L11 Arrays [1D]

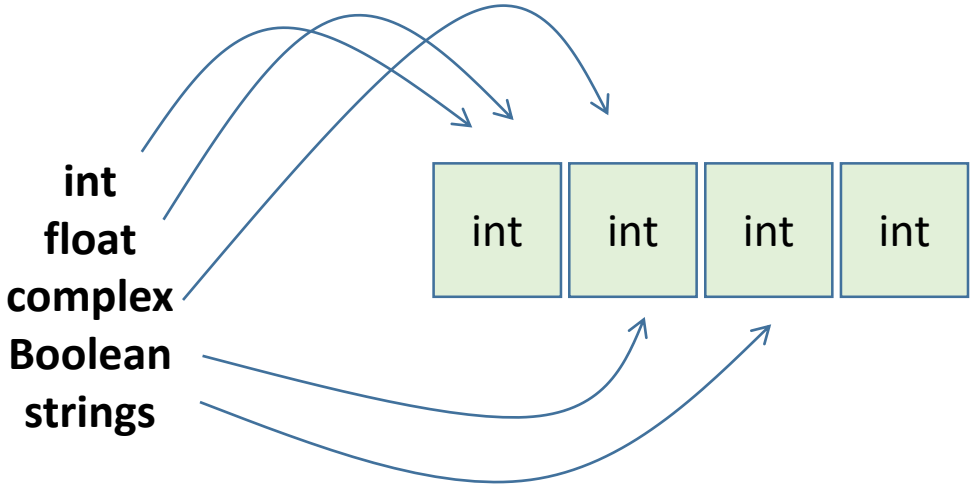
October 2019

Dr. Marco A Arocha

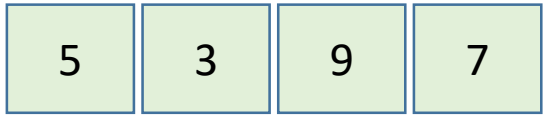
https://docs.python.org/2.5/lib/typesseq.html

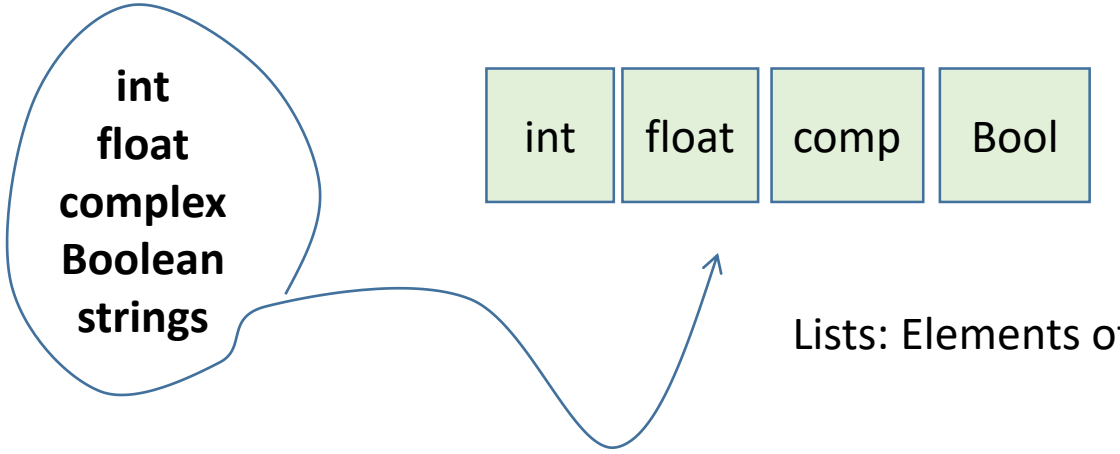
Arrays

An array is a data structure, which can store a collection of elements of the same data type.

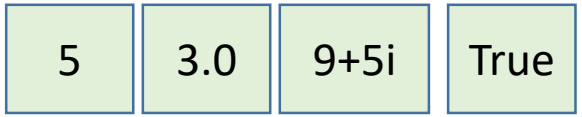


Array





List



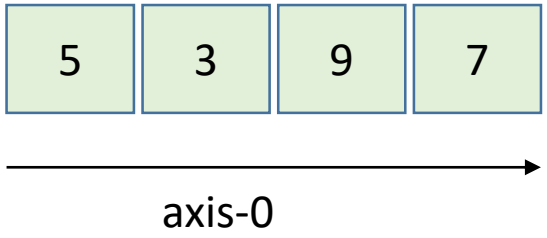
Lists: Elements of the same data type or a “salad bar”

