

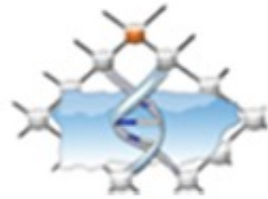


PennState
Huck Institutes of
the Life Sciences

University of Puerto Rico

Puerto Rico - IDeA Networks of Biomedical Research

Excellence



PRINBRE
IDeA Network of Biomedical Research Excellence

Python3 Part 2 – Crash Course in Python3 for Future STEM Coders

Judith S. Rodriguez-Martinez, M.S.

Ph.D. Candidate

Penn State University – University Park Campus

jzr5814@psu.edu



June 5, 2024 at UPR-Ciencias Médicas



- The following material is the result of PR-INBRE research and curriculum development effort to provide a set of educational materials for research training and curriculum changes in biology programs across the island to support PR-INBRE efforts to establish a Community of Practice in Bioinformatics that offers a fruitful environment to increase computational and bioinformatics skills among traditional researchers and students (undergraduate and graduate) in the island. They have been developed as a part of the NIH funded project “**Puerto Rico IDeA Network Biomedical Research Excellence (PRINBRE)**” (Award Number 5P20GM103475).
- Unless otherwise specified, all the information contained within is Copyrighted © by University of Puerto Rico. Permission is granted for use, modify, and reproduce these materials for research and teaching purposes. A copy of the modified material should be sent to help@hpcf.upr.edu.
- Most recent versions of these presentations can be found at <http://inbre.hpcf.upr.edu/>.





Competencies

June 3, 2023 - Record and write simple and common **Python scripts to deal with Bioinformatic needs for biological data analyses** using a Jupyter Notebook

Today – Exploit **datatypes that can record objects**, as well as, reason **logical statements**





Objectives

- Continue to use **Google Colab as an environment** to practice and learn common **Python** lines of code.
- Formulate logical statements through **Booleans and Conditions**
- Create and manage **Python lists and dictionaries**
- Analyze Python Lists and Dictionaries using **Python For Loops and While Loops**





Target Audience

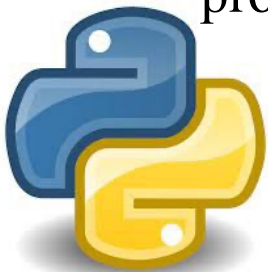
- This training is addressed to beginners, highly motivated (eager to learn what is needed in order to be competitive without the tendency to self-limit when learning computational skills by saying that is difficult) wanting to become familiar with the **Python** programming language and become the Script Master of their bioinformatic analysis.





The importance of Python

- R is great; however, you'll eventually run into the Python language
- No matter your field, python will be a necessity (chemistry, biology, engineering, ecology, mathematics, etc)
- Knowing Python gives you an edge when job searching and navigating graduate research
- The majority bioinformatic tools are python-based and will lead you to understand under the hood code in order to successfully execute said program's purpose.





PennState
Huck Institutes of
the Life Sciences

Lesson 2.0.0 Booleans and Conditions



Booleans

Booleans in Python return whether the statement is **True** or **False**

Operator	Definition	Statement Example
<code>==</code>	Equal-to	<code>2 + 2 == 5</code>
<code>!=</code>	Not equal-to	<code>2 + 2 != 5</code>
<code>></code>	Greater-than	<code>2 > 5</code>
<code><</code>	Less-than	<code>2 < 5</code>
<code>>=</code>	Greater or equal-to	<code>2 >= 2</code> <code>3 >= 2</code>
<code><=</code>	Lesser or equal-to	<code>2 <= 2</code> <code>1 <= 2</code>



Booleans

Booleans in Python return whether the statement is **True** or **False**

Operator	Definition	Statement Example
==	Equal-to	$2 + 2 == 5$
!=	Not equal-to	$2 + 2 != 5$
>	Greater-than	$2 > 5$
<	Less-than	$2 < 5$
>=	Greater or equal-to	$2 \geq 2$ $3 \geq 2$
<=	Lesser or equal-to	$2 \leq 2$ $1 \leq 2$

```
[3] 2 + 2 == 5
```



Booleans

Booleans in Python return whether the statement is **True** or **False**

Operator	Definition	Statement Example
<code>==</code>	Equal-to	<code>2 + 2 == 5</code>
<code>!=</code>	Not equal-to	<code>2 + 2 != 5</code>
<code>></code>	Greater-than	<code>2 > 5</code>
<code><</code>	Less-than	<code>2 < 5</code>
<code>>=</code>	Greater or equal-to	<code>2 >= 2</code> <code>3 >= 2</code>
<code><=</code>	Lesser or equal-to	<code>2 <= 2</code> <code>1 <= 2</code>

```
[3] 2 + 2 == 5
```

```
False
```



Booleans

Booleans in Python return whether the statement is **True** or **False**

Operator	Definition	Statement Example
<code>==</code>	Equal-to	<code>2 + 2 == 5</code>
<code>!=</code>	Not equal-to	<code>2 + 2 != 5</code>
<code>></code>	Greater-than	<code>2 > 5</code>
<code><</code>	Less-than	<code>2 < 5</code>
<code>>=</code>	Greater or equal-to	<code>2 >= 2</code> <code>3 >= 2</code>
<code><=</code>	Lesser or equal-to	<code>2 <= 2</code> <code>1 <= 2</code>

```
[3] 2 + 2 == 5
```

```
False
```

```
[4] 2 + 2 != 5
```



