

Laboratorio Algoritmi e Strutture Dati

Laboratorio di Algoritmi

Scopo del laboratorio

- Nel corso di Algoritmi e Strutture Dati imparo soprattutto a:
 - ▶ Capire come funziona un algoritmo
 - ▶ Verificarne la correttezza
 - ▶ Calcolarne la complessità (soprattutto computazionale)
- Nel Laboratorio di Algoritmi imparo:
 - ▶ Ad implementare algoritmi in un linguaggio orientato alla prototipazione: Python
 - ▶ A modificare i programmi/algoritmi
 - ▶ A testare sperimentalmente il funzionamento dei programmi provando con vari tipi di input di dimensione diversa
 - ▶ A descrivere, con una relazione, l'andamento della sperimentazione

3 CFU ...

Tipo di lavoro

- *"Il Laboratorio di Algoritmi consiste nello svolgimento da parte dello studente di un compito didattico aggiuntivo nell'ambito dell'Insegnamento di Algoritmi e Strutture Dati"*
- Soprattutto lavoro a casa (individuale o di gruppo)
 - ▶ L'orale è **individuale**, ma si incoraggia il lavoro di gruppo nello svolgimento dei compiti a casa!
- Le lezioni (poche) sono in conclusione di alcune (non tutte) lezioni di ASD del martedì
- Si dovrà installare del software a casa
- Se necessario (contattare il docente ASAP) ci si può recare nel laboratorio in Morgagni

Valutazione (**alcune differenze rispetto agli anni scorsi!!**)

- E' un'idoneità (senza voto)
 - ▶ Non saranno accettate consegne in ritardo
 - ▶ Gli esercizi saranno corretti al ricevimento studenti e dopo averli corretti devono essere consegnati su moodle (almeno due settimane prima dell'orale)
 - ▶ Il giorno dell'orale vengono poste domande sugli esercizi svolti
 - ▶ Deve essere lavoro **individuale** quindi ci si aspetta che il codice sia diverso, ma **sicuramente** lo sarà la relazione e ancora di più i risultati sperimentali!!
 - ▶ In caso di "duplicazioni" (anche se individuate all'orale) sarà necessario svolgere un progetto individuale
 - ▶ Analogamente se non si è in grado di raccontare/motivare cosa fatto
 - ▶ Durante il corso sono suggeriti alcuni esercizi, ma solo quelli indicati esplicitamente devono essere consegnati e discussi all'esame

T_EX e L^AT_EX

L^AT_EX

- L^AT_EX *non* è un programma WYSIWYG (*what you see is what you get*)
- Non possiede un'interfaccia grafica per visualizzare in *tempo reale* il documento
- L^AT_EX è un linguaggio di markup utilizzato per generare testi
- La formattazione di equazioni matematiche è considerata migliore di quella ottenuta da altri editor di testo
- Si **compila** un file di testo con il sorgente (.tex) e si genera l'output (per esempio PDF)

T_EX

- T_EX è il “motore” di L^AT_EX
- Il nome deriva dalle prime tre lettere della parola "Tecnologia" in greco ($\tau\epsilon\chi$)
- Si pronuncia "tec"
- La storia di T_EX è lunga e complicata...

T_EX e L^AT_EX

- T_EX si occupa della formattazione dei documenti e interessa soprattutto chi progetta i template dei documenti
- L^AT_EX si occupa del contenuto, quindi è per chi scrive i documenti
- Come informatici ci potrebbero interessare entrambi gli aspetti ...
- ... ma ci focalizziamo sul secondo
- L^AT_EX è un insieme di macro costruite sopra T_EX che ci consentono di indicare i capitoli, i paragrafi le tabelle o le figure
- In L^AT_EX scrivo `\section{...}` in T_EX indicherei il tipo di carattere, l'altezza ecc.

Hello world!

```
\documentclass[]{article}
```

```
\begin{document}
```

```
Hello World!
```

```
\end{document}
```


Risorse

Editor Online

- Overleaf <https://www.overleaf.com>

Editor Offline

- TeXstudio
- TexMaker <http://www.xmlmath.net/texmaker/>
- Kile
- Gummi (linux)

Documentazione

- Documenti su moodle

Homework 1

- Scrivere un breve documento .tex in cui si descrive l'algoritmo insertion sort e si mostra il suo funzionamento per un vettore causale contenente 10 numeri casuali tra 1 e 100
 - ▶ Come otteniamo i numeri casuali? Esempio andare su `https://www.random.org/` e farsi dare i 10 numeri
- Cosa consegnare?
 - ▶ Niente! Non è obbligatorio svolgere l'esercizio, ma in caso di dubbi il docente può aiutare a svolgerlo.

Python 1/3

Cosa è Python?

Linguaggio

- Interpretato
- Interattivo
- Ad oggetti
- Incorpora
 - ▶ moduli
 - ▶ eccezioni
 - ▶ tipizzazione dinamica
 - ▶ tipi di dati dinamici di alto livello
 - ▶ classi
- Molto potente, sintassi chiara
- Portabile

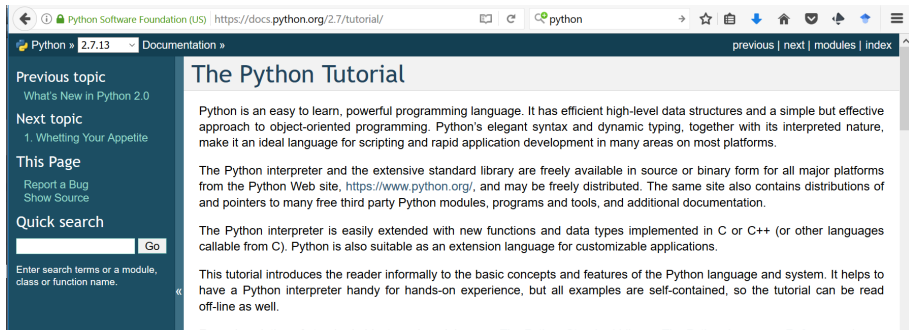
<https://docs.python.org/3/faq/general.html>

Per cosa è utile

- Python è un linguaggio di programmazione general-purpose con un'ampia *standard library* per:
 - ▶ String processing (espressioni regolari, Unicode, differenze tra file)
 - ▶ Protocolli Internet (HTTP, FTP, SMTP, XML-RPC, POP, IMAP, programmazione CGI)
 - ▶ ingegneria del software (unit testing, logging, profiling, parsing Python code)
 - ▶ Interfacce per sistemi operativi (system calls, filesystems, TCP/IP sockets)
- E soprattutto molte altre estensioni

<https://docs.python.org/3/faq/general.html>

Libri?!?



The screenshot shows a web browser displaying the Python 2.7 Tutorial page. The browser's address bar shows the URL <https://docs.python.org/2.7/tutorial/>. The page has a dark blue header with navigation links: "Python » 2.7.13 » Documentation »" and "previous | next | modules | index". The main content area is titled "The Python Tutorial" and contains the following text:

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

The Python interpreter and the extensive standard library are freely available in source or binary form for all major platforms from the Python Web site, <https://www.python.org/>, and may be freely distributed. The same site also contains distributions of and pointers to many free third party Python modules, programs and tools, and additional documentation.

The Python interpreter is easily extended with new functions and data types implemented in C or C++ (or other languages callable from C). Python is also suitable as an extension language for customizable applications.

