



Dhirubhai Ambani Institute of Information and Communication Technology

Gandhinagar, Gujarat

ROBOT PROGRAMMING (IE416)

Mr. Tapas Kumar Maiti

MICRO PROJECT 1

DHRUVAM SHAH- 202101015

PRANALI THAKKAR- 202101271

SIDDHANT MEENA- 202101273

Google Colab Link:

<https://colab.research.google.com/drive/1EdXfJqh0y4qcJzqwq3NLIpnNlb7nyBkU?usp=sharing>

BASIC DATA TYPES

```
x = 15
print(type(x)) # Prints "<class 'int'>"
print(x)       # Prints "15"
print(x + 1)    # Addition
print(x - 1)    # Subtraction
print(x * 2)    # Multiplication
print(x ** 2)   # Exponentiation

<class 'int'>
15
16
14
30
225

x += 1
print(x) # Prints "16"
x *= 2
print(x) # Prints "32"

16
32

[3] y = 15.5
print(type(y)) # Prints "<class 'float'>"
print(y, y + 1, y * 2, y ** 2) # Prints "15.5 16.5 31.0 240.25"

<class 'float'>
15.5 16.5 31.0 240.25
```

0s completed at 6:06 PM

The screenshot shows a Google Colab notebook titled "Untitled2.ipynb". The code is as follows:

```
<class 'bool'>
15.5 16.5 31.0 240.25

[4] t = True
    f = False
    print(type(t)) # Prints "<class 'bool'>"
    print(t and f) # Logical AND(&&); prints "false"
    print(t or f)  # Logical OR(||); prints "True"
    print(not t)   # Logical NOT; prints "false"
    print(t != f)  # Logical XOR; prints "True"

<class 'bool'>
False
True
False
True

[6] hlo = 'hello' # String literals can use single/double quotes
    wld = "world"
    print(hlo)    # Prints "hello"
    print(len(hlo)) # Prints string length
    hw = hlo + ' ' + wld # String concatenation
    print(hw)     # prints "hello world"
    hw2024 = '%s %s %d' % (hlo, wld, 2024)
    print(hw2024) # prints "hello world 2024"

hello
5
hello world
hello world 2024
```

The bottom status bar shows "0s completed at 6:06 PM".

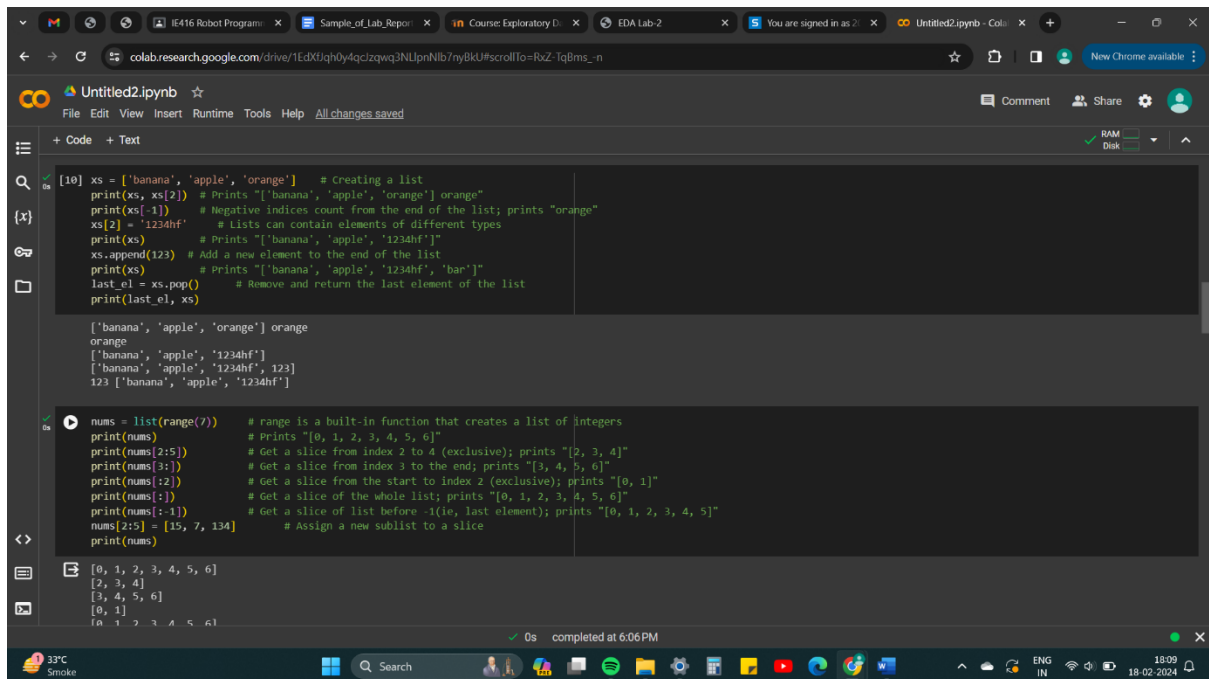
The screenshot shows the same Google Colab notebook with additional code:

```
[7] string = "hello"
    print(string.capitalize()) # Capitalize first letter
    print(string.upper())     # Convert a string to uppercase
    print(string.rjust(7))    # Right-justify a string, padding with spaces
    print(string.center(7))   # Center a string, padding with spaces
    print(string.replace('l', '(ell)')) # Replace all instances of one substring with another;
                                         # prints "he(ell)ello"
    print(string.replace('h', '(leh)'))
    print(' world '.strip()) # Strip(remove) leading and trailing whitespace; prints "world"

Hello
HELLO
hello
he(ell)(ell)o
(leh)ello
world
```

The bottom status bar shows "0s completed at 6:06 PM".

