

# Cloud Archive and Computation

---

Romolo Politi

# Indice

---

# Elenco Lezioni

Lezione 1 - 16/06/2021

Lezione 2 - 18/06/2021

Lezione 3 - 22/06/2021

Lezione 4 - 28/06/2021

Lezione 5 - 30/06/2021

Lezione 6 - 02/07/2021

# Lezione 1

---

# 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

# Tipi di Cloud

---

# Tipi di Cloud

## In Promise



## Out Promise



Google Cloud

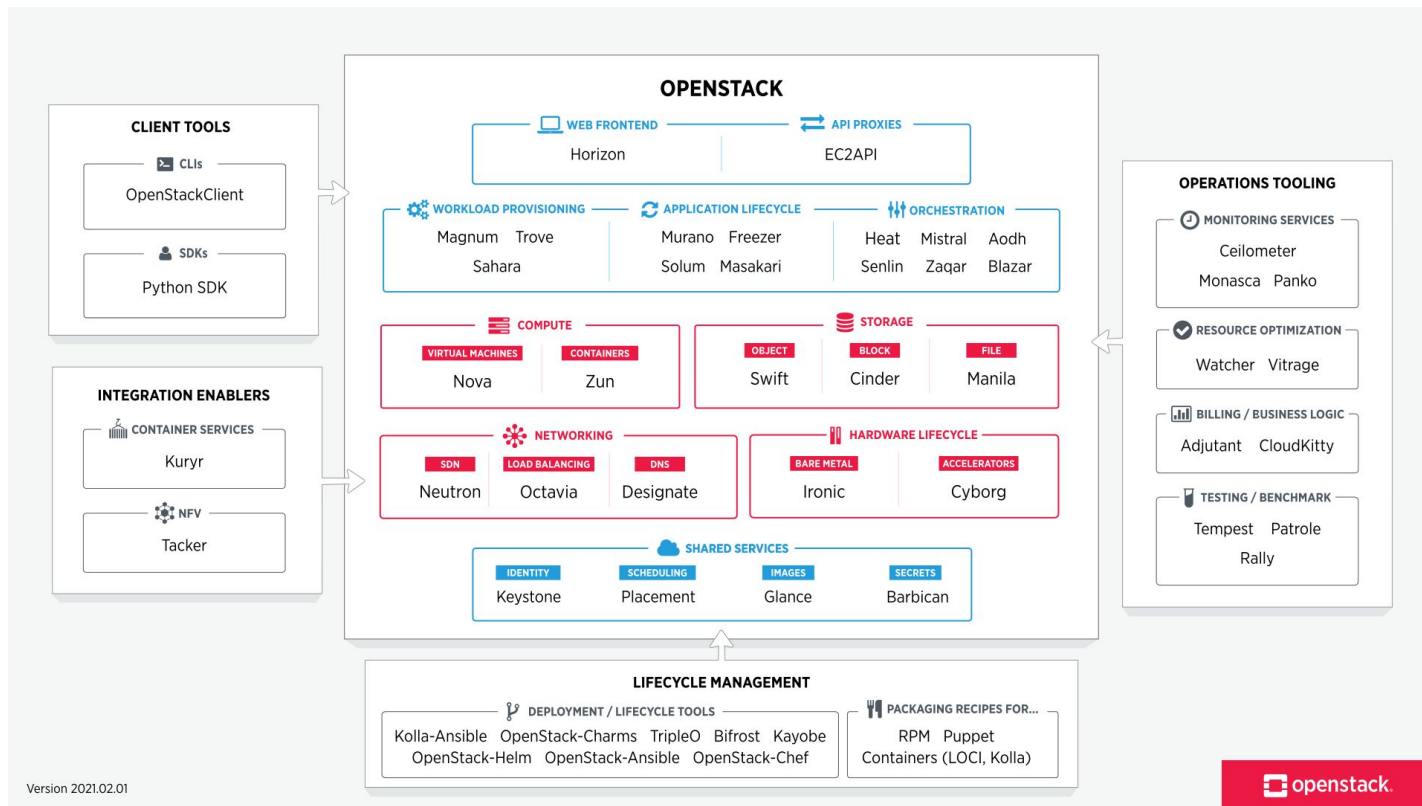


# Struttura del Cloud

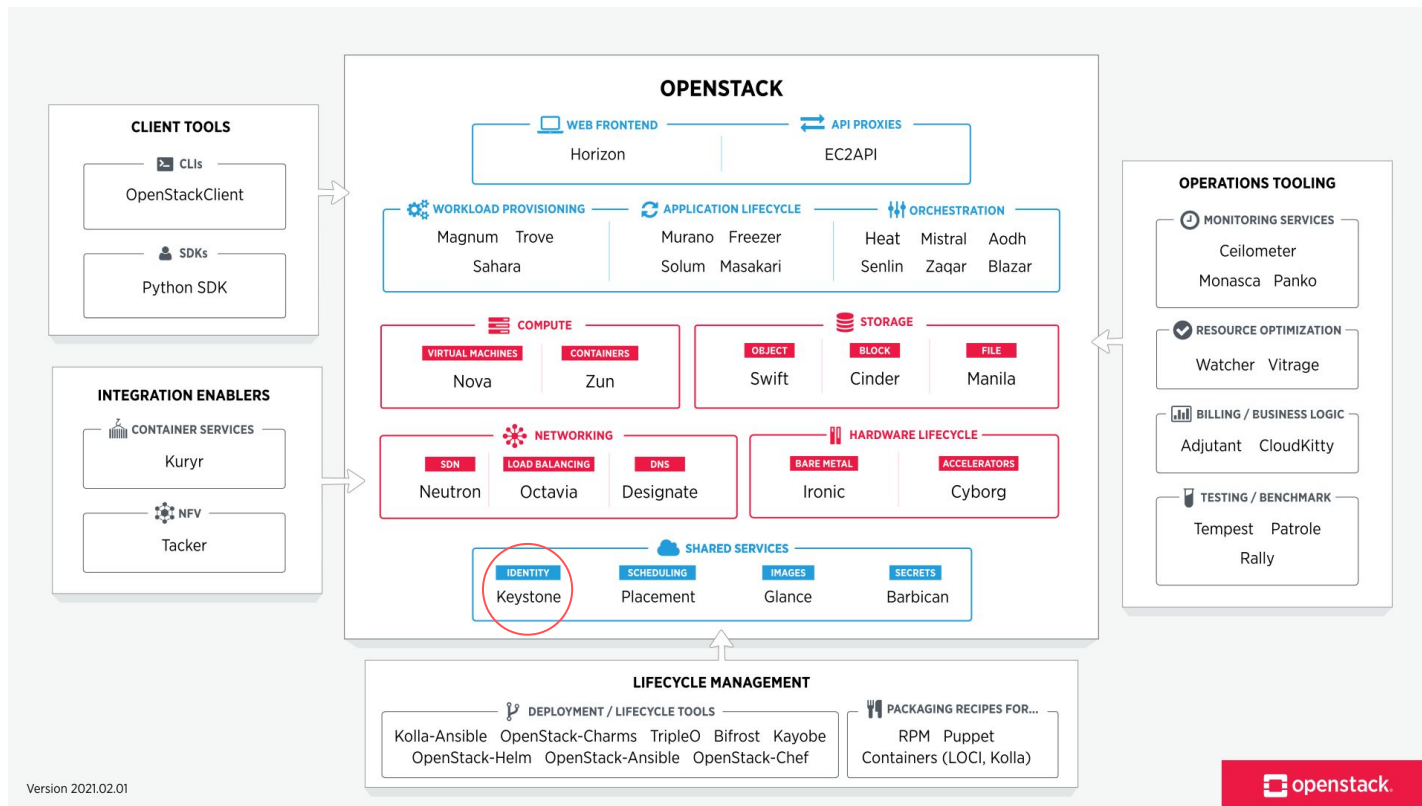
---



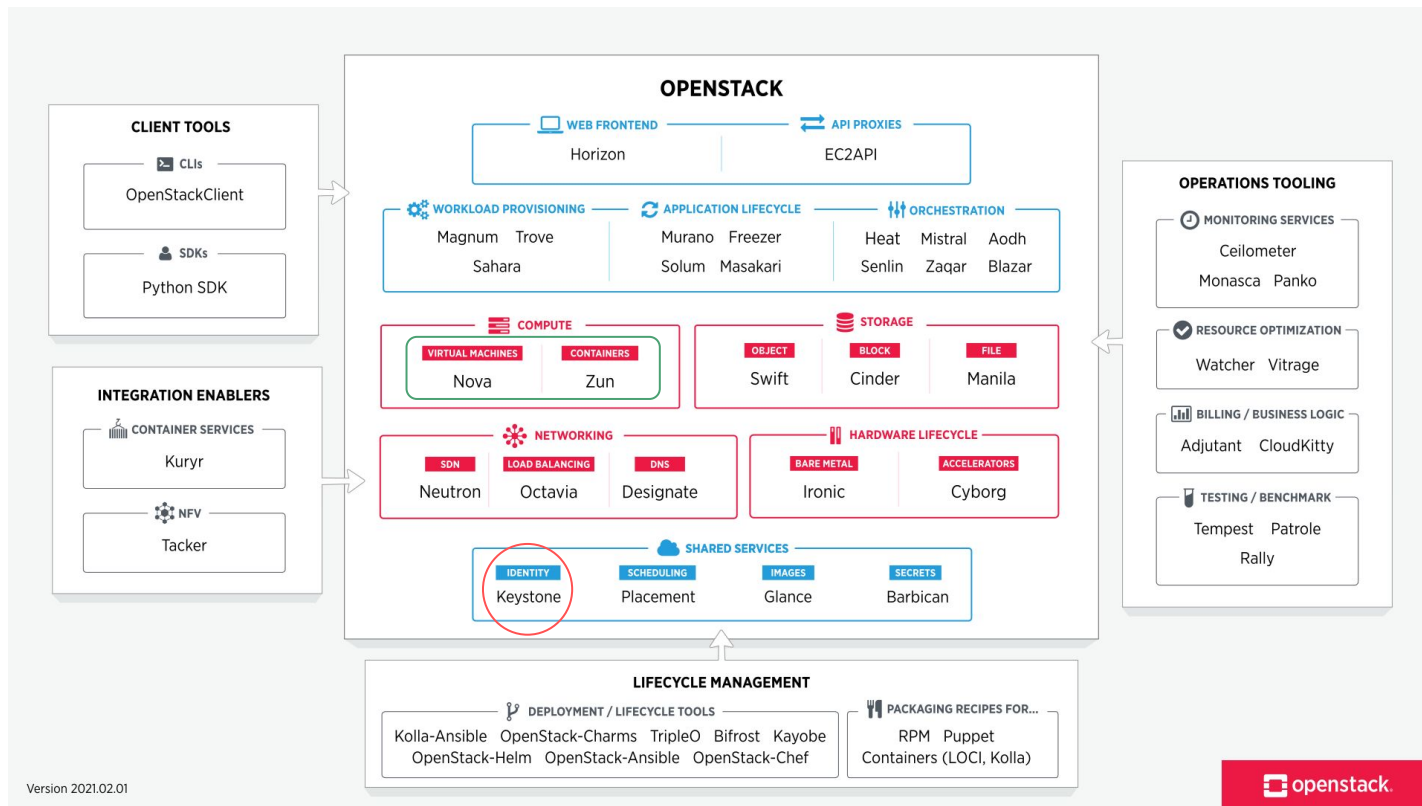
# Struttura di un cloud



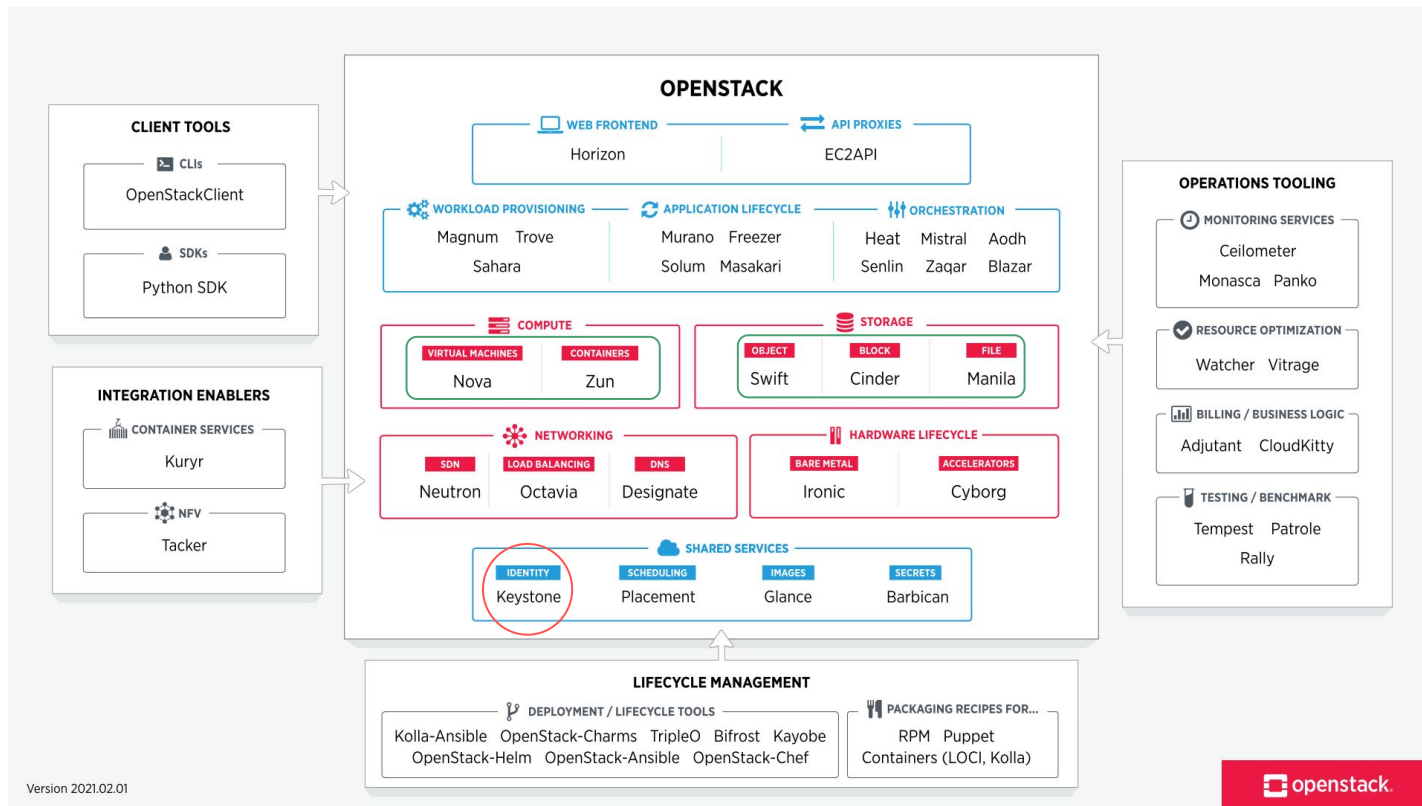
# Struttura di un cloud



# Struttura di un cloud



# Struttura di un cloud



# 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)



Sign in with your Google Account



☒ Stay signed in

[Need help?](#)

# Componenti Principali

- IAM (Identity and Access Management)



# Componenti Principali

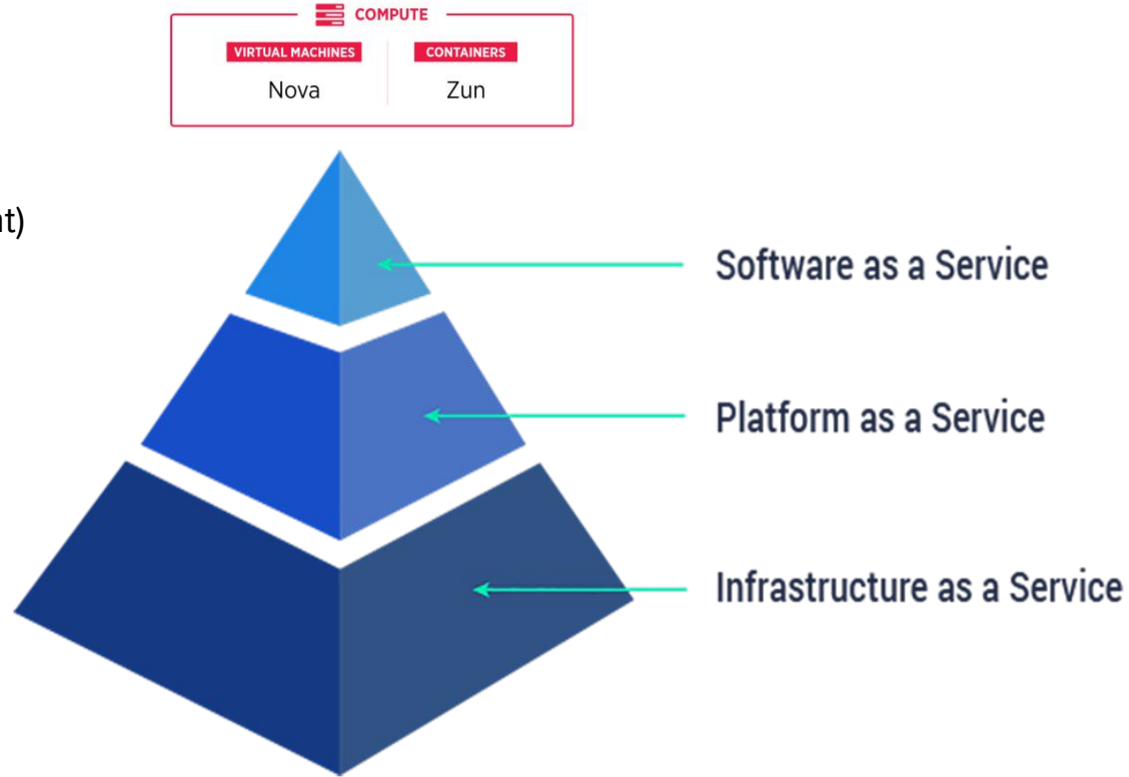
- IAM (Identity and Access Management)
- IaaS
- PaaS
- SaaS





