




Topic	MATRIX OPERATIONS	
Class Description	The student will learn about 1D and 2D arrays. Also, perform different operations on 2D arrays.	
Class	PRO C124	
Class time	45 mins	
Goal	<ul style="list-style-type: none"> Understanding basic operation of 2D Python lists Accessing and updating elements of 2D lists Adding and subtracting two 2D lists Scalar multiplication of 2D matrices 	
Resources Required	<ul style="list-style-type: none"> Teacher Resources: <ul style="list-style-type: none"> Laptop with internet connectivity Earphones with mic Notebook and pen Smartphone Student Resources: <ul style="list-style-type: none"> Laptop with internet connectivity Earphones with mic Notebook and pen 	
Class structure	Warm-Up Teacher-Led Activity 1 Student-Led Activity 1 Wrap-Up	10 mins 10 mins 20 mins 05 mins
Credit	NumPy under BSD license	
WARM-UP SESSION - 10 mins		
<div>  </div> <p>Teacher Starts Slideshow Slide 1 to 4</p> <p>Refer to speaker notes and follow the instructions on each slide.</p>		

Teacher Action	Student Action
<p>Hey <student's name>. How are you? It's great to see you! Are you excited to learn something new today?</p> <p>Following are the WARM-UP session deliverables:</p> <ul style="list-style-type: none"> • Greet the student. • Revision of previous class activities. • Quizzes. 	<p>ESR: Hi, thanks! Yes, I am excited about it!</p> <p>Click on the slide show tab and present the slides</p>
<p align="center">WARM-UP QUIZ Click on In-Class Quiz</p>	
<p align="center">Continue WARM-UP Session Slide 5 to 12</p> 	
<p>Activity Details</p> <p>Following are the session deliverables:</p> <ul style="list-style-type: none"> • Appreciate the student. • Narrate the story by using hand gestures and voice modulation methods to bring in more interest in students. 	
<p align="center">Teacher Ends Slideshow</p> 	
<p align="center">TEACHER-LED ACTIVITY - 10 mins</p>	
<p align="center">Teacher Initiates Screen Share</p>	
<p align="center"><u>ACTIVITY</u></p> <ul style="list-style-type: none"> • Introduction to 1D and 2D arrays • Operations on 2D arrays 	
Teacher Action	Student Action

In the previous class, we learned about the reward matrix.
How was its structure?

Great!!

So we had the reward matrix in a tabular format. Moving on we'll be creating more such matrices to take appropriate action. So, we have to make sure we know how different operations are performed on it.

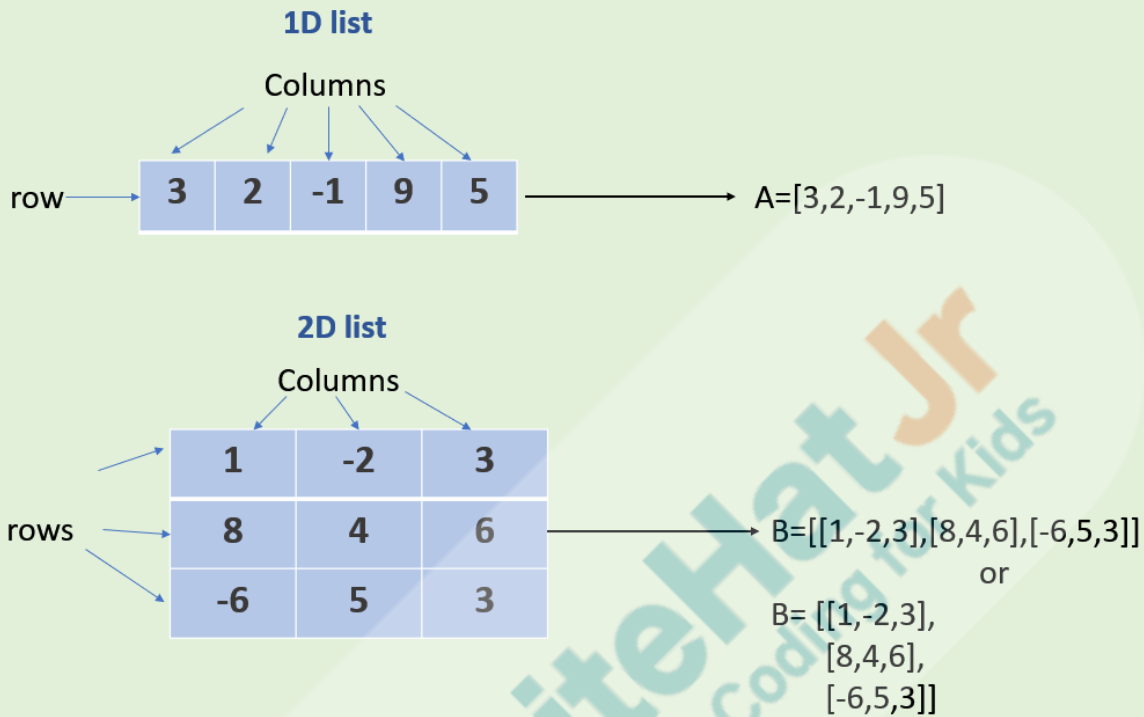
What is a matrix?

