

Advanced Coding and Cloud Computation

...

Romolo Politi

Elenco Lezioni

Elenco Lezioni

- 16 ottobre 2023
- 19 ottobre 2023
- 24 ottobre 2023

Lezione del 16 ottobre 2023

Panoramica del Corso

Panoramica del Corso

Cloud

Struttura del Cloud
Dati nel Cloud
Calcolo nel Cloud

Dati

Dati e metadati
Archiviazione
DB Relazionali e non

Calcolo

Recupero
Manipolazione
Visualizzazione

Ambiente

Virtualizzazione e container
Microservices
DevOps

Programmazione

Fondamenti di programmazione
Python
Versioning e Documentazione

Tools

- La presentazione e gli esempi del corso sono su GitHub:
 - <https://github.com/RomoloPoliti-INAF/PhDCourse2023b>
- Per gli esempi utilizzeremo Python 3.12
- Come framework di sviluppo Microsoft Visual Studio Code
 - <https://code.visualstudio.com>

Struttura del Corso

- La lista degli argomenti mostrata in precedenza è stata costruita per categorie.
- Noi seguiremo un percorso guidato dagli esempi per meglio capire la filosofia che c'è dietro.
- Dopo l'introduzione alla programmazione svilupperemo un esempio di programma complesso (Macchina a Stati).
- In ultimo svilupperemo una WebApp e la prepareremo per la distribuzione su container
- Per alcuni argomenti non scenderemo nel dettaglio perché lo scopo del corso è dare una visione generale della tematica.
- Anche se non verranno discussi, molti dettagli saranno nelle slide o nei link riportati.

Definizione di Cloud

Cosa è il Cloud

E' una erogazione di servizi offerti su richiesta da un fornitore ad un utente finale attraverso la rete internet (come l'archiviazione, l'elaborazione o la trasmissione dati), a partire da un insieme di risorse preesistenti, configurabili e disponibili in remoto sotto forma di architettura distribuita.

wikipedia

Tipi di Cloud

In Promise



Out Promise



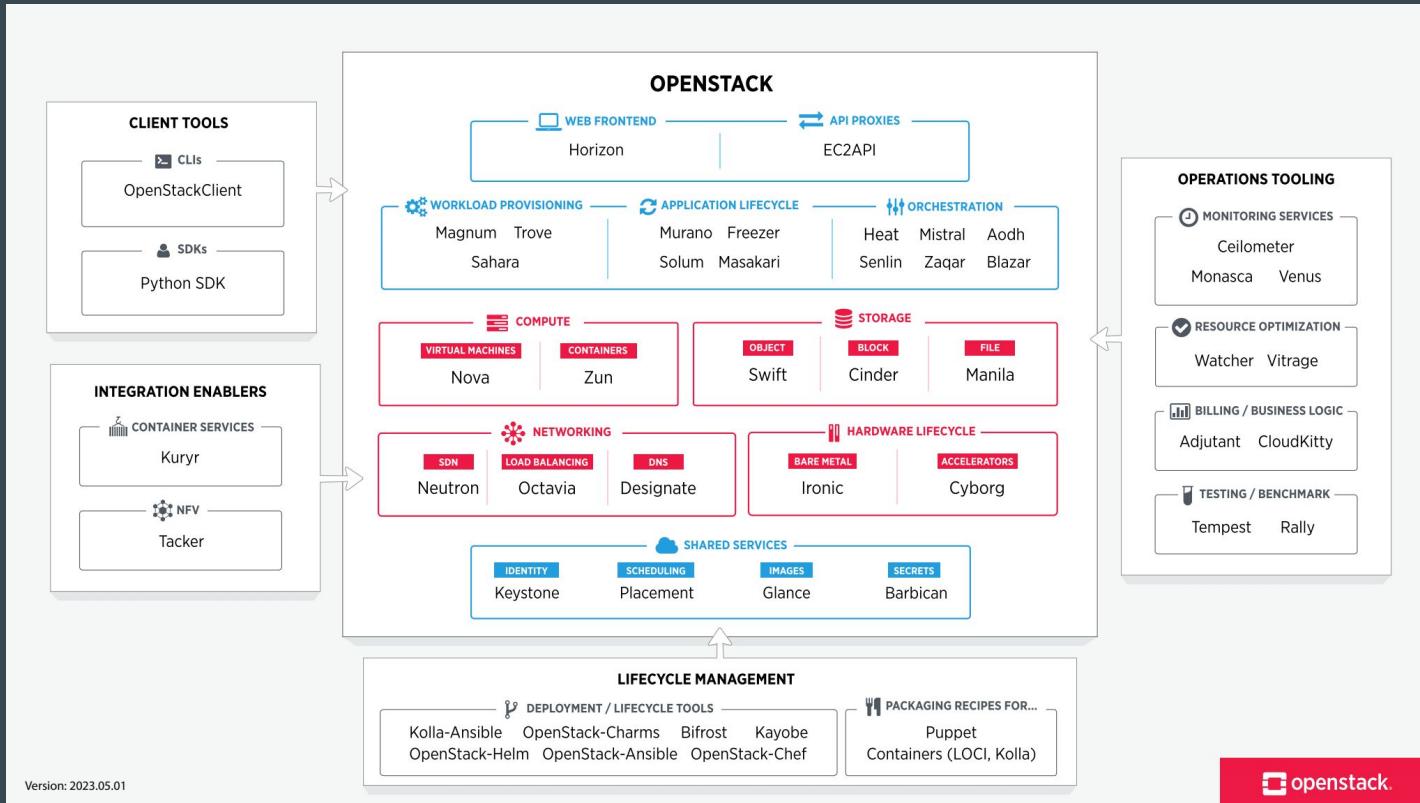
Google Cloud



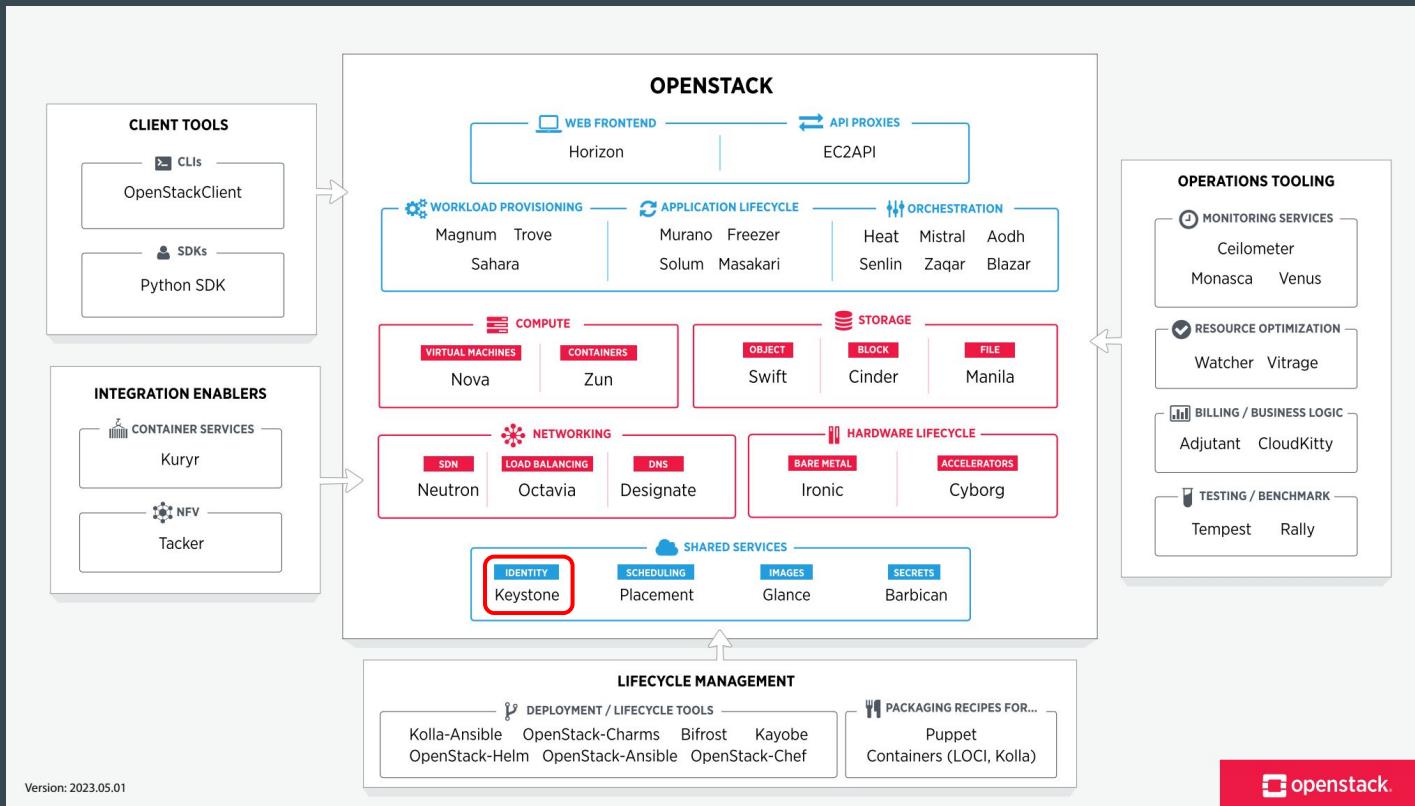
Struttura del Cloud

Struttura di un Cloud

<https://www.openstack.org/>

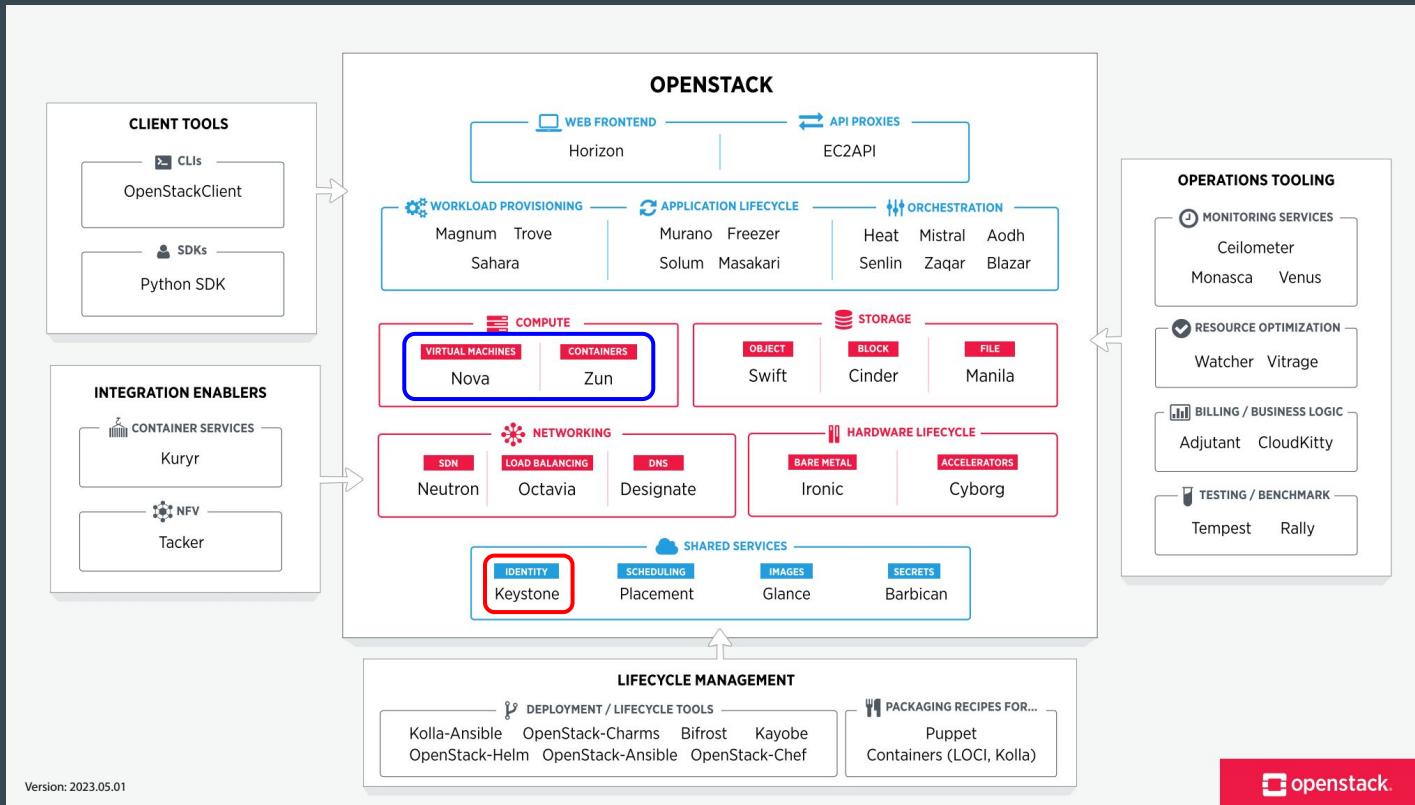


Struttura di un Cloud



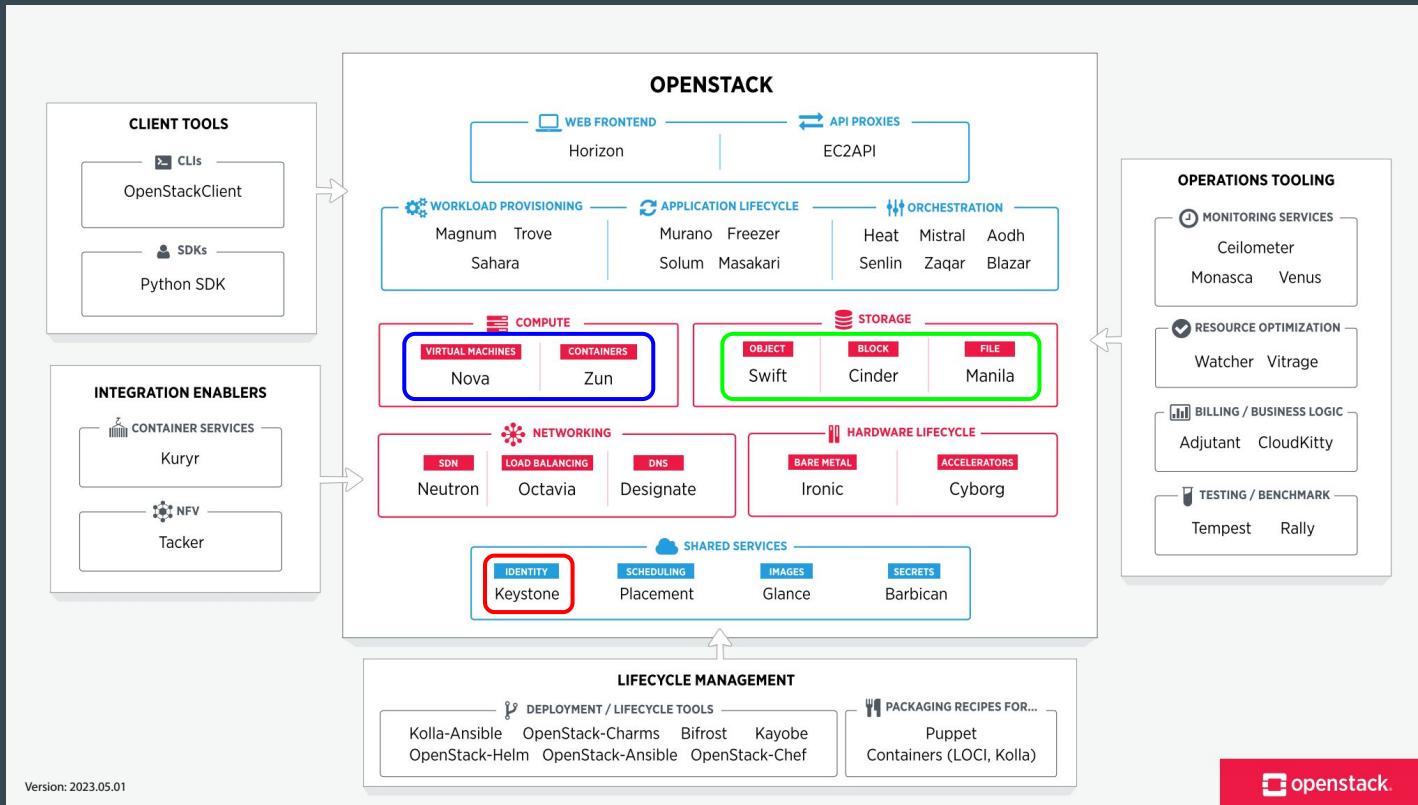
- Identity

Struttura di un Cloud



- Identity
- Compute

Struttura di un Cloud



- Identity
- Compute
- Storage

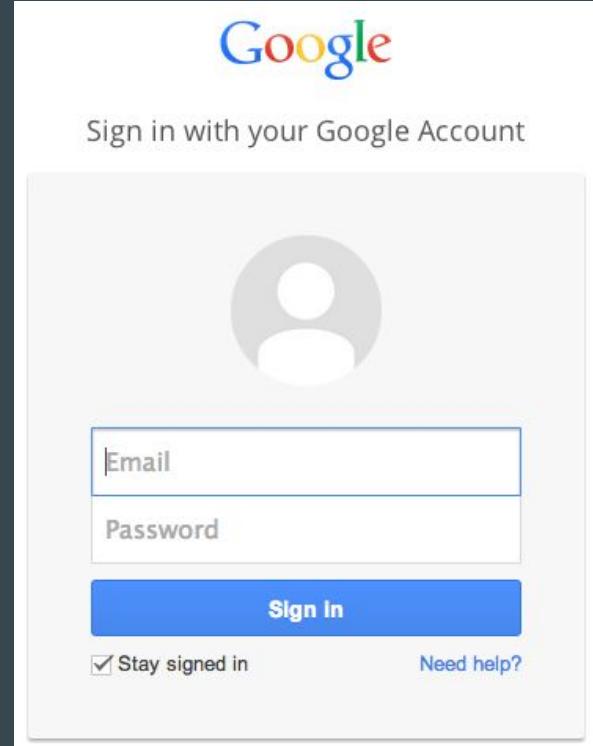
Componenti Principali

- IAM (Identity and Access Management)



Componenti Principali

- IAM (Identity and Access Management)
 - verifica identità
 - lista di risorse dedicate
 - privilegi
 - Credito (cloud off premise)



The image shows a screenshot of a Google sign-in page. At the top right is the Google logo. Below it is the text "Sign in with your Google Account". In the center is a large, light-gray circular placeholder for a profile picture. Below the placeholder are two input fields: one for "Email" and one for "Password", both with placeholder text. To the right of the password field is a "Sign in" button in a blue gradient color. At the bottom left is a checked checkbox labeled "Stay signed in". At the bottom right is a link "Need help?".

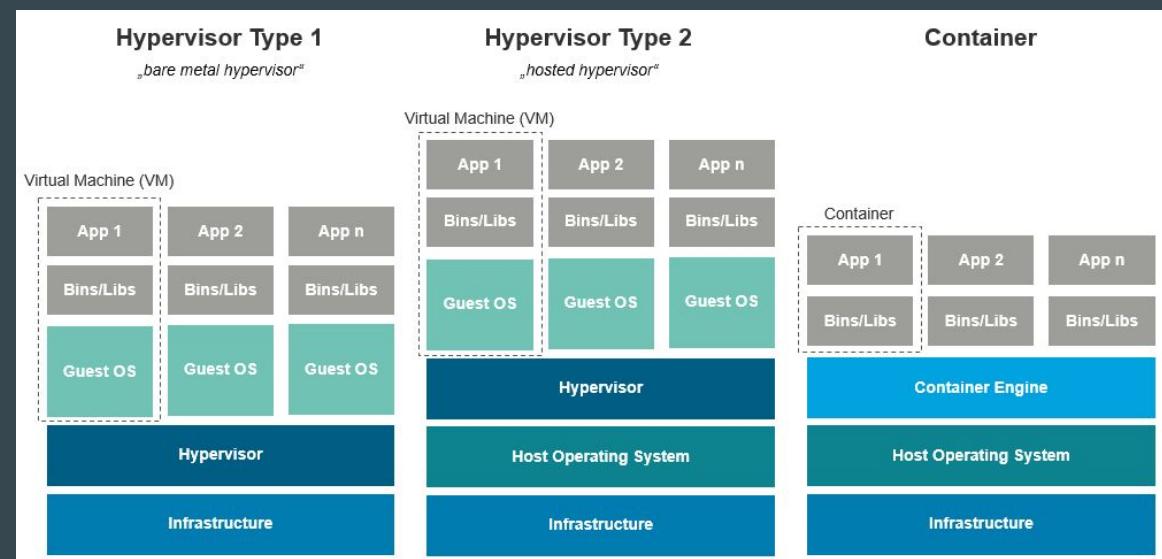
Componenti Principali

- IAM (Identity and Access Management)
 - verifica identità
 - lista di risorse dedicate
 - privilegi
 - Credito (cloud off premise)
- Compute Services



Componenti Principali

- IAM (Identity and Access Management)
 - verifica identità
 - lista di risorse dedicate
 - privilegi
 - Credito (cloud off premise)
- Compute Services



Componenti Principali

- IAM (Identity and Access Management)
 - verifica identità
 - lista di risorse dedicate
 - privilegi
 - Credito (cloud off premise)
- Compute Services
- Storage Services

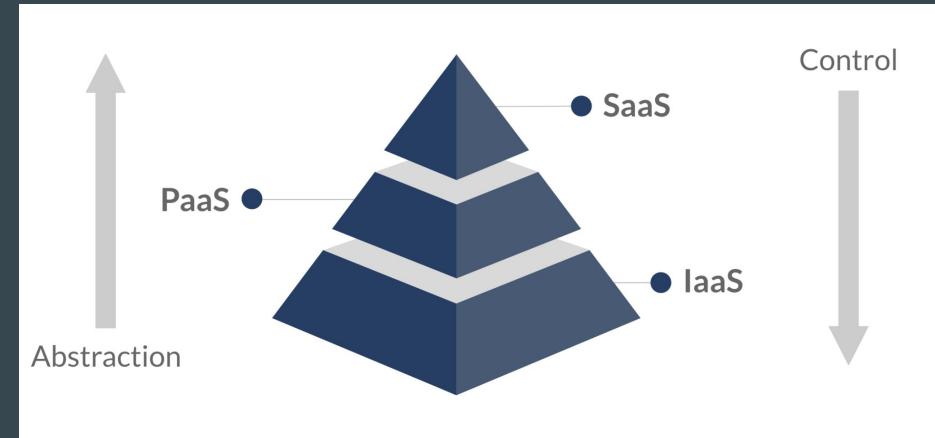


Servizi Cloud

- Infrastructure as a Service
- Platform as a Service
- Software as a Service

Servizi Cloud

- Infrastructure as a Service
- Platform as a Service
- Software as a Service

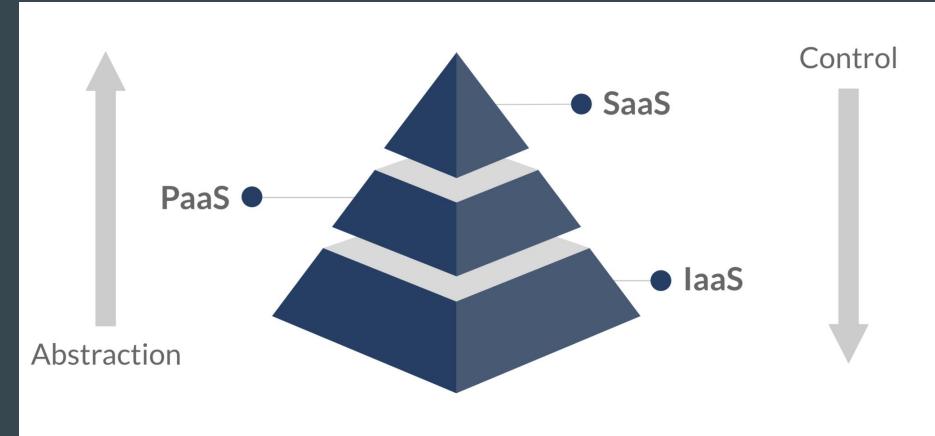


IaaS

Il provider offre un hardware virtuale (CPU, RAM, spazio e schede di rete) e quindi la flessibilità di un'infrastruttura fisica, senza l'onere per l'utente, della gestione fisica dell'hardware

Servizi Cloud

- Infrastructure as a Service
- Platform as a Service
- Software as a Service

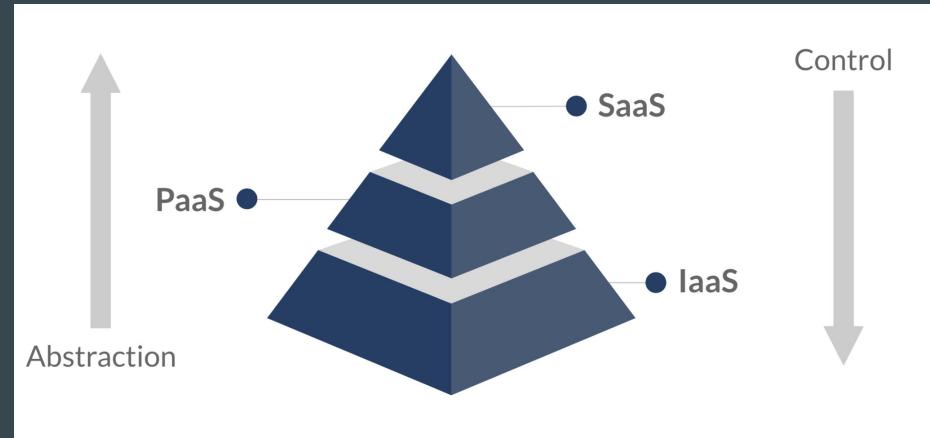


PaaS

Il provider si occupa dell'infrastruttura hardware, mentre l'utente dovrà installare il sistema operativo e occuparsi di sviluppare la sua applicazione

Servizi Cloud

- Infrastructure as a Service
- Platform as a Service
- Software as a Service

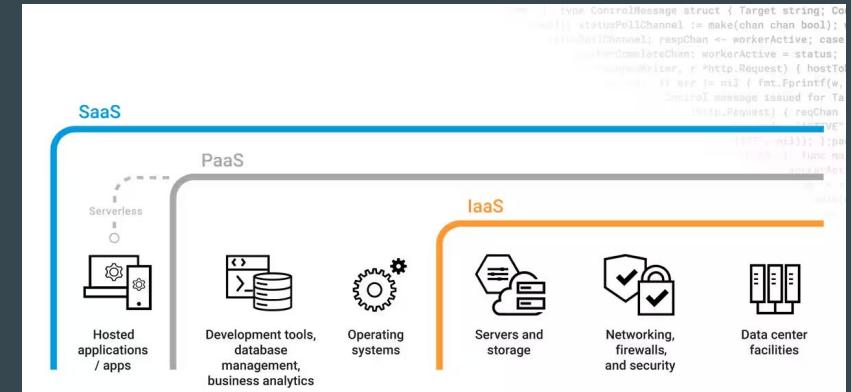
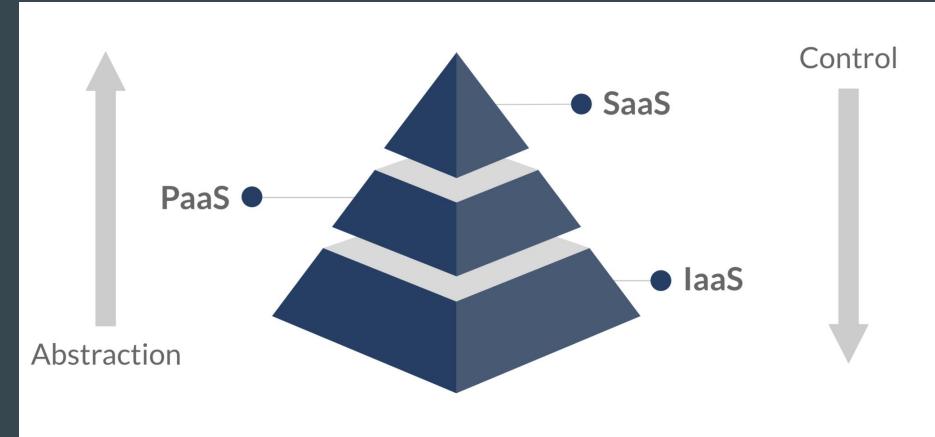


SaaS

L'utente finale non ha bisogno di nessuna conoscenza informatica per utilizzare l'applicazione o i servizi erogati. I servizi sono utilizzabili semplicemente con una connessione internet e un browser.