CONTAINERS: LISTS, DICTIONARIES, SETS, TUPLES



The screenshot shows a Jupyter Notebook titled 'Untitled2.ipynb' in Google Colab. The code is as follows:

```
[10] xs = ['banana', 'apple', 'orange'] # Creating a list
print(xs, xs[2]) # Prints ['banana', 'apple', 'orange'] orange
print(xs[-1]) # Negative indices count from the end of the list; prints "orange"
xs[2] = '1234hf' # Lists can contain elements of different types
print(xs) # Prints ["'banana'", 'apple', '1234hf']
xs.append(123) # Add a new element to the end of the list
print(xs) # Prints ["'banana'", 'apple', '1234hf', 'bar']
last_el = xs.pop() # Remove and return the last element of the list
print(last_el, xs)
```

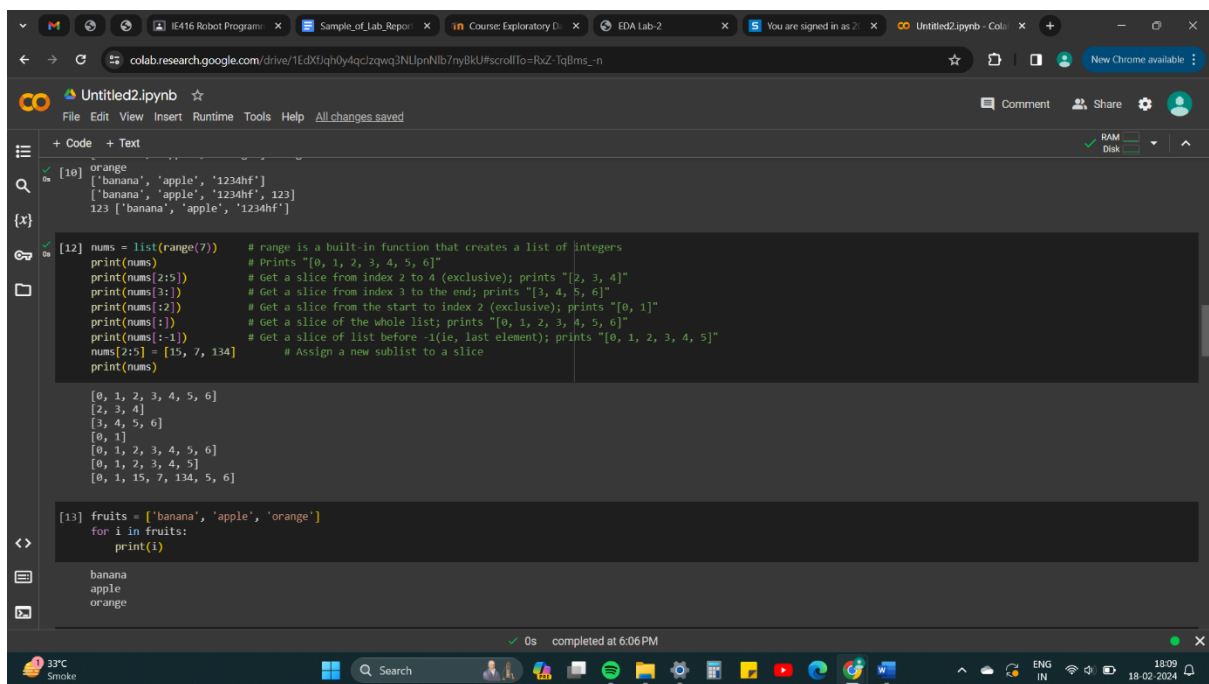
The output of the first cell is:

```
['banana', 'apple', 'orange'] orange
orange
['banana', 'apple', '1234hf']
['banana', 'apple', '1234hf', 123]
123 ['banana', 'apple', '1234hf']
```

```
[11] nums = list(range(7)) # range is a built-in function that creates a list of integers
print(nums) # Prints [0, 1, 2, 3, 4, 5, 6]
print(nums[2:5]) # Get a slice from index 2 to 4 (exclusive); prints "[2, 3, 4]"
print(nums[3:]) # Get a slice from index 3 to the end; prints "[3, 4, 5, 6]"
print(nums[:2]) # Get a slice from the start to index 2 (exclusive); prints "[0, 1]"
print(nums[:]) # Get a slice of the whole list; prints "[0, 1, 2, 3, 4, 5, 6]"
print(nums[:-1]) # Get a slice of list before -1 (ie, last element); prints "[0, 1, 2, 3, 4, 5]"
nums[2:5] = [15, 7, 134] # Assign a new sublist to a slice
print(nums)
```

The output of the second cell is:

```
[0, 1, 2, 3, 4, 5, 6]
[2, 3, 4]
[3, 4, 5, 6]
[0, 1]
[0, 1, 2, 3, 4, 5, 6]
[0, 1, 2, 3, 4, 5]
[0, 1, 15, 7, 134, 5, 6]
```



The screenshot shows a Jupyter Notebook titled 'Untitled2.ipynb' in Google Colab. The code is as follows:

```
[10] orange
['banana', 'apple', '1234hf']
['banana', 'apple', '1234hf', 123]
123 ['banana', 'apple', '1234hf']
```

```
[12] nums = list(range(7)) # range is a built-in function that creates a list of integers
print(nums) # Prints [0, 1, 2, 3, 4, 5, 6]
print(nums[2:5]) # Get a slice from index 2 to 4 (exclusive); prints "[2, 3, 4]"
print(nums[3:]) # Get a slice from index 3 to the end; prints "[3, 4, 5, 6]"
print(nums[:2]) # Get a slice from the start to index 2 (exclusive); prints "[0, 1]"
print(nums[:]) # Get a slice of the whole list; prints "[0, 1, 2, 3, 4, 5, 6]"
print(nums[:-1]) # Get a slice of list before -1 (ie, last element); prints "[0, 1, 2, 3, 4, 5]"
nums[2:5] = [15, 7, 134] # Assign a new sublist to a slice
print(nums)
```

The output of the second cell is:

```
[0, 1, 2, 3, 4, 5, 6]
[2, 3, 4]
[3, 4, 5, 6]
[0, 1]
[0, 1, 2, 3, 4, 5, 6]
[0, 1, 2, 3, 4, 5]
[0, 1, 15, 7, 134, 5, 6]
```

```
[13] fruits = ['banana', 'apple', 'orange']
for i in fruits:
    print(i)
```

The output of the third cell is:

```
banana
apple
orange
```

This screenshot shows a Google Colab notebook titled 'Untitled2.ipynb'. The code in the notebook performs the following steps:

- Cell [13]: Prints the number 1.
- Cell [14]: Creates a list `fruits` containing 'banana', 'apple', and 'orange', then appends 'mango' and prints the list.
- Cell [15]: Uses `enumerate` to iterate over `fruits`, printing the index and value for each item.
- Cell [16]: Creates a list `nums` from `range(7)`, calculates squares, and prints the resulting list.

