Advanced Coding and Cloud Computation

•••

Romolo Politi

PhD Course 2024

Lecture List

Elenco Lezioni

- September 16th 2024
- September 18th 2024
- October 7th 2024

Lecture September 16th 2024

Course Overview

Course Overview

Cloud
- Cloud structure
- Data in the Cloud
- Cloud Computing

Data
- Data and Metadata
- Archives
- Relational and not-relational Database

Computing
- Retrieval
- Manipulation
- Visualization

Environment
- Virtualization and Containers
- Microservices
- DevOps

Coding
- Fundamentals of Coding
- Python
- Versioning and Documentation

Tools

- Slides and Examples available on GitHub:
  - https://github.com/RomoloPoliti-INAF/PhDCourse2024
- The example will be written in Python 3.12
- Microsoft Visual Studio Code will be used as framework
  - https://code.visualstudio.com

Struttura del Corso

- The list of topics shown earlier was organized by categories.
- We will follow an example-driven approach to better understand the philosophy behind it.
- After the introduction to programming, we will develop an example of a complex program (State Machine).
- Lastly, we will develop a WebApp and prepare it for deployment in containers.
- For some topics, we will not go into detail because the purpose of the course is to provide a general overview of the subject.
- Even though they won't be discussed, many details will be available in the slides or through the provided links.

Cloud Definition

What’s Cloud

It is the on-demand availability of computer system resources, especially data storage (cloud storage) and computing power, without direct active management by the user.