This tutorial introduces the reader informally to the basic concepts and features of the Python language and system. It helps to have a Python interpreter handy for hands-on experience, but all examples are self-contained, so the tutorial can be read off-line as well.

On the left side of the page, there is a sidebar with the following sections:

- Previous topic**: What's New in Python 2.0
- Next topic**: 1. Whetting Your Appetite
- This Page**: Report a Bug, Show Source
- Quick search**: A search bar with a "Go" button and a placeholder text "Enter search terms or a module, class or function name."

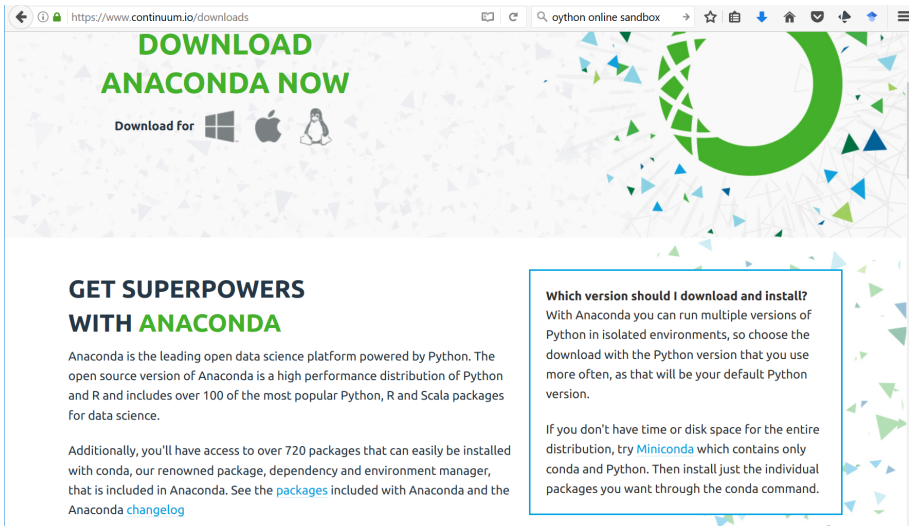
- <https://docs.python.org/3.7/tutorial/>
- 2.* vs 3.*

Alternative per sviluppo SW

- Installare Python (<https://www.python.org/>)
 - ▶ Usare la shell
 - ▶ eseguire un programma .py
- Usare Python in una Console online
 - ▶ <http://www.python.org>
 - ▶ <https://www.tutorialspoint.com/python/>
- Installare IPython (Jupyter - <http://jupyter.org/>)
 - ▶ Shell interattiva con completamento con tab, history...
- Usare IPython online
(https://www.tutorialspoint.com/ipython_terminal_online.php)
- Jupyter Notebook: Web-based interactive computational environment
 - ▶ Usare un Notebook online (<https://jupyter.org/try>)
- Installare una piattaforma.
Esempio Anaconda (distribuzione di Python con più di 100 package, NumPy, Pandas, SciPy, Matplotlib, Jupyter ...)




Install Anaconda

<https://www.anaconda.com/distribution/>

A screenshot of a web browser displaying the Anaconda download page. The browser's address bar shows the URL 'https://www.continuum.io/downloads'. The page features a large green 'C' logo on the right side, composed of many small triangles. The main heading is 'DOWNLOAD ANACONDA NOW' in green. Below it, it says 'Download for' followed by icons for Windows, macOS, and Linux. The section 'GET SUPERPOWERS WITH ANACONDA' is in bold, with 'ANACONDA' in green. The text describes Anaconda as a leading open data science platform powered by Python. A box on the right titled 'Which version should I download and install?' provides advice on choosing a Python version. At the bottom, it mentions access to over 720 packages and the 'changelog' link.

<https://www.continuum.io/downloads>

DOWNLOAD ANACONDA NOW

Download for   

GET SUPERPOWERS WITH ANACONDA

Anaconda is the leading open data science platform powered by Python. The open source version of Anaconda is a high performance distribution of Python and R and includes over 100 of the most popular Python, R and Scala packages for data science.

Additionally, you'll have access to over 720 packages that can easily be installed with conda, our renowned package, dependency and environment manager, that is included in Anaconda. See the [packages](#) included with Anaconda and the Anaconda [changelog](#)

Which version should I download and install?
With Anaconda you can run multiple versions of Python in isolated environments, so choose the download with the Python version that you use more often, as that will be your default Python version.

If you don't have time or disk space for the entire distribution, try [Miniconda](#) which contains only conda and Python. Then install just the individual packages you want through the conda command.

The Shell

- Si invoca python dalla linea di comando
- Utile per matematica di base, per provare idee
- Non si scrivono programmi completi nell'interprete
- Non si può salvare ciò che si scrive

Interprete

```
>>>print("Hello, World!")  
Hello, World!  
>>>var = 9+2  
>>>var*11  
121
```

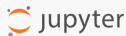
IPython

- Command shell for interactive computing offers introspection, rich media, shell syntax, tab completion, and history
- IPython features:
 - ▶ Interactive shells (terminal and Qt-based).
 - ▶ A browser-based notebook with support for code, text, mathematical expressions, inline plots and other media
 - ▶ Support for interactive data visualization and use of GUI toolkits
 - ▶ Tools for parallel computing

Jupyter Notebook

- Web-based interactive computational environment for creating IPython notebooks
 - An IPython notebook is a JSON document containing an ordered list of input/output cells which can contain code, text, mathematics, plots and rich media
 - IPython notebooks can be converted to a number of open standard output formats (HTML, presentation slides, LaTeX, PDF, ReStructuredText, Markdown, Python)
 - ▶ 'Download As' in the web interface
 - ▶ nbconvert in a shell
- ```
jupyter nbconvert Test.ipynb -to latex
```

# Install Jupyter

[Install](#)[About](#)[Resources](#)[Documentation](#)[NBViewer](#)[Widgets](#)[Blog](#)[Donate](#)

## Installing Jupyter

Get up and running with the Jupyter Notebook on your computer within minutes! Follow the instructions below.

### Prerequisite: Python

While Jupyter runs code in many programming languages, **Python** is a requirement (Python 3.3 or greater, or Python 2.7) for installing the Jupyter Notebook.

We recommend using the [Anaconda](#) distribution to install Python and Jupyter. We'll go through its installation in the next section.

### Installing Jupyter using Anaconda and conda

For new users, we **highly recommend** [installing Anaconda](#). Anaconda conveniently installs Python, the Jupyter Notebook, and other commonly used packages for scientific computing and data science.

We recommend using the Anaconda distribution to install Python and Jupyter. We'll go through its installation in the next section.

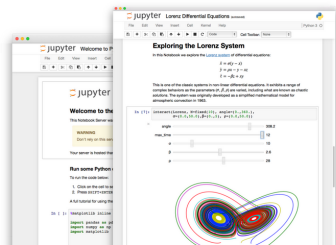
- Download [Anaconda](#). We recommend downloading Anaconda's latest Python 3 version (currently Python 3.5).
- Install the version of Anaconda which you downloaded, following the instructions on the download page.
- Congratulations, you have installed Jupyter Notebook. To run the notebook:

```
jupyter notebook
```

Incluso in Anaconda

poi: jupyter notebook

# Jupyter


[Install](#)
[About](#)
[Resources](#)
[Documentation](#)
[NBViewer](#)
[Widgets](#)
[Blog](#)
[Donate](#)


## The Jupyter Notebook

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and explanatory text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, machine learning and much more.



