pythonsamples Documentation

Release

Colathur Vijayan

Sep 04, 2017

Contents

CHAPTER 1

Introduction

Python is an exhilarating language to code and the power of what can be expressed by its libraries can sometimes appear almost magical when one looks at what a few lines of code can produce. In the current document I have endeavoured to provide samples of some of its constructs that I found interesting in the form of simple examples that demonstrate how handy they can be.

For the purpose of consistency, all examples will be using **Python 3** .

This column is written in the spirit of a **Work in Progress !!!** that will evolve over a period of time.

CHAPTER 2

zip

zip is an `iterable object` that provides an extremely useful technique to pack 2 different sequences that are paired based on their positions and create tuples that bring in these pairs together. For instance,

```
>>> Names = ['John','Jeanne', 'Roberto', 'Michelangelo']
>>> Citizenship = ['US','FR','MX', 'IT']
>>> for x,y in zip(Names, Citizenship):
...     print(x + ' belongs to ' + y)
...
John belongs to US
Jeanne belongs to FR
Roberto belongs to MX
Michelangelo belongs to IT
```

CHAPTER 3

enumerate

enumerate is an `iterable object` that provides an extremely useful technique to create a dictionary from a list whose keys are the list indexes. One of the ways this can be helpful is when you use matplotlib to plot, where in the xticks correspond to numbers, but the label (that appears on the X-axis) needs to be the 2 character State Code.

```
>>> State = ['AK', 'AL', 'AR', 'AZ', 'CA', 'CO', 'CT', 'DC', 'DE', 'FL', 'GA', 'HI',
       'IA', 'ID', 'IL', 'IN', 'KS', 'KY', 'LA', 'MA', 'MD', 'ME', 'MI', 'MN',
       'MO', 'MS', 'MT', 'NC', 'ND', 'NE', 'NH', 'NJ', 'NM', 'NV', 'NY', 'OH',
       'OK', 'OR', 'PA', 'RI', 'SC', 'SD', 'TN', 'TX', 'UT', 'VA', 'VT', 'WA',
       'WI', 'WV', 'WY']
>>> T = dict(enumerate(State))
>>> T
{0: 'AK', 1: 'AL', 2: 'AR', 3: 'AZ', 4: 'CA', 5: 'CO', 6: 'CT', 7: 'DC', 8: 'DE', 9:
↪'FL', 10: 'GA', 11: 'HI', 12: 'IA',
 13: 'ID', 14: 'IL', 15: 'IN', 16: 'KS', 17: 'KY', 18: 'LA', 19: 'MA', 20: 'MD', 21:
↪'ME', 22: 'MI', 23: 'MN', 24: 'MO',
 25: 'MS', 26: 'MT', 27: 'NC', 28: 'ND', 29: 'NE', 30: 'NH', 31: 'NJ', 32: 'NM', 33:
↪'NV', 34: 'NY', 35: 'OH', 36: 'OK',
 37: 'OR', 38: 'PA', 39: 'RI', 40: 'SC', 41: 'SD', 42: 'TN', 43: 'TX', 44: 'UT', 45:
↪'VA', 46: 'VT', 47: 'WA', 48: 'WI',
 49: 'WV', 50: 'WY'}
```

CHAPTER 4

map

map is an `iterable object` that provides an extremely useful technique to apply a function on all elements of a list. For instance, I have found this very handy where in I want a bunch of consecutive Integers that represent a Year to be column indexes of a dataframe. What is happening here is that a range of consecutive integers starting from 2006 and ending at 2015, have all been converted into a list of strings, using map.

```
>>> list(map(str, range(2006, 2016)))
['2006', '2007', '2008', '2009', '2010', '2011', '2012', '2013', '2014', '2015']
```