Basic Data Structures

Basic Python

Winter 2022: Dan Calderone

Python - Basics

Getting Started

importing

```
import numpy as np
from numpy import array
from numpy import array as arr
```

```
x = np.array( [ 1,2,3 ] )
x = array( [ 1,2,3 ] )
x = arr( [ 1,2,3 ] )
```

indenting

```
if a==1 :
    print( 'a is 1!' )
for t in range( 100 ):
    print( t )
```

String commands together

```
result = (func(a,b)[2].T@X[0])[0:2,2:4].dot(y)
```

everything is an object!

Basic Definitions

Variables:

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

Functions:

```
def func(x, y, a = 0, b = 1):
    x = a; y = b;
    print(a+b);
    return x+y
```

Notes
- indenting…
*- (*args, *kwargs)*
- pass by reference

Objects:

```
class Person:
    name = Champ;
    age = 10;
    def   __init__(self,name,age):
        self.name = name;
        self.age = age;
    def  myfunc(self):
        print("Hello my name is ",self.name)
```

Basic Syntax

Conditionals:

```
if  a==0 :
    print('do something');
elif  a==0 :
    print( 'do something else' );
else:
    print( 'do something else' );
```

Loops:

```
for t in range( T ):
   print( 'iteration: ',t );
```

```
t=0;
while t<=T :
    print( 'iteration: ',t );
    t++
```

Python - functions, objects

Functions:

```
def func(x, y, a = 0, b = 1):
    x = a; y = b;
    print(a+b);
    return x+y
```

Arguments:
- *default values*
- *required arguments - first*
- *keyword arguments - second*
- *dictionary inputs*
- *passed by reference*

defaults

```
def func(   x, y,    a = 0, b = 1,   *other ):
```

positional (required) | keyword (optional) | unspecified tuple

```
func(1,2, a=10, b=20)
func(1,2, b=10, a=10)
func(1,2, b=10)

func(1,2, a=10, b=20, (1,'a','blah') )
```

Objects:

```
class Person:
    name = Champ;
    age = 10;

    def   __init__(self,name,age):
        self.name = name;
        self.age = age;

    def  sayName(self, language):
        if language == 'spanish':
            print('Me llamo ',self.name)
        elif language == 'english':
            print("My name is ",self.name)
```

parameter
parameter
constructor
function

```
person1 = Person( "TurnedUpChamp",11 )      initializing…
person1.name
person1.age
person1.sayName('spanish')
```

Python - Base Data Structures

Critical

list : x = **[** 1, 'a', func **]**

dict : X = **{** 'key1': 1,
'key2': 'a',
'key3': func **}**

np.array: A = **np.array([[**1, 2, 3**],**
[3, 2, 1**],**
[2, 1, 3**]])**

Common

tuple - *static*
(1, 'a', func **)**

set - *no order*
{ 1, 'a', func **}**

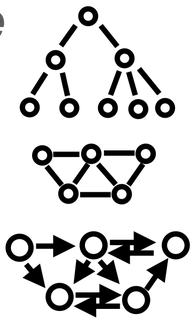
from collections **import** deque
deque:
x = **deque([** 1, 'a', func **])**

np.matrix
pd.dataframe
tf.tensor

CS Uses

linked list
circular linked list

hash table
tree
graph



directed graph

Graph = **{** 'node1': ['node2'],
'node2': ['node1',
'node3'] **}**

queue — *FIFO* — *"First In First Out"*

stack — *LIFO* — *"Last In First Out"*

vector
matrix
tensor

Python - conditionals, loops

Logic:

Booleans: **True False**
AND: **and &**
OR: **or |**
XOR: **^**
NOT: **not !**

COMPARISON: **== < > <= >=**

Conditionals:

```
if a==0 :
    print('do something');
elif a==0 :
    print( 'do something else' );
else:
    print( 'do something else' );
```

Loops:

```
for t in range( T ):
    print( 'iteration: ',t );

t=0;
while t<=T :
    print( 'iteration: ',t );
    t++
```

For Loop

```
times = [ ];
for t in range(T):
    times.append(t);
```

```
fruit = ['apples','banana','orange ];
for k , fruit in enumerate(fruit):
    print('The ',k, 'th fruit is an ',fruit)
```

Python - list comprehensions

Logic:

Booleans: **True False**
AND: **and &**
OR: **or |**
XOR: **^**
NOT: **not !**

COMPARISON: **== < > <= >=**

Conditionals:

```
if a==0 :
    print('do something');
elif a==0 :
    print( 'do something else' );
else:
    print( 'do something else' );
```

Loops:

```
for t in range( T ):
    print( 'iteration: ',t );

t=0;
while t<=T :
    print( 'iteration: ',t );
    t++
```

For Loop

```
times = [ ];
for t in range(T):
    times.append(t);
```

```
fruit = ['apples','banana','orange ];
for k , fruit in enumerate(fruit):
    print('The ',k, 'th fruit is an ',fruit)
```

List Comprehension

```
newlist = [ expression for item in iterable if condition==True]

nums1 = [0,1,2,3,4];   nums2 = [2,4,6,8,10];

a2 = [ a*a for a in nums1 if a != 1 ]
axb = [ a*b for a,b in zip(nums1,nums2) if a != b ]
```

Nested

```
axbxc = [[ a*b*c for a,b in zip(nums1,nums2) if a != b ] for c in nums2 ]
```

Python - map, reduce, filter

Logic:

