Cloud Archive and Computation

Romolo Politi

PhD Course 2023

Elenco Lezioni

Elenco Lezioni

- 01/06/2023
- 08/06/2023
- 20/06/2023

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/PhDCourse2023
- Per gli esempi utilizzeremo Python 3.11.1
- 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.

Lezione del 01/06/2023

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

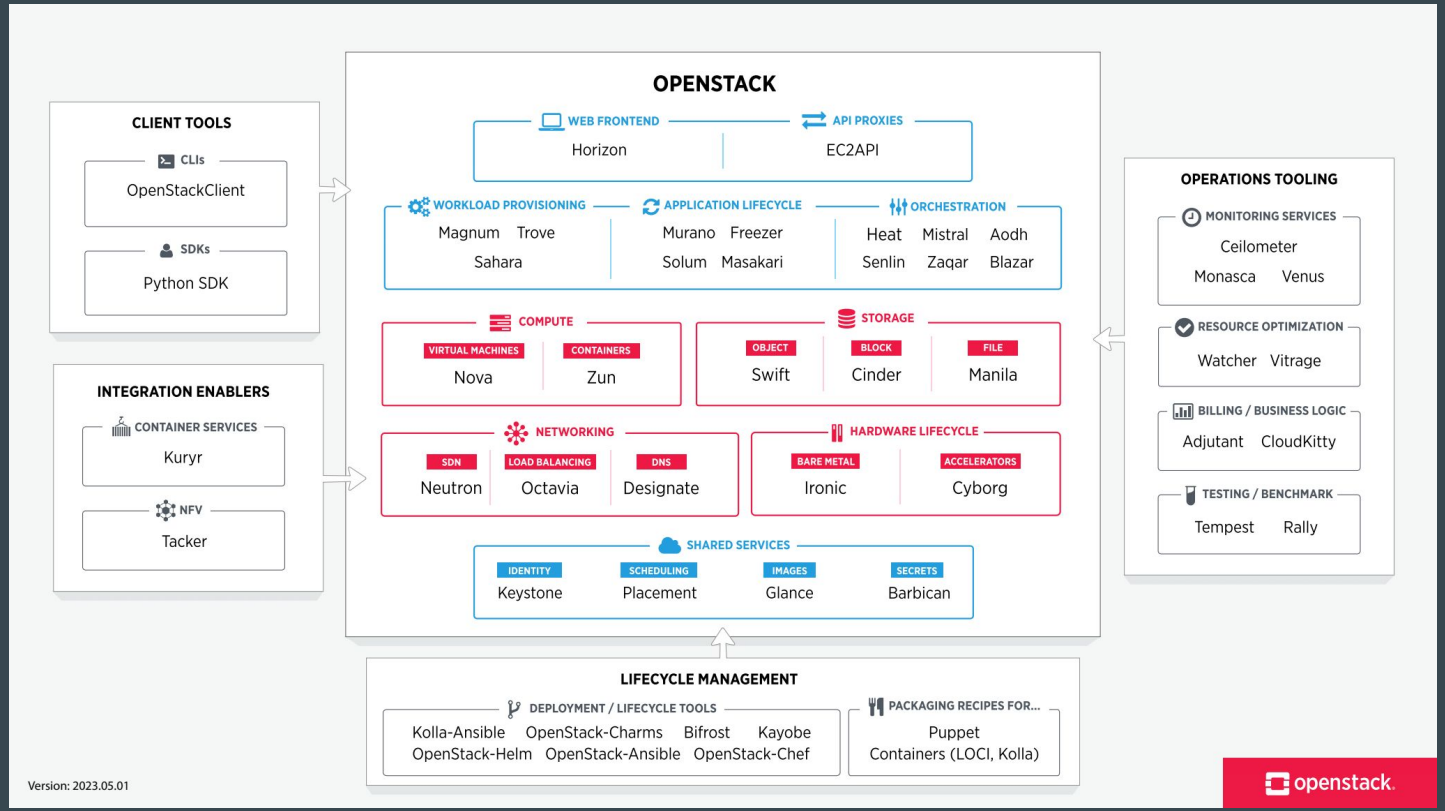
Out Promise

Google Cloud

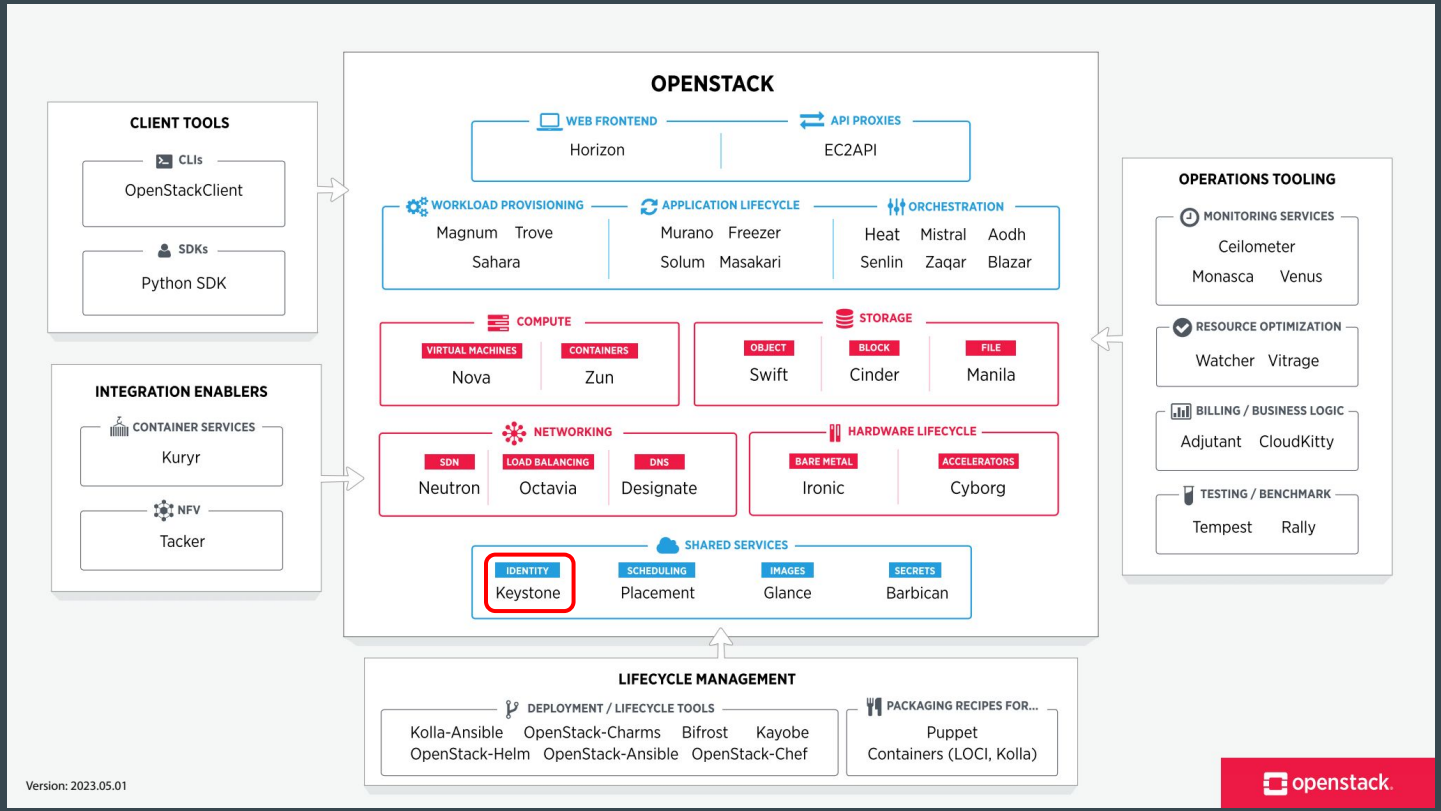
Struttura del Cloud

Struttura di un Cloud

https://www.openstack.org/

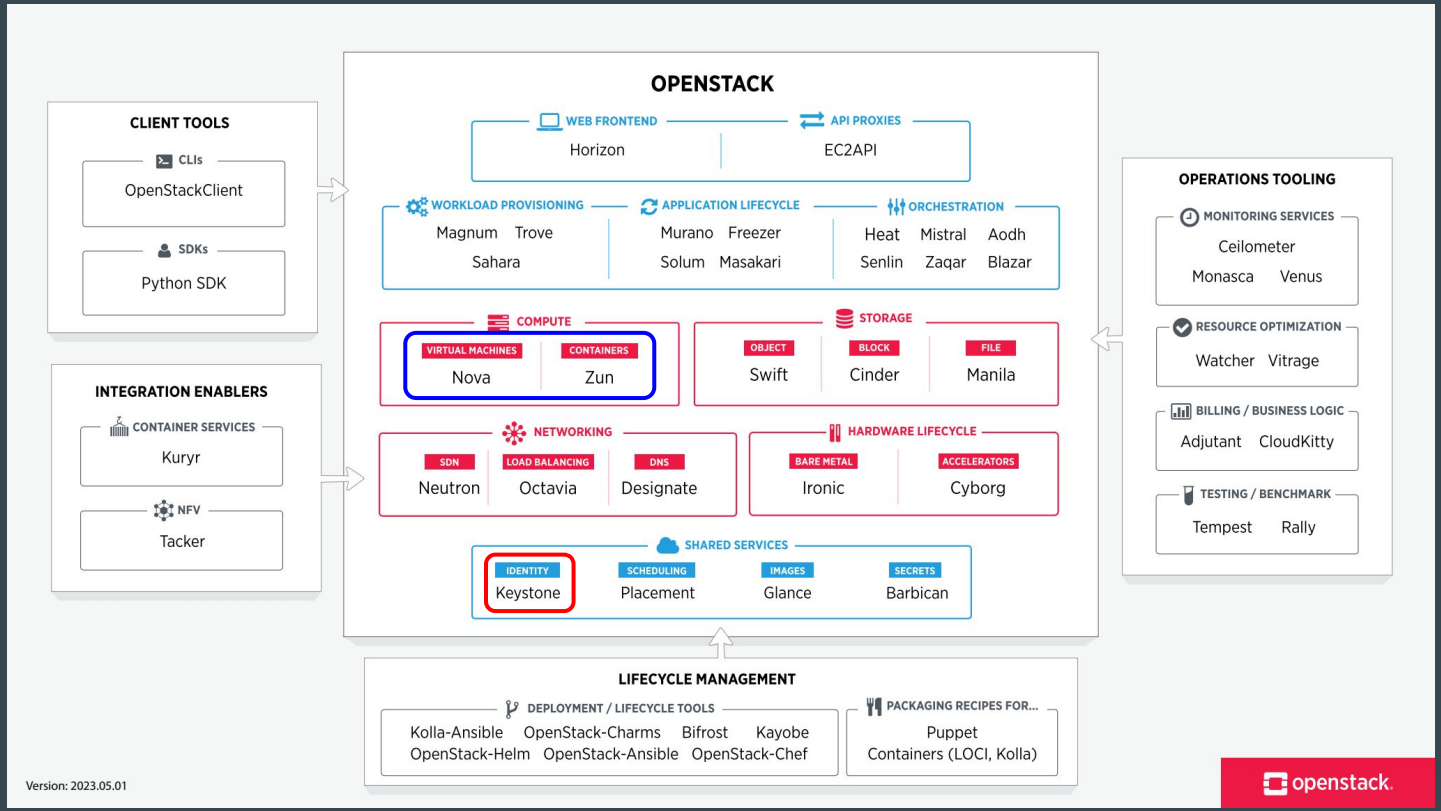


Struttura di un Cloud



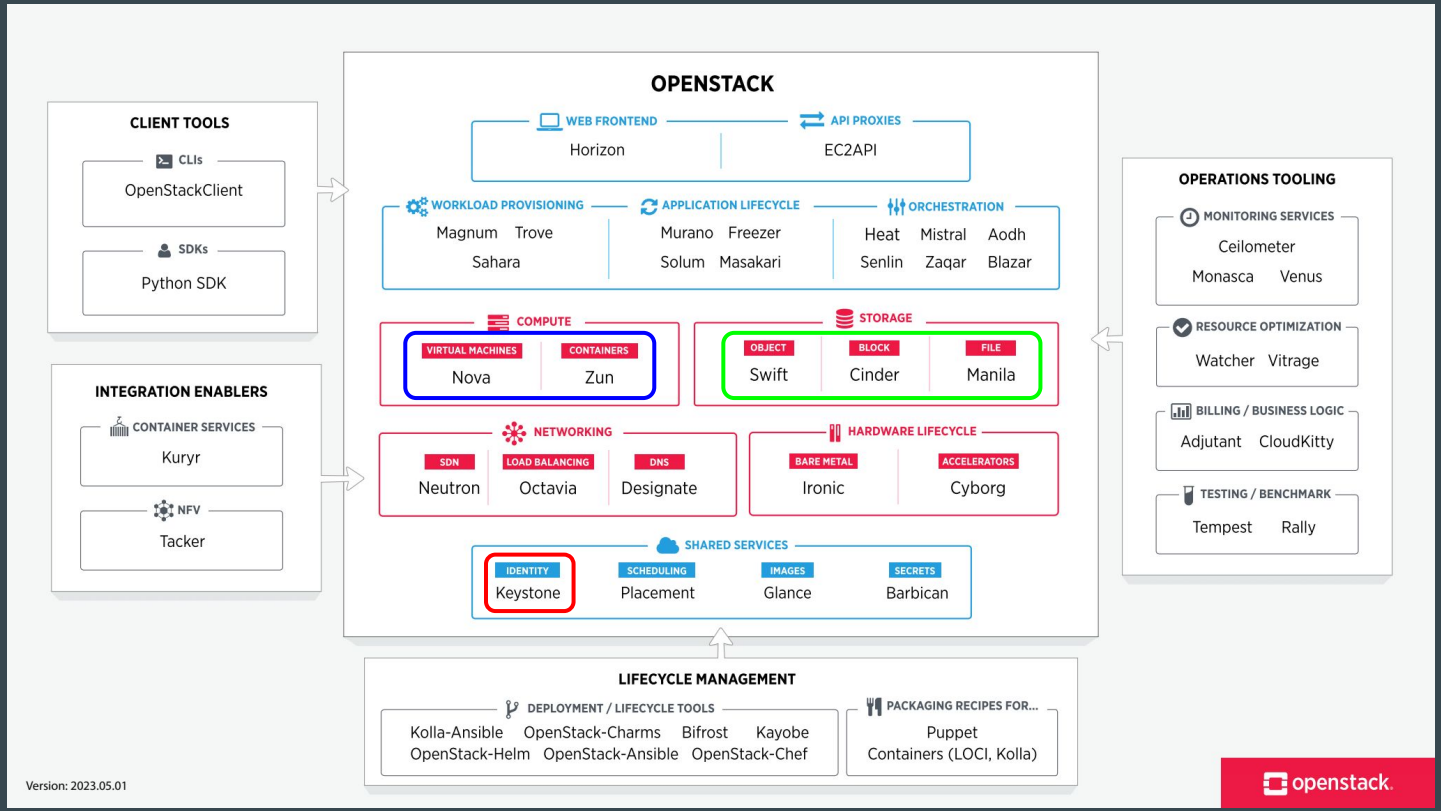