Booleans

Booleans in Python return whether the statement is **True** or **False**

Operator	Definition	Statement Example
<code>==</code>	Equal-to	<code>2 + 2 == 5</code>
<code>!=</code>	Not equal-to	<code>2 + 2 != 5</code>
<code>></code>	Greater-than	<code>2 > 5</code>
<code><</code>	Less-than	<code>2 < 5</code>
<code>>=</code>	Greater or equal-to	<code>2 >= 2</code> <code>3 >= 2</code>
<code><=</code>	Lesser or equal-to	<code>2 <= 2</code> <code>1 <= 2</code>

```
[3] 2 + 2 == 5
```

```
False
```

```
[4] 2 + 2 != 5
```

```
True
```



Booleans

Booleans in Python return whether the statement is **True** or **False**

Operator	Definition	Statement Example
<code>==</code>	Equal-to	<code>2 + 2 == 5</code>
<code>!=</code>	Not equal-to	<code>2 + 2 != 5</code>
<code>></code>	Greater-than	<code>2 > 5</code>
<code><</code>	Less-than	<code>2 < 5</code>
<code>>=</code>	Greater or equal-to	<code>2 >= 2</code> <code>3 >= 2</code>
<code><=</code>	Lesser or equal-to	<code>2 <= 2</code> <code>1 <= 2</code>

```
[3] 2 + 2 == 5
```

```
False
```

```
[4] 2 + 2 != 5
```

```
True
```

```
[45] 2 < 5
```



Booleans

Booleans in Python return whether the statement is **True** or **False**

Operator	Definition	Statement Example
<code>==</code>	Equal-to	<code>2 + 2 == 5</code>
<code>!=</code>	Not equal-to	<code>2 + 2 != 5</code>
<code>></code>	Greater-than	<code>2 > 5</code>
<code><</code>	Less-than	<code>2 < 5</code>
<code>>=</code>	Greater or equal-to	<code>2 >= 2</code> <code>3 >= 2</code>
<code><=</code>	Lesser or equal-to	<code>2 <= 2</code> <code>1 <= 2</code>

```
[3] 2 + 2 == 5
```

```
False
```

```
[4] 2 + 2 != 5
```

```
True
```

```
[45] 2 < 5
```

```
True
```



Conditions

Conditions in Python facilitate decision making in your code

Operator	Decision
if	When the condition is true
else	When the previous condition is false
elif	When the previous conditions are false
if not	When the absence is true



Adding **if** conditions to Booleans

Using an **if** statement

```
[48] if 2 < 5:  
      print(f'Statement is {True}')
```

Answer

Using an **if not** statement

```
[49] if not 2 > 5:  
      print(f'Statement is {False}')
```

Answer



Adding **if** conditions to Booleans

Using an **if** statement

```
[48] if 2 < 5:  
      print(f'Statement is {True}')
```

Statement is True

Using an **if not** statement

```
[49] if not 2 > 5:  
      print(f'Statement is {False}')
```

Answer



Adding **if** conditions to Booleans

Using an **if** statement

```
[48] if 2 < 5:  
      print(f'Statement is {True}')
```

Statement is True

Using an **if not** statement

```
[49] if not 2 > 5:  
      print(f'Statement is {False}')
```

☐ Statement is False



Adding **if-else** conditions to Booleans

Is **ATG** in the sequence?

```
[53] sequence = 'ATGTGGTGGAGA'
     if 'ATG' in sequence:
         print(f'ATG is in {sequence}')
     else:
         print(f'ATG is not in {sequence}')
```

else can be used without a statement so when there are no other options **use else**

Answer



Adding **if-else** conditions to Booleans

Is **ATG** in the sequence?

```
[53] sequence = 'ATGTGGTGGAGA'
      if 'ATG' in sequence:
          print(f'ATG is in {sequence}')
      else:
          print(f'ATG is not in {sequence}')
```

ATG is in ATGTGGTGGAGA

else can be used
without a
statement so when
there are no other
options **use else**



Adding **if-else** conditions to Booleans

Is **ATG** in the sequence?

```
[53] sequence = 'ATGTGGTGGAGA'
      if 'ATG' in sequence:
          print(f'ATG is in {sequence}')
      else:
          print(f'ATG is not in {sequence}')
```

ATG is in ATGTGGTGGAGA

Is **ATG** in this sequence?

```
[54] sequence = 'TGGTGGAGA'
      if 'ATG' in sequence:
          print(f'ATG is in {sequence}')
      else:
          print(f'ATG is not in {sequence}')
```

Answer



Adding **if-else** conditions to Booleans

Is **ATG** in the sequence?

```
[53] sequence = 'ATGTGGTGGAGA'
      if 'ATG' in sequence:
          print(f'ATG is in {sequence}')
      else:
          print(f'ATG is not in {sequence}')
```

ATG is in ATGTGGTGGAGA

Is **ATG** in this sequence?

```
[54] sequence = 'TGGTGGAGA'
      if 'ATG' in sequence:
          print(f'ATG is in {sequence}')
      else:
          print(f'ATG is not in {sequence}')
```

ATG is not in TGGTGGAGA



Adding if-elif-else conditions to Booleans

elif can be used when there are multiple options

```
[39] sequence = 'ATGTGGTGGAGATAG'
      if 'TAA' in sequence:
          print(f'TAA in {sequence}')
      elif 'TAG' in sequence:
          print(f'TAG in {sequence}')
      elif 'TGA' in sequence:
          print(f'TGA in {sequence}')
      else:
          print(f'A stop codon does not exist in {sequence}')
```