The output of the notebook shows the list `[0, 1, 4, 9, 16, 25, 36]`. The bottom status bar indicates the notebook completed at 6:06 PM.

This screenshot shows the same Google Colab notebook with additional code:

- Cell [17]: Uses a list comprehension to create `squares` from `nums` and prints it.
- Cell [19]: Filters `nums` to create `even_squares` and `odd_squares`, then prints both.
- Cell [27]: Creates a dictionary `d` with keys 'mango', 'potato', and 'lotus', and prints entries for 'mango' and 'lotus'.
- Cell [28]: Attempts to print `d['monkey']`, which results in a `KeyError: 'monkey' not a key of d`.

The output shows the filtered lists `[0, 4, 16, 36]` and `[1, 9, 25]`, and the dictionary output: `fruit`, `False`, and `flower`. The bottom status bar shows an error at 6:06 PM.

The screenshot shows a Google Colab notebook titled 'Untitled2.ipynb'. The code in the first cell defines a dictionary `d` with keys 'fruit', 'False', and 'flower'. The second cell attempts to print `d['monkey']`, which results in a `KeyError: 'monkey'`. The third cell demonstrates dictionary methods: `d.get('monkey', 'N/A')` returns 'N/A', `d.get('potato', 'N/A')` returns 'N/A', `del d['potato']` removes the key, and `d.get('potato', 'N/A')` returns 'N/A'. The fourth cell iterates over the dictionary, printing each key-value pair: 'mango is a fruit' and 'lotus is a flower'. The notebook interface includes a menu bar, a toolbar, and a status bar at the bottom showing the temperature as 33°C and the time as 18:10 on 18-02-2024.

```
[27] d = {'fruit': 'mango', 'False': 'lotus', 'flower': 'lotus'}

[28] print(d['monkey']) # KeyError: 'monkey' not a key of d

KeyError                                Traceback (most recent call last)
<ipython-input-28-78fc9745d9cf> in <cell line: 1>()
----> 1 print(d['monkey']) # KeyError: 'monkey' not a key of d

KeyError: 'monkey'

[29] print(d.get('monkey', 'N/A')) # Get an element with a default; prints "N/A"
print(d.get('potato', 'N/A')) # Get an element with a default; prints "vegetable"
del d['potato'] # Remove an element from a dictionary
print(d.get('potato', 'N/A')) # "potato" is no longer a key; prints "N/A"

N/A
vegetable
N/A

[32] for i in d:
    info = d[i]
    print('%s is a %s' % (i, info))

mango is a fruit
lotus is a flower
```

The screenshot shows a Google Colab notebook titled 'Untitled2.ipynb'. The code in the first cell iterates over the dictionary `d` and prints each key-value pair: 'mango is a fruit' and 'lotus is a flower'. The second cell creates a list `nums` from `range(7)` and a dictionary `even_square_dict` with values `x ** 2` for `x` in `nums` where `x % 2 == 0`. The third cell creates a set `fruits` with elements 'mango', 'orange', and 'papaya' and checks if 'papaya' is in the set, which returns `True`. The fourth cell checks if 'melon' is in the set, which returns `False`. The fifth cell adds 'melon' to the set. The sixth cell prints 'melon' in the set and the length of the set, which is 4. The notebook interface includes a menu bar, a toolbar, and a status bar at the bottom showing the temperature as 33°C and the time as 18:10 on 18-02-2024.

```
[33] for i,info in d.items():
    print('%s is a %s' % (i, info))

mango is a fruit
lotus is a flower

[34] nums = list(range(7))
even_square_dict = {x: x ** 2 for x in nums if x % 2 == 0}
print(even_square_dict) # Prints "{0: 0, 2: 4, 4: 16, 6: 36}"

{0: 0, 2: 4, 4: 16, 6: 36}

[35] fruits = {'mango', 'orange', 'papaya'}
print('papaya' in fruits) # Check if an element is in a set; prints "True"

True

[36] print('melon' in fruits) # prints "False"

False

[37] fruits.add('melon') # Add an element to a set

[39] print('melon' in fruits) # Prints "True"
print(len(fruits)) # Number of elements in a set

True
4
```

The screenshot shows a Google Colab notebook titled 'Untitled2.ipynb'. The code is as follows:

```
[40] fruits.add('orange')      # Adding an element that is already in the set does nothing
     print(len(fruits))      # Prints "4"
     fruits.remove('orange')  # Remove an element from a set
     print(len(fruits))

4
3

[41] from math import sqrt
     nums_sqrt = {int(sqrt(x)) for x in range(50)}
     print(nums_sqrt)        # Prints "{0, 1, 2, 3, 4, 5, 6, 7}"

{0, 1, 2, 3, 4, 5, 6, 7}

d = {(x, x + 1): x for x in range(10)} # Create a dictionary with tuple keys
t = (8, 9)                             # Create a tuple
print(type(t))                         # Prints "<class 'tuple'>"
print(d[t])                           # Prints "8"
print(d[(1, 2)])                      # Prints "1"

<class 'tuple'>
8
1

[ ]
```

The bottom status bar shows '0s completed at 6:06 PM'.