wikipedia

Cloud Types

In Promise

openstack™

OpenNebula

Out Promise

Google Cloud

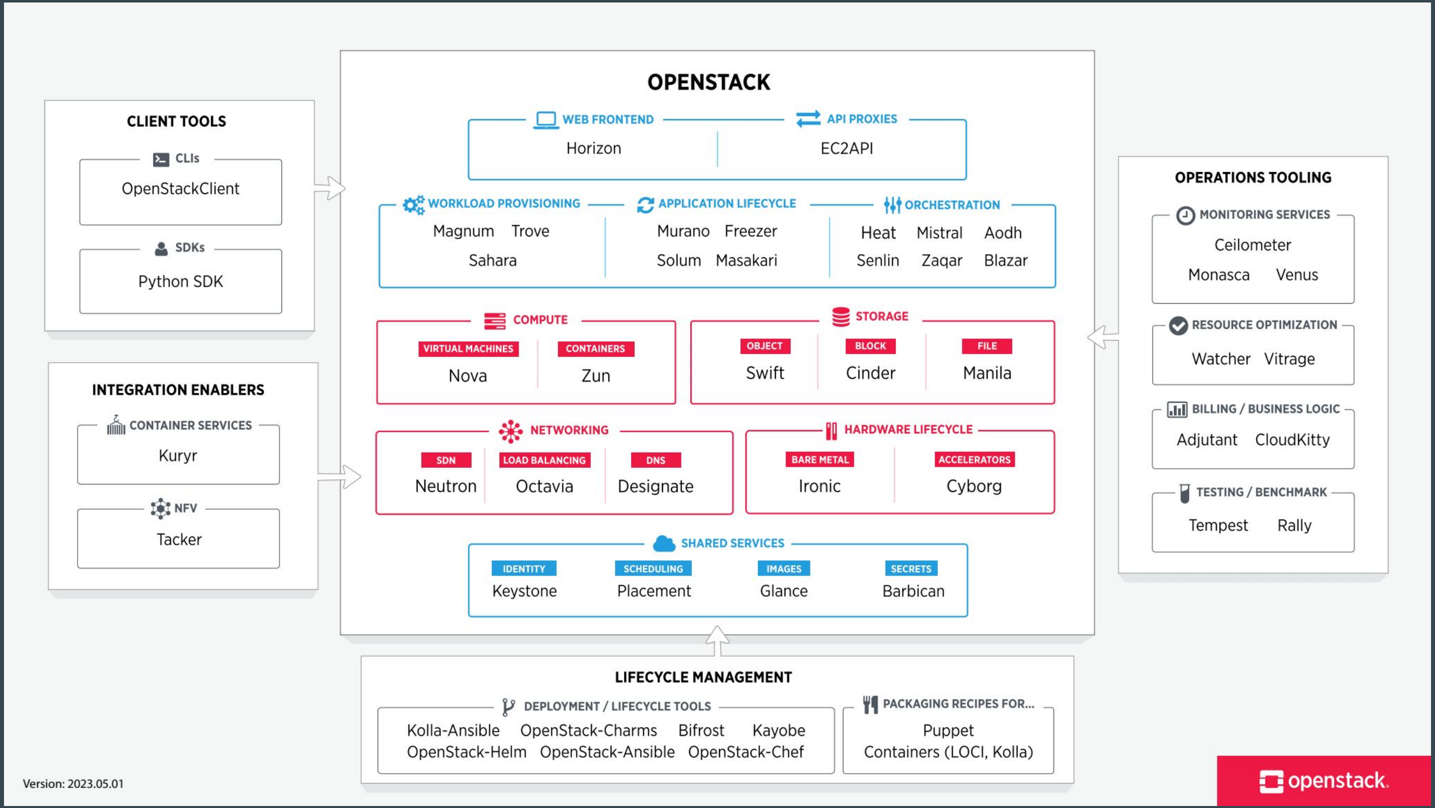
aws

Azure

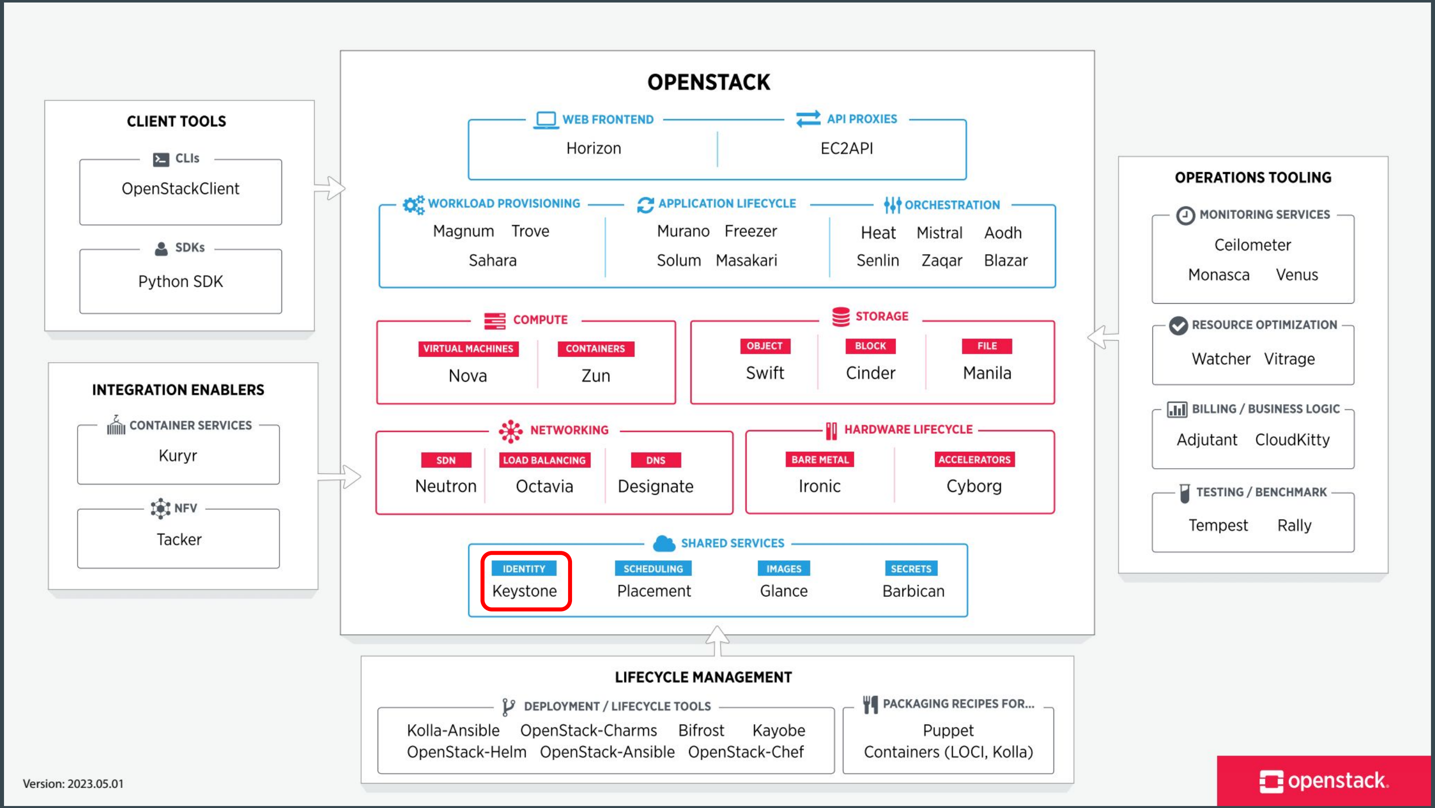
Cloud Structure

Cloud Structure

https://www.openstack.org/

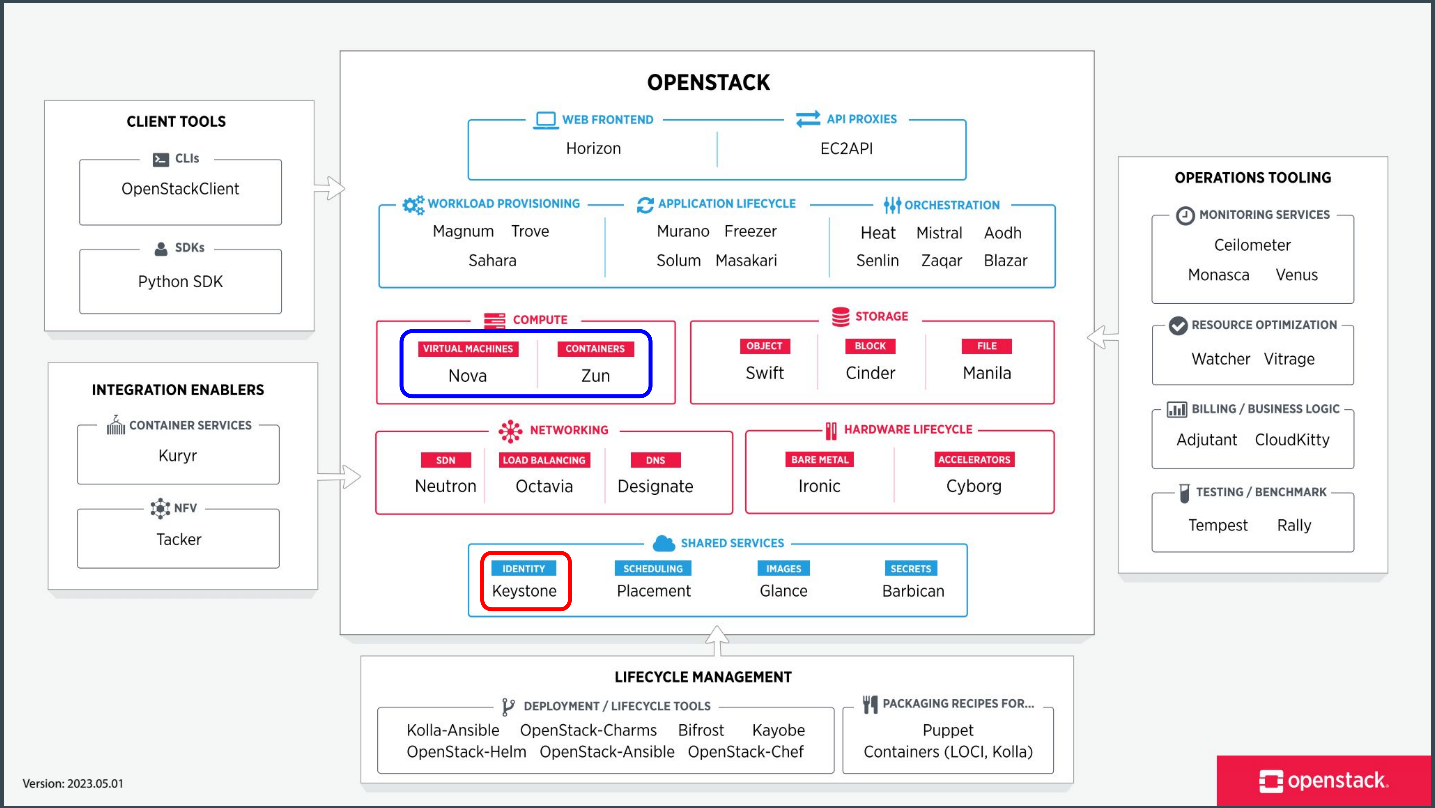


Cloud Structure



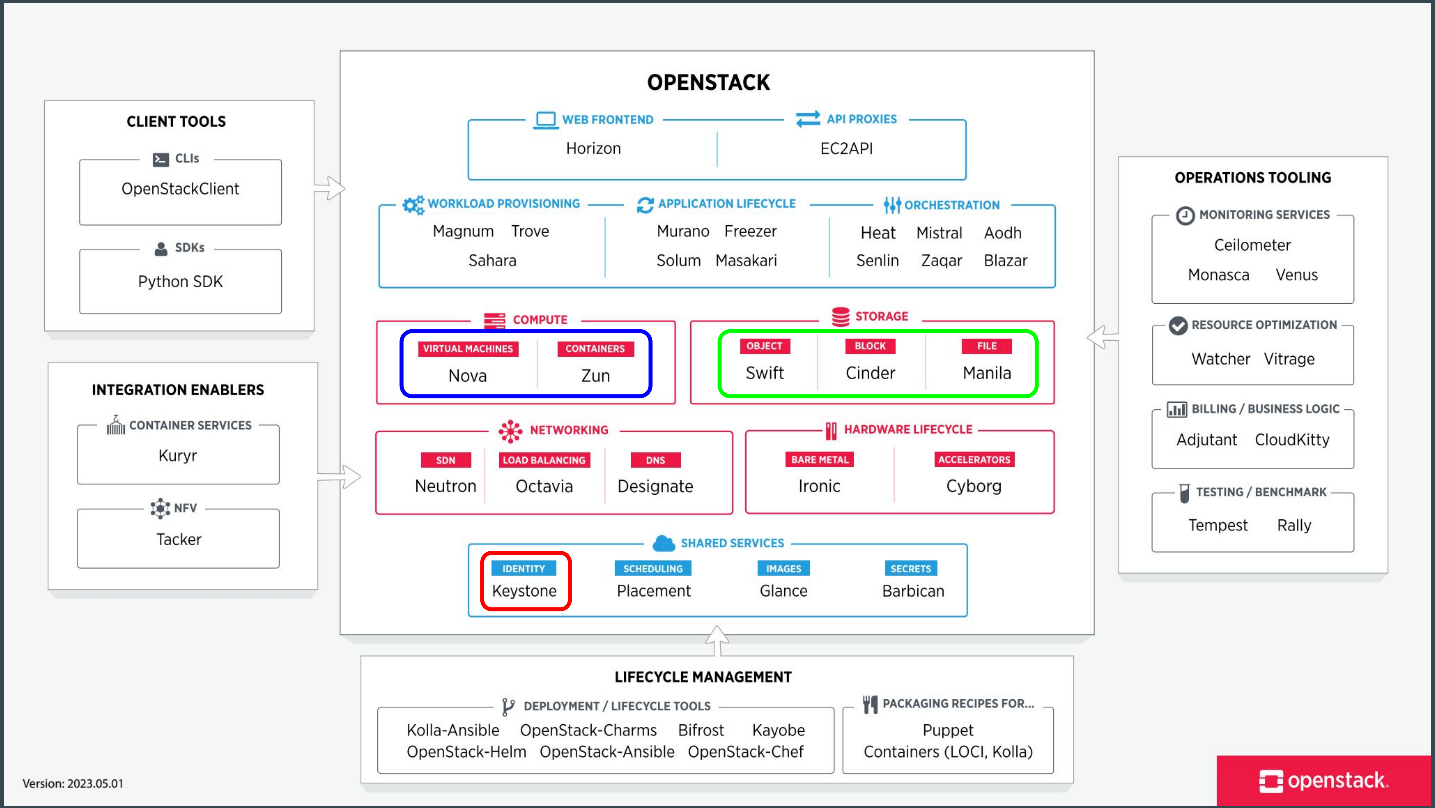
- Identity

Cloud Structure



- Identity
- Compute

Cloud Structure



- Identity
- Compute