Comparison List vs Arrays

	List	Array
Initialize	L=[1,2,3,4,5]	A=numpy.array([1,2,3,4,5])
		A=numpy.array(L)
print	print(L)	print(A)
element's data type	different data types elements allowed	elements must be of the same data type
native vs module	native sequence (no imports)	import numpy # module
math	limited scalar mathematics with each element	linear algebra and matrix math
function	list(range(start, stop, step))	numpy.arange(start, stop, step)

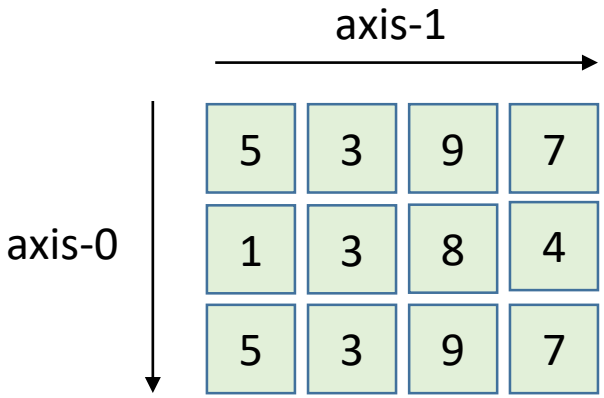
Arrays: 1D, 2D, 3D and so on

1D array



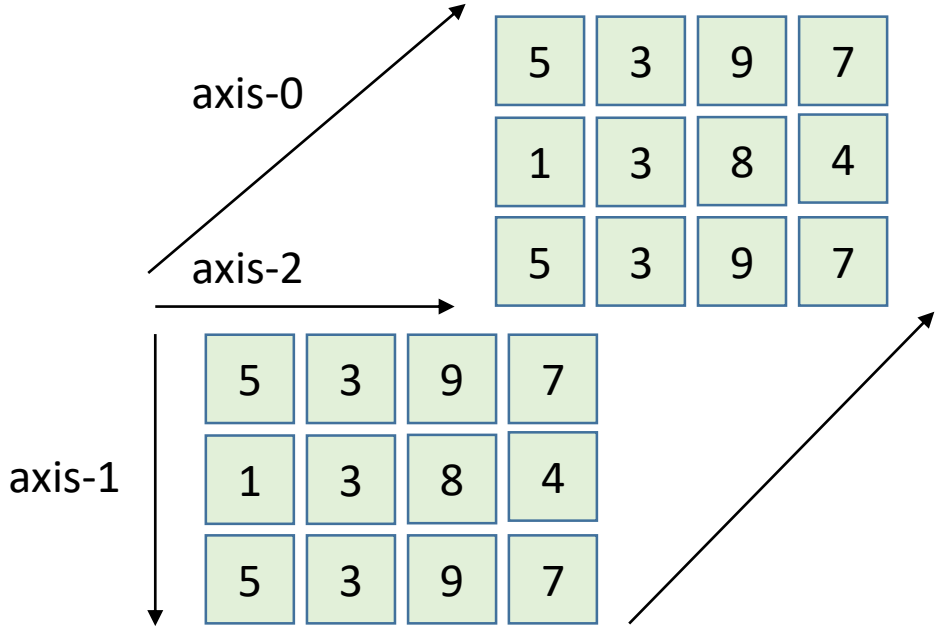
shape(4,)
shape(axis-0)
shape(row)

2D array



shape(3,4)
shape(axis-0,axis-1)
shape(row,col)

3D array



shape(2,3,4)
shape(axis0,axis1,axis2)
shape(page,row,col)

https://www.oreilly.com/library/view/elegant-scipy/9781491922927/ch01.html
https://www.sharpsightlabs.com/blog/numpy-axes-explained/

Arrays

- **Creating numpy arrays**
  - Several different ways to initialize new numpy arrays, for example,
    - ✓a list or tuples
    - ✓using functions dedicated to arrays, such as **arange**, **linspace**, etc.
    - ✓reading data from files
- Import numpy module as:
    - ✓import numpy as np (see slide 9 of L#7 to more details)

A more comprehensive treatment is found in the excellent *NumPy Tutorial*, *NumPy User Guide*, *NumPy Reference*, *Guide to NumPy*, and *NumPy for Matlab Users*, all found at scipy.org.