- Identity

Struttura di un Cloud



- Identity
- Compute

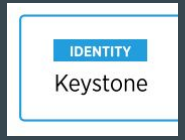
Struttura di un Cloud



- Identity
- Compute
- Storage

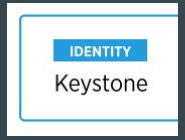
Componenti Principali

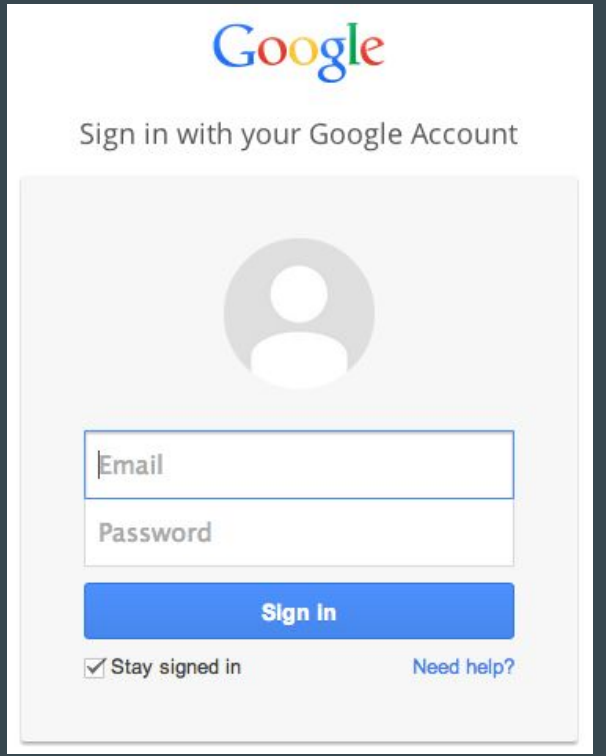
- IAM (Identity and Access Management)



Componenti Principali

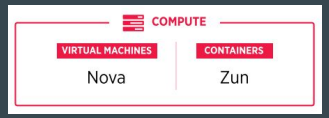
- IAM (Identity and Access Management)
  - verifica identità
  - lista di risorse dedicate
  - privilegi
  - Credito (cloud off premise)





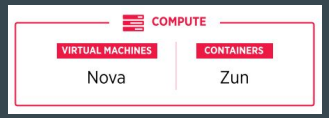
Componenti Principali

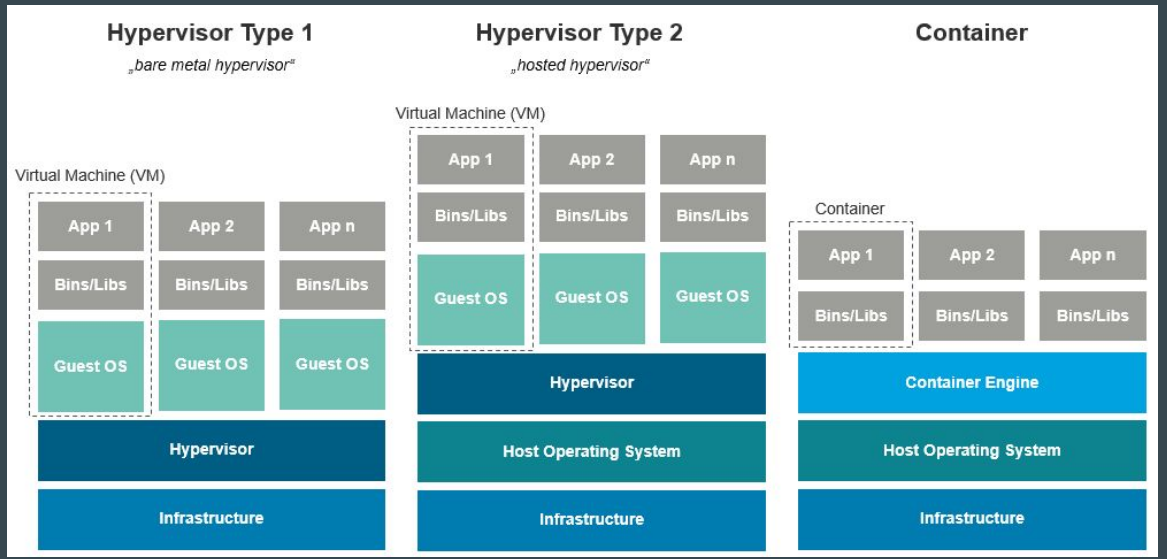
- IAM (Identity and Access Management)