Language of choice



Share notebooks



Interactive widgets



Big data integration

# Jupyter

Hosted by Rackspace [Files](#) [Running](#) [Clusters](#)

Select items to perform actions on them.

[Upload](#) [New ▾](#) [↻](#)

|                          |                                                                                   |                                                                                   |
|--------------------------|-----------------------------------------------------------------------------------|-----------------------------------------------------------------------------------|
| <input type="checkbox"/> | <input type="button" value="▾"/>                                                  |  |
| <input type="checkbox"/> |  | communities                                                                       |
| <input type="checkbox"/> |  | datasets                                                                          |
| <input type="checkbox"/> |  | featured                                                                          |
| <input type="checkbox"/> |  | Welcome Julia - Intro to Gadfly.ipynb                                             |
| <input type="checkbox"/> |  | Welcome R - demo.ipynb                                                            |
| <input type="checkbox"/> |  | Welcome to Haskell.ipynb                                                          |
| <input type="checkbox"/> |  | Welcome to Python.ipynb                                                           |
| <input type="checkbox"/> |  | Welcome to Spark with Python.ipynb                                                |
| <input type="checkbox"/> |  | Welcome to Spark with Scala.ipynb                                                 |

# Jupyter

 Jupyter Welcome to Python (unsaved changes)



File Edit View Insert Cell Kernel Widgets Help

Python 3

          Code

CellToolbar

A full tutorial for using the notebook interface is available [here](#).

```
In []: %matplotlib notebook

import pandas as pd
import numpy as np
import matplotlib

from matplotlib import pyplot as plt
import seaborn as sns

ts = pd.Series(np.random.randn(1000), index=pd.date_range('1/1/2000', periods=1000))
ts = ts.cumsum()

df = pd.DataFrame(np.random.randn(1000, 4), index=ts.index,
 columns=['A', 'B', 'C', 'D'])
df = df.cumsum()
df.plot(); plt.legend(loc='best')
```

Feel free to open new cells using the plus button (+), or hitting shift-enter while this cell is selected.

Behind the scenes, the software that powers this is [tmnjb](#), a Tornado application that spawns [pre-built Docker containers](#) and then uses the [jupyter/configurable-http-proxy](#) to put your notebook server on a unique path.

jupyter Welcome to Python (unsaved changes)



Python 3 ●







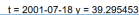




 Markdown
 

 CellToolbar

Figure 1





## Markdown

- Un modo per scrivere contenuto nel Web
- Scritto in "plaintext", caratteri normali con alcuni caratteri speciali
- Usato per commenti in GitHub
- Learning curve poco ripida (si impara in 10 minuti)
- Poche cose, in modo semplice (italic, bold, headers, lists...)
  - ▶ Corsivo: ( *\_* ). Esempio *\_corsivo\_*
  - ▶ Neretto ( **\*\*** ). Esempio **\*\*neretto\*\***
  - ▶ Header (#): Esempi (# Header One)... (### Header Three).
  - ▶ Inline link: link text tra [ ] link tra parentesi ( )  
Esempio: [Visit GitHub] (www.github.com)
  - ▶ Immagine come link: ![TestoAlternativo](http://xxx.jpg)
  - ▶ Liste: \* prima di ogni item
  - ▶ Oppure numeri

# Python

- # un commento
- Python è "space sensitive"
- I blocchi di codice sono definiti dall'indentazione  
Le linee dopo un : **devono** essere indentate

## Esempio

```
for i in (1,2,3,4):
 print(i),
```

```
1 2 3 4
```

# Stringhe

- Semplice stringa "hello world"
- Concatenazione: "hello"+" world" → "hello world"
- Ripetizione: "hello "\*3 → "hello hello hello "
- Indicizzazione: "world"[3] → "l"
  - ▶ Nota: liste python sono zero-offset
- Cercare: "o" in "hello" → True

# Numeri

- Notazione matematica di base:  $1.4$ ,  $2+2$ ,  $2^{**}10$ ,  $1e10$ 
  - ▶ Nota: Divisione intera è con floor:  $2/3 \rightarrow 0$ ,  $2./3 \rightarrow .6667$
- Funzioni matematiche richiedono `import math`
  - ▶ `import math`
  - ▶ `math.sqrt(4) → 2.0`
  - ▶ `from math import *`
  - ▶ `sqrt(4) → 2.0`

# Variabili e Liste

## Variabili Dynamically-Typed

- `x = 5`
- `x = 3.14`
- `x = 'text'`
- `x = '3.14'`

## Liste Dynamically-Typed

- `numbers = [0,1,2,3,4,5]`
- `words = ['algoritmi','strutture']`
- `combo = [12,23,['text','knot']] + words`

# Operatori su Liste

- `words.append('dati') → ['algoritmi','strutture','dati']`
- `words.insert(1,'e') → ['algoritmi','e', 'strutture','dati']`
- `words.reverse() → ['dati','strutture','e','algoritmi']`
- `words.remove('strutture') → ['dati','e','algoritmi']`

# Dizionari o Hash Tables, Associative Arrays, Lookup Tables

- `dictionary = {'indefatigable':'untiring',  
                  'intrepid':'fearlessness','dissemble':'simulate'}`
- `constants = {'pi':3.1415, 'e':2.7182, 'phi':1.6180}`
- `com_dict = {1:[1,2,3],2:[1,0,3],3:[0,4,5]}`

# Operazioni su dizionari

- `com_dict.keys()` → [1,2,3]
- `com_dict.values()` → [[1, 2, 3], [1, 0, 3], [0, 4, 5]]

## Accedere agli elementi

- `com_dict[3]` → [0,4,5]
- `constants['phi']` → 1.6180



# If, While, and For

## If...

```
if condition:
 statements
elif condition:
 statements
else condition:
 statements
```

**Occhio:** Python indenta i blocchi!

## While...

```
while condition:
 statements
```

## For...

```
for var in sequence:
 statements
```

# Esempi

## If...

```
if (x > 0):
 print("Positivo")
elif (x < 0):
 print("Negativo")
else:
 print("Zero")
```

## While...

```
while (1):
 print("Sempre vero")
```

## For...

```
for i in range(5):
 print(i)
```

# Insertion Sort

```
for j in range(1, len(A)) :
 print(A)
 key = A[j]
 i = j-1
 while i >= 0 and A[i] > key:
 A[i+1] = A[i]
 i = i-1
 A[i+1] = key
```

# Homework N.2

## Notebook Python

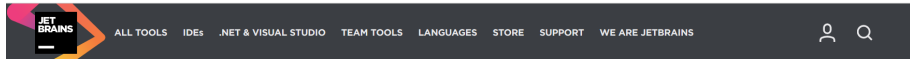
- Installare Jupyter o usarlo online
- Creare un notebook **Cognome\_Matricola\_Es2** con:
  - ▶ Descrivere l'algoritmo Insertion sort
  - ▶ Inizializzare il vettore *A* con i valori dell'esercizio 1
  - ▶ Scrivere il codice in Python dentro il notebook
  - ▶ Eseguire l'ordinamento mostrando i risultati ad ogni iterazione del ciclo principale
  - ▶ Scrivere il codice per la ricerca in array ordinato
  - ▶ Eseguirlo per ricercare un valore presente nell'array

# Python 2/3

# IDE: Pycharm

<https://www.jetbrains.com/help/pycharm/2017.1/meet-pycharm.html>

<https://www.jetbrains.com/help/pycharm/2017.1/quick-start-guide.html>



## PyCharm Edu

[What's New](#) [How It Works](#) [Quickstart](#) **[Download](#)**



Version: 3.5.1

Build: 163.15268

Released: March 21, 2017

[System requirements](#)

## Download PyCharm Edu

Windows

macOS

Linux

DOWNLOAD

149 MB

PyCharm Edu is free & open source.