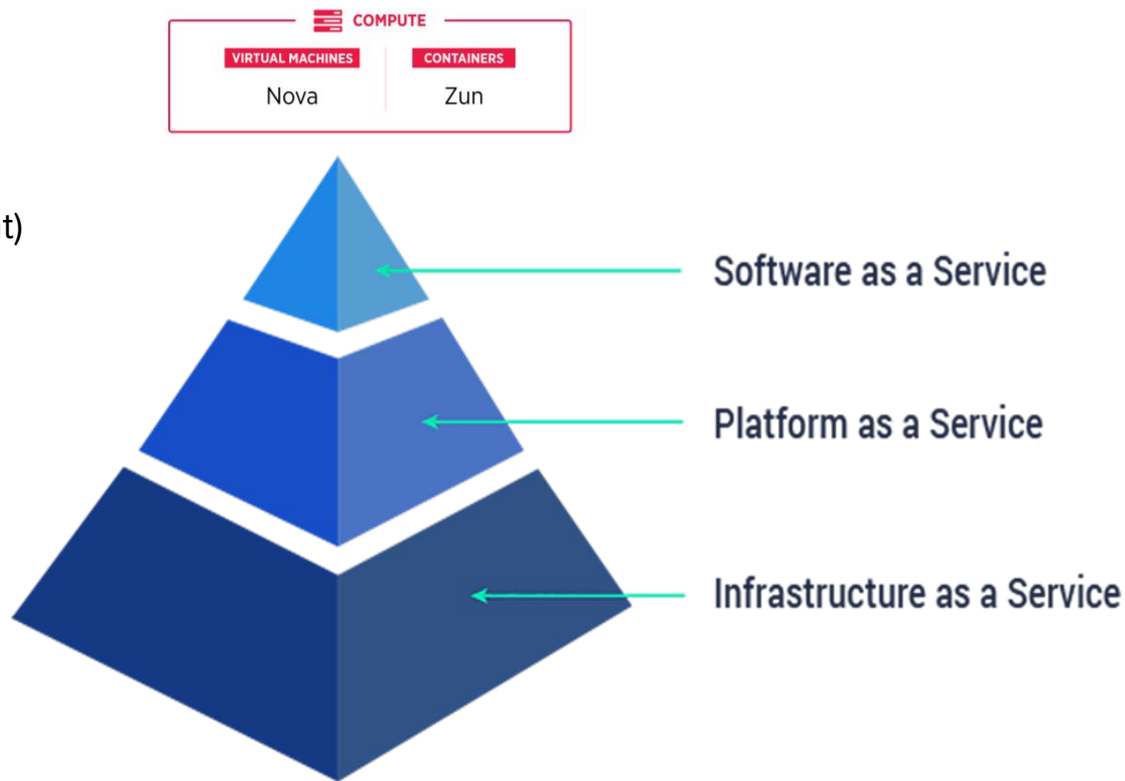
# Componenti Principali

- IAM (Identity and Access Management)
- IaaS
- PaaS
- SaaS



# Componenti Principali

- IAM (Identity and Access Management)
- IaaS
- PaaS
- SaaS

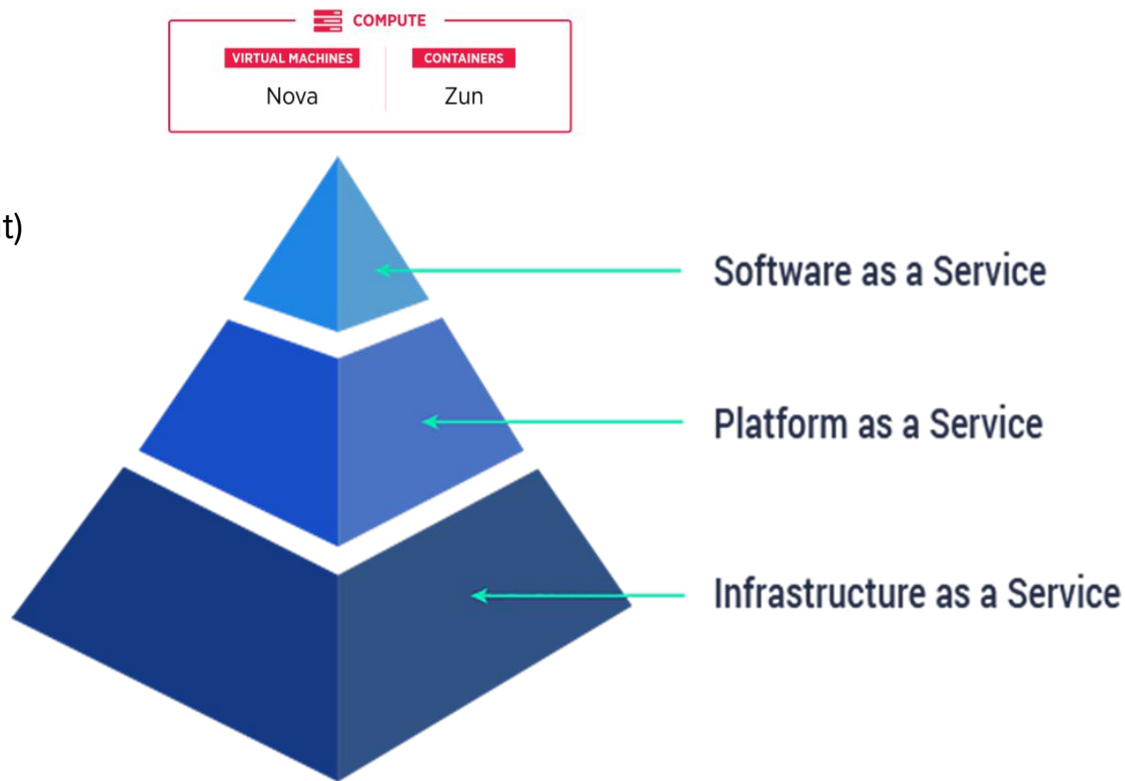


## IaaS

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

# Componenti Principali

- IAM (Identity and Access Management)
- IaaS
- PaaS
- SaaS

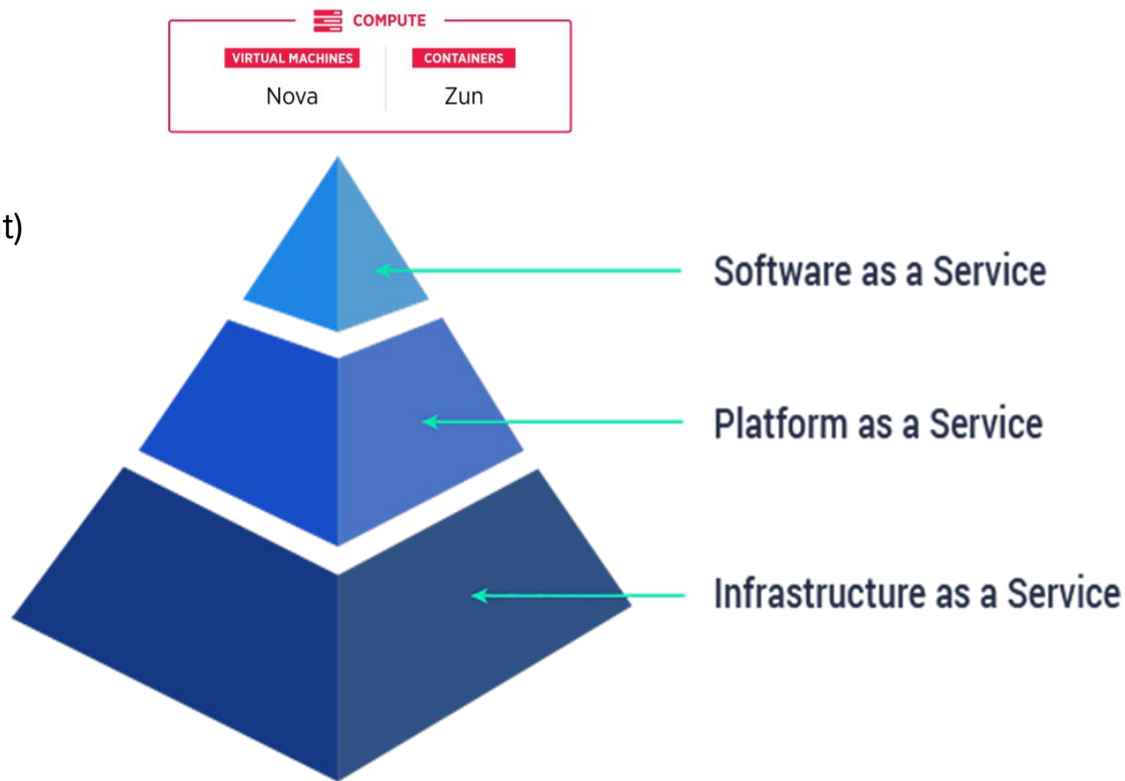


## PaaS

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

# Componenti Principali

- IAM (Identity and Access Management)
- IaaS
- PaaS
- SaaS

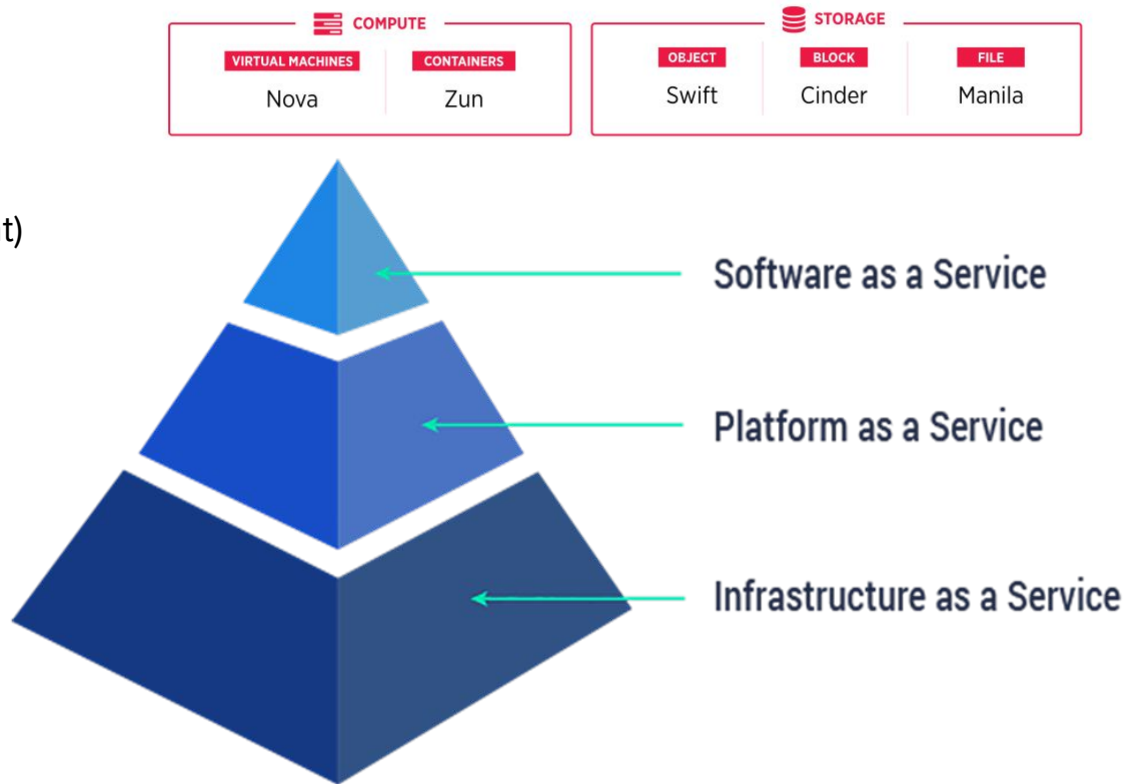


## SaaS

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.

# Componenti Principali

- IAM (Identity and Access Management)
- IaaS
- PaaS
- SaaS

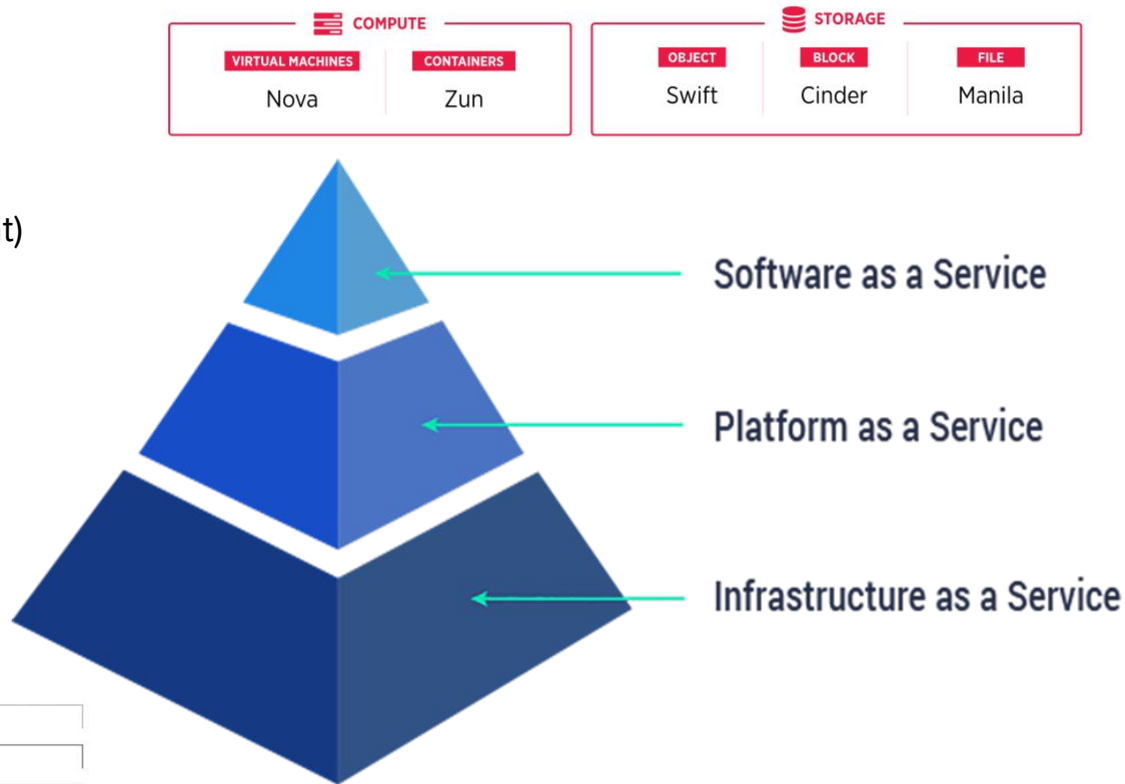


## SaaS

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.

