

Complementi di Programmazione

Struttura progetto Python

CdL Informatica - Università degli studi di Modena e Reggio Emilia
AA 2023/2024

Filippo Muzzini

Struttura progetto

E' buona abitudine organizzare un progetto software in più moduli (e file).

Questo rende più leggibile e navigabile il progetto

In certi linguaggi (specialmente quelli compilati) vengono ottimizzati i tempi di compilazione

- I moduli senza modifiche non vengono ri-compilati

Maggiore riutilizzo di moduli in altri progetti

Struttura progetto

E' buona abitudine organizzare un progetto software in più moduli (e file).

Questo rende più leggibile e navigabile il progetto

In certi linguaggi (specialmente quelli compilati) vengono ottimizzati i tempi di compilazione

- I moduli senza modifiche non vengono ri-compilati

Maggiore riutilizzo di moduli in altri progetti

Struttura progetto

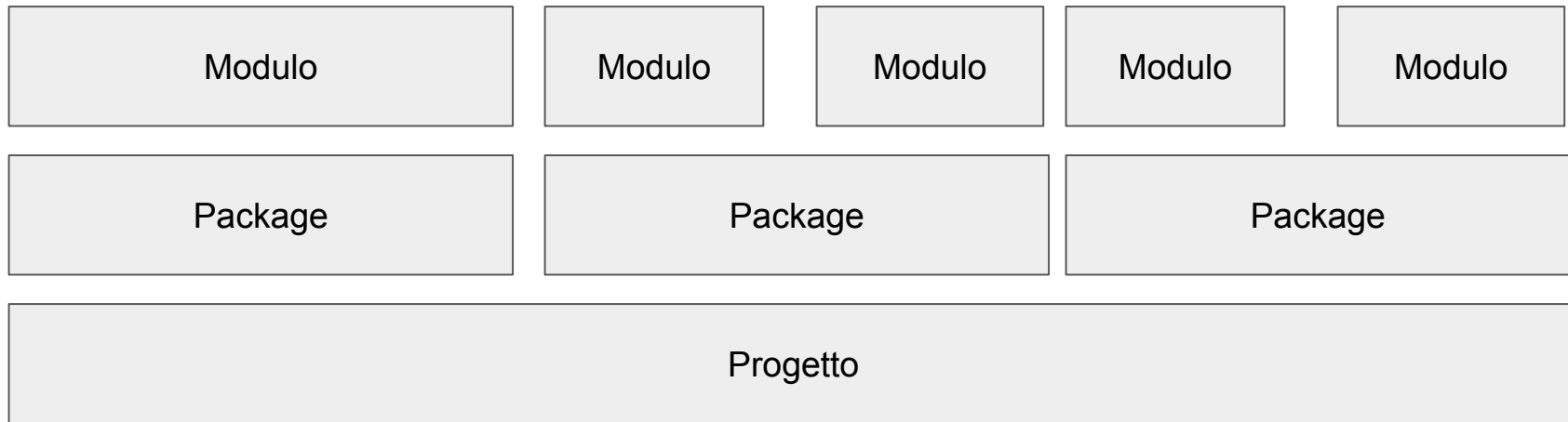
In un modulo vi sono più entità relative a quel modulo.

- un modulo deve avere un fine preciso

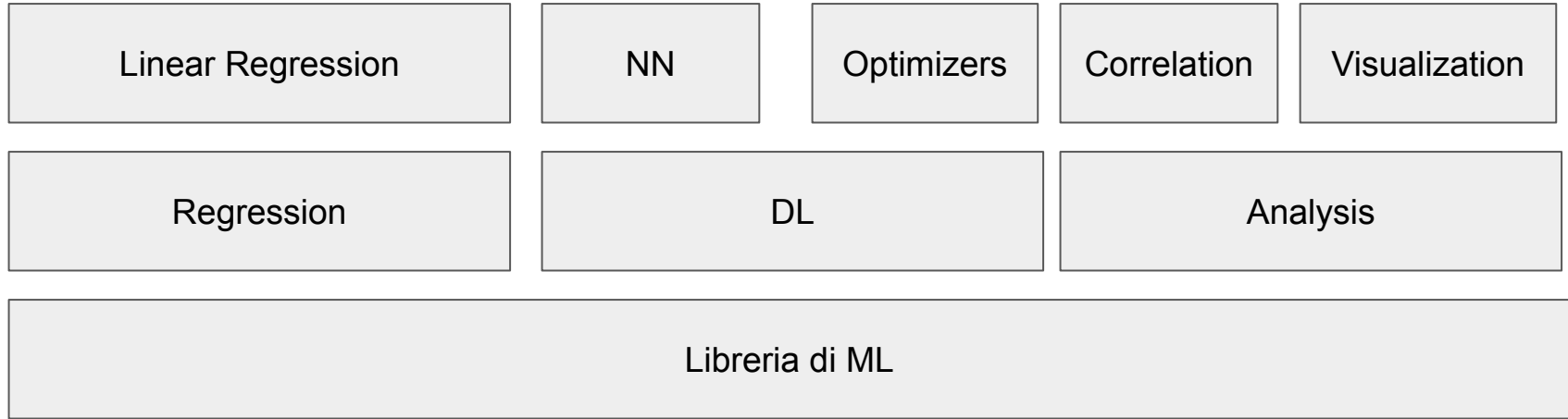
Più moduli correlati formano un package.