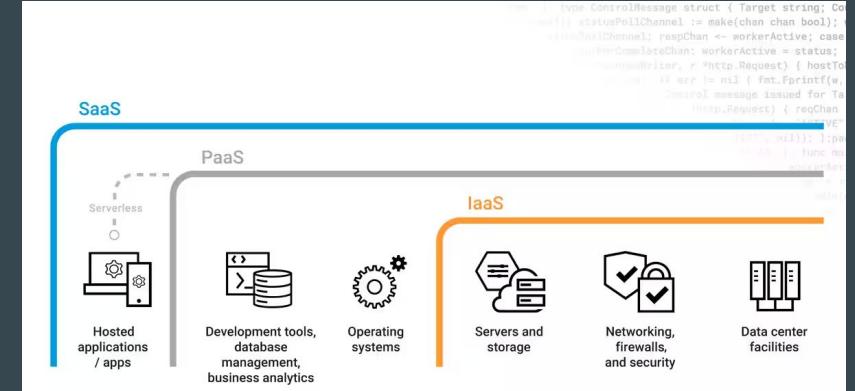
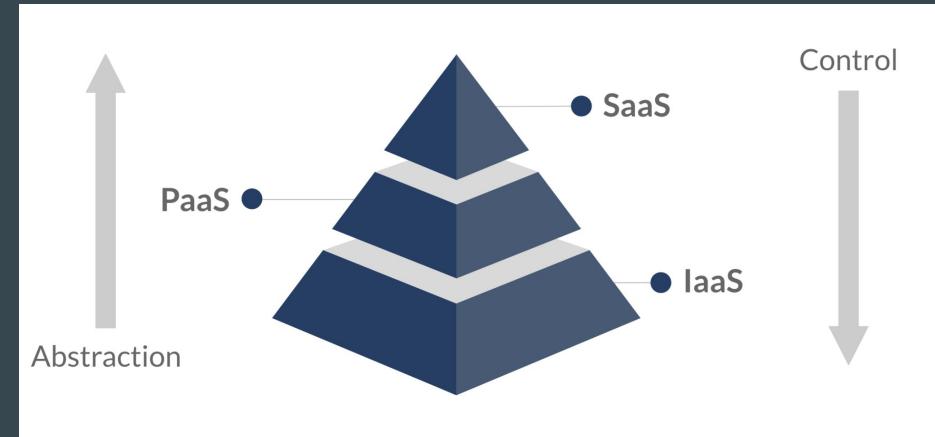
Servizi Cloud

- Infrastructure as a Service
- Platform as a Service
- Software as a Service



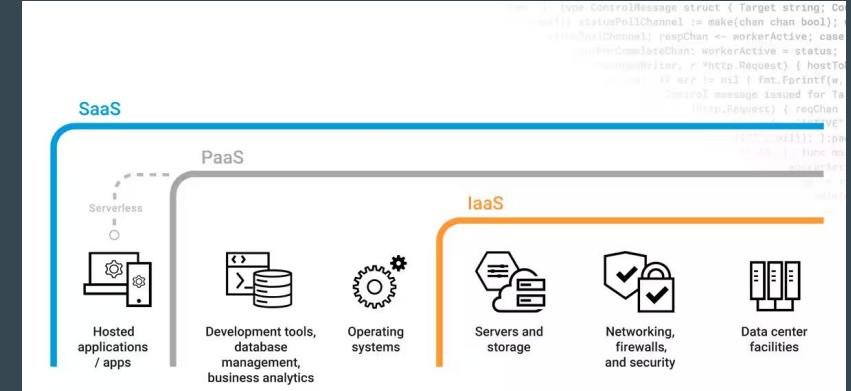
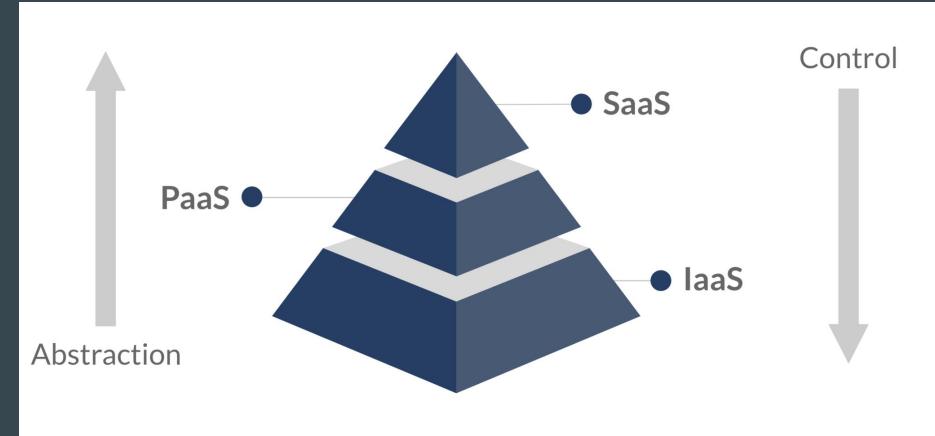
Servizi Cloud

- Infrastructure as a Service
- Platform as a Service
- Software as a Service
- Data as a Service



Servizi Cloud

- Infrastructure as a Service
- Platform as a Service
- Software as a Service
- Data as a Service



Dati e Metadati

Definizione di Dato

Un dato (dal latino *datum* dono, cosa data) è una descrizione elementare codificata di un'informazione, un'entità, di un fenomeno, di una transazione, di un avvenimento o di altro.

Un dato (in informatica) può avere dimensione da 1 bit (booleano) sino a migliaia di milioni di byte.

Definizione di Metadato

Il metadato è, letteralmente, "(dato) per mezzo di un (altro) dato", è un'informazione che descrive un insieme di dati.

Un esempio tipico di metadati è costituito dalla scheda del catalogo di una biblioteca, la quale contiene informazioni circa il contenuto e la posizione di un libro, cioè dati riguardanti più dati che si riferiscono al libro. Un altro contenuto tipico dei metadati può essere la fonte o l'autore dell'insieme di dati descritto, oppure le modalità d'accesso con le eventuali limitazioni.

Un metadato può essere anche un dato aggiunto all'insieme delle informazioni per altri scopi. Ad esempio, se alla scheda del libro della biblioteca aggiungo un ID, ossia un identificatore univoco, quest'ultimo è un metadato.

Esempio Dato - Metadato



X Informazioni

Aggiungi una descrizione

DETTAGLI

20 lug 2019
sab, 13:46 GMT+02:00

motorola moto g(6)
f/1.8 1/899 3.95 mm ISO100

IMG_20190720_134639156_HDR.jpg
12.6 MP 4996 x 3072

Cancata da un dispositivo Android

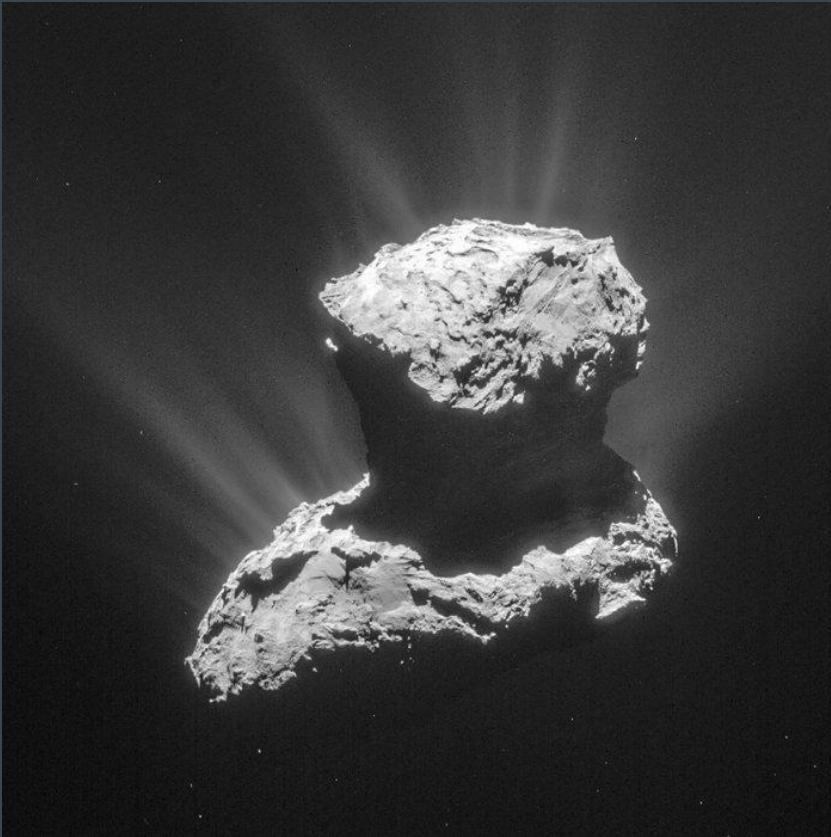
Backup eseguito
Risparmio spazio di archiviazione. Scopri di più
Questo elemento non occupa spazio di archiviazione dell'account. Scopri di più

Amalfi Provincia di Salerno

Pontone
Museo della Carta
Terrazza dell'Infinito
Lido di Roseto
Spaggia di Castiglione
POGEROLA
Attrani
Amalfi
Santa Caterina

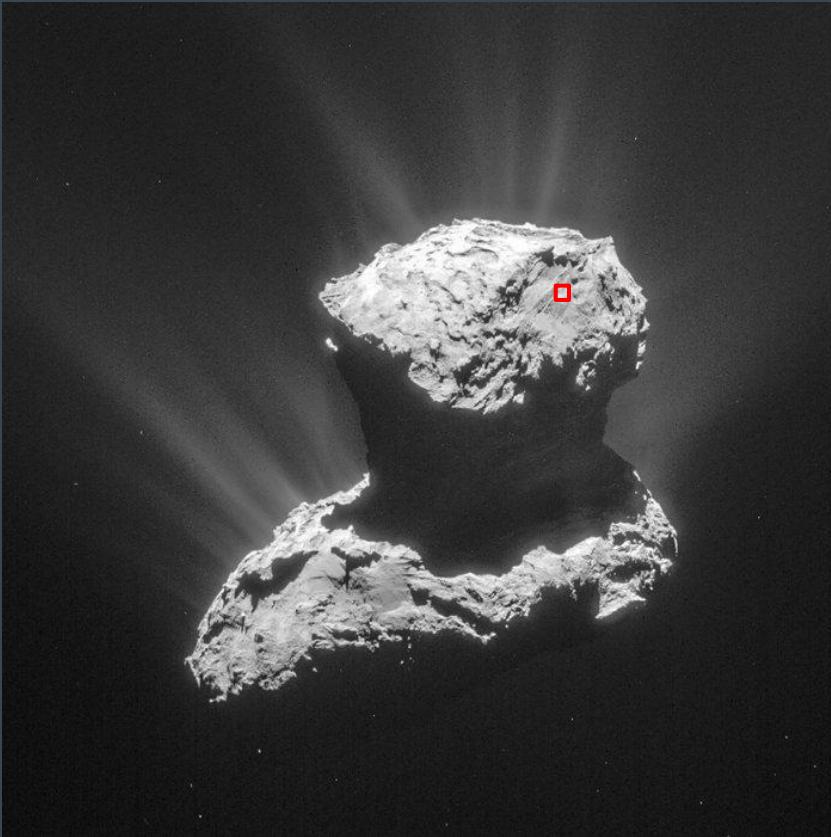
Map date 02/2023

Esempio in Planetologia



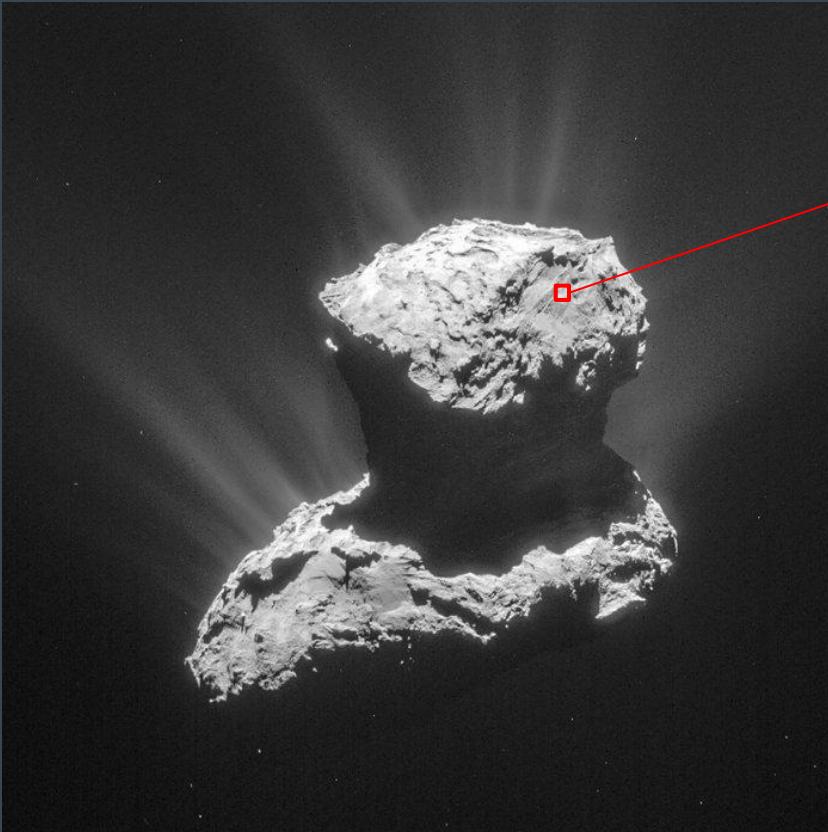
Quale è il dato in questo caso?

Esempio in Planetologia



Quale è il dato in questo caso?

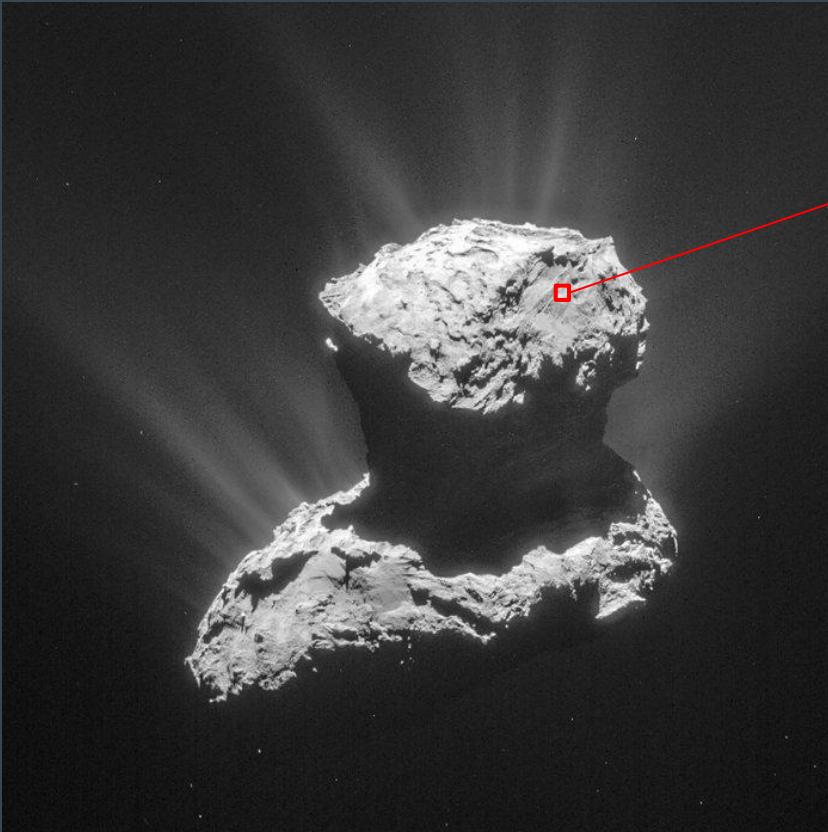
Esempio in Planetologia



Quale è il dato in questo caso?

Il pixel è rappresentato come un numero in virgola mobile a 32 bit.

Esempio in Planetologia

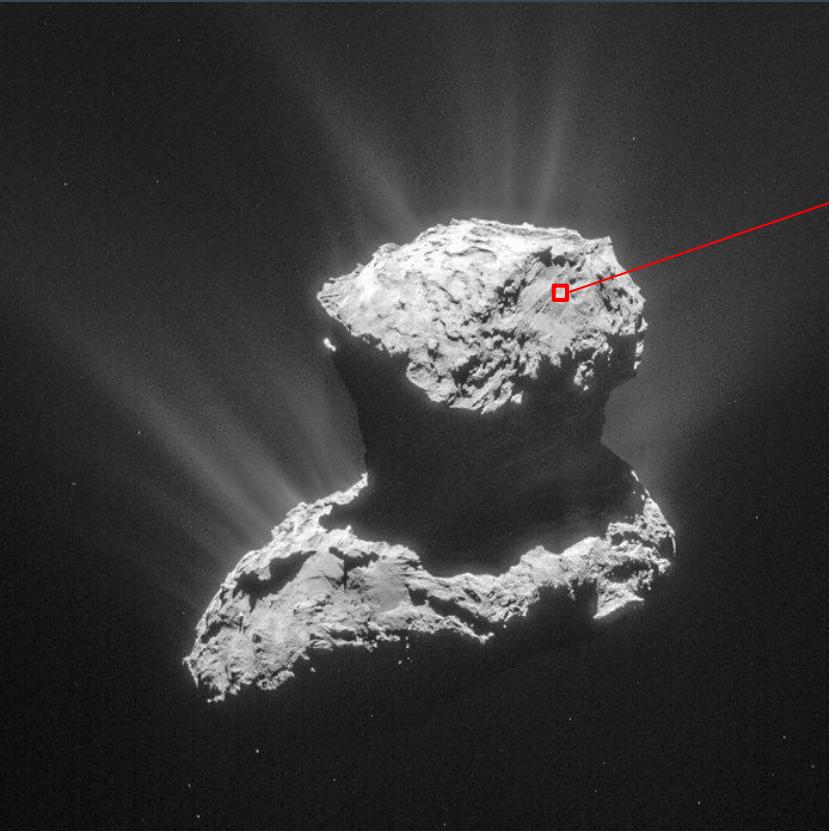


Quale è il dato in questo caso?

Il pixel è rappresentato come un numero in virgola mobile a 32 bit.

Ha significato?

Esempio in Planetologia



Quale è il dato in questo caso?

Il pixel è rappresentato come un numero in virgola mobile a 32 bit.

Ha significato?

La risposta è **NO**.

Si ha necessità di conoscere

- illuminazione,
- posizione della cometa,
- posizione dello spacecraft,
- tempi di esposizione,
- modalità di acquisizione,
- georeferenziazione del pixel

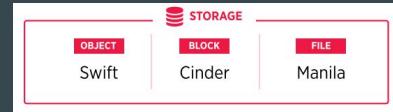
Dati nel Cloud

Archiviazione nel Cloud



<https://www.ibm.com/topics/block-storage>

Archiviazione nel Cloud



Il **block storage** suddivide i dati in componenti separati composti da blocchi di dati di dimensioni fisse, ognuno dotato di un identificatore univoco. Il block storage permette al sistema di storage sottostante di recuperarlo indipendentemente dalla posizione in cui viene memorizzato.

Archiviazione nel Cloud



Il **block storage** suddivide i dati in componenti separati composti da blocchi di dati di dimensioni fisse, ognuno dotato di un identificatore univoco. Il block storage permette al sistema di storage sottostante di recuperarlo indipendentemente dalla posizione in cui viene memorizzato.

Il **file storage** è il formato di storage maggiormente conosciuto: i dati vengono archiviati in file con cui è possibile interagire, contenuti in cartelle all'interno di una directory file gerarchica.

Archiviazione nel Cloud



Il **block storage** suddivide i dati in componenti separati composti da blocchi di dati di dimensioni fisse, ognuno dotato di un identificatore univoco. Il block storage permette al sistema di storage sottostante di recuperarlo indipendentemente dalla posizione in cui viene memorizzato.

Il file storage è il formato di storage maggiormente conosciuto: i dati vengono archiviati in file con cui è possibile interagire, contenuti in cartelle all'interno di una directory file gerarchica.

L'**object storage** è un formato di storage in cui i dati sono archiviati in unità separate chiamate oggetti. Ciascuna unità ha un identificatore univoco, o chiave, che ne permette l'individuazione indipendentemente dalla posizione in cui sono memorizzate in un sistema distribuito.

Archiviazione nel Cloud



Il **block storage** suddivide i dati in componenti separati composti da blocchi di dati di dimensioni fisse, ognuno dotato di un identificatore univoco. Il block storage permette al sistema di storage sottostante di recuperarlo indipendentemente dalla posizione in cui viene memorizzato.

Il **file storage** è il formato di storage maggiormente conosciuto: i dati vengono archiviati in file con cui è possibile interagire, contenuti in cartelle all'interno di una directory file gerarchica.

L'**object storage** è un formato di storage in cui i dati sono archiviati in unità separate chiamate oggetti. Ciascuna unità ha un identificatore univoco, o chiave, che ne permette l'individuazione indipendentemente dalla posizione in cui sono memorizzate in un sistema distribuito.

File Storage

Unix and Unix Like

- **Nome**
- Percorso (path)
- Tipo
- Dimensione
- Proprietario (UID, GID)
- Permessi
- Marcature Temporali
 - creazione
 - modifica
- Tutti i nomi dei file sono “Case Sensitive”. Ciò vuol dire che vivek.txt Vivek.txt VIVEK.txt sono tre file differenti.
- Per i nomi di file si possono usare lettere maiuscole, minuscole ed i simboli “.” (dot), e “_” (underscore).
- Possono essere usati anche altri caratteri speciali come “ ” (blank space) ma hanno un uso complesso (devono essere quotati) e se ne sconsiglia l’uso.
- In pratica il nome di un file può contenere qualsiasi carattere escluso “/” (root folder) che è riservato come separatore tra file e folder nel pathname.
- Non può essere usato il carattere null.
- L’uso del “.” non è necessario ma aumenta la leggibilità specialmente se usato per identificare l’estensione.
- Il nome del file è unico all’interno di un folder.
- In un folder non possono coesistere un folder ed un file con lo stesso nome.