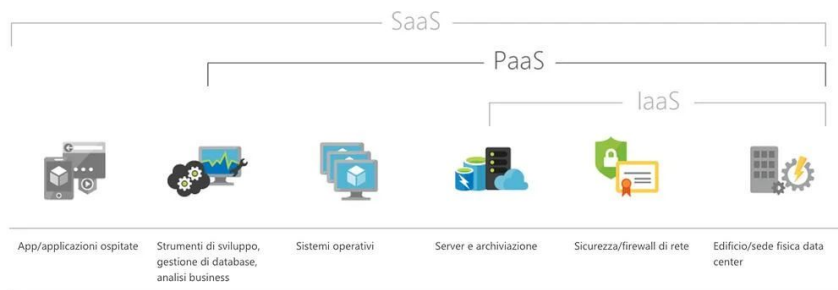
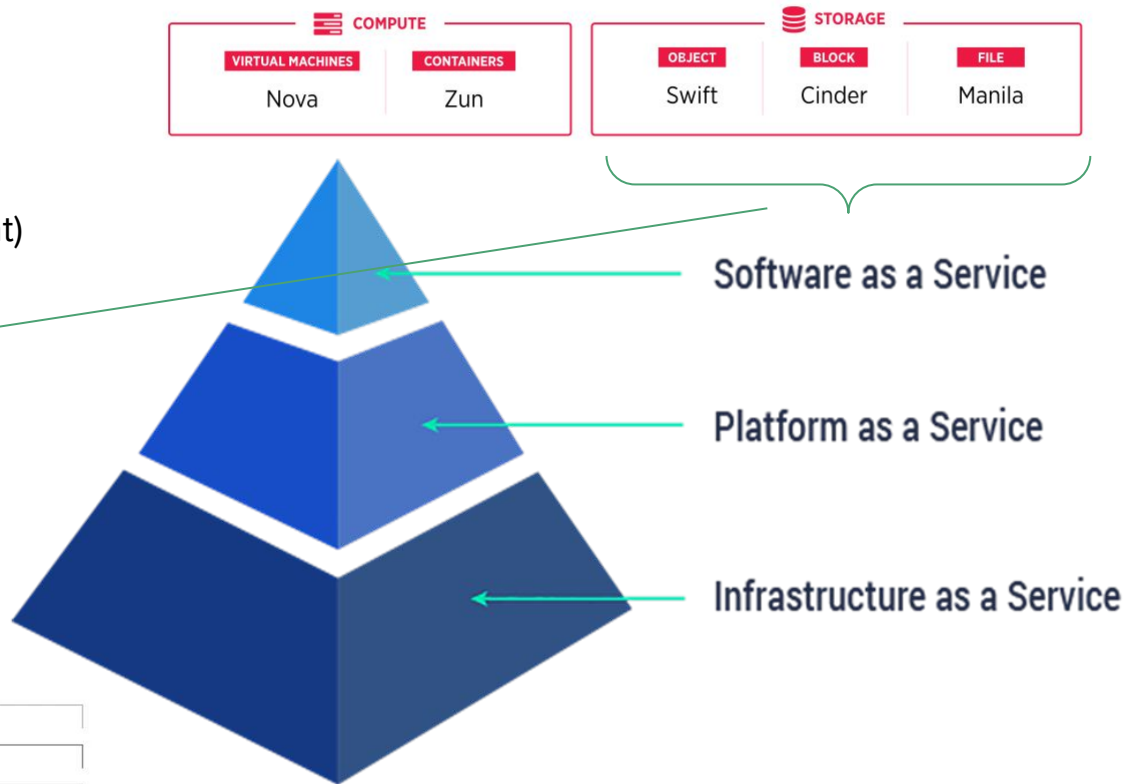
# Componenti Principali

- IAM (Identity and Access Management)
- IaaS
- PaaS
- SaaS



# Componenti Principali

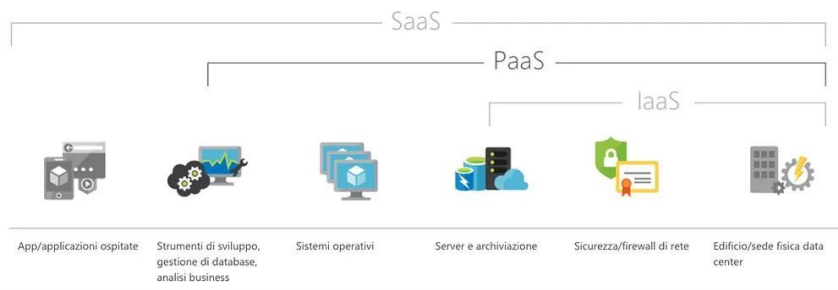
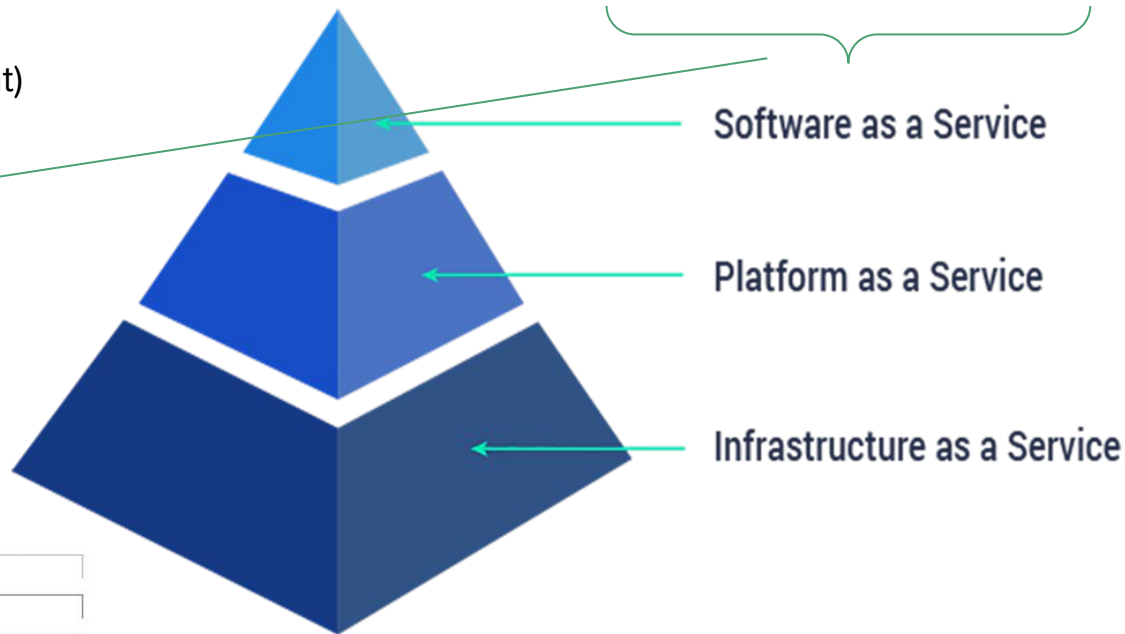
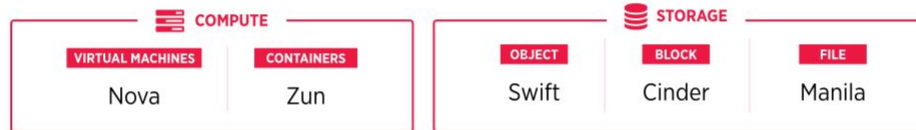
- IAM (Identity and Access Management)
- IaaS
- PaaS
- SaaS
- DaaS



# Componenti Principali

- IAM (Identity and Access Management)
- IaaS
- PaaS
- SaaS
- DaaS

Data as a Service

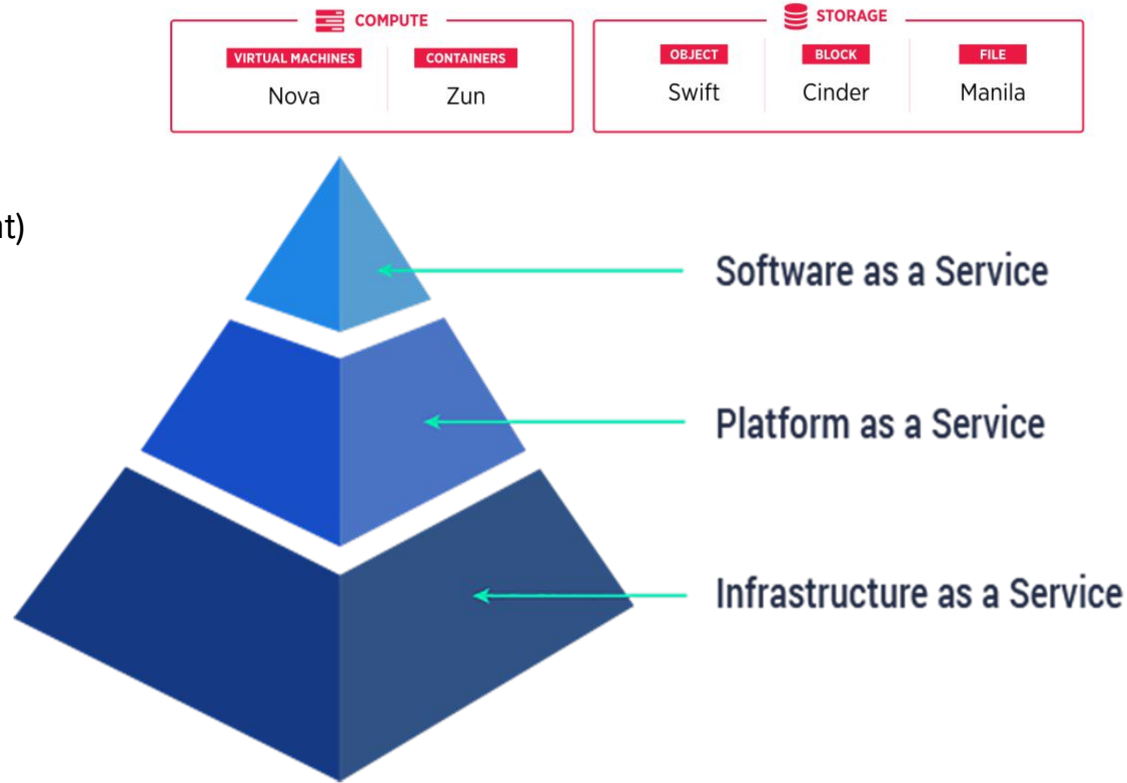




# Componenti Principali

- IAM (Identity and Access Management)
- IaaS
- PaaS
- SaaS
- DaaS

Data as a Service



# Dati e Metadati

---

# Definizione di Dato

Un dato (dal latino datum che significa 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 byte.

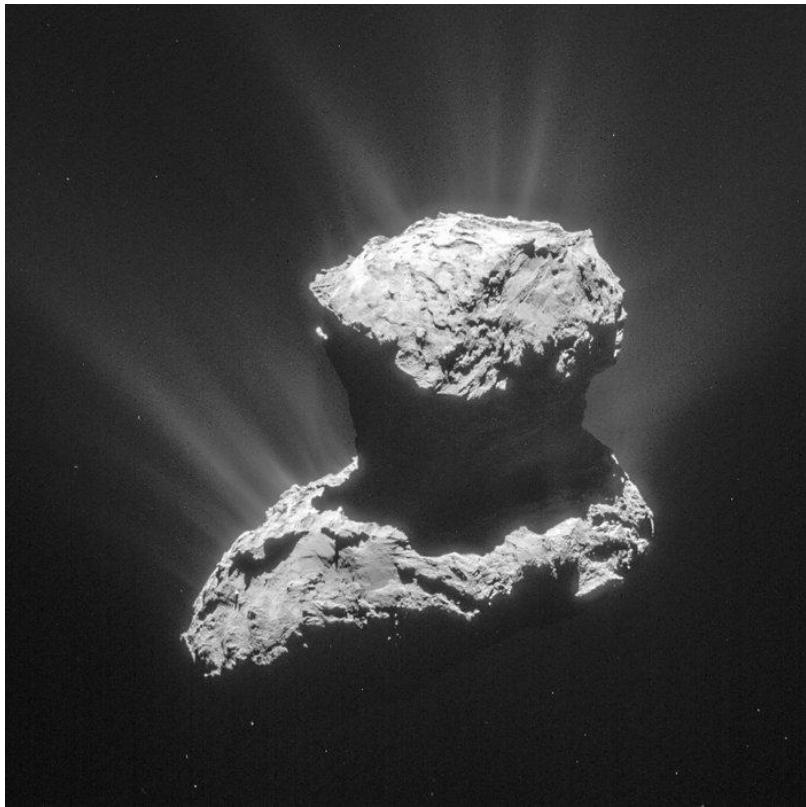
# Definizione di Metadato

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

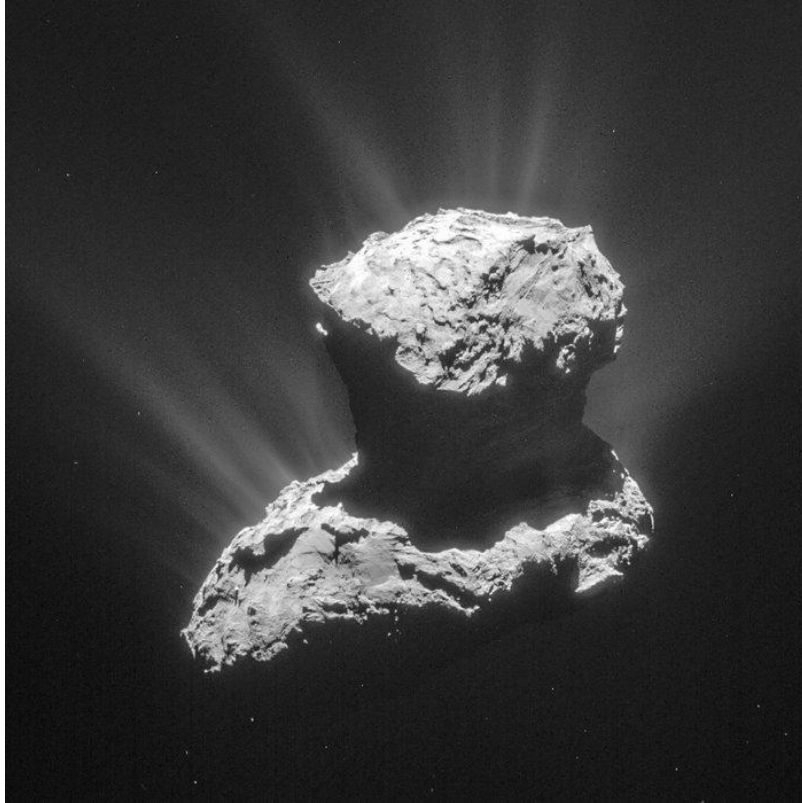
Un esempio tipico di metadati è costituito dalla scheda del catalogo di una biblioteca, la quale contiene informazioni circa il contenuto e la posizione di un libro, cioè dati riguardanti più dati che si riferiscono al libro. Un altro contenuto tipico dei metadati può essere la fonte o l'autore dell'insieme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

# Dato e Metadato in Planetologia

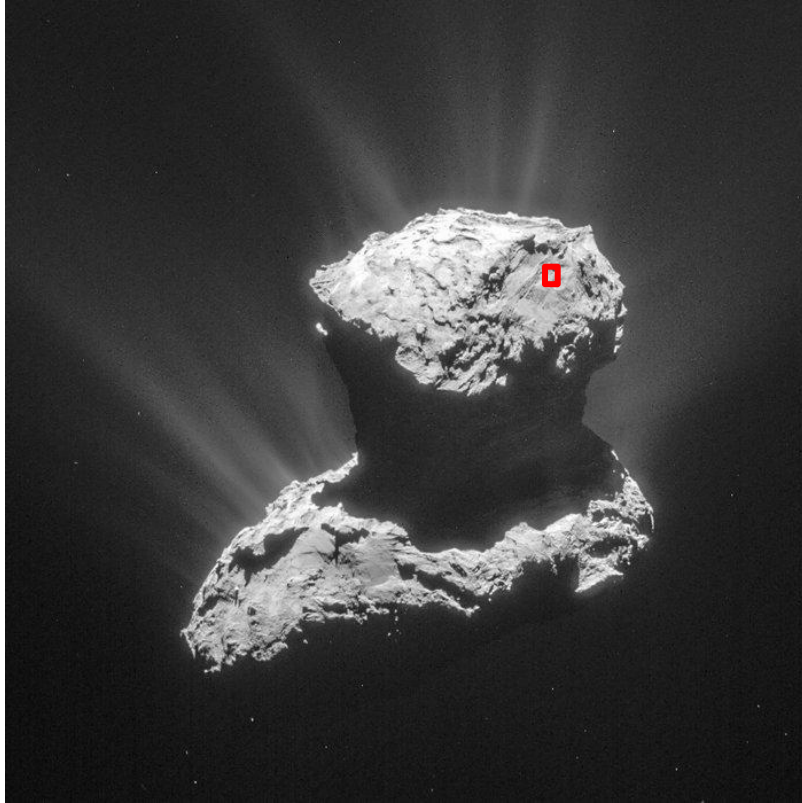


# Dato e Metadato in Planetologia



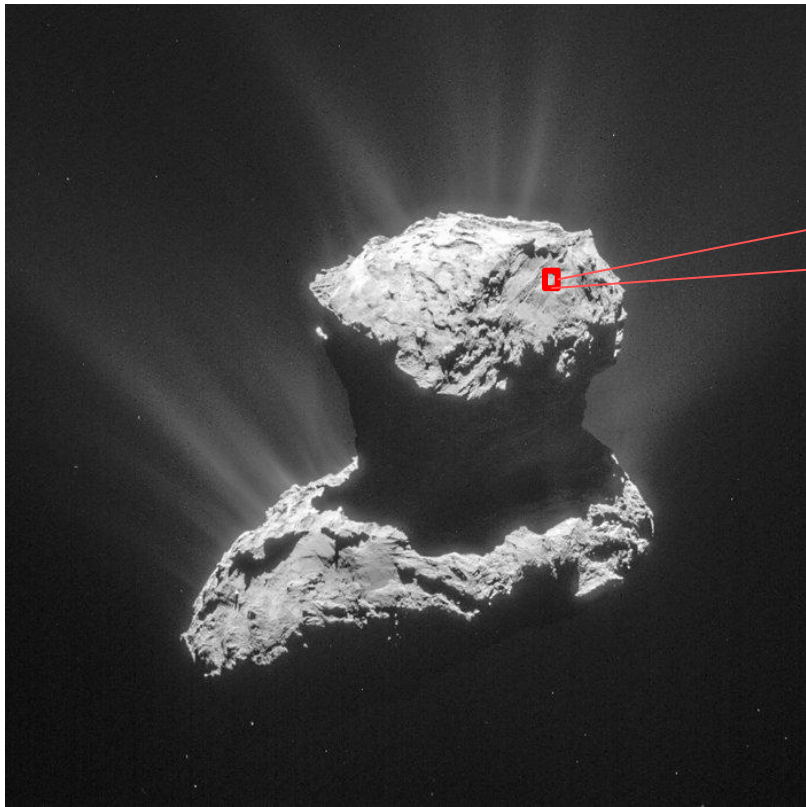
Quale è il dato in questo caso?