Creating 1D Arrays

FILE: createArray1D.py

5	3	9	7

list

```
arr1=np.array([5,3,9,7],float)
```

Not Included

```
# start,stop,step
```

1	3	5	7	9

```
arr2=np.arange(1,10,2,dtype=int)
```

```
# start,stop,points
```

1.	1.5	2.	2.5	3.	3.5	4.	4.5	5.

```
arr3=np.linspace(1,5,9)
```

included

tuple

1	1	1	1	1

```
arr4=np.ones((5,),dtype=int)
```

tuple

0.	0.	0.	0.	0.

```
arr5=np.zeros((5,),dtype=float)
```

7.	7.	7.	7.	7.

```
arr5.fill(7)
```

Replace zero by seven

Creating 1D Arrays of specific data type (1)

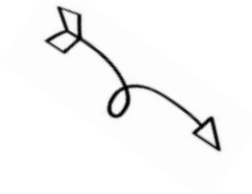
fila1=np.arange(4,dtype=float)

0.	1.	2.	3.

fila2=np.arange(5,dtype=int)

0	1	2	3	4

booleano=np.array([0,1,0],dtype=bool)

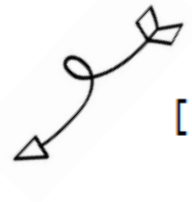


```
[False  True False]
```

False	True	False

0.
1.
2.
3.

col=fila1[:,np.newaxis]
print(col)



```
[[0.]
 [1.]
 [2.]
 [3.]]
```

azar=np.random.random((4,))

0.68764209	0.97134994	0.98634447	0.56372669

```
[ 0.68764209 0.97134994 0.98634447 0.56372669]
```

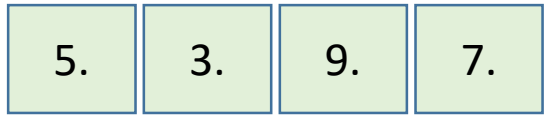
Input a sequence, (File: inputArray.py)

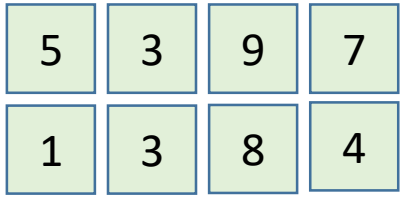
 # File: arrayinput.py

```
import numpy as np

x=input("Enter elements as string '1,2,3,4,5': \n")
print(type(x))
print(x)

x=eval(input("Enter elements as tuple (1,2,3,4,5): \n"))
print(type(x))
print(x)

x=eval(input("Enter elements as list [1,2,3,4,5]: \n"))
print(type(x))
print(x)

x=eval(input("Enter elements as np.array([1,2,3,4,5]): \n"))
print(type(x))
print(x)
```

```
Enter elements as string '1,2,3,4,5':
"1,2,3,4,5“  ← usuario write and <Enter>
<class 'str'>
"1,2,3,4,5"

Enter elements as tuple (1,2,3,4,5):
(1,2,3,4,5)
<class 'tuple'>
(1, 2, 3, 4, 5)

Enter elements as list [1,2,3,4,5]:
[1,2,3,4,5]
<class 'list'>
[1, 2, 3, 4, 5]

Enter elements as np.array([1,2,3,4,5]):
np.array([1,2,3,4,5])
<class 'numpy.ndarray'>
[1 2 3 4 5]
```

Array Basic Output: placeholders {}






```
v = np.array( [5, 3, 9, 7] )

print(v)
[5 3 9 7 ]
print("The 1D array is {}".format(v))
The 1D array is [5 3 9 7]

# type, dtype, shape, size functions:
print(type(v))          #<class 'numpy.ndarray'>
print(v.dtype)          # int32
print(np.shape(v))   # (4,) four elements in axis-0
print(np.size(v))     # 4  four elements
```

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

print(M)
[[5 3 9 7]
[1 3 8 4]]
print("The 2D array is:\n {} ".format(M))
The array is:
 [[5 3 9 7]
 [1 3 8 4]]

# type, dtype, shape, size functions:
print(type(M))          #<class 'numpy.ndarray'>
print(v.dtype)          # int32
print(np.shape(M))   # (2,4)
print(np.size(M))     # 8  elements
```

Array Iteration

```
import numpy as np
x=np.arange(2,10)  # [2,3,4,5,6,7,8,9]
print(x)
```

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

```
# Produces pair of index and elements
index=0
for item in x:
    print(index,item)
    index+=1
```

```
0 2
1 3
2 4
3 5
4 6
5 7
6 8
7 9
```

```
# Produces pair of index and elements
index=0
for item in x:
    print(index,x[index])
    index+=1
```

FILE:enumerateSomething.py

