python

Python Programming

Classes and Web Scraping

UMKC

Feedback is greatly appreciated!

Objective

- Classes
- Object Oriented Concepts
- Instances
- Init
- self
- Private, Protected, Public
- Inheritance
- Scientific Python
- Numpy Package
- Web scraping

Defining a Class

- A *class* is wrapping up of data and functions into one unit.
- It is a software item which contains variables and methods.
- The *class* also stores some data items that are shared by all the instances of the class
- *Objects* are *Instances* of a class

Methods in Classes

- Define a *method* in a *class* by including function definitions within the scope of the class block
- There is usually a special method called `__init__` in most classes. It is called the default constructor.

A simple class def: *student*

__init__ is the default constructor

```
class student:
  """A class representing a
  student """
  def __init__(self,n,a):
      self.full_name = n
      self.age = a
  def get_age(self):
      return self.age
```

self refers to the object itself, like *this* in Java.

Object Oriented Concepts

- Object Oriented Design focuses on
  - Encapsulation:
    - dividing the code into a public interface, and a private implementation of that interface
  - Polymorphism:
    - the ability to overload standard operators so that they have appropriate behavior based on their context
  - Inheritance:
    - the ability to create subclasses that contain specializations of their parents

It's all objects…

- Everything in Python is really an object.
  - We've seen hints of this already…
    "hello".upper()
    list3.append('a')
    dict2.keys()
  - These look like Java or C++ method calls.
  - New object classes can easily be defined in addition to these built-in data-types.
- In fact, programming in Python is typically done in an object oriented fashion.

Creating and Deleting Instances

Instantiating Objects

- Just use the class name with ( ) notation and assign the result to a variable
- __init__ serves as a constructor for the class. Usually does some initialization work
- The arguments passed to the class name are given to its __init__() method
- So, the __init__ method for student is passed "Bob" and 21 and the new class instance is bound to b:

```
b = student("Bob", 21)
```

Constructor: __init__

- An __init__ method can take any number of arguments.
- Like other functions or methods, the arguments can be defined with default values, making them optional to the caller.

- However, the first argument self in the definition of __init__ is special…

Self

- The first argument of every method is a reference to the current instance of the class
- By convention, we name this argument ***self***
- In __init__, *self* refers to the object currently being created; so, in other class methods, it refers to the instance whose method was called
- Similar to the keyword *this* in Java or C++

Self

- Although you must specify `self` explicitly when *defining* the method, you don't include it when *calling* the method.
- Python passes it for you automatically

Defining a method:

(this code inside a class definition.)

```
def set_age(self, num):
    self.age = num
```

Calling a method:

```
>>> x.set_age(23)
```

Deleting instances: No Need to “free”

- When you are done with an object, you don’t have to delete or free it explicitly.
- Python has automatic garbage collection.
- Python will automatically detect when all of the references to a piece of memory have gone out of scope and Automatically frees that memory.
- Generally works well, few memory leaks

Definition of student

```
class student:
  """A class representing a student
  """
  def __init__(self,n,a):
      self.full_name = n
      self.age = a
  def get_age(self):
      return self.age
```

https://www.csee.umbc.edu/courses/691p/notes/python/python3.ppt

Syntax for Access

```
>>> f = student("Bob Smith", 23)

>>> f.full_name # Access attribute
"Bob Smith"

>>> f.get_age() # Access a method
23
```

Two Kinds of Attributes

- The non-method data stored by objects are called attributes
- *Data* attributes
  - Variable owned by a *particular instance* of a class
  - Each instance has its own value for it
  - These are the most common kind of attribute
- *Class* attributes
  - Owned by the *class as a whole*
  - *All class instances share the same value for it*
  - Called "static" variables in some languages
  - Good for (1) class-wide constants and (2) building counter of how many instances of the class have been made

Data Attributes

- Data attributes are created and initialized by an __init__() method.
  - Simply assigning to a name creates the attribute
  - Inside the class, refer to data attributes using **self**
    - for example, **self.full_name**

```
class teacher:
    "A class representing teachers."
    def __init__(self,n):
        self.full_name = n
    def print_name(self):
        print self.full_name
```

Class Attributes

- Because all instances of a class share one copy of a class attribute, when *any* instance changes it, the value is changed for *all* instances
- Class attributes are defined *within* a class definition and *outside* of any method
- Since there is one of these attributes *per class* and not one *per instance*, they're accessed via a different notation:
  - Access class attributes using **self.__class__.name** notation –
  - This is just one way to do this & the safest in general.

```
class sample:
    x = 23
    def increment(self):
        self.__class__.x += 1
```

```
>>> a = sample()
>>> a.increment()
>>> a.__class__.x
24
```

Data vs. Class Attributes

```
class counter:
   overall_total = 0
        # class attribute
   def __init__(self):
       self.my_total = 0
         # data attribute
   def increment(self):
      counter.overall_total = counter.overall_total + 1

      self.my_total = self.my_total + 1
```

```
>>> a = counter()
>>> b = counter()
>>> a.increment()
>>> b.increment()
>>> b.increment()
>>> a.my_total
1
>>> a.__class__.overall_total
3
>>> b.my_total
2
>>> b.__class__.overall_total
3
```

Inheritance

Inheritance

- Basic syntax for a derived class definition:

```
class DerivedClassName(BaseClassName):
  <statement-1>
  . . .
   <statement-N>
```

- As stated before, all methods are virtual by default
  - If a method in DerivedClassName above has the same name and parameters as BaseClassName, the method in the derived class will be implemented when its called

https://www.cs.drexel.edu/~knowak/cs265_fall_2009/Python_Classes_nb.ppt

UMKC

Subclasses

- A class can *extend* the definition of another class
  - Allows use (or extension) of methods and attributes already defined in the previous one.
  - New class: *subclass*. Original: *parent*, *ancestor* or *superclass*
- To define a subclass, put the name of the superclass in parentheses after the subclass's name on the first line of the definition.