Booleans: **True False**
AND: **and &**
OR: **or |**
XOR: **^**
NOT: **not !**

COMPARISON: **== < > <= >=**

Conditionals:

```
if a==0 :
    print('do something');
elif a==0 :
    print( 'do something else' );
else:
    print( 'do something else' );
```

Loops:

```
for t in range( T ):
    print( 'iteration: ',t );

t=0;
while t<=T :
    print( 'iteration: ',t );
    t++
```

For Loop

```
times = [ ];
for t in range(T):
    times.append(t);
```

```
fruit = ['apples','banana','orange ];
for k , fruit in enumerate(fruit):
    print('The ',k, 'th fruit is an ',fruit)
```

map

```
newlist = map( function, iterable )

nums1 = [0,1,2,3,4];   nums2 = [2,4,6,8,10];

def square(a): return a*a;
def prod(a,b): return a*b

a2  = map( square, nums1 )
axb = map( prod, zip(nums1,nums2) )

a2  = map( lambda a : a*a , nums1 )
axb = map( lambda a,b : a*b , zip(nums1,nums2) )
```

lambda functions

Python - map, reduce, filter

Logic:

Booleans: **True False**
AND: **and &**
OR: **or |**
XOR: **^**
NOT: **not !**

COMPARISON: **== < > <= >=**

Conditionals:

```
if a==0 :
    print('do something');
elif a==0 :
    print( 'do something else' );
else:
    print( 'do something else' );
```

Loops:

```
for t in range( T ):
    print( 'iteration: ',t );

t=0;
while t<=T :
    print( 'iteration: ',t );
    t++
```

For Loop

```
times = [ ];
for t in range(T):
    times.append(t);
```

```
fruit = ['apples','banana','orange ];
for k , fruit in enumerate(fruit):
    print('The ',k, 'th fruit is an ',fruit)
```

reduce

```
from functools import reduce
```

value = **reduce(** function, iterable **)**

alist = [1,2,4,8,16];

```
def prod(a,b): return a*b
product = reduce( prod, alist )
```

… starts by applying ans = prod(1,2)
… then applies ans = prod(ans,4)
… then applies ans = prod(ans,8)

```
product = reduce( lambda a,b : a*b, alist )
```

Python - map, reduce, filter

Logic:

Booleans: **True False**
AND: **and &**
OR: **or |**
XOR: **^**
NOT: **not !**

COMPARISON: **== < > <= >=**

Conditionals:

```
if a==0 :
    print('do something');
elif a==0 :
    print( 'do something else' );
else:
    print( 'do something else' );
```

Loops:

```
for t in range( T ):
    print( 'iteration: ',t );

t=0;
while t<=T :
    print( 'iteration: ',t );
    t++
```

For Loop

```
times = [ ];
for t in range(T):
    times.append(t);
```

```
fruit = ['apples','banana','orange ];
for k , fruit in enumerate(fruit):
    print('The ',k, 'th fruit is an ',fruit)
```

filter

```
newlist = filter( function, iterable )

numbers = [0,1,2,3,4,5,6,7,8];

def iseven(a): return np.mod(a,2) == 0
evens   = filter( iseven, numbers )
```

or use list comprehension…

```
evens = [ a for a in numbers if np.mod(a,2)==0 ]
```

…more pythonic

Python - Base Data Structures

list : x = **[** 1, 'a', func **]**

dict : X = **{** 'key1': 1,
'key2': 'a',
'key3': func **}**

np.array: A = **np.array([[**1, 2, 3**],**
[3, 2, 1**],**
[2, 1, 3**]])**

list : x = **[** 1, 'a', func **]**

x.**append**(element)
x.**extend**(otherlist)

x.**insert**(position,element)

num = x.**count**(element)
position = x.**index**('b')

element = x.**pop**(position)
x.**remove**(element)

x.**sort**(reverse=true, key=sortFunc)
x.**reverse**()

y = x.**copy**()
x.**clear**()

Python - Base Data Structures

list : x = **[** 1, 'a', func **]**

dict : X = **{** 'key1': 1,
'key2': 'a',
'key3': func **}**

np.array: A = **np.array([[**1, 2, 3**],**
[3, 2, 1**],**
[2, 1, 3**]])**

dict : X = **{** 'key1': 1, 'key2': 'a', 'key3': func **}**

X['key1']

dictkeys = X.**keys**()

dictvalues = X.**values**()

X.**fromkeys**(keys, values)

element = X.**update**(otherdict)

element = X.**update**(zip(keys, values))

element = X.**pop**(key)

Y = X.**copy**()

X.**clear**()

Python - Base Data Structures

list : x = **[** 1, 'a', func **]**

dict : X = **{** 'key1': 1,
'key2': 'a',
'key3': func **}**

np.array: A = **np.array([[**1, 2, 3**],**
[3, 2, 1**],**
[2, 1, 3**]])**

np.array : A = **np.array([[**1, 2, 3**], [**3, 2, 1**], [**2, 1, 3**]])**

A[np.newaxis,:] A[:,np.newaxis]

np.eye(n)
np.ones([m,n])
np.zeros([m,n])

np.stack([x,x,x]) …stack along new axis

np.vstack([x,x,x]) …stack vertically

np.hstack([x,x,x]) …stack horizontally

np.block([[A,B]
[C,D]]) …block matrix

np.arange(start,stop,step=1)
np.arange(start,stop,num=50)

np.append(A,newarray) …adds newarray to the end

np.insert(A,index,newarray) …adds new array at index

np.reshape(A,newshape) …cycle through deepest axes first

np.concatenate((A,B,C),axis=0) …must have same shape except along axis

np.flip(A,axis=None) …by default flips all axes

np.where(A,axis=None)