Data Structures

Aula 2

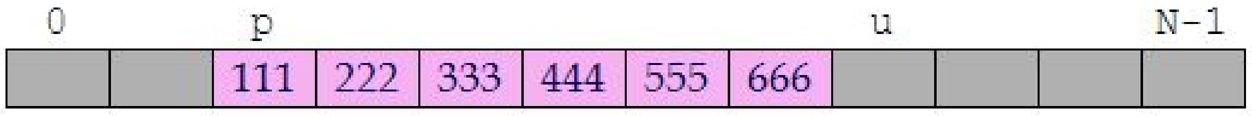
Queues

Dynamic structure that uses the strategy “First-In-First-Out” (FIFO)

The first inserted object in queue is also the first removed

Insertion and remotion O(1)

Some applications: Transport simulation, message, buffer ...



Queues (in python)

```
from collections import deque

fila = deque(["Eric", "John", "Michael"])

fila.append("Terry")

fila.append("Graham")

fila.popleft()

fila.popleft()

fila
```

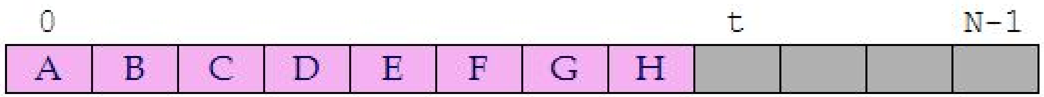
Stacks

Last-In-First-Out (LIFO)

The first inserted item is the last to be removed

Insertion and remotion O(1)

Applications: Expression evaluation and syntax parsing, recursivity ...



Stacks (in python)

```
pilha = [3, 4, 5]

pilha.append(6)

pilha.append(7)

pilha.pop()

print(pilha)

pilha.pop()

pilha.pop()

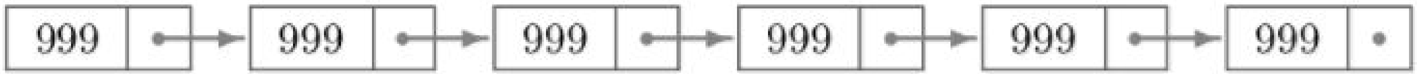
pilha
```

Lists

It’s a sequence of cells: Each cell contains an object of some type and the address to the next cell

Can be used to store a list of elements and form the basis for other abstract data types including the queue, the stack, and their variations

Applications: Storage (hard drive, file systems, memory ...), trees ...



Lists (in python)

```
a = [66.25, 333, 333, 1, 1234.5]

a.insert(2, -1)

a.append(333)

print(a)

a.index(333)

a.remove(333)

print(a)

a.reverse()

print(a)

a.sort()

a
```

Filter

```
def f(x):
    return x % 2 != 0 and x % 3 != 0

res = filter(f, range(2, 25))
for i in res:
    print(i)
```

Map

```
def f(x):
    return x % 2 != 0 and x % 3 != 0


res = map(f, range(2, 25))
for i in res:
    print(i)
```

Reduce

```
from functools import reduce

def f(x, y):
    return x + y

print(reduce(f, range(1, 5)))
```

Set

Do not admit repetitions

It is not ordered

Allows mathematical expressions like union, intersection, difference …

Applications: Verifies if exists an element, eliminate duplicated elements:

```
cesta = ['uva', 'laranja', 'uva', 'abacaxi', 'laranja', 'banana']

frutas = set(cesta)

print(frutas)

'capim' in frutas
```

Set (math operations)

```
a = set('abracadabra')

b = set('alacazam')

print(a)

print(a - b)

print(a | b)

print(a & b)

print(a ^ b)
```