Answer



Adding if-elif-else conditions to Booleans

elif can be used when there are multiple options

```
[39] sequence = 'ATGTGGTGGAGATAG'
      if 'TAA' in sequence:
          print(f'TAA in {sequence}')
      elif 'TAG' in sequence:
          print(f'TAG in {sequence}')
      elif 'TGA' in sequence:
          print(f'TGA in {sequence}')
      else:
          print(f'A stop codon does not exist in {sequence}')
```

```
TAG in ATGTGGTGGAGATAG
```



Adding if-elif-else conditions to Booleans

```
[41] sequence = 'ATGTGGTGGAGATAA'
      if 'TAA' in sequence:
          print(f'TAA in {sequence}')
      elif 'TAG' in sequence:
          print(f'TAG in {sequence}')
      elif 'TGA' in sequence:
          print(f'TGA in {sequence}')
      else:
          print(f'A stop codon does not exist in {sequence}')
```

Answer



Adding if-elif-else conditions to Booleans

```
[41] sequence = 'ATGTGGTGGAGATAA'
      if 'TAA' in sequence:
          print(f'TAA in {sequence}')
      elif 'TAG' in sequence:
          print(f'TAG in {sequence}')
      elif 'TGA' in sequence:
          print(f'TGA in {sequence}')
      else:
          print(f'A stop codon does not exist in {sequence}')
```

TAA in ATGTGGTGGAGATAA



Adding if-elif-else conditions to Booleans

```
[32] sequence = 'ATGTGGTGGAGA'
     if 'TAA' in sequence:
         print(f'TAA not in {sequence}')
     elif 'TAG' in sequence:
         print(f'TAA not in {sequence}')
     elif 'TGA' in sequence:
         print(f'TAA not in {sequence}')
     else:
         print(f'A stop codon does not exist in {sequence}')
```

Answer



Adding if-elif-else conditions to Booleans

```
[32] sequence = 'ATGTGGTGGAGA'
     if 'TAA' in sequence:
         print(f'TAA not in {sequence}')
     elif 'TAG' in sequence:
         print(f'TAA not in {sequence}')
     elif 'TGA' in sequence:
         print(f'TAA not in {sequence}')
     else:
         print(f'A stop codon does not exist in {sequence}')
```

A stop codon does not exist in ATGTGGTGGAGA





PennState
Huck Institutes of
the Life Sciences

Lesson 2.1.0

More Python datatypes





PennState
Huck Institutes of
the Life Sciences

Lesson 2.1.1

Lists



Starting lists

Method 1: Create a **list** of elements

```
[5] stop_codons = ['TAG', 'TGA', 'ATG']  
stop_codons
```

Answer

Store multiple
elements in a
list using
brackets.



Starting lists

Method 1: Create a **list** of elements

```
[5] stop_codons = ['TAG', 'TGA', 'ATG']  
stop_codons  
  
['TAG', 'TGA', 'ATG']
```

Store multiple
elements in a
list using
brackets.



Starting lists

Method 2: Start an **empty list**

```
[18] empty_list = []  
      empty_list
```

Answer

```
[19] empty_list.append('TAG')  
      empty_list.append('TAA')  
      empty_list.append('TGA')  
      empty_list
```

Answer

The **append(*elements*)** function will add new elements to an existing list.



Starting lists

Method 2: Start an empty list

```
[18] empty_list = []  
      empty_list
```

```
      []
```

```
[19] empty_list.append('TAG')  
      empty_list.append('TAA')  
      empty_list.append('TGA')  
      empty_list
```

Answer

The **append(*elements*)** function will add new elements to an existing list.



Starting lists

Method 2: Start an empty list

```
[18] empty_list = []  
      empty_list
```

```
      []
```

```
[19] empty_list.append('TAG')  
      empty_list.append('TAA')  
      empty_list.append('TGA')  
      empty_list
```

```
      ['TAG', 'ATG', 'TGA']
```

The **append(*elements*)** function will add new elements to an existing list.



Functions you can use to evaluate and manipulate **lists**

Note: These **functions** are used similarly to **string** objects!

`len(list)`

`list.count("")`

`list.remove(element)`

`list.append(element)`

`list.sort()`

`list.reverse()`



Let's fix our stop_codons **lists**

What is wrong with the stop_codons **list**?

```
[5] stop_codons = ['TAG', 'TGA', 'ATG']  
stop_codons  
  
['TAG', 'TGA', 'ATG']
```



Let's fix our stop_codons **lists**

What is wrong with the stop_codons **list**?

```
[5] stop_codons = ['TAG', 'TGA', 'ATG']  
stop_codons
```

```
['TAG', 'TGA', 'ATG']
```

TAA ?