  - verifica identità
  - lista di risorse dedicate
  - privilegi
  - Credito (cloud off premise)
- Compute Services



Componenti Principali

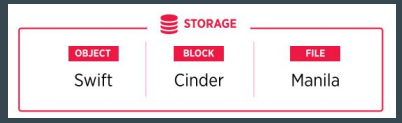
- IAM (Identity and Access Management)
  - verifica identità
  - lista di risorse dedicate
  - privilegi
  - Credito (cloud off premise)
- Compute Services





Componenti Principali

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

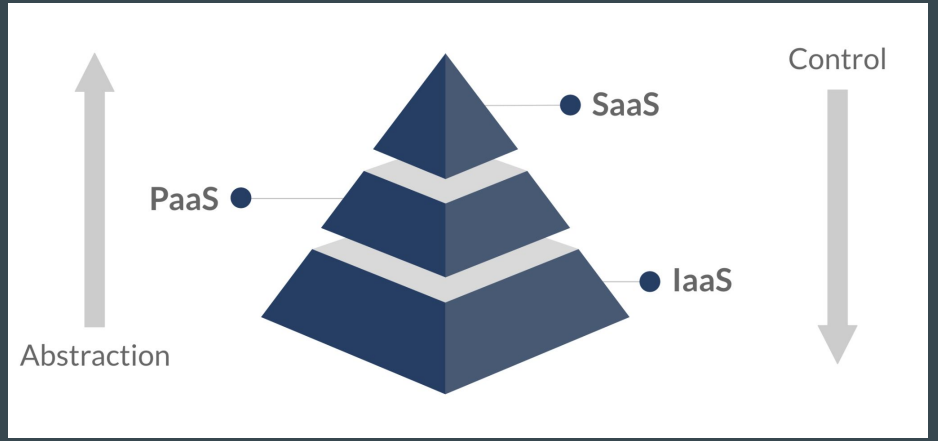


Servizi Cloud

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

Servizi Cloud

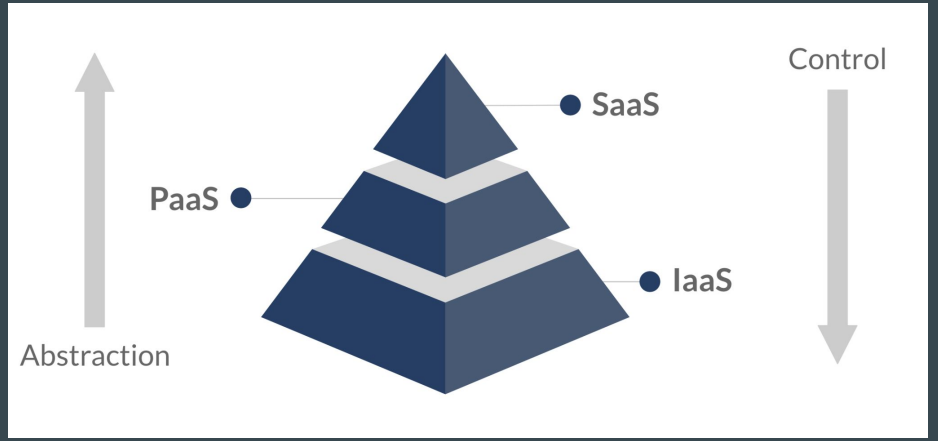
- Infrastructure as a Service
- Platform as a Service
- Software as a Service



