SVILUPPO RISORSE PER L'AZIENDA
CISITA
PARMA

Unione Parmense degli Industriali

GIA Gruppo Imprese
Gruppo Imprese Artigiane

Data analysis with Python

Internet of Things e Analisi predittiva

Gianfranco Lombardo MEng, Ph.D Candidate in ICT
gianfranco.lombardo@unipr.it

Dictionary

- Data structure for key-value items
  - my_dict = { “Alice” : “Boss”
  - “George”: “Project manager”
  - “Mario” : “Developer” }
- Also known as “ associative arrays” or map
- Unordered set of key / value pairs
  - Keys are unique as indices in lists and can be used to access the associated value
  - Keys are usually string, but can be also int or any immutable type

Dictionary

Py

```
>>> tel = {}                          #Initialize an empy dict called tel
>>> tel["mario"] = "052-87653"    #add “Mario” key with his value
>>> print(tel)
{'mario': '052-87653'}

>>> my_dict = {"George":"051-43235"}  #initialize my_dict with one value
>>> my_dict["Mario"] = "052-87653"     #add an item to my_dict
>>> print(my_dict)
{'George': '051-43235', 'Mario': '052-87653'}
```

Iterate over a dictionary

```
my_dict = {"George":"051-43235", “Alice”: “053-74123”,”Bob”:”051-23619”}
for key in my_dict:
        #remember that dictionaries are unordered!
        if key==”George”:
                print(“George number is “+str(my_dict[key]))

my_dict[“George”] = “3409823654”      #change value

my_dict.pop(“Alice”)                          #remove Alice from the dictionary
```

Matrix

- A matrix is an ordered tables of elements
  - Example: Chessboard, Algebra matrices, datasets
- We can see as multidimensional lists!
- Basically a main list with inside other lists with the same number of elements

```
a = [ ['A', 'B', 'C', 'D'],
      ['E', 'F', 'G', 'H'],
      ['I', 'L', 'M', 'N'] ]
```

- To access an element we need two indices:
  - row y
  - column x
  - matrix[y][x]

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ \vdots \\ m \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & \dots & n \end{array} \\ \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ a_{31} & a_{32} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \end{array}$$

Sum of columns in a matrix

Py

```
matrix = [[2, 4, 3, 8],
          [9, 3, 2, 7],
          [5, 6, 9, 1]]
rows = len(matrix)                  # get number of rows
cols = len(matrix[0])               # get number of columns

for x in range(cols):
    total = 0
    for y in range(rows):
        val = matrix[y][x]
        total += val
    print("Column "+str(x)+ " sums to "+str(total))
```

Flatten matrix as a list

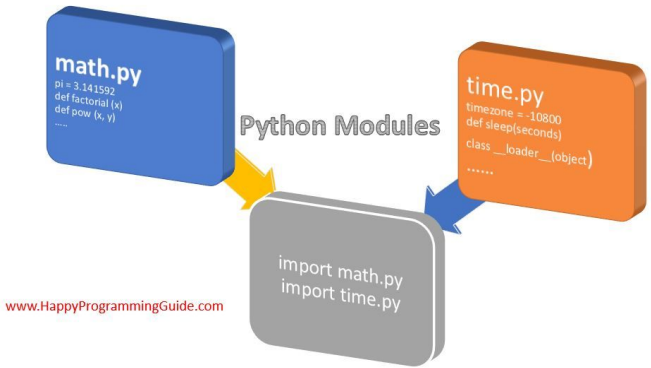
Py

```
matrix = [2, 4, 3, 8,
          9, 3, 2, 7,
          5, 6, 9, 1]
rows = 3  # Cannot be guessed from matrix alone
cols = len(matrix) // rows

for x in range(cols):
    total = 0
    for y in range(rows):
        val = matrix[y * cols + x]    # 2D -> 1D
        total += val
        print("Col #", x, "sums to", total)
```

Python external modules

- Sometimes we need functionalities and operations that are not directly provided by Python
- Or we want to reuse some code already written by another developer
- In those cases we need to import a module in our code!
- Python is the most used language for Data science especially for the richness of different modules
- Examples:
  - Math : for mathematical operations
  - Numpy for matrix and vectors
  - Pandas for data mining
  - Matplotlib for data visualization
  - Scikit-learn for Machine learning
  - Keras and Tensor-flow for Deep learning



Math module

```
import math
y = math.sin(math.pi / 4)
print(y)   # √2 / 2
```

```
from math import sin, pi
print(sin(pi / 4))
```

```
from random import randint
die1 = randint(1, 6)   # like rolling a die
die2 = randint(1, 6)   # like rolling a die
```

How to install a module

- Most of the modules have to be installed
- PIP is the tool already installed with Python that permits to install new modules!
- example: pip install “name of modules”
  - pip install numpy

NumPy

- Efficient library that provides multidimensional array and algorithms
- It is the base of most of the machine learning and data science libraries
- Implemented in Fortran, C, C++
- To use after installation:
  - `import numpy as np`
- Define a vector (or array) in numpy:
  - `a = np.array([2, 3, 4])`