FUNCTIONS

The screenshot shows a Google Colab notebook titled 'Untitled2.ipynb'. The code is as follows:

```
[44] def sign(x):              # sign function that returns sign of a number
     if x > 0:
         return 'positive'
     elif x < 0:
         return 'negative'
     else:
         return 'zero'

     for x in [-15, 0, 13, -47, 19]:
         print(sign(x))

negative
zero
positive
negative
positive

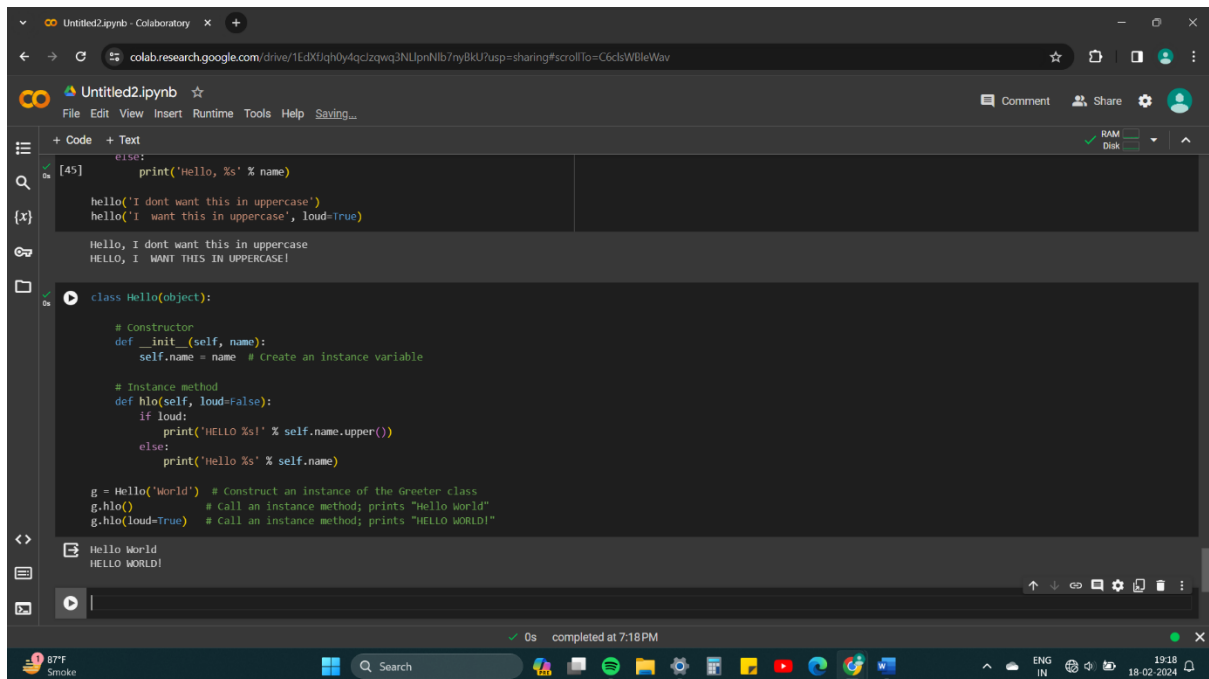
def hello(name, loud=False):    # function which prints uppercase if loud is true
     if loud:
         print('HELLO, %s!' % name.upper())
     else:
         print('Hello, %s' % name)

hello('I dont want this in uppercase')
hello('I want this in uppercase', loud=True)

Hello, I dont want this in uppercase
HELLO, I WANT THIS IN UPPERCASE!
```

The bottom status bar shows '0s completed at 6:22 PM'.

CLASSES



The screenshot shows a Google Colab notebook titled 'Untitled2.ipynb'. The code defines a class 'Hello' that inherits from 'object'. It includes a constructor '__init__' that takes a 'name' parameter and creates an instance variable 'self.name'. It also has an instance method 'hlo' that takes a 'loud' parameter (defaulting to False) and prints 'HELLO' in either uppercase or lowercase based on the 'loud' flag. The code is executed, and the output shows 'Hello, I dont want this in uppercase' and 'HELLO, I WANT THIS IN UPPERCASE!'. Below the code, the output of the class execution is shown: 'Hello World' and 'HELLO WORLD!'.

```
[45] print('Hello, %s' % name)

hello('I dont want this in uppercase')
hello('I want this in uppercase', loud=True)

Hello, I dont want this in uppercase
HELLO, I WANT THIS IN UPPERCASE!

class Hello(object):

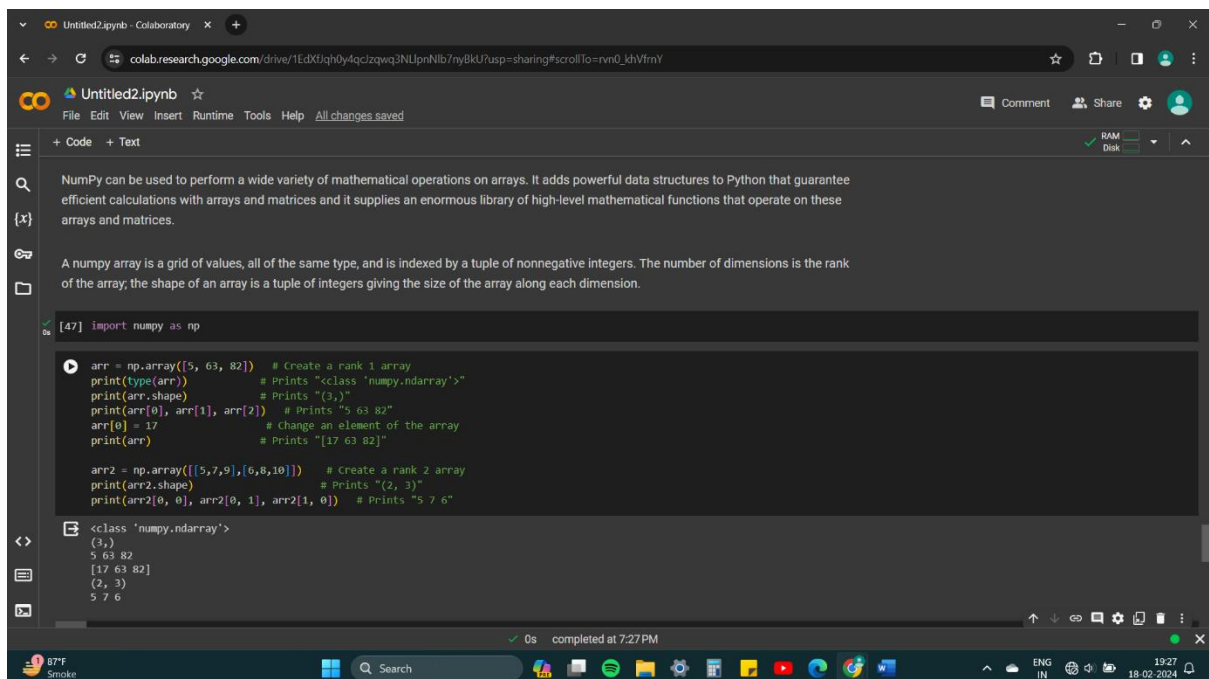
    # Constructor
    def __init__(self, name):
        self.name = name # create an instance variable

    # Instance method
    def hlo(self, loud=False):
        if loud:
            print('HELLO %s!' % self.name.upper())
        else:
            print('hello %s' % self.name)

g = Hello('World') # Construct an instance of the Greeter class
g.hlo() # Call an instance method; prints "Hello World"
g.hlo(loud=True) # Call an instance method; prints "HELLO WORLD!"

Hello World
HELLO WORLD!
```

NUMPY



The screenshot shows a Google Colab notebook titled 'Untitled2.ipynb'. The code imports 'numpy' as 'np'. It then creates a 1D array 'arr' with values [5, 63, 82] and prints its type, shape, and elements. It also modifies the first element to 17. Next, it creates a 2D array 'arr2' with values [[5, 7, 9], [6, 8, 10]] and prints its shape and elements. The output shows the class 'numpy.ndarray' and the arrays' shapes and values.