File Storage

Unix and Unix Like

- Nome
- **Percorso (path)**
- Tipo
- Dimensione
- Proprietario (UID, GID)
- Permessi
- Marcature Temporali
 - creazione
 - modifica

Il set di nomi richiesto per specificare un particolare file in una gerarchia di folder è detto percorso del file o path.
percorso e nome del file formano il cosiddetto pathname.

Il percorso può essere assoluto o relativo:

- nel path assoluto si specifica tutto il percorso dall'inizio del disco (/root):
`/u/politi/projectb/plans/1dft`
- nel path relativo si può indicare il percorso a partire dal folder in cui ci si trova.
`projectb/plans/1dft`

Un path relativo non può iniziare con /.

Simboli speciali:

- . indica il folder corrente
- .. indica il folder di livello superiore

File Storage

Unix and Unix Like

- Nome
- **Percorso (path)**
- Tipo
- Dimensione
- Proprietario (UID, GID)
- Permessi
- Marcature Temporali
 - creazione
 - modifica

Il set di nomi richiesto per specificare un particolare file in una gerarchia di folder è detto percorso del file o path.
percorso e nome del file formano il cosiddetto pathname.

Il percorso può essere assoluto o relativo:

- nel path assoluto si specifica tutto il percorso dall'inizio del disco (/root):
`/u/politi/projectb/plans/1dft`
- nel path relativo si può indicare il percorso a partire dal folder in cui ci si trova.
`projectb/plans/1dft`

Un path relativo non può iniziare con /.

Simboli speciali:

- . indica il folder corrente
- .. indica il folder di livello superiore

File Storage

Unix and Unix Like

- Nome
- Percorso (path)
- **Tipo**
- Dimensione
- Proprietario (UID, GID)
- Permessi
- Marcature Temporali
 - creazione
 - modifica

Il tipo di file viene identificato dal primo carattere della stringa dei permessi.

```
-rwxrwxrwx 1 romolo romolo      658 apr 30 09:56 manage.py
```

I tipi possono essere:

- file regolare
- d directory
- l symbolic link
- c Character file device
- b block device
- s local socket
- p named pipe

LLn.

like

- **Tipo**
 - Dimensione
 - Proprietario (UID, GID)
 - Permessi
 - Marcature Temporali
 - creazione
 - modifica

Il tipo di file viene identificato dal primo carattere della stringa dei permessi.

```
-rwxrwxrwx 1 romolo romolo      658 apr 30 09:56 manage.py
```

I tipi possono essere:

- | | |
|---|-----------------------|
| - | file regolare |
| d | directory |
| l | symbolic link |
| c | Character file device |
| b | block device |
| s | local socket |
| p | named pipe |

II_b.

```
-rwxrwxrwx- 10 root root 2048 Jan 13 07:11 afile.exe
?UUUGGGOOOS 00 UUUUUU GGGGGG ##### ^-- date stamp and file name are obvious ;-)
^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ ^ \--- File Size
| | | | | | | | | | | | \----- Group Name (for example, Users, Administrators, etc)
| | | | | | | | | | | | \----- Owner Acct
| | | | | | | | | | | | \----- Link count (what constitutes a "link" here varies)
| | | | | | | | | | | | \----- Alternative Access (blank means none defined, anything else varies)
| | | | | | | | | | | | \----- Read, Write and Special access modes for [U]ser, [G]roup, and [O]thers (everyone else)
| | | | | | | | | | | | \----- File type flag
```

like

- **Tipo**
- Dimensione
- Proprietario (UID, GID)
- Permessi

Il tipo di file viene identificato dal primo carattere della stringa dei permessi.

```
-rwxrwxrwx 1 romolo romolo 658 apr 30 09:56 manage.py
```

I tipi possono essere:

- file regolare

	Character	Effect on files	Effect on directories
Read permission (first character)	-	The file cannot be read.	The directory's contents cannot be shown.
	r	The file can be read.	The directory's contents can be shown.
Write permission (second character)	-	The file cannot be modified.	The directory's contents cannot be modified.
	w	The file can be modified.	The directory's contents can be modified (create new files or folders; rename or delete existing files or folders); requires the execute permission to be also set, otherwise this permission has no effect.
Execute permission (third character)	-	The file cannot be executed.	The directory cannot be accessed with <code>cd</code> .
	x	The file can be executed.	The directory can be accessed with <code>cd</code> ; this is the only permission bit that in practice can be considered to be "inherited" from the ancestor directories, in fact if <i>any</i> folder in the path does not have the <code>x</code> bit set, the final file or folder cannot be accessed either, regardless of its permissions; see path_resolution(7) for more information.
	s	The setuid bit when found in the user triad; the setgid bit when found in the group triad; it is not found in the others triad; it also implies that <code>x</code> is set.	
	S	Same as <code>s</code> , but <code>x</code> is not set; rare on regular files, and useless on folders.	
	t	The sticky bit; it can only be found in the others triad; it also implies that <code>x</code> is set.	
	T	Same as <code>t</code> , but <code>x</code> is not set; rare on regular files, and useless on folders.	

File Storage

Unix and Unix Like

- Nome
- Percorso (path)
- Tipo
- **Dimensione**
- Proprietario (UID, GID)
- Permessi
- Marcature Temporali
 - creazione
 - modifica

Il tipo di file viene identificato dal primo carattere della stringa dei permessi.

```
-rwxrwxrwx 1 romolo romolo      658 apr 30 09:56 manage.py
```

I tipi possono essere:

- file regolare
- d directory
- l symbolic link
- c Character file device
- b block device
- s local socket
- p named pipe

Object Storage

Nello storage di oggetti, i dati vengono frammentati in unità discrete chiamate appunto oggetti e conservati in un unico repository (Bucket) invece che come file all'interno di cartelle o come blocchi su server.

I volumi dello storage di oggetti operano come unità modulari: ognuno è un repository indipendente che conserva al suo interno i dati, un identificativo univoco che permette di individuare un oggetto in un sistema distribuito e i metadati che descrivono i dati. I metadati sono importanti e includono dettagli come l'età, privacy/sicurezza e limitazioni all'accesso.

Object Storage

Nello storage di oggetti, i dati vengono frammentati in unità discrete chiamate appunto oggetti e conservati in un unico repository (Bucket) invece che come file all'interno di cartelle o cataloghi.

I volumi dei dati sono immagazzinati in modo indipendente e il repository permette di individuare un oggetto in un sistema distribuito e i metadati che descrivono i dati. I metadati sono importanti e includono dettagli come l'età, privacy/sicurezza e limitazioni all'accesso.

I **metadati** dello storage di oggetti possono essere estremamente dettagliati e capaci di archiviare informazioni sul luogo in cui un video è stato girato, sul tipo di fotocamera che è stato utilizzato e sugli attori che compaiono in ogni fotogramma.

Concetto di Data Preservation

Nella gestione dei dati (Data Management) la **Data Preservation** è l'atto di conservare e mantenere sia la sicurezza che l'integrità dei dati. La conservazione avviene attraverso attività formali disciplinate da politiche, regolamenti e strategie volte a proteggere e prolungare l'esistenza e l'autenticità dei dati e dei relativi metadati.

Concetto di Data Preservation

Nella gestione dei dati (Data Management) la **Data Preservation** è l'atto di conservare e mantenere sia la sicurezza che l'integrità dei dati. La conservazione avviene attraverso attività formali disciplinate da politiche, regolamenti e strategie volte a proteggere e prolungare l'esistenza e l'autenticità dei dati e dei relativi metadati.

- short-term
- medium-term
- long-term

Concetto di Data Preservation

Nella gestione dei dati (Data Management) la **Data Preservation** è l'atto di conservare e mantenere sia la sicurezza che l'integrità dei dati. La conservazione avviene attraverso attività formali disciplinate da politiche, regolamenti e strategie volte a proteggere e prolungare l'esistenza e l'autenticità dei dati e dei relativi metadati.

- short-term
- ~~medium term~~
- long-term

Concetto di Data Preservation

Nella gestione dei dati (Data Management) la **Data Preservation** è l'atto di conservare e mantenere sia la sicurezza che l'integrità dei dati. La conservazione avviene attraverso attività formali disciplinate da politiche, regolamenti e strategie volte a proteggere e prolungare l'esistenza e l'autenticità dei dati e dei relativi metadati.

- short-term
- ~~medium term~~
- long-term

Conservazione a breve termine.

Accesso ai materiali digitali per un periodo di tempo definito durante il quale è previsto l'uso ma che non si estende oltre il prevedibile futuro e/o fino a quando non diventa inaccessibile a causa dei cambiamenti tecnologici.

Concetto di Data Preservation

Nella gestione dei dati (Data Management) la **Data Preservation** è l'atto di conservare e mantenere sia la sicurezza che l'integrità dei dati. La conservazione avviene attraverso attività formali disciplinate da politiche, regolamenti e strategie volte a proteggere e prolungare l'esistenza e l'autenticità dei dati e dei relativi metadati.

- short-term
- ~~medium term~~
- long-term

Conservazione a lungo termine

Accesso continuo ai materiali digitali, o almeno alle informazioni in essi contenute, a tempo indeterminato.

Controllo di versione



Git

Git è un software per il controllo di versione distribuito utilizzabile da interfaccia a riga di comando, creato da Linus Torvalds nel 2005.

Un sistema di controllo di versione distribuito o decentralizzato (o **DVCS** da Distributed Version Control System) è una tipologia di controllo di versione che permette di tenere traccia delle modifiche e delle versioni apportate al codice sorgente del software, senza la necessità di dover utilizzare un server centrale, come nei casi classici.

Con questo sistema gli sviluppatori possono collaborare individualmente e parallelamente non connessi su di un proprio ramo (branch) di sviluppo, registrare le proprie modifiche (commit) ed in seguito condividerle con altri o unirle (merge) a quelle di altri, il tutto senza bisogno del supporto di un server centralizzato. Questo sistema permette diverse modalità di collaborazione, proprio perché il server è soltanto un mero strumento d'appoggio.



Glossario

repository: È una “cartella” che contiene tutti i file necessari per il tuo progetto, inclusi i file che tengono traccia di tutte le versioni del progetto.

clone: È la versione locale del repository

remote: È la versione remota del repository che può essere modificata da chiunque abbia accesso al repository.

branch: “rami” vengono utilizzati in Git per l’implementazione di funzionalità tra loro isolate, cioè sviluppate in modo indipendente l’una dall’altra ma a partire dalla medesima radice.

fork: copia del repository appartenente ad un altro utente

commit: snapshot del repository locale compresso con SHA pronto per essere trasferito, dal clone al remote o viceversa.

tag: è un marcatore per evidenziare dei particolari commit



Primi Passi

Git può essere scaricato all'indirizzo <https://git-scm.com/downloads> (tutte le distribuzioni di linux hanno git tra i pacchetti disponibili).

Una volta installato il software, per “copiare” un repository in locale basta utilizzare il comando clone. Ad esempio per il repository del corso:

```
git clone git@github.com:RomoloPoliti-INAF/PhDCourse2023b.git
```



Primi Passi

Git può essere scaricato all'indirizzo <https://git-scm.com/downloads> (tutte le distribuzioni di linux hanno git tra i pacchetti disponibili).

Una volta installato il software, per “copiare” un repository in locale basta utilizzare il comando clone. Ad esempio per il repository del corso:

```
git clone git@github.com:RomoloPoliti-INAF/PhDCourse2023b.git
```

Gli utenti windows devono installare l'applicativo git scaricandolo dal sito [GitHub](#).

Dopo l'installazione sarà necessario riavviare la macchina.



Comandi Git fondamentali

clone Crea una copia locale di un repository remoto

pull Aggiorna la copia locale del repository

add Aggiungi uno o più file alla lista dei contenuti del repository locale

commit Registra i cambiamenti al repository

push aggiorna il repository remoto

Lezione del 19 ottobre 2023

Panoramica del Corso

Cloud

~~Struttura del Cloud~~
~~Dati nel Cloud~~
~~Calcolo nel Cloud~~

Dati

~~Dati e metadati~~
~~Archiviazione~~
DB Relazionali e non

Calcolo

Recupero
Manipolazione
Visualizzazione

Ambiente

Virtualizzazione e container
Microservices
DevOps

Programmazione

Fondamenti di programmazione
Python
Versioning e Documentazione

Database Relazionale

Introduzione

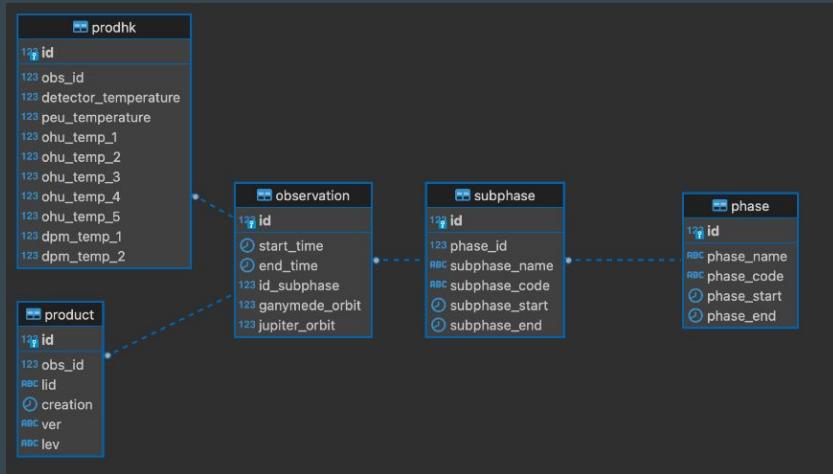
Il termine **relational database management system** (**RDBMS**, sistema per la gestione di basi di dati relazionali) indica un database management system basato sul modello relazionale, introdotto da Edgar F. Codd.

Oltre a questi, anche se meno diffusi a livello commerciale, altri sistemi di gestione di basi di dati che implementano modelli dei dati alternativi a quello relazionale: **gerarchico, reticolare e a oggetti**.

Tra i vari database di relazionali e DB ad oggetti PostgreSQL è quello con la più elevata diffusione



Diagramma Entità-Relazione



IDEF1X model

Diagramma Entità-Relazione - Glossario

Schema	un gruppo di entità con le loro relazioni
Entità	rappresentano classi di oggetti (fatti, cose, persone, ...) che hanno proprietà comuni ed esistenza autonoma ai fini dell'applicazione di interesse. Un'occorrenza di un'entità è un oggetto o istanza della classe che l'entità rappresenta. Non si parla qui del valore che identifica l'oggetto ma dell'oggetto stesso. Un'interessante conseguenza di questo fatto è che un'occorrenza di entità ha un'esistenza indipendente dalle proprietà ad essa associate. In uno schema ogni entità ha un nome che la identifica univocamente e viene rappresentata graficamente tramite un rettangolo con il nome dell'entità al suo interno.
Relazioni	rappresentano un legame tra due o più entità. Il numero di entità legate è indicato dal grado dell'associazione: un buono schema E-R è caratterizzato da una prevalenza di associazioni con grado due.
Tupla	una serie di attributi che descrivono le entità. Tutti gli oggetti della stessa classe entità hanno gli stessi attributi: questo è ciò che si intende quando si parla di oggetti simili.
Attributo	Caratteristica dell'entità.

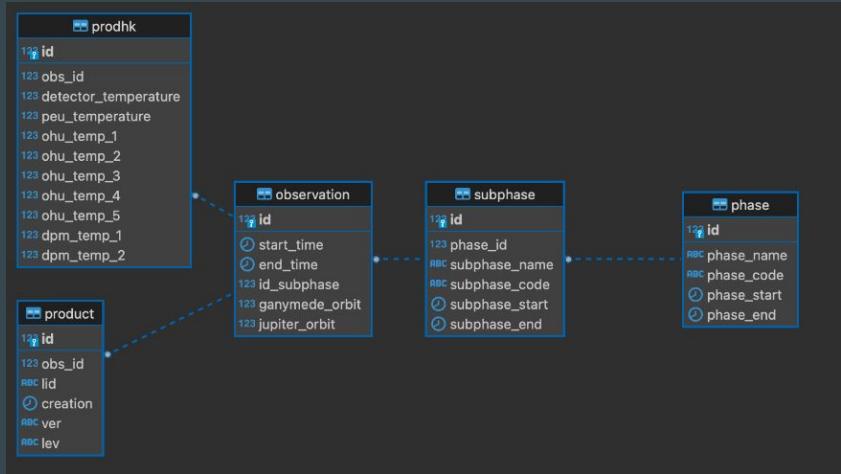
Diagramma Entità-Relazione - Glossario

La scelta degli attributi riflette il livello di dettaglio con il quale vogliamo rappresentare le informazioni delle singole entità e relazioni.

Per ciascuna classe entità o associazione si definisce una chiave.

La chiave è un insieme minimale di attributi che identifica univocamente un'istanza di entità.

Diagramma Entità-Relazione



UML (Unified Modeling Language, "linguaggio di modellizzazione unificato")

IDEF1X model

Linguaggio SQL

Per fare qualche esempio di linguaggio SQL useremo SQLite.

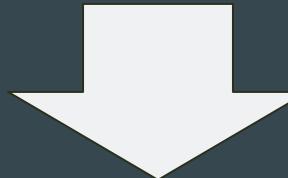
Un frontend potete trovarlo qui <https://sqlitebrowser.org>

Linguaggio SQL

Per fare qualche esempio di linguaggio SQL useremo SQLite.

Un frontend potete trovarlo qui <https://sqlitebrowser.org>

Schema	Entità	Tupla	Attributo	Relazione
--------	--------	-------	-----------	-----------



Database	Tabella	Record	Campo	Chiave esterna
----------	---------	--------	-------	----------------

SQL - Comandi fondamentali

CREATE Crea un database o una tabella

INSERT Crea uno o più nuovi record nella tabella

DROP Cancella un database o una tabella

DELETE Cancella uno o più record

ALTER Modifica un database o una tabella

UPDATE Modifica un record

SELECT Seleziona una serie di record

SQL - Comandi fondamentali

CREATE Crea un database o una tabella

INSERT Crea uno o più nuovi record nella tabella

DROP Cancella un database o una tabella

DELETE Cancella uno o più record

ALTER Modifica un database o una tabella

UPDATE Modifica un record

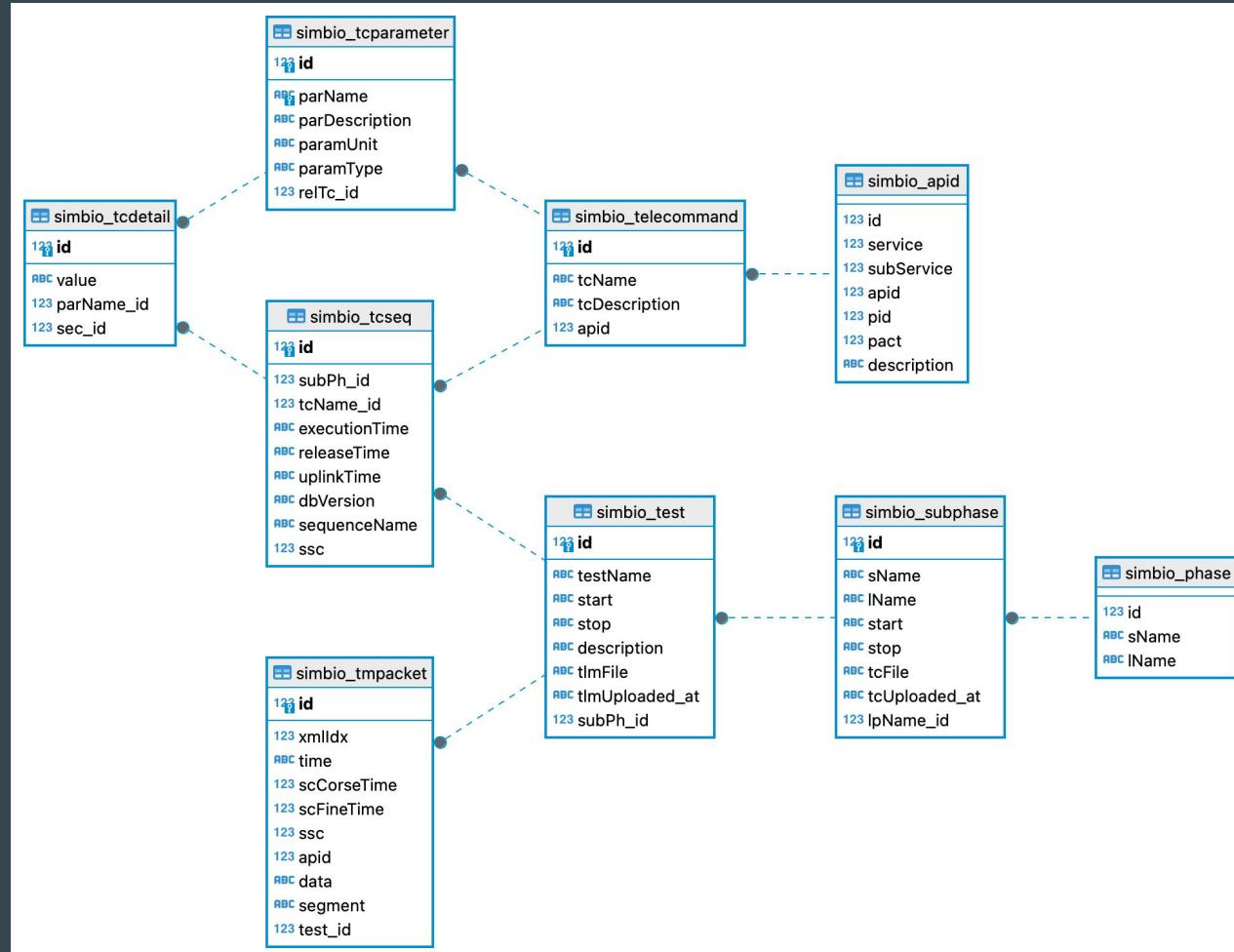
SELECT Seleziona una serie di record

Query

Esempi

Gli esempi utilizzano il database *example.sqlite* che trovate nella cartella *Esempi/02 - sqlLite* del repository. Nella stessa cartella il file *Script.sql* con tutti gli esempi.

Al lato trovate il diagramma ER del database.



Select

SQL> **SELECT tabella.campo FROM tabella;**

Select

```
SQL> SELECT tabella.campo FROM tabella;
```

Esempio 1: Voglio l'elenco di tutti i test (tabella *simbio_test*) presenti nel mio DB.

```
SQL> SELECT * FROM simbio_test;
```

Select

```
SQL> SELECT tabella.campo FROM tabella;
```

Esempio 1: Voglio l'elenco di tutti i test (tabella *simbio_test*) presenti nel mio DB.

```
SQL> SELECT * FROM simbio_test;
```

Esempio 2: dalla lista precedente voglio solo il nome e tempo di inizio e fine

```
SQL> SELECT st.testName, st.start, st.stop FROM simbio_test st,
```

Select

```
SQL> SELECT tabella.campo FROM tabella;
```

Esempio 1: Voglio l'elenco di tutti i test (tabella *simbio_test*) presenti nel mio DB.

```
SQL> SELECT * FROM simbio_test;
```

Esempio 2: dalla lista precedente voglio solo il nome e tempo di inizio e fine

```
SQL> SELECT st.testName, st.start, st.stop FROM simbio_test st,
```



Alias della tabella. Equivalente a:
simbio_test AS st

Select

```
SQL> SELECT tabella.campo FROM tabella;
```

Esempio 1: Voglio l'elenco di tutti i test (tabella *simbio_test*) presenti nel mio DB.

```
SQL> SELECT * FROM simbio_test;
```

Esempio 2: dalla lista precedente voglio solo il nome e tempo di inizio e fine

```
SQL> SELECT test_name, start, stop FROM simbio_test;
```

Esempio 3: le stesse info dell'esempio 2 ma solo quelli eseguiti il 11/12/2018

```
SQL> SELECT test_name, start, stop FROM simbio_test WHERE start > "2018-12-11  
00:00:00" AND stop < "2018-12-11 23:59:59";
```

Select

Esempio 4: Voglio tutti i campi delle sottofasi della fase “CRUISE” ordinati cronologicamente (sapendo che fase e sottofase sono collegati tramite un id):

```
SQL> SELECT sp.* FROM simbio_subphase sp, simbio_phase p WHERE p.id = sp.lpName_id  
AND p.sName = "CRUISE" ORDER BY sp.start;
```

Esercizio

Selezionare il primo telecomando di scienza di VIHI eseguito l'11/12/2018 e fornire il tempo a cui è stato eseguito e tempo di integrazione.

Esercizio - Soluzione

Selezionare il primo telecomando di scienza di VIHI eseguito l'11/12/2018 e fornire il tempo a cui è stato eseguito e tempo di integrazione.

```
SQL>SELECT id FROM simbio_telecommand tc WHERE tc.tcDescription LIKE "%VIHI  
science%" LIMIT 1;
```

[OUT] 306

```
SQL> SELECT executionTime, id FROM simbio_tcseq WHERE  
simbio_tcseq.tcName_id=306 AND executionTime > "2018-12-11" AND  
executionTime < "2018-12-12";
```

[OUT] 2018-12-11 15:54:37.657847+01; 9784

Esercizio - Soluzione

Selezionare il primo telecomando di scienza di VIHI eseguito l'11/12/2018 e fornire il tempo a cui è stato eseguito e tempo di integrazione.