We know that is it the arrangement of numbers in tabular form. In Python, we use 2D lists or arrays to represent a matrix. A Matrix should always have a numerical value.

1D list: It is the list with one row and many columns as shown.

2D list: It is the list of lists. It can have any number of rows and columns.

ESR: Rows represented **states** and **actions** represented columns.



Let's create these lists in Google Colab and check how to perform different operations on them. Let's create 1D and 2D lists.

```
A = [3,2,-1,9,5]
print(A)
B = [[1,-2,3],[8,4,6],[-6,5,3]]
print(B)
```

[3, 2, -1, 9, 5]
[[1, -2, 3], [8, 4, 6], [-6, 5, 3]]

But do you remember we had to use NumPy arrays for different operations? What can be the reason for using arrays instead of lists?

ESR: Dedicated modules provide different dedicated methods and are responsible for fast operations.

Great!! NumPy has different methods to operate on arrays.

Thus, instead of using simple lists, we use a NumPy array. You know very well how we can create these arrays.

NumPy library provides different dedicated operations on arrays and provides faster operations on arrays. Since we are going to learn about 2D arrays today we'll call it Matrix.

[Teacher Activity 1: NumPy](#)

So let's create a NumPy array. For this,

1. Import **NumPy** as **np**.
2. Define the array using the **array()** method.
3. Print 1D array and check its type.

```
#Method to create a 1D array using NumPy
```

```
import numpy as np  
  
arr = np.array([1, 2, 3, 4, 5])  
print(arr)  
print(type(arr))
```

```
[1 2 3 4 5]  
<class 'numpy.ndarray'>
```

When we check the type of 1D array it shows **numpy.ndarray**. This stands for **N-Dimensional** array. It means that NumPy arrays can be of any dimension (rows and columns). Let's create a 2D NumPy array.

#Method-2 to create a 2D array using NumPy

```
arr_2d = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(arr_2d)
type(arr_2d)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
numpy.ndarray
```

We are using only the numbers over here. What do you think about having other data types in lists and arrays?

ESR: Varied

Yes!! In both lists and NumPy arrays, we can have elements of different data types. Let's check that as well.

#Array with elements of different data types

```
arr_2d_1 = np.array([[1,2,3],['a',5,True],[7,8,9]])
print(arr_2d_1)
type(arr_2d_1)
```

```
[['1' '2' '3']
 ['a' '5' 'True']
 ['7' '8' '9']]
numpy.ndarray
```

Although we are using different data types. They are getting converted into a string by NumPy.

Do you remember how we can access any element from a list?

ESR: Using index number.

Exactly. So it stands the same for lists and arrays. Let's check for 1D lists and arrays. We'll use the same lists and

arrays as we had created above.

```
#Accessing elements of 1D list or array
print(A)
print(A[1])
print(arr)
print(arr[2])
```

```
[3, 2, -1, 9, 5]
```

```
2
```