NumPy can be used to perform a wide variety of mathematical operations on arrays. It adds powerful data structures to Python that guarantee efficient calculations with arrays and matrices and it supplies an enormous library of high-level mathematical functions that operate on these arrays and matrices.

A numpy array is a grid of values, all of the same type, and is indexed by a tuple of nonnegative integers. The number of dimensions is the rank of the array; the shape of an array is a tuple of integers giving the size of the array along each dimension.

```
[47] import numpy as np

arr = np.array([5, 63, 82]) # Create a rank 1 array
print(type(arr)) # Prints "<class 'numpy.ndarray'>"
print(arr.shape) # Prints "(3,)"
print(arr[0], arr[1], arr[2]) # Prints "5 63 82"
arr[0] = 17 # change an element of the array
print(arr) # Prints "[17 63 82]"

arr2 = np.array([[5, 7, 9], [6, 8, 10]]) # Create a rank 2 array
print(arr2.shape) # Prints "(2, 3)"
print(arr2[0, 0], arr2[0, 1], arr2[1, 0]) # Prints "5 7 6"

<class 'numpy.ndarray'>
(3,)
5 63 82
[17 63 82]
(2, 3)
5 7 6
```

Untitled2.ipynb · Colaboratory

colab.research.google.com/drive/1EdXlqh0y4qcizqwq3NLJpnNlb7nyBkU?usp=sharing#scrollTo=rvm0_khVfmY

Untitled2.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
#methods for creation of arrays
arr1 = np.zeros((2,2)) # Create an array of all zeros
print(arr1)           # Prints "[[ 0.  0.]
                        #        [ 0.  0.]]"

arr2 = np.ones((2,2))  # Create an array of all ones
print(arr2)           # Prints "[[ 1.  1.]
                        #        [ 1.  1.]]"

arr3 = np.full((2,2), 15) # Create a constant array with all entries 15
print(arr3)           # Prints "[[ 15. 15.]
                        #        [ 15. 15.]]"

arr4 = np.eye(2)       # Create a 2x2 identity matrix
print(arr4)           # Prints "[[ 1.  0.]
                        #        [ 0.  1.]]"

arr5 = np.random.random((2,2)) # Create an array filled with random values
print(arr5)
```

```
[[0. 0.]
 [0. 0.]
 [1. 1.]
 [1. 1.]
 [15 15]
 [15 15]
 [1. 0.]
 [0. 1.]]
[[0.98308973 0.14467126]
 [0.57878895 0.23813458]]
```

0s completed at 7:27 PM

87°F Smoke

Search

19:27 18-02-2024

Untitled2.ipynb · Colaboratory

colab.research.google.com/drive/1EdXlqh0y4qcizqwq3NLJpnNlb7nyBkU?usp=sharing#scrollTo=D6xHh51qNru

Untitled2.ipynb

File Edit View Insert Runtime Tools Help

+ Code + Text

```
[[1. 1.]
 [15 15]
 [15 15]
 [1. 0.]
 [0. 1.]]
[[0.98308973 0.14467126]
 [0.57878895 0.23813458]]
```

```
[51] # Create a rank 2 array with shape (3, 4)
# [[ 1  2  3  4]
#   [ 5  6  7  8]
#   [ 9 10 11 12]]
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
print(a)
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

```
[54] # Use slicing to pull out the subarray consisting of the first 2 rows
# and columns 1 and 2; b is the following array of shape (2, 2):
# [[2 3]
#  [6 7]]
b = a[:2, 1:3]
print(b)
```

```
[[2 3]
 [6 7]]
```

```
[57] # modifying the slice will modify the original array.
print(a[:2, 1:3]) # Prints "[[2 3]
                  #        [6 7]]"
```

0s completed at 7:34 PM

87°F Smoke

Search

19:35 18-02-2024

Untitled2.ipynb · Colaboratory

colab.research.google.com/drive/1EdXlqh0y4qcjzqwq3NLJpnNlb7ny8kU?usp=sharing#scrollTo=D6xHh51qINru

Untitled2.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

RAM Disk

[54]

```
b = a[2, 1:5]
print(b)
```

```
[[2 3]
 [6 7]]
```

[57]

```
# modifying the slice will modify the original array.
print(a[0, 1]) # Prints "2"
b[0, 0] = 50   # b[0, 0] is the same piece of data as a[0, 1]
print(a[0, 1]) # Prints "50"
print(a)
```

```
77
50
[[ 1 50  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

[58]

```
row_r1 = a[1, :] # Rank 1 view of the second row of a
row_r2 = a[1:2, :] # Rank 2 view of the second row of a
print(row_r1, row_r1.shape) # Prints "[5 6 7 8] (4,)"
print(row_r2, row_r2.shape) # Prints "[[5 6 7 8]] (1, 4)"
```

```
[5 6 7 8] (4,)
[[5 6 7 8]] (1, 4)
```

Two ways of accessing the data in the middle row of the array. Mixing integer indexing with slices yields an array of lower rank, while using only slices yields an array of the same rank as the original array:

0s completed at 7:34PM

87°F Smoke

Search

ENG IN 19:35 18-02-2024

Untitled2.ipynb · Colaboratory

colab.research.google.com/drive/1EdXlqh0y4qcjzqwq3NLJpnNlb7ny8kU?usp=sharing#scrollTo=77HDAv7bliem

Untitled2.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

RAM Disk

[58]

```
print(row_r1, row_r1.shape) # Prints "[5 6 7 8] (4,)"
print(row_r2, row_r2.shape) # Prints "[[5 6 7 8]] (1, 4)"
```

```
[5 6 7 8] (4,)
[[5 6 7 8]] (1, 4)
```

Two ways of accessing the data in the middle row of the array. Mixing integer indexing with slices yields an array of lower rank, while using only slices yields an array of the same rank as the original array:

[59]