Licensed under Apache License, Version 2.0

# Interi

```
x = 3
print(type(x)) # => <type 'int'>

print(x) # => 3
print(x + 1) # => 4
print(x - 1) # => 2
print(x * 2) # => 6
print(x ** 2) # => 9
x += 1
print(x) # => 4
x *= 2
print(x) # => 8
y = 2.5
print(type(y)) # => <type 'float'>"
print(y, y + 1, y * 2, y ** 2) # => 2.5 3.5 5.0 6.25
```

# Booleani

```
t = True
f = False
print(type(t)) # => <type 'bool'>

print(t and f) # AND => False
print(t or f) # OR => True

print(not t) # NOT => False
print(t != f) # XOR => True
```



# Stringhe

```
hello = 'hello' # Due modi per le stringhe
world = "world" # singole o doppie virgolette
print(hello) # => hello
print(len(hello)) # => 5

hw = hello + ' ' + world # concatenazione
print(hw) # => hello world

formattazione tipo sprintf
hw1 = '%s %s %d' % (hello, world, 1)
print(hw1) # => hello world 1
```

# Stringhe: metodi

```
s = "hello"
print(s.capitalize()) # => Hello
print(s.upper()) # => HELLO

print(s.rjust(7)) # giustifica a dx => " HELLO"
print(s.center(7)) # centra => " HELLO "
```

```
print(s.replace('l','el')) # => heellelo
print(' world '.strip()) # => world
```

# Liste

Una lista in Python è un array ridimensionabile che può contenere elementi di tipo diverso

```
xs = [3, 1, 2] # Crea una lista
print(xs, xs[2]) # => [3, 1, 2] 2
print(xs[-1]) # Indici <0 da fine lista => 2
```

```
xs[2] = 'abc' # Liste con tipi diversi
print(xs) # => [3, 1, 'abc']
```

```
xs.append('def') # aggiunge in fondo
print(xs) # => [3, 1, 'abc', 'def']
```

```
x = xs.pop() # Rimuove e restituisce l'ultimo elemento
print(x, xs) # => def [3, 1, 'abc']
```

# Slicing

## Permette di accedere a sottoliste

```
nums = range(5) # crea una lista
print(nums) # => [0, 1, 2, 3, 4]
print(nums[2:4]) # da indice 2 a 4 (escluso) => [2,3]

print(nums[2:]) # da 2 alla fine => [2, 3, 4]
print(nums[:2]) # dall'inizio a 2 (escluso) => [0, 1]

print(nums[:]) # Tutta la lista => [0, 1, 2, 3, 4]
print(nums[:-1]) # indici negativi => [0, 1, 2, 3]

nums[2:4] = [8, 9] # nuova sottolista
print(nums) # => [0, 1, 8, 9, 4]
```

## Cicli su liste

```
animals = ['cat', 'dog', 'monkey']
for animal in animals:
 print(animal)
=> "cat", "dog", "monkey" (su linee separate)
```

```
animals = ['cat', 'dog', 'monkey']
for idx, animal in enumerate(animals):
 print('#%d: %s' % (idx + 1, animal))
=> "#1: cat", "#2: dog", "#3: monkey", (su linee separate)
```

# List comprehension

## Calcolo quadrati:

```
nums = [0, 1, 2, 3, 4]
squares = []
for x in nums:
 squares.append(x ** 2)
print(squares) # => [0, 1, 4, 9, 16]
```

## Con list comprehension (un modo veloce per creare liste):

```
nums = [0, 1, 2, 3, 4]
squares = [x ** 2 for x in nums]
print(squares) # => [0, 1, 4, 9, 16]
```

## Anche con condizioni:

```
nums = [0, 1, 2, 3, 4]
even_squares = [x ** 2 for x in nums if x % 2 == 0]
print(even_squares) # => "[0, 4, 16]"
```

# Dizionari

## Memorizza coppie (chiave, valore)

```
d = {'cat': 'cute', 'dog': 'furry'} # Nuovo dizionario
print(d['cat']) # => "cute"
print('cat' in d) # => True

d['fish'] = 'wet' # Aggiunge voce
print(d['fish']) # => "wet"

print(d['monkey']) # => KeyError: ...
print(d.get('monkey', 'N/A')) # Default => "N/A"
print(d.get('fish', 'N/A')) # => "wet"

del d['fish'] # Rimuove un elemento
print(d.get('fish', 'N/A')) # ora manca => "N/A"
```

## Cicli con dizionari

```
d = {'person': 2, 'cat': 4, 'spider': 8}
for animal in d:
 legs = d[animal]
 print('A %s has %d legs' % (animal, legs))
```

=>

```
"A person has 2 legs"
"A spider has 8 legs"
"A cat has 4 legs"
```



# Insiemi

```
animals = {'cat', 'dog'}
print('cat' in animals) # => "True"
print('fish' in animals) # => "False"
```

```
animals.add('fish') # Aggiunge un elemento
print('fish' in animals) # => "True"
print(len(animals)) # => "3"
```

```
animals.add('cat') # Aggiunge un elemento presente
print(len(animals)) # => "3"
```

```
animals.remove('cat') # Rimuove un elemento
print(len(animals)) # => "2"
```

# Tupla

## Lista ordinata di valori immutabile

```
Crea un dizionario con tuple
d = {(x, x + 1): x for x in range(10)}
print(type(d))
```

```
=> <type 'dict'>
```

```
t = (5, 6)
print(type(t))
```

```
Crea una tupla
=> <type 'tuple'>
```

```
print(d[t])
print(d[(1, 2)])
```

```
=> 5 (posizione)
=> 1
```

# Funzioni

```
def sign(x):
 if x > 0:
 return 'positive'
 elif x < 0:
 return 'negative'
 else:
 return 'zero'

for x in [-1, 0, 1]:
 print(sign(x))

=> "negative", "zero", "positive"
```

# Funzioni con argomenti

```
def hello(name, loud=False):
 if loud:
 print('HELLO, %s!') % name.upper()
 else:
 print('Hello, %s') % name

hello('Bob') # => "Hello, Bob"
hello('Fred', loud=True) # => "HELLO, FRED!"
```

## Esempio classe Python

```
class Employee:
 'Common base class for all employees'
 empCount = 0

 def __init__(self, name, salary):
 self.name = name
 self.salary = salary
 Employee.empCount += 1

 def displayCount(self):
 print("Total Employee %d" % Employee.empCount)

 def displayEmployee(self):
 print("Name : ",self.name,", Salary: ", self.salary)
```

## Esempio classe Python

- `empCount` è una variabile della classe
  - Si accede con `Employee.empCount`
  - `__init__()`: costruttore
  - Il primo argomento di ogni metodo nella classe è `self`
  - Questo è omesso quando si chiama
- 
- `emp1 = Employee("Zara", 2000)`
  - `emp2 = Employee("Manni", 5000)`
  - `emp1.displayEmployee()`
  - `emp2.displayEmployee()`
  - `print("Total Employee %d" % Employee.empCount)`

