Introduction to Programming

PYTHON

Chapter 4 – Sequences, Sets, Arrays and Dictionary

Presenter: **Dr. Nguyen Dinh Long**

Email: dinhlonghcmut@gmail.com

Google-site: https://sites.google.com/view/long-dinh-nguyen

Oct. 2022

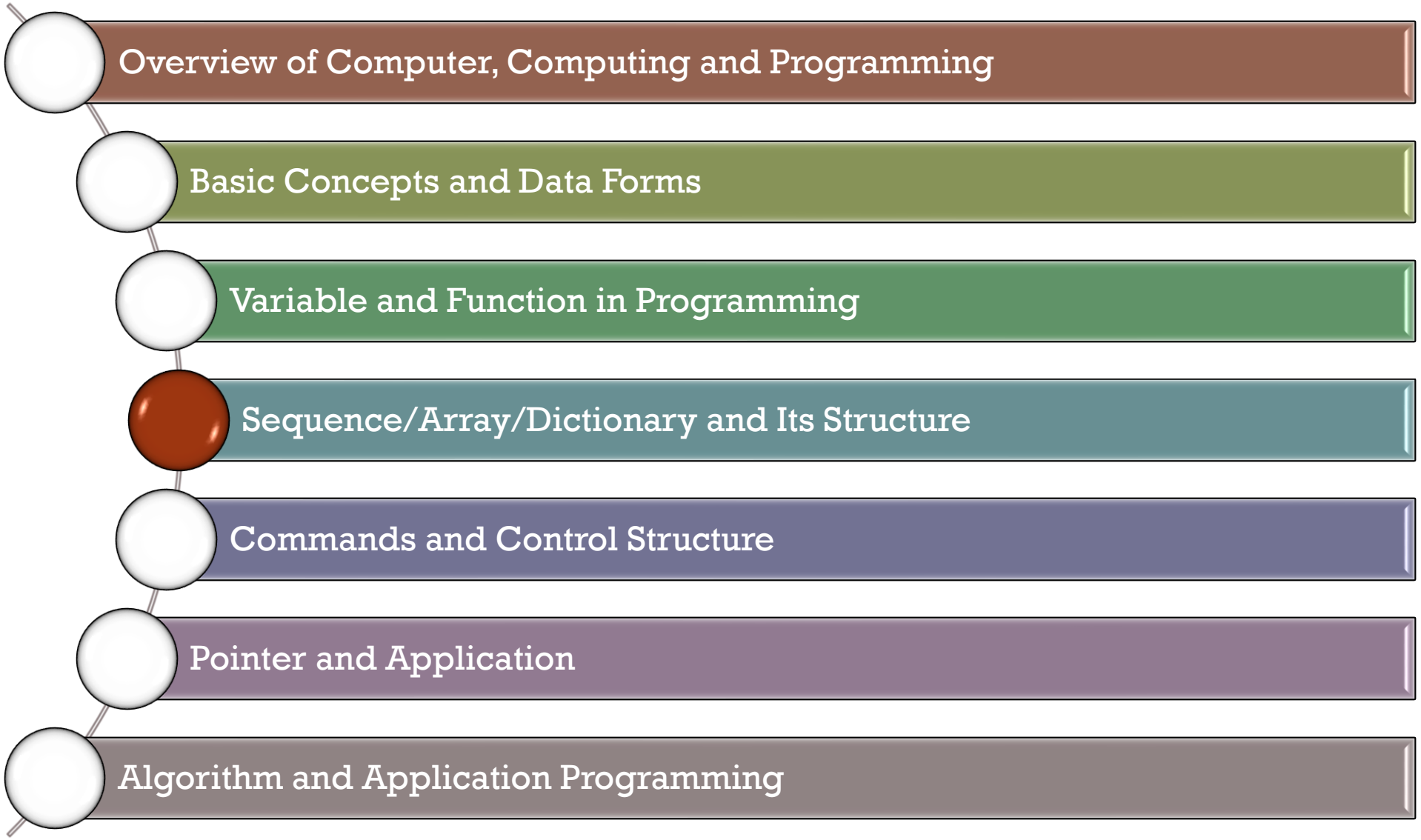
Programming

Hamilton was a self-taught programmer, working in the US in the 1960's. Owing to the success of her previous work, Hamilton was the first programmer to be hired for the Apollo project. She became the Director of Software Engineering at the MIT Instrumentation lab. Her lab developed the on-board flight software for NASA's Apollo space project, which took humankind to the moon.

The achievement was a monumental task at a time when computer technology was in its infancy: The astronauts had access to only 72 kilobytes of computer memory (a 256-gigabyte cell phone today carries almost a million times more storage space). Programmers had to use paper punch cards to feed information into room-sized computers with no screen interface.

Margaret Hamilton, NASA's lead software engineer for the Apollo, stands next to the code she wrote by hand that took humanity to the moon in 1969.

Outline


Overview of Computer, Computing and Programming
Basic Concepts and Data Forms
Variable and Function in Programming
Sequence/Array/Dictionary and Its Structure
Commands and Control Structure
Pointer and Application
Algorithm and Application Programming

References

Main:

- Maurizio Gabbrielli and Simone Martini, 2010. *Programming Languages: Principles and Paradigms*, Springer.
- Cao Hoàng Trụ, 2004. *Ngôn ngữ lập trình- Các nguyên lý và mô hình*, Nhà xuất bản Đại học Quốc gia Tp. Hồ Chí Minh

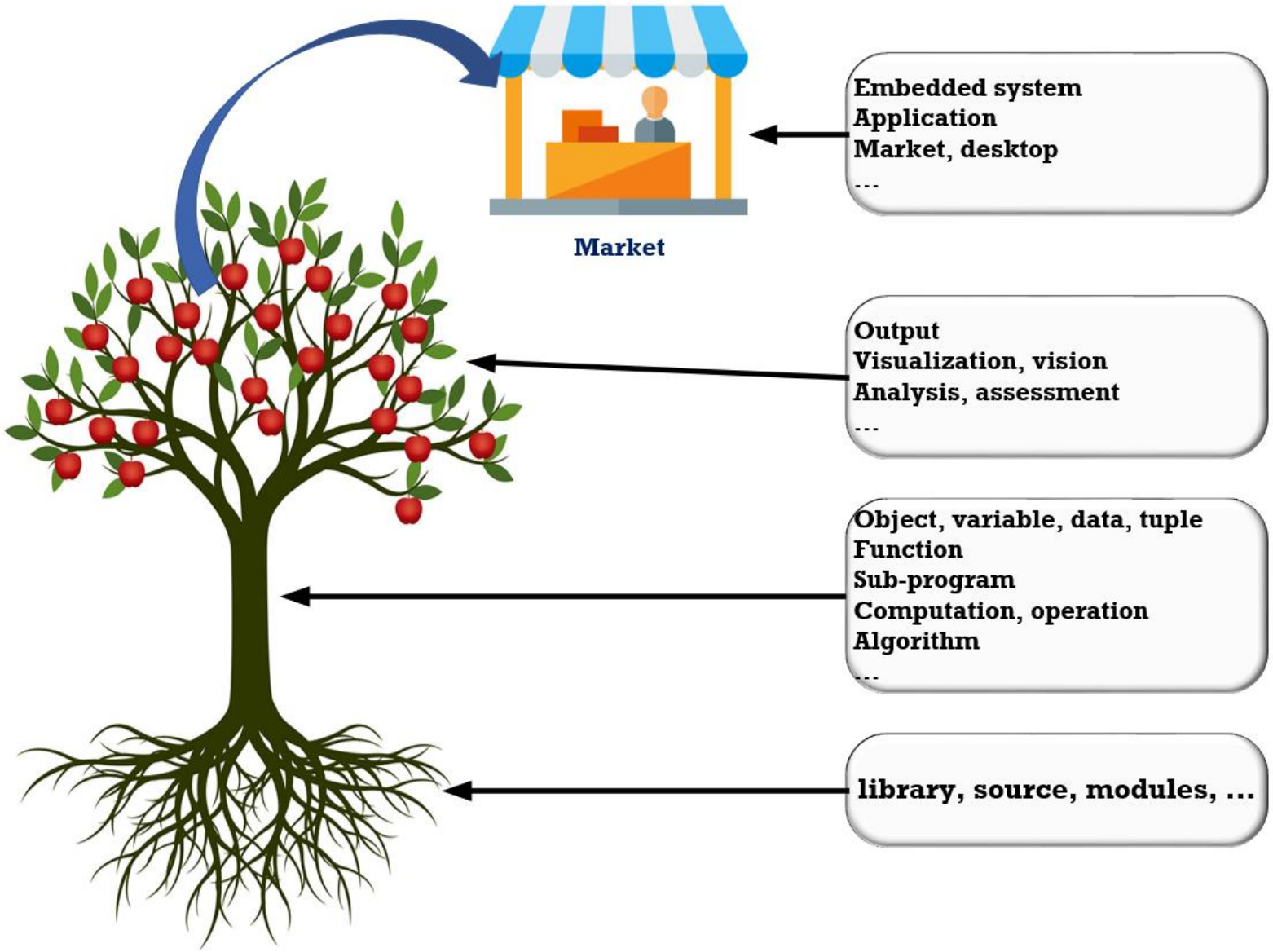
More:

- Wes McKinney, 2013. *Python for Data Analysis*, O'Reilly Media.
- Guido van Rossum, Fred L. Drake, Jr.,, 2012. *The Python Library Reference*, Release 3.2.3.

- Slides here are collected and modified from several sources in Universities and Internet.