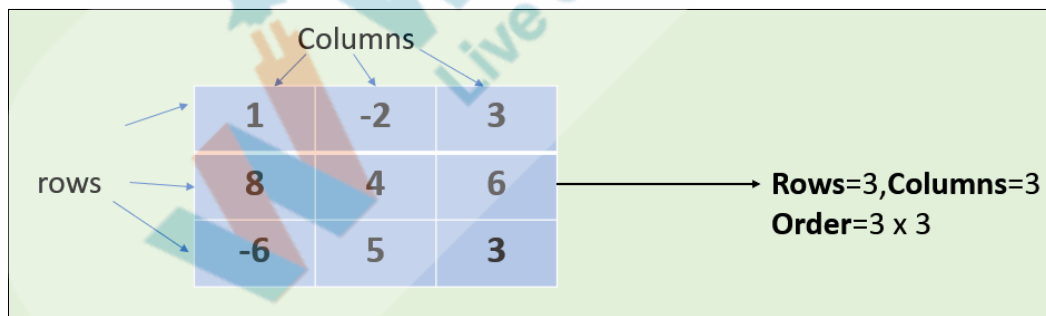
```
[1 2 3 4 5]
```

```
3
```

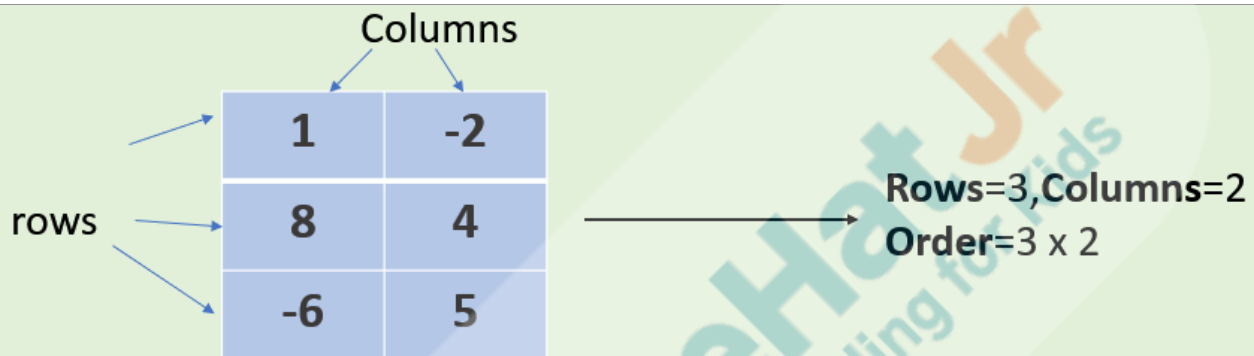
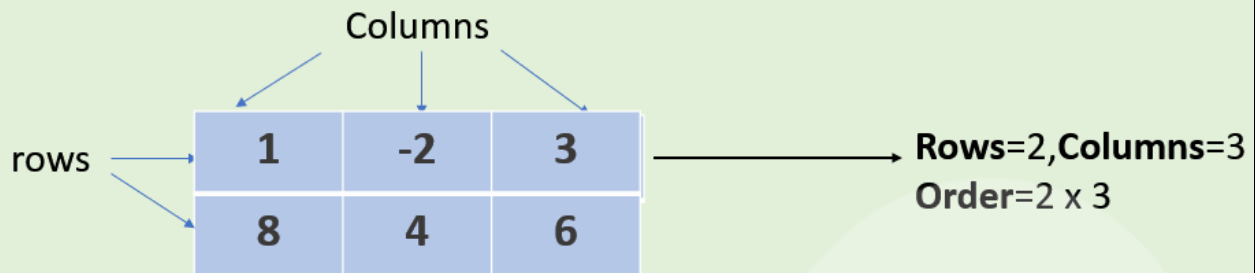
Similarly, what if we need to access an element from a 2D list?

We'll have to check the row and column number of the 2D array or matrix to access elements. The index starts from 0 onwards. If we have 3 rows and 3 columns it'll be a matrix with order 3x3.

ESR: Look for row and column numbers of the element



So, we can have a different number of rows and columns. And thus the order varies.



So let's try to access the element of the array that we have just created. There are two methods used.
One we can give rows, columns in a single **square[]** brackets or separately in two **square brackets**.

```
#accessing elements of 2D array method1
print(arr_2d)
print(arr_2d[1,2])
#accessing elements of 2D array method2
print(arr_2d[0][1])
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

```
6
2
```

Using the same methods we can update the elements of the arrays by providing row and column numbers.