```
col_r1 = a[:, 1]
col_r2 = a[:, 1:2]
print(col_r1, col_r1.shape) # Prints "[ 50  6 10] (3,)"
print(col_r2, col_r2.shape) # Prints "[[ 50]
                                [  6]
                                [10]] (3, 1)"
```

```
[50  6 10] (3,)
[[50]
 [ 6]
 [10]] (3, 1)
```

When you index into numpy arrays using slicing, the resulting array view will always be a subarray of the original array. In contrast, integer array indexing allows you to construct arbitrary arrays using the data from another array.

[60]

```
a = np.array([[1,2], [3, 4], [5, 6]])
```

```
# An example of integer array indexing.
# The returned array will have shape (2,) and
print(a[[0, 1, 2], [0, 1, 0]]) # Prints "[1 4 5]" ie, (0,0), (1,1), (2,0)
```

0s completed at 7:49PM

84°F Smoke

Search

ENG IN 19:49 18-02-2024

Untitled2.ipynb · Colaboratory

colab.research.google.com/drive/1EdXlqh0y4qczqwq3NLpnNlb7ny8kU?usp=sharing#scrollTo=77HDav/bleim

Untitled2.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

[10]] (3, 1)

When you index into numpy arrays using slicing, the resulting array view will always be a subarray of the original array. In contrast, integer array indexing allows you to construct arbitrary arrays using the data from another array.

```
[50] a = np.array([[1,2], [3, 4], [5, 6]])

# An example of integer array indexing.
# The returned array will have shape (3,) and
print(a[[0, 1, 2], [0, 1, 0]]) # Prints "[1 4 5]" 1e, (0,0), (1,1), (2,0)

print(np.array([a[0, 0], a[1, 1], a[2, 0]])) # Prints "[1 4 5]"

print(a[[0, 0], [1, 1]]) # Prints "[2 2]"

print(np.array([a[0, 1], a[0, 1]])) # Prints "[2 2]"

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

0s completed at 7:49 PM

Untitled2.ipynb · Colaboratory

colab.research.google.com/drive/1EdXlqh0y4qczqwq3NLpnNlb7ny8kU?usp=sharing#scrollTo=KqB9Pkf3pODC

Untitled2.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
[60] a = np.array([ [10, 11, 12], [7,8,9], [4,5,6], [1,2,3] ])

# Create an array of indices
b = np.array([0, 2, 0, 1])

# Select one element from each row of a using the indices in b
print(a[np.arange(4), b]) # Prints "[10 9 4 2]"

# Mutate one element(+10) from each row of 'a' using the indices in b
a[np.arange(4), b] += 10

print(a)

[10 9 4 2]
[[20 11 12]
 [ 7 8 19]
 [14 5 6]
 [ 1 12 3]]

[63] a = np.array([[1,2], [3, 4], [5, 6]])

bool_idx = (a >= 4)

print(bool_idx) # Prints "[[False False]
               # [False True]
               # [ True  True]]"

# consisting of the elements of a corresponding to the True values of bool_idx
print(a[bool_idx]) # Prints "[4 5 6]"

print(a[a == 4]) # Prints "[1 4 5 6]"
```

0s completed at 8:05 PM

```
Untitled2.ipynb - Colaboratory
colab.research.google.com/drive/1EdXlqh0y4qcjzqwq3NLpnNlb7ny8kU?usp=sharing#scrollTo=KqB9Pkf3pODC

+ Code + Text
[61] [10 9 4 2]
[[20 11 12]
 [ 7 8 19]
 [14 5 6]
 [ 1 12 3]]

{x}

[63] a = np.array([[1,2], [3, 4], [5, 6]])

bool_idx = (a >= 4)

print(bool_idx) # Prints "[[False False]
               #       [False True]
               #       [ True  True]]"

# consisting of the elements of a corresponding to the True values of bool_idx
print(a[bool_idx]) # Prints "[4 5 6]"

print(a[a >= 4]) # Prints "[4 5 6]"

[[False False]
 [False  True]
 [ True  True]]
[4 5 6]
[4 5 6]

[65] x = np.array([1, 2])
print(x.dtype) # Prints "int64"

x = np.array([1.0, 2.0])
print(x.dtype) # Prints "float64"

0s completed at 8:05PM
84°F Smoke
```

```
Untitled2.ipynb - Colaboratory
colab.research.google.com/drive/1EdXlqh0y4qcjzqwq3NLpnNlb7ny8kU?usp=sharing#scrollTo=KqB9Pkf3pODC

+ Code + Text
print(a[bool_idx]) # Prints "[4 5 6]"

[63] print(a[a >= 4]) # Prints "[4 5 6]"

[[False False]
 [False  True]
 [ True  True]]
[4 5 6]
[4 5 6]

[65] x = np.array([1, 2])
print(x.dtype) # Prints "int64"

x = np.array([1.0, 2.0])
print(x.dtype) # Prints "float64"

x = np.array([1, 2], dtype=np.int64) # Force a particular datatype
print(x.dtype)

a = np.array([[1,2], [3, 4], [5, 6]])
print(a.dtype)

int64
float64
int64
int64

[67] x = np.array([[1.73,2.2],[3.5,4.71]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)

# Elementwise sum
print(x + y)
```

```
Untitled2.ipynb - Colaboratory x +
colab.research.google.com/drive/1EdXlqh0y4qcjzqwq3NLJpnNlb7ny8kU?usp=sharing#scrollTo=KqB9Pkf3pODC

Untitled2.ipynb ☆
File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text
[67] x = np.array([[1.73,2.2],[3.5,4.71]], dtype=np.float64)
     y = np.array([[5,6],[7,8]], dtype=np.float64)

     # Elementwise sum
     print(x + y)
     print(np.add(x, y))

     # Elementwise difference
     print(x - y)
     print(np.subtract(x, y))

     # Elementwise product
     print(x * y)
     print(np.multiply(x, y))

     # Elementwise division
     print(x / y)
     print(np.divide(x, y))

     # Elementwise square root
     print(np.sqrt(x))

[[ 6.73  8.2 ]
 [10.5 12.71]]
[[ 6.73  8.2 ]
 [10.5 12.71]]
[[ -3.27 -3.8 ]
 [-3.5  -3.29]]
[[ -3.27 -3.8 ]
 [-3.5  -3.29]]
[[ 8.65 13.2 ]
 [24.5 37.68]]
[[ 6.73  8.2 ]
 [10.5 12.71]]
[[ 6.73  8.2 ]
 [10.5 12.71]]
[[ -3.27 -3.8 ]
 [-3.5  -3.29]]
[[ -3.27 -3.8 ]
 [-3.5  -3.29]]
[[ 8.65 13.2 ]
 [24.5 37.68]]