Computer programs

- **General structure:**



Content of Chapter 4

1. Sequences: lists, strings and tuples, its structure
2. Sequences examples
3. Set, dictionary and examples
4. Arrays: vectors, lists, matrix, tensor, its structure
5. Arrays examples
6. Practice

Sequences and Structure

- **Lots of data:**
  - **arrays**
  - **ndarrays**

Austria	5	3	6
Canada	11	8	10
France	5	4	6
Germany	14	10	7
Italy	3	2	5
Japan	4	5	4
Netherlands	8	6	6
Norway	14	14	11
Russia	2	6	9
South Korea	5	8	4
Sweden	7	6	1
Switzerland	5	6	4
United States	9	8	6

Rank	NOC	Gold	Silver	Bronze	Total
1	Norway (NOR)	14	14	11	39
2	Germany (GER)	14	10	7	31
3	Canada (CAN)	11	8	10	29
4	United States (USA)	9	8	6	23
5	Netherlands (NED)	8	6	6	20
6	Sweden (SWE)	7	6	1	14
7	South Korea (KOR)	5	8	4	17
8	Switzerland (SUI)	5	6	4	15
9	France (FRA)	5	4	6	15
10	Austria (AUT)	5	3	6	14
11	Japan (JPN)	4	5	4	13
12	Italy (ITA)	3	2	5	10
13	Olympic Athletes from Russia (OAR)	2	6	9	17
14	Czech Republic (CZE)	2	2	3	7
15	Belarus (BLR)	2	1	0	3
16	China (CHN)	1	6	2	9
17	Slovakia (SVK)	1	2	0	3
18	Finland (FIN)	1	1	4	6
19	Great Britain (GBR)	1	0	4	5
20	Poland (POL)	1	0	1	2
21	Hungary (HUN)	1	0	0	1
	Ukraine (UKR)	1	0	0	1
23	Australia (AUS)	0	2	1	3
24	Slovenia (SLO)	0	1	1	2
25	Belgium (BEL)	0	1	0	1

List, Set and Dictionary

- **When do we use list, set or dictionary?**

.

- If we need to manage an ordered sequence of objects
  → Use a **List**

- If we need to manage an unordered set of values
  → Use a **Set**

- If we need to associate values with keys, so that we can easily look up the values by the keys
  → Use a **Dictionary**

Arrays and Structure

- **???**

When we are targeting to store the small sequence of elements and do not want to perform any mathematical operations, then **the list** is the preferred choice to make because list data structure will allow you to store an ordered and mutable (easily modification can be made) and indexed sequence storage of items without importing any explicit module.

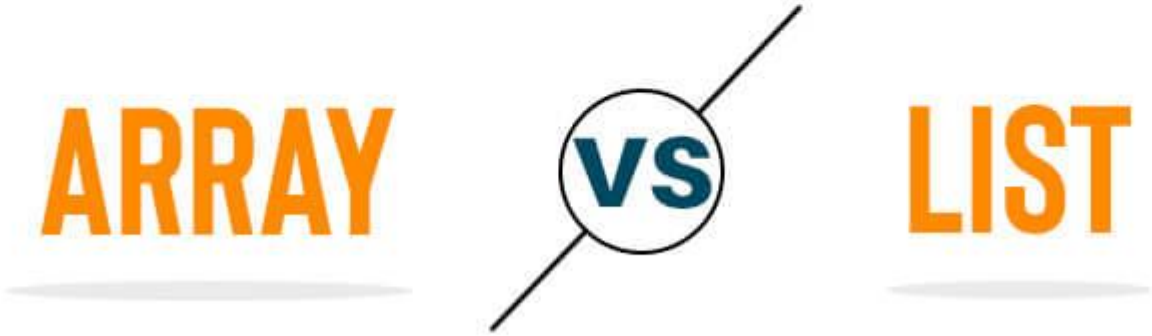
- More efficient data storage of elements therefore it is considered to use an array when we need to deal with a long sequence of data.
- If we are looking forward to performing the numerical operations on a combination of elements, it is recommended to use an array as an array data structure that relies heavily on data analytics and data science.

- Array and list both have their own advantages and disadvantages. You should understand the difference between array and list in Python. We can use them according to the requirement of the data to be stored and the operations that are to be performed on the elements stored.

Arrays and Structure

- **Array vs. List**

.



In python, both array and the list are used to store the data as a data structure. This data structure can be used for iteration and indexing. What is an array and what is a list actually and the main difference between array and list in python with a complete table for Array vs List.

Arrays and Structure

What is a List?

- The list is the most important data type in python language.
- In Python language, the list is written as the list of commas separated values inside the square bracket.
- The most important advantage of the list is the elements inside the list is not compulsorily be of the same data type along with negative indexing.
- All the operation of the string is similarly applied on list data type such as slicing, concatenation; create the nested list containing another list.

For example:

```
# creating a list of items with different data types
sample_list = [10,"sid",['A','B']]
print(sample_list)
```

Arrays and Structure

❑ What is an Array?

- An array is a data structure that holds fix number of elements and these elements should be of the same data type.
- Most of the data structure makes use of an array to implement their algorithm.
- There is two important part of the array:
  - Element: Each item store in the array is called an element
  - Index: Every element in the array has its own numerical value to identify the element.
- These elements allocate contiguous memory locations that allow easy modifications in data. To declare an array in python language, we will use the array module.

For example:

```
# creating an array containing the same data type elements

import array

arr = array.array('j', [a, b, c])

# accessing elements of array
for j in arr:
     print(j)
```

Arrays and Structure

- **Difference between Array and List in Python**

- Replaceability: Python list can be replaceable for array data structure only with few exceptional cases.
- Data Types Storage: Array can store elements of only one data type but List can store the elements of different data types too. Hence, Array stores homogeneous data values, and the list can store heterogeneous data values.
- Importing Module: List is the in-build data structure of python language hence no module or package is to be imported before using it. But the array is not an in-build data structure for python language. Hence, we need to import the “array” module before creating and using arrays.
- Numerical Operation: Array provides an advantage in performing Mathematical operations in the python language because the NumPy module provides us with an array structure to store data values and manipulate them easily. But the list on other hand does not reflect the results. The list is capable of performing the mathematical operations but they are less efficient in comparison to the array.
- Modification Capabilities: Array has very poor performance in resizing and modifying the memory location but the list on the other hand is an in-build data structure and hence can be resized and modify very easily and efficiently.

Arrays and Structure

- **Difference …**

List	Array
Contains elements of different data types	Contains elements of same data types
Explicitly importing module is not required to declare a list	Need to import the module explicitly to declare an array
Cannot handle arithmetic operations	Can handle arithmetic operations
Can be nested inside another list	Must contain all elements of the same size
Mostly used in the shorter sequence of data elements	Mostly used in the longer sequence of data elements
Easy modifications like addition, deletion, and update of data elements are done	It is difficult to modify an array since addition, deletion and update operation is performed on a single element at a time
We can print the entire list without the help of an explicit loop	To print or access array elements, we will require an explicit loop
For easy addition of element, large memory storage is required	In comparison to the list, it is more compact in-memory size

Arrays and Structure

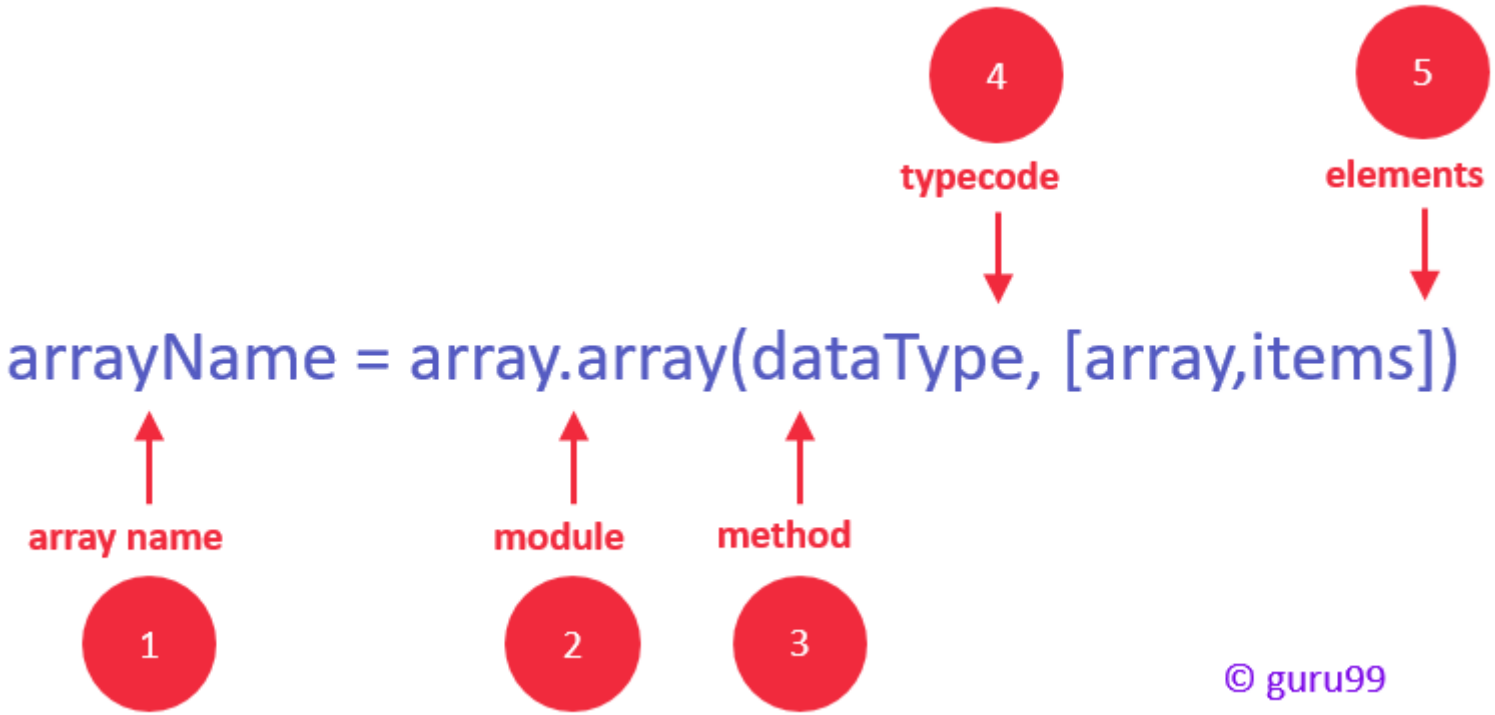
❑ Array applications

- Arrangement of the leader-board of a game can be done simply through arrays to store the score and arrange them in descending order to clearly make out the rank of each player in the game.
- A simple question Paper is an array of numbered questions with each of them assigned some marks.
- 2D arrays, commonly known as, matrices, are used in image processing.
- It is also used in speech processing, in which each speech signal is an array.
- Your viewing screen is also a multidimensional array of pixels.
- Book titles in a Library Management Systems.
- Online ticket booking.
- Contacts on a cell phone.
- For CPU scheduling in computer.
- To store the possible moves of chess on a chessboard.
- To store images of a specific size on an android or laptop.