```
# Produces pair of tuples (position,number)
for item in enumerate(x):
    print(item)
```

```
(0, 2)
(1, 3)
(2, 4)
(3, 5)
(4, 6)
(5, 7)
(6, 8)
(7, 9)
```

```
# Unpack tuples
for pos,num in enumerate(x):
    print(pos,num)
```

```
0 2
1 3
2 4
3 5
4 6
5 7
6 8
7 9
```

Enumerate function

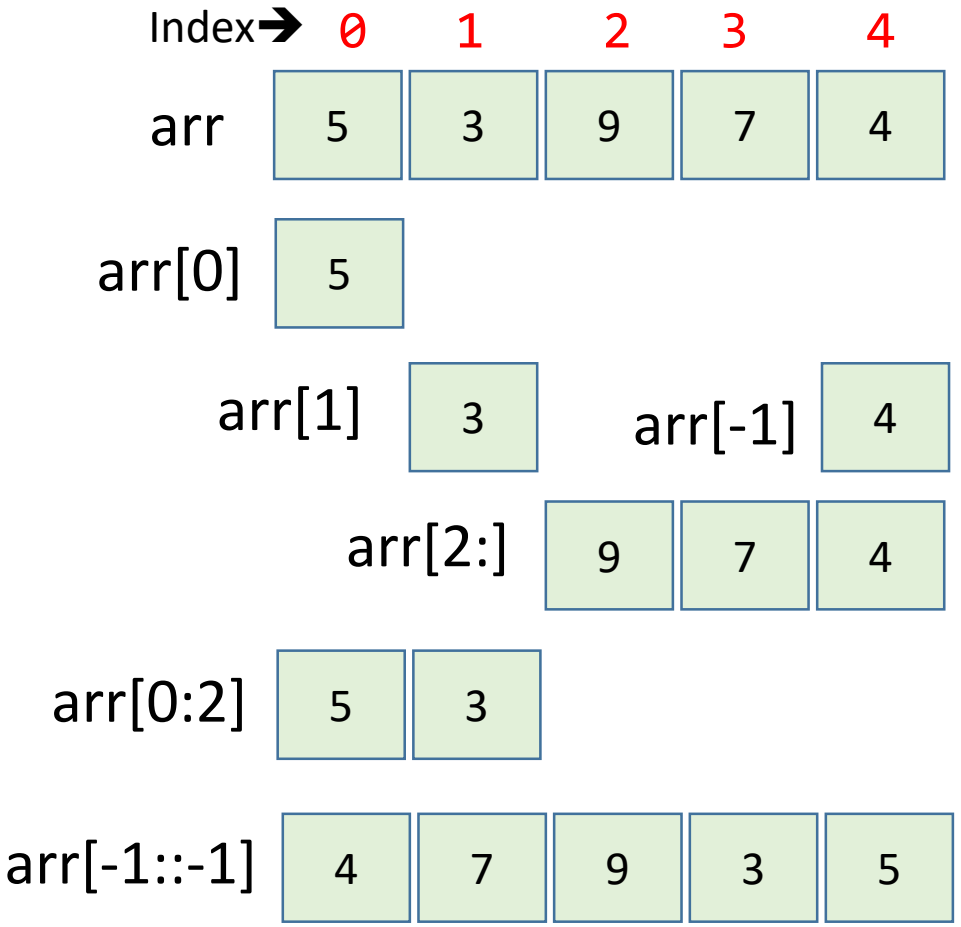
```
# Makes pos and num equal
for pos,num in enumerate(x,2):
    print(pos,num)
```

```
2 2
3 3
4 4
5 5
6 6
7 7
8 8
9 9
```

```
# You can start in any number
for pos,num in enumerate(x,10):
    print(pos,num)
```