0s completed at 8:05PM
```

```
Untitled2.ipynb - Colaboratory x +
colab.research.google.com/drive/1EdXlqh0y4qcjzqwq3NLJpnNlb7ny8kU?usp=sharing#scrollTo=KqB9Pkf3pODC

Untitled2.ipynb ☆
File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text
[67] # Elementwise product
     print(x * y)
     print(np.multiply(x, y))

     # Elementwise division
     print(x / y)
     print(np.divide(x, y))

     # Elementwise square root
     print(np.sqrt(x))

[[ 6.73  8.2 ]
 [10.5 12.71]]
[[ 6.73  8.2 ]
 [10.5 12.71]]
[[ -3.27 -3.8 ]
 [-3.5  -3.29]]
[[ -3.27 -3.8 ]
 [-3.5  -3.29]]
[[ 8.65 13.2 ]
 [24.5 37.68]]
[[ 6.73  8.2 ]
 [10.5 12.71]]
[[ 6.73  8.2 ]
 [10.5 12.71]]
[[ -3.27 -3.8 ]
 [-3.5  -3.29]]
[[ -3.27 -3.8 ]
 [-3.5  -3.29]]
[[ 8.65 13.2 ]
 [24.5 37.68]]
[[ 0.346  0.36666667]
 [0.5  0.58875  ]]
[[ 0.346  0.36666667]
 [0.5  0.58875  ]]
[[ 1.31529464 1.4832397 ]
 [1.87662869 2.17025344]]

[68] v = np.array([9,10])
     w = np.array([11, 12])

0s completed at 8:05PM
```

Untitled2.ipynb · Colaboratory

colab.research.google.com/drive/1EdXlqh0y4qcjzqwq3NLpnNlb7ny8kU?usp=sharing#scrollTo=KqB9Pkf3pODC

Untitled2.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
[68] v = np.array([9,10])
w = np.array([11, 12])

# Inner product of vectors
print(v.dot(w))
print(np.dot(v, w))

# Matrix / vector product; produce a rank 1 array
print(x.dot(v))
print(np.dot(x, v))

# Matrix / matrix product; produce a rank 2 array
print(x.dot(y))
print(np.dot(x, y))

219
219
[[37.57 78.6 ]
 [37.57 78.6 ]
 [[24.05 27.98]
 [50.47 58.68]]
 [[24.05 27.98]
 [50.47 58.68]]

[69] print(np.sum(x)) # Compute sum of all elements
print(np.sum(x, axis=0)) # Compute sum of each column
print(np.sum(x, axis=1)) # Compute sum of each row

12.14
[5.23 6.91]
[3.93 8.21]
```

0s completed at 8:05PM

84°F Smoke

Search

ENG IN

20:06 18-02-2024

Untitled2.ipynb · Colaboratory

colab.research.google.com/drive/1EdXlqh0y4qcjzqwq3NLpnNlb7ny8kU?usp=sharing#scrollTo=KqB9Pkf3pODC

Untitled2.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```
[69] print(np.sum(x)) # Compute sum of all elements
print(np.sum(x, axis=0)) # Compute sum of each column
print(np.sum(x, axis=1)) # Compute sum of each row

12.14
[5.23 6.91]
[3.93 8.21]

[70] print(y)

print(y.T) #transpose of a matrix

#taking the transpose of a rank 1 array does nothing
v = np.array([1,2,3])
print(v)
print(v.T)

[[5. 6.]
 [7. 8.]
 [[5. 7.]
 [6. 8.]
 [1 2 3]
 [1 2 3]
```

0s completed at 8:05PM

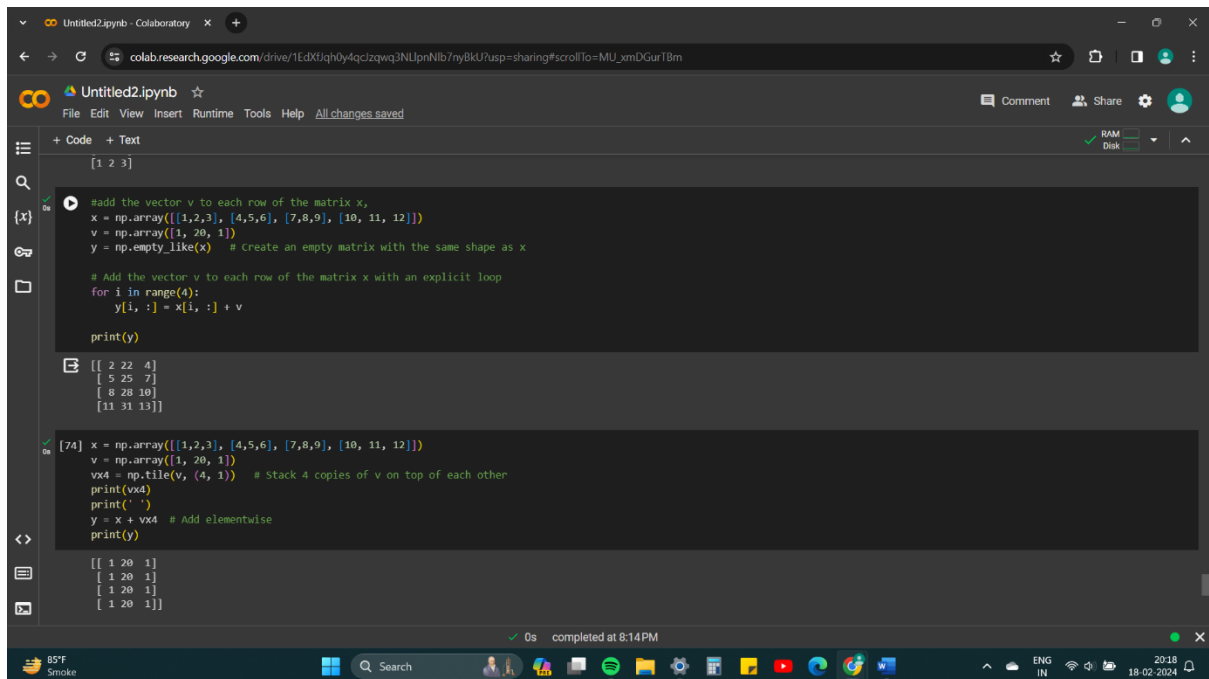
84°F Smoke

Search

ENG IN

20:06 18-02-2024

BROADCASTING



The screenshot shows a Google Colab notebook titled 'Untitled2.ipynb'. The code defines a matrix `x` of shape (4, 3) and a vector `v` of shape (3,). It then creates an empty matrix `y` of the same shape as `x`. A loop is used to add the vector `v` to each row of `x` and store the result in `y`. The output shows the resulting matrix `y`.