Arrays and Structure

❑ **Arrays:**

- An array is a data structure that lets us hold multiple values of the same data type. Think of it as a container that holds a fixed number of the same kind of object. Python makes coding easier for programmers.
- An array is used to store more than one value at a time. It can hold multiple values in a single variable, and also helps you reduce the overall size of the code. Arrays save time.



Arrays and Structure

❑ Creating an Array in Python

An array is created by importing an array module to the Python program.

Syntax: from array import *

arrayName = array(typecode, [Initializers])

```
In [ ]: #an array
        from array import *

        array1 = array('i', [10,20,30,40,50])
```

Data type	Description
bool_	Boolean (True or False) stored as a byte
int_	Default integer type (same as C long; normally either int64 or int32)
intc	Identical to C int (normally int32 or int64)
intp	Integer used for indexing (same as C ssize_t; normally either int32 or int64)
int8	Byte (–128 to 127)
int16	Integer (–32768 to 32767)
int32	Integer (–2147483648 to 2147483647)
int64	Integer (–9223372036854775808 to 9223372036854775807)
uint8	Unsigned integer (0 to 255)
uint16	Unsigned integer (0 to 65535)
uint32	Unsigned integer (0 to 4294967295)
uint64	Unsigned integer (0 to 18446744073709551615)
float_	Shorthand for float64
float16	Half-precision float: sign bit, 5 bits exponent, 10 bits mantissa
float32	Single-precision float: sign bit, 8 bits exponent, 23 bits mantissa
float64	Double-precision float: sign bit, 11 bits exponent, 52 bits mantissa
complex_	Shorthand for complex128
complex64	Complex number, represented by two 32-bit floats
complex128	Complex number, represented by two 64-bit floats

Arrays and Structure

- **Accessing the Elements of an Array**

To access an array element, you need to specify the index number. Indexing starts at 0 - not at 1.

Hence, the index number is always one less than the length of the array.

```
In [14]: from array import *

         array1 = array('i', [1,2,3,4,5])

         print (array1[0])

         print (array1[2])

         1
         3
```

Arrays and Structure

❑ Basic Array Operations

- Insertion Operation

We can add value to an array by using the insert() function.

This function inserts the element at the respective index.

Syntax: insert(index, value)

```
In [16]: from array import *

         array1 = array('i', [1,2,3,4,5])
         array1.insert(3,9)
         print(array1)

         array('i', [1, 2, 3, 9, 4, 5])
```

- Deletion Operation

Array elements can be removed by using remove() or pop().

```
In [17]: from array import *

         array1 = array('i', [1,2,3,4,5])
         array1.remove(1)
         print(array1)

         array('i', [2, 3, 4, 5])
```

- Array Concatenation

Two arrays can be combined by using the + symbol.

```
In [22]: from array import *

         array1 = array('i', [1,2,3,4,5])
         array2 = array('i', [7,8,9])
         array3 = array1 + array2
         print(array3)

         array('i', [1, 2, 3, 4, 5, 7, 8, 9])
```

Arrays and Structure

❑ Basic Array Operations

- Looping Through an Array

We can get all the elements in an array by using a loop.

```
In [23]: from array import *

         array1 = array('i', [1,2,3,4,5])
         for x in array1:
             print(x)

         1
         2
         3
         4
         5
```

- Reversing the Elements

You can reverse the order of elements by using the reverse() method.

```
In [2]: from array import *

        array1 = array('i', [1,2,3,4,5])
        array1.reverse()
        print(array1)

        array('i', [5, 4, 3, 2, 1])
```

- Count() Method

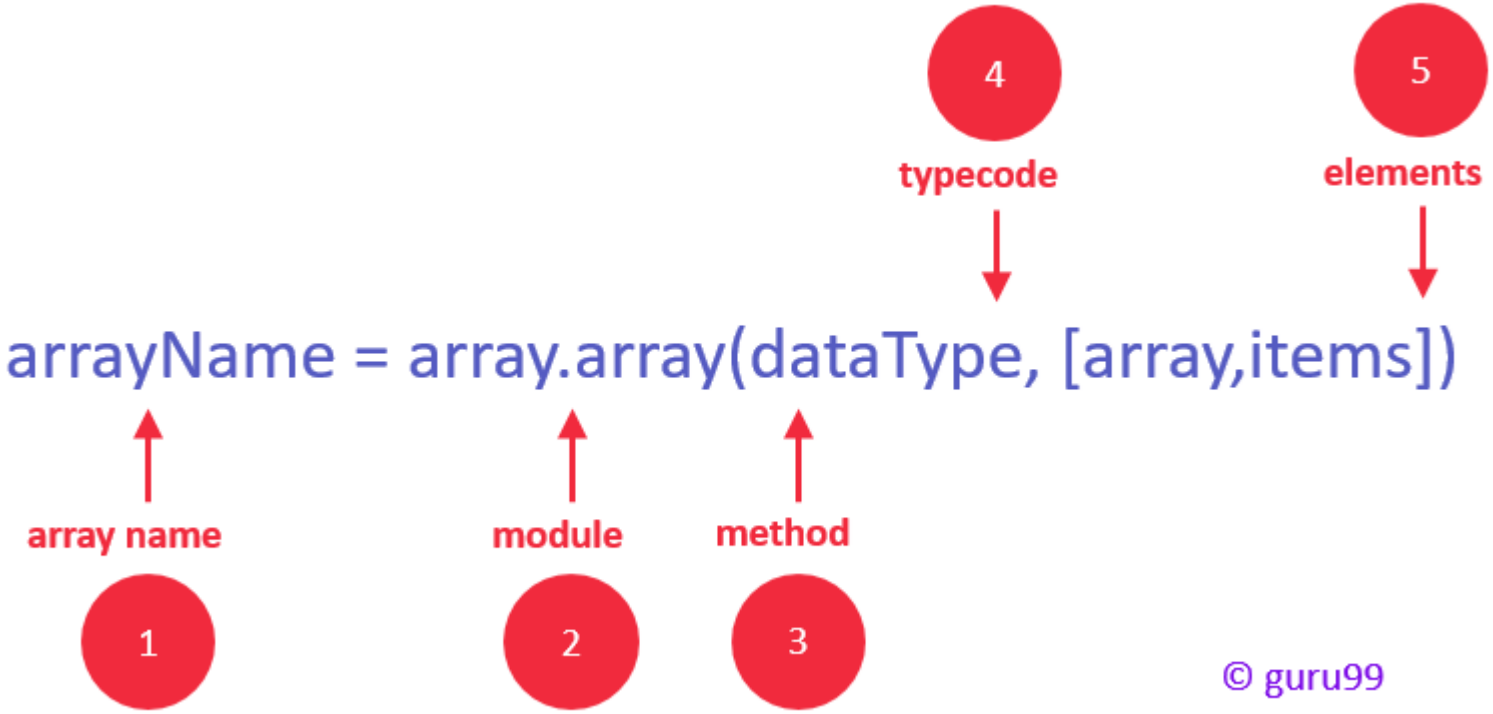
The count() method is used to count
the number of occurrences of a particular element.

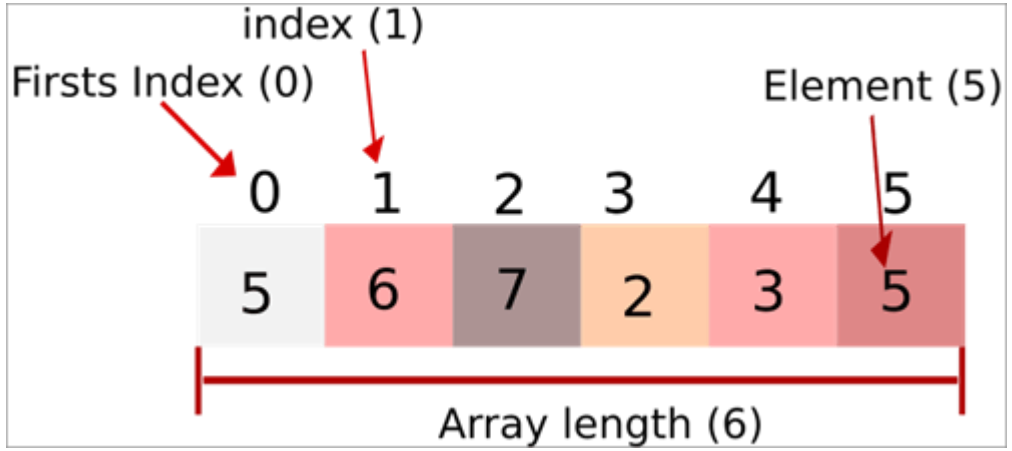
```
In [7]: from array import *

        array1 = array('i', [1,2,3,3,3,4,5])
        array1.count(3)

Out[7]: 3
```

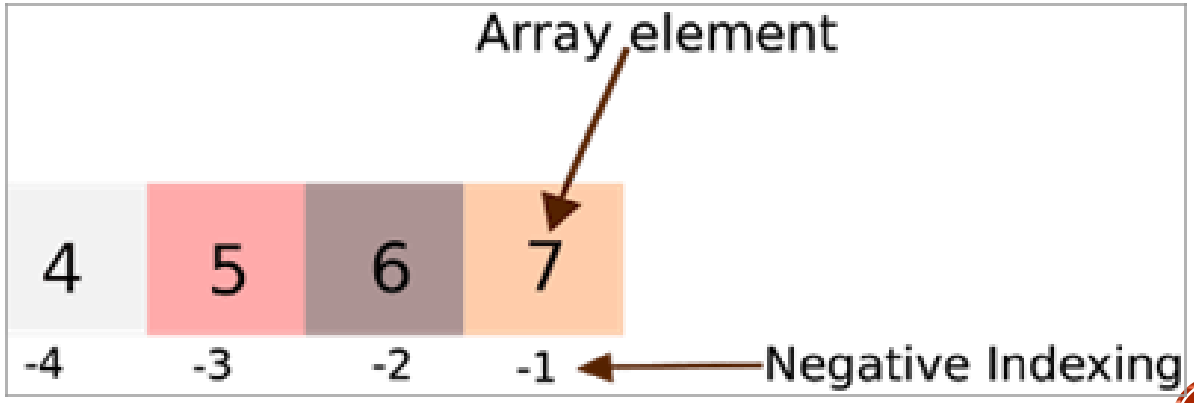
Arrays and Structure

- **Python libraries for Arrays:**

.