  Class Cs_student(student):
  - Python has no 'extends' keyword like Java.
  - Multiple inheritance is supported.

Redefining Methods

- To *redefine a method* of the parent class, include a new definition using the same name in the subclass.
  - The old code won't get executed.
- To execute the method in the parent class *in addition to* new code for some method, explicitly call the parent's version of the method.

  parentClass.methodName(self, a, b, c)
  - **The only time you ever explicitly pass 'self' as an argument is when calling a method of an ancestor.**

https://www.cs.drexel.edu/~knowak/cs265_fall_2009/Python_Classes_nb.ppt

UMKC

Definition of a class extending student

```
Class Student:
   "A class representing a student."

   def __init__(self,n,a):
       self.full_name = n
       self.age = a

   def get_age(self):
       return self.age
-----------------------------------------------
Class Cs_student (student):
   "A class extending student."

   def __init__(self,n,a,s):
       student.__init__(self,n,a) #Call __init__ for student
       self.section_num = s

   def get_age(self):    #Redefines get_age method entirely
       print "Age: " + str(self.age)
```

Passing another class as parent

Private Data and Methods

- Any attribute/method with 2 leading under-scores in its name (but none at the end) is **private** and can't be accessed outside of class

- Note: Names with two underscores at the beginning ***and the end*** are for built-in methods or attributes for the class

https://www.cs.drexel.edu/~knowak/cs265_fall_2009/Python_Classes_nb.ppt

UMKC

Private,Protected and Public

Name	Notation	Behaviour
name	Public	Can be accessed from inside and outside
_name	Protected	Like a public member, but they shouldn't be directly accessed from outside.
__name	Private	Can't be seen and accessed from outside

https://www.cs.drexel.edu/~knowak/cs265_fall_2009/Python_Classes_nb.ppt

Example

```
class MyClass():
    def __init__(self):
            self.__superprivate = "Hello"
            self._semiprivate = "world!"

mc = MyClass()
print (mc.__superprivate)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: myClass instance has no attribute '__superprivate'
print (mc._semiprivate)
      world!
print (mc.__dict__)
{'_MyClass__superprivate': 'Hello', '_semiprivate': ', world!'}
```

Use case 1- Bank Account

```python
import datetime

history = {}


class Account(object):
    def __init__(self):
        self.pin = 1234    # public variable

    def checkCode(self, value):
        return self.pin == value

    def checkBalance(self):
        print ('Current balance is ' + str(self.balance) + ' dollars.')

    def checkTransactions(self):
        print ('Previous transactions:')
        for item in sorted(history):
            print (item + '\t\t' + str(history[item]) + ' dollars.')
```

Class Declaration

Default Constructor

Member Functions

https://github.com/MiguelSOliveira/Python-Projects

```python
    def deposit(self, pin, value):
        if self.checkCode(pin):
            self.balance += value
            history[str(datetime.datetime.now())] = '+' + str(value)
            print ('Successfully deposited ' + str(value) + ' dollars from your account.')
        else:
            print ('Wrong pin code. Try again.')

    def createCreditCard(self, value):
        import random
        self.MBalance = value
        print ('You have Successfully created an Credit card with ' + str(self.MBalance) + ' dollars
        card_ID = [random.randint(0, 9) for _ in range(23)]
        print ('Your card id is ' + "".join(str(id) for id in card_ID))
        card_secret = [random.randint(0, 9) for _ in range(4)]
        print ('Your secret code is ' + "".join(str(secret) for secret in card_secret))

    def changePIN(self, oldvalue, newvalue):
        if self.checkCode(oldvalue):
            self.pin = newvalue
            print ('Pin code has been successfully changed.')
        else:
            print ('Wrong pin code. Try again.')
```

Other Functions of Account Class

https://github.com/MiguelSOliveira/Python-Projects

```
58  class CheckingAccount(Account):
59      def __init__(self):
60          super(CheckingAccount, self).__init__()
61          self.balance = 12000
62
63
64  class SavingsAccount(Account):
65      def __init__(self):
66          super(SavingsAccount, self).__init__()
67          self.balance = 5000
68
69
70  class BusinessAccount(Account):
71      def __init__(self):
72          super(BusinessAccount, self).__init__()
73          self.balance = 10000
74
75
```

Inheritance: "Account" is passed in class declaration

Output

```python
checksAcc = CheckingAccount()
print ('Enter pin:')
pin = input()
if checksAcc.checkCode(int(pin)):
    checksAcc.withdraw(1234, 4000)
    checksAcc.deposit(1234, 22000)
    checksAcc.withdraw(1234, 22000)
    checksAcc.deposit(1234, 24000)
    checksAcc.withdraw(1234, 2000)
    checksAcc.deposit(1234, 1000)
    checksAcc.withdraw(1234, 2000)
    checksAcc.deposit(1234, 9000)
checksAcc.checkTransactions()
checksAcc.checkBalance()

myAcc = SavingsAccount()
hisAcc = BusinessAccount()

checksAcc.createCreditCard(500)
checksAcc.changePIN(1234, 5678)
```

BankAccount

```
C:\python\python.exe C:/Users/Puchu/PycharmProjects/MyFirstPythonProject/part-5/BankAccount.py
Enter pin:
1234
Successfully withdrawn 4000 dollars from your account.
Successfully deposited 22000 dollars from your account.
Successfully withdrawn 22000 dollars from your account.
Successfully deposited 24000 dollars from your account.
Successfully withdrawn 2000 dollars from your account.
Successfully deposited 1000 dollars from your account.
Successfully withdrawn 2000 dollars from your account.
Successfully deposited 9000 dollars from your account.
Previous transactions:
2017-06-05 00:14:17.181082      +9000 dollars.
Current balance is 38000 dollars.
You have Successfully created an Credit card with 500 dollars.
Your card id is 17699317098649121539842
Your secret code is 8462
Pin code has been successfully changed.

Process finished with exit code 0
```