[OUT] 2018-12-11 15:54:37.657847+01; 9784

SQL> SELECT *id* FROM *simbio_tcparameter* WHERE *parDescription* LIKE "%VIHI
integration%";

[OUT] 578

SQL> SELECT *value* FROM *simbio_tcdetail* WHERE *sec_id*=9784 AND *parName_id*=578;

[OUT] 3

Esercizio - Soluzione più elegante

Selezionare il primo telecomando di scienza di VIHI eseguito l'11/12/2018 e fornire il tempo a cui è stato eseguito e tempo di integrazione.

```
SQL>SELECT tseq.executionTime, simbio_tcdetail.value FROM simbio_tcseq AS tseq JOIN simbio_tcdetail ON tseq.id = simbio_tcdetail.sec_id WHERE tseq.tcName_id = (SELECT stc.id FROM simbio_telecommand stc WHERE stc.tcDescription LIKE "%VIHI%" AND stc.tcDescription LIKE "%science%") AND tseq.executionTime > "2018-12-11" AND tseq.executionTime < "2018-12-12" AND simbio_tcdetail.parName_id = (SELECT tcp.id FROM simbio_tcpparameter AS tcp WHERE tcp.parDescription LIKE "%VIHI%" AND tcp.parDescription LIKE "%integration%") ORDER BY tseq.executionTime LIMIT 1
```

eXtensible Markup Language - XML

Cosa è l'XML

XML (sigla di **eXtensible Markup Language**, lett. "linguaggio di marcatura estendibile") è un metalinguaggio per la definizione di linguaggi di markup, ovvero un linguaggio basato su un meccanismo sintattico che consente di definire e controllare il significato degli elementi contenuti in un documento o in un testo.

Cosa è l'XML

XML (sigla di **eXtensible Markup Language**, lett. "linguaggio di marcatura estendibile") è un metalinguaggio per la definizione di linguaggi di markup, ovvero un linguaggio basato su un meccanismo sintattico che consente di definire e controllare il significato degli elementi contenuti in un documento o in un testo.

Nella logica e nella teoria dei linguaggi formali per **metalinguaggio** si intende un linguaggio formalmente definito che ha come scopo la definizione di altri linguaggi artificiali, definiti linguaggi obiettivo o linguaggi oggetto (nell'ambito di SGML e di XML si usa anche il termine applicazioni).

Tale definizione tende ad essere formalmente rigorosa e completa, tanto da potersi utilizzare per la costruzione o la validazione di strumenti informatici di sostegno per i linguaggi obiettivo.

Cosa è l'XML

XML (sigla di **eXtensible Markup Language**, lett. "linguaggio di marcatura estendibile") è un metalinguaggio per la definizione di linguaggi di markup, ovvero un linguaggio basato su un meccanismo sintattico che consente di definire e controllare il significato degli elementi contenuti in un documento o in un testo.

```
<?xml version="1.0" encoding="UTF-8"?>
<utenti>
    <utente anni="20">
        <nome>Ema</nome>
        <cognome>Princi</cognome>
        <indirizzo>Torino</indirizzo>
    </utente>
    <utente anni="54">
        <nome>Max</nome>
        <cognome>Rossi</cognome>
        <indirizzo>Roma</indirizzo>
    </utente>
</utenti>
```

Cosa è l'XML

XML (sigla di **eXtensible Markup Language**, lett. "linguaggio di marcatura estendibile") è un metalinguaggio per la definizione di linguaggi di markup, ovvero un linguaggio basato su un meccanismo sintattico che consente di definire e controllare il significato degli elementi contenuti in un documento o in un testo.

```
<?xml version="1.0" encoding="UTF-8"?> → Preambolo
<utenti>
    <utente anni="20">
        <nome>Ema</nome>
        <cognome>Princi</cognome>
        <indirizzo>Torino</indirizzo>
    </utente>
    <utente anni="54">
        <nome>Max</nome>
        <cognome>Rossi</cognome>
        <indirizzo>Roma</indirizzo>
    </utente>
</utenti>
```

Cosa è l'XML

XML (sigla di **eXtensible Markup Language**, lett. "linguaggio di marcatura estendibile") è un metalinguaggio per la definizione di linguaggi di markup, ovvero un linguaggio basato su un meccanismo sintattico che consente di definire e controllare il significato degli elementi contenuti in un documento o in un testo.

```
<?xml version="1.0" encoding="UTF-8"?> → Preambolo
<utenti> → Tag
  <utente anni="20">
    <nome>Ema</nome>
    <cognome>Princi</cognome>
    <indirizzo>Torino</indirizzo>
  </utente>
  <utente anni="54">
    <nome>Max</nome>
    <cognome>Rossi</cognome>
    <indirizzo>Roma</indirizzo>
  </utente>
</utenti>
```

Cosa è l'XML

XML (sigla di **eXtensible Markup Language**, lett. "linguaggio di marcatura estendibile") è un metalinguaggio per la definizione di linguaggi di markup, ovvero un linguaggio basato su un meccanismo sintattico che consente di definire e controllare il significato degli elementi contenuti in un documento o in un testo.

```
<?xml version="1.0" encoding="UTF-8"?> → Preambolo
<utenti> → Tag
  <utente anni="20">
    <nome>Ema</nome>
    <cognome>Princi</cognome>
    <indirizzo>Torino</indirizzo> → Elemento
  </utente>
  <utente anni="54">
    <nome>Max</nome>
    <cognome>Rossi</cognome>
    <indirizzo>Roma</indirizzo>
  </utente>
</utenti>
```

Fondamenti di programmazione Python

Fondamenti di Python

Linguaggio: Python 3.12

Ambiente di Sviluppo: Microsoft Visual Studio Code

Argomenti

- Package e Moduli
- Variabili
- Classi e Oggetti
- Versionamento del software
- Dichiarazione di Condizione
- Operatori
- Cicli
- Funzioni
- Decoratori
- Namespace
- Lambda
- I/O
- Eccezioni
- PyPI

Packages:

- argparse
- click
- rich
- rich-click
- logging
- pandas
- numpy
- scipy
- matplotlib
- multiprocessing
- sqlite
- ElementTree

Elementi di base

Carattere per commento di linea: #

Commento multilinea :

'''

'''

'''

Indentatura

Elementi di base

Carattere per commento di linea: #

Commento multilinea :

'''

'''

Indentatura

PEP

PEP sta per **Python Enhancement Proposal**. Una PEP è un documento di progettazione che fornisce informazioni alla comunità Python o descrive una nuova funzionalità per Python o i suoi processi o ambiente.

Una PEP dovrebbe fornire una specifica tecnica concisa della caratteristica e una motivazione per la caratteristica.

- **Standards Track PEP** descrive una nuova funzionalità o implementazione di Python;
- **Informational PEP** descrive il design di una nuova funzionalità, detta le guide generali o fornisce informazioni alla comunità Python;
- **Process PEP** descrive un processo di Python o propone un cambiamento ad un processo.

(PEP 1)

La più importante di tutte è la PEP 8, **Style Guide for Python Code**, che standardizza come deve essere scritto il codice in Python.

Ogni volta che una informazione deriva da una PEP indicheremo (PEP#) dove # è il numero della PEP.

Metodi Dunder o Metodi Magici

I metodi o variabili dunder (contrazione di *double underscore*) sono dei metodi speciali utilizzati per l'overload di funzioni primitive (*build-in*).

I più comuni a livello di modulo sono:

__version__ in cui si inserisce il numero di versione

__author__ in cui si inserisce il nome dell'autore.

Esamineremo i singoli dunder quando li incontreremo nella scrittura del codice.

Shell o Script?

Sono due i metodi principali per eseguire dei comandi Python:

Shell o Script?

Sono due i metodi principali per eseguire dei comandi Python:

utilizzando la shell python (python3)

Shell o Script?

Sono due i metodi principali per eseguire dei comandi Python:

utilizzando la shell python (python3)

```
> python3
Python 3.12.0 (v3.12.0:0fb18b02c8, Oct  2 2023, 09:45:56) [Clang 13.0.0 (clang-1300.0.29.30)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello World!")
Hello World!
>>> exit()
```



✓ took 19s ✘ at 10:21:22 ⏺

Shell o Script?

Sono due i metodi principali per eseguire dei comandi Python:

utilizzando la shell python (python3)

utilizzando uno script python

Shell o Script?

Sono due i metodi principali per eseguire dei comandi Python:

utilizzando la shell python (python3)

utilizzando uno script python

```
> echo "print('Hello World!')">>> test.py
> python3 test.py
Hello World!
```



✓ at 10:23:43 ⌂

Shell o Script?

Sono due i metodi principali per eseguire dei comandi Python:

utilizzando la shell python (python3)

utilizzando uno script python

Si può rendere eseguibile uno script.

```
> echo "#! /usr/bin/env python3\nprint('Hello World!')">> test.py  
> chmod u+x test.py  
> ./test.py  
Hello World!
```



✓ at 10:25:32 ⌂

Shell o Script?

Sono due i metodi principali per eseguire dei comandi Python:

utilizzando la shell python (python3)

utilizzando uno script python

Si può rendere eseguibile uno script.

In questo caso è necessario specificare l'interprete dei comandi

```
> echo "#! /usr/bin/env python3\nprint('Hello World!')">> test.py
> chmod u+x test.py
> ./test.py
Hello World!
```

at 10:25:32

Shell o Script?

Sono due i metodi principali per eseguire dei comandi Python:

utilizzando la shell python (python3)

utilizzando uno script python

Si può rendere eseguibile uno script.

In questo caso è necessario specificare l'interprete dei comandi

```
> echo "#! /usr/bin/env python3\nprint('Hello World!')">> test.py
> chmod u+x test.py
> ./test.py
Hello World!
```

at 10:25:32

Users > romolo.politi > test.py

```
1  #! /usr/bin/env python3
2  print('Hello World')
3
```

Shell o Script?

Sono due i metodi principali per eseguire dei comandi Python:

utilizzando la shell python (python3)

utilizzando uno script python

Si può rendere eseguibile uno script.

In questo caso è necessario specificare l'interprete dei comandi

Noi utilizzeremo principalmente degli script

Glossario Python

Il modulo è un file contenente definizioni e istruzioni Python.

Un modulo può definire funzioni, classi e variabili.

Un modulo può anche includere codice eseguibile.

Il raggruppamento del codice correlato in un modulo semplifica la comprensione e l'utilizzo del codice.

Rende anche il codice organizzato logicamente.

Le variabili sono contenitori per dati.

Python non ha comandi per inizializzare variabili. Vengono inizializzate alla prima assegnazione.

Sono Case Sensitive.

Sono caratterizzate dal tipo.

Tipi di variabili

Per conoscere il tipo di una variabile si utilizza il comando
`type(var)`

Example	Data Type
<code>x = "Hello World"</code>	<code>str</code>
<code>x = 20</code>	<code>int</code>
<code>x = 20.5</code>	<code>float</code>
<code>x = 1j</code>	<code>complex</code>
<code>x = ["apple", "banana", "cherry"]</code>	<code>list</code>
<code>x = ("apple", "banana", "cherry")</code>	<code>tuple</code>
<code>x = range(6)</code>	<code>range</code>
<code>x = {"name": "John", "age": 36}</code>	<code>dict</code>
<code>x = {"apple", "banana", "cherry"}</code>	<code>set</code>
<code>x = frozenset({"apple", "banana", "cherry"})</code>	<code>frozenset</code>
<code>x = True</code>	<code>bool</code>
<code>x = b"Hello"</code>	<code>bytes</code>
<code>x = bytearray(5)</code>	<code>bytearray</code>
<code>x = memoryview(bytes(5))</code>	<code>memoryview</code>

```
Text Type: str
Numeric Types: int, float, complex
Sequence Types: list, tuple, range
Mapping Type: dict
Set Types: set, frozenset
Boolean Type: bool
Binary Types: bytes, bytearray, memoryview
```

Tipi di variabili

Per conoscere il tipo di una variabile si utilizza il comando
type(var)

Example	Data Type
x = "Hello World"	str
x = 20	int
x = 20.5	float
x = 1j	complex
x = ["apple", "banana", "cherry"]	list
x = ("apple", "banana", "cherry")	tuple
x = range(6)	range
x = {"name": "John", "age": 36}	dict
x = {"apple", "banana", "cherry"}	set
x = frozenset({"apple", "banana", "cherry"})	frozenset
x = True	bool
x = b"Hello"	bytes
x = bytearray(5)	bytearray
x = memoryview(bytes(5))	memoryview

Text Type:	str
Numeric Types:	int, float, complex
Sequence Types:	list, tuple, range
Mapping Type:	dict
Set Types:	set, frozenset
Boolean Type:	bool
Binary Types:	bytes, bytearray, memoryview

In questo caso abbiamo usato un tipo speciale di stringa detta binary string.
Tipi di stringhe “speciali”

b binary string
f formatted string
r raw string

Definizione di Classe ed Oggetto

Vocabolario per l'**Object-Oriented Programming** (OOP)

Classe: un progetto che consiste nella definizione di metodi ed attributi

Oggetto: un'istanza di una classe. si può pensare ad un oggetto come a qualcosa del mondo reale, come un penna gialla, un cagnolino etc. In ogni caso un oggetto può essere molto più astratto

Attributo: un descrittore o una caratteristica. Ad esempio lunghezza,colore etc.

Metodo: un'azione che la classe o l'oggetto possono ricevere.

Classi ed Oggetti

Vediamo un esempio pratico.

Versioning

Versioning:

MAJOR.MINOR.PATCH

versione **MAJOR** quando modificate l'API in modo non retrocompatibile,

versione **MINOR** quando aggiungete funzionalità in modo retrocompatibile

versione **PATCH** quando correggete bug in modo retrocompatibile.

[Versionamento Semantico 2.0.0](#)

Versioning

Tipologia:

devel In via di sviluppo

alpha nella prima fase di test

beta ultima fase di test

Release Candidate pronta al rilascio

final: versione di rilascio (secondo il versionamento semantico questa tipologia non viene indicata)

A questi viene aggiunto un numero che indica la build, cioè l'avanzamento del tipo.

Esempio

Nella cartella *Esempi/03 - Class_and_Object* trovate un jupyter notebook con gli esempi ed un modulo chiamato *version.py*

Modulo version (file `version.py`)

All'inizio del modulo trovate l'inizializzazione di un dizionario chiamato `types`.

Essendo stato definito all'esterno di una funzione o classe il dizionario ha un valore **globale**, cioè è visto da tutti gli oggetti presenti nel modulo.

```
1  types = {  
2      'd': 'dev',  
3      'a': 'alpha',  
4      'b': 'beta',  
5      'rc': 'candidate',  
6      'f': 'final',  
7  }
```

Classe Version

Subito dopo la definizione di classe troviamo un commento multilinea.

- Tutti i commenti inseriti subito dopo la definizione di una classe o di una funzione vengono archiviati nella variabile dunder **__doc__**. E' buona norma inserire sempre un commento dopo la dichiarazione di un oggetto.

```
11     """Version Class
12         | respecting the semantic versioning 2.0.0
13     """
```

Classe Version

Subito dopo la definizione di classe troviamo un commento multilinea.

- Tutti i commenti inseriti subito dopo la definizione di una classe o di una funzione vengono archiviati nella variabile dunder `__doc__`. E' buona norma inserire sempre un commento dopo la dichiarazione di un oggetto.
- In una classe i metodi predefiniti sono dei dunder. Se si ridefiniscono i metodi originali vengono sovrascritti. Il dunder `__init__` viene richiamato ogni volta che un oggetto viene creato.
- Tutti i metodi di una classe hanno come primo argomento la classe stessa. Solitamente viene indicata con la variabile `self`.

```
15 | def __init__(self, version: tuple):  
16 |     self.version = version
```

Classe Version

Subito dopo la definizione di classe troviamo un commento multilinea.

- Tutti i commenti inseriti subito dopo la definizione di una classe o di una funzione vengono archiviati nella variabile dunder `__doc__`. E' buona norma inserire sempre un commento dopo la dichiarazione di un oggetto.
- In una classe i metodi predefiniti sono dei dunder. Se si ridefiniscono i metodi originali vengono sovrascritti. Il dunder `__init__` viene richiamato ogni volta che un oggetto viene creato.
- Tutti i metodi di una classe hanno come primo argomento la classe stessa.

Solitamente nella definizione di una classe si specifica il tipo delle variabili. Nella definizione di una funzione è buona norma indicare il tipo della variabile in ingresso. Questo permette una migliore leggibilità del codice e di conseguenza un debug più semplice.

```
15 |     def __init__(self, version: tuple):  
16 |         self.version = version
```

Classe Version

prima della definizione di una funzione possiamo trovare delle stringhe che iniziano con il carattere **@**.

Queste stringhe vengono chiamati **decoratori**. Sono delle funzioni che permettono di manipolare la funzione successiva, applicando dei blocchi di codice standard prima e/o dopo l'esecuzione. Vedremo in seguito come costruire un decoratore.

Il decoratore **@property** in Python è un modo conciso e pulito per definire **getter**, **setter** e **deleter** per gli attributi di una classe.

Classe Version

la funzione con il decoratore **@property** è il **getter**, cioè la funzione che viene chiamata ogni volta che si accede all'attributo (o proprietà) della classe.

Nel nostro caso ogni volta che richiamiamo l'attributo version ci viene restituita una stringa composta dai numeri di versione:

```
18     @property
19     def version(self):
20         if self._type is None:
21             adv = ""
22         else:
23             adv = f"-{self._type}"
24             if self._build is not None :
25                 adv += f".{self._build}"
26         return f"{self._major}.{self._minor}.{self._patch}{adv}"
```

Classe Version - setter

il **setter** dell'attributo version (**@version.setter**) analizza la stringa con cui viene inizializzata la classe, la divide nei 5 campi principali e verifica che il quarto sia una lettera e sia tra quelle ammesse, mentre le altre siano degli interi. Questi valori validati, vengono inseriti all'interno di attributi privati (iniziano con il carattere `_`) tramite la funzione riflessiva **setattr**.

Programmazione riflessiva

La programmazione riflessiva è un paradigma di programmazione che consente al codice di interagire con se stesso a livello di metadati. Questo significa che il codice può accedere e manipolare informazioni sul proprio tipo, struttura e comportamento.

In Python, la programmazione riflessiva può essere eseguita utilizzando una serie di funzioni e metodi built-in. Ad esempio, la funzione `type()` può essere utilizzata per ottenere il tipo di un oggetto, la funzione `dir()` può essere utilizzata per ottenere un elenco degli attributi e dei metodi di un oggetto, la funzione `getattr()` può essere utilizzata per accedere a un attributo di un oggetto, e la funzione `setattr()` può essere utilizzata per creare un attributo di un oggetto..

La programmazione riflessiva può essere utilizzata per una serie di scopi, tra cui:

- **Manipolazione di metadati:** La programmazione riflessiva può essere utilizzata per accedere e manipolare informazioni sui tipi, le strutture e il comportamento del codice.
- **Test di codice:** La programmazione riflessiva può essere utilizzata per scrivere test di unità che verificano il comportamento del codice a livello di metadati.
- **Generazione di codice:** La programmazione riflessiva può essere utilizzata per generare codice nuovo o modificare il codice esistente.

La programmazione riflessiva può essere uno strumento potente per i programmatore Python, ma è importante utilizzarla con cautela. La programmazione riflessiva può rendere il codice più complesso e difficile da mantenere.

Classe Version - setter

Nel caso in cui la validazione non venga superata viene generata un'eccezione con il comando **raise**.

Eccezioni

In Python, le eccezioni sono eventi che indicano che un errore è stato riscontrato durante l'esecuzione di un programma. Le eccezioni possono essere generate da una varietà di cause, tra cui:

- **Operazioni non valide:** Ad esempio, dividere un numero per zero o accedere a un attributo di un oggetto che non esiste.
- **Errori di runtime:** Ad esempio, un errore di memoria o un errore di I/O.
- **Errori di sintassi:** Ad esempio, un errore di punteggiatura o un errore di tipo.

Quando si verifica un'eccezione, il normale flusso di esecuzione del programma viene interrotto. Il programma passa il controllo a un gestore di eccezioni, che è un blocco di codice che può gestire l'eccezione.

In Python, le eccezioni sono gestite utilizzando la sintassi `try-except`. La sintassi `try-except` consente di eseguire un blocco di codice e gestire eventuali eccezioni che vengono generate.

Esistono diversi tipi di eccezioni in Python. Ogni tipo di eccezione è rappresentato da una classe di eccezioni. La classe di eccezioni `Exception` è la classe base di tutte le eccezioni.

Ecco alcuni esempi di tipi di eccezioni in Python:

- **ArithmeticError:** Eccezione di errore aritmetico, ad esempio divisione per zero.
- **AssertionError:** Eccezione sollevata quando un'asserzione fallisce.
- **AttributeError:** Eccezione sollevata quando un tentativo di accesso a un attributo di un oggetto fallisce.
- **EOFError:** Eccezione sollevata quando si raggiunge la fine di un file.
- **ImportError:** Eccezione sollevata quando non è possibile importare un modulo o un pacchetto.
- **KeyError:** Eccezione sollevata quando si tenta di accedere a una chiave che non esiste in un dizionario.
- **LookupError:** Eccezione sollevata quando si tenta di accedere a un elemento di una sequenza che non esiste.
- **NameError:** Eccezione sollevata quando si tenta di utilizzare un nome che non è stato definito.
- **TypeError:** Eccezione sollevata quando si utilizza un tipo di dati non valido.
- **ValueError:** Eccezione sollevata quando si utilizza un valore non valido.

La gestione delle eccezioni è un aspetto importante della programmazione Python. Consente di rendere i programmi più robusti e di gestire gli errori in modo efficace.

Classe Version

il metodo dunder `__str__` è un metodo che viene invocato ogni volta che cerca di stampare un oggetto.

Il default è <version.Version object at 0x10d0739b0>

<modulo.Classe indirizzo di memoria dell'oggetto>

Nel nostro caso viene restituita la stringa : Version 1.2.3

```
62     def __str__(self) -> str:
63         if self._type == "final":
64             return f"Version {self._major}.{self._minor}.{self._patch}"
65         else:
66             return f"Version {self.version}"
67
```

Lezione del 24 ottobre 2023

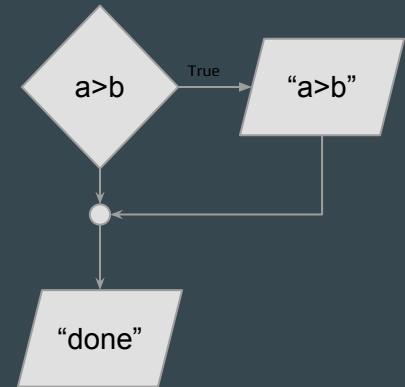
Dichiaratori di Condizione

(Conditional Statements)

Il più semplice è **if**.

Ci permette di escludere una parte di codice se una condizione logica non è verificata.

```
if a > b:  
    print("a>b")  
    print("done")
```



Dichiaratori di Condizione

(Conditional Statements)

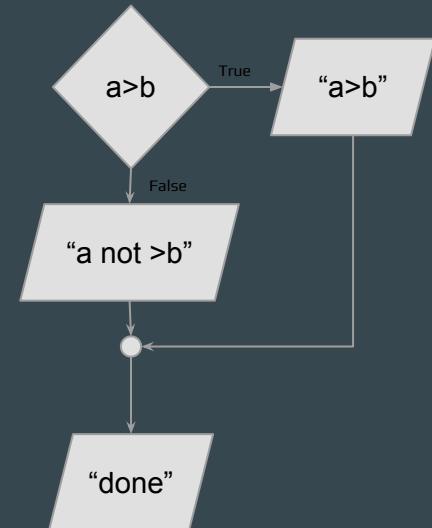
Il più semplice è **if**.