# Dato e Metadato in Planetologia



Quale è il dato in questo caso?

# Dato e Metadato in Planetologia

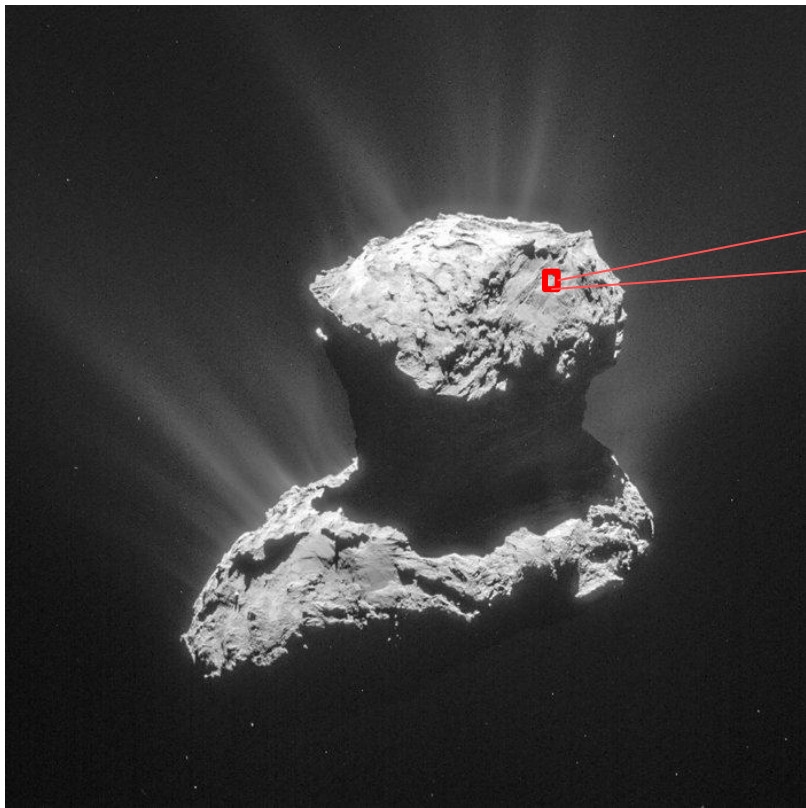


Quale è il dato in questo caso?

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



# Dato e Metadato in Planetologia

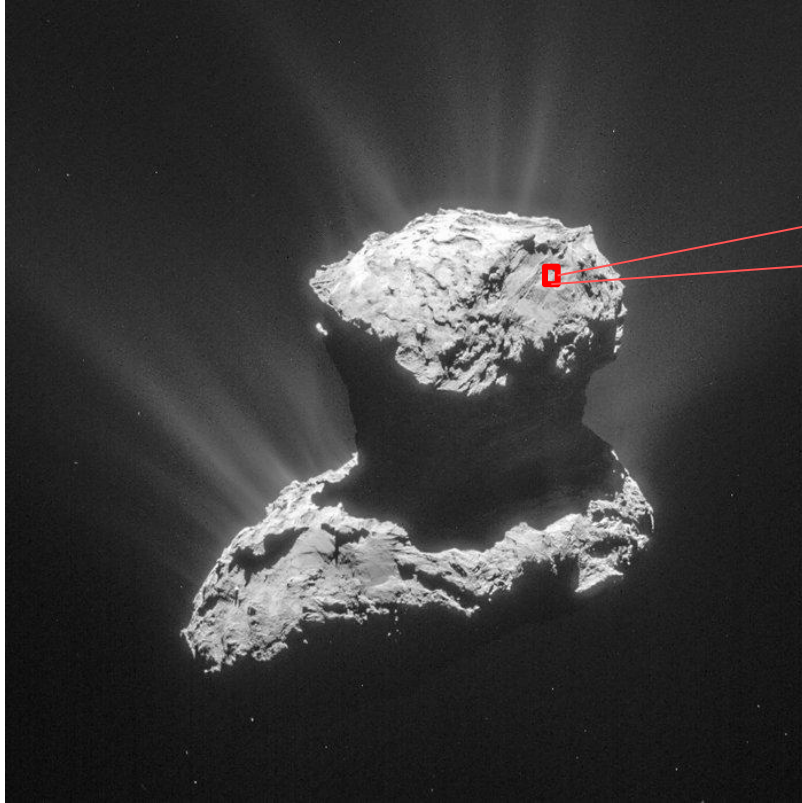


Quale è il dato in questo caso?

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

Ha significato?

# Dato e Metadato in Planetologia



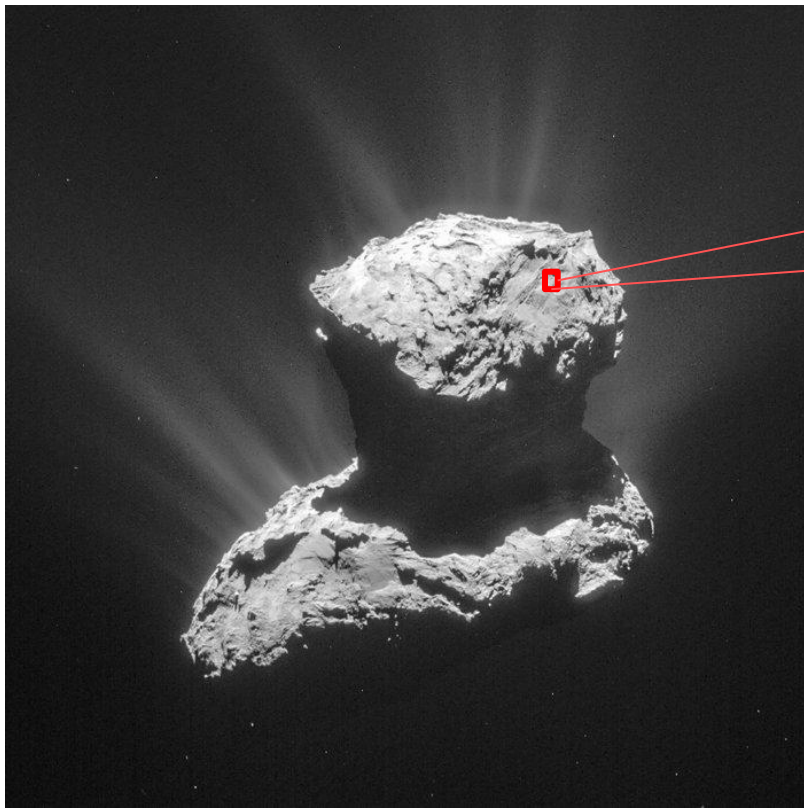
Quale è il dato in questo caso?

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

Ha significato?

**No**

# Dato e Metadato in Planetologia



Quale è il dato in questo caso?

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

Ha significato?

**No**

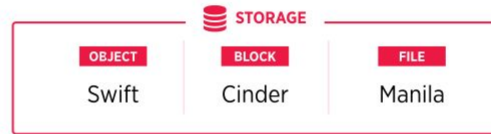
Si ha necessità di conoscere

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

# Dati nel Cloud

---

# Dati nel Cloud



# Differenza tra **Object Storage** e **File Storage**

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.

# Differenza tra **Object Storage** e **File Storage**

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.

*Unix and Unix Like*

# Differenza tra **Object Storage** e **File Storage**

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.

Unix and Unix Like

- 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.

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



# Differenza tra **Object Storage** e **File Storage**

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.

Unix and Unix Like

Max 255 caratteri

- 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.

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

# Differenza tra **Object Storage** e **File Storage**

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.

*Unix and Unix Like*

I set di nomi richiesti 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/ldft`
- nel path relativo si può indicare il percorso a partire dal folder in cui ci si trova.  
`projectb/plans/ldft`

Un path relativo non può iniziare con /.

Simboli speciali:

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

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

# Differenza tra **Object Storage** e **File Storage**

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.

**Max 1024 caratteri**

*Unix and Unix Like*

I set di nomi richiesti 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/ldft`
- nel path relativo si può indicare il percorso a partire dal folder in cui ci si trova.

`projectb/plans/ldft`

Un path relativo non può iniziare con /.

Simboli speciali:

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

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

# Differenza tra **Object Storage** e **File Storage**

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.

*Unix and Unix Like*

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

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

## Univ

Il **file storage** è il formato di storage maggiormente conosciuto: i dati

```
-rwxrw-r-- 10 root root 2048 Jan 13 07:11 afile.exe  
?UUUGGGGOOS 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
```

```
-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

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

## Univ

Il **file storage** è il formato di storage maggiormente conosciuto: i dati

```
-rwxrw-r-- 10 root root 2048 Jan 13 07:11 afile.exe  
?UUUGGGGOOS 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
```