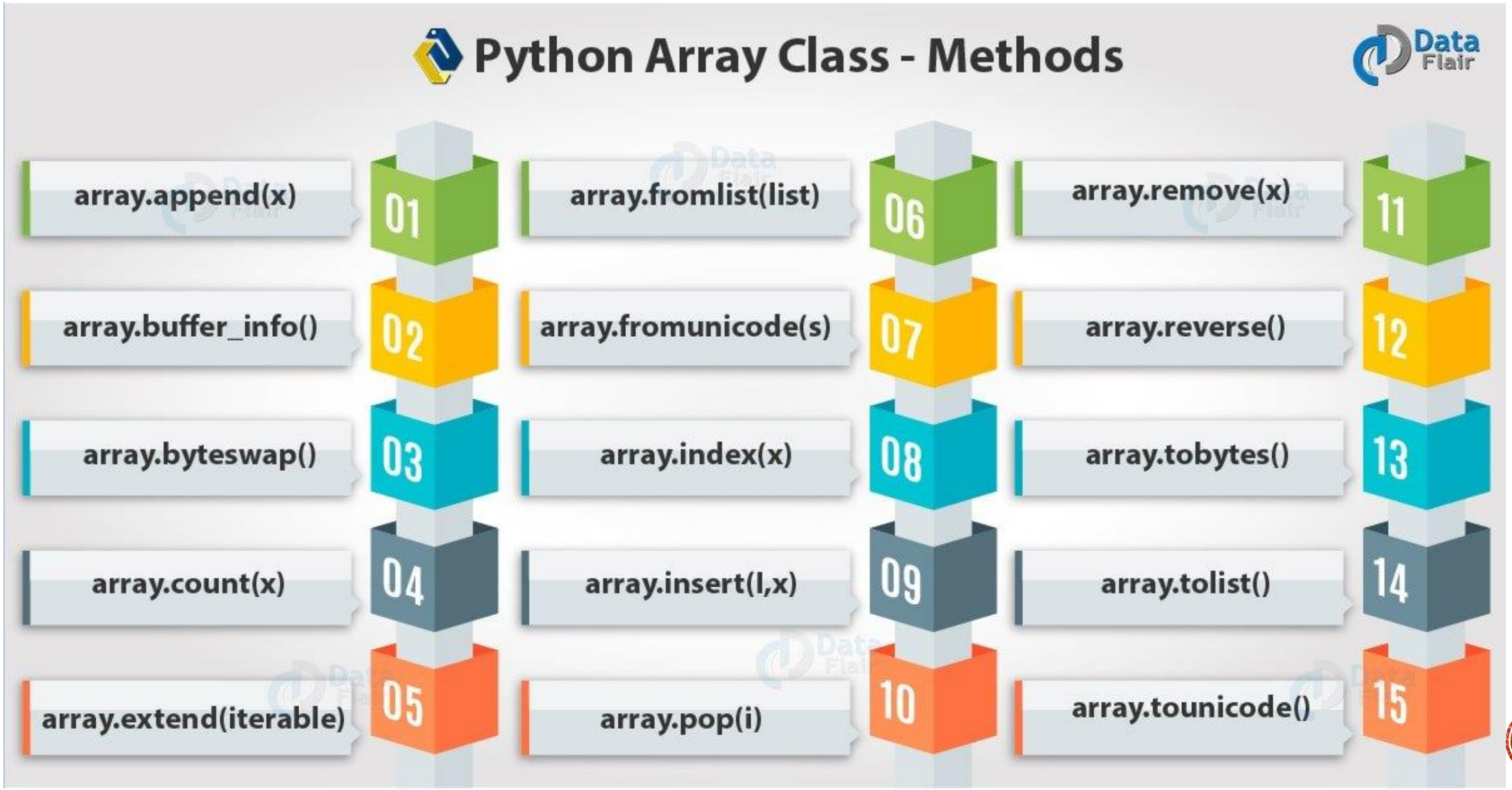




Arrays and Structure

- **Python libraries for Arrays:**

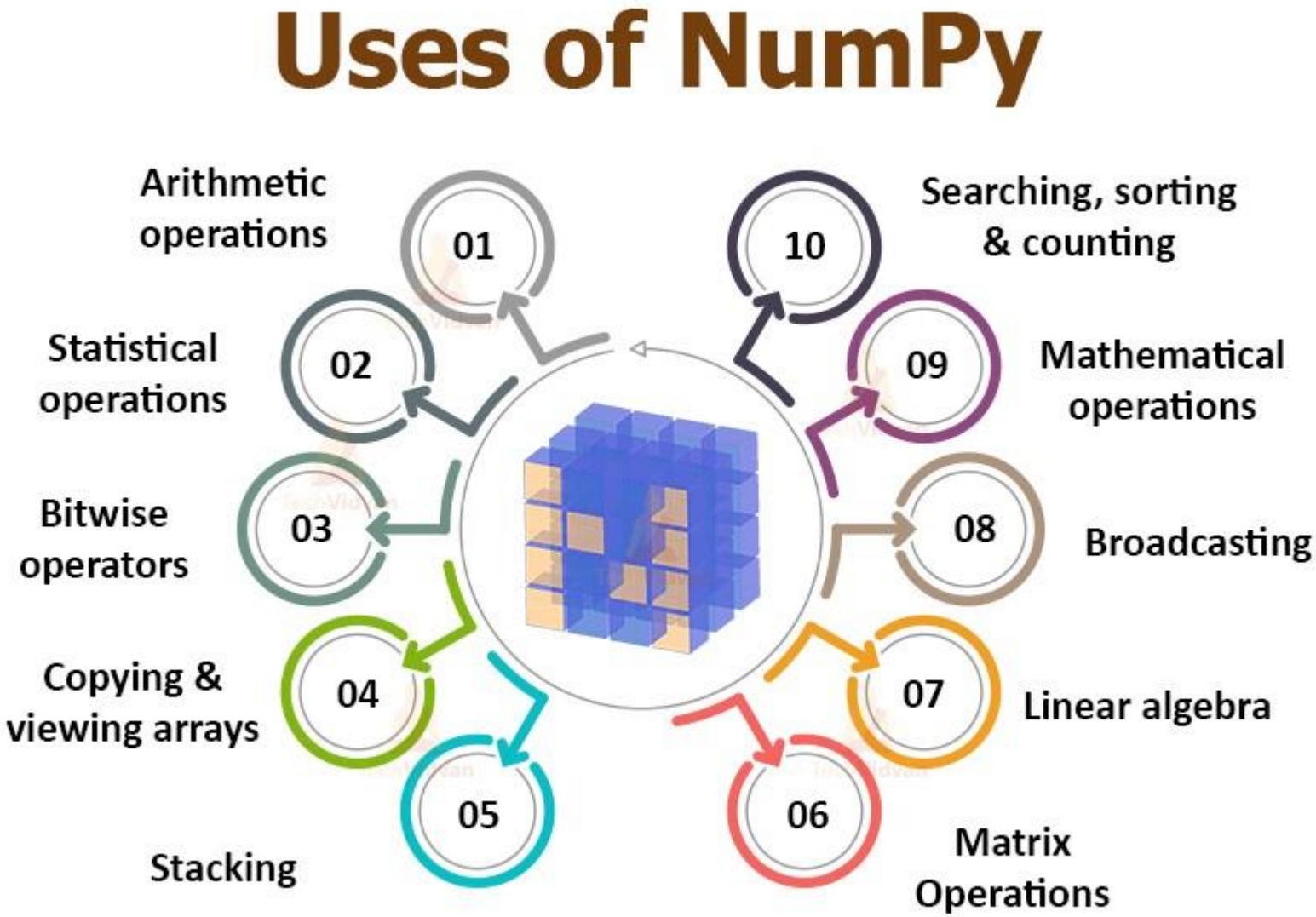
.