```
[1 2 3]

In [74]: #add the vector v to each row of the matrix x,
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 20, 1])
y = np.empty_like(x) # Create an empty matrix with the same shape as x

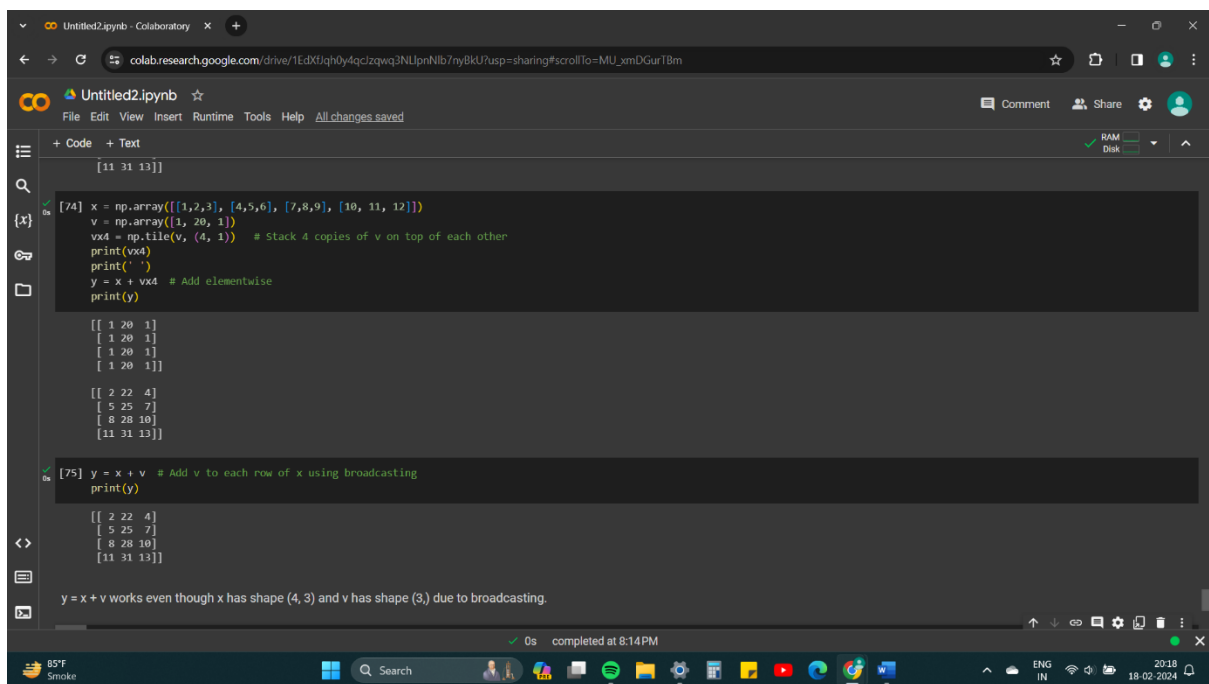
# Add the vector v to each row of the matrix x with an explicit loop
for i in range(4):
    y[i, :] = x[i, :] + v

print(y)

Out[74]: [[ 2 22  4]
 [ 5 25  7]
 [ 8 28 10]
 [11 31 13]]

In [74]: x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 20, 1])
vx4 = np.tile(v, (4, 1)) # Stack 4 copies of v on top of each other
print(vx4)
print(' ')
y = x + vx4 # Add elementwise
print(y)

Out[74]: [[ 1 20  1]
 [ 1 20  1]
 [ 1 20  1]
 [ 1 20  1]]
```



The screenshot shows the same Google Colab notebook, but with a different code block. It defines the same matrix `x` and vector `v`. It then uses `np.tile` to create a matrix `vx4` of shape (4, 3) by stacking 4 copies of `v`. The matrix `y` is then calculated as `y = x + vx4`, demonstrating broadcasting. The output shows the resulting matrix `y`.

```
[11 31 13]]

In [74]: x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 20, 1])
vx4 = np.tile(v, (4, 1)) # Stack 4 copies of v on top of each other
print(vx4)
print(' ')
y = x + vx4 # Add elementwise
print(y)

Out[74]: [[ 1 20  1]
 [ 1 20  1]
 [ 1 20  1]
 [ 1 20  1]]

[[ 2 22  4]
 [ 5 25  7]
 [ 8 28 10]
 [11 31 13]]

In [75]: y = x + v # Add v to each row of x using broadcasting
print(y)

Out[75]: [[ 2 22  4]
 [ 5 25  7]
 [ 8 28 10]
 [11 31 13]]

y = x + v works even though x has shape (4, 3) and v has shape (3,) due to broadcasting.
```

Untitled2.ipynb · Colaboratory

colab.research.google.com/drive/1EdXlqh0y4qc2zqwq3NLpnNlb7my8KU?usp=sharing#scrollTo=Bo19udSjtwZD

Untitled2.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

✓

[76]

```
v = np.array([15,30,45]) # v has shape (3,)
w = np.array([2,3])      # w has shape (2,)
# To compute an outer product, we first reshape v to be a column
# vector of shape (3, 1); we can then broadcast it against w to yield
print(np.reshape(v, (3, 1)) * w)
```

```
[[ 30 45]
 [ 60 90]
 [ 90 135]]
```

✓

```
# Multiply a matrix by a constant:
# numpy treats scalars as arrays of shape ();
print(x * 2)
```

```
[[ 2  4  6]
 [ 8 10 12]
 [14 16 18]
 [20 22 24]]
```

0s completed at 8:25 PM

85°F Smoke

Search

ENG IN

20:25 18-02-2024