```
-rw-rw-rw- 1 romelo romelo 658 apr 30 00:56 manage.py
```

	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 <b>cd</b> .
	x	The file can be executed.	The directory can be accessed with <b>cd</b> ; 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 <b>x</b> bit set, the final file or folder cannot be accessed either, regardless of its permissions; see <a href="#">path_resolution(7)</a> for more information.
	s	The <b>setuid</b> bit when found in the <b>user</b> triad; the <b>setgid</b> bit when found in the <b>group</b> triad; it is not found in the <b>others</b> triad; it also implies that <b>x</b> is set.	
	S	Same as <b>s</b> , but <b>x</b> is not set; rare on regular files, and useless on folders.	
	t	The sticky bit; it can only be found in the <b>others</b> triad; it also implies that <b>x</b> is set.	
	T	Same as <b>t</b> , but <b>x</b> is not set; rare on regular files, and useless on folders.	

## Univ

maggiormente conosciuto: i dati

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

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

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

# Differenza tra **Object Storage** e **File Storage**

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.

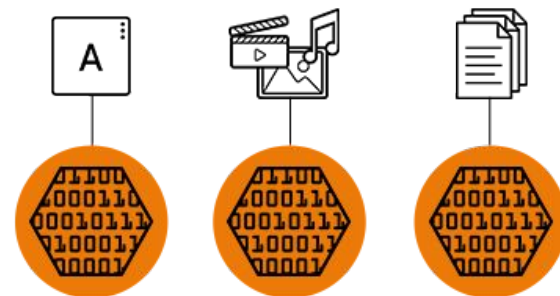


# Differenza tra **Object Storage** e **File Storage**

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.

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



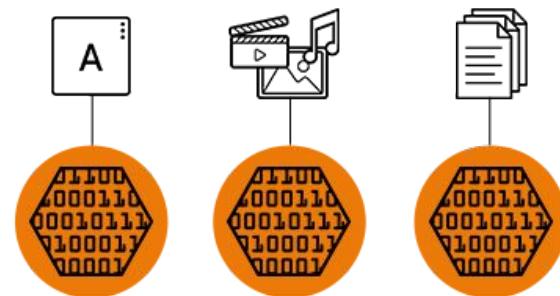
# Differenza tra **Object Storage** e **File Storage**

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.

Nello storage di oggetti, i dati vengono frammentati in unità discrete chiamate appunto oggetti e conservati in un unico repository 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.



# Differenza tra **Object Storage** e **File Storage**

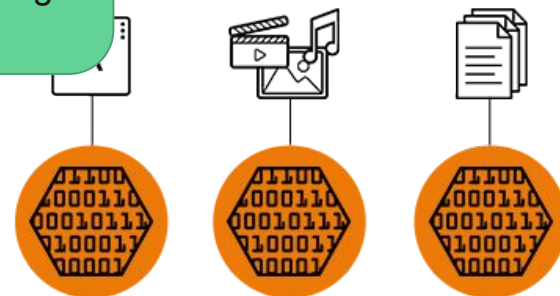
Il **file storage** è il formato di storage maggiormente conosciuto: i dati vengono archiviati in file con cui è possibile interagire, contenuti in cartelle all'interno di una directory file gerarchica.

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

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.

Nello storage di oggetti i dati sono chiamati appunto oggetti, 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.



# Differenza tra **Object Storage** e **File Storage**

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.

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.

# Lezione 2

---

# 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

# Calcolo nel Cloud

---

# Calcolo nel Cloud

## PaaS

Ho un ambiente di lavoro del tutto simile al pc ma con risorse molto più elevate.

### **Schema di sviluppo**

Sviluppo in locale con dataset limitati.

Trasferimento su cloud dei codici/dati

Esecuzione su larga scala.

(Elevato numero di CPU/GPU, RAM, FLOPs, etc)



# Calcolo nel Cloud

## PaaS

Ho un ambiente di lavoro del tutto simile al pc ma con risorse molto più elevate.

### Schema di sviluppo

Sviluppo in locale con dataset limitati.

Trasferimento su cloud dei codici/dati

Esecuzione su larga scala.

(Elevato numero di CPU/GPU, RAM, FLOPs, etc)

## SaaS

Sviluppo un codice che come GUI, solitamente, una pagina web.

### Schema di sviluppo

Sviluppo in locale con opzioni di debug attive

Trasferimento su server e messa online.

# Calcolo nel Cloud

## PaaS

Ho un ambiente di lavoro del tutto simile al pc ma con risorse molto più elevate.

### Schema di sviluppo

Sviluppo in locale con dataset limitati.

Trasferimento su cloud dei codici/dati

Esecuzione su larga scala.

(Elevato numero di CPU/GPU, RAM, FLOPs, etc)

## SaaS

Sviluppo un codice che come GUI, solitamente, una pagina web.

### Schema di sviluppo

Sviluppo in locale con opzioni di **debug attivo**

Trasferimento su server e messa online.

# Calcolo nel Cloud

## PaaS

Ho un ambiente di lavoro del tutto simile al pc ma con risorse molto più elevate.

### Schema di sviluppo

Sviluppo in locale con dataset limitati.

Trasferimento su cloud dei codici/dati

Esecuzione su larga scala

(Elevato numero di CP

## SaaS

Sviluppo un codice che come GUI, solitamente, una pagina web.

### Schema di sviluppo

Sviluppo in locale con opzioni di **debug active**

Trasferimento su server e messa online.

Le opzioni di debug non sono mai attive sul server di produzione, questo perchè il debug, per sua natura, apre alla possibilità di Hacking/Cracking del codice e della macchina

# Archiviazione (Parte 1)

---

# Sistemi di archiviazione

## Cosa

- archiviazione di file (file storage)
- archiviazione di oggetti (object storage/database)

# Sistemi di archiviazione

## Cosa

- archiviazione di file (file storage)
- archiviazione di oggetti (object storage/database)

## Come

- archiviazione gerarchica/relazionale
- archiviazione piana

# Sistemi di archiviazione

## Cosa

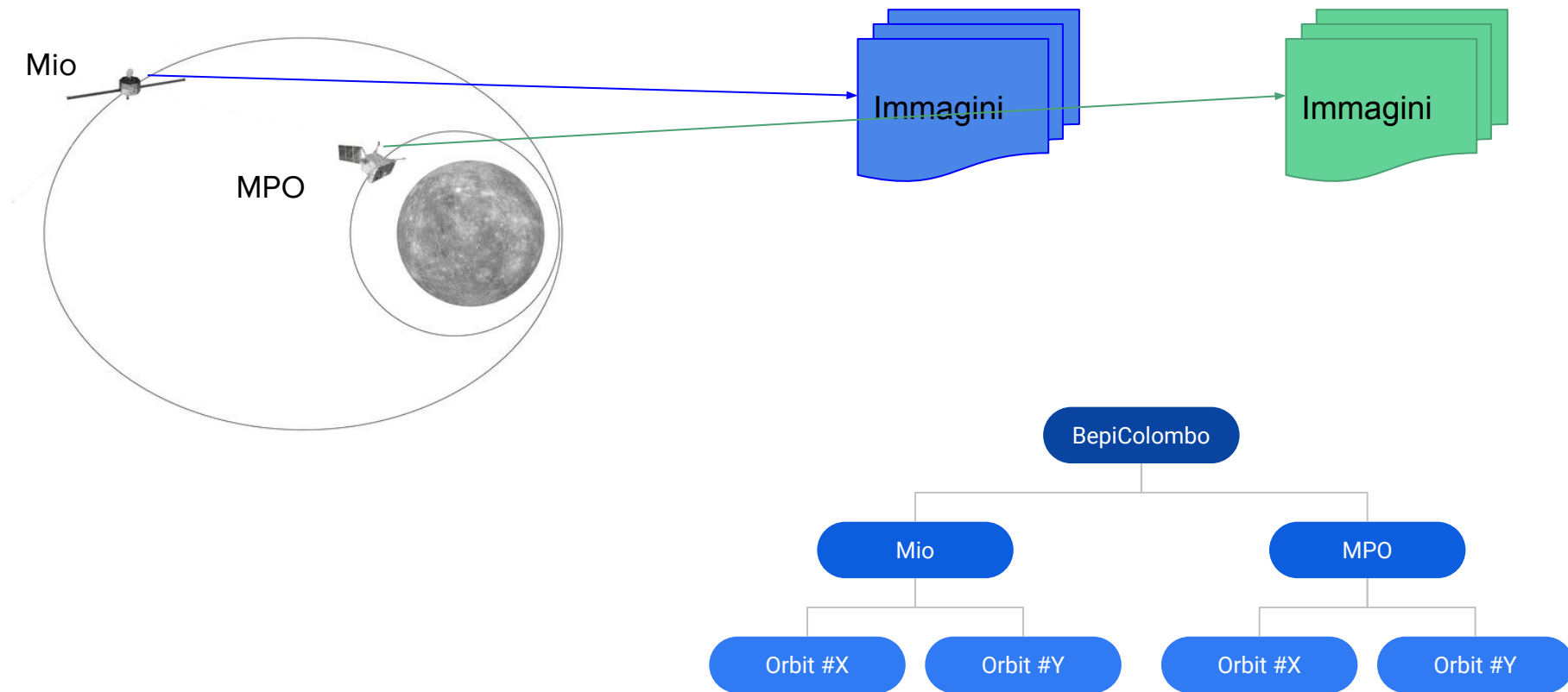
- archiviazione di file (file storage)
- archiviazione di oggetti (object storage/database)

## Come

- archiviazione gerarchica/relazionale
- archiviazione piana

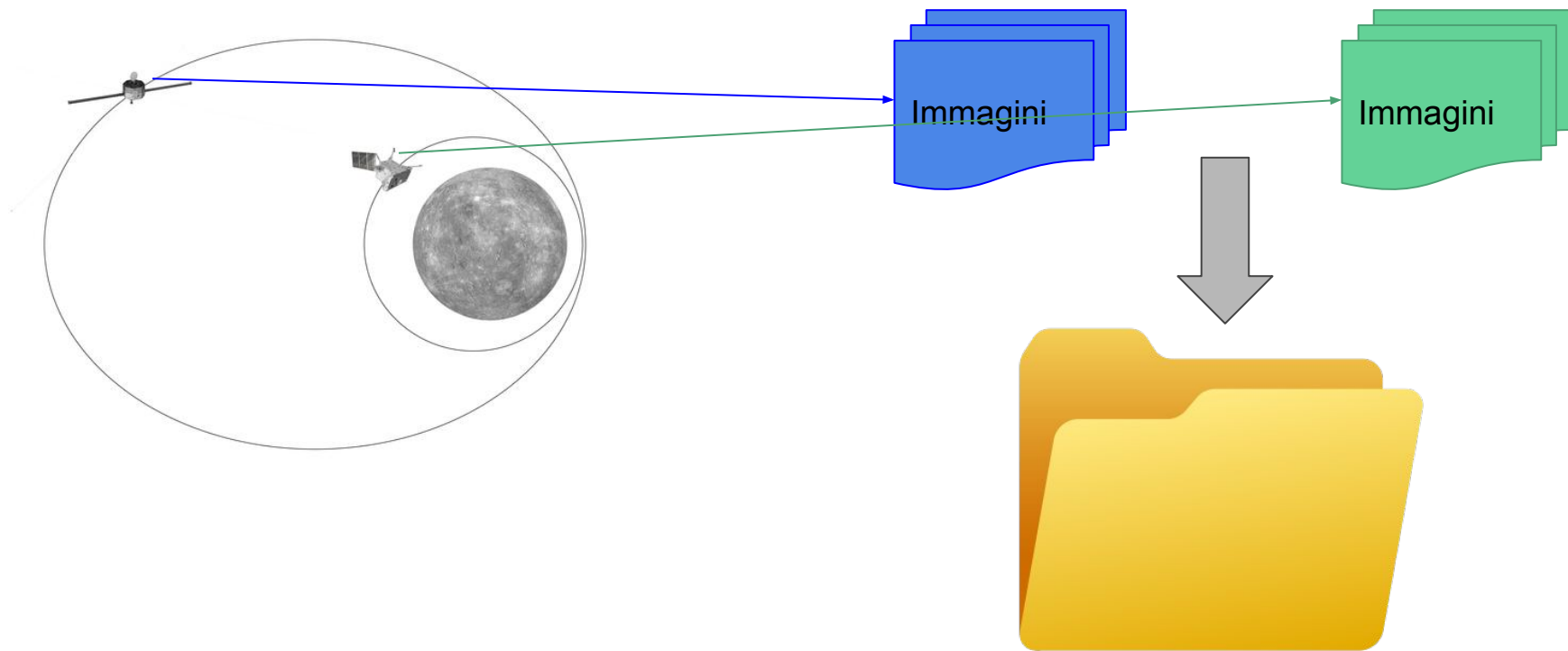
## Archiviazione Ibrida

# Esempio Archiviazione gerarchica





# Esempio Archiviazione Piana



Quali dei due è più efficiente?

# Quali dei due è più efficiente?

La risposta è...

# Quali dei due è più efficiente?

La risposta è... un'altra domanda

# Quali dei due è più efficiente?

La risposta è... un'altra domanda

Chi deve “vedere l'archivio”?

# Quali dei due è più efficiente?

La risposta è... un'altra domanda

Chi deve “vedere l'archivio”?

**Macchina**

Archiviazione piana

# Quali dei due è più efficiente?

La risposta è... un'altra domanda

Chi deve “vedere l'archivio”?

## **Macchina**

Archiviazione piana

## **Uomo**

Archiviazione gerarchica

# Concetto di Data Preservation



# Concetto di Data Preservation

Nella gestione dei dati (*Data Management*), il ***Data Preservation*** è il processo di che permette di mantenere l'accesso ai dati, in modo che questi possano essere trovati, compresi ed utilizzati in futuro.

# Concetto di Data Preservation

Nella gestione dei dati (*Data Management*), il **Data Preservation** è il processo di che permette di mantenere l'accesso ai dati, in modo che questi possano essere trovati, compresi ed utilizzati in futuro.

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

# Concetto di Data Preservation

Nella gestione dei dati (*Data Management*), il **Data Preservation** è il processo di che permette di mantenere l'accesso ai dati, in modo che questi possano essere trovati, compresi ed utilizzati in futuro.

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

# Concetto di Data Preservation

Nella gestione dei dati (*Data Management*), il **Data Preservation** è il processo di che permette di mantenere l'accesso ai dati, in modo che questi possano essere trovati, compresi ed utilizzati in futuro.

- 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*), il **Data Preservation** è il processo di che permette di mantenere l'accesso ai dati, in modo che questi possano essere trovati, compresi ed utilizzati in futuro.

- 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.

# Archiviazione Ibrida

## **LTP**

File Storage

Archiviazione gerarchica

## **STP**

Object storage

Archiviazione piana

Database Relazionale

# Database Relazionale

---

# Cos'è un Database relazionale

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](#).

([Wikipedia](#))

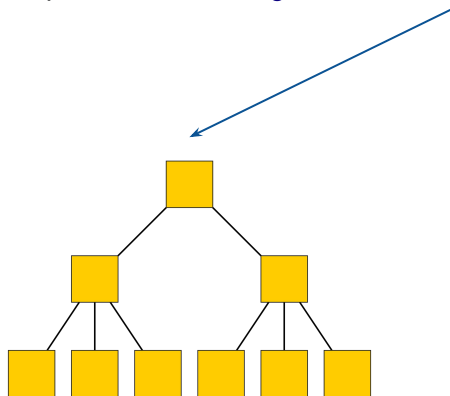


# Cos'è un Database relazionale

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**.

([Wikipedia](#))

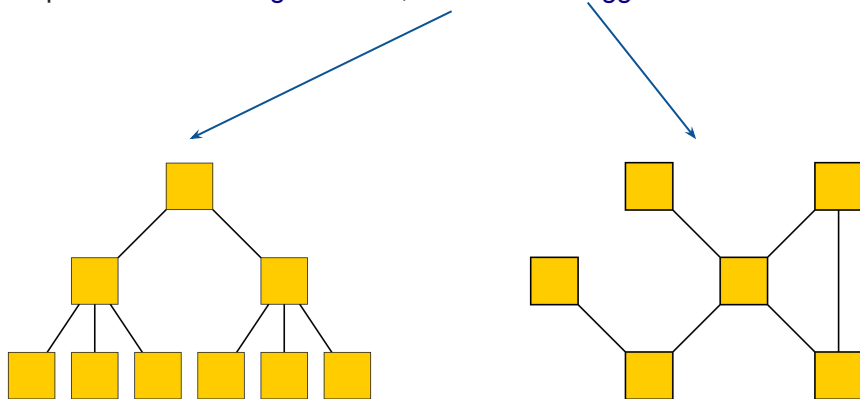


# Cos'è un Database relazionale

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**.

([Wikipedia](#))

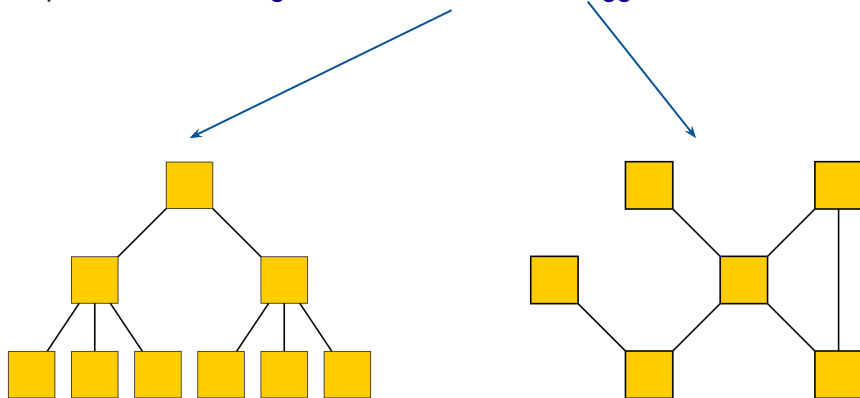


# Cos'è un Database relazionale

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**.

([Wikipedia](#))



Se controllate la lista dei vari database troverete che tra i relazionali e DB ad oggetti troverete che è presente PostgreSQL



# Cos'è un Database relazionale

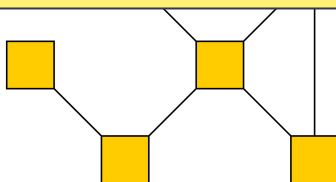
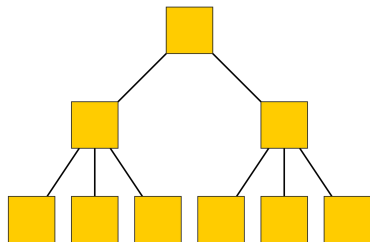
Il termine **relational database management system** (**RDBMS**, sistema per la gestione di **basi di dati** relazionali) indica un **database management system** basato sul **modello relazionale**, introdotto da **Edgar F. Codd**.

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**, **reticolo**

## Nota per programmatore:

in Python esistono diverse librerie che permettono l'accesso ai dati di un DB. Tra queste alcune convertono il db relazionale in classi python

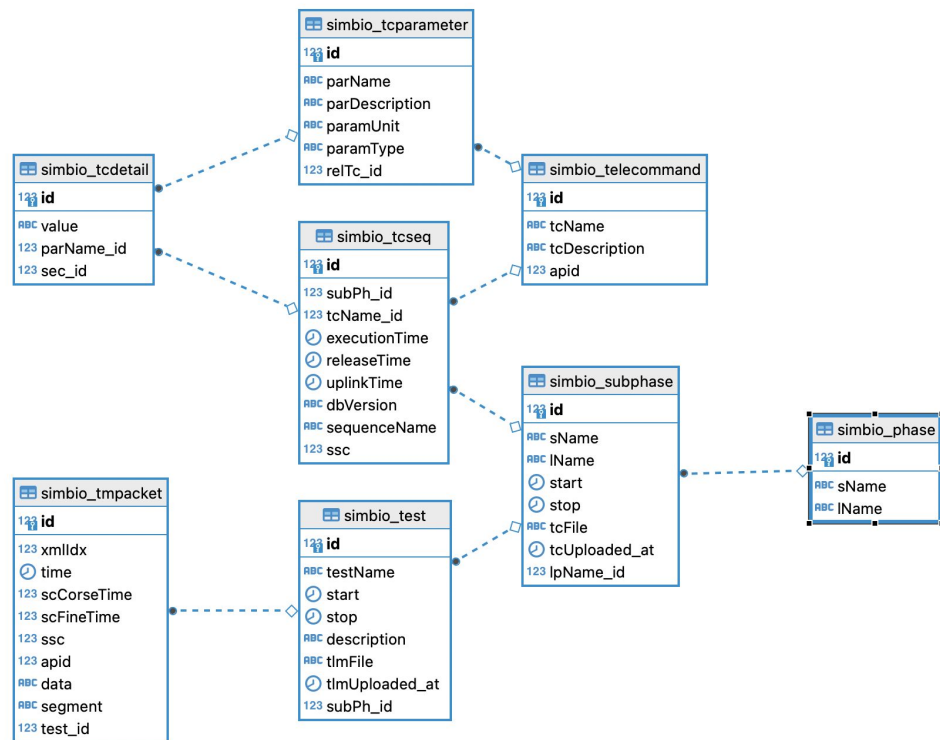
([Wikipedia](#))



dei vari database troverete che tra i relazionali e DB ad oggetti troverete che è presente PostgreSQL



# Diagrammi ER (Entità Relazione)



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>

<b>Schema</b>	<b>Relazione</b>	<b>Tupla</b>	<b>Attributo</b>
---------------	------------------	--------------	------------------

<b>Database</b>	<b>Tabella</b>	<b>Record</b>	<b>Campo</b>
-----------------	----------------	---------------	--------------

# Linguaggio SQL

Per fare qualche esempio di linguaggio SQL useremo SQLite.

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

Database	Tabella	Record	Campo
----------	---------	--------	-------

- Creazione (create per database e tabella, insert per record)
- Cancellazione (drop per database e tabella, delete per record)
- Aggiornamento (alter per database e tabella, update per record)
- Ricerca (o selezione) (select)



# Linguaggio SQL

Per fare qualche esempio di linguaggio SQL useremo SQLite.

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

Database	Tabella	Record	Campo
----------	---------	--------	-------

- Creazione (create per database e tabella, insert per record)
- Cancellazione (drop per database e tabella, delete per record)
- Aggiornamento (alter per database e tabella, update per record)
- **Ricerca (o selezione) (select)**

**Query**

# Select (1)

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

# Select (1)

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 (1)

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** testName, start, stop **FROM** simbio\_test;

## Select (1)

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** testName, start, stop **FROM** simbio\_test;

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

SQL> **SELECT** testName, start, stop **FROM** simbio\_test **WHERE** start > "2018-12-11"  
**AND** stop < "2018-12-12";

## Select (2)

**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 simbio_subphase.* FROM simbio_subphase, simbio_phase WHERE  
      simbio_phase.id = simbio_subphase.lpName_id AND  
      simbio_phase.sName="CRUISE" ORDER BY simbio_subphase.start;
```

# Esercizio

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

## Esercizio

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

```
SQL>SELECT * FROM simbio_telecommand WHERE  
simbio_telecommand.tcDescription LIKE "%VIHI science%";
```

```
[OUT] 306
```

```
SQL> SELECT * 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

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

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

SQL> **SELECT** \* from simbio\_tcpparameter **WHERE** parDescription **LIKE** "%VIHI integration%"

**[OUT]** 578

SQL> **SELECT** \* from simbio\_tcdetail **WHERE** sec\_id=9784 **AND** parName\_id=578

**[OUT]** 3

# Esercizio

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

```
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 "%VIHl%" 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 tcp  
WHERE tcp.parDescription LIKE "%VIHl%" and tcp.parDescription LIKE  
"%integration%") ORDER BY tseq.executionTime LIMIT 1
```

# Archiviazione (Parte 2)

---

# Archiviazione Ibrida

## **LTP**

File Storage

Archiviazione gerarchica

## **STP**

Object storage

Archiviazione piana

Database Relazionale

# Archiviazione Ibrida

## **LTP**

File Storage

Archiviazione gerarchica

## **STP**

Object storage

Archiviazione piana

Database Relazionale

# LTP File Storage

## **Dati:**

immagine



File binario

## **Metadati:**

Housekeeping

Puntamento

etc.



File XML

# Lezione 3

---

# 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



# 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.