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

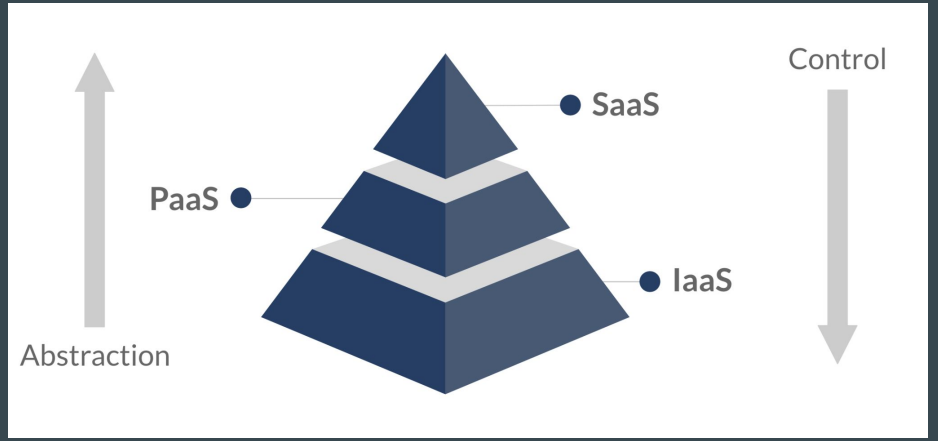
- Infrastructure as a Service
- Platform as a Service
- Software as a Service



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

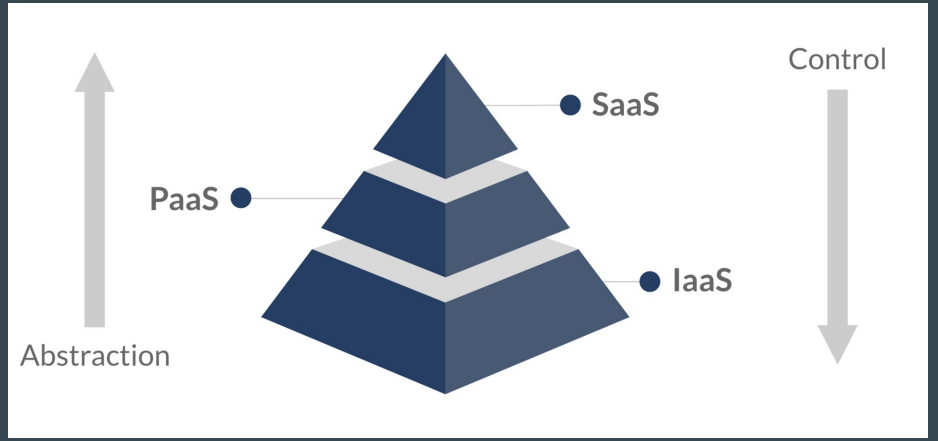
- Infrastructure as a Service
- Platform as a Service
- Software as a Service

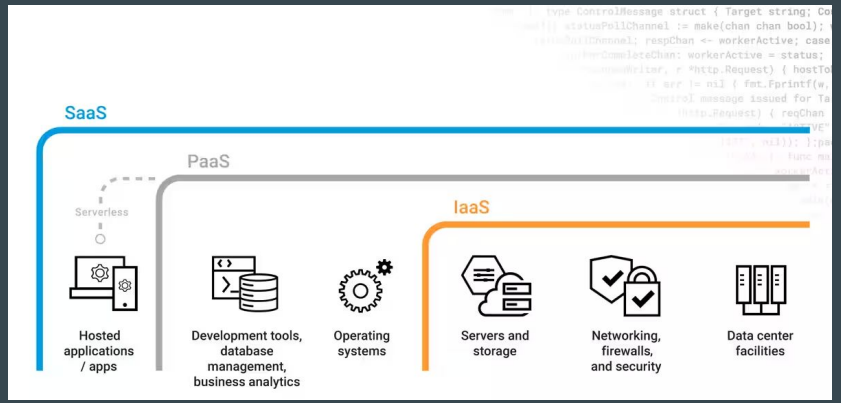


SaaS
L'utente finale non ha bisogno di nessuna conoscenza informatica per utilizzare l'applicazione o i servizi erogati. I servizi sono utilizzabili semplicemente con una connessione internet e un browser.

Servizi Cloud

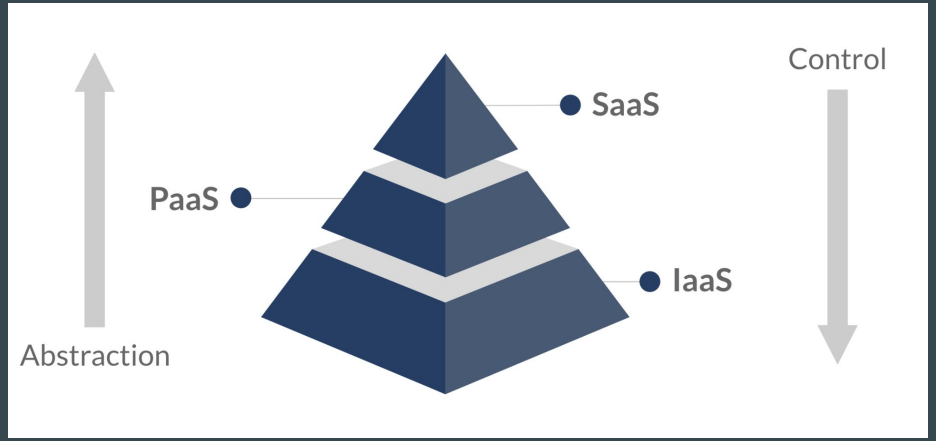
- Infrastructure as a Service
- Platform as a Service
- Software as a Service