Zeros, ones, ranges

```
>>> np.zeros(4)
array([0, 0, 0, 0])

>>> np.ones((2, 3), dtype=np.int16)  # dtype can also be specified
array([[[1, 1, 1],
        [1, 1, 1]], dtype=int16)

>>> np.empty((2, 3))  # uninitialized, output may vary
array([[3.73603959e-262, 6.02658058e-154, 6.55490914e-260],
       [5.30498948e-313, 3.14673309e-307, 1.00000000e+000]])

>>> np.arange(10, 30, 5)  # from 10, to 30, step 5
array([10, 15, 20, 25])

>>> np.linspace(0, 2, 5)  # 5 numbers from 0 to 2
     array([0, 0.5, 1, 1.5, 2])
```

Reshape and resize

```
>>> a = np.arange(12)  # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
>>> a = a.reshape(3, 4)  # a new array
>>> a
array([[0, 1, 2, 3],
       [4, 5, 6, 7],
       [8, 9, 10, 11]])
>>> a.shape      # Shape gets number of elements along each axes
(3, 4)

>>> a.resize(2, 6)  # modifies the array itself
>>> a
array([[0, 1, 2, 3, 4, 5],
           [6, 7, 8, 9, 10, 11]])
```

Shuffle and choice

```
>>> a = np.arange(6)
>>> np.random.shuffle(a)   # modifies the array itself
array([5, 3, 2, 4, 1, 0]])

>>> np.random.choice(["A","B"]) #Choose randomly an element in a list
>>> "B"

>>>np.random.choice(["one", "two"], (2, 3))
>>> array([['two', 'one', 'one'],
       ['two', 'two', 'one']])


#Generate a matrix by choosing randomly elements in the list
```

Elementwise operations

- A new array holds the result

```
>>> a = np.array([20, 30, 40, 50])
>>> conditions= a < 35
array([True, True, False, False])

>>> b = np.arange(4)
>>> b_square = b ** 2
array([0, 1, 4, 9])
>>> c = a - b
array([20, 29, 38, 47])
```

Aggregate functions

```
>>> b = np.arange(12).reshape(3, 4)
>>> b
array([[0, 1, 2, 3],
       [4, 5, 6, 7],
       [8, 9, 10, 11]])
>>> b.sum()                    # guess also min and max
66
>>> b.sum(axis=1)              # sum of each row
array([6, 22, 38])
>>> b / b.max(axis=0)          # norm each column
array([[0., 0.11, 0.2, 0.27],
       [0.5, 0.55, 0.6, 0.63],
           [1., 1., 1., 1.  ]])
```

Indexing and slicing

```
b = np.array([[0, 1, 2, 3],
       [10, 11, 12, 13],
       [20, 21, 22, 23]])
>>> b[2, 3]
23
>>> b[:, 1]  # each row in the second column of b
array([1, 11, 21])
>>> b[1:3, 0:2]
array([[10, 11],
           [20, 21]])
```

CSV files

- CSV files (Comma separated values) is one of the most common file format in data science
  - It requires that the first line is an header where each field is separated with a comma
  - The other lines are values separated by a comma
  - Basically, it represents always a table
  - Available also in Microsoft Excel

- It is possible read a CSV file with operators that with the Python open() by splitting each line for commas or other ad-hoc modules are available like csv or Pandas (later in this lesson…)

CSV EXAMPLE

```
Name,Age,Job,City                    #Header
George,34,Waiter,Chicago
Alice,27,Developer,New York
Mario,57,Plumber, Rome
Lauren,42,Teacher,Detroit
Robert,29,Engineer,London
```

CSV module

```
import csv
matrix = []
with open('people.csv',
newline='') as f:
    reader = csv.reader(f)
    for row in reader:
        matrix.append(row)
print(matrix)

with open('people_2.csv', 'w',
newline='') as f:
    writer = csv.writer(f)
    for row in matrix:
        writer.writerow(row)
```

```
###### without module
matrix = []
with open('people.csv', newline='')
as f:
    reader = f.readlines()
    for line in reader:
        row =
line.replace("\r\n","").split(",")
        matrix.append(row)
print(matrix)


.....
```

Exercise:

- Ask to the user to add 10 people information:
  - Name
  - ID
  - Age
- Save these information in a dictionary where the key is always the ID and the value is a list [name,age]
- Then ask again to add more information for these people by asking the ID and add to each one:
  - Salary
  - Years of work
- Finally, export the entire dictionary into registry.csv file
  - header: Name,ID,Age,Salary,Years of work

Exercise

- Read the previous registry.csv
  - Skip the first row and save as a separate list called headers
  - Memorize the other rows in a matrix
  - Then ask to the user the name of a field:
    - Retrieve which column has to be considered from headers
    - Compute min and max for that column

Chessboard

- Generate a zero vector with dimension 64 as a pseudo matrix
  - Zero means black
  - One means white
- Iterate over the matrix with two “for” and put “1” if the column is pair (c%2 == 0)
- Transform the list as numpy array and then reshape as a 8x8 matrix
- Print the matrix

BREAK