```
<?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"?>
```

```
<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>
```

Preambolo

# 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>
```

Tag

# 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>
```

Tag

Elemento

```
</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>
```

Tag

Elemento

Attributo

# XML Schema Definition (XSD)

```
<?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>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema attributeFormDefault="qualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="utenti">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="utente" maxOccurs="unbounded" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element type="xs:string" name="nome"/>
              <xs:element type="xs:string" name="cognome"/>
              <xs:element type="xs:string" name="indirizzo"/>
            </xs:sequence>
            <xs:attribute type="xs:integer" name="anni" use="optional"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

# XML Schema Definition (XSD)

## Schema

**xmlns:xs** indica dove è definito il namespace degli elementi e degli attributi.

**elementFormDefault** indica che gli elementi usati in questo schema sono qualificati nel namespace

**attributeFormDefault** indica che gli elementi usati in questo schema sono qualificati nel namespace

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema attributeFormDefault="qualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="utenti">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="utente" maxOccurs="unbounded" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element type="xs:string" name="nome"/>
              <xs:element type="xs:string" name="cognome"/>
              <xs:element type="xs:string" name="indirizzo"/>
            </xs:sequence>
            <xs:attribute type="xs:integer" name="anni" use="optional"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```



# XML Schema Definition (XSD)

## Element

**name** indica il nome dell'elemento

**maxOccurs** indica il numero massimo che quell'elemento può apparire. unbounded vuol dire che non esiste un numero massimo.

**minOccurs** indica il numero minimo che quell'elemento può apparire

**type** indica il tipo di contenuto:

- xs:string
- xs:integer
- xs:decimal
- xs:boolean
- xs:date
- xs:time

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema attributeFormDefault="qualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="utenti">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="utente" maxOccurs="unbounded" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element type="xs:string" name="nome"/>
              <xs:element type="xs:string" name="cognome"/>
              <xs:element type="xs:string" name="indirizzo"/>
            </xs:sequence>
            <xs:attribute type="xs:integer" name="anni" use="optional"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

# XML Schema Definition (XSD)

## Element

**name** indica il nome dell'elemento

**maxOccurs** indica il numero massimo di volte che quell'elemento può apparire. vuol dire che non esiste un numero massimo.

**minOccurs** indica il numero minimo di volte che quell'elemento può apparire

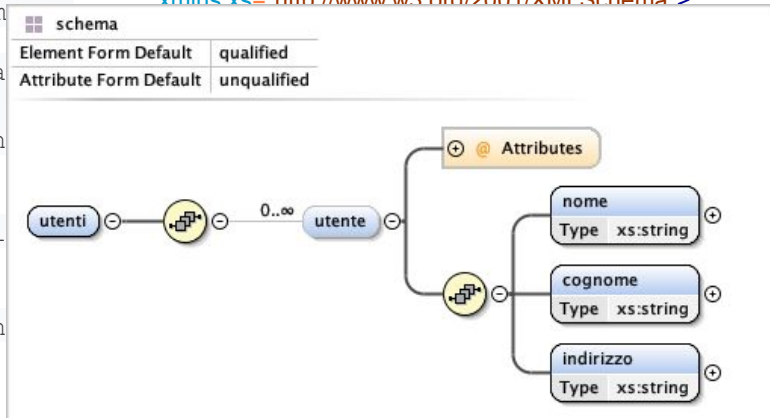
**type** indica il tipo di contenuto

- xs:string
- xs:integer
- xs:decimal
- xs:boolean
- xs:date
- xs:time

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<xs:schema attributeFormDefault="qualified" elementFormDefault="qualified"
```

```
xmlns:xs="http://www.w3.org/2001/XMLSchema">
```



```
</xs:sequence>  
</xs:complexType>  
</xs:element>  
</xs:schema>
```

ended" minOccurs="0">

e"/>  
nome"/>  
zzo"/>

use="optional"/>

# XML Schema Definition (XSD)

## Element

**name** indica il nome dell'elemento

**maxOccurs** indica il numero massimo che quell'elemento può apparire. unbounded vuol dire che non esiste un numero massimo.

**minOccurs** indica il numero minimo che quell'elemento può apparire

**type** indica il tipo di contenuto:

- xs:string
- xs:integer
- xs:decimal
- xs:boolean
- xs:date
- xs:time

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema attributeFormDefault="qualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="utenti">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="utente" maxOccurs="unbounded" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element type="xs:string" name="nome"/>
              <xs:element type="xs:string" name="cognome"/>
              <xs:element type="xs:string" name="indirizzo"/>
            </xs:sequence>
            <xs:attribute type="xs:integer" name="anni" use="optional"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Un tipo particolare è il **complexType**, questo indica che quell'elemento non è atomico ma formato da altri elementi

# XML Schema Definition (XSD)

## Attribute

**name** indica il nome dell'elemento

**use** indica se l'attributo è obbligatorio (required) o opzionale (optional)

**type** indica il tipo di contenuto:

- xs:string
- xs:integer
- xs:decimal
- xs:boolean
- xs:date
- xs:time

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema attributeFormDefault="qualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="utenti">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="utente" maxOccurs="unbounded" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element type="xs:string" name="nome"/>
              <xs:element type="xs:string" name="cognome"/>
              <xs:element type="xs:string" name="indirizzo"/>
            </xs:sequence>
            <xs:attribute type="xs:integer" name="anni" use="optional"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

# complexType (1)

Il complexType è in pratica un oggetto.

Posso definire il complexType esternamente dell'elemento che lo incapsula e utilizzarlo più volte.

# complexType (2)

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">

  <xs:element name="utenti">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="utente"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="utente">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="nome"/>
        <xs:element ref="cognome"/>
        <xs:element ref="indirizzo"/>
      </xs:sequence>
      <xs:attribute name="anni" use="required" type="xs:integer"/>
    </xs:complexType>
  </xs:element>

  <xs:element name="nome" type="xs:string"/>
  <xs:element name="cognome" type="xs:string"/>
  <xs:element name="indirizzo" type="xs:string"/>
</xs:schema>
```

## complexType (2)

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<xs:schema attributeFormDefault="qualified" elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

```
<xs:element name="utenti" type="utente" />
```

```
<xs:complexType name="utente" >
```

```
<xs:sequence>
```

```
<xs:element type="xs:string" name="nome" />
```

&lt;xs:element type="xs:string" name="cognome"/&gt;

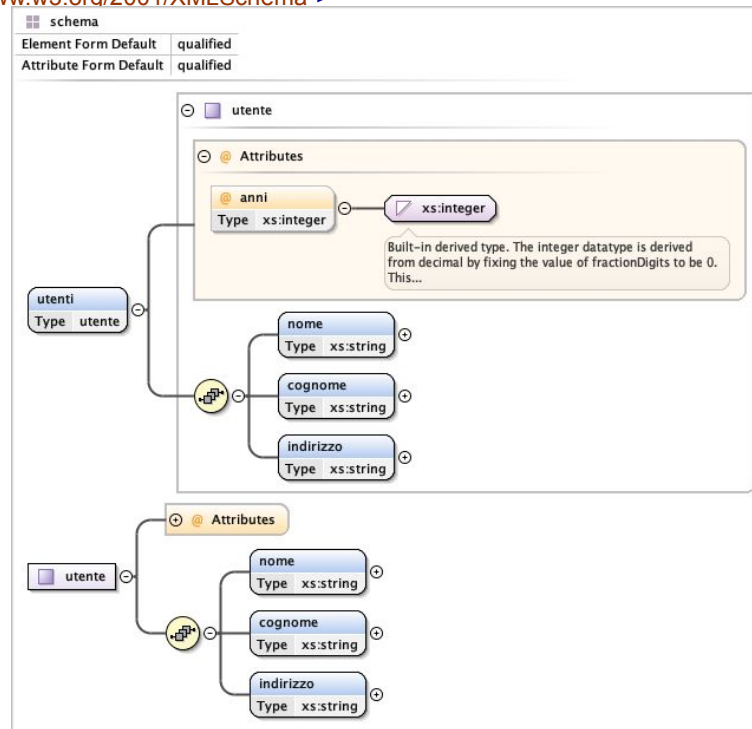
```
<xs:element type="xs:string" name="indirizzo"/>
```

&lt;/xs:sequence&gt;

```
<xs:attribute type="xs:integer" name="anni" use="optional"/>
```

&lt;/xs:complexType&gt;

&lt;/xs:schema&gt;



## complexType (3)

Il complexType è in pratica un oggetto.

Posso definire il complexType esternamente dell'elemento che lo incapsula e utilizzarlo più volte.

Può essere esteso.



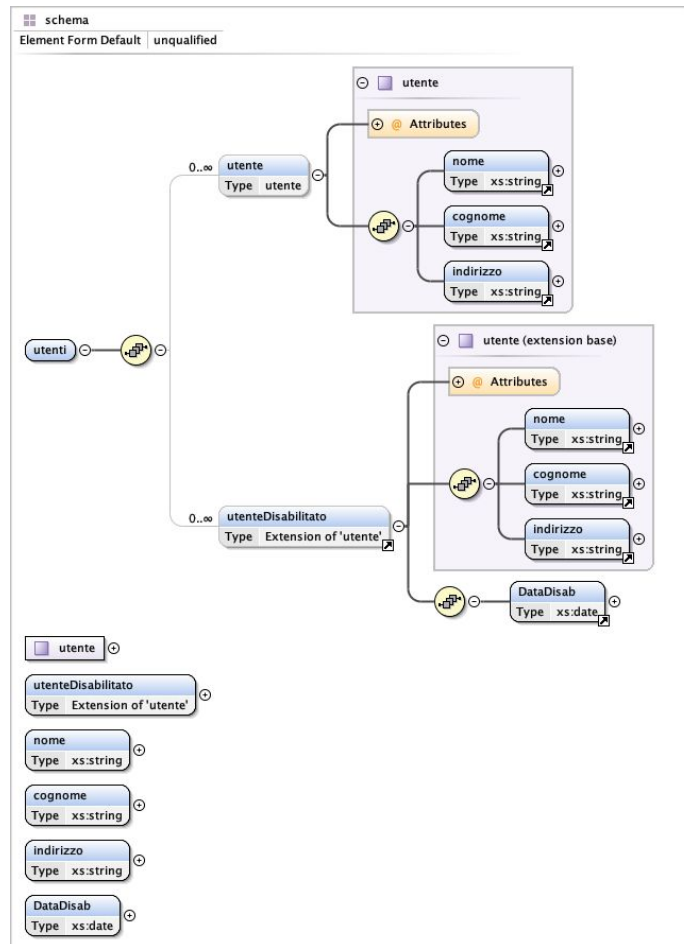
# complexType (4)

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified">
  <xs:element name="utenti">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" minOccurs="0" name="utente" type="utente"/>
        <xs:element maxOccurs="unbounded" minOccurs="0" ref="utenteDisabilitato"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="utente">
    <xs:sequence>
      <xs:element ref="nome"/>
      <xs:element ref="cognome"/>
      <xs:element ref="indirizzo"/>
    </xs:sequence>
    <xs:attribute name="anni" use="required" type="xs:integer"/>
  </xs:complexType>

  <xs:element name="utenteDisabilitato">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="utente">
          <xs:sequence>
            <xs:element ref="DataDisab"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>

  <xs:element name="nome" type="xs:string"/>
  <xs:element name="cognome" type="xs:string"/>
  <xs:element name="indirizzo" type="xs:string"/>
  <xs:element name="DataDisab" type="xs:date"/>
</xs:schema>
```



# complexType

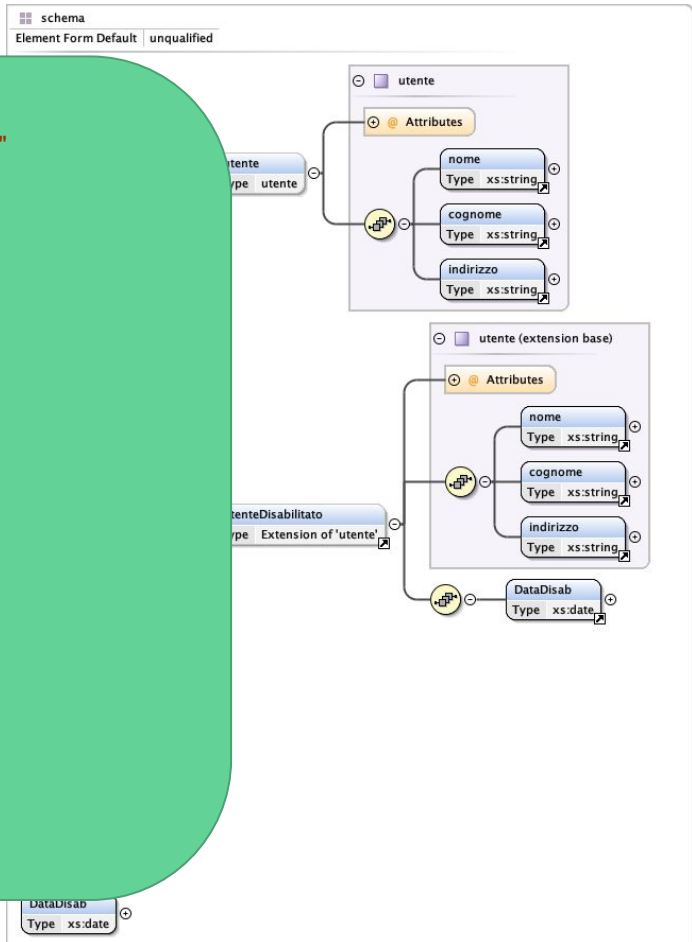
```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  <xs:element name="utenti">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" minOccurs="1" name="utente"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="utente">
    <xs:sequence>
      <xs:element ref="nome"/>
      <xs:element ref="cognome"/>
      <xs:element ref="indirizzo"/>
    </xs:sequence>
    <xs:attribute name="anni" use="required" type="xs:string"/>
  </xs:complexType>

  <xs:element name="utenteDisabilitato">
    <xs:complexType>
      <xs:complexContent>
        <xs:extension base="utente">
          <xs:sequence>
            <xs:element ref="DataDisab"/>
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
  </xs:element>

  <xs:element name="nome" type="xs:string"/>
  <xs:element name="cognome" type="xs:string"/>
  <xs:element name="indirizzo" type="xs:string"/>
  <xs:element name="DataDisab" type="xs:date"/>
</xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<utenti xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="nm/schema02.xsd">
  <utente anni="20">
    <nome>nome0</nome>
    <cognome>cognome0</cognome>
    <indirizzo>indirizzo0</indirizzo>
  </utente>
  <utente anni="30">
    <nome>nome2</nome>
    <cognome>cognome2</cognome>
    <indirizzo>indirizzo2</indirizzo>
  </utente>
  <utenteDisabilitato anni="40">
    <nome>mome3</nome>
    <cognome>cognome3</cognome>
    <indirizzo>indirizzo3</indirizzo>
    <DataDisab>2021-02-10</DataDisab>
  </utenteDisabilitato>
</utenti>
```



# Fondamenti di programmazione

---

# Fondamenti di Python

**Linguaggio:** Python 3.9.5

**Ambiente di sviluppo:** Microsoft Visual Studio Code

# Indice

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

## Packages:

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

# Elementi di base

Carattere per commento di linea: #

Commento multilinea :

```
'''
```

```
...
```

```
'''
```

Indentatura

# Header del file

```
#!/usr/bin/env python3
'''
    Module description
'''
__author__ = 'Romolo Politi'
__copyright__ = 'Copyright 2021, Test Project'
__credits__ = ['Romolo Politi', 'Antonio Franco', 'Fulvio Sarcinella']
__license__ = 'GNU GPLv3'
__version__ = '0.1.0'
__maintainer__ = 'Romolo Politi'
__email__ = 'Romolo.Politi@inaf.it'
__status__ = 'Prototype'
__date__ = "2021/06/22"

import os

print("Hello, World!")
```

# Header del file (modulo)

```
#!/usr/bin/env python3
'''
    Module description
'''
__author__ = 'Romolo Politi'
__copyright__ = 'Copyright 2021, Test Project'
__credits__ = ['Romolo Politi', 'Antonio Franco', 'Fulvio Sarcinella']
__license__ = 'GNU GPLv3'
__version__ = '0.1.0'
__maintainer__ = 'Romolo Politi'
__email__ = 'Romolo.Politiinaf.it'
__status__ = 'Prototype'
__date__ = "2021/06/22"

import os

print("Hello, World!")
```

Un 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.



# Header del file (variabili)