Arrays and Structure

- **Python libraries for Arrays:**

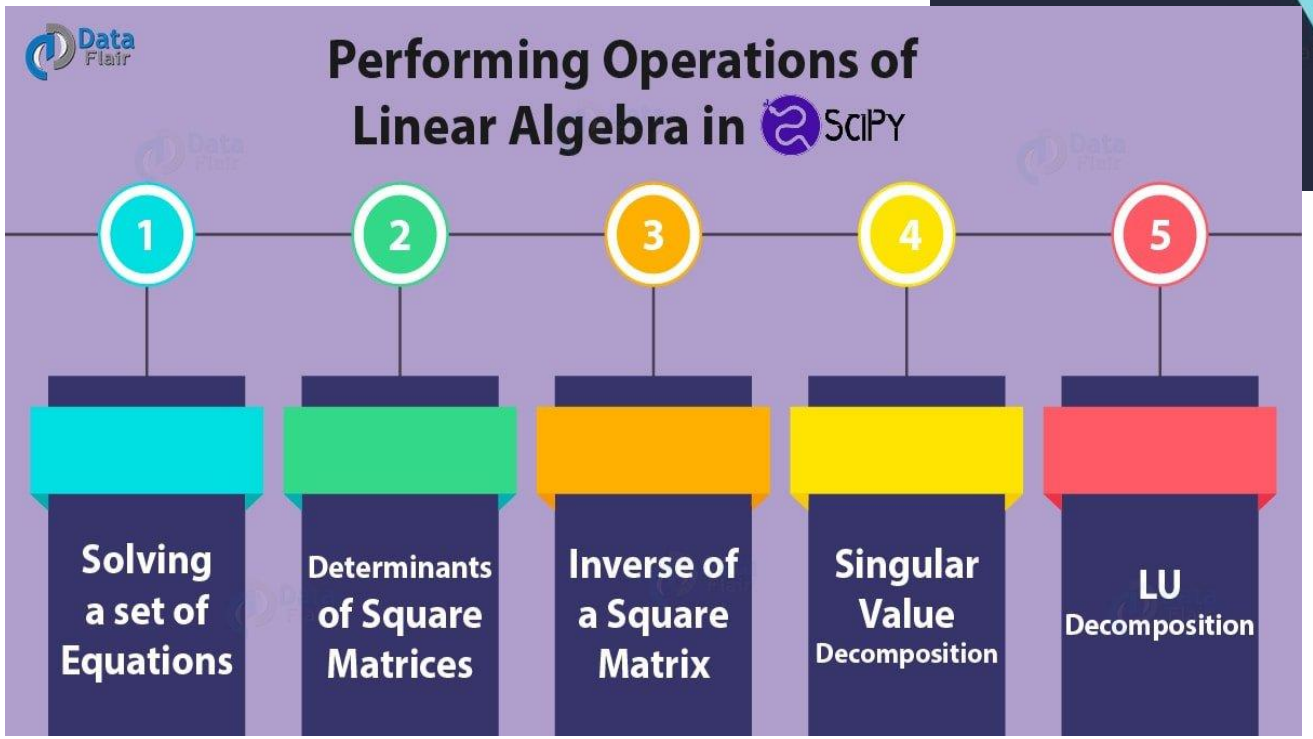
.



Arrays and Structure

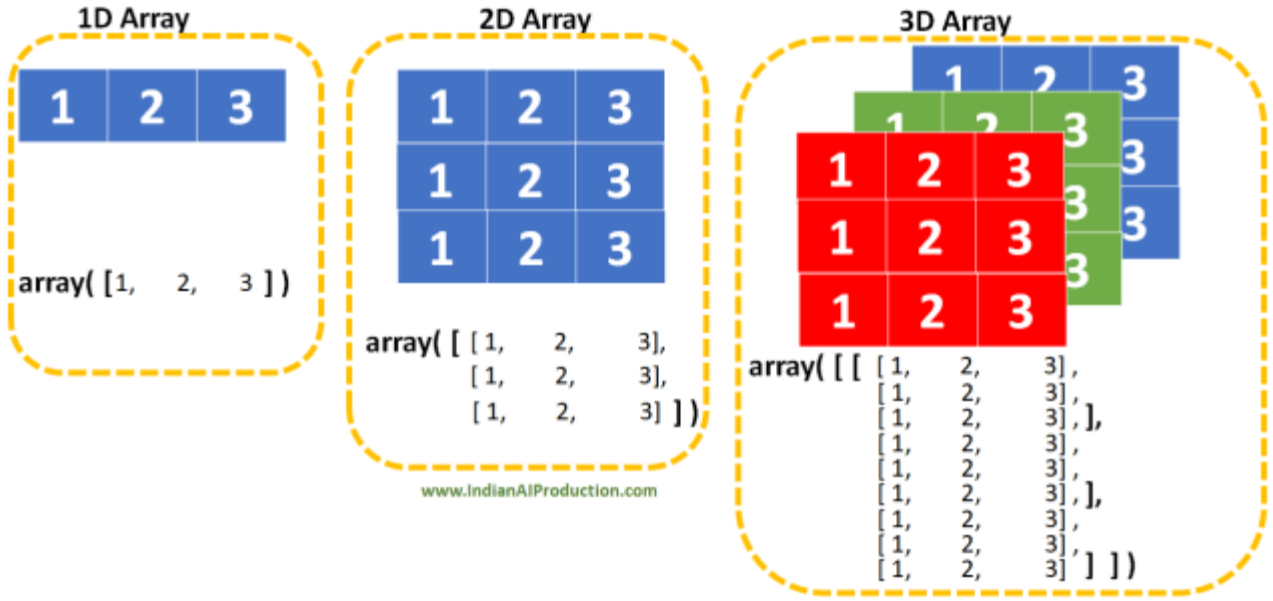
- **Python libraries for Arrays:**

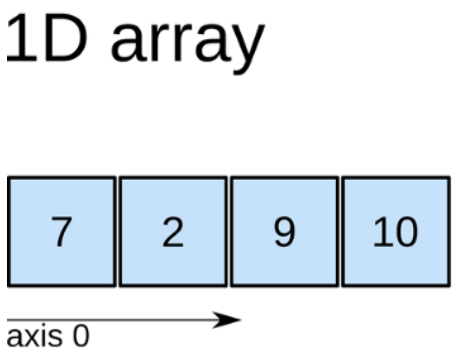
.

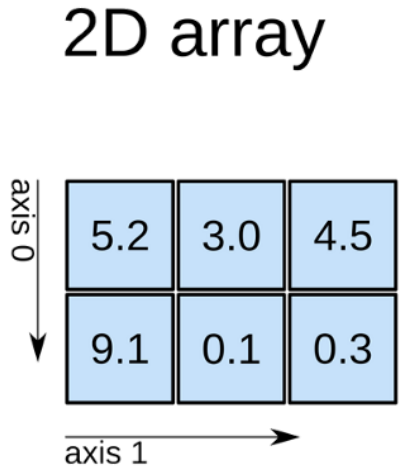


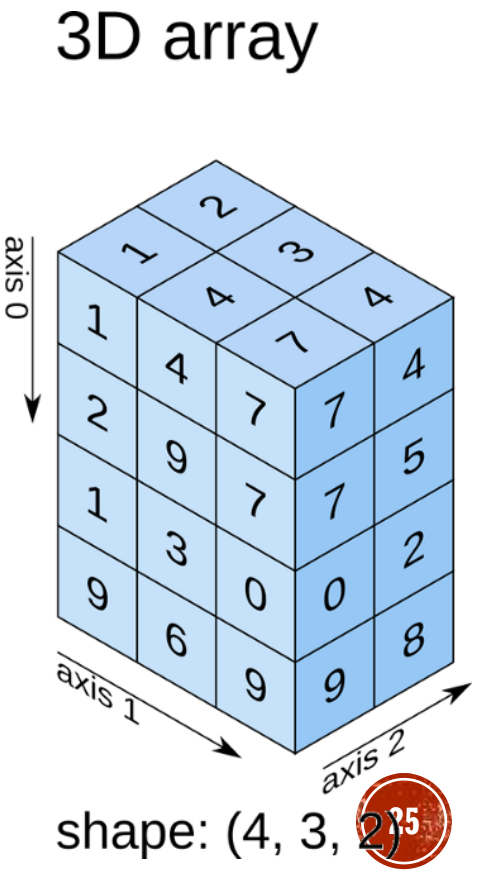
Arrays and Structure

- **Array types**







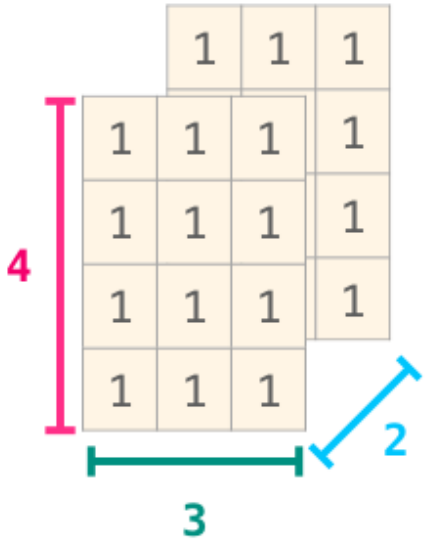


Arrays and Structure

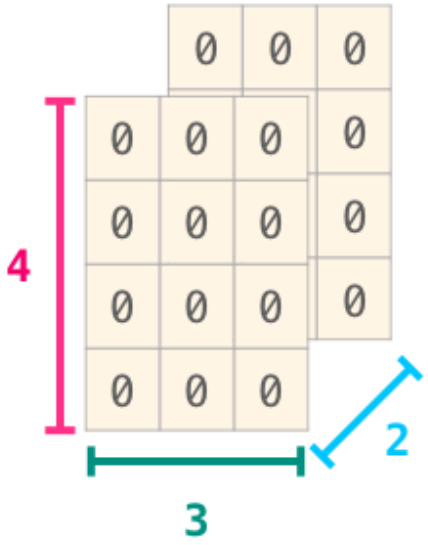
- **Array types**

.