Instance Creation

Use Case 2 - Multiple Inheritance

```
class Clock(object):

    def __init__(self, hours, minutes, seconds):
        """
        The paramaters hours, minutes and seconds have to be
        integers and must satisfy the following equations:
        0 <= h < 24
        0 <= m < 60
        0 <= s < 60
        """

        self.set_Clock(hours, minutes, seconds)

    def set_Clock(self, hours, minutes, seconds):
        """
        The parameters hours, minutes and seconds have to be
        integers and must satisfy the following equations:
        0 <= h < 24
```

Clock Class

http://www.python-course.eu/python3_multiple_inheritance.php

```python
class Calendar(object):

    months = (31,28,31,30,31,30,31,31,30,31,30,31)
    date_style = "British"

    @staticmethod
    def leapyear(year):
        """
        The method leapyear returns True if the parameter year
        is a leap year, False otherwise
        """
        if not year % 4 == 0:
            return False
        elif not year % 100 == 0:
            return True
        elif not year % 400 == 0:
            return False
        else:
            return True


    def __init__(self, d, m, y):
```

Calendar Class

http://www.python-course.eu/python3_multiple_inheritance.php

```python
class CalendarClock(Clock, Calendar):
    """
        The class CalendarClock implements a clock with integrated
        calendar. It's a case of multiple inheritance, as it inherits
        both from Clock and Calendar
    """

    def __init__(self,day, month, year, hour, minute, second):
        Clock.__init__(self,hour, minute, second)
        Calendar.__init__(self,day, month, year)


    def tick(self):
        """
        advance the clock by one second
        """
        previous_hour = self._hours
        Clock.tick(self)
        if (self._hours < previous_hour):
            self.advance()

```

Calendar Class
Clock Class

We have created one class CalenderCLock to inherit both Clock and Calendar class features

Notice the super class constructor calling

http://www.python-course.eu/python3_multiple_inheritance.php

Output

```python
if __name__ == "__main__":
    x = CalendarClock(31,12,2013,23,59,59)
    print("One tick from ",x, end=" ")
    x.tick()
    print("to ", x)

    x = CalendarClock(28,2,1900,23,59,59)
    print("One tick from ",x, end=" ")
    x.tick()
    print("to ", x)

    x = CalendarClock(28,2,2000,23,59,59)
    print("One tick from ",x, end=" ")
    x.tick()
    print("to ", x)

    x = CalendarClock(7,2,2013,13,55,40)
    print("One tick from ",x, end=" ")
    x.tick()
    print("to ", x)
```

Run CalendarClock

```
C:\python\python.exe C:/Users/Puchu/PycharmProjects/MyFirstPythonProject/part-5/CalendarClock.py
One tick from  31/12/2013, 23:59:59 to  01/01/2014, 00:00:00
One tick from  28/02/1900, 23:59:59 to  01/03/1900, 00:00:00
One tick from  28/02/2000, 23:59:59 to  29/02/2000, 00:00:00
One tick from  07/02/2013, 13:55:40 to  07/02/2013, 13:55:41

Process finished with exit code 0
```

http://www.python-course.eu/python3_multiple_inheritance.php

Numpy
Web scraping

Scientific Python?

- Extra features required:
  - fast, multidimensional arrays
  - libraries of reliable, tested scientific functions
  - plotting tools
- NumPy is at the core of nearly every scientific Python application or module since it provides a fast N-d array datatype that can be manipulated in a vectorized form.

https://webvalley.fbk.eu/static/media/uploads/presentations/introductiontonumpy2.pdf

Scientific Python Packages

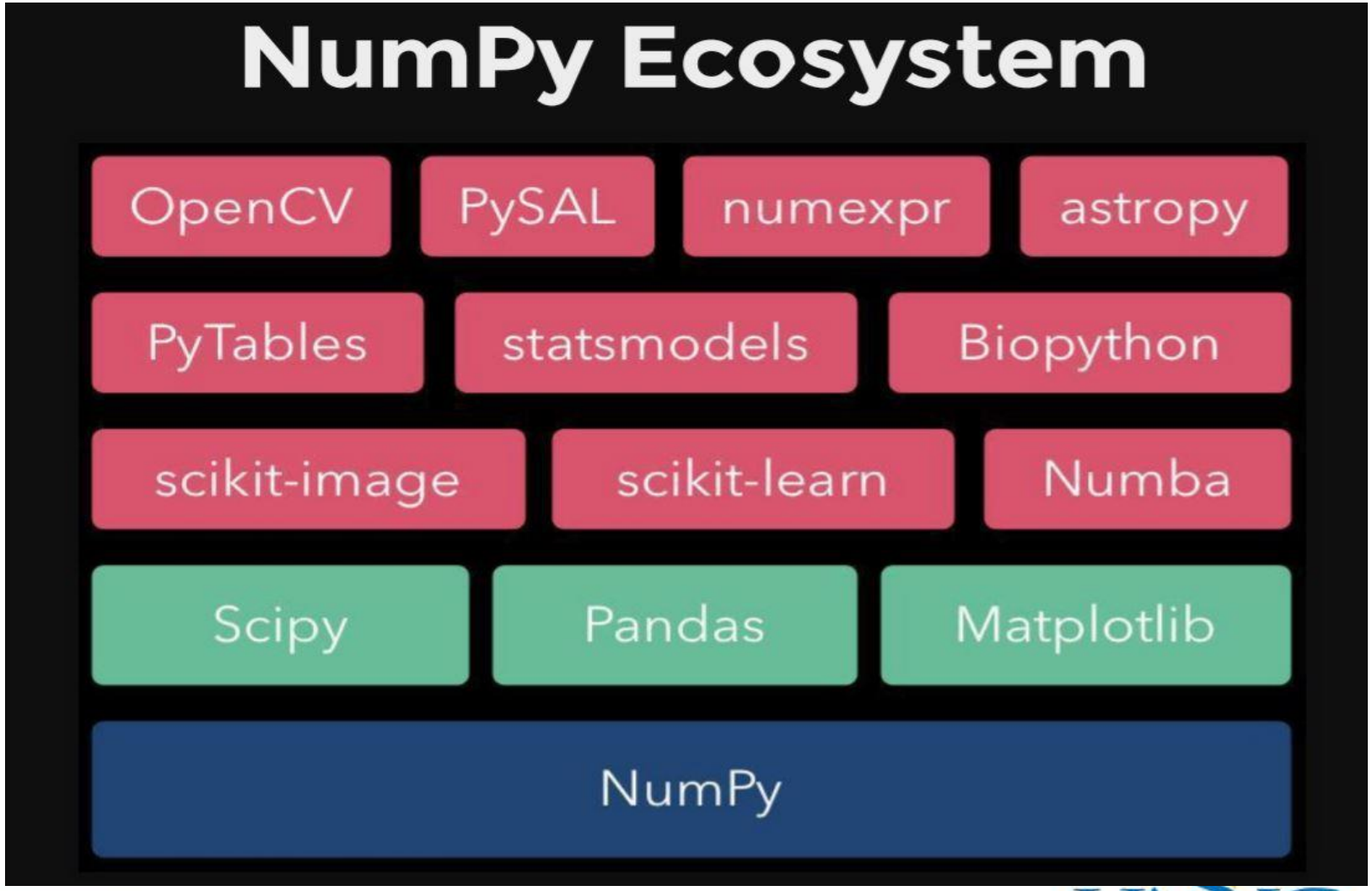
- **numpy** - *mainly* useful for its *N*-dimensional array objects
- **pandas** - Python data analysis library, including structures such as dataframes
- **matplotlib** - 2D plotting library producing publication quality figures
- **scikit-learn** - the machine learning algorithms used for data analysis and data mining tasks