```
#!/usr/bin/env python3
'''
    Module description
'''
__author__ = 'Romolo Politi'
__copyright__ = 'Copyright 2021, Test Project'
__credits__ = ['Romolo Politi', 'Antonio Franco', 'Fulvio Sarcinella']
__license__ = 'GNU GPLv3'
__version__ = '0.1.0'
__maintainer__ = 'Romolo Politi'
__email__ = 'Romolo.Politiinaf.it'
__status__ = 'Prototype'
__date__ = "2021/06/22"

import os

print("Hello, World!")
```

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.

# Header del file (variabili)

```
#!/usr/bin/env python3
'''
    Module description
'''
__author__ = 'Romolo Politi'
__copyright__ = 'Copyright'
__credits__ = ['Romolo Politi']
__license__ = 'GNU GPLv3'
__version__ = '0.1.0'
__maintainer__ = 'Romolo Politi'
__email__ = 'Romolo.Politi@unipi.it'
__status__ = 'Prototype'
__date__ = "2021/06/22"

import os

print("Hello, World!")
```

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

abili. Vengono

# Header del file(variabili)

```
#!/usr/bin/env python3
'''
    Module description
'''
__author__ = 'Romolo Politi'
__copyright__ = 'Copyright 2021, Test Project'
__credits__ = ['Romolo Politi', 'Antonio Franco', 'Fulvio Sarcinella']
__license__ = 'GNU GPLv3'
__version__ = '0.1.0'
__maintainer__ = 'Romolo Politi'
__email__ = 'Romolo.Politiinaf.it'
__status__ = 'Prototype'
__date__ = "2021/06/22"

import os

print("Hello, World!")
```

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. (*type(var)*)

# Esempio variabili

## Example

```
x = "Hello World"
```

```
x = 20
```

```
x = 20.5
```

```
x = 1j
```

```
x = ["apple", "banana", "cherry"]
```

```
x = ("apple", "banana", "cherry")
```

```
x = range(6)
```

```
x = {"name": "John", "age": 36}
```

```
x = {"apple", "banana", "cherry"}
```

```
x = frozenset({"apple", "banana", "cherry"})
```

```
x = True
```

```
x = b"Hello"
```

```
x = bytearray(5)
```

```
x = memoryview(bytes(5))
```

## Data Type

str

int

float

complex

list

tuple

range

dict

set

frozenset

bool

bytes

bytearray

memoryview

# Header del file

```
#!/usr/bin/env python3
'''
    Module description
'''
__author__ = 'Romolo Politi'
__copyright__ = 'Copyright 2021, Test Project'
__credits__ = ['Romolo Politi', 'Antonio Franco', 'Fulvio Sarcinella']
__license__ = 'GNU GPLv3'
__version__ = '0.1.0'
__maintainer__ = 'Romolo Politi'
__email__ = 'Romolo.Politiinaf.it'
__status__ = 'Prototype'
__date__ = "2021/06/22"

import os

print("Hello, World!")
```

## Vesioning:

### MAJOR.MINOR.PATCH

1. versione MAJOR quando modificate l'API in modo non retrocompatibile,
2. versione MINOR quando aggiungete funzionalità in modo retrocompatibile, e
3. versione PATCH quando correggete bug in modo retrocompatibile.

[Versionamento Semantico 2.0.0](#)

# Definizione di Classe ed Oggetto

## Object-Oriented Programming (OOP) Vocabulary

### **Class**

a blueprint which is consisting of methods and attributes.

### **Object**

an instance of a class. It can help to think of objects as something in the real world like a yellow pencil, a small dog, a yellow shoe, etc. However, objects can be more abstract.

### **Attribute**

a descriptor or characteristic. Examples would be colour, length, size, etc. These attributes can take on specific values like blue, 3 inches, large, etc.

### **Method**

an action that a class or object could take.

# Definizione di Classe ed Oggetto

## Object-Oriented Programming (OOP) Vocabulary

### **Class**

a blueprint which is consisting of methods and attributes.

### **Object**

an instance of a class. It can help to think of objects as something in the real world like a yellow pencil, a small dog, a yellow shoe, etc. However, objects can be more abstract.

### **Attribute**

a descriptor or characteristic. Examples would be colour, length, size, etc. These attributes can take on specific values like blue, 3 inches, large, etc.

### **Method**

an action that a class or object could take.

<https://www.honeybadger.io/blog/python-instantiation-metaclass/>

# Dichiarazione di condizione (condition statement)

if condizione (True/False):

statement

else:

statement



# Dichiarazione di condizione (condition statement)

if condizione (True/False):

statement

else:

statement

if condizione (True/False):

statement

elif condizione:

statement

else:

statement

# Logical Conditions

- Uguaglianza:  $a == b$
- Non Uguaglianza:  $a != b$
- Minore di:  $a < b$
- Minore o uguale a :  $a <= b$
- Maggiore di:  $a > b$
- Maggiore o uguale a:  $a >= b$

# Logical Conditions

- Uguaglianza: `a == b`
- ~~Non Uguaglianza: `a != b`~~
- Minore di: `a < b`
- Minore o uguale a : `a <= b`
- Maggiore di: `a > b`
- Maggiore o uguale a: `a >= b`

Si preferisce l'uso della negazione:

```
not a == b
```

# Logical Conditions

- Uguaglianza: `a == b`
- ~~Non Uguaglianza: `a != b`~~
- Minore di: `a < b`
- Minore o uguale a : `a <= b`
- Maggiore di: `a > b`
- Maggiore o uguale a: `a >= b`

Si preferisce l'uso della negazione:

```
not a == b
```

Le condizioni possono essere concatenate:

```
a == b and not c == 0
```

```
a == b or not c == 0
```

# Logical Conditions

- Uguaglianza: `a == b`
- ~~Non Uguaglianza: `a != b`~~
- Minore di: `a < b`
- Minore o uguale a : `a <= b`
- Maggiore di: `a > b`
- Maggiore o uguale a: `a >= b`

Si preferisce l'uso della negazione:

```
not a == b
```

Le condizioni possono essere concatenate:

```
a == b and not c == 0
```

```
a == b or not c == 0
```

Operatore **in**:

```
a in [0,1]
```

# Formato compatto

Può essere utilizzato un formato compatto:

```
if a > b: print("a is greater than b")
```

Oppure:

```
print("A") if a > b else print("=") if a == b else print("B")
```

# Match

```
match subject:

    case <pattern_1>:

        <action_1>

    case <pattern_2>:

        <action_2>

    case <pattern_3>:

        <action_3>

    case _:

        <action_wildcard>
```

```
a=1
match subject:
    case 1:
        print("One")
    case 2:
        print("Two")
    case 3:
        print("Three")
    case _:
        print(f"The value {a} is not valid")
```

# Match

```
match subject:
```

```
    case <pattern_1>:
```

```
        <action_1>
```

```
    case <pattern_2>:
```

```
        <action_2>
```

```
    case <pattern_3>:
```

```
        <action_3>
```

```
    case _:
```

```
        <action_wildcard>
```

**NB:** Questo statement sarà disponibile dalla versione 3.10

```
a=1
```

```
match subject:
```

```
    case 1:
```

```
        print("Two")
```

```
    case 3:
```

```
        print("Three")
```

```
    case _:
```

```
        print(f"The value {a} is not valid")
```



# Cicli

While

for

Esegue un blocco di codice finché la condizione rimane vera.

# Cicli

## While

Esegue un blocco di codice finché la condizione rimane vera.

```
import time
a=10
while a>0:
    print(f"Countdown: {a:>2}", end="\r")
    a-=1
    time.sleep(0.2)
```

## for

# Cicli

while

Esegue un blocco di codice finché la condizione rimane vera.

```
import time
a=10
while a>0:
    print(f"Countdown: {a:>2}", end="\r")
    a-=1
    time.sleep(0.2)
```

for

Vedi notebook 03 Esempio While

# Cicli

while

Esegue un blocco di codice finché la condizione rimane vera.

```
import time
a=10
while a>0:
    print(f"Countdown: {a:>2}", end="\r")
    a-=1
    time.sleep(0.2)
```

Vedi notebook 03 Esempio While

for

Esegue un blocco di codice iterandolo su di una sequenza.

# Cicli

## while

Esegue un blocco di codice finché la condizione rimane vera.

```
import time
a=10
while a>0:
    print(f"Countdown: {a:>2}", end="\r")
    a-=1
    time.sleep(0.2)
```

## for

Esegue un blocco di codice iterandolo su di una sequenza.

```
frutta = ["mele", "pere", "banane"]
for elem in frutta:
    print(f"Mi piacciono le {elem}")
```

Vedi notebook 03 Esempio While

# Cicli

## while

Esegue un blocco di codice finché la condizione rimane vera.

```
import time
a=10
while a>0:
    print(f"Countdown: {a:>2}", end="\r")
    a-=1
    time.sleep(0.2)
```

Vedi notebook 03 Esempio While

## for

Esegue un blocco di codice iterandolo su di una sequenza.

```
frutta = ["mele", "pere", "banane"]
for elem in frutta:
    print(f"Mi piacciono le {elem}")
```

Vedi notebook 04 Esempio for

# Cicli

Uscita:

**continue** Interrompe l'esecuzione e passa all'elemento successivo.

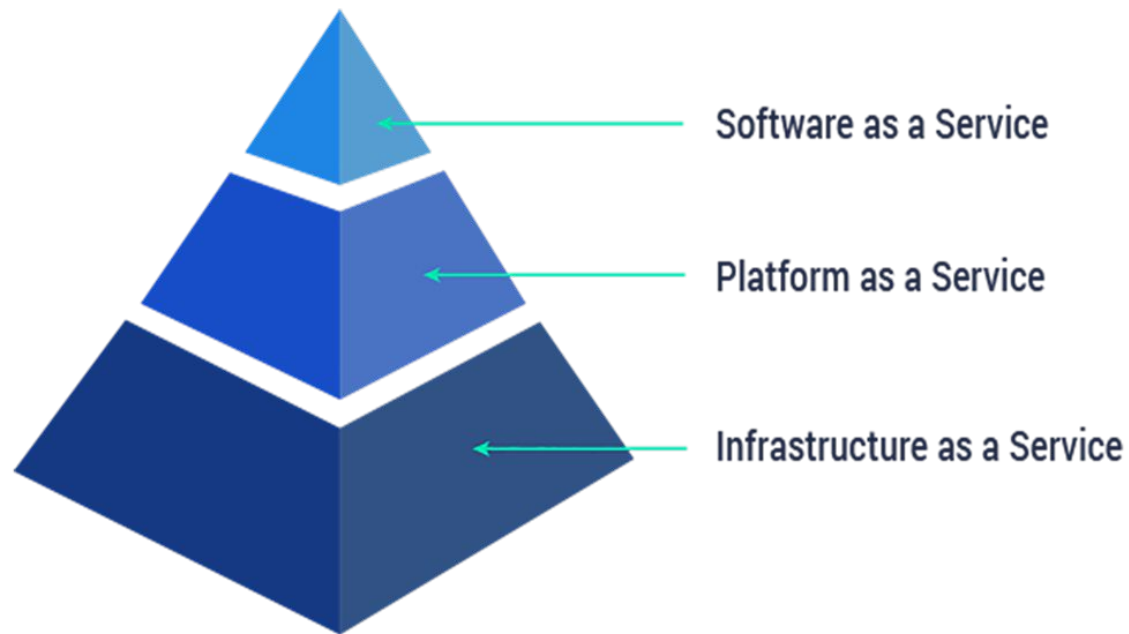
**break** Interrompe il ciclo e passa alla prima istruzione fuori del ciclo.

# Lezione 4

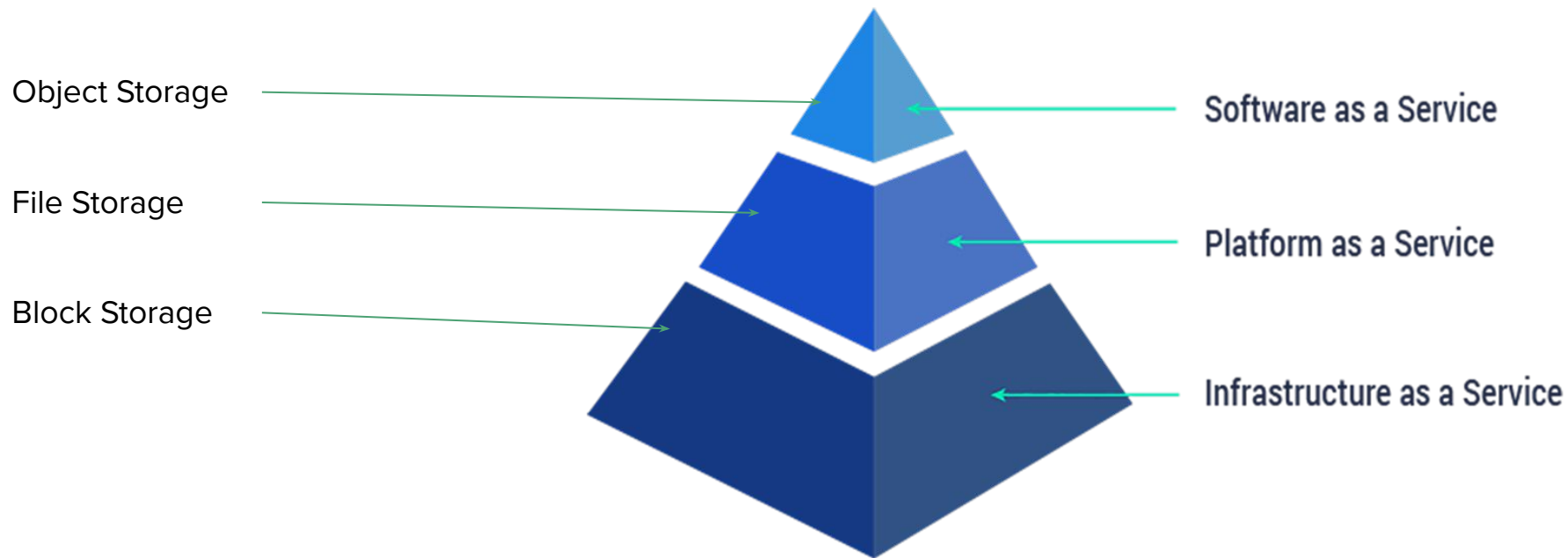
---



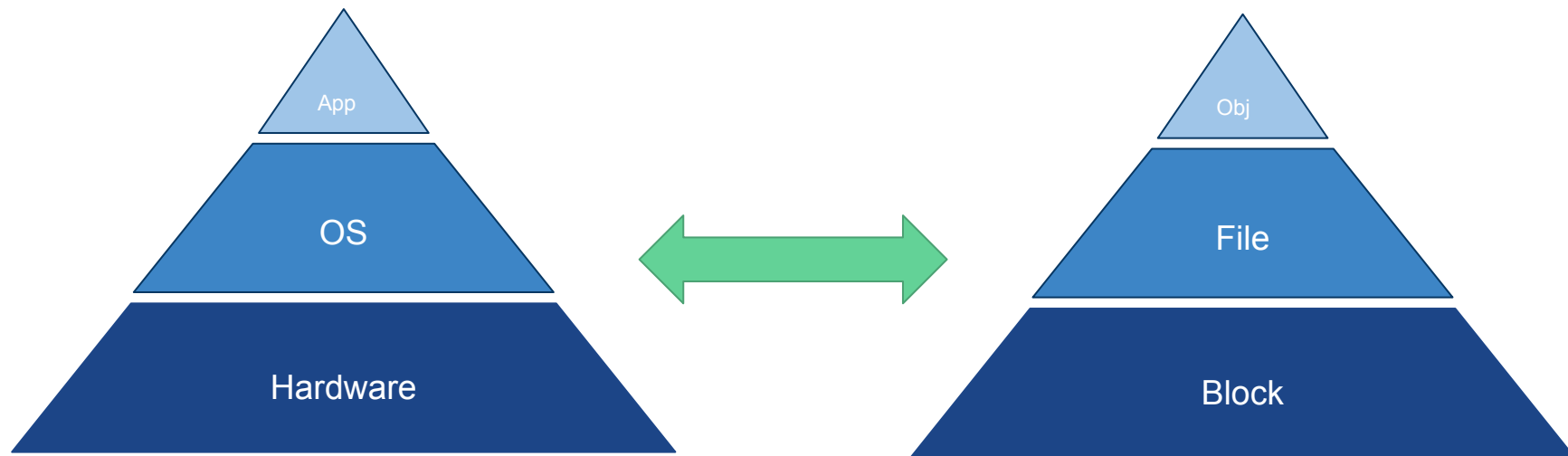
# Chiarificazioni Storage



# Chiarificazioni Storage



# Chiarificazioni Storage



# 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

# Indice

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

## Packages:

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

# Funzioni

La funzione è un blocco di codice che viene eseguito su comando.

# Funzioni

La funzione è un blocco di codice che viene eseguito su comando.

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

# Funzioni

La funzione è 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:



# Funzioni

La funzione è un blocco di codice che viene eseguito su comando.

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

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