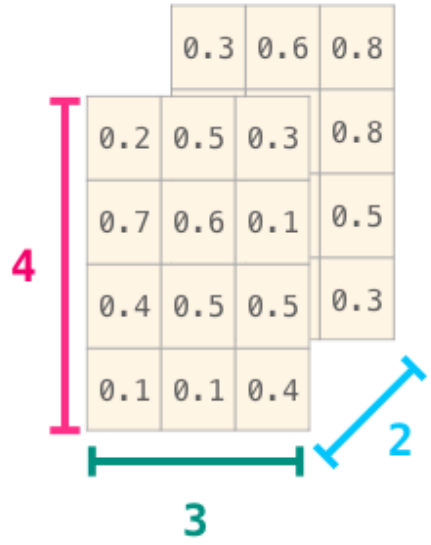
`np.ones((4,3,2))`



`np.zeros((4,3,2))`



`np.random.random((4,3,2))`



Arrays and Structure

- **Import the library**

```
import numpy as np
```

- **Setting up the Vector and Matrix**

```
vector = np.array([10, 20, 30, 40, 50, 60])
print("Original Vector:  ", vector)
```

```
matrix = np.array([[11, 22, 33],
                   [44, 55, 66],
                   [77, 88, 99]])
print("Original Matrix:  ", matrix)
```

- **Calculating of vector and matrix**

```
V = vector.T
print("Transpose Vector:  ", V)
```

```
M = matrix.T
print("Transpose Matrix:  ", M)
```

Arrays and Structure

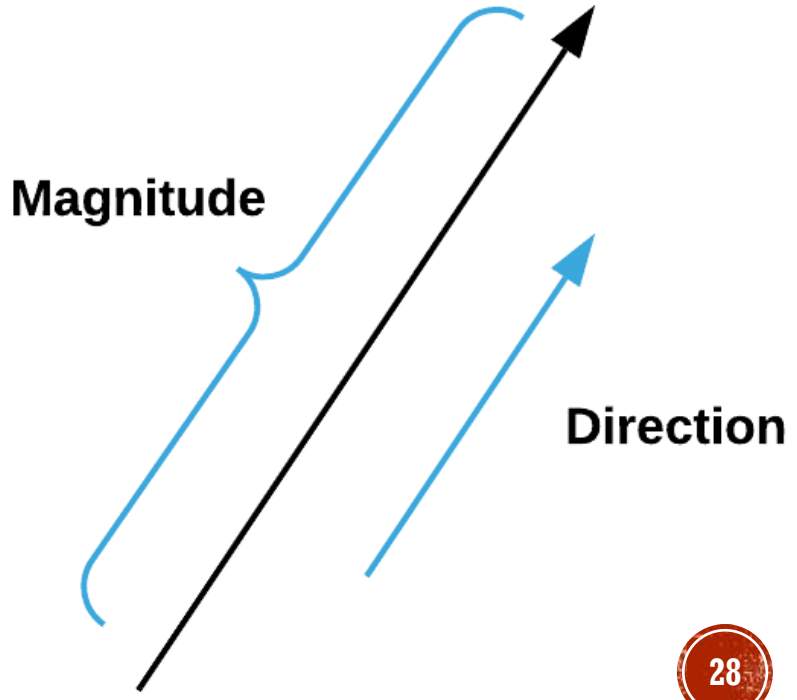
- **Making a Vector (1D array)?**

According to Google, a Vector is a quantity having direction as well as magnitude, especially as determining the position of one point in space relative to another.

Vectors are very important in Machine Learning as they not just describe magnitude but also the direction of the features.

```
import numpy as np

row_vector = np.array([1,2,3])
print(row_vector)
```

```
import numpy as np

col_vector = np.array([[1],[2],[3]])
print(col_vector)
```

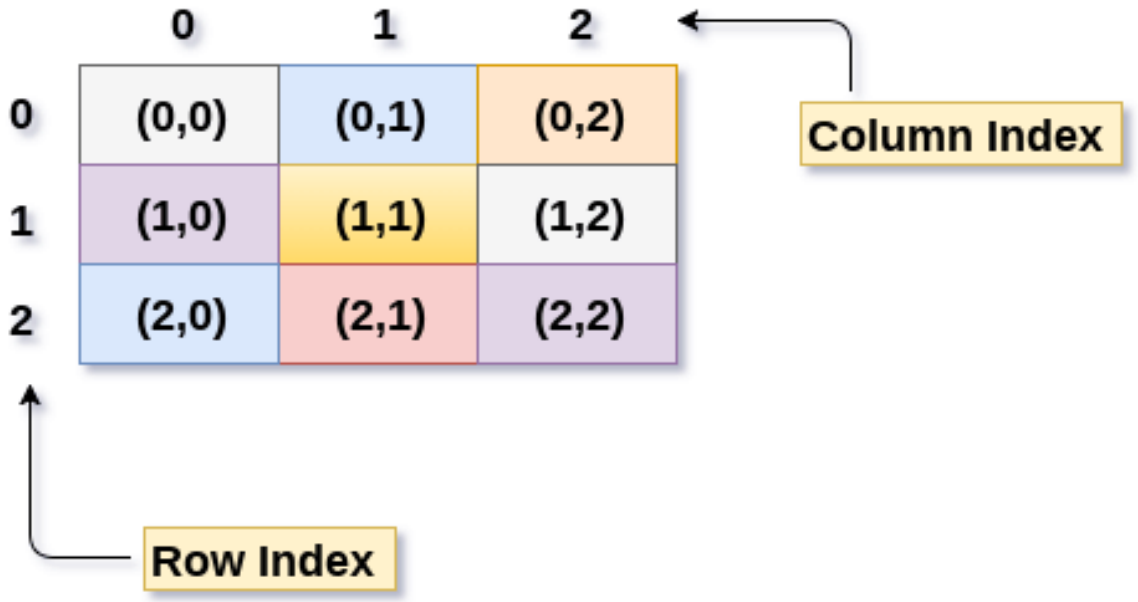


Arrays and Structure

- **Making a Matrix (2D array)?**

A matrix can be simply understood as a two-dimensional array.
We can make a matrix with NumPy by making a multi-dimensional array.

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



Arrays and Structure

- **Examples of Matrix (2D array)**

.

```
A = [[1, 4, 5, 12],
     [-5, 8, 9, 0],
     [-6, 7, 11, 19]]

print("A =", A)
print("A[1] =", A[1])      # 2nd row
print("A[1][2] =", A[1][2])   # 3rd element of 2nd row
print("A[0][-1] =", A[0][-1])   # Last element of 1st Row

column = [];         # empty list
for row in A:
  column.append(row[2])

print("3rd column =", column)
```

Arrays and Structure

- **Examples of Matrix (2D array)**

Array of integers, floats and complex Numbers

```
import numpy as np

A = np.array([[1, 2, 3], [3, 4, 5]])
print(A)

A = np.array([[1.1, 2, 3], [3, 4, 5]]) # Array of floats
print(A)

A = np.array([[1, 2, 3], [3, 4, 5]], dtype = complex) # Array of complex numbers
print(A)
```

Arrays and Structure

- **Examples of Matrix (2D array)**

Array of zeros and ones

```
import numpy as np

zeors_array = np.zeros( (2, 3) )
print(zeors_array)

'''
 Output:
 [[0. 0. 0.]
  [0. 0. 0.]]
'''

ones_array = np.ones( (1, 5), dtype=np.int32 ) // specifying dtype
print(ones_array)      # Output: [[1 1 1 1 1]]
```

Here, we have specified `dtype` to 32 bits (4 bytes). Hence, this array can take values from `-2^-31` to `2^-31-1`.

Arrays and Structure

- **Examples of Matrix (2D array)**

Using arange() and shape()

```
import numpy as np

A = np.arange(4)
print('A =', A)

B = np.arange(12).reshape(2, 6)
print('B =', B)

'''
Output:
A = [0 1 2 3]
B = [[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]]
'''
```

Arrays and Structure

❑ Making a Matrix (2D array) – Sparse Matrix?

- A sparse matrix is the one in which most of the items are zero. Common scenarios in data processing and machine learning is processing matrices in which most of the elements are zero.
- For example, consider a matrix whose rows describe every video on Youtube and columns represents each registered user.
  Each value represents if the user has watched a video or not. Majority of the values in this matrix will be zero. The advantage with sparse matrix is that it doesn't store the values which are zero. This results in a huge computational advantage and storage optimization as well.

```
from scipy import sparse

original_matrix = np.array([[1, 0, 3], [0, 0, 6], [7, 0, 0]])
sparse_matrix = sparse.csr_matrix(original_matrix)
print(sparse_matrix)
```

Arrays and Structure

- **Applying Operations to Vectors, Matrices:**

Addition of Two Matrices

```
import numpy as np

A = np.array([[2, 4], [5, -6]])
B = np.array([[9, -3], [3, 6]])
C = A + B      # element wise addition
print(C)

'''
Output:
[[11  1]
 [ 8  0]]
 '''
```

Arrays and Structure

- **Applying Operations to Vectors, Matrices:**

Multiplication of Two Matrices

```
import numpy as np

A = np.array([[3, 6, 7], [5, -3, 0]])
B = np.array([[1, 1], [2, 1], [3, -3]])
C = A.dot(B)
print(C)

'''
Output:
[[ 36 -12]
 [ -1   2]]
'''
```

Arrays and Structure

❑ Dot Products of Vectors

Dot Products of Vectors is a way of multiplying 2 vectors. It tells you about how much of the vectors are in the same direction, as opposed to the cross product which tells you the opposite, how little the vectors are in the same direction (called orthogonal).

```
a = np.array([3, 5, 6])
b = np.array([23, 15, 1])

np.dot(a, b)
```

❑ Adding, Subtracting and Multiplying Matrices

Adding and Subtracting multiple matrices is quite straightforward operation in matrices. There are two ways in which this can be done. Let's look at the code snippet to perform these operations.

```
np.add(matrix, matrix)
```

```
matrix * matrix
```

```
np.subtract(matrix, matrix)
```

```
np.dot(matrix, matrix)
```

Arrays and Structure

❑ Applying Operations to Vectors, Matrices:

- It is a common scenario when we need to apply a common operation to multiple vector elements. This can be done by defining a lambda and then vectorizing the same.

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

mul_5 = lambda x: x * 5
vectorized_mul_5 = np.vectorize(mul_5)

vectorized_mul_5(matrix)
```

```
vectorized_mul_5(matrix)
```

```
array([[ 5, 10, 15],
       [20, 25, 30],
       [35, 40, 45]])
```

```
matrix * 5
```

Arrays and Structure

- **Mean, Variance and Standard Deviation**

With NumPy, it is easy to perform operations related to descriptive statistics on vectors.

Mean of a vector:

```
np.mean(matrix)
```

```
np.mean(matrix)
```

```
5.0
```

Variance of a vector:

```
np.var(matrix)
```

```
np.var(matrix)
```

```
6.666666666666667
```

Standard deviation of a vector:

```
np.std(matrix)
```

```
np.std(matrix)
```

```
2.581988897471611 2
```

Arrays and Structure

❑ Transposing a Matrix

Transposing is a very common operation which you will hear about whenever you are surrounded by matrices. Transposing is just a way to swap columnar and row values of a matrix.