- Storage

Main Components

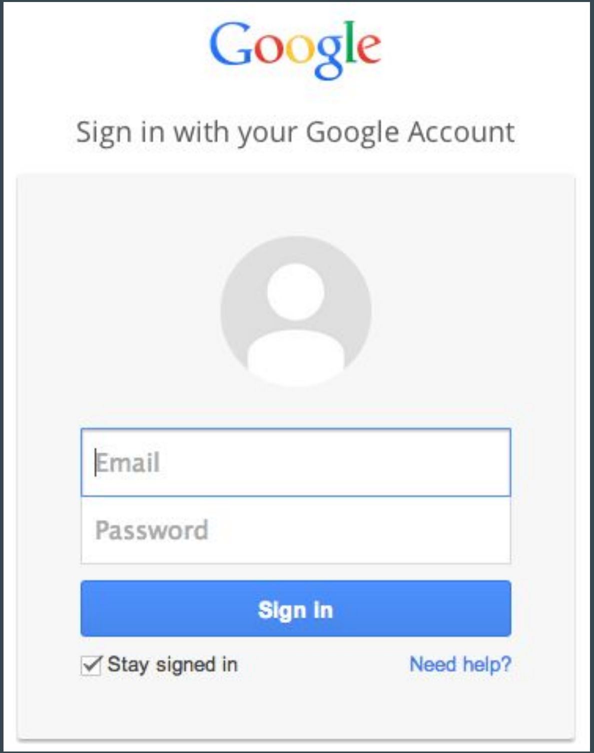
- IAM (Identity and Access Management)

IDENTITY
Keystone

Main Components

- IAM (Identity and Access Management)
  - identity check
  - list of resources
  - privileges
  - credits (cloud off premise)

IDENTITY
Keystone


Google

Sign in with your Google Account

Email

Password

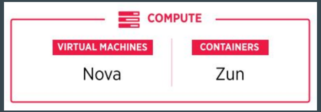
Sign in

✓ Stay signed in

Need help?