Not a
stop
codon



Let's fix our stop_codons lists

How should we fix this?

```
[5] stop_codons = ['TAG', 'TGA', 'ATG']  
stop_codons  
  
['TAG', 'TGA', 'ATG']
```

Answer



Let's fix our stop_codons lists

How should we fix this?

```
[5] stop_codons = ['TAG', 'TGA', 'ATG']
stop_codons

['TAG', 'TGA', 'ATG']
```

```
[6] stop_codons.remove('ATG')
stop_codons
```

```
['TAG', 'TGA']
```

```
[7] stop_codons.append('TAA')
stop_codons
```

```
['TAG', 'TGA', 'TAA']
```





PennState
Huck Institutes of
the Life Sciences

Lesson 2.1.2

Dictionaries



Starting a dictionary

Method 1: Create a dictionary with elements

```
[33] leucine_codons = {'CTT': 'Leu', 'CTC': 'Leu',  
                       'CTA': 'Leu', 'CTG': 'Leu',  
                       'TTA': 'Leu', 'TTG': 'Leu'}
```

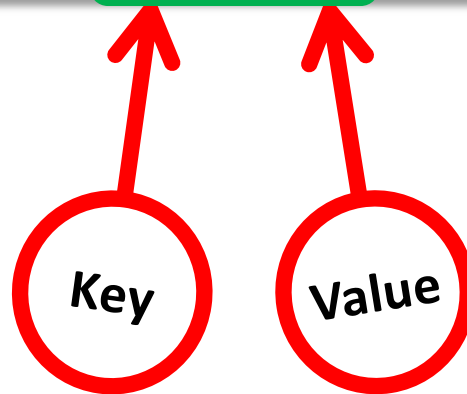
An array of association that stores a key and its value and is represented by a colon



Starting a dictionary

Method 1: Create a **dictionary** with elements

```
[33] leucine_codons = {'CTT': 'Leu', 'CTC': 'Leu',  
                      'CTA': 'Leu', 'CTG': 'Leu',  
                      'TTA': 'Leu', 'TTG': 'Leu'}
```



An array of association that stores a key and its value and is represented by a colon



Starting a dictionary

Method 1: Create a **dictionary** with elements

```
[33] leucine_codons = {'CTT': 'Leu', 'CTC': 'Leu',  
                      'CTA': 'Leu', 'CTG': 'Leu',  
                      'TTA': 'Leu', 'TTG': 'Leu'}  
  
leucine_codons
```

Answer

An array of association that stores a key and its value and is represented by a colon



Starting a dictionary

Method 1: Create a dictionary with elements

```
[33] leucine_codons = {'CTT': 'Leu', 'CTC': 'Leu',  
                      'CTA': 'Leu', 'CTG': 'Leu',  
                      'TTA': 'Leu', 'TTG': 'Leu'}
```

leucine_codons

```
{'CTT': 'Leu',  
 'CTC': 'Leu',  
 'CTA': 'Leu',  
 'CTG': 'Leu',  
 'TTA': 'Leu',  
 'TTG': 'Leu'}
```

An array of association that stores a key and its value and is represented by a colon



Starting a dictionary

Method 1: Create a dictionary with elements

```
[33] leucine_codons = {'CTT': 'Leu', 'CTC': 'Leu',  
                      'CTA': 'Leu', 'CTG': 'Leu',  
                      'TTA': 'Leu', 'TTG': 'Leu'}
```

leucine_codons

```
{'CTT': 'Leu',  
 'CTC': 'Leu',  
 'CTA': 'Leu',  
 'CTG': 'Leu',  
 'TTA': 'Leu',  
 'TTG': 'Leu'}
```

An array of association that stores a key and its value and is represented by a colon



Key

Value

Starting a dictionary

Method 2: Add elements to an empty/existing dictionary

```
[36] empty_dictionary = {}  
      empty_dictionary
```

Answer

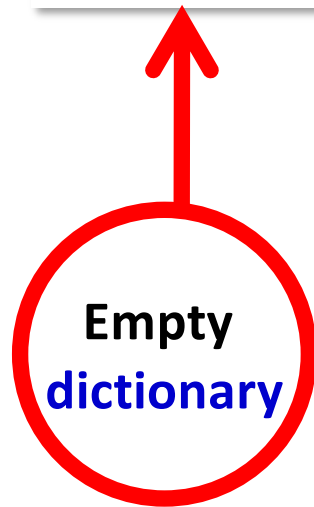


Starting a dictionary

Method 2: Add elements to an empty/existing dictionary

```
[36] empty_dictionary = {}  
      empty_dictionary
```

```
{}
```



Adding elements to a dictionary

Method 2: Add elements to an empty/existing dictionary

```
[36] empty_dictionary = {}  
      empty_dictionary
```

```
{}
```

```
[37] empty_dictionary['CTT'] = 'Leu'  
      empty_dictionary['CTC'] = 'Leu'  
      empty_dictionary['CTA'] = 'Leu'  
      empty_dictionary['CTG'] = 'Leu'  
      empty_dictionary['TTA'] = 'Leu'  
      empty_dictionary['TTG'] = 'Leu'
```

Note: the **value** of a key can be any **data type** which include strings, integers, floats, lists, or nested dictionaries



Adding elements to a dictionary

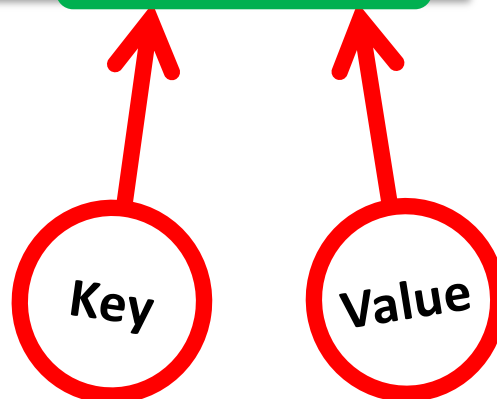
Method 2: Add elements to an empty/existing dictionary

```
[36] empty_dictionary = {}  
      empty_dictionary
```

```
{}
```

```
[37] empty_dictionary['CTT'] = 'Leu'  
      empty_dictionary['CTC'] = 'Leu'  
      empty_dictionary['CTA'] = 'Leu'  
      empty_dictionary['CTG'] = 'Leu'  
      empty_dictionary['TTA'] = 'Leu'  
      empty_dictionary['TTG'] = 'Leu'
```