Iterating through lists

```
knights = {'gallahad': 'the pure', 'robin': 'the brave'}

for k, v in knights.iteritems():

    print (k, v)

# two or more lists

questions = ['name', 'quest', 'favorite color']

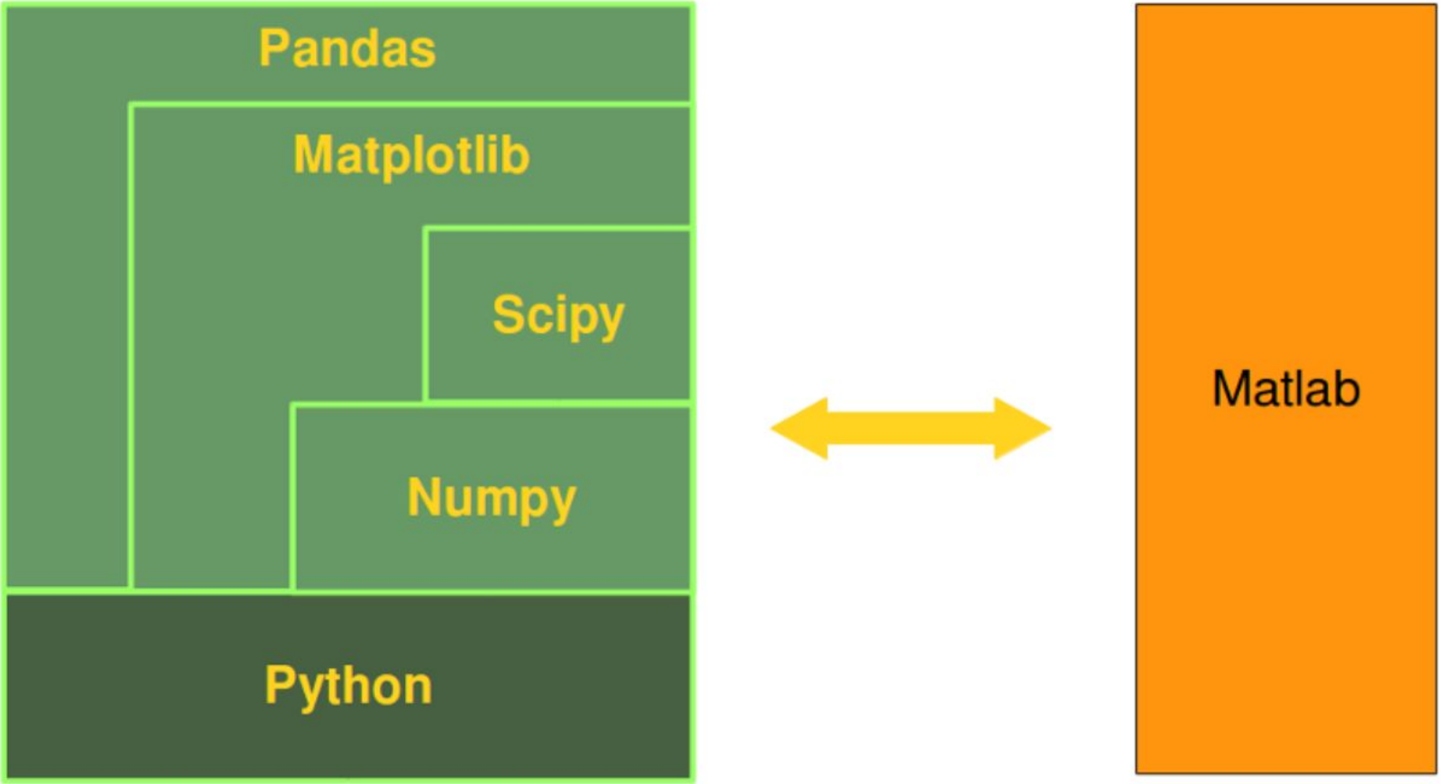
answers = ['lancelot', 'the holy grail', 'blue']

for q, a in zip(questions, answers):

    print ('What is your {0}?  It is {1}.'.format(q, a))
```

Numpy

Matlab alternative



Source: https://www.python-course.eu/numpy.php

Comparison between Core Python and Numpy

The advantages of Core Python:

- high-level number objects: integers, floating point
- containers: lists with cheap insertion and append methods, dictionaries with fast lookup

Advantages of using Numpy with Python:

- array oriented computing
- efficiently implemented multi-dimensional arrays
- designed for scientific computation

Source: https://www.python-course.eu/numpy.php

Numpy array

The most important attributes of a ndarray object are:

- Ndarray.ndim
- Ndarray.shape
- Ndarray.size
- Ndarray.dtype
- ndarray.itemsize

Numpy array (some useful functions)

```
import numpy as np

a = np.arange(15)

print(a)

print(a.reshape(3, 5))

print(a.shape)
```

Test the previous commands !