Main Components

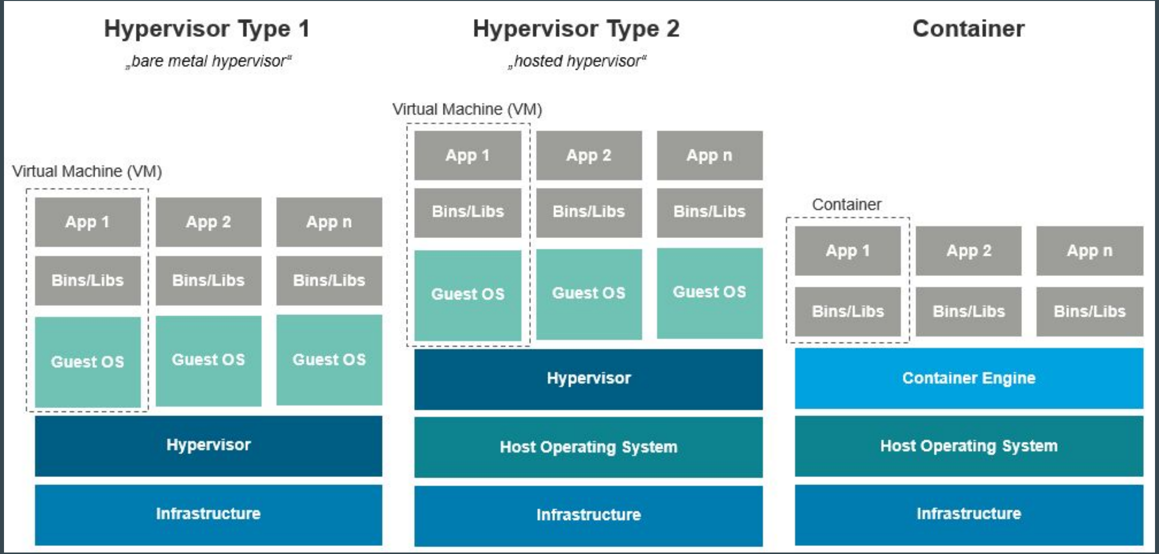
- IAM (Identity and Access Management)
  - identity check
  - list of resources
  - privileges
  - credits (cloud off premise)
- Compute Services



Main Components

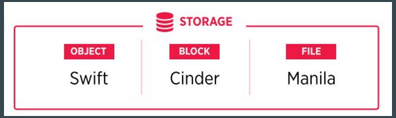
- IAM (Identity and Access Management)
  - identity check
  - list of resources
  - privileges
  - credits (cloud off premise)
- Compute Services

COMPUTE
VIRTUAL MACHINES
Nova
CONTAINERS
Zun



Main Components

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

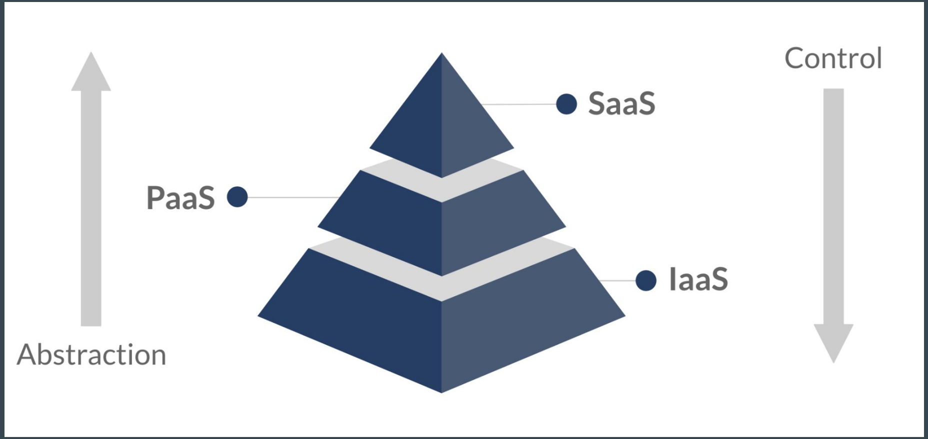


Servizi Cloud

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

Cloud Services

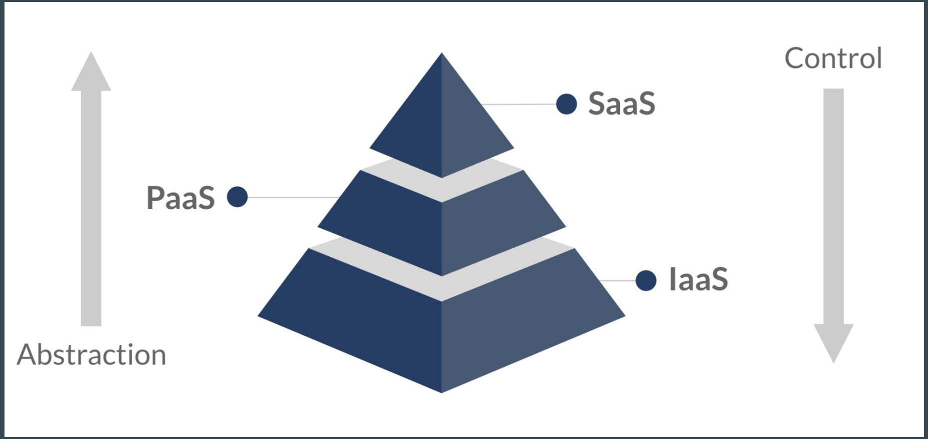
- Infrastructure as a Service
- Platform as a Service
- Software as a Service



IaaS
provides virtualized computing resources (CPU, RAM, disks, etc.) over the internet, allowing users to manage and scale hardware infrastructure without physical ownership.