# Garbage collection

- Tutto è un oggetto in Python, anche le costanti

```
a = 33 # Crea oggetto <33>
b = a # Incrementa contatore rif. a <33>
c = [b] # Incrementa contatore rif. a <33>

del a # Decrementa contatore rif. a <33>
b = 100 # Decrementa contatore rif. a <33>
c[0] = -1 # Decrementa contatore rif. a <33>
```

## Random array

```
def random_list(n):
 # Permutazione casuale dei primi n interi
 A=range(n)
 random.shuffle(A)
 return A

>>> A = random_list(10)
>>> A
[0, 1, 4, 8, 9, 4, 5, 2, 7, 3]
>>> insertion_sort(A)
>>> A
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```



# Misurare tempo di esecuzione

```
from timeit import default_timer as timer

start = timer()
...
end = timer()
print(end - start)
```

- Non usare `time.time` oppure `time.clock`: possono fornire risultati non corretti
- Non includere istruzioni `print` nel codice

## `timeit.default_timer()`

- Define a default timer, in a platform-specific manner
- On Windows, `time.clock()` has microsecond granularity, but `time.time()`'s granularity is 1/60th of a second
- On Unix, `time.clock()` has 1/100th of a second granularity, and `time.time()` is much more precise.
- On either platform, `default_timer()` measures wall clock time, not the CPU time

# La relazione

# Relazione per esercizi

Deve contenere:

- **breve** introduzione che descrive il problema
- una **breve** descrizione delle caratteristiche teoriche degli algoritmi e delle strutture dati utilizzate
- una valutazione a priori delle **prestazioni attese** degli algoritmi analizzati sperimentalmente
- una descrizione degli **esperimenti** che verranno fatti (non un semplice elenco)
- la **documentazione** del codice implementato
- i risultati sperimentali, sia in **tabelle** che con **grafici**
- l'**analisi** completa di tali risultati, effettuata in modo critico

# La teoria

- Fa riferimento a quanto studiato nel corso di Algoritmi e Strutture Dati
- Deve essere solo la parte finalizzata all'esperimento
- Non va bene un semplice copia/incolla dagli appunti (libro)
  - ▶ Anzi, forse sarebbe anche troppo...
- Bisogna descrivere gli aspetti più importanti e come questi indichino indirettamente quali test eseguire
- Se serve un teorema, basta mostrarne l'applicazione non serve la dimostrazione

# Documentazione del codice

La documentazione deve includere:

- uno schema del contenuto e delle interazioni fra i moduli
- uno schema delle classi
- un'analisi delle scelte implementative effettuate
  - ▶ se erano possibili alternative, indicare perché è stata fatta una certa scelta
- una descrizione dei metodi implementati, indicando in particolare l'input/output e la funzione svolta

# Descrizione degli esperimenti condotti

Bisogna descrivere:

- i dati utilizzati
  - ▶ Se sono stati generato automaticamente, come questo avviene
  - ▶ Altrimenti da dove provengono e quali sono le loro caratteristiche
- Specifiche della piattaforma di test (hardware, sistema operativo);
- Quali misurazioni vengono effettuate
  - ▶ Che tipo di misure
  - ▶ Quante volte si eseguono i vari test
- Come si effettuano le misurazioni (porzioni di codice osservate, numero di run effettuati)

# Presentazione risultati sperimentali

Presentati sia in tabelle che con grafici

- Le tabelle devono contenere tutti i dati (al limite in un file allegato)
- I valori nelle tabelle devono avere un numero di cifre significative appropriato (python può fornire numeri con precisione arbitraria)
- Un grafico serve per evidenziare l'andamento di un valore, ma non sostituisce la tabella.
- A volte possono essere presentati vari grafici per una tabella per mostrare aspetti diversi
- Un grafico non chiaro o che non mostri qualcosa di interessante è inutile
- Non importa la bellezza di un grafico
- Tutti grafici le tabelle e le figure devono essere
  - ▶ Descritti da una didascalia (lunga qb...)
  - ▶ Citati nel testo

`\label{} ... \ref{}`



# Analisi dei risultati sperimentali

- Un esperimento **non** è una semplice collezione di dati.
- I risultati di ogni esperimento vanno commentati ed analizzati in modo critico, citando i grafici e le tabelle corrispondenti
- Nell'analisi si verifica se le ipotesi teoriche vengono verificate con i dati sperimentali
- Al termine dell'analisi degli esperimenti un paragrafo di conclusioni è spesso utile per sintetizzare i risultati ottenuti

## Esercizio N.1: Valutazione performance Python

- Questo **deve essere consegnato** per l'esame
- E deve essere discusso con il docente (al ricevimento) secondo le tempistiche indicate sul sito di e-learning
- Scrivere uno o più programmi Python che permettono di:
  - ▶ Generare un vettore casuale e/o con caratteristiche opportune
  - ▶ Implementare l'algoritmo `Insertion_sort()`
  - ▶ Implementare a scelta `Quick_sort()` o `Merge_sort()`
- Scrivere un programma che esegua dei test misurando i tempi di esecuzione degli algoritmi precedenti su:
  - ▶ Dati di dimensione crescente (array con 10, 1000, 10000, ... dati)
  - ▶ Arrestandosi quando il tempo di esecuzione è maggiore di alcuni minuti
  - ▶ Usando dati casuali e dati che costituiscano il caso migliore/peggiore per un algoritmo
- Scrivere una relazione in cui si descrivono gli esperimenti svolti (riportando i risultati sperimentali ottenuti) e si analizzano i risultati alla luce della teoria
- Impacchettare i file .py e la relazione (tex e PDF) in un file `Cognome_Matricola_Es1.zip` e caricarlo su e-l!

# Python 3/3

# Numpy

- Numpy è la libreria principale per calcolo scientifico in Python
- Gestisce efficientemente array multidimensionali
- All'inizio del codice:

```
import numpy as np
```

- Array:
  - ▶ Un array numpy è una matrice di valori (tutti dello stesso tipo)
  - ▶ Indicizzati con una tupla di valori non negativi
  - ▶ Numero di dimensioni: **rank** dell'array
  - ▶ **Shape**: tupla con le dimensioni

- Tutorial numpy:

```
docs.scipy.org/doc/numpy-dev/user/quickstart.html
```

# Array

```
import numpy as np
a = np.array([1, 2, 3]) # Crea array con rank 1 (vettore)
print(type(a)) # => "<type 'numpy.ndarray'>"
print(a.shape) # => "(3,)"

print(a[0], a[1], a[2]) # => "1 2 3"
a[0] = 5 # Cambia un elemento dell'array
print(a) # => "[5, 2, 3]"

b = np.array([[1,2,3],[4,5,6]]) # Crea array di rank 2
print(b.shape) # => "(2,3)"
print(b[0,0], b[0,1], b[1,0]) # => "1 2 4"
```

## Creazione array

```
import numpy as np
a = np.zeros((2,2))
print(a) # Crea array di zero
 # => "[[0. 0.]
 # [0. 0.]]"

b = np.ones((1,2))
print(b) # Crea array con 1
 # => "[[1. 1.]]"

c = np.full((2,2), 7)
print(c) # Crea array costante
 # => "[[7. 7.]
 # [7. 7.]]"

d = np.eye(2)
print(d) # Crea matrice identita 2x2
 # => "[[1. 0.]
 # [0. 1.]]"

e = np.random.random((2,2)) # Array con valori casuali
print(e)
```

## Indicizzazione array