```
def myFunc(arg, parl = str):  
    print(f"Hello {arg}")  
    if parl:  
        print("Oggi è una bella giornata")  
  
myFunc("Antonio")  
myFunc("Antonio", True)  
myFunc("Antonio", parl = True)
```

# 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.

# Funzioni - Decorazioni

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.

# Lo spazio dei nomi - Namespace

Con la parola **namespace in Python** si intende un dizionario Python che contiene i nomi delle variabili (chiavi) ed i valori di tali variabili (valori) in modo da tenere traccia delle variabili utilizzate in quel particolare contesto. In generale, uno spazio dei nomi (a volte chiamato anche contesto) è un sistema di denominazione per creare dei nomi univoci onde evitare ambiguità.

# Lo spazio dei nomi - Namespace

Con la parola **namespace in Python** si intende un dizionario Python che contiene i nomi delle variabili (chiavi) ed i valori di tali variabili (valori) in modo da tenere traccia delle variabili utilizzate in quel particolare contesto. In generale, uno spazio dei nomi (a volte chiamato anche contesto) è un sistema di denominazione per creare dei nomi univoci onde evitare ambiguità.

I principali **namespace in Python**

- **global names**
- **local names**
- **built-in names**



è il namespace delle funzioni ed operatori di Python

# Lo spazio dei nomi - Namespace

Con la parola **namespace in Python** si intende un dizionario Python che contiene i nomi delle variabili (chiavi) ed i valori di tali variabili (valori) in modo da tenere traccia delle variabili utilizzate in quel particolare contesto. In generale, uno spazio dei nomi (a volte chiamato anche contesto) è un sistema di denominazione per creare dei nomi univoci onde evitare ambiguità.

I principali **namespace in Python**

- **global names**
- **local names**
- **built-in names**



è il namespace interno alle singole funzioni e classi

# Lo spazio dei nomi - Namespace

Con la parola **namespace in Python** si intende un dizionario Python che contiene i nomi delle variabili (chiavi) ed i valori di tali variabili (valori) in modo da tenere traccia delle variabili utilizzate in quel particolare contesto. In generale, uno spazio dei nomi (a volte chiamato anche contesto) è un sistema di denominazione per creare dei nomi univoci onde evitare ambiguità.

I principali **namespace in Python**

- **global names**
- **local names**
- **built-in names**



è il namespace del modulo

# Lo spazio dei nomi - Namespace

Quando si importa un modulo/package si importa anche il namespace.

Per tale motivo per utilizzare le funzioni importate si deve premettere il nome del modulo, se non diversamente specificato.



# Lo spazio dei nomi - Namespace

Quando si importa un modulo/package si importa anche il namespace.

Per tale motivo per utilizzare le funzioni importate si deve premettere il nome del modulo, se non diversamente specificato.

```
import pandas
```

```
df = pandas.DataFrame()
```

# Lo spazio dei nomi - Namespace

Quando si importa un modulo/package si importa anche il namespace.

Per tale motivo per utilizzare le funzioni importate si deve premettere il nome del modulo, se non diversamente specificato.

```
import pandas  
import pandas as pd
```

```
df = pandas.DataFrame()  
df = pd.DataFrame()
```

# Lo spazio dei nomi - Namespace

Quando si importa un modulo/package si importa anche il namespace.

Per tale motivo per utilizzare le funzioni importate si deve premettere il nome del modulo, se non diversamente specificato.

```
import pandas
import pandas as pd
from pandas import DataFrame
```

```
df = pandas.DataFrame()
df = pd.DataFrame()
df = DataFrame()
```

# Lo spazio dei nomi - Namespace

Quando si importa un modulo/package si importa anche il namespace.

Per tale motivo per utilizzare le funzioni importate si deve premettere il nome del modulo, se non diversamente specificato.

```
import pandas
import pandas as pd
from pandas import DataFrame
from pandas import *
```

```
df = pandas.DataFrame()
df = pd.DataFrame()
df = DataFrame()
```

# 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
```

# 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

[\(https://www.python.org/dev/peps/pep-0008/\)](https://www.python.org/dev/peps/pep-0008/)



# 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

[\(https://www.python.org/dev/peps/pep-0008/\)](https://www.python.org/dev/peps/pep-0008/)

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

```
#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

[\(https://www.python.org/dev/peps/pep-0008/\)](https://www.python.org/dev/peps/pep-0008/)

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

3. Quando l'uso di più linee di codice rendono il codice più leggibile

# Quando non Usare le Lambda

3. Quando l'uso di più linee di codice rendono il codice più leggibile

```
df['Gender'] = df['Status'].map(lambda x: 'Male' if x=='father' or x=='son' else 'Female')
```

# Quando non Usare le Lambda

## 3. Quando l'uso di più linee di codice rendono il codice più leggibile

```
df['Gender'] = df['Status'].map(lambda x: 'Male' if x=='father' or x=='son' else 'Female')
```

```
df['Gender'] = ''  
df.loc[(df['Status'] == 'Father') | (df['Status'] == 'Son'), 'Gender'] = 'Male'  
df.loc[(df['Status'] == 'Mother') | (df['Status'] == 'Daughter'), 'Gender'] = 'Female'
```

# Input e Output

Lo standard output è lo schermo.       $\Rightarrow$     `print()`

Lo standard input è la tastiera.       $\Rightarrow$     `input()`

# Input e Output

Lo standard output è lo schermo.  $\Rightarrow$  `print()`

Lo standard input è la tastiera.  $\Rightarrow$  `input()`

L'output su file:

- apertura del device  $\Rightarrow$  `open(<nomefile>,<modalità>)`
- lettura /scrittura  $\Rightarrow$  `read, readline, readlines / write`
- chiusura del device  $\Rightarrow$  `close()`

# Eccezioni

**Le eccezioni sono eventi scatenati da errori di varia natura.**

- Quando prevediamo che in un dato contesto si possa verificare un'eccezione, possiamo scrivere del codice per “gestirla” (in inglese *to handle*).  
Si parla dunque di *handled exception*.
- Se invece un'eccezione si verifica in un contesto imprevisto, non viene “gestita” (*unhandled exception*, eccezione non gestita) e quindi causerà l'uscita dal blocco di codice in cui si presenta.

Se però il blocco di codice più esterno ha “previsto” l'eccezione scatenata, potrà occuparsene.



# Eccezioni

Il meccanismo di gestione delle eccezioni è il seguente:

try:

    <statement>

except [<nome eccezione>]:

    <statement>

else:

    <statement>

finally:

    <statement>

# Operatore di contesto

```
with EXPRESSION as TARGET:  
    SUITE
```

Espressioni semanticamente equivalenti

```
manager = (EXPRESSION)  
enter = type(manager).__enter__  
exit = type(manager).__exit__  
value = enter(manager)  
hit_except = False  
  
try:  
    TARGET = value  
    SUITE  
except:  
    hit_except = True  
    if not exit(manager, *sys.exc_info()):  
        raise  
finally:  
    if not hit_except:  
        exit(manager, None, None, None)
```

# Package e Moduli

Programmi Python possono essere organizzati in files, contenuti in una gerarchia di directory.

Un file costituisce un **Modulo**, un insieme di istruzioni e dati auto-consistente.

Il nome del file è il nome del modulo, senza il suffisso **".py"** . Ma i moduli possono essere in un formato compresso (con zip), ed in questo caso hanno suffisso **".egg"** (da Python 2.6), oppure possono essere files compilati, scritti con un altro linguaggio, ed in questo caso hanno suffisso **".so"** .

I nomi dei file contenenti i moduli sono soggetti alle stesse regole dei nomi di variabili, infatti Python usa il nome del file, senza suffisso, come riferimento al modulo. Per cui nomi con spazi o caratteri speciali non sono accettati.

I file dei moduli possono iniziare con una stringa che descrive il modulo e viene conservata nell'attributo **\_\_doc\_\_** del modulo stesso. Un dizionario con tutti gli oggetti, le variabili e le funzioni del modulo è messo da Python nell'attributo **\_\_dict\_\_** del modulo.

Diversi moduli sono organizzati in **"packages"**, che occupano una directory. Il nome del package e' il nome della directory. Per essere considerata un package una directory deve contenere un file di nome **\_\_init\_\_.py** , che puo' anche essere vuoto, ma in genere contiene istruzioni che che inizializzano il package.

Un package può contenere subpackages, in sottodirectory.

# PyPI (Python Package Index)

PyPi è il repository che contiene tutti i package di python che possono essere installati tramite il modulo pip

```
python3 -m pip install <nome del package>
```

<http://pypi.org>

```
packaging tutorial/  
├── LICENSE  
├── pyproject.toml  
├── README.md  
├── setup.cfg  
├── src/  
│   └── example package/  
│       ├── init .py  
│       └── example.py  
└── tests/
```

# PyPI

pyproject.toml

Contiene la lista dei tool di costruzione del package. Solitamente non viene modificato

```
[build-system]  
requires = [  
    "setuptools>=42",  
    "wheel"  
]  
build-backend = "setuptools.build_meta"
```

# PyPI

pyproject.toml

Contiene la lista dei tool di cui  
è modificato

setup.cfg o **setup.py**

```
import setuptools

with open("README.md", "r") as fh:
    long_description = fh.read()

setuptools.setup(
    name="SCOSpy",
    version="0.4.4",
    author="Romolo Politi",
    author_email="Romolo.Politi@inaf.it",
    description="Python library to read the SCOS-2000 header",
    license="General Public License v3 (GPLv3)",
    long_description=long_description,
    long_description_content_type="text/markdown",
    url="https://www.ict.inaf.it/gitlab/romolo.politi/scospy",
    packages=setuptools.find_packages(),
    install_requires=[
        'bitstring',
    ],
    classifiers=[
        "Programming Language :: Python :: 3.9",
        "License :: OSI Approved :: GNU General Public License v3 (GPLv3)",
        "Operating System :: OS Independent",
    ],
    python_requires='>=3.6',
)
```



# Markdown

Markdown è un linguaggio di markup leggero e non invasivo che permette di aggiungere elementi di formattazione a documenti di testo in chiaro.

# Delivery su PyPI

```
python3 -m venv <DIR>  
source <DIR>/bin/activate
```

## Costruiamo il nostro pacchetto

```
python3 -m build
```

## Effettuiamo l'upload (su testPyPi)

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

## Installiamo il pacchetto su un ambiente virtuale per test

```
python3 -m pip install --index-url https://test.pypi.org/simple/ --no-deps expkg[YOUR-USERNAME]
```

## Effettuiamo l'upload definitivo

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



# Python Hand on

---

# Pianificazione del progetto

Ambiente di sviluppo: Microsoft Visual Code

Software di versioning: Git

Repository: GitHub

# Creiamo un Repository su GitHub

Attiviamo un account GitHub



 Overview

 **Repositories** 1

 Projects

 Packages

Type ▾

Language ▾

Sort ▾

 New

# Creiamo un Repository su GitHub

## Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository](#).

Owner \*



RomoloPoliti-INAF ▾

Repository name \*

/

Great repository names are short and memorable. Need inspiration? How about **vigilant-waffle**?

Description (optional)



**Public**

Anyone on the internet can see this repository. You choose who can commit.



**Private**

You choose who can see and commit to this repository.

### Initialize this repository with:

Skip this step if you're importing an existing repository.

☐ **Add a README file**

This is where you can write a long description for your project. [Learn more](#).

☐ **Add .gitignore**

Choose which files not to track from a list of templates. [Learn more](#).

☐ **Choose a license**

A license tells others what they can and can't do with your code. [Learn more](#).

Create repository

# Inizializziamo il repository in locale

```
echo "# PhDCourse2021" >> README.md
```

```
git init
```

```
git add README.md
```

```
git commit -m "first commit"
```

```
git branch -M main
```

```
git remote add origin https://github.com/RomoloPoliti-INAF/PhDCourse2021.git
```

```
git push -u origin main
```

# Comandi principali Git (con non rigorosa spiegazione)

Tutti i comandi sono preceduti dall'interprete 'git'

**clone:** duplica il repository remoto in locale

**status:** mostra lo stato del nostro repository locale

**add:** aggiunge uno o più file all'*indice*, che non è altro che un'istantanea del nostro repository.

**commit:** impacchetta l'indice (creando un commit packet) per essere caricato sul repository remoto.

**push:** carica il commit sul repository remoto

**pull:** scarica l'ultimo commit sul repository locale

# .gitignore

Accade che alcuni file non debbano essere caricati sul repository remoto:

- cartelle create da un package manager come NPM o bower
- directory contenenti i file da caricare su un server per un'applicazione web
- File di supporto generati dal sistema operativo o da altri strumenti
- file eseguibili generati a partire dal codice sorgente
- archivi compressi come file.gz o.zip

In questo caso viene creato un file .gitignore con l'elenco di questi file.

.gitignore accetta espressioni RegEx

# Lezione 5

---



# Indice

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

## Packages:

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

# Argomenti in Python

Il modulo `sys` gestisce gli argomenti da linea di comando e li archivia nella lista `argv`.

```
./codEx02.py Romolo
```

```
argv[0] ⇒ ./codEx02.py
```

```
argv[1] ⇒ Romolo
```

Consigliato l'uso del package **argparse**

# Hand on Python

Esempi sui codici della cartella bin e DataFrame (Jupyter Notebook)

# Esercizio

dato il file SunPathDaily\_40.3350551\_18.1112515\_1624612273248.csv (in data)  
creare un software (sul modello di codEx07) che legge il file e crea un grafico  
Azimut vs Elevazione.

# Lezione 6

---

# Esempio di Manipolazione di DataFrame

Vedi Jupyter file: Gantt Interchannel SIMBIO-SYS

# Progettiamo un codice

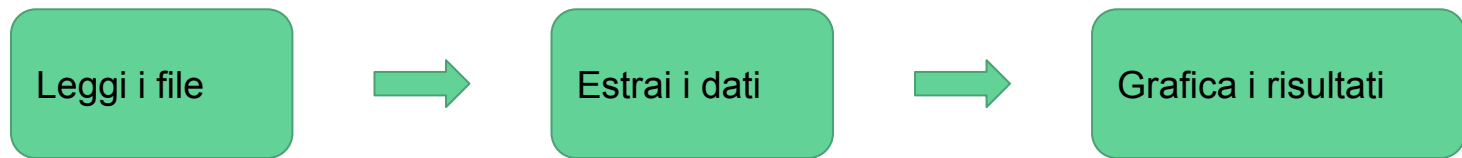
Diagramma a Blocchi

Definiamo le macro operazioni che il software deve compiere

# Progettiamo un codice

Diagramma a Blocchi

Definiamo le macro operazioni che il software deve compiere





# Progettiamo un codice

## Diagramma a Blocchi

Definiamo le macro operazioni che il software deve compiere

## Diagramma di flusso

Definiamo il flusso di ogni singolo blocco

# Progettiamo un codice

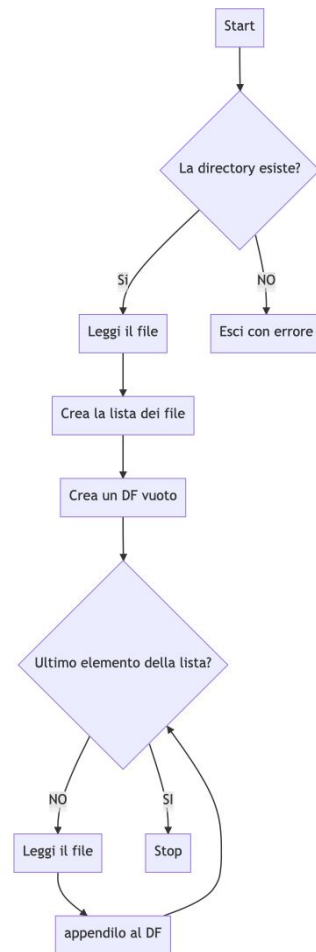
## Diagramma a Blocchi

Definiamo le macro operazioni che il software deve compiere

## Diagramma di flusso

Definiamo il flusso di ogni singolo blocco

Leggi i file



# Progettiamo un codice

## Diagramma a Blocchi

Definiamo le macro operazioni che il software deve compiere

## Diagramma di flusso

Definiamo il flusso di ogni singolo blocco

## Documentazione

# Jupyter per app

<https://www.youtube.com/watch?v=VtchVpoSdoQ>

# Profiling di un codice

Per eseguire il profile di un codice Python si può utilizzare il modulo cProfile

```
$python3 -m cProfile -o output.dat mioProgramma.py
```

Il file di output è un binario. Può essere visualizzato utilizzando il modulo snakeviz

```
$snakeviz output.dat
```