Ci permette di escludere una parte di codice se una condizione logica non è verificata.

if...else

Ci permette di scegliere il blocco di codice da eseguire a seconda di una condizione logica

```
if a > b:  
    print("a>b")  
else:  
    print("a not > b")  
print("done")
```



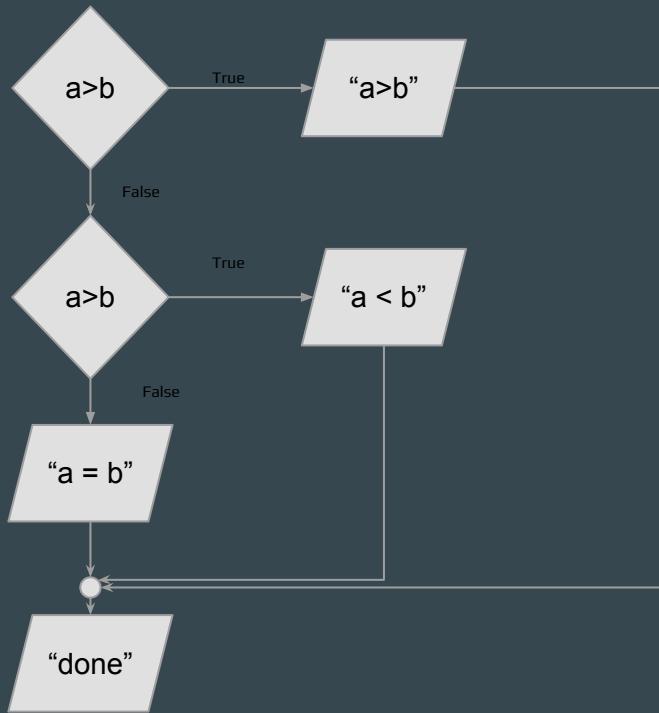
Dichiaratori di Condizione

Per eseguire dei confronti multipli si utilizza la dichiarazione `if...elif...else`

```
if a > b:  
    print("a>b")  
elif a < b:  
    print("a<b")  
else:  
    print("a=b")  
print("done")
```

Dichiaratori di Condizione

Per eseguire dei confronti multipli si utilizza la dichiarazione `if...elif...else`



```
if a > b:  
    print("a>b")  
elif a < b:  
    print("a<b")  
else:  
    print("a=b")  
print("done")
```

Dichiaratori di Condizione

Con Python 3.10 viene introdotto lo statement `match`:

Dichiaratori di Condizione

Con Python 3.10 viene introdotto lo statement `match`:

```
match a:  
    case 1:  
        print("1")  
    case 2:  
        print("2")  
    else:  
        print("a not 1 or 2")
```

Operatori

Aritmetici

Operatori

Aritmetici

- + addizione
- sottrazione
- * moltiplicazione
- / divisione
- ** esponente
- % modulo
- // divisione intera (floor division)

Operatori

Aritmetici

- + addizione
- sottrazione
- * moltiplicazione
- / divisione
- ** esponente
- % modulo
- // divisione intera (floor division)

Modulo:

è il resto della divisione

$$5\%2 = 1$$

Operatori

Aritmetici

- + addizione
- sottrazione
- * moltiplicazione
- / divisione
- ** esponente
- % modulo
- // divisione intera (floor division)

Modulo:

è il resto della divisione

$$5 \% 2 = 1$$

Divisione intera:

è la parte intera del risultato della divisione

$$5 // 2 = 2$$

Operatori

Aritmetici

Confronto

Operatori

Aritmetici

Confronto

```
== uguale  
!= diverso  
> maggiore  
< minore  
>= maggiore uguale  
<= minore uguale
```

Operatori

Aritmetici

Confronto

Logici

Operatori

Aritmetici

Confronto

Logici

and
or
not

Operatori

Aritmetici

Confronto

Logici

and
or
not

True **and** True = True
True **and** False = False
False **and** False = False

Operatori

Aritmetici

Confronto

Logici

and
or
not

True **or** True = True
True **or** False = True
False **or** False = False

Operatori

Aritmetici

Confronto

Logici

and
or
not

not True = False
not False = True

Operatori

Aritmetici

Confronto

Logici

Assegnazione

Operatori

Aritmetici

Confronto

Logici

Assegnazione

```
=  
+= incremento  
-= decremento  
*= moltiplicatore  
/= divisore  
%=  
//=  
**=
```

Operatori

Aritmetici

Confronto

Logici

Assegnazione

Identità

Operatori

Aritmetici

Confronto

Logici

Assegnazione

Identità

is
is not

Operatori

Aritmetici

Confronto

Logici

Assegnazione

Identità

is
is not

Gli operatori di identità vengono usati per verificare se due operandi sono uguali(se si riferiscono allo stesso oggetto), ovvero se puntano alla stessa locazione di memoria.

```
type(1) is int = True  
type("1") is int = False  
type("1") is str = True
```

Operatori

Aritmetici

Confronto

Logici

Assegnazione

Identità

Appartenenza (Membership)

Operatori

Aritmetici

Confronto

Logici

Assegnazione

Identità

Appartenenza (Membership)

in
not in

Operatori

Aritmetici

Confronto

Logici

Assegnazione

Identità

Appartenenza (Membership)

in
not in

```
x='casa'  
'c' in x = True  
'o' in x = False
```

Operatori

Aritmetici

Confronto

Logici

Assegnazione

Identità

Appartenenza (Membership)

in
not in

x='casa'

'c' in x = True
'o' in x = False

Ricordiamo che:

'casa' == ['c','a','s','a']

Operatori

Aritmetici

Confronto

Logici

Assegnazione

Identità

Appartenenza (Membership)

Binari (Bitwise)

Operatori

Aritmetici

Confronto

Logici

Assegnazione

Identità

Appartenenza (Membership)

Binari (Bitwise)

Confrontano bit per bit e ritornano

& and 1 nelle posizioni in cui entrambi bit sono uguali ad 1

| or 1 nelle posizioni in cui almeno un bit è uguale a 1

^ xor 1 solo se uno dei due bit è uguale a 1

Operatori

Aritmetici

Confronto

Logici

Assegnazione

Identità

Appartenenza (Membership)

Binari (Bitwise)

Confrontano bit per bit e ritornano

& and 1 nelle posizioni in cui entrambi bit sono uguali ad 1

| or 1 nelle posizioni in cui almeno un bit è uguale a 1

^ xor 1 solo se uno dei due bit è uguale a 1

0b110 & 0b010 = 0b010 (2)

0b100 & 0b001 = 0b000 (0)

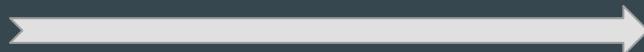
0b110 | 0b011 = 0b111 (7)

0b110 ^ 0b011 = 0b101 (5)

Dichiaratori di Condizione

Possiamo annidare (*nesting*) le condizioni o generare una condizione composta

`if cond1:`

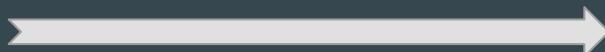


Permette di esplorare tutte e quattro le possibilità

`if cond2:`

....

`if cond1 and cond2:`



O tutte e due True o tutte e due False

....

I Cicli

Python ha due comandi di loop primitivi:

While

viene eseguito sino a quando una condizione viene verificata

```
i = 1
while i < 6:
    print(i)
    i += 1
```

For

I Cicli

Python ha due comandi di loop primitivi:

While

viene eseguito sino a quando una condizione viene verificata

For

Viene eseguito un numero predefinito di volte

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    print(x)
```

I Cicli - while

Ci sono dei comandi che permettono di controllare il flusso all'interno di un ciclo:

`break` Interrompe il ciclo anche se la condizione è True

I Cicli - while

Ci sono dei comandi che permettono di controllare il flusso all'interno di un ciclo:

break

Interrompe il ciclo anche se la condizione è True

```
i = 1
while i < 6:
    print(i)
    if i == 3:
        break
    i += 1
```

I Cicli - while

Ci sono dei comandi che permettono di controllare il flusso all'interno di un ciclo:

`break` Interrompe il ciclo anche se la condizione è True

`continue` Interrompe l'iterazione corrente e passa alla successiva

I Cicli - while

Ci sono dei comandi che permettono di controllare il flusso all'interno di un ciclo:

`break` Interrompe il ciclo anche se la condizione è True

`continue` Interrompe l'iterazione corrente e passa alla successiva

```
i = 0
while i < 6:
    i += 1
    if i == 3:
        continue
    print(i)
```

I Cicli - while

Ci sono dei comandi che permettono di controllare il flusso all'interno di un ciclo:

`break` Interrompe il ciclo anche se la condizione è True

`continue` Interrompe l'iterazione corrente e passa alla successiva

`else` Esegue un blocco di codice quando la condizione diventa False

I Cicli - while

Ci sono dei comandi che permettono di controllare il flusso all'interno di un ciclo:

`break` Interrompe il ciclo anche se la condizione è True

`continue` Interrompe l'iterazione corrente e passa alla successiva

`else` Esegue un blocco di codice quando la condizione diventa False

```
i = 1
while i < 6:
    print(i)
    i += 1
else:
    print("i non è più minore di 6")
```

I Cicli - for

Ci sono dei comandi che permettono di controllare il flusso all'interno di un ciclo:

break Interrompe il ciclo anche se la condizione è True

continue Interrompe l'iterazione corrente e passa alla successiva

else Esegue un blocco di codice quando la condizione diventa False

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    if x == "banana":
        break
    print(x)
```

I Cicli - for

Ci sono dei comandi che permettono di controllare il flusso all'interno di un ciclo:

break Interrompe il ciclo anche se la condizione è True

continue Interrompe l'iterazione corrente e passa alla successiva

else Esegue un blocco di codice quando la condizione diventa False

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    if x == "banana":
        continue
    print(x)
```

I Cicli - for

Ci sono dei comandi che permettono di controllare il flusso all'interno di un ciclo:

`break` Interrompe il ciclo anche se la condizione è True

`continue` Interrompe l'iterazione corrente e passa alla successiva

`else` Esegue un blocco di codice quando la condizione diventa False

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    if x == "banana":
        continue
    print(x)
else:
    print("Finally finished!")
```

I Cicli - for

Ci sono dei comandi che permettono di controllare il flusso all'interno di un ciclo:

`break` Interrompe il ciclo anche se la condizione è True

`continue` Interrompe l'iterazione corrente e passa alla successiva

`else` Esegue un blocco di codice quando la condizione diventa False

Ricordiamoci che le stringhe sono dei vettori

```
fruits = "apple"
for x in fruits:
    print(x)
```

I Cicli - for

Con for sono spesso usate due funzioni:

`range` Ritorna una lista di numeri

I Cicli - for

Con for sono spesso usate due funzioni:

`range` Ritorna una lista di numeri

```
for n in range(3, 20, 2):  
    print(n)
```

I Cicli - for

Con for sono spesso usate due funzioni:

range Ritorna una lista di numeri

enumare Converte una collezione in un oggetto numerato

I Cicli - for

Con for sono spesso usate due funzioni:

range Ritorna una lista di numeri

enumare Converte una collezione in un oggetto numerato

```
x = ('apple', 'banana', 'cherry')
y = enumerate(x)
```

Le Funzioni

Le funzioni sono un blocco di codice che viene eseguito su comando:

Le Funzioni

Le funzioni sono un blocco di codice che viene eseguito su comando:

```
def myFunct():
    print("Ciao")
```

```
myFunct()
```

Le Funzioni

Le funzioni sono un blocco di codice che viene eseguito su comando:

```
def myFunct():
    print("Ciao")

myFunct()
```

Ad una funzione si possono passare degli argomenti o dei parametri:

Le Funzioni

Le funzioni sono un blocco di codice che viene eseguito su comando:

```
def myFunct():
    print("Ciao")

myFunct()
```

Ad una funzione si possono passare degli argomenti o dei parametri:

```
def myFunct(arg, par1=False)
    print(f"Hello {arg}")
    if par1:
        print("Oggi è una bella giornata")

myFunct("Simone")
myFunct("Simone", True)
myFunct("Simone", par1 = True)
```

Le Funzioni

Se non si conosce il numero degli argomenti che verranno passati alla funzione, si aggiungerà uno ***** prima del nome del parametro nella definizione della funzione.

Se non si conosce il numero degli parametri che verranno passati alla funzione, si aggiungerà ****** prima del nome del parametro nella definizione della funzione.

E' possibile "estendere" una funzione tramite delle speciali funzioni dette decorazioni, che permettono di eseguire del codice prima e/o dopo l'esecuzione della funzione base.

Queste vengono applicate alla funzione aggiungendo subito prima della definizione della funzione @ + il nome della decorazione.

Le funzioni Lambda

In Python le **funzioni Lambda** sono dei particolari costrutti sintattici derivati dal linguaggio Lisp e chiamati anche **funzioni anonime**; rispetto alle comuni funzioni definite dall'utente esse non sono associate ad un nome, da qui la caratteristica di essere "anonime", non vengono introdotte dalla parola chiave def, prevedendo invece la keyword **lambda**, e possono essere seguite soltanto da un'unica espressione.

Le funzioni Lambda

In Python le **funzioni Lambda** sono dei particolari costrutti sintattici derivati dal linguaggio Lisp e chiamati anche **funzioni anonime**; rispetto alle comuni funzioni definite dall'utente esse non sono associate ad un nome, da qui la caratteristica di essere "anonime", non vengono introdotte dalla parola chiave def, prevedendo invece la keyword **lambda**, e possono essere seguite soltanto da un'unica espressione.

```
def x(a): return a + 10  
  
print(x(5))
```

Le funzioni Lambda

In Python le **funzioni Lambda** sono dei particolari costrutti sintattici derivati dal linguaggio Lisp e chiamati anche **funzioni anonime**; rispetto alle comuni funzioni definite dall'utente esse non sono associate ad un nome, da qui la caratteristica di essere "anonime", non vengono introdotte dalla parola chiave def, prevedendo invece la keyword **lambda**, e possono essere seguite soltanto da un'unica espressione.

```
def x(a): return a + 10
```

```
print(x(5))
```

```
x = lambda a: a+10
```

```
print(x(5))
```

Le funzioni Lambda

In Python le **funzioni Lambda** sono dei particolari costrutti sintattici derivati dal linguaggio Lisp e chiamati anche **funzioni anonime**; rispetto alle comuni funzioni definite dall'utente esse non sono associate ad un nome, da qui la caratteristica di essere "anonime", non vengono introdotte dalla parola chiave def, prevedendo invece la keyword **lambda**, e possono essere seguite soltanto da un'unica espressione.

```
def x(a): return a + 10
```

```
print(x(5))
```

```
x = lambda a: a+10
```

```
print(x(5))
```

La versatilità delle funzioni lambda si evidenzia quando sono utilizzate all'interno di un'altra funzione:

Le funzioni Lambda

In Python le **funzioni Lambda** sono dei particolari costrutti sintattici derivati dal linguaggio Lisp e chiamati anche **funzioni anonime**; rispetto alle comuni funzioni definite dall'utente esse non sono associate ad un nome, da qui la caratteristica di essere "anonime", non vengono introdotte dalla parola chiave def, prevedendo invece la keyword **lambda**, e possono essere seguite soltanto da un'unica espressione.

```
def x(a): return a + 10
```

```
print(x(5))
```

```
x = lambda a: a+10
```

```
print(x(5))
```

La versatilità delle funzioni lambda si evidenzia quando sono utilizzate all'interno di un'altra funzione:

```
def myfunc(n):  
    return lambda a: a * n
```

```
mytripler = myfunc(3)  
print(mytripler(11))
```

Quando non Usare le Lambda

1. Se si assegna un nome ad una funzione lambda

Quando non Usare le Lambda

1. Se si assegna un nome ad una funzione lambda

#Bad

```
triple = lambda x: x*3
```

#Good

```
def triple(x):  
    return x*3
```

Se provate ad incollare la prima riga su Visual Studio IntellyCode convertirà automaticamente la lambda in una funzione per rispettare i canoni di best Practice PEP8

Quando non Usare le Lambda

1. Se si assegna un nome ad una funzione lambda
2. Se si deve usare una funzione all'interno di una lambda

Quando non Usare le Lambda

1. Se si assegna un nome ad una funzione lambda
2. Se si deve usare una funzione all'interno di una lambda

#Bad

```
map(lambda x: abs(x), list_3)
```

#Good

```
map(abs, list_3)
```

#Good

```
map(lambda x: pow(x, 2), float_nums)
```

Quando non Usare le Lambda

1. Se si assegna un nome ad una funzione lambda
2. Se si deve usare una funzione all'interno di una lambda

#Bad

```
map(lambda x: abs(x), list_3)
```

#Good

```
map(abs, list_3)
```

#Good

```
map(lambda x: pow(x, 2), float_nums)
```

La funzione map() esegue una specifica funzione su ogni elemento di un iterabile:

```
def myfunc(a):
    return len(a)

x = map(myfunc, ('apple', 'banana', 'cherry'))

print(x)

#convert the map into a list, for readability:
print(list(x))
```

Quando non Usare le Lambda

1. Se si assegna un nome ad una funzione lambda
2. Se si deve usare una funzione all'interno di una lambda
3. Quando l'uso di più linee di codice rendono il codice più leggibile

Lo Zen di Python (PEP20)

1. Beautiful is better than ugly.
2. Explicit is better than implicit.
3. Simple is better than complex.
4. Complex is better than complicated.
5. Flat is better than nested.
6. Sparse is better than dense.
7. Readability counts.
8. Special cases aren't special enough to break the rules.
9. Although practicality beats purity.
10. Errors should never pass silently.
11. Unless explicitly silenced.
12. In the face of ambiguity, refuse the temptation to guess.
13. There should be one-- and preferably only one --obvious way to do it.
14. Although that way may not be obvious at first unless you're Dutch.
15. Now is better than never.
16. Although never is often better than *right* now.
17. If the implementation is hard to explain, it's a bad idea.
18. If the implementation is easy to explain, it may be a good idea.
19. Namespaces are one honking great idea -- let's do more of those!

Macchina a Stati Finiti

Finite State Machine

E' un tipo di automa che permette di descrivere con precisione e in maniera formale il comportamento di molti sistemi tramite una quintupla: QIUt_w

Q insieme degli stati interni

I insieme degli input

U insieme delle uscite

t funzione di transizione

w funzione di uscita

FSM - Introduzione

Argparse

Per passare i parametri da linea di comando utilizziamo il modulo argparse

```
parser=argparse.ArgumentParser(description='Finite State Machine')
parser.add_argument('-c', '--command', metavar='COMMAND', help='Command File', default='timeline.txt')
parser.add_argument('-d', '--debug', action='store_true', help='Debug Mode')
parser.add_argument('-v', '--verbose', action='store_true', help='Verbose Mode')
args=parser.parse_args()
```

in questo caso introduciamo un parametro opzionale per indicare il file di comandi dell'automa e due flag per settare la modalità di debug e la modalità di verbose

<https://docs.python.org/3/howto/argparse.html>

FSM - Introduzione - Logging

Per effettuare il logging utilizziamo il modulo logging e lo inizializziamo nel seguente modo:

```
import logging
from commons import FMODE

__version__ ="1.2.0"
def logInit(logName, logger, logLevel=20,fileMode=FMODE.APPEND):
    """Inizialize the logger"""
    flag = False
    if not logLevel in [0, 10, 20, 30, 40, 50]:
        oldLevel=logLevel
        flag=True
        logLevel = 20
    logging.basicConfig(filename=logName,
                        level=logLevel,
                        filemode=fileMode,
                        format='%(asctime)s | %(levelname)-8s | %(name)-7s | %(module)-10s | %(funcName)-20s | %(message)s',
                        datefmt='%m/%d/%Y %I:%M:%S %p')
    al = logging.getLogger(logger)
    if flag:
        al.warning(f"Log level {oldLevel} is not valid. Used the default value 20" )
    return al # logging
```

FSM - Introduzione - Logging

setup di Visual Studio Code:

Installazione dell'estensione **Log Viewer**.

Aggiunta nel setting del workspace di:

```
"settings": {  
    "logViewer.watch": [  
        {  
            "title": "General Log",  
            "pattern": "Examples/StateMachine/StateMachine.log"  
        },  
    ]  
}
```

```
1 05/30/2022 08:15:08 PM | DEBUG | StateMachine | state | run | Reading the command file  
2 |
```

FSM - Introduzione - Modalità Verbosa

La modalità verbosa è quella modalità in cui tutti i messaggi di log vengono anche stampati a schermo.

Per renderla più leggibile utilizziamo il modulo *rich*

```
5   from rich import print  
12  |     if debug and verbose:  
13  |         print(f"MSG.DEBUG}Reading the command file")
```

FSM - Introduzione - Modalità Verbosa

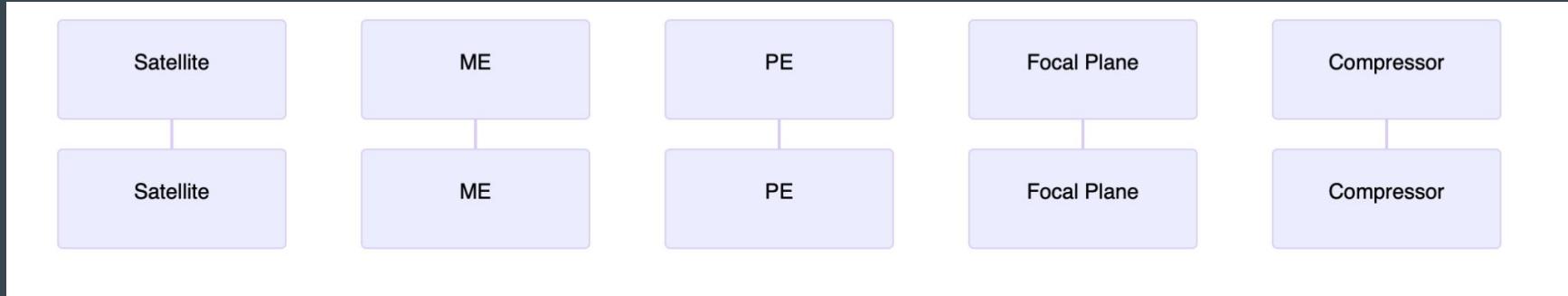
La modalità verbosa è quella modalità in cui tutti i messaggi di log vengono anche stampati a schermo.

Per renderla più leggibile utilizziamo il modulo *rich*

```
5   from rich import print  
12  |     if debug and verbose:  
13  |         print(f"{MSG.DEBUG}Reading the command file")
```

```
class MSG:  
    DEBUG="[green] [DEBUG] [/green] "  
    INFO="[blue] [INFO] [/blue] "
```

FSM - Schema



ME: Main Electronics

PE: Proximity Electronics

Sono gli unici elementi “attivi” (dotati di processore).

La PE guida l’HW. La ME ha l’incarico di validare e smistare i telecomandi ed organizzare i pacchetti.

Il compressore è solitamente una **FPGA** (Field Programmable Gate Array) ovvero un dispositivo hardware elettronico formato da un circuito integrato le cui funzionalità logiche di elaborazione sono appositamente programmabili

FSM - Database dei Comandi

Il database dei comandi è il luogo in cui sono contenuti tutti i comandi che possiamo dare alla nostra macchina con l'informazione del sotto modulo di destinazione, lo stato d'ingresso, transiente e di uscita.

FSM - Database dei Comandi

Il database dei comandi è il luogo in cui sono contenuti tutti i comandi che possiamo dare alla nostra macchina con l'informazione del sotto modulo di destinazione, lo stato d'ingresso, transiente e di uscita.

Questo database nelle missioni spaziali è detto **Mission Information Database (MIB)**.

FSM - Database dei Comandi

Il database dei comandi è il luogo in cui sono contenuti tutti i comandi che possiamo dare alla nostra macchina con l'informazione del sotto modulo di destinazione, lo stato d'ingresso, transiente e di uscita.

Nel nostro caso usiamo un file CSV denominato ***commandsTable.csv***

	TC	Destination	Initial State	Transient State	Final State	Description
	NSS00001	ME	OFF	BUSY	IDLE	ME Switch on
	NSS00002	ME	IDLE	BUSY	OFF	ME Switch off
	NSS00003	PE	OFF	BUSY	ON	PE Switch on

FSM - Database dei Comandi

Il database dei comandi è il luogo in cui sono contenuti tutti i comandi che possiamo dare alla nostra macchina con l'informazione del sotto modulo di destinazione, lo stato d'ingresso, transiente e di uscita.

Nel nostro caso usiamo un file CSV denominato ***commandsTable.csv***

```
def readCmdDb():
    with open('commandsTable.csv','r') as f:
        lines=f.readlines()
    commandTable={}
    for line in lines[1:]:
        seg=line.strip().split(',')
        commandTable[seg[0]]={
            'destination':seg[1],
            'initial':seg[2],
            'transient':seg[3],
            'final':seg[4]
        }
    return commandTable
```

	TC	Destination	Initial State	Transient State	Final State	Description
	NSS00001	ME	OFF	BUSY	IDLE	ME Switch on
	NSS00002	ME	IDLE	BUSY	OFF	ME Switch off
	NSS00003	PE	OFF	BUSY	ON	PE Switch on

FSM - Command Stack

Il command stack è la sequenza di comandi, con i relativi delay, che devono essere eseguiti dall'automa.

Per quello che ci riguarda si tratta di un semplice file di testo

FSM - Command Stack

Il command stack è la sequenza di comandi, con i relativi delay, che devono essere eseguiti dall'automa.

Per quello che ci riguarda si tratta di un semplice file di testo

```
1 # Time (seconds), TC
2 | 1,NSS00001
3 | # 3,NSS00002
4 | 8,NSS00003
5 | 12,NSS00002
6
```

FSM - Command Stack

Il command stack è la sequenza di comandi, con i relativi delay, che devono essere eseguiti dall'automa.

Per quello che ci riguarda si tratta di un semplice file di testo

```
with open(command, FMODE.READ) as f:  
    lines=f.readlines()  
  
for line in lines:  
    if line.strip().startswith('#'):  
        continue  
    else:  
        print(line.strip())
```

```
1 # Time (seconds), TC  
2 1,NSS00001  
3 # 3,NSS00002  
4 8,NSS00003  
5 12,NSS00002  
6
```

FSM - Command Stack

Il command stack è la sequenza di comandi, con i relativi delay, che devono essere eseguiti dall'automa.