```
import numpy as np
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
Crea array con shape (3, 4)
[[1 2 3 4]
[5 6 7 8]
[9 10 11 12]]

b = a[:2, 1:3]
Usa slicing per avere il sottoarray con le prime 2 righe
e colonne 1 e 2 ottiene b(shape (2, 2)):
[[2 3]
[6 7]]

Uno slice di un array e' una sua vista
print(a[0, 1]) # => "2"
b[0, 0] = 77 # b[0, 0] stessi dati di a[0, 1]
print(a[0, 1]) # => "77"
```

## Integer array indexing

- Con slicing il nuovo array è una parte di un altro
- Con integer indexing costruisco array arbitrari

```
import numpy as np
a = np.array([[1,2], [3, 4], [5, 6]])

array restituito ha shape (3,2)
print(a[[0, 1, 2], [0, 1, 0]]) # => "[1 4 5]"
Equivalente a
print(np.array([a[0, 0], a[1, 1], a[2, 0]])) # => "[1 4 5]"

Posso riusare elementi :
print(a[[0, 0], [1, 1]]) # => "[2 2]"
Equivalente:
print(np.array([a[0, 1], a[0, 1]])) # => "[2 2]"
```



Trucco... selezionare o cambiare un elemento da ogni riga:

```
import numpy as np

Crea un nuovo array
a = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])

print(a) # => "array([[1, 2, 3],
[4, 5, 6],
[7, 8, 9],
[10, 11, 12]])"

Crea un array di indici
b = np.array([0, 2, 0, 1])

Seleziona un elemento da ogni riga di a usando indici in
print(a[np.arange(4), b]) # => "[1 6 7 11]"
```

```
...
```

```
Cambia elementi
```

```
a[np.arange(4), b] += 10
```

```
print(a) # => "array([[11, 2, 3],
[4, 5, 16],
[17, 8, 9],
[10, 21, 12]])
```

## Boolean array indexing

```
import numpy as np

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

bool_idx = (a > 2) # Trova elementi di a > 2;
 # restituisce array numpy di booleani

print(bool_idx) # => "[[False False]
 # [True True]
 # [True True]]"

Array di rango 1 con gli elementi True di bool_idx
print(a[bool_idx]) # => "[3 4 5 6]"

In modo piu' conciso:
print(a[a > 2]) # => "[3 4 5 6]"
```

# Array math

```
import numpy as np

x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)

Somma [[6.0 8.0]
[10.0 12.0]]
print(x + y)
print(np.add(x, y)) # equivalenti

Differenza [[-4.0 -4.0]
[-4.0 -4.0]]
print(x - y)
print(np.subtract(x, y)) # equivalenti
```

# Array math

```
...

Prodotto [[5.0 12.0]
[21.0 32.0]]
print(x * y)
print(np.multiply(x, y)) # equivalenti

[[0.2 0.33333333]
[0.42857143 0.5]]
print(x / y)
print(np.divide(x, y)) # equivalenti

[[1. 1.41421356]
[1.73205081 2.]]
print(np.sqrt(x))
```

# sum

```
import numpy as np
```

```
x = np.array([[1,2],[3,4]])
```

```
print(np.sum(x)) # Somma tutti gli elementi => 10
```

```
print(np.sum(x, axis=0)) # Somma di ogni colonna => [4 6]
```

```
print(np.sum(x, axis=1)) # Somma di ogni riga => [3 7]
```

# Trasposizione

```
import numpy as np

x = np.array([[1,2], [3,4]])
print(x) # => "[[1 2]
 # [3 4]]"
print(x.T) # => "[[1 3]
 # [2 4]]"

Trasposta di un vettore non cambia:
v = np.array([1,2,3])
print(v) # => "[1 2 3]"
print(v.T) # => "[1 2 3]"
```

# Broadcasting

- Meccanismo che permette di effettuare operazioni aritmetiche su array di diverse dimensioni
- Spesso usiamo un piccolo array per eseguire operazioni su un array più grande
- Esempio sommiamo un vettore  $v$  ad ogni riga di una matrice  $x$ :



```
import numpy as np

x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
y = np.empty_like(x) # matrice vuota con lo stesso shape
```

1) Si puo' usare un ciclo:

```
for i in range(4):
 y[i, :] = x[i, :] + v

otteniamo
[[2 2 4]
[5 5 7]
[8 8 10]
[11 11 13]]
print(y)
```

I cicli in Python possono essere lenti!!

2) Costruisco matrice `vv` accatastando `v` varie volte e poi sommando elemento per elemento:

```
import numpy as np

x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
vv = np.tile(v, (4, 1)) # 4 copie di v
print(vv) # => [[1 0 1]
 # [1 0 1]
 # [1 0 1]
 # [1 0 1]]

y = x + vv # somma x e vv elementwise
print(y) # => "[[2 2 4
 # [5 5 7]
 # [8 8 10]
 # [11 11 13]]]"
```

### 3) con broadcasting non creo copie di $v$

```
import numpy as np
```

```
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
```

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

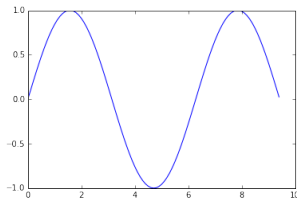
```
y = x + v # Somma v a ogni riga di x con broadcasting
print(y) # => "[[2 2 4]
 # [5 5 7]
 # [8 8 10]
 # [11 11 13]]"
```

$y = x + v$  funziona anche se  $x$  ha shape (4, 3) e  $v$  ha shape (3,)

## Plot in matplotlib

```
import numpy as np
import matplotlib.pyplot as plt

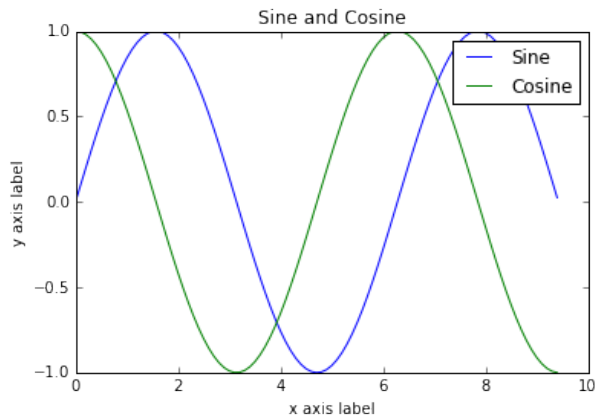
Calcola le coordinate x e y per punti della funzione seno
x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x)
Disegna i punti
plt.plot(x, y)
plt.show() # Visualizza il plot
```



```
import numpy as np
import matplotlib.pyplot as plt

Seno e coseno
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

usiamo matplotlib
plt.plot(x, y_sin)
plt.plot(x, y_cos)
plt.xlabel('x axis label')
plt.ylabel('y axis label')
plt.title('Sine and Cosine')
plt.legend(['Sine', 'Cosine'])
plt.show()
```



# Pickle

Il modulo pickle implementa la serializzazione e de-serializzazione di un oggetto Python

- pickling converte un oggetto Python in una sequenza di byte
- unpickling è l'operazione inversa

Pickling è anche noto come serializzazione (serialization), marshalling o flattening

# Pickle

```
1 # Salviamo un dizionario in un file
2 import pickle
3
4 favorite_color = { "lion": "yellow", "kitty": "red" }
5
6 pickle.dump(favorite_color, open("save.p", "wb"))
```

```
1 # Carichiamo il dizionario
2 import pickle
3
4 favorite_color = pickle.load(open("save.p", "rb"))
5 # favorite_color e' { "lion": "yellow", "kitty": "red"
```



# Strutture dati in Python

## Stack implementato con una lista Python

