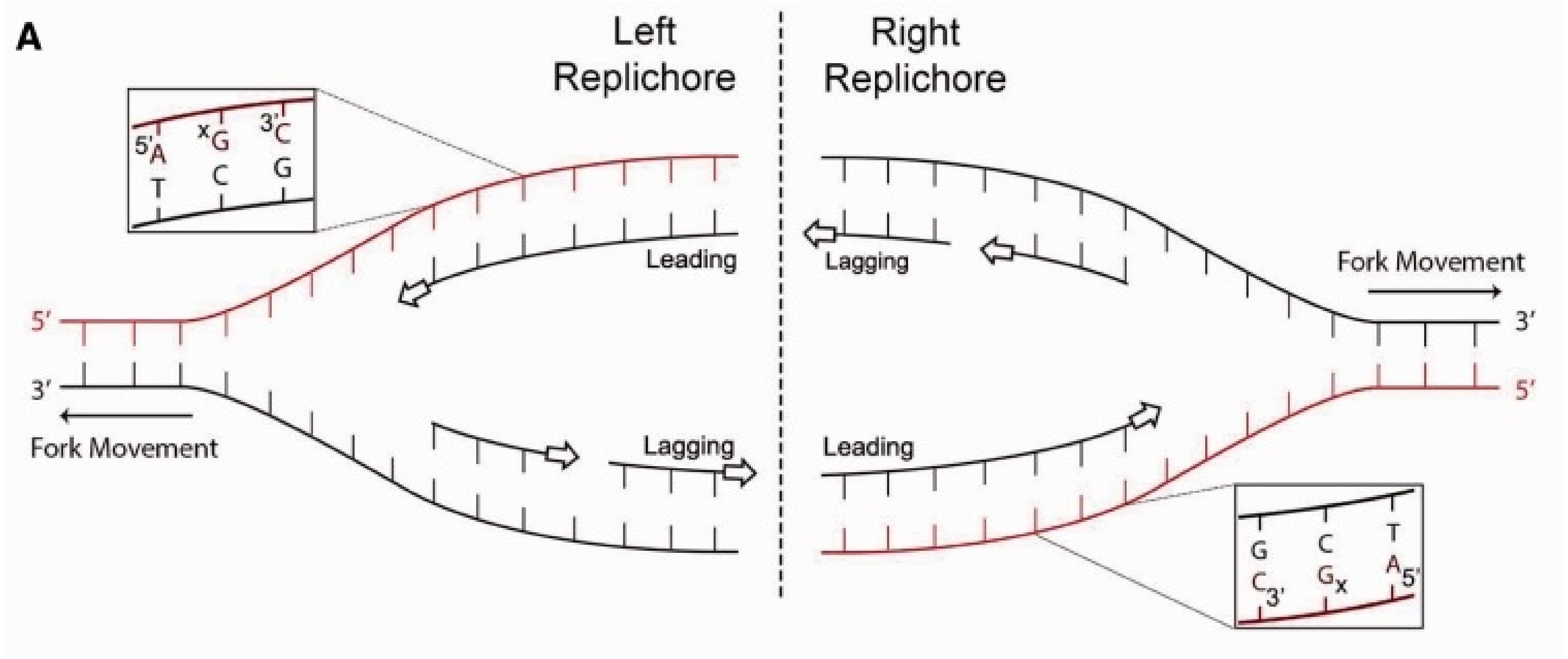
The reverse complement is a reverse string with complementary DNA G>C, A>T, T>A, C>G.

5’ ATGTCGAGATTACCATGGG 3’

3’ TACAGCTCAAATGGTACCC 5’

Dictionary for translation

Most DNA is double stranded, so for every strand of DNA there is a **reverse complement.** During DNA replication the leading and lagging strand are synthesized using leading and lagging strand synthesis from 5'>3'.



Asymmetric Context-Dependent Mutation Patterns Revealed through Mutation–Accumulation Experiments. Sung et al., 2015. MBE

Dictionary for translation

In a single piece of double stranded DNA, there are six frames, forward and reverse. On each strand, there are three frames.

5’ ATCGAACAA 3’   #Frame 1, starts at index position 0 = encodes IEQ
3’ TAGCTTGTT 5’

5’ ATCGAACAA 3’   #Frame 2, starts at index position 1 = encodes SN
3’ TAGCTTGTT 5’

5’ ATCGAACAA 3’   #Frame 3, starts at index position 2 = encodes RT
3’ TAGCTTGTT 5’

Dictionary for translation

In a single piece of double stranded DNA, there are six frames, forward and reverse. On each strand, there are three frames. Forward strand frames:

```
5’ ATCGAACAA 3’          #Frame 1, starts at index position 0
3’ TAGCTTGTT  5’          #Frame 1 is ATC (I), GAA (E), CAA (Q)

5’ ATCGAACAA 3’          #Frame 2, starts at index position 1  = encodes SN
3’ TAGCTTGTT  5’          #Frame 2 is TCG (S), AAC (N)

5’ ATCGAACAA 3’          #Frame 3, starts at index position 2  = encodes RT
3’ TAGCTTGTT  5’          #Frame 3 is CGA (R), ACA (T)

5’ ATCGAACAA 3’          #The fourth index is in the same frame as frame one
3’ TAGCTTGTT  5’
```

Dictionary for translation

In a single piece of double stranded DNA, there are six frames, forward and reverse. On each strand, there are three frames. Reverse strand frames.

Note these frames are reverse complement of the string. (reverse order, and A>T, T>A, G>C, C>G)

```
5’ ATCGAACAA 3’        #Frame 4, starts at index position last index
3’ TAGCTTGTT  5’       #Frame 4 is TTG (L), TTC (F), GAT (D)

5’ ATCGAACAA 3’        #Frame 5, starts at index position -1
3’ TAGCTTGTT  5’       #Frame 5 is TGT (C), TCG (S)

5’ ATCGAACAA 3’        #Frame 6, starts at index position -2
3’ TAGCTTGTT  5’       #Frame is GTT (V), CGA (R)
```

Dictionary for translation

In a single piece of double stranded DNA, there are six frames, forward and reverse. In each strand, there are three frames.

```
5’ ATCGAACAA 3’          #Frame 1, starts at index position 0 = encodes IEQ
3’ TAGCTTGTT  5’

5’ ATCGAACAA 3’          #Frame 2, starts at index position 1  = encodes SN
3’ TAGCTTGTT  5’

5’ ATCGAACAA 3’          #Frame 6, starts in reverse at index position 7 = encodes CS
3’ TAGCTTGTT  5’
```

When working with DNA sequencing, we often don’t know where genes (or other elements) begin. However, we do know that most genes start with a start codon M (ATG), and end with a stop codon * (TAA, TGA, TAG).

Dictionary for translation

In a single piece of double stranded DNA, there are six frames, forward and reverse. In each strand, there are three frames.

```
5’ ATCGAACAA 3’          #Frame 1, starts at index position 0 = encodes IEQ
3’ TAGCTTGTT  5’

5’ ATCGAACAA 3’          #Frame 2, starts at index position 1  = encodes SN
3’ TAGCTTGTT  5’

5’ ATCGAACAA 3’          #Frame 6, starts in reverse at index position 7 = encodes CS
3’ TAGCTTGTT  5’
```

Today, you will work on translating DNA into amino acids. We will perform 6-frame DNA translation.

You will use all the tools that you learned to identify all possible genes within a single piece of DNA.

Quiz 22

- On canvas now