https://webvalley.fbk.eu/static/media/uploads/presentations/introductiontonumpy2.pdf

UMKC

Numpy N-dimensional Array manipulations

- Fundamental package for scientific computing with Python
- N-dimensional array object
- Linear algebra, Fourier transform, random number capabilities
- Building block for other packages (e.g. Scipy)
- Open source



https://webvalley.fbk.eu/static/media/uploads/presentations/introductiontonumpy2.pdf

Arrays – Numerical Python (Numpy)

- Lists ok for storing small amounts of one-dimensional data

```
>>> a = [1,3,5,7,9]
>>> print(a[2:4])
[5, 7]
>>> b = [[1, 3, 5, 7, 9], [2, 4, 6, 8, 10]]
>>> print(b[0])
[1, 3, 5, 7, 9]
>>> print(b[1][2:4])
[6, 8]
```

```
>>> a = [1,3,5,7,9]
>>> b = [3,5,6,7,9]
>>> c = a + b
>>> print c
[1, 3, 5, 7, 9, 3, 5, 6, 7, 9]
```

- But, can't use directly with arithmetical operators (+, -, *, /, …)
- Need efficient arrays with arithmetic and better multidimensional tools
- **Numpy**

```
>>> import numpy
```

- Similar to lists, but much more capable, except fixed size

https://webvalley.fbk.eu/static/media/uploads/presentations/introductiontonumpy2.pdf

UMKC

Import numpy – Basic Operations

```python
import numpy as np

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

print(a)

b = a.reshape((3,3))

print(b)

c=b * 10 + 4

print(c)

Af = np.array([1, 2, 3], float)

print(Af)
```