```
class Stack:
 def __init__(self):
 self.items = []

 def is_empty(self):
 return self.items == []

 def push(self, item):
 self.items.append(item)

 def pop(self):
 return self.items.pop()

 def size(self):
 return len(self.items)
```

# Usare stack

```
from Stack import Stack

s = Stack()
print(s.is_empty())
s.push(4)
s.push('dog')
print(s.peek())
s.push(True)
print("size", s.size())
print(s.is_empty())
s.push(7.3)
print(s.pop())
print(s.pop())
print(s.size())
```

# Lista concatenata

Prima definiamo i nodi della lista

```
class Node:
 def __init__(self, init_data):
 self.data = init_data
 self.next = None // come Nil e Null

 def get_data(self):
 return self.data

 def get_next(self):
 return self.next

 def set_data(self, new_data):
 self.data = newdata

 def set_next(self, new_next):
 self.next = new_next
```

## Nella lista concatenata aggiungo elementi in testa

```
class LinkedList:
 def __init__(self):
 self.head = None

 def is_empty(self):
 return self.head == None

 def add(self, item):
 temp = Node(item)
 temp.set_next(self.head)
 self.head = temp

 def size(self):
 current = self.head
 count = 0
 while current != None:
 count = count + 1
 current = current.get_next()
 return count
```

## Lista concatenata

Cerchiamo elementi e stampiamo la lista

...

```
def search(self, item):
 current = self.head
 found = False
 while current != None and not found:
 if current.get_data() == item:
 found = True
 else:
 current = current.get_next()
 return found
```

```
def PrintL(self):
 current = self.head
 previous = None
 while current != None:
 print('..' , current.get_data())
 current = current.get_next()
```

# Lista concatenata

## Cancello elemento

```
...
def remove(self, item):
 current = self.head
 previous = None
 found = False
 while not found:
 if current.get_data() == item:
 found = True
 else:
 previous = current
 current = current.get_next()
 if previous == None:
 self.head = current.get_next()
 else:
 previous.set_next(current.get_next())
```

# Lista concatenata

## Uso la lista

```
from LinkedList import LinkedList
```

```
mylist = LinkedList()
```

```
mylist.add(31)
```

```
mylist.add(77)
```

```
mylist.add(17)
```

```
mylist.add(93)
```

```
print mylist.size()
```

```
print mylist.search(17)
```

```
print mylist.search(22)
```

```
mylist.PrintL()
```

## ABR: Classe Node

```
class Node:
 def __init__(self, key):
 self.key = key
 self.left = None
 self.right = None
 def get(self):
 return self.key
 def set(self, key):
 self.key = key
 def getChildren(self):
 children = []
 if (self.left != None):
 children.append(self.left)
 if (self.right != None):
 children.append(self.right)
 return children
```



## ABR: Classe ABR

```
class ABR:
 def __init__(self):
 self.root = None

 def setRoot(self, key):
 self.root = Node(key)

 def insert(self, key):
 if (self.root is None):
 self.setRoot(key)
 else:
 self.insertNode(self.root, key)
```

## ABR: Classe ABR

```
class ABR:
 ...
 def insertNode(self, currentNode, key):
 if (key <= currentNode.key):
 if (currentNode.left):
 self.insertNode(currentNode.left, key)
 else:
 currentNode.left = Node(key)
 elif (key > currentNode.key):
 if (currentNode.right):
 self.insertNode(currentNode.right, key)
 else:
 currentNode.right = Node(key)
```

## ABR: Classe ABR

```
class ABR:
 ...
 def find(self, key):
 return self.findNode(self.root, key)

 def findNode(self, currentNode, key):
 if (currentNode is None):
 return False
 elif (key == currentNode.key):
 return True
 elif (key < currentNode.key):
 return self.findNode(currentNode.left, key)
 else:
 return self.findNode(currentNode.right, key)
```

## ABR: Classe ABR

```
class ABR:
 ...
 def inorder(self):
 def _inorder(v):
 if(v is None):
 return
 if(v.left is not None):
 _inorder(v.left)
 print(v.key)
 if(v.right is not None):
 _inorder(v.right)

 _inorder(self.root)
```

## ABR: main

```
def main():
 tree = ABR()
 tree.insert(4)
 tree.insert(5)

 for x in range(20, 10, -1):
 tree.insert(x)
 print tree.find(5)
 print tree.find(2)
 tree.inorder()