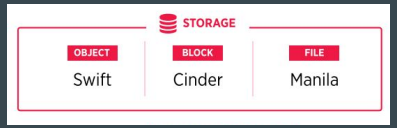


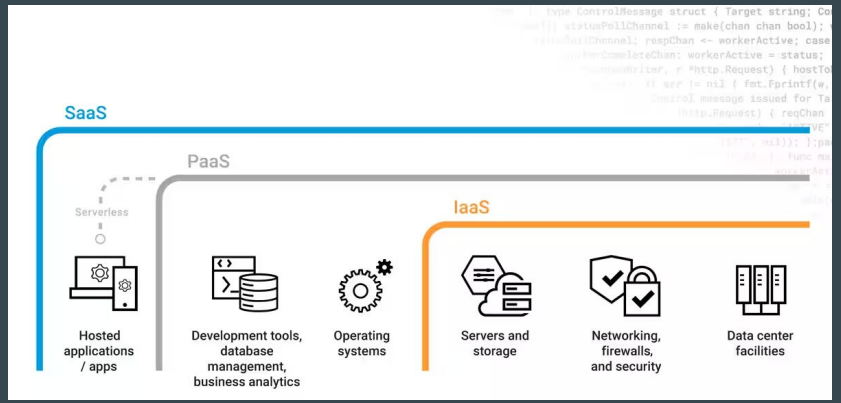


Servizi Cloud

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

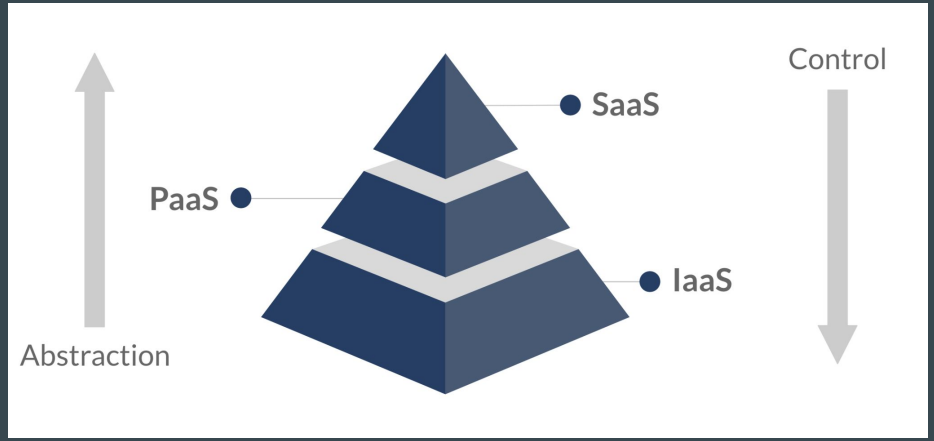


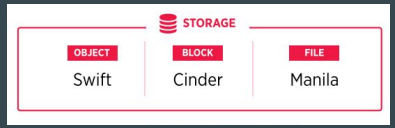


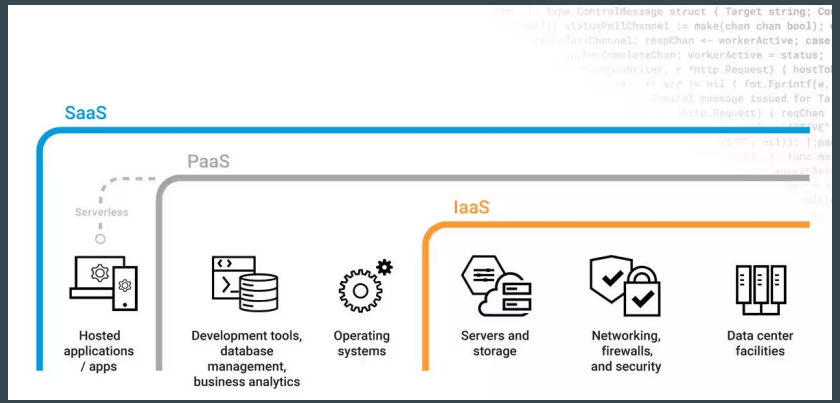


Servizi Cloud

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







https://www.bmc.com/blogs/saas-vs-paas-vs-iaas-whats-the-difference-and-how-to-choose/

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

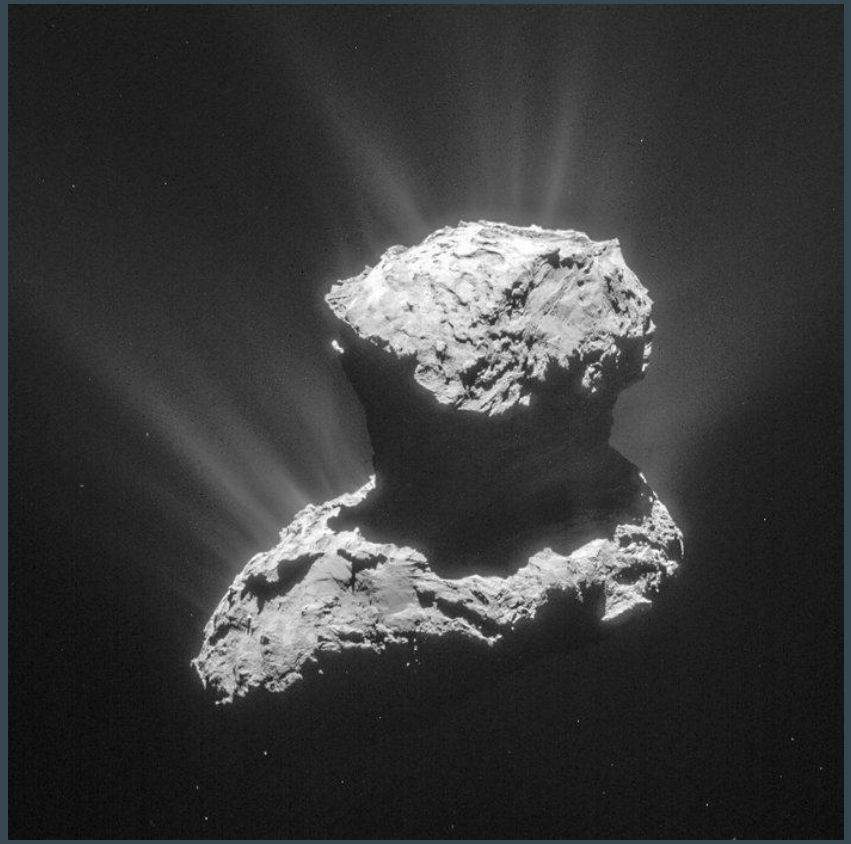
Il metadato è, letteralmente, "(dato) per mezzo di un (altro) dato", è un'informazione che descrive un insieme di dati.

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.