Note: the **value** of a key can be any **data type** which include strings, integers, floats, lists, or nested dictionaries



Adding elements to a dictionary

Method 2. Start an empty dictionary using curly brackets

```
[36] empty_dictionary = {}  
      empty_dictionary
```

```
{}
```

```
[37] empty_dictionary['CTT'] = 'Leu'  
      empty_dictionary['CTC'] = 'Leu'  
      empty_dictionary['CTA'] = 'Leu'  
      empty_dictionary['CTG'] = 'Leu'  
      empty_dictionary['TTA'] = 'Leu'  
      empty_dictionary['TTG'] = 'Leu'
```

Note: the **value of a key can be any **data type** which include strings, integers, floats, lists, or nested dictionaries**

So what data type are these keys storing?



Adding elements to a dictionary

Method 2. Start an empty dictionary using curly brackets

```
[36] empty_dictionary = {}  
      empty_dictionary
```

```
{}
```

```
[37] empty_dictionary['CTT'] = 'Leu'  
      empty_dictionary['CTC'] = 'Leu'  
      empty_dictionary['CTA'] = 'Leu'  
      empty_dictionary['CTG'] = 'Leu'  
      empty_dictionary['TTA'] = 'Leu'  
      empty_dictionary['TTG'] = 'Leu'  
      empty_dictionary
```

Note: the **value of a key can be any **data type** which include strings, integers, floats, lists, or nested dictionaries**

Answer



Adding elements to a dictionary

Method 2. Start an empty dictionary using curly brackets

```
[36] empty_dictionary = {}  
      empty_dictionary
```

```
{}
```

```
[37] empty_dictionary['CTT'] = 'Leu'  
      empty_dictionary['CTC'] = 'Leu'  
      empty_dictionary['CTA'] = 'Leu'  
      empty_dictionary['CTG'] = 'Leu'  
      empty_dictionary['TTA'] = 'Leu'  
      empty_dictionary['TTG'] = 'Leu'  
      empty_dictionary
```

```
{'CTT': 'Leu',
```

```
'CTC': 'Leu',  
'CTA': 'Leu',  
'CTG': 'Leu',  
'TTA': 'Leu',  
'TTG': 'Leu'}
```

Key

Value

Note: the **value of a key can be any **data type** which include strings, integers, floats, lists, or nested dictionaries**



How to use a dictionary

```
[8] leucine_codons
```

```
{ 'CTT': 'Leu',  
  'CTC': 'Leu',  
  'CTA': 'Leu',  
  'CTG': 'Leu',  
  'TTA': 'Leu',  
  'TTG': 'Leu' }
```

Note: Calling a key returns its value



How to use a dictionary

```
[8] leucine_codons
```

```
{'CTT': 'Leu',  
 'CTC': 'Leu',  
 'CTA': 'Leu',  
 'CTG': 'Leu',  
 'TTA': 'Leu',  
 'TTG': 'Leu'}
```

Note: Calling a key returns its value

Let's **call (find/consult)** the key 'CTG'



How to use a dictionary

```
[8] leucine_codons
```

```
{ 'CTT': 'Leu',  
  'CTC': 'Leu',  
  'CTA': 'Leu',  
  'CTG': 'Leu',  
  'TTA': 'Leu',  
  'TTG': 'Leu' }
```

Note: Calling a key returns its value

Let's call (find/consult) the key 'CTG'

```
[9]
```

```
leucine_codons['CTG']
```

1



How to use a dictionary

```
[8] leucine_codons
```

```
{ 'CTT': 'Leu',  
  'CTC': 'Leu',  
  'CTA': 'Leu',  
  'CTG': 'Leu',  
  'TTA': 'Leu',  
  'TTG': 'Leu' }
```

```
[9]
```

```
leucine_codons['CTG']
```

Answer

Note: Calling a key
returns
(produces/result)
its value



How to use a dictionary

```
[8] leucine_codons
```

```
{'CTT': 'Leu',  
 'CTC': 'Leu',  
 'CTA': 'Leu',  
 'CTG': 'Leu',  
 'TTA': 'Leu',  
 'TTG': 'Leu'}
```

```
[9]
```

```
leucine_codons['CTG']
```

```
'Leu'
```