Per quello che ci riguarda si tratta di un semplice file di testo

```
with open(command, FMODE.READ) as f:  
    lines=f.readlines()  
  
for line in lines:  
    if line.strip().startswith('#'):  
        continue  
    else:  
        print(line.strip())
```

```
1 # Time (seconds), TC  
2 1,NSS00001  
3 # 3,NSS00002  
4 8,NSS00003  
5 12,NSS00002  
6
```

```
class FMODE:  
    READ='r'  
    APPEND='a'  
    WRITE='w'
```

FSM - Orologio di Sistema

Poiché i comandi sono dati in un tempo relativo dobbiamo creare un orologio di sistema.

Per prima cosa registriamo il momento in cui viene inizializzata la macchina:

```
def __init__(self):
    self.start=time.time()
```

FSM - Orologio di Sistema

Poiché i comandi sono dati in un tempo relativo dobbiamo creare un orologio di sistema.

Per prima cosa registriamo il momento in cui viene inizializzata la macchina:

```
def __init__(self):
    self.start=time.time()
```

Creiamo poi un metodo per leggere il tempo:

```
def getSeconds(self):
    now=time.time()-self.start
    return now
```

FSM - L'automa principale

```
class StateMachine:  
    def __init__(self, name, initialState, tranTable):  
        self.name = name  
        self.state = initialState  
        self.transitionTable = tranTable
```

FSM - L'automa principale

```
class StateMachine:  
    def __init__(self, name, initialState, tranTable):  
        self.name = name  
        self.state = initialState  
        self.transitionTable = tranTable
```

```
class PE(StateMachine):  
    def __init__(self, name, initialState, tranTable):  
        super().__init__(name, initialState, tranTable)  
        self.Commands= PECommands()
```

```
class ME(StateMachine):  
    def __init__(self, name, initialState, tranTable):  
        super().__init__(name, initialState, tranTable)  
        self.PE=PE('PE',STATE.OFF,tranTable)  
        self.Commands = MECommands()
```

FSM - I Comandi

```
class MECommands:  
    def __init__(self, verbose: bool = False, console: Console = None):  
        self.verbose = verbose  
        self.console = console  
  
    @message(text='Booting...')  
    def NSS00001(self):  
        """Boot Command"""  
        print(f'{MSG.INFO}[magenta]TM(5,1) [/magenta] - Boot Report')  
        sleep(5)  
  
    @message(text='Shuting down...')  
    def NSS00002(self):  
        """Shooting Down Command"""  
        sleep(5)  
  
class PECommands:  
    def __init__(self, verbose: bool = False, console: Console = None):  
        self.verbose = verbose  
        self.console = console  
  
    @message(text="PE...ON ")  
    def NSS00003(self):  
        """PE ON Command"""  
        sleep(1)
```

FSM - I Comandi - Decoratore

```
15  def message(text: str):
16      def decorate(f):
17          @wraps(f)
18          def inner(*args, **kwargs):
19              with Status(text, spinner='aesthetic', console=args[0].console):
20                  ret = f(*args, **kwargs)
21                  if args[0].verbose:
22                      print(f"\u001b[32m{MSG.INFO} {f.__name__} executed\u001b[0m")
23                  return ret
24          return inner
25      return decorate
```

Argomenti

- Package e Moduli
- Variabili
- Classi e Oggetti
- Versionamento del software
- Dichiarazione di Condizione
- Operatori
- Cicli
- Funzioni
- Decoratori
- Namespace
- Lambda
- I/O
- Eccezioni
- PyPI

Packages:

- `argparse`
 - click
 - rich
 - rich-click
- `logging`
 - pandas
 - numpy
 - scipy
 - matplotlib
 - multiprocessing
 - sqlite
 - ElementTree

Packages - rich

Rich è una libreria Python per scrivere “rich text” (con colore e stile) sul terminale e per visualizzare contenuti avanzati come tabelle, markdown e codice con evidenziazione della sintassi.

Rich permette di rendere visivamente accattivanti le applicazioni a riga di comando e presentare i dati in modo più leggibile. Rich può anche essere un utile aiuto per il debugging mediante una bella stampa e l'evidenziazione della sintassi delle strutture di dati.

Moduli principali:

- Console
- Prompt
- Progress
- Table
- Panel

Packages - rich - console

Per il controllo completo sulla formattazione del terminale, Rich offre una classe Console.

Permette di gestire messaggi di stato, separatori, formattazione, spinner, etc. oltre a fornire la possibilità di salvare tutto ciò che è stato stampato a schermo

Packages - rich - prompt

Rich ha una serie di classi Prompt che chiedono all'utente un input con un loop fino a quando non viene ricevuta una risposta valida.

Un esempio delle funzionalità si può ottenere con il comando:

```
python3 -m rich.prompt
```

Packages - rich - progress

Rich può visualizzare informazioni continuamente aggiornate sullo stato di avanzamento di attività/copie di file di lunga durata, ecc.

Le informazioni visualizzate sono configurabili, l'impostazione predefinita visualizzerà una descrizione dell'attività, una barra di avanzamento, la percentuale di completamento e il tempo rimanente stimato.

Un esempio delle funzionalità si può ottenere con il comando:

```
python3 -m rich.progress
```

Packages - rich - table

La classe Table di Rich offre una varietà di modi per rendere i dati tabulari al terminale.

Un esempio delle funzionalità si può ottenere con il comando:

```
python3 -m rich.table
```

Packages - rich - panel

Per disegnare un bordo attorno al testo o ad altri renderizzabili, si può utilizzare Panel con l'oggetto renderizzabile come primo argomento posizionale.

Un esempio delle funzionalità si può ottenere con il comando:

```
python3 -m rich.panel
```

FSM

Torniamo alla nostra macchina e guardiamo il codice nel modulo newState2.

```
usage: newState2.py [-h] [-f FILE] [-i] [-C FILE] [-d] [-v]
```

Finite State Machine

options:

-h, --help	show this help message and exit
-f FILE, --command-file FILE	Command File
-i, --interactive	enable the interactive mode
-C FILE, --configure FILE	Configuration file
-d, --debug	Debug Mode
-v, --verbose	Verbose Mode

FSM

Torniamo alla nostra macchina e guardiamo il codice nel modulo newState2.

```
usage: newState2.py [-h] [-f FILE] [-i] [-C FILE] [-d] [-v]
```

Finite State Machine

options:

-h, --help show this help message and exit

-f FILE, --command-file FILE Command File

-i, --interactive enable the intercative mode

-C FILE, --configure FILE Configuration file

-d, --debug Debug Mode

-v, --verbose Verbose Mode

FSM - Configuration File

```
logFile: StateMachine.log  
cmdHistory: history.csv
```

Scritto nel formato YAML

YAML (pronunciato 'jæməl, in rima con camel) è un formato per la serializzazione di dati utilizzabile da esseri umani. Il linguaggio sfrutta concetti di altri linguaggi come il C, il Perl e il Python e idee dal formato XML e dal formato per la posta elettronica (RFC2822).

FSM - interactive mode

Utilizza prompt per prendere i comandi da linea di comando ed eseguirli.

Trascrive su di un file la timeline dell'esecuzione.

FSM - Final

Il codice finale con l'acquisizione e compressione dell'immagine è nel folder StateMachine2.

In questo codice è stata soppressa la scrittura dei pacchetti, per rendere più leggibile il codice.

HTML

HTML

HTML (HyperText Markup Language) è un linguaggio di markup.

La struttura fondamentale di una pagina HTML è:

```
<!DOCTYPE html>

<html>

  <head>
    <title>Page Title</title>
  </head>

  <body>
    <h1>My First Heading</h1>
    <p>My first paragraph.</p>
  </body>

</html>
```



HTML - Classi e ID

Le **classi** servono per definire una tipologia di elementi, cioè per attribuire uno scopo e/o una presentazione ad un sottoinsieme di elementi con caratteristiche e funzionalità comuni di una pagina HTML.

Gli **ID** servono per definire un elemento unico in una pagina, con un unico e preciso scopo, nella maggior parte dei casi tale scopo è determinare una sezione in una pagina HTML.

In sostanza, quando sappiamo che un elemento sarà unico useremo un id. Negli altri casi, se non disponiamo di alternative possiamo usare una classe.

HTML - Class e ID

```
<div class="city">  
  <h2>London</h2>  
  
  <p>London is the capital of England.</p>  
  
</div>
```

```
<div class="city">  
  <h2>Paris</h2>  
  
  <p>Paris is the capital of France.</p>  
  
</div>
```

```
<h1 id="myHeader">My Header</h1>
```

HTML - CSS

CSS (Cascading Style Sheets) è un linguaggio usato per definire la formattazione di documenti HTML, XHTML e XML.

Può essere inserito nella pagina o come file esterno.

Nella pagina può essere inserito in modalità inline o interna

Modalità inline

```
<h1 style="color:blue;">A Blue Heading</h1>  
<p style="color:red;">A red paragraph.</p>
```

Modalità interna

```
<head>  
  <style>  
    body {background-color: powderblue;}  
    h1   {color: blue;}  
    p    {color: red;}  
  </style>  
</head>
```

Modalità esterna

```
<head>  
  <link rel="stylesheet" href="styles.css">  
</head>
```

CSS



HTML - CSS

Vediamo un esempio pratico (06 - HTML/esempio[1-6].html)

HTML - Fundamental Tags

hx header (con x tra 1 e 6)

hr horizontal rule

p paragrafo

br interruzione di linea

table, tr,td tabella , riga, cella o dato

a ancora per i link

div division, definisce un blocco della pagina html

HTML - JavaScript

Un esempio è illustrato in `esempio_09.html`.

In questo caso abbiamo letto la libreria dal CDN di Google.

Abbiamo eseguito una funzione anonima appena si è verificato l'evento `ready` sull'oggetto `document` (appena la pagina è stata caricata).

La funzione ha creato un handler sull'elemento con id `btn` (# -> id, . -> class).

Quando su `#btn` si verifica l'evento `click` esegue una funzione anonima, che esegue uno `slide` (con l'opzione di alternanza, cioè in se out o out se in) dell'elemento `#principale` con un tempo di 1000ms

HTML - Toolkit

i toolkit sono delle librerie con gli stili già predefiniti e che possono essere integrate facilmente nelle nostre pagine.

i principali sono:

- Bootstrap
- JQuery-UI

Spesso è richiesto del codice JavaScript per il funzionamento.

CDN (Content Delivery Network) permettono di distribuire le librerie da server terzi tramite un sistema distribuito geograficamente e quindi con elevata efficienza.

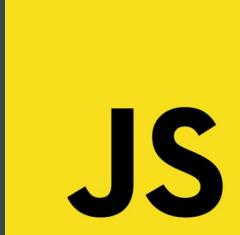
HTML - JavaScript

JavaScript è un linguaggio di programmazione multi paradigma **orientato agli eventi**, comunemente utilizzato nella programmazione Web lato client (esteso poi anche al lato server, NODE.js) per la creazione, in siti web e applicazioni web, di effetti dinamici interattivi tramite funzioni di script invocate da eventi innescati a loro volta in vari modi dall'utente sulla pagina web in uso.

Vedi esempio_08.html

Anche in questo caso si possono utilizzare librerie:

- jQuery
- Cash
- Zepto
- Syncfusion Essential JS2
- UmbrellaJS



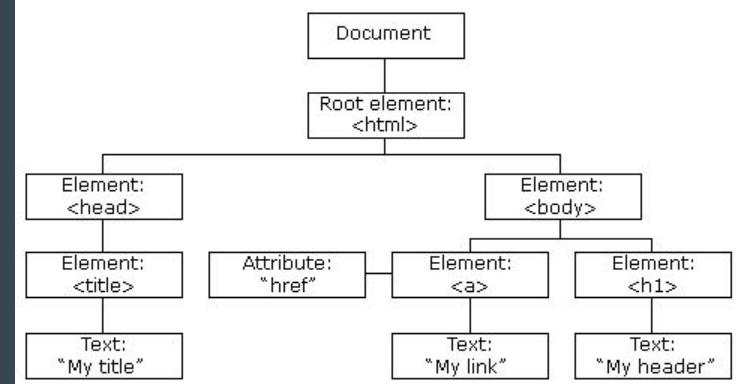
HTML - JavaScript - DOM

HTML DOM (Document Object Model)

Quando la pagina viene letta il browser crea un oggetto di tipo DOM per interpretarla

```
<!DOCTYPE html>

<html>
  <head>
    <title>My Title</title>
  </head>
  <body>
    <h1>My header</h1>
    <a href="#">My Link</a>
  </body>
</html>
```



HTML - JS -AJAX

AJAX Asynchronous JavaScript And XML

Non è un linguaggio di programmazione ma una tecnica di programmazione che utilizza l'oggetto **XMLHttpRequest** per effettuare richieste al server e JavaScript e HTML DOM per visualizzarli.

HTML -JS -AJAX Example

```
// Using the core $.ajax() method
$.ajax({
  // The URL for the request
  url: "post.php",
  // The data to send (will be converted to a query string)
  data: {
    id: 123
  },
  // Whether this is a POST or GET request
  type: "GET",
  // The type of data we expect back
  dataType : "json",
})
// Code to run if the request succeeds (is done);
// The response is passed to the function
.done(function( json ) {
  $("<h1>").text( json.title ).appendTo( "body" );
  $("<div class='content'>").html( json.html ).appendTo( "body" );
})
// Code to run if the request fails; the raw request and
// status codes are passed to the function
.fail(function( xhr, status, errorThrown ) {
  alert( "Sorry, there was a problem!" );
  console.log( "Error: " + errorThrown );
  console.log( "Status: " + status );
  console.dir( xhr );
})
// Code to run regardless of success or failure;
.always(function( xhr, status ) {
  alert("The request is complete!");
});
});
```

Python Web Framework

Web Framework

I web framework sono delle collezioni di pacchetti o moduli che permettono di scrivere delle WebApplication o dei servizi senza doversi preoccupare di dettagli di basso livello come i protocolli, i socket o la gestione dei thread o processi.

In genere, i framework forniscono supporto per una serie di attività come l'interpretazione delle richieste (ottenimento di parametri del modulo, gestione di cookie e sessioni), la produzione di risposte (presentazione dei dati come HTML o in altri formati), la memorizzazione dei dati in modo persistente e così via.

Poiché un'applicazione Web non banale richiederà un numero di diversi tipi di astrazioni, spesso impilate l'una sull'altra, quei framework che tentano di fornire una soluzione completa per le applicazioni sono spesso noti come framework full-stack in quanto tentano di fornire componenti per ogni strato della pila.

Web Framework

Full-Stack Framework

- Dash
- Django
- Masonite
- TurboGears
- web2py

Non Full-Stack Framework

- aiohttp
- Bottle
- CherryPy
- Falcon
- FastAPI
- Flask
- Hug
- Pyramid

Web Framework

Full-Stack Framework

- Dash
- **Django**
- Masonite
- TurboGears
- web2py

Non Full-Stack Framework

- aiohttp
- Bottle
- CherryPy
- Falcon
- FastAPI
- **Flask**
- Hug
- Pyramid

Django

django

django è un Python Web-Framework di alto livello che incoraggia uno sviluppo rapido e un design pulito e pragmatico.

Si prende carico di gran parte dei problemi di sviluppo web, così ci si può concentrare sulla scrittura della applicazione senza dover reinventare la ruota. È open source.



Installazione

django è disponibile su PyPi, quindi è installabile via pip o conda.

```
$ python3 -m pip install --upgrade pip
```

```
$ python3 -m pip install django
```

Per verificare che l'installazione sia andata a buon fine si può verificare il numero di versione:

```
$ python3 -m django --version
```

Progetti

Una istanza django viene chiamata **progetto**.

Un **progetto** include la configurazione del database, le opzioni specifiche di Django e le impostazioni specifiche dell'applicazione.

Per creare un progetto chiamato *mysite* bisogna digitare:

```
$ django-admin startproject mysite
```

Questo creerà un albero contenente i file di configurazione.

Struttura del progetto

```
mysite
└── manage.py
└── mysite
    ├── __init__.py
    ├── asgi.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py
```

1 directory, 6 files

- il folder esterno **mysite/** è il contenitore del progetto. Il nome di questo folder non è importante per django e può essere rinominato a proprio piacimento.
- **manage.py**: E' una command-line utility che permette di interagire con il progetto in diversi modi.
- il folder interno **mysite/** è il Package del progetto. il nome del folder è il nome del package.
- **mysite/__init__.py**: è un file vuoto che informa Python che questo folder deve essere considerato un package.
- **mysite/settings.py**: contiene la configurazione del progetto Django. Contiene anche tutte le informazioni necessarie per configurare i vari parametri.
- **mysite/urls.py**: Contiene le dichiarazioni di tutte le URL del progetto Django.
- **mysite/asgi.py**: è l'entry-point per rilasciare web server ASGI-compatible (**Asynchronous Server Gateway Interface**)
- **mysite/wsgi.py**: è l'entry-point per rilasciare web server WSGI-compatible (**Web Server Gateway Interface**)

Struttura del progetto

```
mysite
└── manage.py
└── mysite
    ├── __init__.py
    ├── asgi.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py
```

L'Asynchronous Server Gateway Interface (ASGI) è una convenzione di chiamata per i server Web per inoltrare le richieste a framework e applicazioni del linguaggio di programmazione Python con funzionalità asincrone.

- il folder esterno **mysite/** è il contenitore del progetto. Il nome di questo folder non è importante per django e può essere rinominato a proprio piacimento.
- **manage.py**: E' una command-line utility che permette di interagire con il progetto in diversi modi.
- il folder interno **mysite/** è il Package del progetto. il nome del folder è il nome del package.
- **mysite/__init__.py**: è un file vuoto che informa Python che questo folder deve essere considerato un package.
- **mysite/settings.py**: contiene la configurazione del progetto Django. Contiene anche tutte le informazioni necessarie per configurare i vari parametri.
- **mysite/urls.py**: Contiene le dichiarazioni di tutte le URL del progetto Django.
- **mysite/asgi.py**: è l'entry-point per rilasciare web server ASGI-compatible (**Asynchronous Server Gateway Interface**)
- **mysite/wsgi.py**: è l'entry-point per rilasciare web server WSGI-compatible (**Web Server Gateway Interface**)

Struttura del progetto

```
mysite
└── manage.py
└── mysite
    ├── __init__.py
    ├── asgi.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py
```

L'interfaccia Web Server Gateway è una semplice convenzione di chiamata per i server Web per inoltrare richieste ad applicazioni Web o framework scritti nel linguaggio di programmazione Python.

- il folder esterno **mysite/** è il contenitore del progetto. Il nome di questo folder non è importante per django e può essere rinominato a proprio piacimento.
- **manage.py**: E' una command-line utility che permette di interagire con il progetto in diversi modi.
- il folder interno **mysite/** è il Package del progetto. il nome del folder è il nome del package.
- **mysite/__init__.py**: è un file vuoto che informa Python che questo folder deve essere considerato un package.
- **mysite/settings.py**: contiene la configurazione del progetto Django. Contiene anche tutte le informazioni necessarie per configurare i vari parametri.
- **mysite/urls.py**: Contiene le dichiarazioni di tutte le URL del progetto Django.
- **mysite/asgi.py**: è l'entry-point per rilasciare web server ASGI-compatible (**Asynchronous Server Gateway Interface**)
- **mysite/wsgi.py**: è l'entry-point per rilasciare web server WSGI-compatible (**Web Server Gateway Interface**)

Struttura del progetto

```
mysite
└── manage.py
└── mysite
    ├── __init__.py
    ├── asgi.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py
```

1 directory, 6 files

- il folder esterno **mysite/** è il contenitore del progetto. Il nome di questo folder non è importante per django e può essere rinominato a proprio piacimento.
- **manage.py**: E' una command-line utility che permette di interagire con il progetto in diversi modi.
- il folder interno **mysite/** è il Package del progetto. il nome del folder è il nome del package.
- **mysite/__init__.py**: è un file vuoto che informa Python che questo folder deve essere considerato un package.
- **mysite/settings.py**: contiene la configurazione del progetto Django. Contiene anche tutte le informazioni necessarie per configurare i vari parametri.
- **mysite/urls.py**: Contiene le dichiarazioni di tutte le URL del progetto Django.
- **mysite/asgi.py**: è l'entry-point per rilasciare web server ASGI-compatible (**Asynchronous Server Gateway Interface**)
- **mysite/wsgi.py**: è l'entry-point per rilasciare web server WSGI-compatible (**Web Server Gateway Interface**)

I server ASGI e WSGI non saranno argomento di questo corso

Progetto e database

Ogni progetto si appoggia ad un database. Tutte le informazioni relative al database sono contenute nel file **mysite/settings.py**.

Come si vede il default è SQLite3.

Tutti i principali RDBMS sono supportati.

```
# Database
# https://docs.djangoproject.com/en/4.0/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

Avvio del server

Prima di avviare il server è necessario inizializzare il database.

```
$ python3 manage.py migrate
```

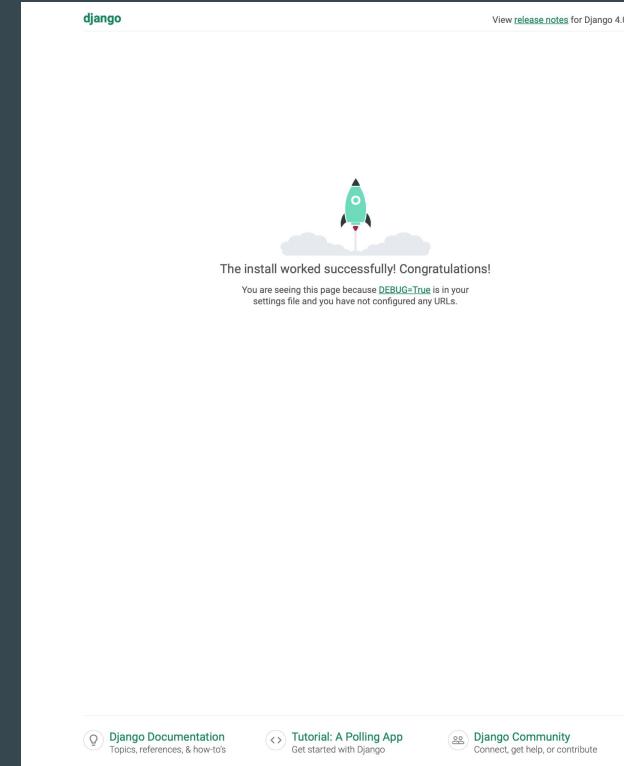
Questo creerà nel database le tabelle necessarie all'amministrazione del progetto.

A questo punto si può effettuare l'avvio del server:

```
$ python3 manage.py runserver
```

il progetto sarà disponibile all'indirizzo:

<http://127.0.0.1:8000>



Amministrazione

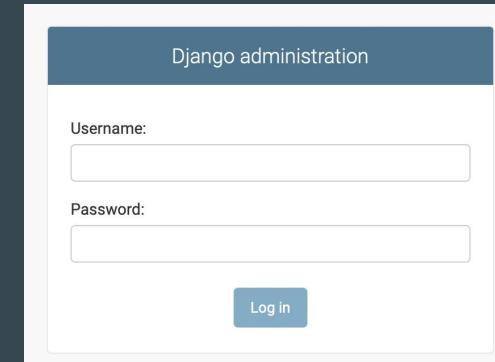
Di default il progetto dispone di una interfaccia di amministrazione, raggiungibile attraverso l'indirizzo

<http://127.0.0.1:8000/admin>

Il super user viene configurato tramite il comando:

```
$ python3 manage.py createsuperuser
```

```
~/Documents/Dottorato/PhDCourse2022/Examples/mysite on 2022-01-11 14:45:18
python3 manage.py createsuperuser
Username (leave blank to use 'romolo.politi'): Romolo.Politi
Email address: romolo.politi@inaf.it
Password:
Password (again):
Superuser created successfully.
```



Amministrazione

L'interfaccia di amministrazione permette di gestire gli utenti, i gruppi e le tabelle associate al progetto

The screenshot shows the Django administration interface. At the top, there's a header bar with the text "Django administration" on the left and "WELCOME, ROMOLO.POLTI. VIEW SITE / CHANGE PASSWORD / LOG OUT" on the right. Below the header, the main title "Site administration" is displayed. A navigation bar titled "AUTHENTICATION AND AUTHORIZATION" contains two items: "Groups" and "Users". Each item has a "+ Add" button and a "Change" link. To the right of the main content area, there are two sidebar boxes. The first sidebar box is titled "Recent actions" and "My actions", both of which currently show "None available".

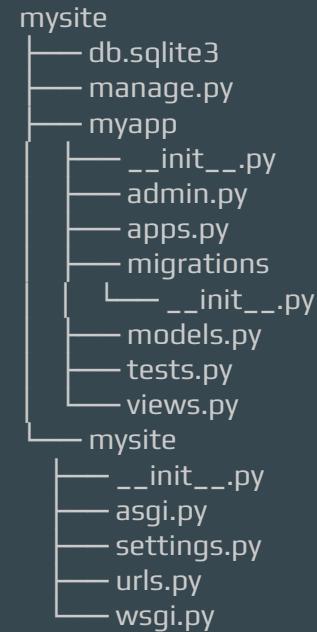
Applicazione django

Un progetto può contenere più applicazioni, che sono i singoli programmi che vengono esposti sul web.

Per creare un'applicazione bisogna eseguire il seguente comando:

```
$ python3 manage.py startapp myapp
```

Viene creata una cartella chiamata myapp con i file di configurazione della app



La prima pagina di myApp

Prima di costruire la prima pagina è necessario costruire l'url.

Creiamo un file **urls.py** nel cartella *myapp* ed inseriamo il seguente codice:

```
from django.urls import path
from myapp import views
urlpatterns = [
    path('', views.index, name='index'),
]
```

Che indica che quando richiamo l'url root (/) della app deve essere richiamata la funzione *index* del modulo **view** all'interno della app

La prima pagina di myApp

Nel modulo urls.py all'interno della cartella mysite importiamo il file contenente le url della mia app inserendo la seguente linea:

```
path('myapp/', include('myapp.urls')),
```

nell'array *urlpattern*.

Per ultimo inseriamo nel modulo myapp/views.py la funzione index:

```
from django.shortcuts import render
from django.http import HttpResponse

# Create your views here.

def index(request):
    var="Hello World"
    return HttpResponse(var)
```

La prima pagina di myApp

Il risultato sarà visibile alla pagina:

<http://127.0.0.1:8000/myapp/>

I Modelli

Prima di creare il modello registriamo la nostra app all'interno del file mysite/settings.py aggiungendo all'elenco delle app la nostra:

```
'myapp.apps.MyappConfig'
```

I modelli sono delle classi che verranno utilizzate da Django per creare delle tabelle.

I modelli sono delle classi contenute nel file *myapp/models.py*

In questo caso abbiamo creato un modello che contiene due campi carattere (nome e cognome) di lunghezza massima 100, un campo email e un generico campo testo.