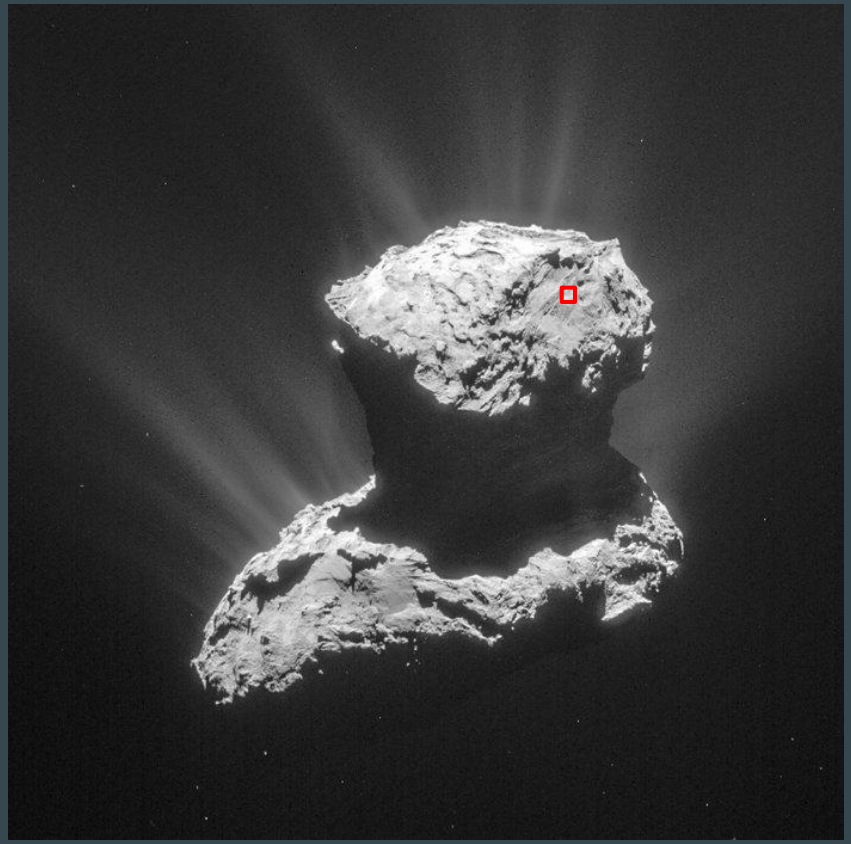
Esempio Dato - Metadato

Esempio in Planetologia



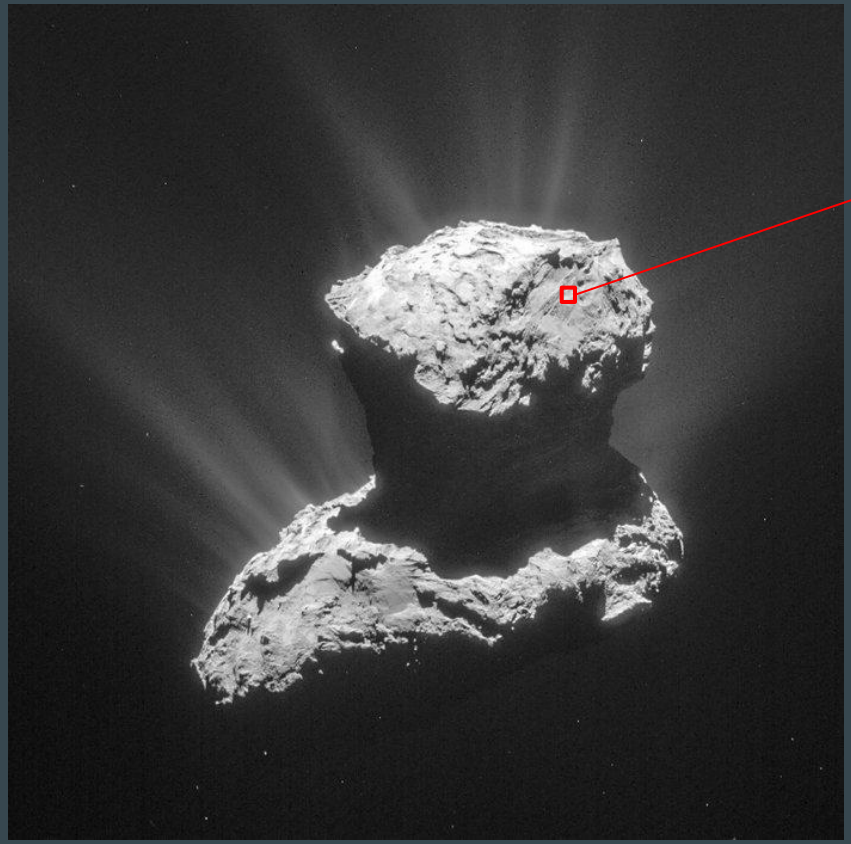
Quale è il dato in questo caso?

Esempio in Planetologia



Quale è il dato in questo caso?

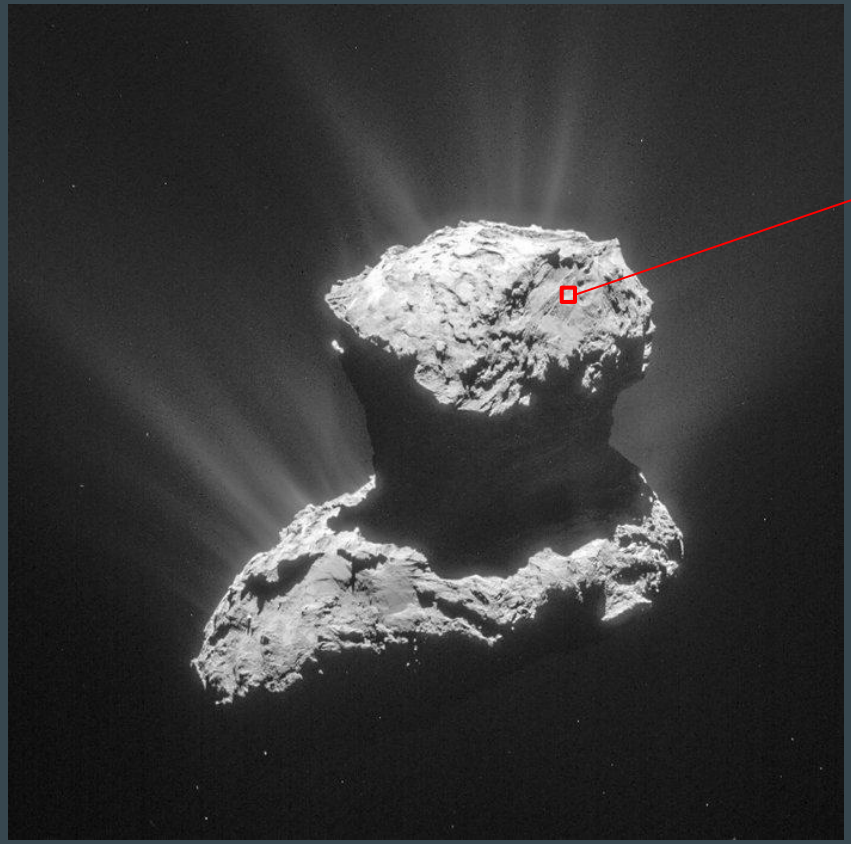
Esempio in Planetologia



Quale è il dato in questo caso?

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

Esempio in Planetologia

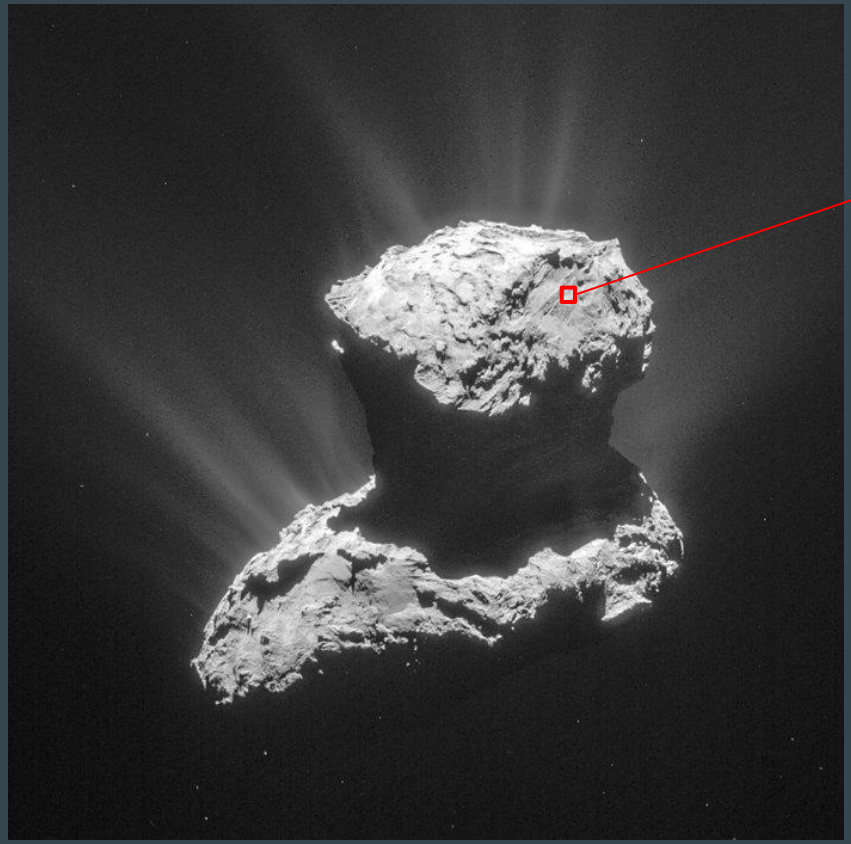


Quale è il dato in questo caso?

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.

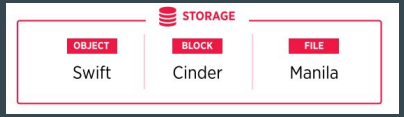
Ha significato?

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

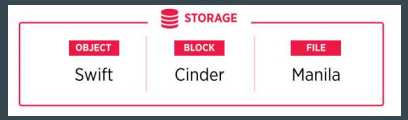
Dati nel Cloud

Archiviazione nel Cloud



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

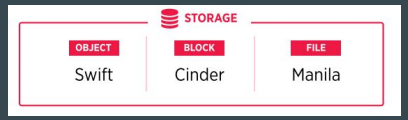
Archiviazione nel Cloud



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.

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

Archiviazione nel Cloud

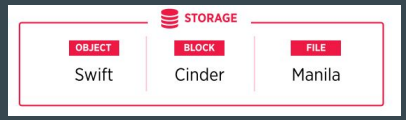


Il **block storage** suddivide i dati in componenti separati composti da blocchi di dati di dimensioni fisse, ognuno dotato di un identificatore univoco. Il block storage permette al sistema di storage sottostante di recuperarlo indipendentemente dalla posizione in cui viene memorizzato.

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.

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

Archiviazione nel Cloud



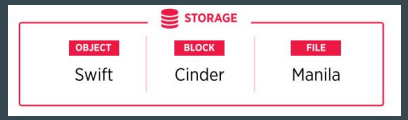
Il block storage suddivide i dati in componenti separati composti da blocchi di dati di dimensioni fisse, ognuno dotato di un identificatore univoco. Il block storage permette al sistema di storage sottostante di recuperarlo indipendentemente dalla posizione in cui viene memorizzato.