Note: Calling a key
returns
(produces/result)
its value



2

A dictionary with all codons and their associated amino acid can be used to translate a protein-coding sequence

```
[3] codon_table = {
    'TCA': 'S', 'TCC': 'S', 'TCG': 'S', 'TCT': 'S', 'AGC': 'S', 'AGT': 'S', # Serine
    'TTC': 'F', 'TTT': 'F', # Phenylalanine
    'TTA': 'L', 'TTG': 'L', 'CTA': 'L', 'CTC': 'L', 'CTG': 'L', 'CTT': 'L', # Leucine
    'TAC': 'Y', 'TAT': 'Y', # Tyrosine
    'TAA': '*', 'TAG': '*', 'TGA': '*', # Stop
    'TGC': 'C', 'TGT': 'C', # Cysteine
    'TGG': 'W', # Tryptophane
    'CCA': 'P', 'CCC': 'P', 'CCG': 'P', 'CCT': 'P', # Proline
    'CAC': 'H', 'CAT': 'H', # Histidine
    'CAA': 'Q', 'CAG': 'Q', # Glutamine
    'CGA': 'R', 'CGC': 'R', 'CGG': 'R', 'CGT': 'R', # Arginine
    'ATA': 'I', 'ATC': 'I', 'ATT': 'I', # Isoleucine
    'ATG': 'M', # Methionine
    'ACA': 'T', 'ACC': 'T', 'ACG': 'T', 'ACT': 'T', # Threonine
    'AAC': 'N', 'AAT': 'N', # Asparagine
    'AAA': 'K', 'AAG': 'K', # Lysine
    'AGA': 'R', 'AGG': 'R', # Arginine
    'GTA': 'V', 'GTC': 'V', 'GTG': 'V', 'GTT': 'V', # Valine
    'GCA': 'A', 'GCC': 'A', 'GCG': 'A', 'GCT': 'A', # Alanine
    'GAC': 'D', 'GAT': 'D', # Aspartic Acid
    'GAA': 'E', 'GAG': 'E', # Glutamic Acid
    'GGA': 'G', 'GGC': 'G', 'GGG': 'G', 'GGT': 'G', # Glycine
}
```



Lesson 2.2.0

Automating repetitive tasks using Python For loops





Lesson 2.2.1

For Loops



Let's say I have a list of μL measurements
that I want to convert to mL

measurement	μL	mL
0	5	?
1	47	?
2	17	?
3	91	?



Let's say I have a list of μL measurements that I want to convert to mL

measurement	μL	mL
0	5	?
1	47	?
2	17	?
3	91	?

Solution option 1:

```
[4] 5 * (0.001/1)
```

```
0.005
```

```
[5] 47 * (0.001/1)
```

```
0.047
```

```
[6] 17 * (0.001/1)
```

```
0.017
```

```
[7] 91 * (0.001/1)
```

```
0.091
```



Let's say I have a list of μL measurements that I want to convert to mL

measurement	μL	mL
0	5	?
1	47	?
2	17	?
3	91	?

Solution option 2:

Create a variable x and substitute for each value

```
[1] x=5  
x * (0.001/1)
```

0.005

```
[2] x=47  
x * (0.001/1)
```

0.047

```
[3] x=17  
x * (0.001/1)
```

0.017

```
[4] x=91  
x * (0.001/1)
```

0.091



What if we have a hundred μL measurements that we converted to mL?

measurement	μL	mL
0	5	?
1	47	?
2	17	?
3	91	?
.	.	?
.	.	?
.	.	?
99	77	?



What if we have a hundred μL measurements that we converted to mL?

measurement	μL	mL
0	5	?
1	47	?
2	17	?
3	91	?
.	.	?
.	.	?
.	.	?
99	77	?