```
matrix.T
```

```
matrix.T
```

```
array([[1, 4, 7],
       [2, 5, 8],
       [3, 6, 9]])
```

❑ Flattening a Matrix

We can convert a matrix into a one-dimensional array if we wish to process its elements in a linear fashion.

```
matrix.flatten()
```

```
matrix.flatten()
```

```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Arrays and Structure

- **Eigendecomposition: Eigenvalues and Eigenvectors**

Eigenvectors are very commonly used in Machine Learning packages. So, when a linear transformation function is presented as a matrix, then X, Eigenvectors are the vectors that change only in scale of the vector but not its direction.

```
Xv = γv
```

Here, X is the square matrix and γ contains the Eigenvalues. Also, v contains the Eigenvectors. With NumPy, it is easy to calculate Eigenvalues and Eigenvectors.

```
evalues, evectors = np.linalg.eig(matrix)
```

```
evalues, evectors = np.linalg.eig(matrix)
```

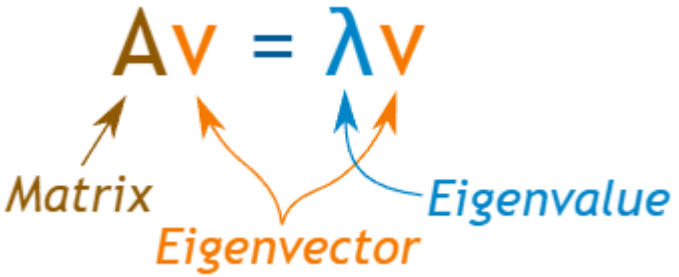
```
evalues
```

```
array([  1.61168440e+01,  -1.11684397e+00,  -3.73313677e-16])
```

```
evectors
```

```
array([[-0.23197069, -0.78583024,  0.40824829],
       [-0.52532209, -0.08675134, -0.81649658],
       [-0.8186735 ,  0.61232756,  0.40824829]])
```

Arrays and Structure



❑ Eigendecomposition: Data Science

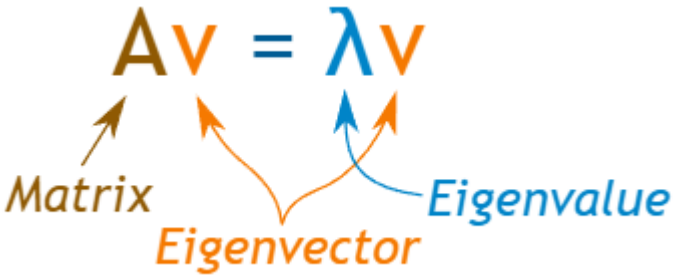
- Suppose there are a string of assets whose value needs to be assessed based on the risk factors. These risk factors usually are correlated. To ensure profitability with a tolerable level of risk is the primary goal of a risk analyst. To calculate the correlation and risk tolerance, the analyst might use a covariance matrix. Switching to maths might sound abrupt but, that's how things are done in the real world. A college level maths concept to tackle million dollar problem.

- Eigendecomposition is used to decompose a matrix into eigenvectors and eigenvalues which are eventually applied in methods used in machine learning, such as in the Principal Component Analysis method or PCA. Decomposing a matrix in terms of its eigenvalues and its eigenvectors gives valuable insights into the properties of the matrix.

- Machine learning involves lots of data. If in any way we are able to reduce the data size without losing actual representation of those values then its a piece of pie for programmers.

- Principal Component Analysis or PCA is performed for dimensionality reduction. With larger datasets, finding significant features gets difficult. So, in order to check for the correlation between two variables and if they could be dropped off the table to make the machine learning model more robust.

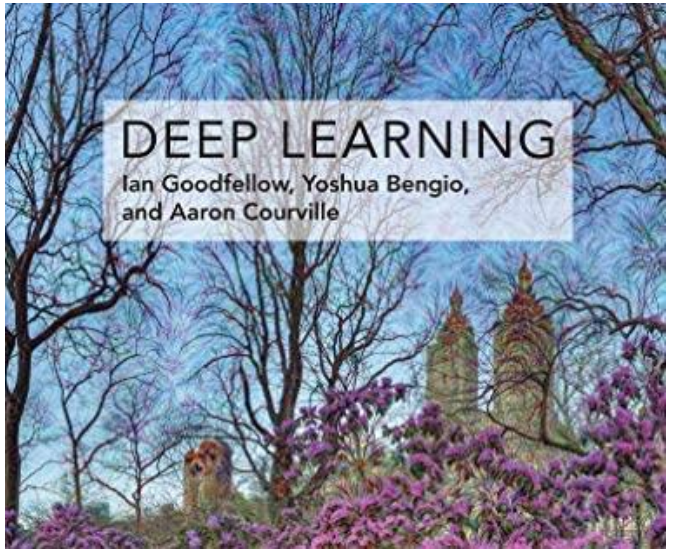
Arrays and Structure



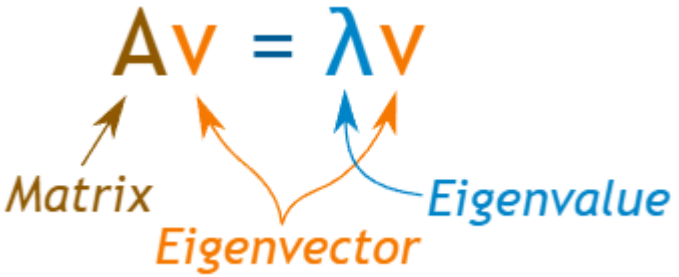
- **Eigendecomposition: Data Science**

- In order to understand a thing or problem in a better manner, we tend to break down things into smaller components and understand the things' properties by understanding these smaller components. When we break down things into their most elementary components or basic elements, we can get a great understanding of the things.
- For example, if we want to understand a wooden table, we can understand it in a better manner by understanding the basic elements, wood, of which it is made. We can then understand the properties of wooden tables in a better manner.
- Matrices can be broken down or decomposed in ways that can show information about their functional properties that are not obvious from the representation of the matrix as an array of elements. One of the most widely used kinds of matrix decomposition is called eigen-decomposition, in which we decompose a matrix into a set of eigenvectors and eigenvalues.

You may want to check out this page from *Deeplearning book* by Ian Goodfellow, Yoshua Bengio, and Aaron C.
By the way, this reasoning technique of breaking down things and arriving at the most basic elements of which thing is made to understand and innovate is also called first principles thinking.



Arrays and Structure



- **Eigendecomposition for Applications**

1. System of Communication: Claude Shannon utilized eigenvalues to calculate the theoretical limit of how much information can be carried via a communication channel. The eigenvectors and eigenvalues of the communication channel (represented as a matrix) are calculated. The eigenvalues are then essentially the gains of the channel's fundamental modes, which are recorded by the eigenvectors.

2. Bridge Construction: The smallest magnitude eigenvalue of a system that models the bridge is the natural frequency of the bridge. Engineers use this knowledge to guarantee that their structures are stable.

3. Automobile Stereo System Design: Eigenvalue analysis is also employed in the design of car stereo systems, where it aids in the reproduction of car vibration caused by music.

4. Electrical Engineering: The use of eigenvalues and eigenvectors to decouple three-phase systems via symmetrical component transformation is advantageous.

5. Mechanical Engineering: Eigenvalues and eigenvectors enable us to "decompose" a linear process into smaller, more manageable tasks. The eigenvectors in the principle directions are the eigenvectors, and the associated eigenvalue is the percentage deformation in each principle direction.

6. Eigenvalue analysis is commonly used by oil firms to explore land for oil: Because oil, dirt, and other substances all produce linear systems with varying eigenvalues, eigenvalue analysis can help pinpoint where oil reserves lie. The waves are modified when they move through the different substances in the earth. The oil corporations are directed to possible drilling sites based on the study of these waves.

Arrays and Structure

❑ Slicing of a Matrix

Slicing of a one-dimensional NumPy array is similar to a list.

```
import numpy as np
letters = np.array([1, 3, 5, 7, 9, 7, 5])

# 3rd to 5th elements
print(letters[2:5])        # Output: [5, 7, 9]

# 1st to 4th elements
print(letters[:-5])        # Output: [1, 3]

# 6th to last elements
print(letters[5:])         # Output:[7, 5]

# 1st to last elements
print(letters[:])          # Output:[1, 3, 5, 7, 9, 7, 5]

# reversing a list
print(letters[::-1])           # Output:[5, 7, 9, 7, 5, 3, 1]
```