```
10 2
11 3
12 4
13 5
14 6
15 7
16 8
17 9
```

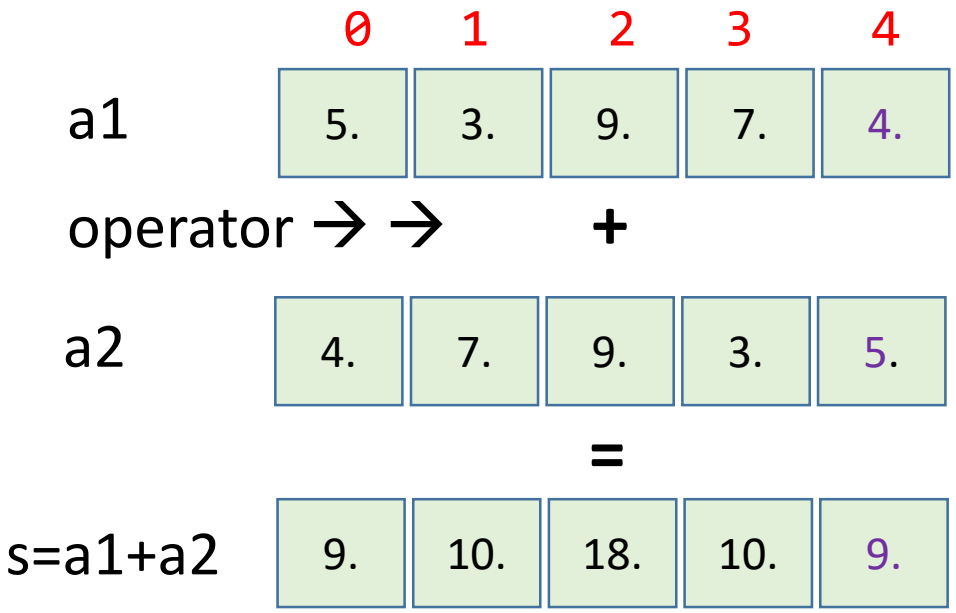
Indexing & Slicing




```
arr=np.array([5,3,9,7,4])

print(arr)     # [5 3 9 7 4]

print(arr[0]) # 5

print(arr[1],arr[-1])  # 3  4

print(arr[2:])  # [9 7 4]

print(arr[0:2])  # [5 3]

print(arr[-1::-1]) # [4 7 9 3 5]
```

Values can be: integer, real, complex, boolean
Indexes are integers starting at zero: 0, 1, 2, …

Array Mathematics: elementwise operations(1)



Arrays have same shape

```
a1=np.array([5,3,9,7,4],float)

a2=np.array([4,7,9,3,5],float)