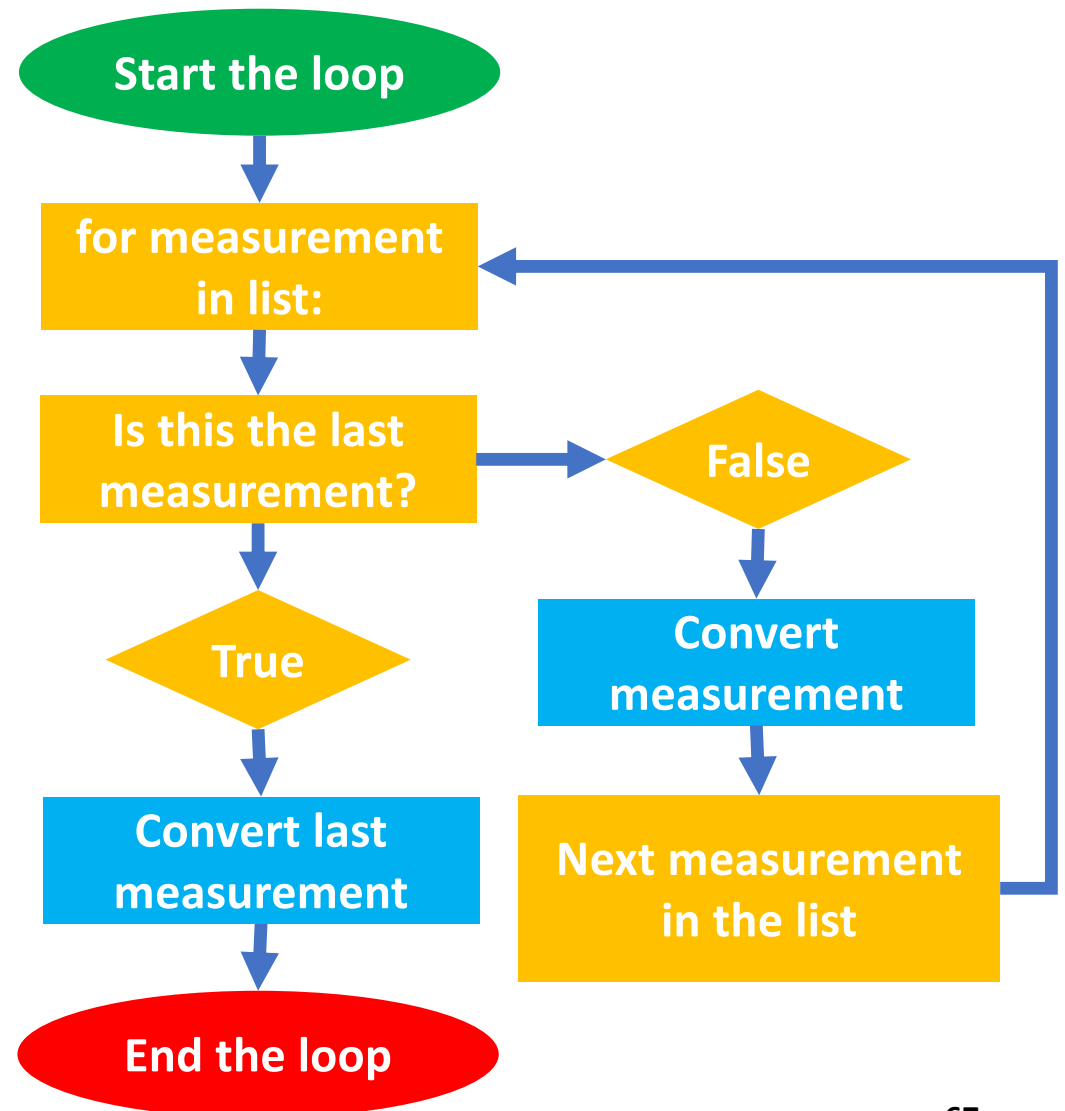
Convert to a **list**

What if we have a hundred μL measurements that we converted to mL?

measurement	μL	mL
0	5	?
1	47	?
2	17	?
3	91	?
.	.	?
.	.	?
.	.	?
99	77	?



Convert to a **list**



Let's say I have a list of μL measurements
that I want to convert to mL

measurement	μL	mL
0	5	?
1	47	?
2	17	?
3	91	?

Solution option 3:

```
[9] measurements=[5,27,17,91]
```

Answer



Let's say I have a list of μL measurements that I want to convert to mL

measurement	μL	mL
0	5	?
1	47	?
2	17	?
3	91	?

Solution option 3:

```
[9] measurements=[5,27,17,91]

for x in measurements:
    print(f'iteration for {x}' )
    result = x * (0.001/1)
    print(f'Measurement (microliter): {x}')
    print(f'Measurement (milliliter): {result}')
```

```
iteration for 5
Measurement (microliter): 5
Measurement (milliliter): 0.005
iteration for 27
Measurement (microliter): 27
Measurement (milliliter): 0.027
iteration for 17
Measurement (microliter): 17
Measurement (milliliter): 0.017
iteration for 91
Measurement (microliter): 91
Measurement (milliliter): 0.091
```





Lesson 2.2.2

Exploit information from sequences using **For loops**



Can we **For loop** through a string that is a sequence?

```
[4] sequence="ATGTGGTGG"  
sequence
```

```
'ATGTGGTGG'  
012345678
```



Can we **For loop** through a string that is a sequence?

```
[4] sequence="ATGTGGTGG"  
sequence
```

```
'ATGTGGTGG'  
012345678
```

```
[11] for nucleotide in sequence:  
      print(nucleotide)
```

Answer



Can we **For loop** through a string that is a sequence?

```
[4] sequence="ATGTGGTGG"  
sequence
```

```
'ATGTGGTGG'  
012345678
```

```
[11] for nucleotide in sequence:  
      print(nucleotide)
```

```
A  
T  
G  
T  
G  
G  
T  
G  
G
```



For looping through a string that is a sequence?

```
[4] sequence="ATGTGGTGG"  
sequence
```

```
'ATGTGGTGG'  
012345678
```

```
[11] for nucleotide in sequence:  
      print(nucleotide)
```

```
A  
T  
G  
T  
G  
G  
T  
G  
G
```

**What if we
need to use
the indexes of
this sequence?**



For loop through the indexes of a string that is a sequence

How would you produce indexes?

```
[4] sequence="ATGTGGTGG"  
sequence
```

```
'ATGTGGTGG'  
012345678
```



Using indexes to slice a String

Index a range of the sequence

'ATGTGGTGG'
0 1 2 3 4 5 6 7 8

[9] sequence[2:8]

Index is
included and
creates a new
string

'GTGGTG'
2 7

Index was
not included
in new string



For loop through the indexes of a string that is a sequence

How would you produce indexes?

Hint: a combination between **range()** and **len()**

```
[4] sequence="ATGTGGTGG"  
sequence
```

```
'ATGTGGTGG'  
012345678
```



For loop through the indexes of a string that is a sequence

How would you produce indexes?

```
[4] sequence="ATGTGGTGG"  
sequence
```

```
'ATGTGGTGG'  
012345678
```



Hint: a combination between **range()** and **len()**

```
[12] for nucleotide_position in range(len(sequence)):  
      print(nucleotide_position)  
      print(sequence[nucleotide_position])
```

Answer



For loop through the indexes of a string that is a sequence

How would you produce indexes?

```
[4] sequence="ATGTGGTGG"  
sequence
```

```
'ATGTGGTGG'  
012345678
```



Hint: a combination between **range()** and **len()**

```
[12] for nucleotide_position in range(len(sequence)):  
      print(nucleotide_position)  
      print(sequence[nucleotide_position])
```

```
0  
A  
1  
T  
2  
G  
3  
T  
4  
G  
5  
G  
6  
T  
7  
G  
8  
G
```



For loop through the indexes of a string that is a sequence

Do we remember how to index a **range** of a **string**?