Cloud Services

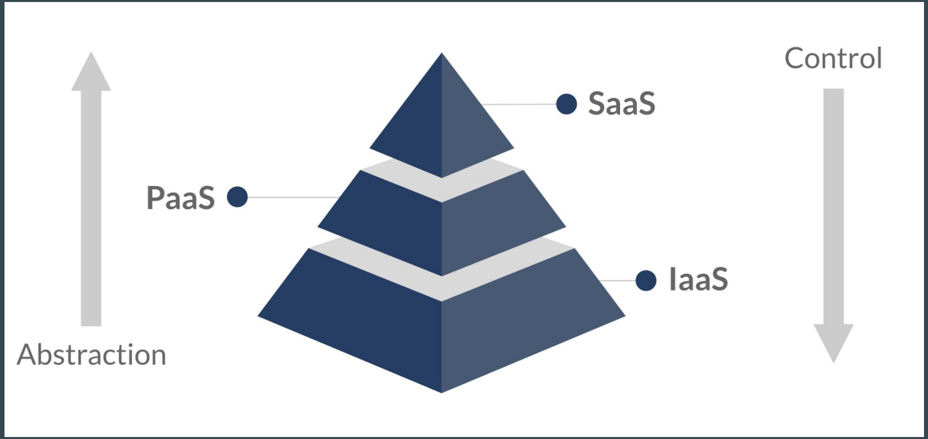
- Infrastructure as a Service
- Platform as a Service
- Software as a Service



PaaS
It provides a cloud-based environment where developers can build, deploy, and manage applications without dealing with the underlying infrastructure.

Servizi Cloud

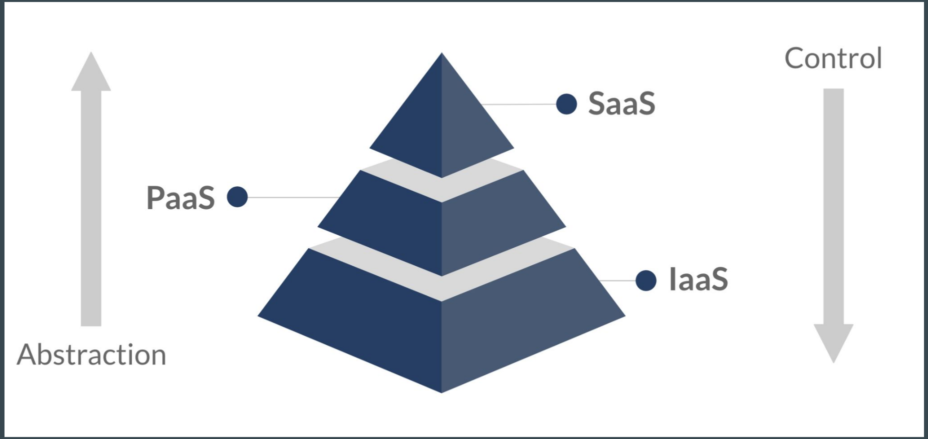
- Infrastructure as a Service
- Platform as a Service
- Software as a Service

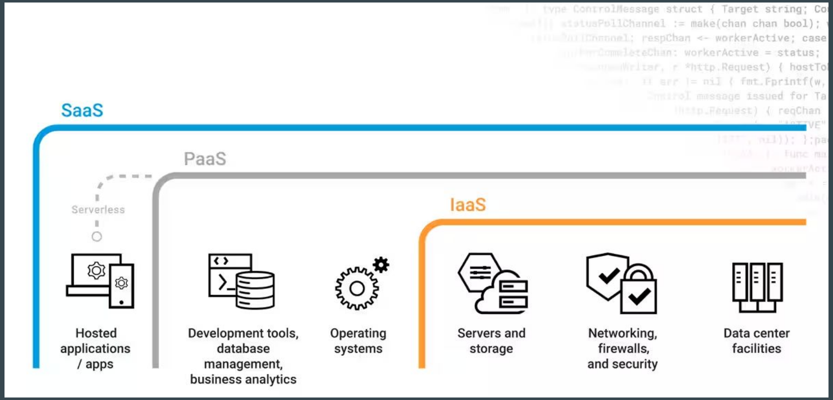


SaaS
it is a cloud-based model where applications are hosted and provided over the internet, allowing users to access and use software without managing the underlying infrastructure.

Servizi Cloud

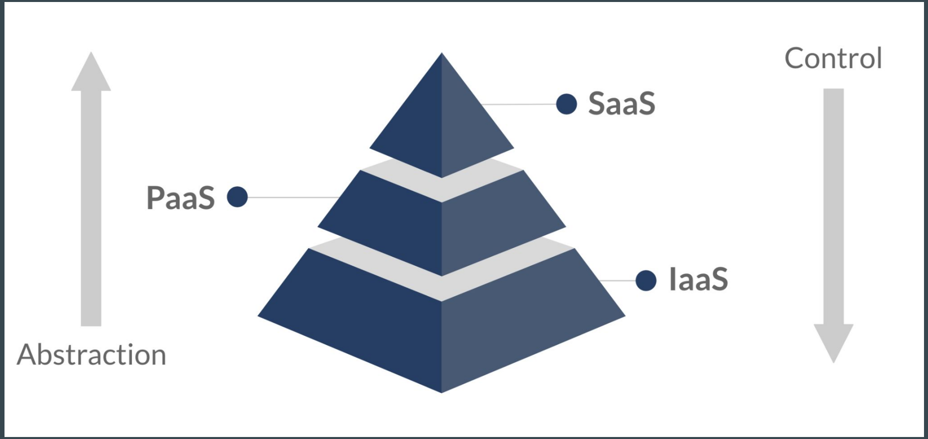
- Infrastructure as a Service
- Platform as a Service
- Software as a Service

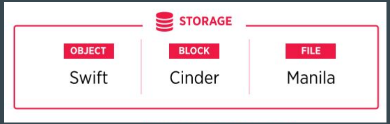


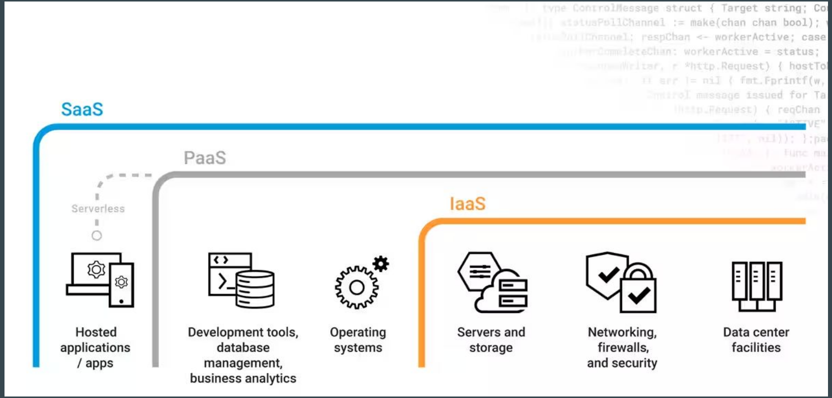

SaaS
PaaS
Serverless
IaaS
Hosted applications / apps
Development tools, database management, business analytics
Operating systems
Servers and storage
Networking, firewalls, and security
Data center facilities