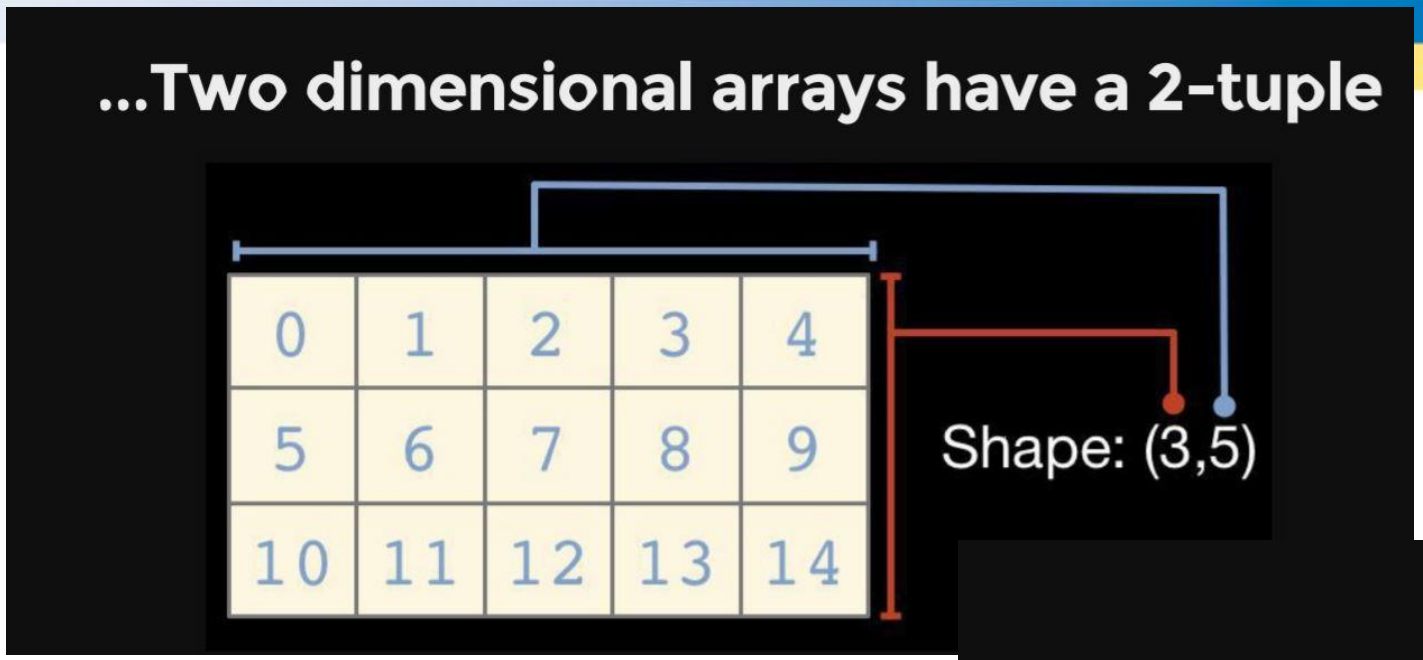
Run: cluster numpyEx

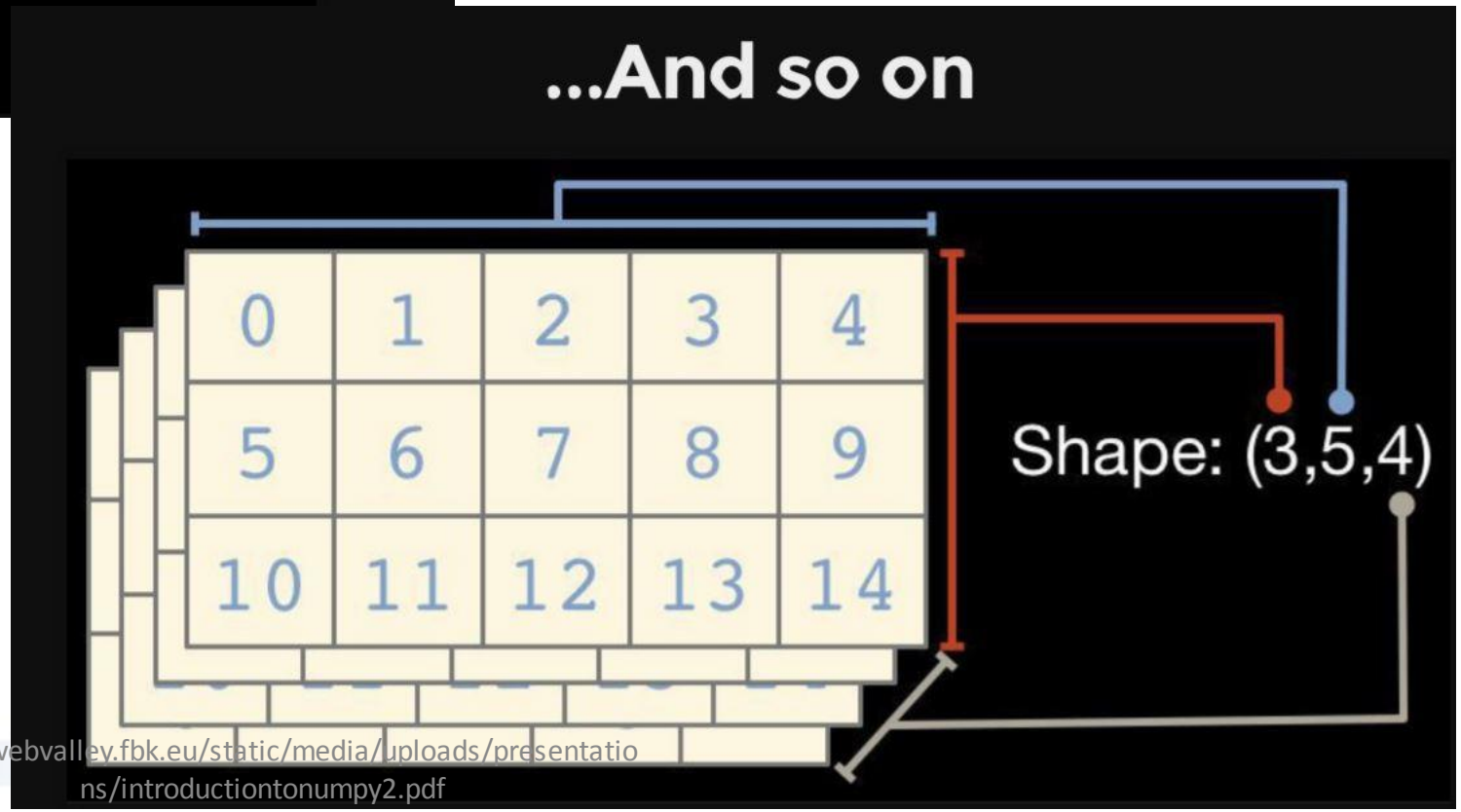
```
[1 2 3 4 5 6 7 8 9]
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[[14 24 34]
 [44 54 64]
 [74 84 94]]
[ 1.  2.  3.]

Process finished with exit code 0
```

https://webvalley.fbk.eu/static/media/uploads/presentations/introductiontonumpy2.pdf