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.

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.

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.

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

File Storage

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

Lunghezza massima 255 caratteri

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.

File Storage

- 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

Lunghezza massima 255 caratteri

File Storage

- 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

Lunghezza massima 1024 caratteri

File Storage

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

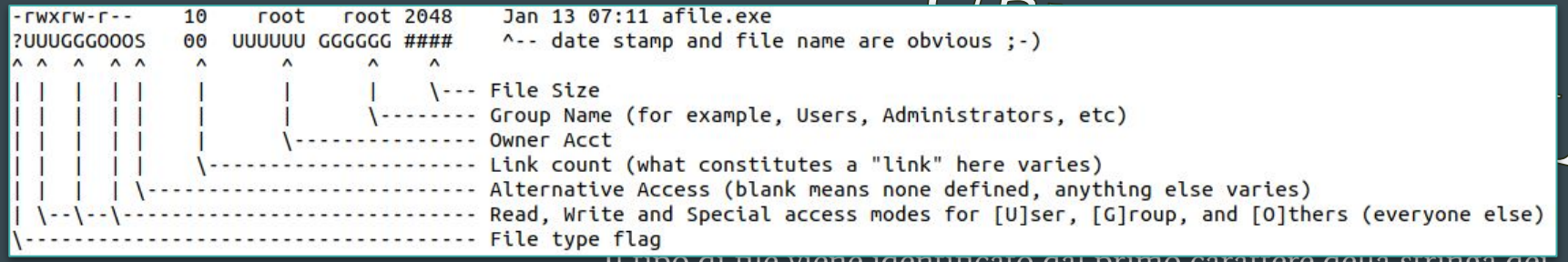
Il tipo di file viene identificato dal primo carattere della stringa dei permessi.

```
-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

Uni...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

```
-rwxrw-r--    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
```

- **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 **cd**.
	`x`	The file can be executed.	The directory can be accessed with **cd**; this is the only permission bit that in practice can be considered to be "inherited" from the ancestor directories, in fact if *any* folder in the path does not have the `x` 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 **u**ser triad; the **setgid** bit when found in the **g**roup triad; it is not found in the **o**thers triad; it also implies that `x` is set.	
	`S`	Same as `s`, but `x` is not set; rare on regular files, and useless on folders.	
	`t`	The **sticky** bit; it can only be found in the **o**thers triad; it also implies that `x` is set.	
	`T`	Same as `t`, but `x` is not set; rare on regular files, and useless on folders.	

https://it.wikipedia.org/wiki/Permessi_(informatica)

Unix and Unix Like

File Storage

- 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 co

I volumi de pository indipender 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.

https://github.com/torvalds/linux

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/PhDCourse2023.git

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 08/06/2023

Database Relazionale

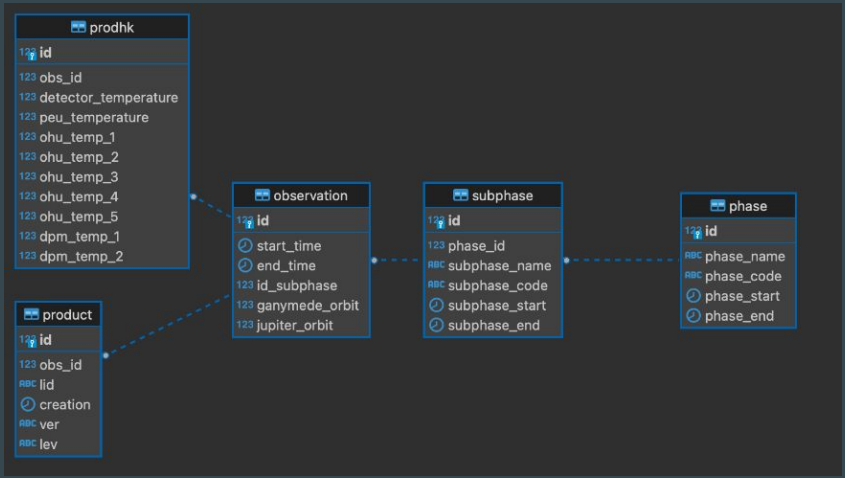
Introduzione

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.

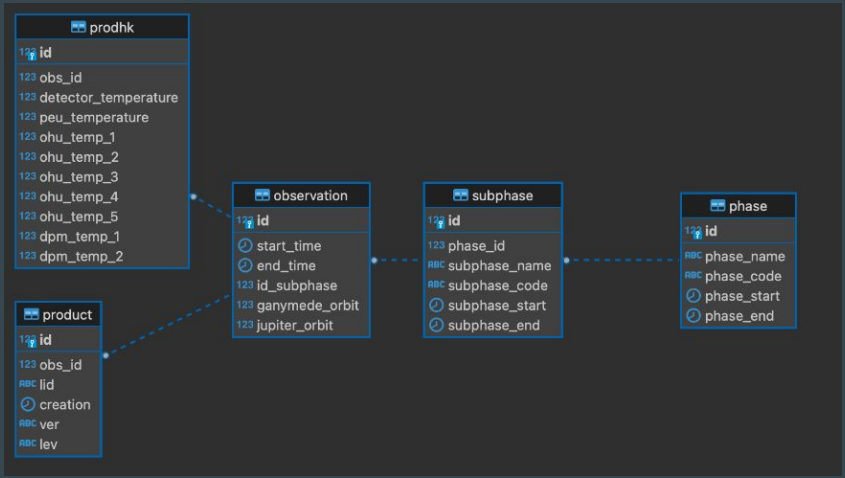
Diagramma Entità-Relazione - Glossario

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

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 test_name, start, stop 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 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' AND
        stop < '2018-12-11';
```

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 * FROM simbio_telecommand tc WHERE tc.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 - 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 * FROM simbio_tcparameter WHERE parDescription LIKE "%VIHI integration%";

[OUT] 578

SQL> SELECT * 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_tcparameter 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"?>
<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.

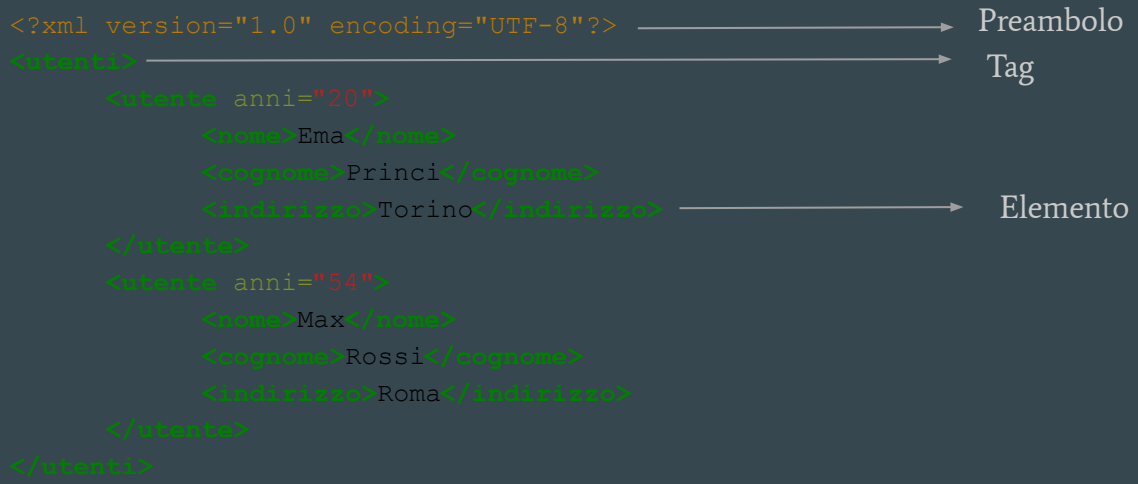
```
<?xml version="1.0" encoding="UTF-8"?>
<utenti>
	<utente anni="20">
		<nome>Ema</nome>
		<cognome>Princi</cognome>
		<indirizzo>Torino</indirizzo>
	</utente>
	<utente anni="54">
		<nome>Max</nome>
		<cognome>Rossi</cognome>
		<indirizzo>Roma</indirizzo>
	</utente>
</utenti>
```

Preambolo

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.