- Il package racchiude più moduli inerenti allo stessa area del programma

Struttura progetto



Struttura progetto



Struttura progetto Python

In Python i moduli corrispondono ai file sorgenti.

un file -> un modulo

nello stesso file possono essere definite le entità utili a quel modulo

- funzioni
- costanti
- classi

Struttura progetto Python

In Python un package è quindi fatto da più file (più moduli).

Un package è una directory

E' possibile organizzare il progetto in package e sotto-package

Per far sì che una directory sia vista come package è necessario inserire dentro di essa un file `__init__.py` (anche vuoto)

Utilizzo di un modulo

Per utilizzare un modulo esso va importato.

import <modulo>

Da questo momento è possibile utilizzare le entità del modulo

Utilizzo di un modulo

Per utilizzare un modulo esso va importato.

import <modulo>

Importazione di una entità specifica

from <modulo> import <entità>


Importazione di tutte le entità

from <modulo> import *

Utilizzo di un modulo

Per utilizzare un modulo esso va importato.

Utilizzo

import <modulo>  **<modulo>.<entità>**

Importazione di una entità specifica

from <modulo> import <entità>  **<entità>**

Importazione di tutte le entità

from <modulo> import *  **<entità>**

Utilizzo di un modulo

Per utilizzare un modulo contenuto in un package la sintassi è la stessa

```
import <package>.<modulo>
```

```
import <package>.<sub-package>.<modulo>
```

```
import <package>.<sub-package>.<modulo>.<entità>
```

Utilizzo di un modulo

Quando si importa un modulo viene eseguito quel file.

Quindi, oltre a definire le entità, è possibile eseguire delle inizializzazioni

Utilizzo di un modulo

```
s = "Risultato"
```

```
def fun(x):
```

```
    print(f'{s}: {x}')
```

Utilizzo di un modulo

Al momento dell'import **s** verrà inizializzato e verrà creata l'entità **fun**

Occhio che anche **s** è un entità e potrà essere acceduta

Utilizzo di un package

Anche i package possono essere importati.

l'inizializzazione di un package può essere messa nel file `__init__.py`

Utilizzo di un package

Nel momento che importo il package verrà eseguita l'inizializzazione.

Se importo un modulo in un package verrà eseguita prima l'inizializzazione del package e poi del modulo.

Utilizzo di un package

Quando importo i moduli dei package in questa forma

```
from <package> import *
```

Utilizzo di un package

Quando importo i moduli dei package in questa forma

```
from <package> import *
```

non vengono importati tutti i moduli del package per motivi di performance
(vorrebbe dire eseguire tutte le inizializzazioni)

Utilizzo di un package

Quando importo i moduli dei package in questa forma

```
from <package> import *
```

non vengono importati tutti i moduli del package per motivi di performance

(vorrebbe dire eseguire tutte le inizializzazioni)

Di per se non viene importato nulla

Utilizzo di un package

Quando importo i moduli dei package in questa forma

```
from <package> import *
```

non vengono importati tutti i moduli del package per motivi di performance
(vorrebbe dire eseguire tutte le inizializzazioni)

Di per se non viene importato nulla

Utilizzo di un package

Per dire quali moduli di un package importare bisogna specificarli nel file `__init__.py` tramite la lista speciale `__all__`

```
__all__ = ["modulo1", "modulo2"]
```

Librerie

L'insieme di package compone il nostro programma. Se esso è pensato per il riutilizzo in altri progetti lo chiamiamo **libreria**.

Librerie

È un pezzo di codice riutilizzabile che possiamo usare importandolo nel nostro programma e possiamo semplicemente usarlo importando quella libreria e chiamando il metodo di quella libreria con un punto.

E' organizzata in package (tipicamente)

Librerie

Oltre alle librerie standard (già incluse nell'interprete), Python ha un enorme catalogo di librerie pronte all'uso.

<https://docs.python.org/3/library/>

<https://pypi.org/>

Librerie standard

Le libreria standard le abbiamo già usate (Es. sys)

ma ve ne sono molte altre:

- string -> manipolazione stringhe
- datetime -> manipolazione date
- math -> funzioni matematiche
- ec...

Librerie standard

Inoltre su Pypi vi sono moltissime librerie.

Per installarle si usa 'pip3' (o 'pip')

pip3 install <libreria>

Librerie standard

pip è packet manager (come apt) che installa (e rimuove) librerie python.