UMKC

Reshape functions



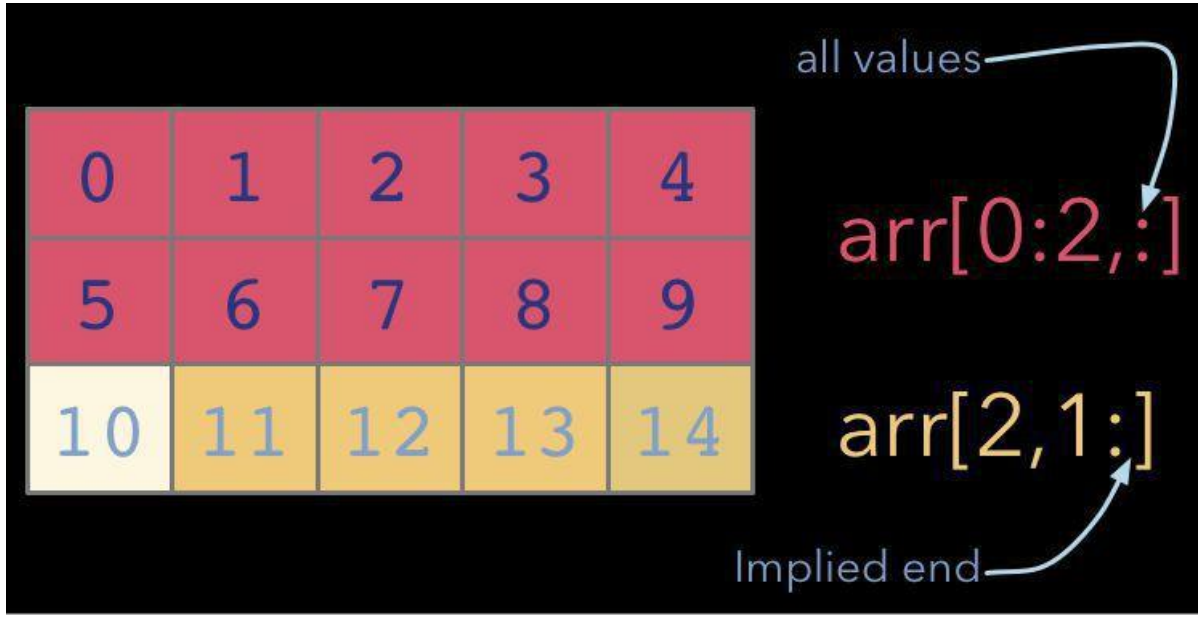


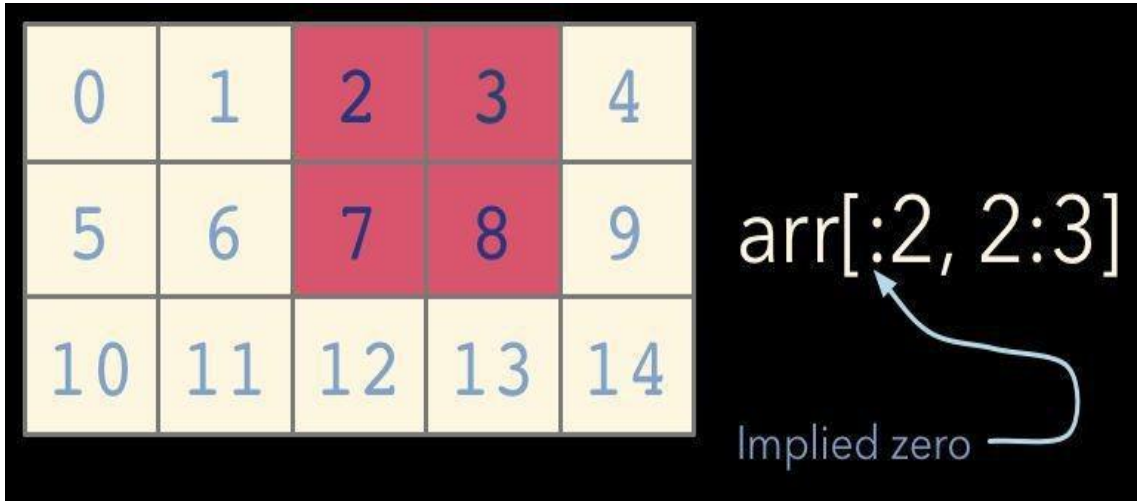
https://webvalley.fbk.eu/static/media/uploads/presentations/introductiontonumpy2.pdf

others

Input	Description	Output
np.arange(10)	Range of values	[0 1 2 3 4 5 6 7 8 9]
np.linspace(0,1,5)	By Specifying the number of elements	[0. 0.25 0.5 0.75 1.]
np.zeros((2,2))	Zero-initialized	[[0. 0.] [0. 0.]]
np.ones((2,2))	One-initialized	[1. 1.] [1. 1.]]
np.empty((2,2))	uninitialized	[[0. 0.] [0. 0.]]
np.eye(3)	Constant diagonal value	[[1. 0. 0.] [0. 1. 0.] [0. 0. 1.]]
np.diag([1,2,3,4])	Multiple diagonal values	[[1 0 0 0] [0 2 0 0] [0 0 3 0] [0 0 0 4]]

Indexing and Slicing as usual lists

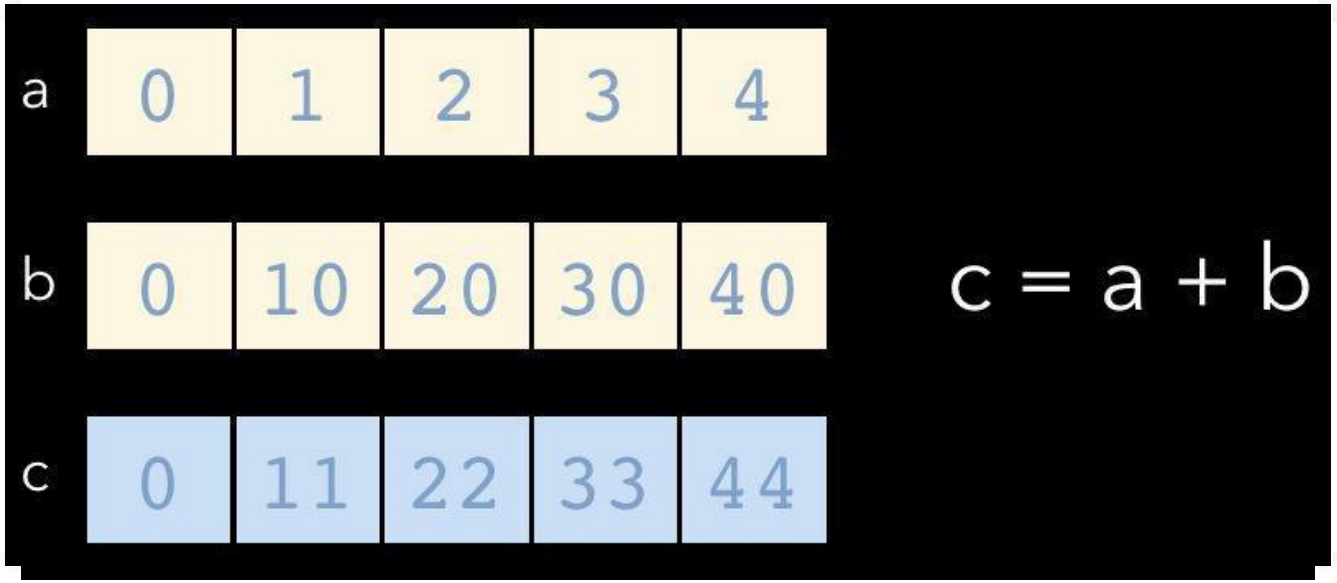




https://webvalley.fbk.eu/static/media/uploads/presentations/introductiontonumpy2.pdf