Exercises

Print the same result of np.arange(15) without use numpy

Print the same result of a.reshape(3, 5) without use numpy

Numpy array (some useful functions)

```
Z = np.zeros( (3,4) )

print(Z)

O = np.ones( (2,3,4) )

print(O)

print(np.empty( (2,3) ) )
```

Exercises

print the same result of np.zeros((3,4)) without use numpy

print the same result of np.ones((2,3,4)) without use numpy

print the same result of np.empty((2,3)) without use numpy

Numpy array (some useful functions)

```
L =  np.linspace( 0, 2, 9 )                    # 9 numbers from 0 to 2

print(L)

x = np.linspace( 0, 2*pi, 100 )        # useful to evaluate function at lots of points

print(x)
```

Exercises

print the same result of np.linspace(0, 2, 9) without use numpy

print the same result of np.linspace(0, 2*pi, 100) without use numpy

Numpy operations

```
A = np.array( [[1,1], [0,1]] )

B = np.array( [[2,0], [3,4]] )

print(A*B)                    # elementwise product


print(A.dot(B))               # matrix product


print(np.dot(A, B))
```

Exercises

A = np.array([[1,1], [0,1]])

B = np.array([[2,0], [3,4]])

Do elementwise product “A*B” without numpy

Do matrix product “A.dot(B)” without numpy

Numpy shape manipulation

```
import numpy as np

a = np.random.random((2,3))

print(a)

print(a.ravel())

print(a.reshape(3,2))

print(a)
a.resize((3,2))
print(a)
```

Exercises

Do the same list modification performed by np.resize((3,2)) without numpy

Do the numpy.ravel() method without numpy

Matplotlib

Plotting

```
import matplotlib.pyplot as plt

C = np.array([20.1, 20.8, 21.9, 22.5, 22.7, 22.3, 21.8, 21.2, 20.9, 20.1])

plt.plot(C)

plt.show()
```

Plotting (labels)

import matplotlib.pyplot as plt

plt.plot([1,2,3,4])

plt.ylabel('some numbers')

plt.show()

```plt.xlabel(`label of x`)

plt.title('title of figure')

```

# Plotting (axis)

```
import matplotlib.pyplot as plt

plt.plot([1,2,3,4], [1,4,9,16], 'ro')

plt.axis([0, 6, 0, 20])

plt.show()
```

Plotting (line styles)

```
import matplotlib.pyplot as plt


# evenly sampled time at 200ms intervals

t = np.arange(0., 5., 0.2)


# red dashes, blue squares and green triangles

plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')

plt.show()
```

Plotting (multiple figures and axes)

```
import numpy as np

import matplotlib.pyplot as plt

def f(t):

    return np.exp(-t) * np.cos(2*np.pi*t)

t1 = np.arange(0.0, 5.0, 0.1)

t2 = np.arange(0.0, 5.0, 0.02)
```

```
plt.figure(1)

plt.subplot(211)

plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')


plt.subplot(212)

plt.plot(t2, np.cos(2*np.pi*t2), 'r--')

plt.show()
```

Plotting (annotate)

```
import numpy as np

import matplotlib.pyplot as plt

ax = plt.subplot(111)

t = np.arange(0.0, 5.0, 0.01)

s = np.cos(2*np.pi*t)

line, = plt.plot(t, s, lw=2)

plt.annotate('local max', xy=(2, 1),
xytext=(3, 1.5),
arrowprops=dict(facecolor='black',
shrink=0.05),)

plt.ylim(-2,2)

plt.show()
```

Histograms

```
import numpy as np

import matplotlib.pyplot as plt



mu, sigma = 2, 0.5

v = np.random.normal(mu,sigma,10000)

plt.hist(v, bins=50, normed=1)       # matplotlib version (plot)

plt.show()
```

Histograms (numpy)

```
import numpy as np

import matplotlib.pyplot as plt



(n, bins) = np.histogram(v, bins=50, normed=True)  # NumPy version (no plot)

plt.plot(.5*(bins[1:]+bins[:-1]), n)

plt.show()
```