```
#To update an element from 2D list/arrays
print(arr_2d)
print('-----')
# method 1
arr_2d[1,2]=-4
print(arr_2d)
print('-----')
# method 2
arr_2d[1,2]=8
print(arr_2d)
```

```
[[ 1  2  3]
 [ 4  5 10]
 [ 7  8  9]]
```

```
-----
[[ 1  2  3]
 [ 4  5 -4]
 [ 7  8  9]]
```

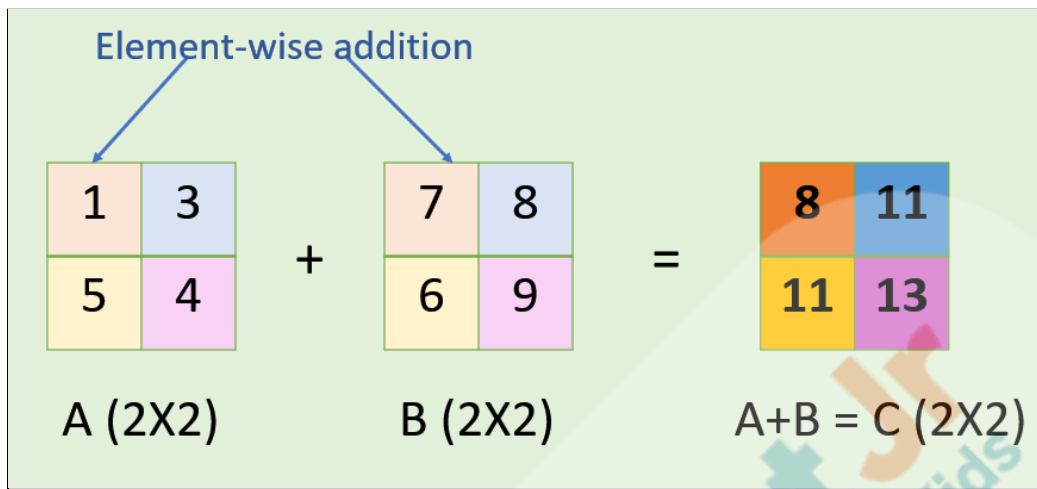
```
-----
[[1 2 3]
 [4 5 8]
 [7 8 9]]
```

This way we can access and update elements and perform different operations on them.

Matrix addition:

The addition operation is performed element-wise, thus the order of the matrices for addition and subtraction has to be the same.

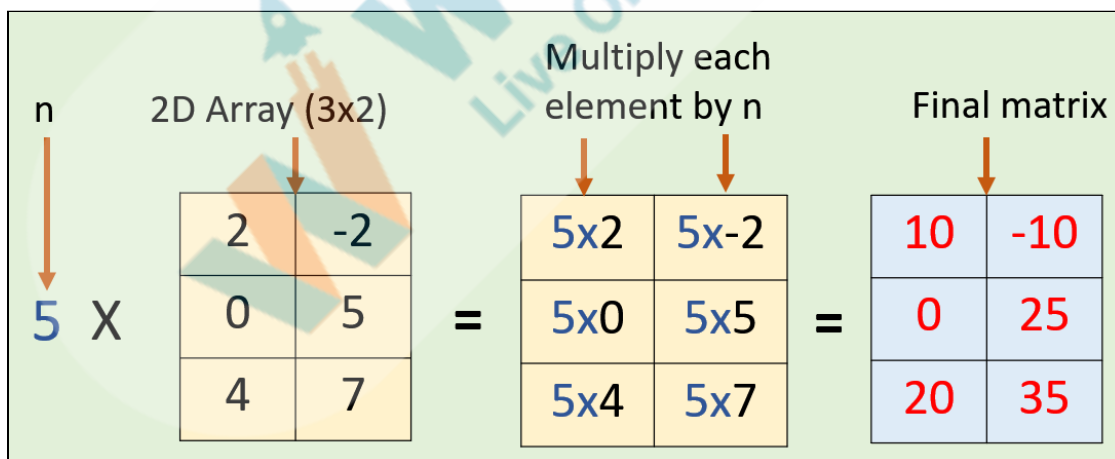
Let's add and subtract the matrices. In this method, we can use a **for** loop and perform the element-wise operation.



But the multiplication is different. There are two types of multiplication performed.



- Matrix multiplication
- Scalar multiplication

Next, whenever a matrix is multiplied by any number it is known as Scalar multiplication. Here each element is multiplied by the number.



So, you clearly understood how matrix multiplication works. You can create any two 2D matrices. We'll try to perform different operations on them.

Teacher Stops Screen Share

So now it's your turn. Please share your screen with me.	
<div style="text-align: center;">  Teacher Starts Slideshow Slide 13 to 14 Refer to speaker notes and follow the instructions on each slide. </div>	
We have one more class challenge for you. Can you solve it? Let's try. I will guide you through it.	
<div style="text-align: center;">  Teacher Ends Slideshow </div>	
STUDENT-LED ACTIVITY - 20 mins	
<ul style="list-style-type: none"> • Ask the student to press the ESC key to come back to the panel. • Guide the student to start Screen Share. • The teacher gets into Full Screen. 	
Student Initiates Screen Share	
<div style="text-align: center;"> <u>ACTIVITY</u> <ul style="list-style-type: none"> • Perform matrix multiplication using the concepts learned </div>	
Teacher Action	Student Action
So you can open Student Activity 1 and continue with the Boilerplate code. Let's try to add the matrices first, but we'll have to use the indirect method. <ol style="list-style-type: none"> 1. Create two 2x2 matrices for addition. 2. Define a 2x2 matrix to store results. 3. Now, take the number of rows and columns into two variables. 	

4. Next use a nested **for** loop to iterate through the rows and columns.
5. Use the first loop to access the rows. Now, use the second loop to access the columns in that row.
6. Next, update the resultant matrix.
7. Print the resultant matrix.