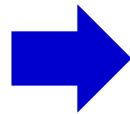
For loop through the indexes of a string that is a sequence

How would you produce a coding sequence?



INPUT

'ATGTGGTGG'
0 1 2 3 4 5 6 7 8



OUTPUT

['AUG', 'UGG', 'UGG']
0 1 2



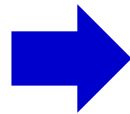
For loop through the indexes of a string that is a sequence

How would you produce a coding sequence?



INPUT

'ATGTGGTGG'
0 1 2 3 4 5 6 7 8



OUTPUT

['AUG', 'UGG', 'UGG']
0 1 2

```
[12] sequence="ATGTGGTGG"
```

Identify your input
sequence



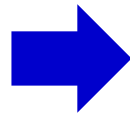
For loop through the indexes of a string that is a sequence

How would you produce a coding sequence?



INPUT

'ATGTGGTGG'
0 1 2 3 4 5 6 7 8



OUTPUT

['AUG', 'UGG', 'UGG']
0 1 2

```
[12] sequence="ATGTGGTGG"
```

```
RNA_sequence=sequence.replace("T","U")
```

Replace T for U in
the sequence



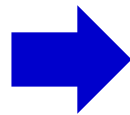
For loop through the indexes of a string that is a sequence

How would you produce a coding sequence?



INPUT

'ATGTGGTGG'
0 1 2 3 4 5 6 7 8



OUTPUT

['AUG', 'UGG', 'UGG']
0 1 2

```
[12] sequence="ATGTGGTGG"
```

```
RNA_sequence=sequence.replace("T","U")
```

```
coding_sequence=[]
```

Initiate an empty list



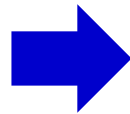
For loop through the indexes of a string that is a sequence

How would you produce a coding sequence?



INPUT

'ATGTGGTGG'
0 1 2 3 4 5 6 7 8



OUTPUT

['AUG', 'UGG', 'UGG']
0 1 2

```
[12] sequence="ATGTGGTGG"

RNA_sequence=sequence.replace("T","U")

coding_sequence=[]
temp=0
```

Initiate your counter at 0



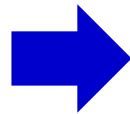
For loop through the indexes of a string that is a sequence

How would you produce a coding sequence?



INPUT

'ATGTGGTGG'
0 1 2 3 4 5 6 7 8



OUTPUT

['AUG', 'UGG', 'UGG']
0 1 2

```
[12] sequence="ATGTGGTGG"
```

```
RNA_sequence=sequence.replace("T","U")
```

```
coding_sequence=[]
```

```
temp=0
```

```
for x in RNA_sequence:
```

Start a for loop



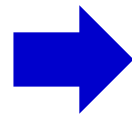
For loop through the indexes of a string that is a sequence

How would you produce a coding sequence?



INPUT

'ATGTGGTGG'
0 1 2 3 4 5 6 7 8



OUTPUT

['AUG', 'UGG', 'UGG']
0 1 2

```
[12] sequence="ATGTGGTGG"

RNA_sequence=sequence.replace("T","U")

coding_sequence=[]
temp=0
for x in RNA_sequence:
    if RNA_sequence[temp:temp+3] != '':
```

If statement



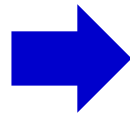
For loop through the indexes of a string that is a sequence

How would you produce a coding sequence?



INPUT

'ATGTGGTGG'
0 1 2 3 4 5 6 7 8



OUTPUT

['AUG', 'UGG', 'UGG']
0 1 2

```
[12] sequence="ATGTGGTGG"

RNA_sequence=sequence.replace("T","U")

coding_sequence=[]
temp=0
for x in RNA_sequence:
    if RNA_sequence[temp:temp+3] != '':
        coding_sequence.append(RNA_sequence[temp:temp+3])
        temp=temp+3
```

The body of the
If statement



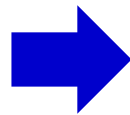
For loop through the indexes of a string that is a sequence

How would you produce a coding sequence?



INPUT

'ATGTGGTGG'
0 1 2 3 4 5 6 7 8



OUTPUT

['AUG', 'UGG', 'UGG']
0 1 2

```
[12] sequence="ATGTGGTGG"

RNA_sequence=sequence.replace("T","U")

coding_sequence=[]
temp=0
for x in RNA_sequence:
    if RNA_sequence[temp:temp+3] != '':
        coding_sequence.append(RNA_sequence[temp:temp+3])
        temp=temp+3
print(coding_sequence)

['AUG', 'UGG', 'UGG']
```





Proficiency Assessment

Exercise 1: Return the total number of guanines Sequence A.

You can use the `count()` function in python to check your work! 😊

Start with a variable called counter equal to 0

For loop through each nucleotide of the sequence

Use a condition statement to identify a guanine in the sequence

Record by tallying the guanines found

Exercise 2: How many differences are between Sequence A and Sequence B

Refer to Exercise 1 for ideas

Exercise 3: You will be given a list of protein-coding sequences and other resources (i.e. dictionary). Find the following for each sequence:

Length of the DNA sequence

Transcript of the DNA sequence (RNA sequence)

Coding sequence as a list

Amino acid sequence

Bonus: add something yourself! 😊