Servizi Cloud

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

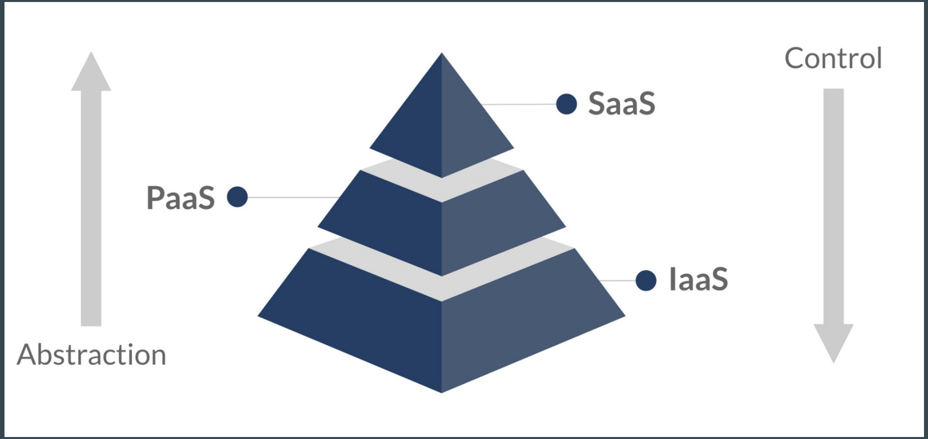


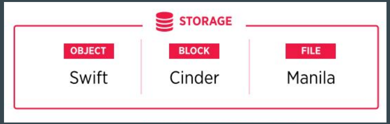


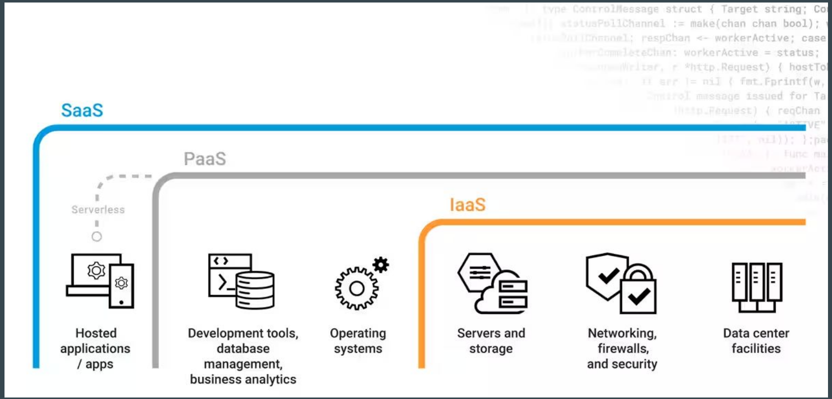

SaaS
PaaS
Serverless
IaaS
Hosted applications / apps
Development tools, database management, business analytics
Operating systems
Servers and storage
Networking, firewalls, and security
Data center facilities

Servizi Cloud

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






SaaS
PaaS
Serverless
IaaS
Hosted applications / apps
Development tools, database management, business analytics
Operating systems
Servers and storage
Networking, firewalls, and security
Data center facilities

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

Data and Metadata

Data Definition

In computing, a data is a collection of facts, figures, or details that can be processed or analyzed. It often consists of numbers, text, or other types of information that are recorded and stored electronically.

Data serves as the foundation for creating information and insights through analysis and interpretation. It can be raw, unprocessed input or structured and organized to facilitate meaningful conclusions. In various contexts, data is used to make decisions, generate reports, or drive machine learning algorithms. Proper management and understanding of data are crucial for effective decision-making and problem-solving.