```
a=np.array([[1,3],
            [5,4]])
b=np.array([[7,8],
            [6,9]])

rows=2
cols=2
c=[[0,0],
   [0,0]]

for i in range(rows):
    for j in range(cols):
        c[i][j]=a[i][j]+b[i][j]
print(c)

[[8, 11], [11, 13]]
```

Great!!! we are able to add two matrices as the answer is correct.

Note: Cross-check the answer by taking one or two elements and multiplying and adding them.

Well done. Next comes **scalar multiplication**.

1. Create any matrix or you can use any of the previously created matrices.
1. Define a number that has to be multiplied to the matrix.
2. Define a resultant matrix to store the result. It'll be of the same size as your input matrix.

```
b=np.array([[2,-2],  
           [0,5],  
           [4,7]])      # 3x2 matrix  
  
print(b,'\n')  
n=5                      #scalar  
mul=np.zeros([3,2])
```

3. Use nested **for** loop to iterate through each element and store the result in the corresponding element of the resultant matrix.

```
b=np.array([[2,-2],  
           [0,5],  
           [4,7]])      # 3x2 matrix  
  
print(b,'\n')  
n=5                      #scalar  
mul=np.zeros([3,2])  
for i in range(3):  
    for j in range(2):  
        #print(b[i][j])  
        mul[i][j]= n* b[i][j]  
        #print(mul[i][j])
```

4. Print the resultant matrix and cross-check the result.

```
for rows in mul:
    print(rows)
```

```
[[ 2 -2]
 [ 0  5]
 [ 4  7]]
```

```
[ 10. -10.]
[  0. 25.]
[20. 35.]
```

Awesome!! We are able to get the correct results. But, to our surprise, we have a method in NumPy that can be used to perform scalar multiplication. Let's use it to check the results.