if __name__ == "__main__":
 main()
```

La parte in fondo serve per eseguire main solo quando chiamato come programma autonomo.

## Homework N.3: Alberi binari di ricerca

- Scrivere i programmi python che implementano l'albero binario di ricerca a partire dal codice per ABR visto in questa lezione
  - ▶ Probabilmente l'implementazione dell'ABR vista in questa lezione deve essere completata con eventuali metodi o attributi mancanti
- Scrivere i programmi per eseguire un insieme di test che ci permettano di comprendere vantaggi e svantaggi di ABR
- Scrivere una relazione che descriva quanto fatto
- Anche questo esercizio NON è obbligatorio

# Esercizi

- Gli **homework** non sono obbligatori, ma se svolti durante il corso sono utili a comprendere gli argomenti e possono sempre essere discussi con il docente
- Gli **esercizi** sono necessari per superare l'esame di Laboratorio di Algoritmi.
  - ▶ Vengono assegnati durante l'anno
  - ▶ ma devono essere terminati almeno due settimane prima dell'appello e discussi con il docente al ricevimento studenti
  - ▶ Il giorno dell'orale vengono poste domande sugli esercizi svolti



# **Esercizi Laboratorio di Algoritmi**

**A.A. 2018-19**

## Esercizio N.1: Valutazione performance Python

- Questo **deve essere consegnato** per l'esame
- E deve essere discusso con il docente (al ricevimento) secondo le tempistiche indicate sul sito di e-learning
- Scrivere uno o più programmi Python che permettono di:
  - ▶ Generare un vettore casuale e/o con caratteristiche opportune
  - ▶ Implementare l'algoritmo `Insertion_sort()`
  - ▶ Implementare a scelta `Quick_sort()` o `Merge_sort()`
- Scrivere un programma che esegua dei test misurando i tempi di esecuzione degli algoritmi precedenti su:
  - ▶ Dati di dimensione crescente (array con 10, 1000, 1000, ... dati)
  - ▶ Arrestandosi quando il tempo di esecuzione è maggiore di alcuni minuti
  - ▶ Usando dati casuali e dati che costituiscano il caso migliore/peggiore per un algoritmo
- Scrivere una relazione in cui si descrivono gli esperimenti svolti (riportando i risultati sperimentali ottenuti) e si analizzano i risultati alla luce della teoria
- Impacchettare i file .py e la relazione (tex e PDF) in un file `Cognome_Matricola_Es1.zip` e caricarlo su e-l!

# Esercizi

- Esercizio 1 +  
due a scelta tra i seguenti (almeno uno sui grafi):
  - ▶ Alberi Rosso-Neri vs Alberi Binari di Ricerca
  - ▶ Alberi Rosso-Neri + Statistiche d'ordine dinamiche
  - ▶ Hash con concatenamento vs indirizzamento aperto
  - ▶ Edit distance + N-gram
  - ▶ Componenti fortemente connesse
  - ▶ Componenti connesse e MST Kruskal o Prim
  - ▶ Studio dei tempi di esecuzione per vari algoritmi con Raspberry PI e/o Arduino
  - ▶ Componenti connesse in immagine documenti (caratteri) con Raspberry PI
  - ▶ Su proposta studenti (ad esempio guardando algoritmi sul libro non trattati a lezione)
- In alternativa è possibile scegliere tre esercizi tra i precedenti escludendo l'esercizio 1

## Esercizio A: Alberi rosso-neri vs Alberi Binari di Ricerca

- Vogliamo analizzare le differenze tra Alberi Binari di Ricerca e Alberi Rosso-Neri
- Per fare questo dovremo:
  - ▶ Scrivere i programmi Python che implementano l'albero RN a partire dal codice per ABR visto a lezione
    - ★ Probabilmente l'implementazione dell'ABR vista a lezione deve essere completata con eventuali metodi o attributi mancanti
    - ★ Per alberi RN non è obbligatorio considerare la cancellazione, ma l'inserimento non si può omettere
  - ▶ Scrivere i programmi per eseguire un insieme di test che ci permettano di comprendere vantaggi e svantaggi di ABR
  - ▶ Scrivere una relazione che descriva quanto fatto

## Esercizio B: Alberi rosso-neri con Statistiche d'ordine dinamiche

- Vogliamo confrontare varie implementazioni di statistiche d'ordine dinamiche
- Per fare questo dovremo:
  - ▶ Scrivere i programmi Python che implementano l'albero RN a partire dal codice per ABR visto a lezione
    - ★ Per alberi RN non è obbligatorio considerare la cancellazione, ma l'inserimento non si può omettere
    - ★ Implementare statistiche d'ordine dinamiche utilizzando liste concatenate
  - ▶ Scrivere i programmi per eseguire un insieme di test che ci permettano di comprendere vantaggi e svantaggi dei due modi per realizzare statistiche d'ordine dinamiche
  - ▶ Scrivere una relazione che descriva quanto fatto

## Esercizio C: Hash

- Vogliamo capire qual è il comportamento delle tabelle hash al crescere del fattore di caricamento  $\alpha = \frac{n}{m}$
- Per fare questo scriveremo:
  - ▶ Un programma che implementa le tabelle hash con gestione delle collisioni basate su concatenamento e su indirizzamento aperto (ispezione lineare)
    - ★ La funzione hash deve essere calcolata col metodo delle divisioni
  - ▶ Oltre al costruttore devono essere implementati inserimento, cancellazione e ricerca per i due metodi
  - ▶ Un programma che conta quante collisioni si hanno eseguendo un numero variabile di inserimenti in una tabella hash
  - ▶ Un programma che esegue gli esperimenti
  - ▶ Una relazione

## Esercizio D: Edit distance

- Vogliamo studiare l'algoritmo di Edit Distance e come si possa utilizzare per trovare parole vicine ad una query  $Q$  in un lessico  $L$ 
  - ▶ E' possibile scaricare lessici di parole dal Web (esempio: <https://github.com/napolux/paroleitaliane>)
- Per fare questo dovremo scrivere programmi Python che:
  - ▶ implementano l'edit distance
  - ▶ costruiscono indici di n-gram di parole (si può scegliere il modo in cui si costruiscono gli indici)
  - ▶ data una query  $Q$  trovano la parola più vicina
  - ▶ eseguono un insieme di test che ci permettano di comprendere vantaggi e svantaggi dell'utilizzo di indici di n-gram per eseguire le query
- Scrivere una relazione che descriva quanto fatto

## Esercizio E: Componenti fortemente connesse

- Vogliamo studiare l'algoritmo per calcolare le componenti fortemente connesse di un grafo
- Per fare questo dovremo scrivere i seguenti programmi Python:
  - ▶ generazione di grafi casuali con un numero di nodi a scelta ed una determinata probabilità di presenza di archi tra vertici (es. partire da una matrice di adiacenza con tutti 0 e poi cambiare archi ad 1 con una certa probabilità )
  - ▶ implementare la visita in profondità di grafi
  - ▶ implementare l'algoritmo per trovare le componenti fortemente connesse nei grafi
  - ▶ Un programma che permetta di condurre esperimenti su grafi casuali con dimensione crescente e con probabilità di presenza di archi crescente.
- Scrivere una relazione che descriva quanto fatto



## Esercizio F: Componenti connesse e MST

- Per studiare gli algoritmi per trovare le componenti connesse si scrivano i seguenti programmi:
  - ▶ generazione di grafi casuali con un numero di nodi a scelta ed una determinata probabilità di presenza di archi tra vertici (es. partire da una matrice di adiacenza con tutti 0 e poi cambiare archi ad 1 con una certa probabilità )
  - ▶ generazione di grafi pesati casuali
  - ▶ ricerca delle componenti connesse
  - ▶ algoritmo di Kruscal
    - ★ struttura dati UNION-FIND
  - ▶ in alternativa algoritmo di Prim
    - ★ struttura dati coda con priorità (con min-hash)
  - ▶ Un programma che permetta di condurre esperimenti su grafi casuali con dimensione crescente e con probabilità di presenza di archi crescente.
- Scrivere inoltre una relazione che descriva quanto fatto

## Esercizio G: tempi di esecuzione con Raspberry PI e/o Arduino

- Questo esercizio è volutamente generico in modo da poter avvogliere proposte di studenti
- L'idea è lo studio e il confronto delle prestazioni di vari algoritmi (potrebbero andare bene anche algoritmi di ordinamento) su dispositivi mobili
- Va bene sia l'utilizzo di Raspberry PI che Arduino (o entrambi)
- Non è necessario conoscere già come funzionano, ne parliamo a ricevimento studenti
- Questi dispositivi sono disponibili presso il laboratorio di informatica per essere utilizzabili dagli studenti (contattare il docente)
- Ovviamente anche in questo caso si implementano vari algoritmi, si eseguono dei test e si scrive una relazione

## Esercizio H: Componenti connesse in immagini

- Si vuole scrivere un programma per trovare le componenti connesse in immagini di documenti (corrispondenti quindi a caratteri) con Raspberry PI e una camera a questo collegata
- I dettagli tecnici saranno discussi con il docente a lezione (ci sono vari modi per svolgere questo esercizio)
- Questi dispositivi (Raspberry e camera) sono disponibili presso il laboratorio di informatica per essere utilizzabili dagli studenti (contattare il docente)
- Ovviamente anche in questo caso si implementano vari algoritmi, si eseguono dei test e si scrive una relazione

## Esercizio I: su proposta di studenti

- Gli studenti possono sempre proporre autonomamente esercizi (ovviamente concordandoli con il docente a ricevimento studenti)
- Ad esempio guardando algoritmi sul libro non trattati a lezione o non coperti da questi esercizi
- Ovviamente anche in questo caso si implementano vari algoritmi, si eseguono dei test e si scrive una relazione

# In generale

Nota: ciò che non è specificato può essere scelto come si ritiene più opportuno

## Cosa consegnare

- Il codice sviluppato: programmi .py non notebook!
- I dati di ingresso e uscita per poter duplicare gli esperimenti
- Descrivere gli esperimenti condotti in una relazione, allegando sia il file .pdf che il .tex e tutte le illustrazioni
  - ▶ cosa ci si aspetta per la relazione è descritto in dettaglio nei lucidi generali su Laboratorio di Algoritmi
- Impacchettare in `Cognome_Matricola_Es_X.zip` e caricare su e-l