Universal Functions (ufuncs)

NumPy ufuncs are functions that operate element-wise on one or more arrays



ufuncs dispatch to optimized C inner-loops based on array dtype

https://webvalley.fbk.eu/static/media/uploads/presentations/introductiontonumpy2.pdf

NumPy has many built-in ufuncs

- comparison: `<, <=, ==, !=, >=, >`
- arithmetic: `+, -, *, /, reciprocal, square`
- exponential: `exp, expm1, exp2, log, log10, log1p, log2, power, sqrt`
- trigonometric: `sin, cos, tan, acsin, arccos, atctan`
- hyperbolic: `sinh, cosh, tanh, acsinh, arccosh, atctanh`
- bitwise operations: `&, |, ~, ^, left_shift, right_shift`
- logical operations: `and, logical_xor, not, or`
- predicates: `isfinite, isinf, isnan, signbit`
- other: `abs, ceil, floor, mod, modf, round, sinc, sign, trunc`

https://webvalley.fbk.eu/static/media/uploads/presentations/introductiontonumpy2.pdf

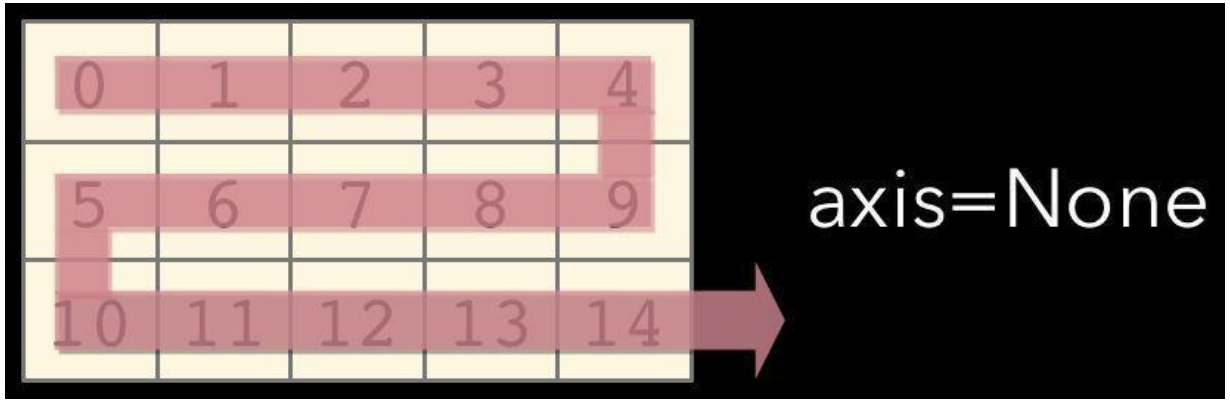
UMKC

Axis

Array method reductions take an optional `axis` parameter that specifies over which axes to reduce

`axis=None` reduces into a single scalar

```
In [7]: a.sum
()
Out[7]: 105
```



`axis=None` is the default

https://webvalley.fbk.eu/static/media/uploads/presentations/introductiontonumpy2.pdf

Array Methods

- **Predicates**
  - `a.any(), a.all()`
- **Reductions**
  - `a.mean(), a.argmin(), a.argmax(), a.trace(), a.cumsum(), a.cumprod()`
- **Manipulation**
  - `a.argsort(), a.transpose(), a.reshape(...), a.ravel(), a.fill(...), a.clip(...)`
- **Complex Numbers**
  - `a.real, a.imag, a.conj()`

https://webvalley.fbk.eu/static/media/uploads/presentations/introductiontonumpy2.pdf

UMKC

NumPy Functions

Input	Output	Desc
np.random.random((2,3))	[[0.0028206 0.73486004 0.07416516] [0.69691992 0.65554942 0.67732808]]	Random
np.random.normal(loc=1.0, scale=2.0, size=(2,2))	[[0.3378941 2.44143865] [-0.88674669 0.90112657]]	Random with loc and scale
np.savetxt("a_out.txt", a)		Save to file
np.loadtxt("a_out.txt")		Load from file

Numpy Usecase 3: numpyEx.py

```python
g=np.eye(3)
print(g)

g=np.diag([1,2,3,4])
print(g)


print(np.sum(a))

h=a.sum()
print(h)

h=np.random.random((2,3))
print(h)

h=np.random.normal(loc=1.0, scale=2.0, size=(2,2))
print(h)

np.savetxt("h_out.txt", h)
```

```
 [ 1.  1.]]
[[ 0.  0.  0.]]
[[ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]]
[[1 0 0 0]
 [0 2 0 0]
 [0 0 3 0]
 [0 0 0 4]]
45
45
[[ 0.22198311  0.1343603   0.3637289 ]
 [ 0.53936198  0.69237278  0.51634516]]
[[ 2.21331627  1.25249759]
 [-2.22609217  1.54270202]]

Process finished with exit code 0
```

Different functions of numpy and their output

Usecase 4

```python
import numpy as np

X = np.random.rand(3, 2)
print(X)


Y = X - X.mean(axis=1).reshape(-1,1)

print(Y)
```

```
C:\Users\Puchu\Anaconda3\python.exe C:/U
[[ 0.26533479  0.78140176]
 [ 0.39502827  0.06115333]
 [ 0.40918897  0.57880807]]
[[-0.25803349  0.25803349]
 [ 0.16693747 -0.16693747]
 [-0.08480955  0.08480955]]

Process finished with exit code 0
```

Creating random matrix

Calculating mean for each row and then broadcasting to each element

output

Web Scraping

- A computer software technique of extracting information from websites.