```
print(a)
e=np.multiply(a,n)
print(e)
```



```
[[1 3]
 [5 4]]
[[ 5 15]
 [25 20]]
```





So, using the **multiply()** method, we got the same type of results.

Thus, we saw how to perform operations on different types of 2D matrices. We learned to access and update matrix elements and two methods of matrix multiplication as well.

Teacher Guides Student to Stop Screen Share

WRAP-UP SESSION - 05 mins

<div>  </div> <p>Teacher Starts Slideshow Slide 15 to 20</p>	
<p>Activity details</p> <p>Following are the WRAP-UP session deliverables:</p> <ul style="list-style-type: none"> • Appreciate the student. • Revise the current class activities. • Discuss the quizzes. 	
<p>WRAP-UP QUIZ Click on In-Class Quiz</p>	
<div>  </div> <p>Continue WRAP-UP Session Slide 21 to 26</p>	
<p>Activity Details</p> <p>Following are the session deliverables:</p> <ul style="list-style-type: none"> • Explain the facts and trivia • Next class challenge • Project for the day • Additional Activity (Optional) 	
<p><u>FEEDBACK</u></p> <ul style="list-style-type: none"> • Appreciate and compliment the student for trying to learn a difficult concept. • Get to know how they are feeling after the session. • Review and check their understanding. 	
Teacher Action	Student Action
You get “hats-off” for your excellent work!	<i>Make sure you have given at least 2 hats-off during the class for:</i>

<p>In the next class, we'll be creating a Q-matrix to help the agent take appropriate action. We'll also be performing operations on the matrices to get the results.</p>	<div>Creatively Solved Activities  +10</div> <div>Great Question  +10</div> <div>Strong Concentration  +10</div>
<p align="center">PROJECT OVERVIEW DISCUSSION Refer the document below in Activity Links Sections</p>	
<p align="center">Teacher Clicks</p>	<p align="center"></p>
<p align="center">ADDITIONAL ACTIVITIES (Optional)</p>	
<p>Additional Activities:</p> <p>In matrix multiplication, we'll:</p> <ol style="list-style-type: none"> 1. Multiply the elements of each row of the first matrix by the elements of each column in the second matrix. 2. Add the products. 3. This will be the first element of the resultant matrix. 4. Repeat the procedure for each element and place the elements row-wise. 5. If matrix1 has order ixk and matrix2 has order kxj. The resultant matrix will be ixj. <p>Now since we are multiplying the elements of each row of the first matrix by the elements of each column in the second matrix, we have to make sure that they contain the</p>	

same element. So the condition to multiply two matrices is the number of rows of matrix1= number of columns of matrix2.

To perform matrix multiplication,

1. Create two matrices of 3x3 and 3x2.
2. Create a resultant matrix containing all zeros. Use the **zeros()** method of NumPy to create it. The resultant matrix should have the order 3x2.
3. Define the number of rows and columns for these matrices.

#To multiply elements of 2D list/arrays

```
a=np.array([[2,3,-4],
            [6,1,0],
            [8,9,3]]) # 3x3 matrix

b=np.array([[2,-2],
            [0,5],
            [4,7]]) # 3x2 matrix

c= np.zeros([3,2])

print(c)
rows_a=3
cols_a=rows_b=2
cols_b=3
```

4. Now, use a nested for loop to iterate through the rows and columns of matrices. We know that the first matrix has to be traversed row-wise and the second one should be traversed column-wise.
5. Add the result, after multiplying the elements.

```
#To multiply elements of 2D list/arrays

a=np.array([[2,3,-4],
            [6,1,0],
            [8,9,3]])    # 3x3 matrix

b=np.array([[2,-2],
            [0,5],
            [4,7]])    # 3x2 matrix

c= np.zeros([3,2])

print(c)
rows_a=3
cols_a=rows_b=2
cols_b=3

for i in range(3):
    for j in range(2):
        for k in range(3):
            c[i][j]=c[i][j]+a[i][k]*b[k][j]
            #print(c[i][j])
```

6. Print the result by iterating through rows of the resultant matrix. This is done to print it in the form of a matrix.

```
for rows in c:
    print(rows)
```

7. Run this code and check the answer.

```
for rows in c:  
    print(rows)
```

```
[[0. 0.]  
 [0. 0.]  
 [0. 0.]  
  
 [-12. -17.]  
 [12. -7.]  
 [28. 50.]
```

Check the answer by manually multiplying the elements of one row with one column and adding them together.

ACTIVITY LINKS		
Activity Name	Description	Links
Teacher Activity 1	NumPy	https://numpy.org/doc/
Teacher Activity 2	Reference Code	https://colab.research.google.com/drive/15nRiEfJWc772_eH7QSmbp2FcASCu52o6?usp=sharing
Teacher Reference 1	Project	https://s3-whjr-curriculum-uploads.whjr.online/81237581-b7a2-4a9f-bcff-c2b09d9769b7.pdf
Teacher Reference 2	Project Solution	https://colab.research.google.com/drive/1OR_XELVVNIgJIR6LOMpCy0i3V4Rwu6FO?usp=sharing
Teacher Reference 3	Visual-Aid	https://s3-whjr-curriculum-uploads.whjr.online/ed82ae75-c412-41ea-b3aa-11b4cba4ce62.html
Teacher Reference 4	In-Class Quiz	https://s3-whjr-curriculum-uploads.whjr.online/844a348b-6863-4d24-84a0-11bcedb54fb2.pdf
Student Activity 1	Boilerplate Code	https://colab.research.google.com/drive/1Xns32WtAKq4mBi23ZCtvARJ9t1uxcOo_?usp=sharing