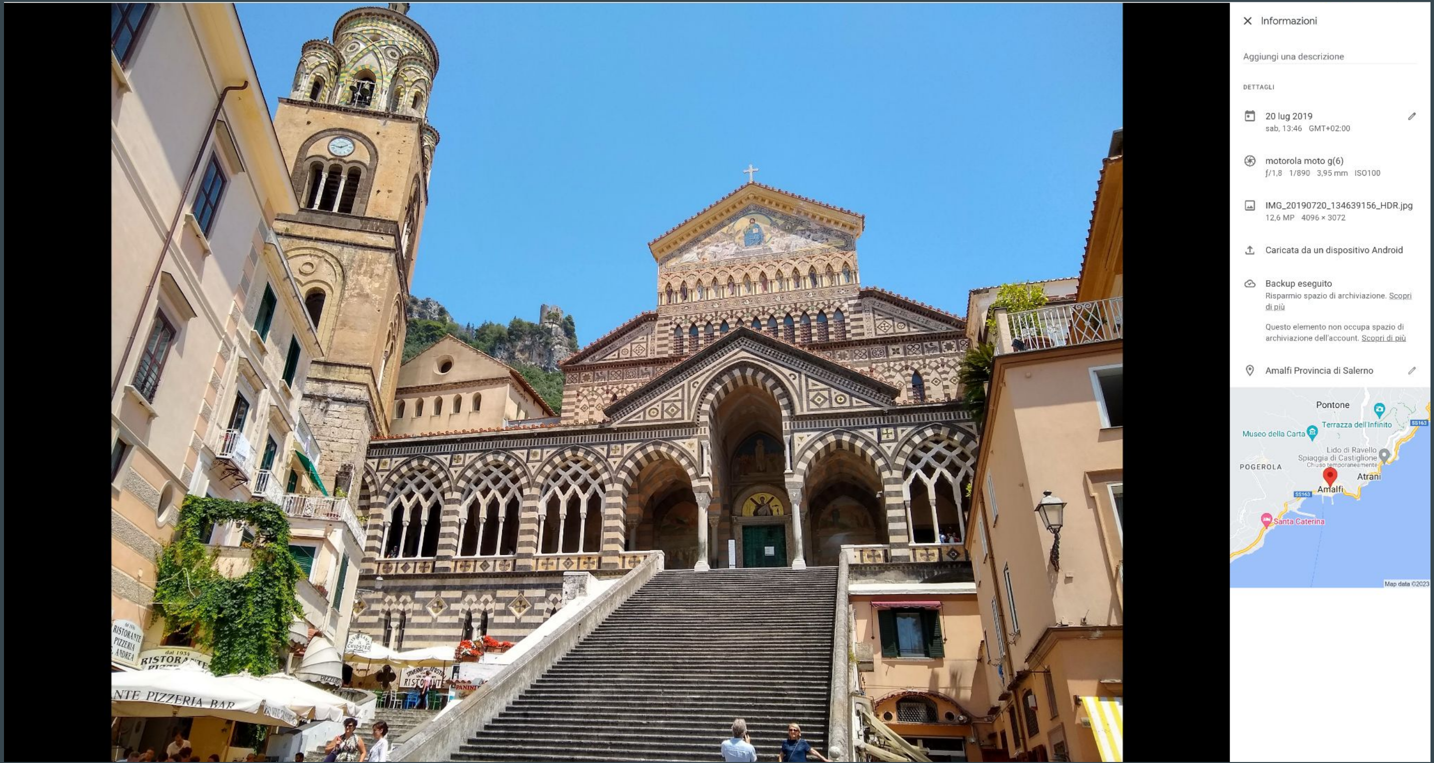
Metadata Definition

A metadata is data that provides information about other data. It describes various attributes of data, such as its origin, format, and relationships to other data, which helps in organizing, managing, and retrieving it efficiently.

Metadata can include details like the creator of a file, the date it was created, and how it should be used. It is essential for data cataloging and improving the accessibility and usability of information.

By offering context and structure, metadata enhances data searchability and interoperability across different systems and platforms.