s=a1+a2    # [9. 10. 18. 10. 9.]
```

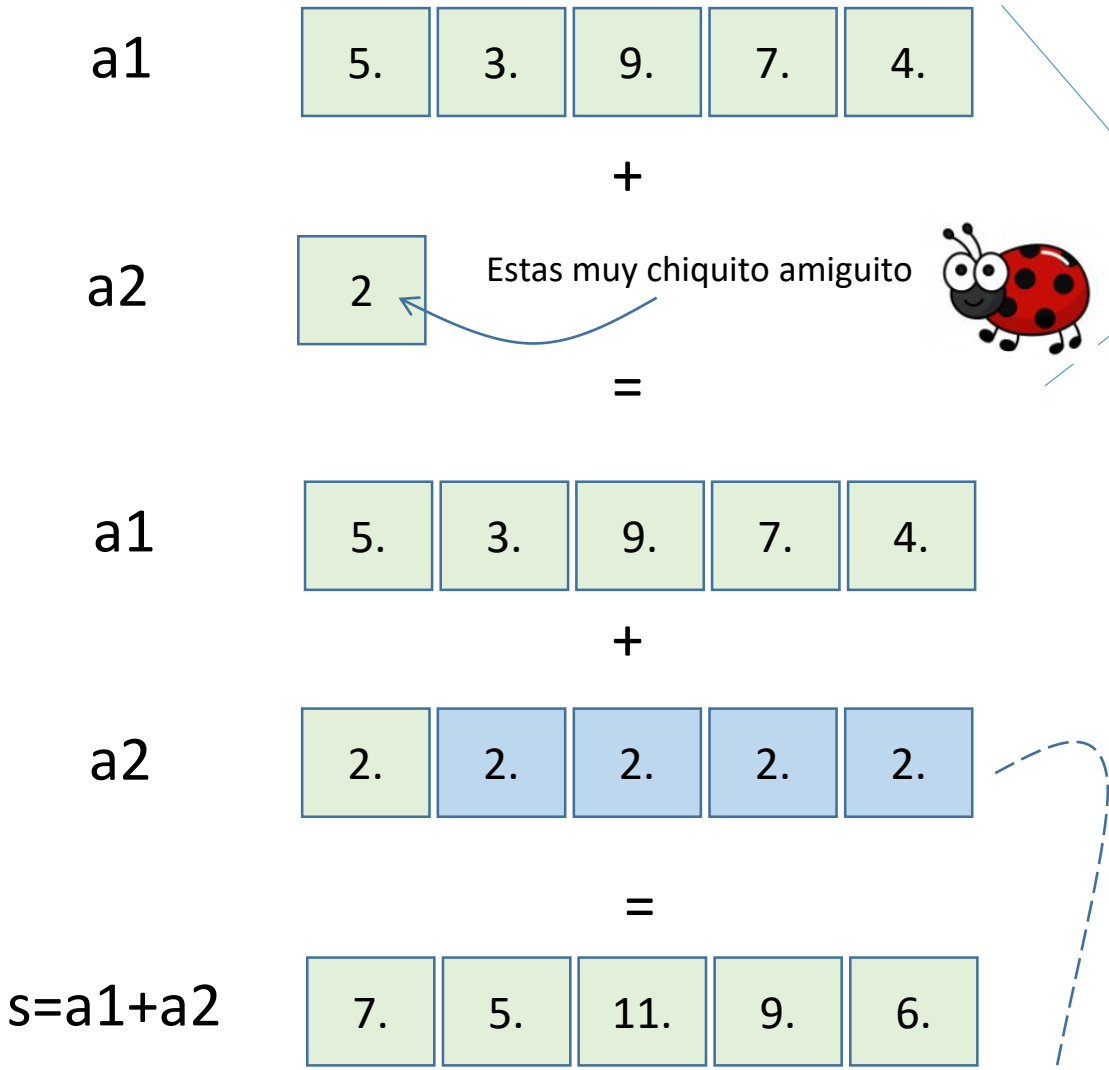
```
print(a1+a2)    [ 9. 10. 18. 10.  9.]
print(a1-a2)    [ 1. -4.  0.  4. -1.]
print(a1*a2)    [20. 21. 81. 21. 20.]
print(a1/a2)    [1.25       0.42857143 1.         2.33333333 0.8        ]
print(a1%a2)    [1. 3. 0. 1. 4.]