```
class Example(models.Model):
    name = models.CharField(max_length=100)
    surname = models.CharField(max_length=100)
    email = models.EmailField()
    text = models.TextField()
```

I Modelli

Convertiamo la nostra classe in tabella. Per fare questo eseguiamo i comandi

```
$ python3 manage.py makemigrations
```

Migrations for 'myapp':

myapp/migrations/**0001**_initial.py

- Create model Example

```
$ python3 manage.py sqlmigrate myapp 0001
```



Crea un layer di astrazione

Converte layer in comandi per il Database scelto

sqlmigrate

```
BEGIN;

-- 

-- Create model Example

-- 

CREATE TABLE "myapp_example" ("id" integer NOT NULL PRIMARY
KEY AUTOINCREMENT, "name" varchar(100) NOT NULL, "surname"
varchar(100) NOT NULL, "email" varchar(254) NOT NULL, "text"
text NOT NULL);

COMMIT;
```

I Modelli

Infine inseriamo lo schema nella tabella tramite il comando

```
$ python3 manage.py migrate
```

Operations to perform:

Apply all migrations: admin, auth, contenttypes, myapp, sessions

Running migrations:

Applying myapp.0001_initial... OK

Applying sessions.0001_initial... OK

I Modelli

Ora il modello è nel database del nostro progetto.

Aggiungiamo il nome del nostro modello nel modulo di amministrazione myapp/admin.py aggiungendo la riga :

```
from .models import *
# Register your models here.

admin.site.register(Example)
```

ora la nostra tabella sarà sulla pagina di gestione del sito.

I Modelli

Django administration

WELCOME, ROMOLO.POLITI. [VIEW SITE](#) / [CHANGE PASSWORD](#) / [LOG OUT](#)

[Home](#) › [Myapp](#) › [Examples](#) › Add example

Start typing to filter...

AUTHENTICATION AND AUTHORIZATION

[Groups](#) [+ Add](#)

[Users](#) [+ Add](#)

MYAPP

[Examples](#) [+ Add](#)

Add example

Name:

Surname:

Email:

Text:

[Save and add another](#)

[Save and continue editing](#)

SAVE

I Modelli

Se inseriamo un elemento nell'elenco avremo:

Select example to change ADD EXAMPLE +

Action: Go 0 of 1 selected

<input type="checkbox"/>	EXAMPLE
<input type="checkbox"/>	Example object (1)

1 example

I Modelli

Per poter avere un elenco di oggetti più leggibile modifichiamo il modello, modificando il metodo `__str__`

```
class Example(models.Model):
    name = models.CharField(max_length=100)
    surname = models.CharField(max_length=100)
    email = models.EmailField()
    text = models.TextField()

    def __str__(self):
        return f'{self.name} {self.surname}'
```

I Modelli

Select example to change

ADD EXAMPLE +

Action: -----



Go

0 of 1 selected

EXAMPLE

Pippo Baudo

1 example

I Template

I template sono dei prototipi di pagina web che il framework elabora e riempie prima di esporli al web.

Per prima cosa creiamo le cartelle *templates/myapp* nella cartella *myapp*.

all'interno creiamo la nostra prima pagina web che sarà utilizzata come base per le altre

I Template

```
<!DOCTYPE html>
<html>

<head>
    <title>{{ title }}</title>
</head>
<body>
    {% block main %}
    {% endblock %}
</body>
</html>
```

Chiamiamo il file `_base.html`

I Template

```
<!DOCTYPE html>
<html>

<head>
    <title>{{ title }}</title>
</head>
<body>
    {% block main %}
    {% endblock %}
</body>
</html>
```

Chiamiamo il file `_base.html`

solitamente i file che iniziano con `_` sono
file interni

I Template

```
<!DOCTYPE html>
<html>

<head>
    <title>{{ title }}</title>
</head>
<body>
    {% block main %}
    {% endblock %}
</body>
</html>
```

Chiamiamo il file `_base.html`

`title` è il nome di una variabile e sarà sostituita con il suo valore in fase di render

I Template

```
<!DOCTYPE html>  
<html>  
  
<head>  
    <title>{{ title }}</title>  
</head>  
<body>  
    {% block main %}  
    {% endblock %}  
</body>  
</html>
```

Chiamiamo il file `_base.html`

`title` è il nome di una variabile e sarà sostituita con il suo valore in fase di render

Definisce un blocco che potrà essere riempito in seguito

I Template

sempre nella cartella template/myapp creiamo un file chiamato index.html

```
{% include 'myapp/_base.html' %}

{% block main %}

<div class="container">
    <div class="row">
        <div class="col-md-12">
            <h1>{{ title }}</h1>
        </div>
    </div>
{% endblock %}
```

includiamo il prototipo

riempiamo il blocco main

I Template

costruiamo ora la nostra view

```
def test(request):
    tmpl=loader.get_template('myapp/index.html')
    context={
        'title': 'My First title',
    }
    return HttpResponse(tmpl.render(context,request))
```

e il relativo url

```
path('test/', views.test, name='test'),
```

Visualizzazione di Modelli

Creiamo una view chiamata **list**:

```
def list(request):
    tmpl=loader.get_template('myapp/list.html')
    dat=Example.objects.order_by('surname')[:]
    context={
        'title': 'Data list',
        'data':dat,
    }
    return HttpResponse(tmpl.render(context,request))
```

Visualizzazione di Modelli

Creiamo una view chiamata **list**:

```
def list(request):
    tmpl=loader.get_template('myapp/list.html')
    dat=Example.objects.order_by('surname')[:]
    context={
        'title': 'Data list',
        'data':dat,
    }
    return HttpResponseRedirect(tmpl.render(context,request))
```

leggiamo un template chiamato list.html

Visualizzazione di Modelli

Creiamo una view chiamata **list**:

```
def list(request):
    tmpl=loader.get_template('myapp/list.html')
    dat=Example.objects.order_by('surname')[:]
    context={
        'title': 'Data list',
        'data':dat,
    }
    return HttpResponseRedirect(tmpl.render(context,request))
```

leggiamo un tamplate chiamato list.html

selezioniamo tutti gli oggetti nella classe Example ordinati per cognome

Visualizzazione di Modelli

Creiamo una view chiamata **list**:

```
def list(request):
    tmpl=loader.get_template('myapp/list.html')
    dat=Example.objects.order_by('surname')[:]
    context={
        'title': 'Data list',
        'data':dat,
    }
    return HttpResponseRedirect(tmpl.render(context,request))
```

leggiamo un template chiamato list.html

selezioniamo tutti gli oggetti nella classe Example ordinati per cognome

facciamo il render del template ed esponiamo il tutto

Visualizzazione di Modelli

Costruiamo il template list.html

```
{% include 'myapp/_base.html' %}  
{% block main %}  
<div class="container">  
    <div class="row">  
        <div class="col-md-12">  
            <table>  
                <thead>  
                    <tr> <th>Name</th> <th> Surname</th> <th>Email</th> </tr>  
                </thead>  
                <tbody>  
                    {% for item in data %}  
                        <tr> <td>{{ item.name }}</td> <td>{{ item.surname }}</td> <td>{{ item.email }}</td> </tr>  
                    {% endfor %}  
                </tbody>  
            </table>  
        </div>  
    </div>  
</div>  
{% endblock %}
```

Visualizzazione di modelli

Si otterrà

Name Surname	Email
Pippo Baudo	pippo.baudo@gmail.com
Bruno Vespa	bruno.vespa@gmail.com

Template Avanzati

Template avanzati

Creiamo una nuova app chiamata myapp2

```
$ python3 manage.py startapp myapp2
```

registriamo l'app in *mysite/settings.py*, aggiungendo la riga:

```
'myapp2.apps.Myapp2Config',
```

nella lista delle app installate.

Registriamo gli url aggiungendo a *mysite/urls.py* la riga

```
path('myapp2/', include('myapp2.urls')),  
path('', include('myapp2.urls')),
```

l'ultima riga permette di indirizzare alla app quando si va alla home del progetto.

Template avanzati

Creiamo un folder per i template: *templates/myapp2*

Creiamo due template interni:

_base.html

_navbar.html

_base.html

```
<!DOCTYPE html>
<html>

<head>
    <title>{{ title }}</title>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0-beta1/dist/css/bootstrap.min.css">
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0-beta1/dist/js/bootstrap.bundle.min.js"></script>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
    {% block script %}
    {% endblock %}
</head>

<body>
    {% include 'myapp2/_navbar.html' %}
    <div class="container">
        <div class="row">
            <div class="col-md-2"></div>
            <div class="col-md-8">
                {% block content %}{% endblock %}
            </div>
            <div class="col-md-2"></div>
        </div>
    </div>
</body>

</html>
```

_base.html

```
<!DOCTYPE html>
<html>

<head>
    <title>{{ title }}</title>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0-beta1/dist/css/bootstrap.min.css">
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0-beta1/dist/js/bootstrap.bundle.min.js"></script>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
    {% block script %}
    {% endblock %}
</head>

<body>
    {% include 'myapp2/_navbar.html' %}
    <div class="container">
        <div class="row">
            <div class="col-md-2"></div>
            <div class="col-md-8">
                {% block content %}{% endblock %}
            </div>
            <div class="col-md-2"></div>
        </div>
    </div>
</body>
</html>
```

Inseriamo:

- una variabile per il titolo
- il CDN per i CSS di Bootstrap
- il CDN per il JS di Bootstrap
- il CDN per jQuery
- un blocco per lo script
- includiamo la barra di navigazione
- un blocco per il contenuto

_base.html

```
<!DOCTYPE html>
<html>

<head>
    <title>{{ title }}</title>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0-beta1/dist/css/bootstrap.min.css">
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0-beta1/dist/js/bootstrap.bundle.min.js"></script>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
    {% block script %}
    {% endblock %}
</head>

<body>
    {% include 'myapp2/_navbar.html' %}
    <div class="container">
        <div class="row">
            <div class="col-md-2"></div>
            <div class="col-md-8">
                {% block content %}{% endblock %}
            </div>
            <div class="col-md-2"></div>
        </div>
    </div>
</body>
</html>
```

Da notare che è stato utilizzato il layout a colonne di Bootstrap.

Inseriamo:

- una variabile per il titolo
- il CDN per i CSS di Bootstrap
- il CDN per il JS di Bootstrap
- il CDN per jQuery
- un blocco per lo script
- includiamo la barra di navigazione
- un blocco per il contenuto

_navbar.html

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <div class="container-fluid">
    <a class="navbar-brand" href="#">Navbar</a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarSupportedContent">
      <ul class="navbar-nav me-auto mb-2 mb-lg-0">
        <li class="nav-item">
          <a class="nav-link active" aria-current="page" href="{% url 'home' %}">Home</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="{% url 'plist' %}">Table</a>
        </li>
        <li class="nav-item dropdown">
          <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button" data-bs-toggle="dropdown" aria-expanded="false">
            Dropdown
          </a>
          <ul class="dropdown-menu" aria-labelledby="navbarDropdown">
            <li><a class="dropdown-item" href="/admin">Admin</a></li>
            <li><a class="dropdown-item" href="#">Another action</a></li>
            <li><hr class="dropdown-divider"></li>
            <li><a class="dropdown-item" href="#">Something else here</a></li>
          </ul>
        </li>
        <li class="nav-item">
          <a class="nav-link disabled" href="#" tabindex="-1" aria-disabled="true">Disabled</a>
        </li>
      </ul>
      <form class="d-flex">
        <input class="form-control me-2" type="search" placeholder="Search" aria-label="Search">
        <button class="btn btn-outline-success" type="submit">Search</button>
      </form>
    </div>
  </div>
</nav>
```

_navbar.html

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <div class="container-fluid">
    <a class="navbar-brand" href="#">Navbar</a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarSupportedContent">
      <ul class="navbar-nav me-auto mb-2 mb-lg-0">
        <li class="nav-item">
          <a class="nav-link active" aria-current="page" href="{% url 'home' %}">Home</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="{% url 'plist' %}">Table</a>
        </li>
        <li class="nav-item dropdown">
          <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button" data-bs-toggle="dropdown" aria-expanded="false">
            Dropdown
          </a>
          <ul class="dropdown-menu" aria-labelledby="navbarDropdown">
            <li><a class="dropdown-item" href="/admin">Admin</a></li>
            <li><a class="dropdown-item" href="#">Another action</a></li>
            <li><hr class="dropdown-divider"></li>
            <li><a class="dropdown-item" href="#">Something else here</a></li>
          </ul>
        </li>
        <li class="nav-item">
          <a class="nav-link disabled" href="#" tabindex="-1" aria-disabled="true">Disabled</a>
        </li>
      </ul>
      <form class="d-flex">
        <input class="form-control me-2" type="search" placeholder="Search" aria-label="Search">
        <button class="btn btn-outline-success" type="submit">Search</button>
      </form>
    </div>
  </div>
</nav>
```

Si tratta di un template di barra di navigazione presa dagli esempi di Bootstrap ed adattata alle nostre esigenze.

Viene utilizzata come esempio, non tutti i link sono funzionanti

Home Page

Creiamo la home page:

vista (*myapp2/views.py*)

```
def home(request):
    context={
        'title': 'My app2',
    }
    return render(request, 'myapp2/index.html', context)
```

Url (*myapp2/urls.py*)

```
path('', views.home, name='home'),
```

Template (*myapp2/templates/myapp2/index.html*)

```
{% include 'myapp2/_base.html' %}
{% block content %}
    <h1 class="text-center">Welcome to MyApp2</h1>
{% endblock %}
```

Home Page

Navbar Home Table Dropdown ▾ Disabled

Search

Search

Welcome to MyApp2

Creiamo un modello

Creiamo un modello Example costruendo la classe nel file myapp2/models.py

```
class Example(models.Model):
    name = models.CharField(max_length=100)
    surname = models.CharField(max_length=100)
    email = models.EmailField()
    text = models.TextField()

    def __str__(self):
        return f'{self.name} {self.surname}'
```

e registriamolo nella pagina di amministrazione, aggiungendo nel file myapp2/admin.py la riga

```
admin.site.register(Example)
```

Creiamo un modello

Creiamo un modello Example costruendo la classe nel file myapp2/models.py

```
class Example(models.Model):
    name = models.CharField(max_length=100)
    surname = models.CharField(max_length=100)
    email = models.EmailField()
    text = models.TextField()

    def __str__(self):
        return f"{self.name} {self.surname}"
```

e registriamolo nella pagina di amministrazione, aggiungendo nel file myapp2/admin.py la riga

```
admin.site.register(Example)
```

Ricordiamoci che dopo aver creato il modello è necessario eseguire le operazioni di

- makemigrations
- sqlmigrate
- migrate

Modello di Esempio

Controlliamo il modello dal sito di amministrazione (ricordiamoci, se non è stato fatto, di creare il superuser come mostrato nella lezione precedente)

The screenshot shows the Django Admin interface with the following structure:

- Site administration** header.
- AUTHENTICATION AND AUTHORIZATION** section:
 - Groups**: + Add, Change
 - Users**: + Add, Change
- MYAPP** section:
 - Examples**: + Add, Change
- MYAPP2** section:
 - Examples**: + Add, Change
- Recent actions** sidebar:
 - My actions**
 - + pippo Baudo Example
 - + Bruno Vespa Example
 - + Example object (1) Example

Modello di Esempio

Inseriamo un elemento nel modello

The screenshot shows a user interface for managing a "MYAPP" application. On the left, there is a sidebar with a search bar and several sections: "AUTHENTICATION AND AUTHORIZATION" (Groups, Users), "MYAPP" (Examples), and "MYAPP2" (Examples). The "Examples" section under "MYAPP2" is highlighted with a yellow background. The main content area is titled "Add example" and contains fields for Name, Surname, Email, and Text, each with an associated input field. At the bottom, there are three buttons: "Save and add another", "Save and continue editing", and a large blue "SAVE" button.

Start typing to filter...

AUTHENTICATION AND AUTHORIZATION

Groups [+ Add](#)

Users [+ Add](#)

MYAPP

Examples [+ Add](#)

MYAPP2

Examples [+ Add](#)

Add example

Name:

Surname:

Email:

Text:

Save and add another

Save and continue editing

SAVE

Lista degli elementi

Per visualizzare la lista degli elementi creiamo prima di tutto la view:

```
def plist(request):
    dat=Example.objects.order_by('surname')
    context={
        'title': 'Data list',
        'data':dat,
    }
    return render(request, 'myapp2/list.html',context)
```

In cui selezioniamo tutti gli elementi presenti nel modello **Example** ordinati secondo il campo **surname** ed esportiamo questa lista nel template **list.html**

il Template list

```
{% include 'myapp2/_base.html' %}  
{% block content %}  
<h1 class="text-center">Lista</h1>  
<table class="table table-hover">  
    <thead>  
        <tr><th>Nome</th><th>Email</th></tr>  
    </thead>  
    <tbody>  
        {% for item in data %}  
            <tr>  
                <td><a href="{% url 'scheda2' item.pk %}">{{ item.name }} {{ item.surname }}</a></td>  
                <td>{{ item.email }}</td>  
  
            </tr>  
        {% endfor %}  
    </tbody>  
</table>  
{% endblock %}
```

il Template list

```
{% include 'myapp2/_base.html' %}  
{% block content %}  
<h1 class="text-center">Lista</h1>  
<table class="table table-hover">  
  <thead>  
    <tr><th>Nome</th><th>Email</th></tr>  
  </thead>  
  <tbody>  
    {% for item in data %}  
      <tr>
```

Navbar Home Table Dropdown ▾ Disabled

Search

Search

Lista

Nome	Email
Pippo Baudo	pippo.baudo@gmail.com
Emilio Solfrizzi	emilio.solfrizzi@gmail.com
bruno vespa	bruno.vespa@gmail.com

Scheda

View

```
def scheda(request,id):
    dat=get_object_or_404(Example,id=id)
    context={
        'title': 'Scheda',
        'data':dat,
    }
    return render(request, 'myapp2/scheda.html',context)
```

Template

```
{% include 'myapp2/_base.html' %}
{% block content %}
<div class="card" style="width: 18rem;">
    <div class="card-body">
        <h5 class="card-title">{{ data.name }} {{ data.surname }}</h5>
        <h6 class="card-subtitle mb-2 text-muted"> {{ data.email }}</h6>
        <p class="card-text">{{ data.text }}</p>
    </div>
</div>
{% endblock %}
```

Scheda

View

```
def scheda(request,id):
    dat=get_object_or_404(Example,id=id)
    context={
        'title': 'Scheda',
        'data':dat
```

Navbar Home Table Dropdown ▾ Disabled

Search

Search

Pippo Baudo

pippo.baudo@gmail.com

Test

```
<div class="card-body">
    <h5 class="card-title">{{ data.name }} {{ data.surname }}</h5>
    <h6 class="card-subtitle mb-2 text-muted"> {{ data.email }}</h6>
    <p class="card-text">{{ data.text }}</p>
</div>
</div>
{%
    endblock %}
```

I form

Vogliamo aggiungere un nuovo elemento alla nostra lista.

Per fare questo dobbiamo costruire una **form**.

Creiamo un file **forms.py** in myapp2

```
class ExampleForm(ModelForm):
    class Meta:
        model = Example
        fields=['name','surname','email','text']
```

e creiamo un template chiamato form.html

```
{% include 'myapp2/_base.html' %}
{% block content %}
<h1 class="text-center">New Entry</h1>
<form action="{{ action }}" method="post">
{% csrf_token %}
<fieldset>
    {{ formset }}
</fieldset>
<input type="submit" role="button" class="btn btn-primary" value="{{ label }}>
</form>
{% endblock %}
```

La view in questo caso è

```
def add(request):
    if request.method=='POST':
        form = ExampleForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect(reverse('plist'))
    else:
        form = ExampleForm()
    context={
        'title': 'Add data',
        'formset': form,
        'label': 'Add',
        'action': '/myapp2/list/add/',
    }
    return render(request, 'myapp2/form.html',context)
```

I form

Navbar Home Table Dropdown ▾ Disabled

Search

Search

New Entry

Name: Surname: Email: Text:

Add

```
{% include 'myapp2/base.html'%}
```

```
{% block content %}
```

```
<h1 class="text-center">New Entr</h1>
```

I form

Per ottenere un aspetto più gradevole costruiamo un template per il form:

```
<div class="row mb-3 align-bottom">
  <div class="col">
    <span style="color:red;">{{ form.name.errors }}</span>
    <input type="text" class="form-control" placeholder="{{ form.name.label }}" aria-label="{{ form.name.label }}"
      value="{{ form.name.value| default:'' }}" name="{{ form.name.html_name }}>
  </div>
  <div class="col">
    <span style="color:red;">{{ form.surname.errors }}</span>
    <input type="text" class="form-control" placeholder="{{ form.surname.label }}" aria-label="{{ form.surname.label }}"
      value="{{ form.surname.value| default:'' }}" name="{{ form.surname.html_name }}>
  </div>
</div>
<div>
  <div class="mb-3">
    <span style="color:red;">{{ form.email.errors }}</span>
    <input type="email" class="form-control" placeholder="{{ form.email.label }}" aria-label="{{ form.email.label }}"
      value="{{ form.email.value| default:'' }}" name="{{ form.email.html_name }}>
  </div>
  <div class="mb-3">
    <span style="color:red;">{{ form.text.errors }}</span>
    <textarea class="form-control" id="exampleFormControlTextarea1" rows="3"
      placeholder="{{ form.text.label }}" aria-label="{{ form.text.label }}"
      name="{{ form.text.html_name }}>{{ form.text.value| default:'' }}</textarea>
  </div>
</div>
```

I form

registriamo il template aggiungendolo alla definizione del form:

```
class ExampleForm(ModelForm):  
  
    template_name = 'myapp2/form_template.html'  
  
    class Meta:  
  
        model = Example  
  
        fields=['name','surname','email','text']
```

I form

registriamo il template aggiungendolo alla definizione del form:

The screenshot shows a web application interface with a navigation bar at the top. The navigation bar includes links for 'Navbar', 'Home', 'Table', 'Dropdown ▾', and 'Disabled'. To the right of the navigation bar is a search bar with a placeholder 'Search' and a green 'Search' button. Below the navigation bar, the main content area has a title 'New Entry' centered above a form. The form consists of several input fields: 'Name' (text input), 'Surname' (text input), 'Email' (text input), and 'Text' (text area). At the bottom left of the form is a blue 'Add' button.

Navbar Home Table Dropdown ▾ Disabled

Search

Search

New Entry

Name

Surname

Email

Text

Add

I form

possiamo abbellire la nostra form inserendo due altri pulsanti

```
<a href="{% url 'add' %}" role="button" class="btn btn-danger">Reset</a>  
  
<a href="{% url 'plist' %}" role="button" class="btn btn-secondary">Cancel</a>
```

Navbar Home Table Dropdown ▾ Disabled

Search

Search

New Entry

Name

Surname

Email

Text

Add

Reset

Cancel

La lista

possiamo aggiungere Funzionalità alla lista tipo Edit e Delete aggiungendo due pulsanti

```
<div class="btn-group" role="group" aria-label="Basic example">
  <a href="{% url 'edit' item.pk %}" role="button" class="btn btn-primary"><svg xmlns="http://www.w3.org/2000/svg" width="16" height="16" fill="currentColor" class="bi bi-pencil-fill" viewBox="0 0 16 16">...</svg> Edit</a>
  <a href="{% url 'delete' item.pk %}" role="button" class="btn btn-danger confirmation"><svg xmlns="http://www.w3.org/2000/svg" width="16" height="16" fill="currentColor" class="bi bi-trash" viewBox="0 0 16 16" >...</svg> Delete</a>
</div>
```

Nome	Email	Tools
Pippo Baudo	pippo.baudo@gmail.com	 Edit  Delete
Emilio Solfrizzi	emilio.solfrizzi@gmail.com	 Edit  Delete
bruno vespa	bruno.vespa@gmail.com	 Edit  Delete
		Add

La lista

con relative view

```
def edit(request,id):
    if request.method == 'POST':
        form = ExampleForm(
            request.POST, instance=get_object_or_404(Example, id=id))
        if form.is_valid():
            form.save()
        return redirect(reverse('plist'))
    else:
        form = ExampleForm(instance=get_object_or_404(Example,id=id))

    context={
        'title': 'Add data',
        'formset': form,
        'label': 'Update',
        'action': f'/myapp2/list/edit/{id}/',
    }
    # context.update(csrf(request))

    return render(request, 'myapp2/form.html',context)

def delete(request,id):
    dat=get_object_or_404(Example,id=id)
    dat.delete()
    return redirect(reverse('plist'))
```

La lista

Possiamo aggiungere uno script js per chiedere la conferma prima della cancellazione

```
{% block script %}  
<script type="text/javascript">  
    $(document).ready(function() {  
        $('.confirmation').on('click', function () {  
            return confirm('Are you sure?');  
        });  
    } );  
  
</script>  
{% endblock %}
```

Relazione tra Modelli

Creiamo un nuovo modello **Lavoro**, e colleghiamo questo al modello **Example**:

```
# Create your models here.
class Lavoro(models.Model):
    lavoro= models.CharField(max_length=100)
    def __str__(self) -> str:
        return self.lavoro

class Example(models.Model):
    name = models.CharField(max_length=100)
    surname = models.CharField(max_length=100)
    email = models.EmailField()
    text = models.TextField()
    lavoro = models.ForeignKey(Lavoro, on_delete=models.CASCADE)

    def __str__(self):
        return f"{self.name} {self.surname}"
```

ForeignKey è la chiave esterna che permette di collegare i due modelli. Nel caso in cui l'elemento collegato venga cancellato dal modello **Lavoro**, anche l'elemento in **Example** verrà cancellato (`on_delete = models.CASCADE`)

Relazione tra Modelli

Se a questo punto cerchiamo di effettuare la migrazione otterremo un errore perché non si riesce ad associare nessun valore al campo *lavoro* del modello **Example**, poiché il modello **Lavoro** è vuoto.

Questo problema si verifica perché non è stato creato un design del database prima della creazione. Cosa fare ora?

L'operazione più semplice è effettuare un reset della base dati, cancellando sia il file del db (db.sqlite3) che la cartella con le migrations (myapp2/migrations).

Per evitare di dover fare manualmente il data entry facciamo il backup del database.