Di default va a cercare tali librerie su Pypi ma è possibile specificare altri repository.

Librerie standard

pip è packet manager (come apt) che installa (e rimuove) librerie python.

Di default va a cercare tali librerie su Pypi ma è possibile specificare altri repository.

Librerie molto usate: <http://www.python.it/about/applicazioni/>

Librerie standard

Ma l'interprete dove pesca le librerie aggiuntive per importarle?

```
import sys
```

```
sys.path
```

Lista delle directory in cui vengono cercati i package.

In ordine

Librerie standard

Lista delle directory in cui vengono cercati i package.

In ordine

Notare che la prima voce è la directory corrente, ovvero prima si guardano i package del progetto

Librerie standard

Lista delle directory in cui vengono cercati i package.

In ordine

Notare che la prima voce è la directory corrente, ovvero prima si guardano i package del progetto

Normalmente è seguita dai path presenti nella variabile di ambiente PYTHONPATH e poi da /usr/lib/python.x.x

Ambienti

Modificando questi path si possono caricare altri package a runtime (`sys.path` è modificabile) o scegliere tra una versione di un package o un'altra

Ambienti

Modificando questi path si possono caricare altri package a runtime (sys.path è modificabile) o scegliere tra una versione di un package o un'altra

Spesso progetti diversi sullo stesso PC vogliono differenti versioni della stessa libreria

Ambienti

Modificando questi path si possono caricare altri package a runtime (`sys.path` è modificabile) o scegliere tra una versione di un package o un'altra

Spesso progetti diversi sullo stesso PC vogliono differenti versioni della stessa libreria

Per evitare conflitti si possono installare le due versioni in directory differenti e poi modificare il `PYTHONPATH` a seconda del progetto

Ambienti

Oppure creare un ambiente virtuale <https://docs.python.org/3/library/venv.html>

creazione ambiente

```
python3 -m venv /path/to/new/virtual/environment
```

attivazione ambiente

```
/path/to/new/virtual/environment/
```

Ambienti

Oppure creare un ambiente virtuale <https://docs.python.org/3/library/venv.html>

creazione ambiente

python3 -m venv /path/to/new/virtual/environment

attivazione ambiente

se non è installato venv
pip3 install virtualenv

/path/to/new/virtual/environment/

Ambienti

Una volta attivato l'ambiente pip3 installerà le librerie solo in questo ambiente

L'interprete cercherà le librerie in questo ambiente

Ogni progetto dovrebbe avere il suo ambiente per evitare conflitti

Ambienti

si disattiva un ambiente con

deactivate

Ambienti

E' possibile esportare le versioni delle librerie di un ambiente con pip3

pip3 freeze > requirements.txt

Ambienti

E' possibile esportare le versioni delle librerie di un ambiente con pip3

pip3 freeze > requirements.txt

Ed è possibile importarle sempre con pip3

pip3 install -r requirements.txt

Moduli e script

Come abbiamo visto i moduli eseguono codice quando vengono importati.

Stesso codice che verrebbe eseguito dall'interprete se il file del modulo venisse lanciato come script.

Tuttavia a volta si vuole distinguere questo comportamento:

Es. venv si comporta diversamente se lanciato per creare un ambiente (come abbiamo visto) o se importato per utilizzare le sue classi
(<https://docs.python.org/3/library/venv.html#api>)

Moduli e script

E' possibile distinguere ciò utilizzando la variabile **__name__**

__name__ prende il nome del modulo quando è importato

prende invece **__main__** quando il file è lanciato come script

Moduli e script

E' possibile distinguere ciò utilizzando la variabile `__name__`

`__name__` prende il nome del modulo quando è importato

prende invece `__main__` quando il file è lanciato come script

è buona norma mettere le istruzioni che si vogliono eseguire dentro un

```
if __name__ == "__main__":
```

```
    ...
```

Moduli e script

è buona norma mettere le istruzioni che si vogliono eseguire dentro un

```
if __name__ == "__main__":
```

```
    ...
```

In questo modo lo stesso file può essere importato senza problemi da un altro file come se fosse un modulo

Moduli e script

```
python3 -m venv /path/to/new/virtual/environment
```

Indica che il modulo deve essere lanciato come script... In pratica come fare `python3 venv.py` (ma non sapendo dove sia tale file meglio lasciarlo cercare all'interprete)

Build

Per distribuire il codice è buona norma buildarlo in un pacchetto (su Pypi sono uploadati in questa forma)

pip3 install build

python3 -m build

Non basta deve esserci un file **toml** che dia i metadati del progetto (come l'autore, le dipendenze ecc...)

chiamato **pyproject.toml**

Build

il toml può essere anche vuoto.

Alla fine del processo nella cartella dist vi sarà il pacchetto da poter distribuire.

Build

Nel toml si possono mettere molte informazioni:

<https://packaging.python.org/en/latest/guides/writing-pyproject-toml/#writing-pyproject-toml>