Fondamenti di programmazione Python

Fondamenti di Python

Linguaggio: Python 3.11.2

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)

```
romolo.politi@Bishop ~ % python3
Python 3.10.4 (v3.10.4:9d38120e33, Mar 23 2022, 17:29:05) [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()
romolo.politi@Bishop ~ %
```

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

```
romolo.politi@Bishop ~ % echo "print('Hello World')" >>test.py
romolo.politi@Bishop ~ % python3 test.py
Hello World
romolo.politi@Bishop ~ %
```

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.

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

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

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

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

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

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

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

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 inizializare variabili. Vengono inzializzate 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
`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`

Tipi di variabili

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

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

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

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.

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

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.

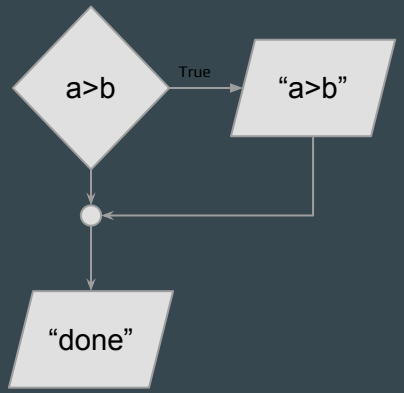
Dichiaratori di Condizione

(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)

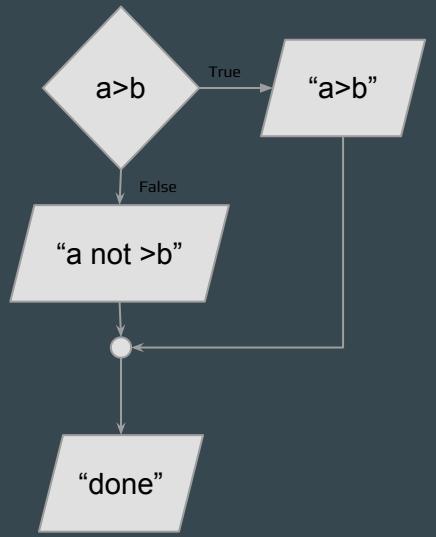
Il più semplice è if.

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")
```



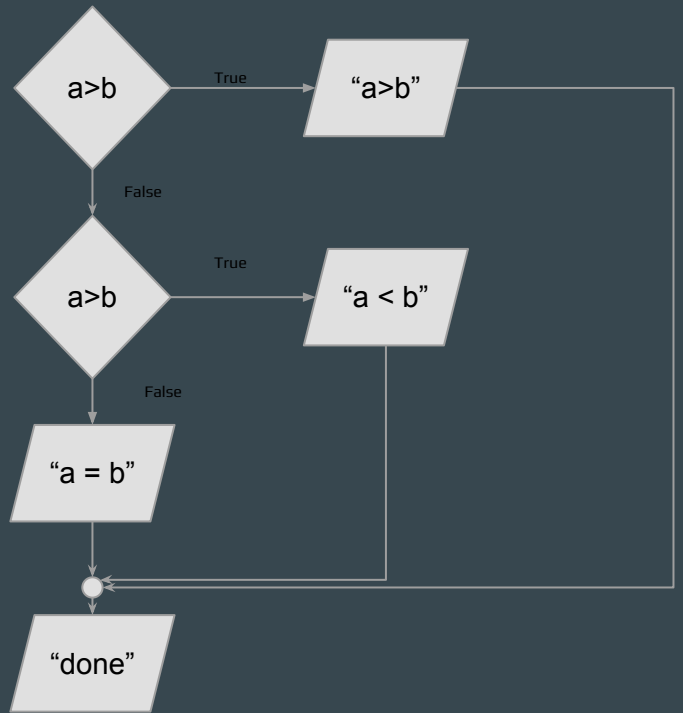
Dichiaratori di Condizione

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

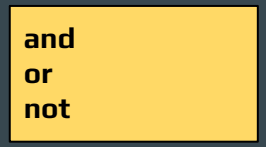
Logici

Operatori

Aritmetici

Confronto

Logici



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

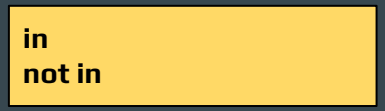
Confronto

Logici

Assegnazione

Identità

Appartenenza (Membership)



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)
```

Lezione del 20/06/2023

Dichiaratori di Condizione

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

```
if cond1:
    if cond2:
        ....
```

Permette di esplorare tutte e quattro le possibilità

```
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:bool = 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!

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: QIUtw

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')
   a1 = logging.getLogger(logger)
   if flag:
       a1.warning(f"Log level {oldLevel} is not valid. Used the default value 20" )
   return a1  # logging
```

https://docs.python.org/3/library/logging.html

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.

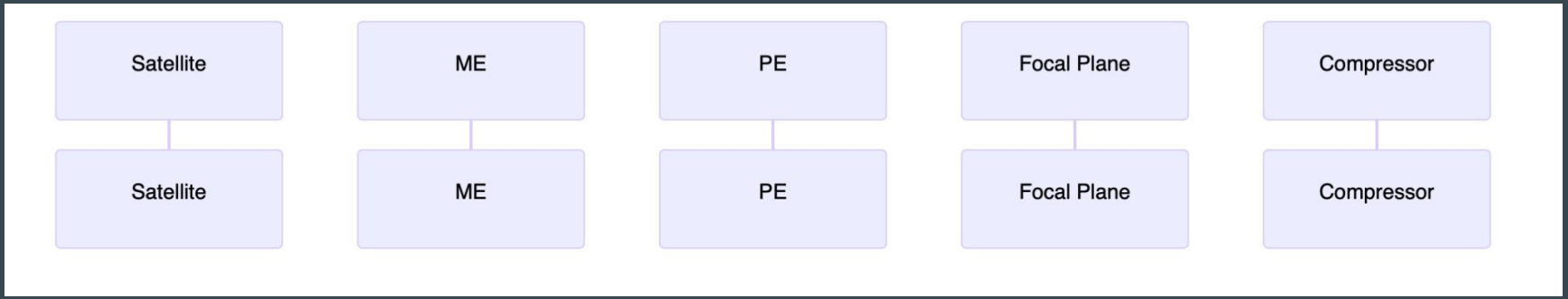
Per renderla più leggibile utilizziamo il modulo ***rich***

```
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	Intitial State	Transient State	Final State	Descrption
	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	Intitial State	Transient State	Final State	Descrption
	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

```
# Time (seconds), TC
 1,NSS00001
#  3,NSS00002
 8,NSS00003
12,NSS00002

```

```
with open(command,FMODE.READ) as f:
    lines=f.readlines()

for line in lines:
    if line.strip().startswith('#'):
        continue
    else:
        print(line.strip())
```

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

```
# Time (seconds), TC
 1,NSS00001
#  3,NSS00002
 8,NSS00003
12,NSS00002

```

```
with open(command,FMODE.READ) as f:
    lines=f.readlines()

for line in lines:
    if line.strip().startswith('#'):
        continue
    else:
        print(line.strip())
```

```
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

```python
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

```python
def message(text: str):
    def decorate(f):
        @wraps(f)
        def inner(*args, **kwargs):
            with Status(text, spinner='aesthetic', console=args[0].console):
                ret = f(*args, **kwargs)
                if args[0].verbose:
                    print(f"{MSG.INFO} {f.__name__} executed")
            return ret
        return inner
    return decorate
```