- for business, hobbies, research...
- Look for right URLs to scrap.
- Look for right content from webpages.
- Saving data into data store.

Web Scraping

This technique mostly focuses on the transformation of unstructured data (HTML format) on the web into structured data

BeautifulSoup library

- Python third-party library for extracting data from html and xml files
- Works with html.parser, lxml, html5lib
- Provides ways to navigate, search and modify the parse tree based on the position in the parse tree, tag name, tag attributes, CSS classes using regular expressions, user defined functions etc.
- Excellent tutorial with examples at http://www.crummy.com/software/BeautifulSoup/bs4/doc/
- Supports Python 2.7 and 3

Get familiar with HTML (Tags)

- While performing web scarping, we deal with html tags.
- Thus, we must have good understanding of them

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Heading</h1>

<p>My first paragraph.</p>

</body>
</html>
```

Get familiar with HTML (Tags)

- This syntax has various tags as elaborated below:
- **<!DOCTYPE html> :** HTML documents must start with a type declaration
- HTML document is contained between **<html>** and **</html>**
- The visible part of the HTML document is between **<body>** and **</body>**
- HTML headings are defined with the **<h1>** to **<h6>** tags
- HTML paragraphs are defined with the **<p>** tag

Get familiar with HTML (Tags)

- HTML links are defined with the **<a>** tag, “This is a link for test.com”
- HTML tables are defined with<Table>, row as <tr> and rows are divided into data as <td>

```
<table style="width:100%">
  <tr>
    <td>Jill</td>
    <td>Smith</td>
    <td>50</td>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson</td>
    <td>94</td>
  </tr>
</table>
```

Example

- import Requests
- from bs4 import BeautifulSoup

- html = requests.get("http://sampleshop.pl")
- bsObj = BeautifulSoup(html.content, "html.parser")
- print(bsObj.h1)

Simple ways to navigate data structure

```
print(soup.title)
print(soup.title.string)
```

```
<title>The Dormouse's story</title>
The Dormouse's story
```

```
print(soup.p)
print(soup.a)
print(soup.find_all('a'))
```

```
<p class="title"><b>The Dormouse's story</b></p>
<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>
[<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>, <a
```

```
print(soup.find(id="link3"))
```

```
<a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>
```

Extracting all the URLs found within a page's <a> tags

```
24  for link in soup.find_all('a'):
25      print(link.get('href'))
```

```
Run  test

C:\Python27\python.exe "C:/Users/sari
http://example.com/elsie
http://example.com/lacie
http://example.com/tillie

Process finished with exit code 0
```

- <a> tag defines a hyperlink which is used to link from one page to another page
- href attribute indicates the link destination

Basic functions: Getting headers, titles, body

- soup.text
- soup.head
- soup.title
- soup.body
- soup.findall(‘a’)
- soup.find(‘div’,{‘class’:’noprint’})

Use case: Wikipedia

- The goal here is to extract all the data related to a keyword from Wikipedia, then save those data in a file.

- Libraries to be imported:

```
import requests
from bs4 import BeautifulSoup
import os
```

Keyword for searching in the wiki

```
search = input('type something to search in wiki: ')
limit = input('how many results do you want to get?: ')

if not os.path.exists(search):
    print("Creating file " + search)
    file2 = open(search+'.txt','a+',encoding='utf-8')

search_spider(search, limit)
```

1. Create a file in the project with the keyword being searched

2. This function call the search api in wiki to find all the pages with the keyword that user has identified

```
/usr/bin/python3.5 /home/saria/Downloads/wiki
type something to search in wiki: music
```

Search the wiki with specified keyword

```
import requests
from bs4 import BeautifulSoup
import os


def search_spider(sea, lim):
    url = "https://en.wikipedia.org/w/index.php?limit="+lim+"&offset=0&search="+sea
    source_code = requests.get(url)
    plain_text = source_code.text
    soup = BeautifulSoup(plain_text, "html.parser")
    result_list = soup.findAll('div', {'class': "mw-search-result-heading"})
    for div in result_list:
        link = div.find('a')
        href = "https://en.wikipedia.org"+link.get('href')
        get_data(href)
```

1. Search the wiki for the keyword

2. Using BeautifulSoap to parse the html

3. Including all the div with this class in the result

4. Analyzing the first result

Get the data and save the result in the file

```
def get_data(url):
    source_code =   urllib.request.urlopen(url)
    plain_text = source_code
    soup = BeautifulSoup(plain_text, "html.parser")
    body = soup.find('div', {'class': 'mw-parser-output'})
    file2.write(str(body.text))
    print(body.text)
```

1. Get the url of the result, ex: Barack Obama

2. Parse the html page using BeautifulSoup

3. Open the created text file

4. Finding all the div with this class name

5. Writing the cleaned text in the file

Result

```
wiki-scrap-master ~/Downloads/wiki-scrap-master
music
    Music.txt
.gitignore
_config.yml
wikiScrap.py
External Libraries
```

```
Music:
For other uses, see Music (disambiguation).

Music



A painting on an ancient Greek vase depicts a music lesson (c. 510 BCE).



Medium
Sound, silence, time


Originating culture
Various


Originating era
Paleolithic era



Performing arts



Ballet
Circus skills
Clown
Dance
```

```
wikiScrap
/usr/bin/python3.5 /home/saria/Downloads/wiki-scrap-master/wikiScrap.py
type something to search in wiki: music
how many results do you want to get?: 1
Creating folder music
scanning: Music
```

References

- https://github.com/saria85/PythonProgramming-summer2017
- https://beautiful-soup-4.readthedocs.io/en/latest/
- http://www.w3resource.com/pythonexercises/https://www.slideshare.net/milkers/beautiful-soup?qid=64c9989d-94f7-4811-b3102cd7cfcb272e&v=&b=&from_search=6
- https://www.learnpython.org/