print(a1//a2)   [1. 0. 1. 2. 0.]
```

Actual output

Operator can be: +, -, *, /, %, //

AM: Scalar-array elementwise operations(2) & Broadcasting

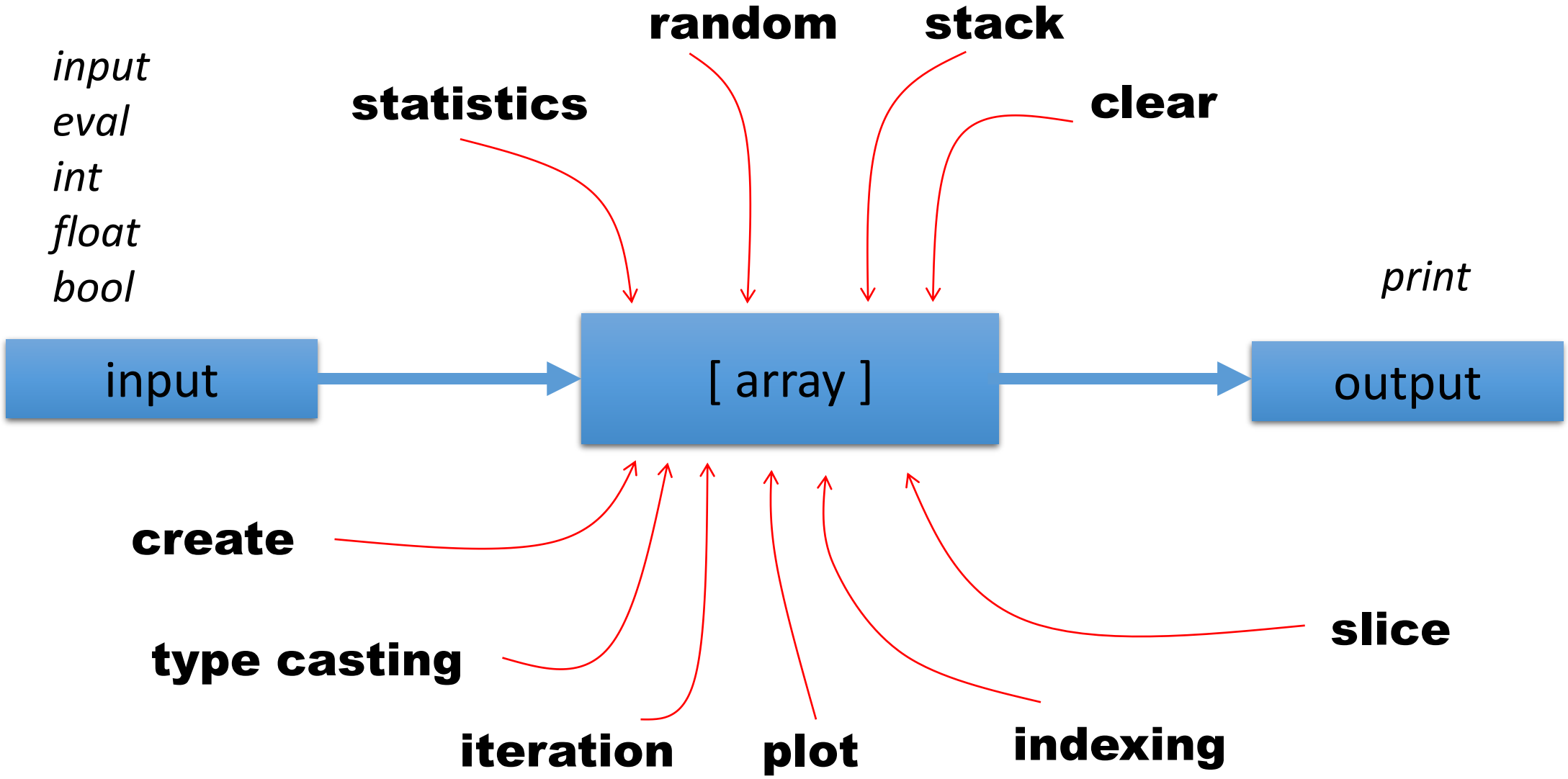
broadcasting refers to the ability of NumPy to treat arrays of different shapes during arithmetic operations