Arrays and Structure

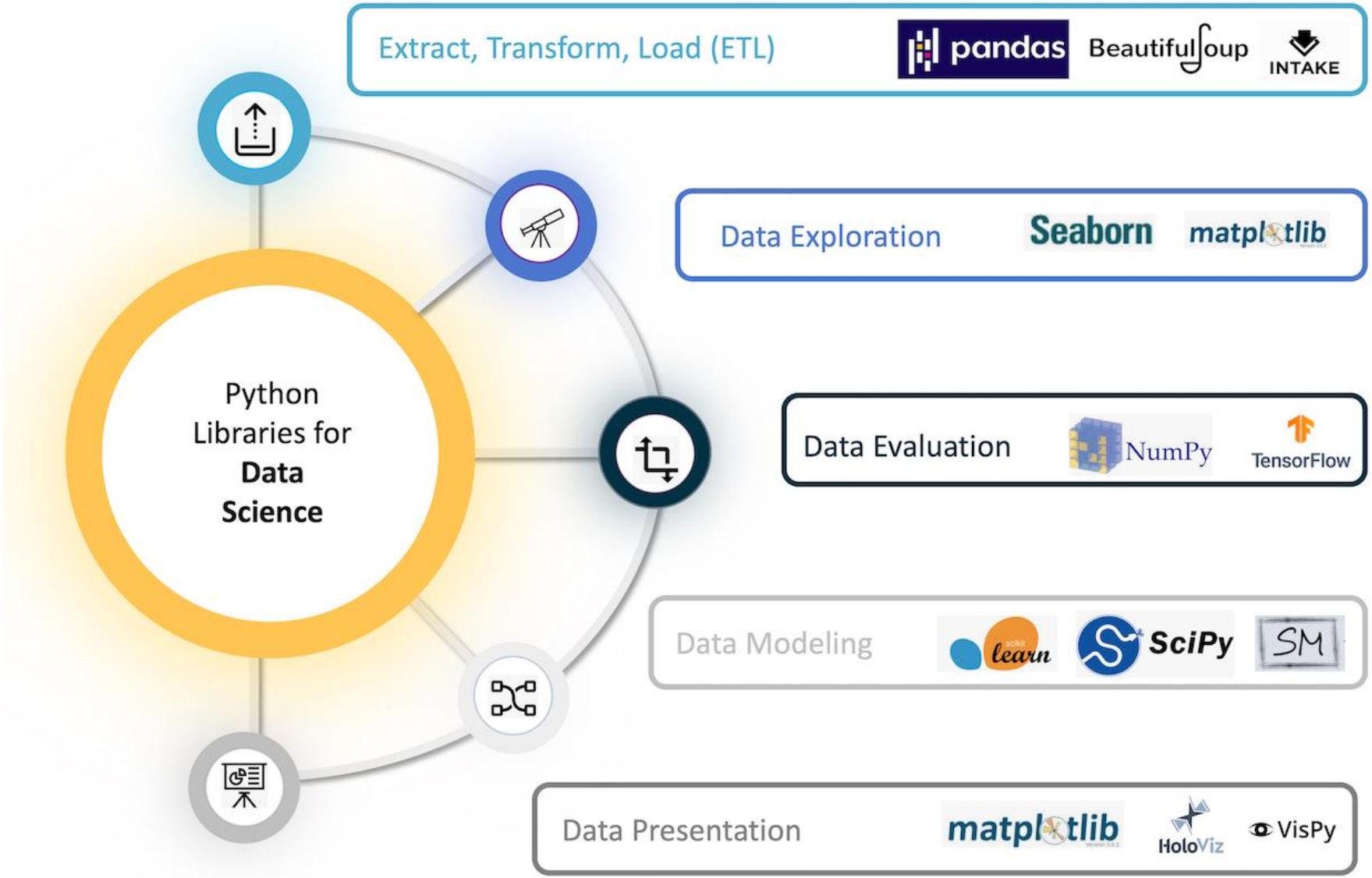
❑ Slicing of a Matrix

Slicing of a one-dimensional NumPy array is similar to a list.

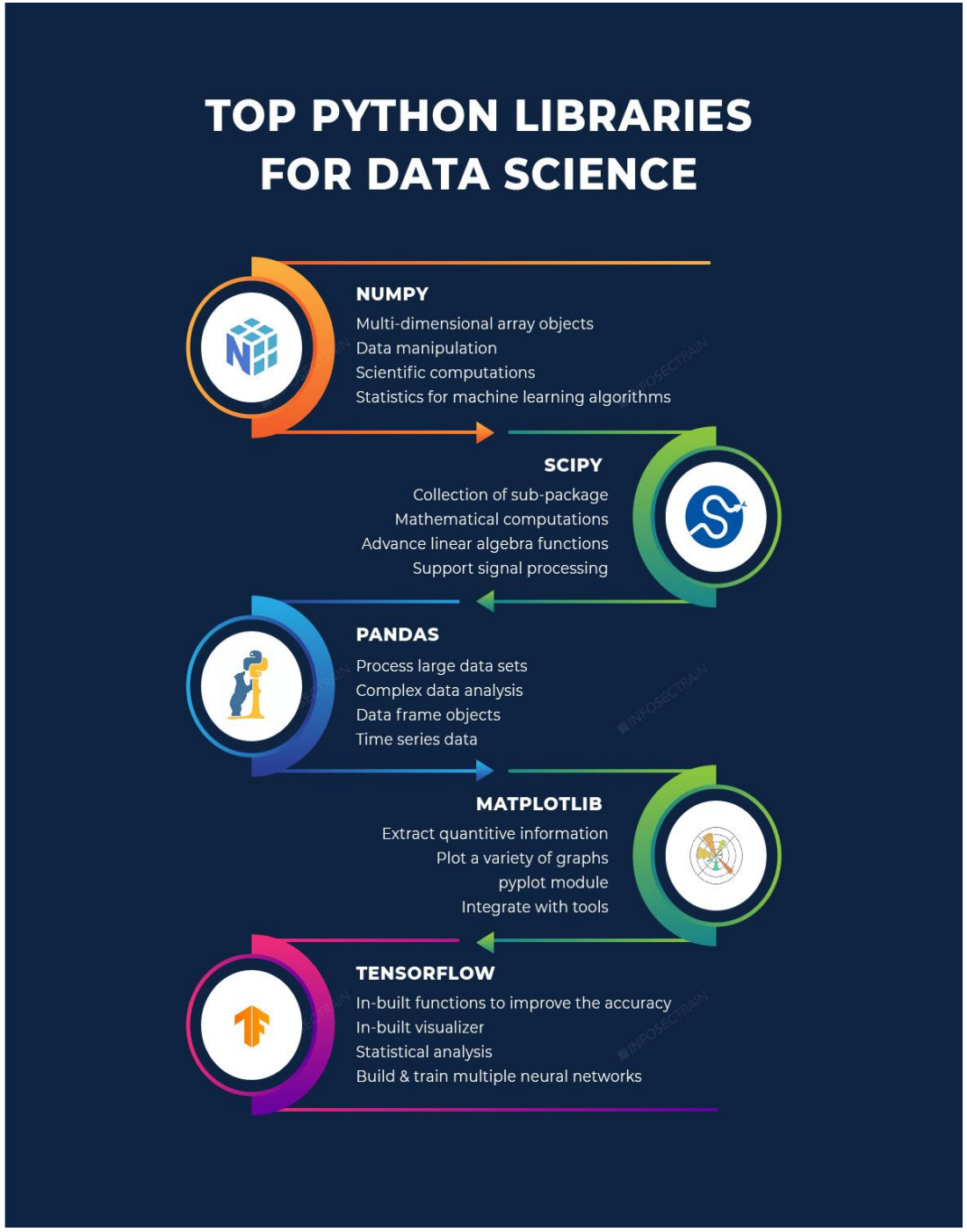
```
import numpy as np
A = np.array([[1, 4, 5, 12, 14], [-5, 8, 9, 0, 17], [-6, 7, 11, 19, 21]])

print(A[:2, :4]) # two rows, four columns
''' Output:
[[ 1 4 5 12] [-5 8 9 0]] '''

print(A[:1,]) # first row, all columns
''' Output:
[[ 1 4 5 12 14]] '''

print(A[:,2]) # all rows, second column
''' Output:
[ 5 9 11] '''

print(A[:, 2:5]) # all rows, third to the fifth column
'''Output:
[[ 5 12 14] [ 9 0 17] [11 19 21]] '''
```

Python for Data Science

.



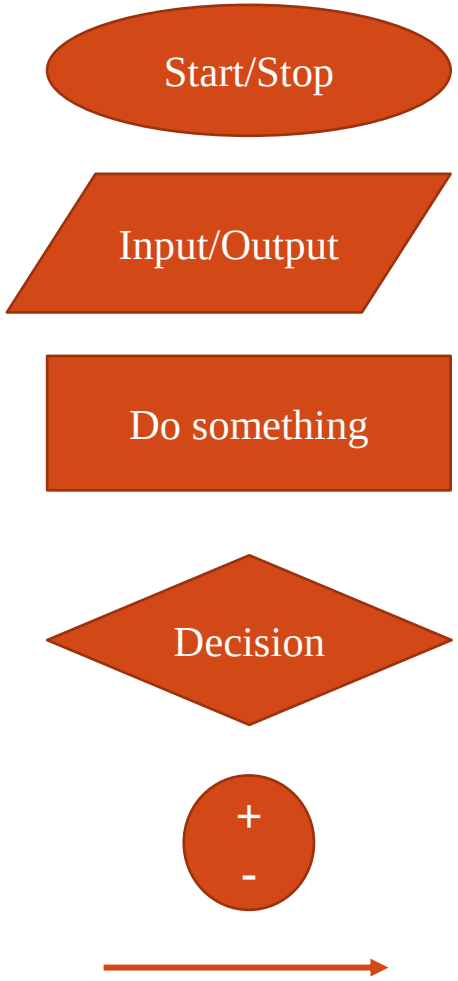
Python for Data Science



Bài tập thực hành

Viết một chương trình thống kê và tính toán, xuất nhập dữ liệu dạng mảng (array) từ bảng dữ liệu đã được cung cấp. (*theo bảng số liệu kết quả thể thao trong slide*)

- Tính tổng số huy chương vàng theo quy luật
  (2 Silver = 1 Gold, 3 Bronze = 1 Gold)
- Sắp xếp thứ hạng các quốc gia (theo số huy chương vàng đã tính)
- Phân tích dữ liệu, đánh giá phổ kết quả theo số medals của các quốc gia
- Plot biểu đồ theo các đánh giá phổ kết quả, sử dụng bar chart, pie chart, histogram
- Xuất danh sách thứ hạng, phân tích dữ liệu, xuất file và lưu file dữ liệu.



Bài tập thực hành

- **Xây dựng flow chart (algorithm):**
  - Xây dựng bảng dữ liệu

    → file excel → file .txt
  - Load file dữ liệu ndarray, truy xuất dữ liệu vào chương trình
  - Thống kê, Tính toán, So sánh, Phân tích:
    - Tính số Gold mỗi quốc gia theo quy luật đã cho
    - So sánh số Gold các quốc gia để sắp xếp thứ hạng
  - Tạo list danh sách các thứ hạng từ cao đến thấp theo số medals
  - Phân tích số liệu để đánh giá kết quả tham gia thể thao
  - Plot các phổ kết quả sau khi được phân tích, đánh giá
  - Tổng hợp, xuất dữ liệu đã tính toán, phân tích
  - Tạo file dữ liệu (file .txt), save as vào máy tính
  - Kết thúc