```
$ python3 manage.py dumpdata myapp2.Example -o myapp2/fixtures/example.json
```

dove **myapp2.Example** è il modello da salvare e con l'opzione -o si indica il file di output.

Relazione tra Modelli

una volta effettuato il dump del database e cancellati i file possiamo rigenerare il db

```
$ python3 manage.py migrate
```

Rigeneriamo tutte le tabelle di base del progetto

```
$ python3 manage.py createsuperuser
```

Ricreiamo il superuser

```
$ python3 manage.py makemigrations myapp2
```

Convertiamo i modelli in metalinguaggio

```
$ python3 manage.py sqlmigrate myapp2 0001
```

Convertiamo il metalinguaggio in SQL

```
$ python3 manage.py migrate myapp2
```

Inseriamo le strutture delle tabelle nel db

Relazione tra Modelli

Prima di caricare nel sistema il dump fatto in precedenza, è necessaria effettuare delle correzioni alla base dati:

1. inserire un valore per il modello **Lavoro** nel file *myapp2/fixtures/lavoro.json*:

```
{  
    "model": "myapp2.Lavoro",  
    "pk": 1,  
    "fields": {  
        "lavoro": "Attore"  
    }  
},
```

2. inserire il campo lavoro agli elementi di **Example**, ad esempio:

```
{  
    "model": "myapp2.example",  
    "pk": 1,  
    "fields": {  
        "name": "Pippo",  
        "surname": "Baudo",  
        "email": "pippo.baudo@gmail.com",  
        "text": "Test",  
        "lavoro": 1  
    }  
},
```

Relazione tra Modelli

Per aggiungere il campo *lavoro* possiamo utilizzare il seguente script:

```
import json

data=json.load(open('example.json'))
for item in data:
    item['fields']['lavoro']=1
json.dump(data,open('example_cor.json','w'))
```

Ora basterà effettuare il data entry con i comandi:

```
$ python3 manage.py loaddata myapp2/fixtures/lavoro.json
```

```
$ python3 manage.py loaddata myapp2/fixtures/example.json
```

Effettuiamo il data entry manualmente per essere sicuri che venga popolato prima il modello **Lavoro** per evitare errori.

Relazione tra Modelli

Se al template list.html, aggiungiamo la colonna lavoro

```
<td>{{ item.lavoro }}</td>
```

otterremo:

Nome	Email	Lavoro	Tools
Pippo Baudo	Attore	pippo.baudo@gmail.com	 Edit Delete
Emilio Solfrizzi	Attore	emilio.solfrizzi@gmail.com	 Edit Delete
bruno vespa	Attore	bruno.vespa@gmail.com	 Edit Delete
			 Add

Relazione tra Modelli

A questo punto possiamo modificare il form:

```
class ExampleForm(ModelForm):
    template_name = 'myapp2/form_template.html'
    class Meta:
        model = Example
        fields=['name','surname','email','text','lavoro']
        widgets = {
            'email': EmailInput(attrs={
                'placeholder': 'Email',
                'aria-label': 'Email',
                'class': 'form-control'
            }),
            'lavoro': Select(attrs={
                'class': 'form-select',
                'aria-label': 'Seleziona il lavoro',
            })
        }
```

Relazione tra Modelli

ed il relativo template:

```
<div class="row mb-3 align-bottom">
  <div class="col">
    <span style="color:red;">{{ form.name.errors }}</span>
    <input type="text" class="form-control" placeholder="{{ form.name.label }}" aria-label="{{ form.name.label }}"
           value="{{ form.name.value| default:'' }}" name="{{ form.html_name }}>
  </div>
  <div class="col">
    <span style="color:red;">{{ form.surname.errors }}</span>
    <input type="text" class="form-control" placeholder="{{ form.surname.label }}" aria-label="{{ form.surname.label }}"
           value="{{ form.surname.value| default:'' }}" name="{{ form.surname.html_name }}>
  </div>
</div>
<div>
  <div class="mb-3">
    <span style="color:red;">{{ form.email.errors }}</span>
    {{ form.email }}
  </div>
  <div class="mb-3">
    <span style="color:red;">{{ form.text.errors }}</span>
    <textarea class="form-control" id="exampleFormControlTextarea1" rows="3"
              placeholder="{{ form.text.label }}" aria-label="{{ form.text.label }}"
              name="{{ form.text.html_name }}>{{ form.text.value| default:'' }}</textarea>
  </div>
  <div class="mb-3">
    {{ form.lavoro }}
  </div>
</div>
```

Relazione tra Modelli

ed otterremo:

Modify Entry

bruno

vespa

bruno.vespa@gmail.com

questa è una nota

Giornalista

Update

Reset

Cancel

Nessuna modifica è richiesta a livello di view.

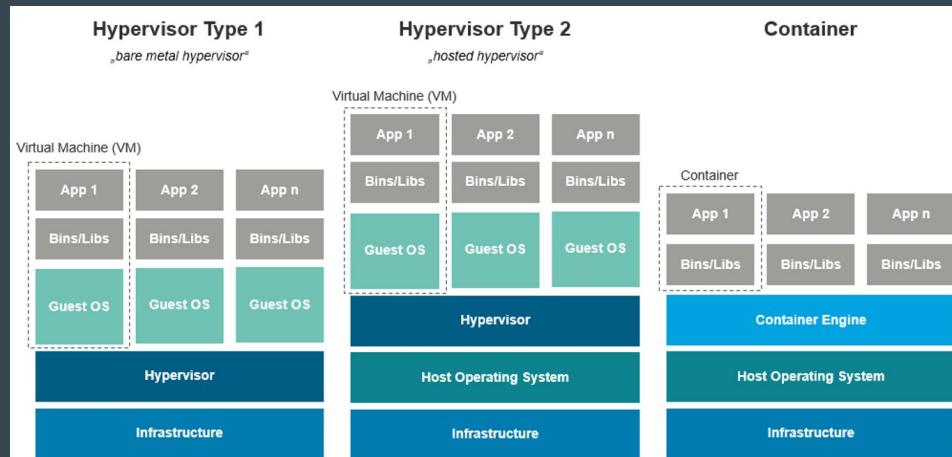
Docker

Docker

Docker è una piattaforma software che permette di creare, testare e distribuire applicazioni con la massima rapidità.

Docker raccoglie il software in unità standardizzate chiamate **container** che offrono tutto il necessario per la loro corretta esecuzione, incluse librerie, strumenti di sistema, codice e runtime.

Con Docker, è possibile distribuire e ricalibrare le risorse per un'applicazione in qualsiasi ambiente, tenendo sempre sotto controllo il codice eseguito.



Glossario

- Una **Immagine Docker** è un **file immutabile** che contiene il codice sorgente, le librerie, le dipendenze, i tool ed ogni altro file necessario ad eseguire la nostra applicazione.
- Un **Container Docker** è un **ambiente runtime virtualizzato** dove gli utenti possono isolare le applicazioni dal proprio sistema operativo. I container sono compatti portabili e permettono di eseguire le applicazioni velocemente e con semplicità.

Docker Desktop

il metodo più veloce per installare Docker è l'installazione di Docker Desktop:

<https://www.docker.com/products/docker-desktop/>

Nella pagina è presente anche il link alla DockerCon 2022 in cui c'è una interessante lezione introduttiva a Docker.

Una volta installato posso andare sul sito DockerHub in cui sono presenti tutte le immagini utilizzabili da Docker.

<https://hub.docker.com/>

Dal prompt lanciamo il comando:

```
$ docker pull python:3.11.4-alpine
```

in questo caso verrà scaricata un'immagine docker con python che viene eseguito su un sistema linux **alpine** minimale.

Docker Desktop

The screenshot shows the Docker Desktop application window. The top bar includes the Docker Desktop logo, a 'Update to latest' button, a search bar, and user account settings. The left sidebar has navigation links for Containers, Images (which is selected), Volumes, Dev Environments (BETA), and Learning Center. Below this is an 'Extensions' section with an 'Add Extensions' button. The main content area is titled 'Images' with a 'Local' tab selected, showing 5 images. A summary at the top indicates 206.62 MB / 3.46 GB in use across 5 images, last refreshed 3 minutes ago. The image list table has columns for Name, Tag, Status, Created, Size, and Actions. The five images listed are:

Name	Tag	Status	Created	Size	Actions
python	3.11.4	Unused	20 days ago	1 GB	...
python	3.11.4-alpine	Unused	26 days ago	51.98 MB	...
redhat/ubi8	latest	In_use	1 year ago	206.62 MB	...
djaccess_web	latest	Unused	1 year ago	2.88 GB	...
postgres	latest	Unused	1 year ago	376.09 MB	...

At the bottom, resource usage is shown as RAM 3.87 GB, CPU 60.55%, Disk 47.79 GB avail. of 62.67 GB, and a note that the user is Not connected to Hub. The footer also shows version v4.20.1.

Docker Desktop

è possibile effettuare la ricerca e l'installazione direttamente da Docker Desktop

The screenshot shows the Docker Desktop application window. At the top, there's a blue header bar with the Docker Desktop logo, an 'Update to latest' button, and a search bar containing the placeholder text 'Search for local and remote images, containers, and more...'. A green arrow points from the explanatory text above to the search bar. Below the header is a sidebar with icons for Containers, Images (which is selected), Volumes, Dev Environments (BETA), and Learning Center. The main area is titled 'Images' with a 'Local' tab selected, showing 5 images used 206.62 MB / 3.46 GB in total. The images listed are:

Name	Tag	Status	Created	Size	Actions
python	3.11.4	Unused	20 days ago	1 GB	▶ ⋮ 🗑
python	3.11.4-alpine	Unused	26 days ago	51.98 MB	▶ ⋮ 🗑
redhat/ubi8	latest	In use	1 year ago	206.62 MB	▶ ⋮ 🗑
djaccess_web	latest	Unused	1 year ago	2.88 GB	▶ ⋮ 🗑
postgres	latest	Unused	1 year ago	376.09 MB	▶ ⋮ 🗑

At the bottom, there's a footer with system status: RAM 3.87 GB, CPU 60.55%, Disk 47.79 GB avail. of 62.67 GB, and a note 'Not connected to Hub'. The footer also includes version information: v4.20.1.

Docker e VS Code

Installiamo il plugin Docker per Visual Studio.

Colleghiamo un terminale al container.

Siamo in grado di eseguire tutti i programmi python all'interno del nostro container.

Una procedura più corretta sarebbe creare un utente all'interno del container ed eseguire i programmi come utente.

Ciò deve essere fatto in produzione.

In fase di sviluppo, dove la sicurezza è secondaria, si può trascurare questo passaggio.

Dockerfile

Possiamo creare la nostra immagine utilizzando uno script che permette di personalizzare l'ambiente di lavoro. Questo script è chiamato Dockerfile.

```
FROM ubuntu:latest
RUN apt-get update && apt-get install -y python3 python3-pip python3-dev
WORKDIR /flask
COPY ./requirements.txt /flask/requirements.txt
COPY ./myapp.py /flask/myapp.py
RUN pip install -r requirements.txt
ENTRYPOINT [ "python3" ]
CMD [ "myapp.py" ]
```

Per generare la nostra immagine eseguiremo il comando:

```
$ docker build -t flask-test:0.1.0 .
```

Dockerfile

Per creare un container da un'immagine useremo il comando:

```
$ docker run -d -p 5010:5010 flask-test:0.1.0
```

dove

-d (detach) facciamo eseguire il container in background

-p indichiamo quale porta dell'host deve essere collegata alla porta del container
ed infine il nome e tag dell'immagine

Flask

micro framework

Flask

Flask è un **micro framework non full stack**.

"Micro" non significa che l'intera applicazione Web è contenuta in un singolo file Python (anche se ciò è possibile), né significa che Flask manchi di funzionalità.

Il "micro" in micro framework significa che Flask mira a mantenere un *core* semplice ma estensibile.

Flask non prenderà molte decisioni al posto del programmatore, ad esempio quale database utilizzare.

Le decisioni che prende, ad esempio quale motore di creazione di modelli utilizzare, sono facili da modificare.

Tutto il resto dipende dal programmatore “in modo che Flask possa essere tutto ciò di cui hai bisogno e niente di cui non hai bisogno”.

Installazione

Avendo creato ambienti virtuali o container Python (soluzione consigliata) il comando per installare Flask è

```
$ python3 -m pip install Flask
```

Primo esempio

Nella cartella **Examples/Flask** è riportata una semplice applicazione auto contenuta.

In questo caso la nostra applicazione è un unico file.

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def hello_world():
    print("hello world")
    return "<p>Hello, World!</p>"

if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=5010)
```

Secondo Esempio

Il secondo esempio nella cartella *Examples/Flask2* è la riproduzione dell'applicazione sviluppata in django.

In primo luogo si vede che il database deve essere creato con le istruzioni SQL (file **schema.sql**) ed inizializzato con il comando

```
$ flask init-db (ricordiamoci di esportare la variabile d'ambiente con il nome  
dell'app)
```

In secondo luogo tutte le interazioni con il database sono mediate dal modulo db.py e le query devono essere scritte in linguaggio SQL.

Secondo Esempio

Le differenze con django :

- gli url che vengono forniti tramite un decoratore.
- non esiste un layer di astrazione del db
- non esiste un layer di astrazione per i forum
- I template sono in formato Jinja2
- non esiste un sistema di amministrazione
- Non esistono richieste asincrone

click

click

Click è un pacchetto Python per creare interfacce a riga di comando in modo componibile con il minimo codice necessario. È il "Kit di creazione dell'interfaccia della riga di comando". È altamente configurabile ma viene fornito con impostazioni predefinite sensibili pronte all'uso.

Click in tre punti:

- nidificazione arbitraria dei comandi
- generazione automatica della pagina di aiuto
- supporta il caricamento lento dei sottocomandi in fase di esecuzione

click

Click è un pacchetto Python per creare interfacce a riga di comando in modo componibile con il minimo codice necessario. È il "Kit di creazione dell'interfaccia della riga di comando". È altamente configurabile ma viene fornito con impostazioni predefinite sensibili pronte all'uso.

Noi utilizzeremo una variante che si integra con rich, rich-click

Click in tre punti:

- nidificazione arbitraria dei comandi
- generazione automatica della pagina di aiuto
- supporta il caricamento lento dei sottocomandi in fase di esecuzione

click - Esempio

```
import click

@click.command()
@click.option('--count', default=1, help='Number of greetings.')
@click.option('--name', prompt='Your name', help='The person to greet.')
def hello(count, name):
    """Simple program that greets NAME for a total of COUNT times."""
    for x in range(count):
        click.echo(f"Hello {name}!")

if __name__ == '__main__':
    hello()
```

click - Elementi

In click possiamo distinguere tre tipi di elementi

- Opzioni
- Argomenti
- Comandi / gruppi di comandi

click - Opzioni

Definite dal decoratore `@click.option()`. I principali parametri sono (Examples/10 - click/example3.py):

- **nome**, sequenza di stringhe che definisce il nome dell'opzione e.g. “-f”, “--foo”. La variabile prende il nome dall'opzione con --. In alternativa si può inserire una terza voce con il nome della variabile.
- **type**. Si può specificare il tipo di variabile per fare automaticamente dei test di validità.
- **default**, valore di default da assegnare alla variabile
- **help**, testo da mostrare nell'help

click - Opzioni

- **is_flag**, questa variabile è un bool ed è impostata di default in False
- **count**, booleano e nella variabile viene indicato il numero di volte in cui viene usata l'opzione
- **callback**, se l'opzione è presente viene eseguita una funzione.

click - Argomenti

Gli argomenti sono molto simili alle opzioni, ma sono posizionali.

Gli argomenti possono essere contrassegnati come obbligatori o non obbligatori dall'opzione **required**.

click - Comandi e Gruppi

la più importante funzione di click è il concetto di annidamento arbitrario di comandi.

Questa funzionalità è implementata tramite le classi **Command** e **Group**.

```
import click

@click.group()
@click.option('--debug/--no-debug', default=False)
def cli(debug):
    click.echo(f"Debug mode is {'on' if debug else 'off'}")

@cli.command()
def sync():
    click.echo('Syncing')
```

click - Comandi e Gruppi

in Examples/10 - click/example5.py viene come impostare le opzioni di un singolo comando e come passare le opzioni generali tramite la funzione `pass_context`

Esercizio hand on

Esercizio Hands On

Partendo dalla Macchina a Stati Finiti vista in precedenza, creiamo una nuova macchina che abbia **tre modalità di esecuzione** (batch, interattiva e web) da lanciare come sotto comandi, con **diversi livelli di verbosità** ed una **modalità di debug**.

Si consiglia di utilizzare un file di configurazione ed una relativa classe.

Argomenti

- Package e Moduli
- Variabili
- Classi e Oggetti
- Versionamento del software
- Dichiarazione di Condizione
- Operatori
- Cieli
- Funzioni
- Decoratori
- Namespace
- Lambda
- I/O
- Eeezioni
- PyPI

Packages:

- argparse
- click
- rich
- rich-click
- logging
- django
- Flask
 - pandas
 - numpy
 - scipy
 - matplotlib
 - multiprocessing
- tread
- sqlite
- ElementTree

Esercizio Hands On

Proseguo dell'esercizio guidato

PyPI

PyPI (Python Project Index, <https://pypi.org/>) è il repository di pacchetti Python a cui si accede di default con il comando `pip`.

In questa ultima parte vedremo come strutturare un modulo per poterlo pubblicare su PyPI e renderlo disponibile alla comunità.

Struttura del progetto

Creiamo una struttura di folder come riportato

```
packaging_tutorial/  
└── src/  
    └── example_package_me/  
        ├── __init__.py  
        └── example.py
```

È buona norma inserire tutti gli script nel folder src. In questo folder vengono raggruppati in cartelle, perchè in un pacchetto possono essere pubblicati più moduli.

Struttura del progetto

il file `__init__.py` lo lasciamo vuoto.

Nel file `example.py` inseriamo il codice:

```
def add_one(number):
    return number + 1
```

Struttura del progetto

Aggiungiamo i file necessari alla creazione del pacchetto:

```
packaging_tutorial/
├── LICENSE
├── pyproject.toml
├── README.md
└── src/
    └── example_package_me/
        ├── __init__.py
        └── example.py
└── tests/
```

Il folder **tests** è destinato ai test sul codice. Al momento lasciamo la cartella vuota.

Struttura del progetto - LICENSE

Il file contiene il contratto di licenza con cui viene rilasciato il software.

Introduciamo il concetto di *copyleft*:

Il copyleft altro non è che una modalità di esercizio del diritto d'autore che sfrutta i principi di base del diritto d'autore non per controllare la circolazione dell'opera bensì per stabilire un modello virtuoso di circolazione dell'opera, che si contrappone al modello detto proprietario. Il copyleft non potrebbe dunque esistere al di fuori del complesso delle norme sul diritto d'autore.

Struttura del progetto - LICENSE

Si parla di software Open Source

Il file contiene il contratto di licenza con cui viene rilasciato il software.

Introduciamo il concetto di *copyleft*:

Il copyleft altro non è che una modalità di esercizio del diritto d'autore che sfrutta i principi di base del diritto d'autore non per controllare la circolazione dell'opera bensì per stabilire un modello virtuoso di circolazione dell'opera, che si contrappone al modello detto proprietario. Il copyleft non potrebbe dunque esistere al di fuori del complesso delle norme sul diritto d'autore.

Struttura del progetto - LICENSE

Con "**copyleft debole**" ci si riferisce alle licenze per cui non tutte le opere derivate ereditano la licenza copyleft, spesso a seconda del modo in cui sono derivate.

Con "**copyleft forte**" si intendono quelle licenze per cui tutte le opere derivate e le librerie collegate dinamicamente ad esse, ereditano la licenza copyleft.

Non Copyleft

Licenze BSD, licenza MIT, licenza Apache, WTFPL, licenza PHP

Copyleft Debole

GNU Lesser General Public License (LGPL), Mozilla Public License (MPL), Eclipse Public License (EPL)

Copyleft Forte

GNU General Public License (GPL)

Struttura del progetto - README.md

Contiene la descrizione del package, istruzioni per l'installazione e utilizzo.

Un esempio è riportato di seguito:

```
# Example Package

This is a simple example package. You can use
[Github-flavored
Markdown](https://guides.github.com/features/mastering-markdown/)
to write your content.
```

Struttura del progetto - project.toml

pyproject.toml indica agli strumenti di creazione "frontend" come pip e build quale strumento "backend" utilizzare per creare pacchetti di distribuzione per il tuo progetto.

```
[build-system]
requires = ["setuptools>=61.0"]
build-backend = "setuptools.build_meta"
```

Struttura del progetto - project.toml

Possono essere aggiunti dei metadati per descrivere meglio il progetto.

```
[project]
  name = "example_package_me"
  version = "0.0.1"
  authors = [
    { name="Example Author", email="author@example.com" },
  ]
  description = "A small example package"
  readme = "README.md"
  requires-python = ">=3.7"
  classifiers = [
    "Programming Language :: Python :: 3",
    "License :: OSI Approved :: MIT License",
    "Operating System :: OS Independent",
  ]

[project.urls]
  "Homepage" = "https://github.com/pypa/sampleproject"
  "Bug Tracker" = "https://github.com/pypa/sampleproject/issues"
```

L'elenco completo dei metadati è disponibile all'indirizzo:
[https://packaging.python.org/en/latest/specifications/declaring-project-metadata](https://packaging.python.org/en/latest/specifications/declaring-project-metadata/#declaring-project-metadata)

Generazione del pacchetto di distribuzione

Installiamo il pacchetto build:

```
python3 -m pip install --upgrade build
```

Creiamo i pacchetti:

```
python3 -m build
```

Viene generata una cartella dist con due file:

```
dist/
└── example_package_me-0.0.1-py3-none-any.whl
    └── example_package_me-0.0.1.tar.gz
```

Il file tar.gz è la distribuzione del codice sorgente, mentre il file .whl è una distribuzione già “costruita”.

Installazione del pacchetto

```
python3 -m pip install example_package_me-0.0.1-py3-none-any.whl
```

Se si vuole invece caricare il pacchetto su PyPI installiamo il pacchetto **twine**

```
python3 -m pip install --upgrade twine
```

dopo di che effettuiamo l'upload

```
python3 -m twine upload --repository testpypi dist/*
```

Installazione del pacchetto

```
python3 -m pip install example_package_me-0.0.1-py3-none-any.whl
```

Se si vuole invece caricare il pacchetto su PyPI installiamo il pacchetto **twine**

```
python3 -m pip install --upgrade twine
```

dopo di che effettuiamo l'upload

```
python3 -m twine upload --repository testpypi dist/*
```



Repository di test

Installazione del pacchetto

```
python3 -m pip install --index-url https://test.pypi.org/simple/ example-package-me
```

L'upload del software sul server principale eseguiamo il comando:

```
python3 -m twine upload dist/*0.0.1*
```

e l'installazione avverrà con il classico comando

```
python3 -m pip install example-package-me
```