```
a1=np.array([5,3,9,7,4],float)

a2=np.array([2],float)  # or a2=2.0
```

```
print(a1+a2)    [ 7.  5. 11.  9.  6.]
print(a1-a2)    [3. 1. 7. 5. 2.]
print(a1*a2)    [10.  6. 18. 14.  8.]
print(a1/a2)    [2.5 1.5 4.5 3.5 2. ]
print(a1%a2)    [1. 1. 1. 1. 0.]
print(a1//a2)   [2. 1. 4. 3. 2.]
```

a2 value is repeated to maTch the shape and size of a1



IMPLEMENT ARRAYS IN PROGRAMMING

Next you will find some applications

Elementwise Operations & Enumerate: Example

Compute a table of Celsius to Fahrenheit in the range from 0 to 100 in steps of 10 using elementwise operations and the enumerate function

```
import numpy as np


# Create two arrays:
C=np.arange(0,101,10)      # creates C array
F=C*(9.0/5.0)+32.0         # create F array


# Print one-by-one
print('%3s %3s'%('C','F'))
for pos,cc in enumerate(C):
    print('%3.0f %3.1f'%(cc,F[pos]))
```

```
  C   F
  0 32.0
 10 50.0
 20 68.0
 30 86.0
 40 104.0
 50 122.0
 60 140.0
 70 158.0
 80 176.0
 90 194.0
100 212.0
```

FILE: ElementwiseOperations.py

FILE: plotCoronaV.py

numpy.append function: Example

Creando Conciencia

Exponential Growth is a mathematical function that can be used in several situations. The formula can tell us the number of infected cases at a certain moment in time.

$$x(t) = x_o b^t$$

In which:

$x(t)$ is the number of cases at any given time t (e.g., days)

x_o is the number of cases at the beginning (@ $t = 0$, $day\ zero$), the initial value

b is the number of people infected by each sick person, also called the growth factor

Experts say that coronavirus is very contagious, and b may be equal to 1.2. Write code to estimate the number of infected persons after t=90 days if the PR government didn't put in-place the current 'encierrese-en-su-casa' measures. Assume, as an engineer does, that $x_o = 1$. In what day (more or less) we could be all infected (Dios mio tragame la lengua y evita eso) if the current PR population is 3.195 million and assuming all people is living normally (i.e., no 'encierrese-en-su-casa').

https://towardsdatascience.com/modeling-exponential-growth-49a2b6f22e1f

Code and output:

This a simple solution which generates a table of days versus cases.

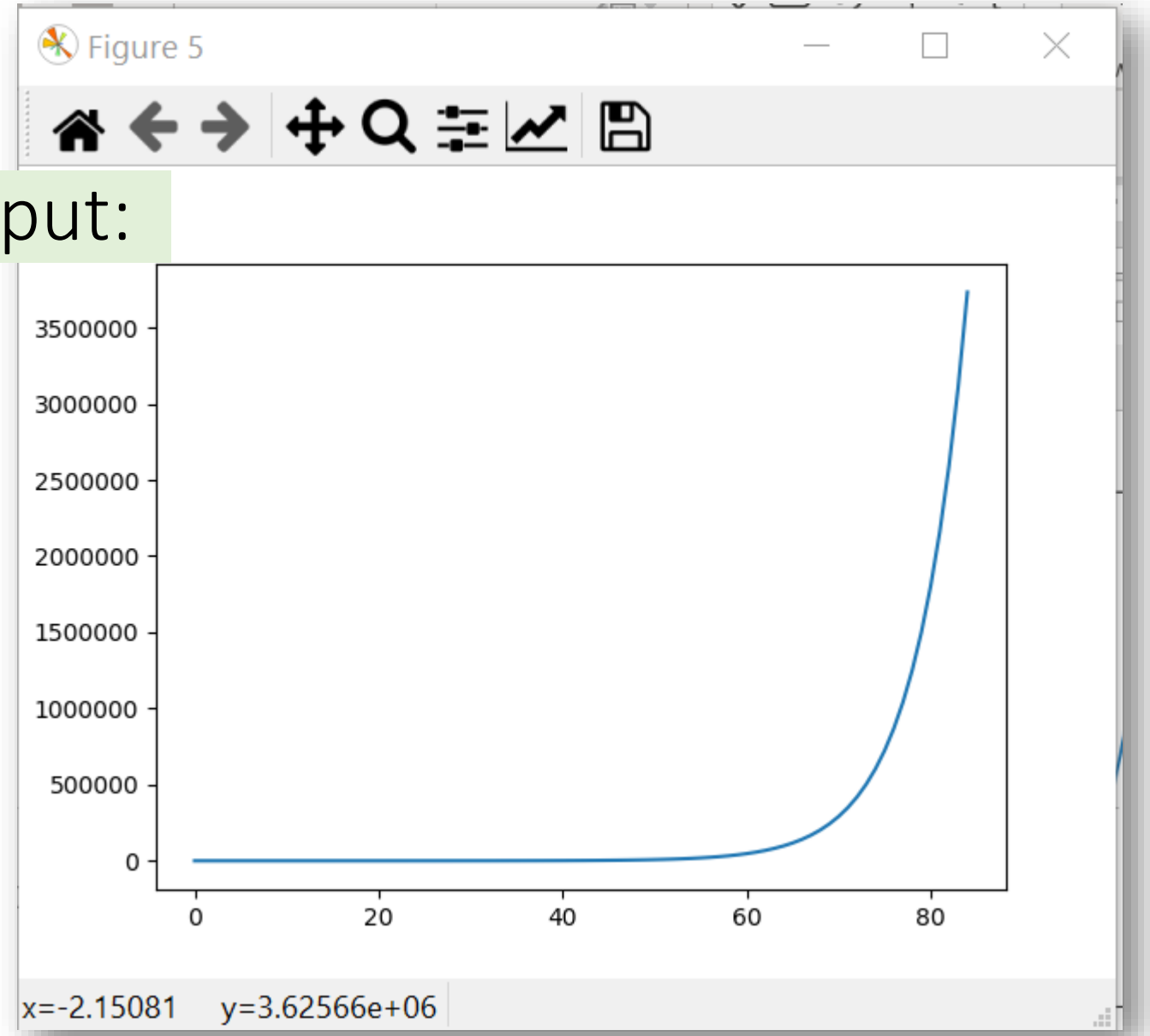
```
b=1.2
xo=1.0; x=xo
t=0.0
print('days cases')
while x<=3.195e6:
    x=xo*b**t
    print('%.0f %9.0f'%(t,x))
    t+=1.0

print("\nAll infected by day",int(t-1),"th")
```

```
days cases
0         1
1         1
2         1
3         2
4         2
5         2
6         3
```

```
78    1500159
79    1800190
80    2160228
81    2592274
82    3110729
83    3732875

All infected by day 83 th
```

```
# time vs infected cases
b=1.2
xo=1.0; x=xo
t=0.0
tt=np.array([t])
xx=np.array([x])
print('days cases')
while x<=3.195e6:
    x=xo*b**t
    print('%.0f %9.0f'%(t,x))
    xx=np.append(xx,x)
    t+=1.0
    tt=np.append(tt,t)

print("\nOne third infected by day",int(t-1),"th")

# Plot starts here
fig = plt.figure()
ax = plt.axes()
ax.plot(tt,xx)
```

Code and output:



This solution adds a plot. Add the following imports:
import numpy as np
import matplotlib.pyplot as plt

ADIOS

ADIOS

ADIOS

ADIOS

ADIOS

ADIOS