Data and Metadata Example



Informazioni

Aggiungi una descrizione

DETTAGLI

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

motorola moto g(6)
ƒ/1,8 1/890 3,95 mm ISO100

IMG_20190720_134639156_HDR.jpg
12,6 MP 4096 × 3072

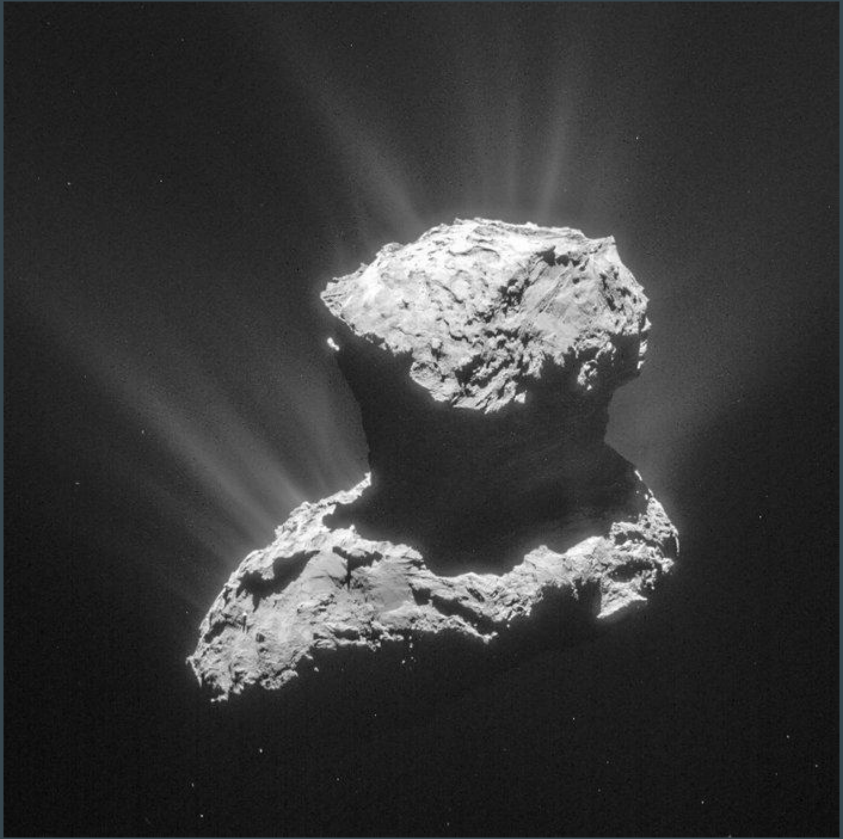
Caricata da un dispositivo Android

Backup eseguito
Risparmio spazio di archiviazione. Scopri di più

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

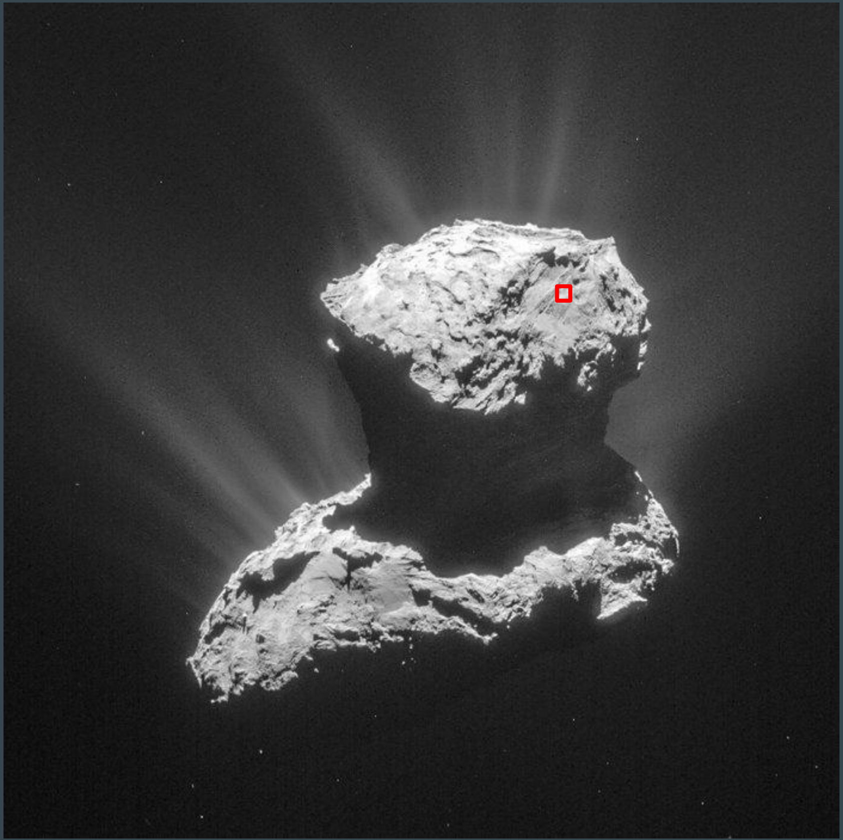
Amalfi Provincia di Salerno

Planetological Example



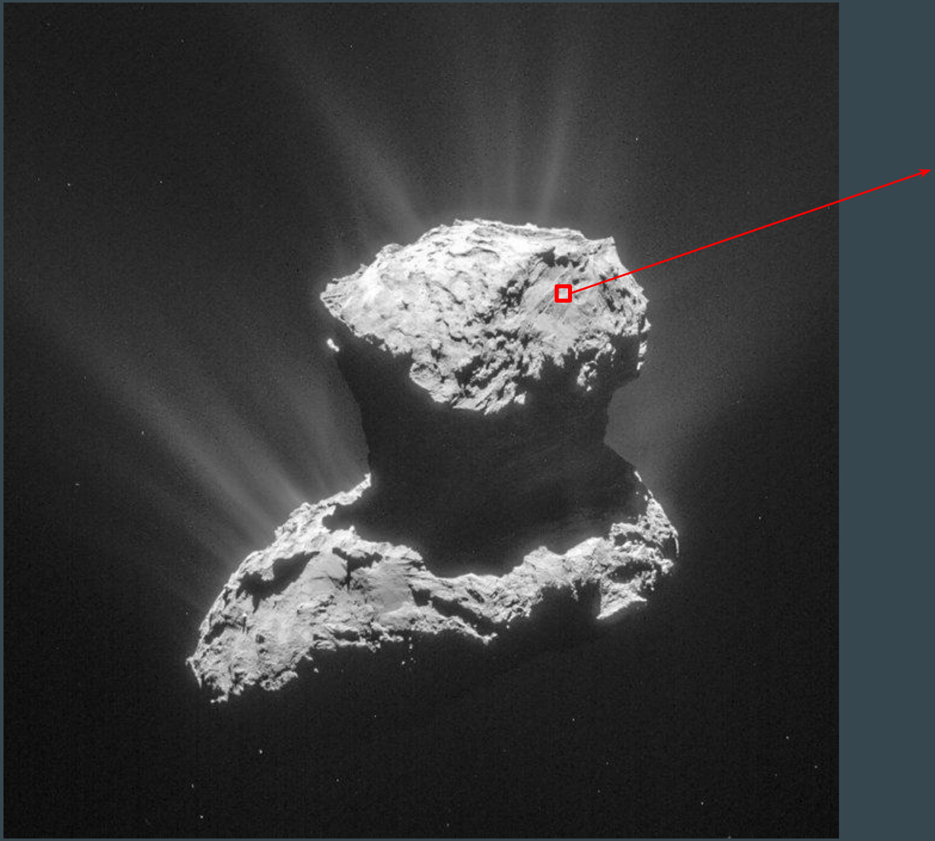
Which is the data?

Planetological Example



Which is the data?

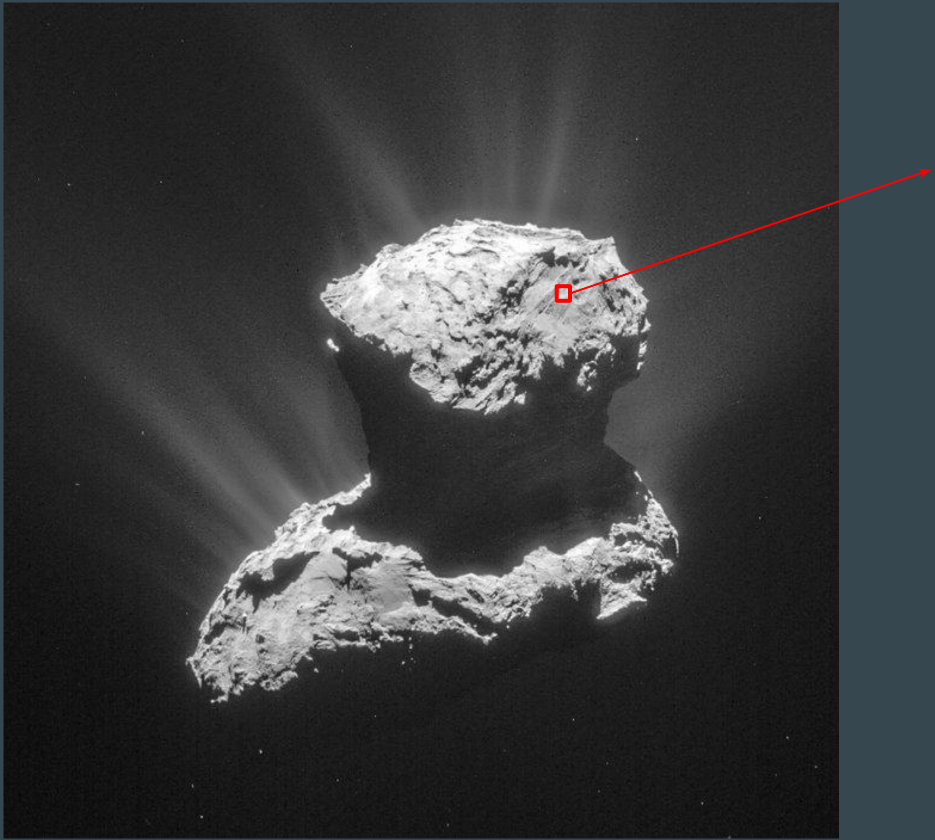
Planetological Example



Which is the data?

The pixel is represented as a 32-bit floating-point number.

Planetological Example

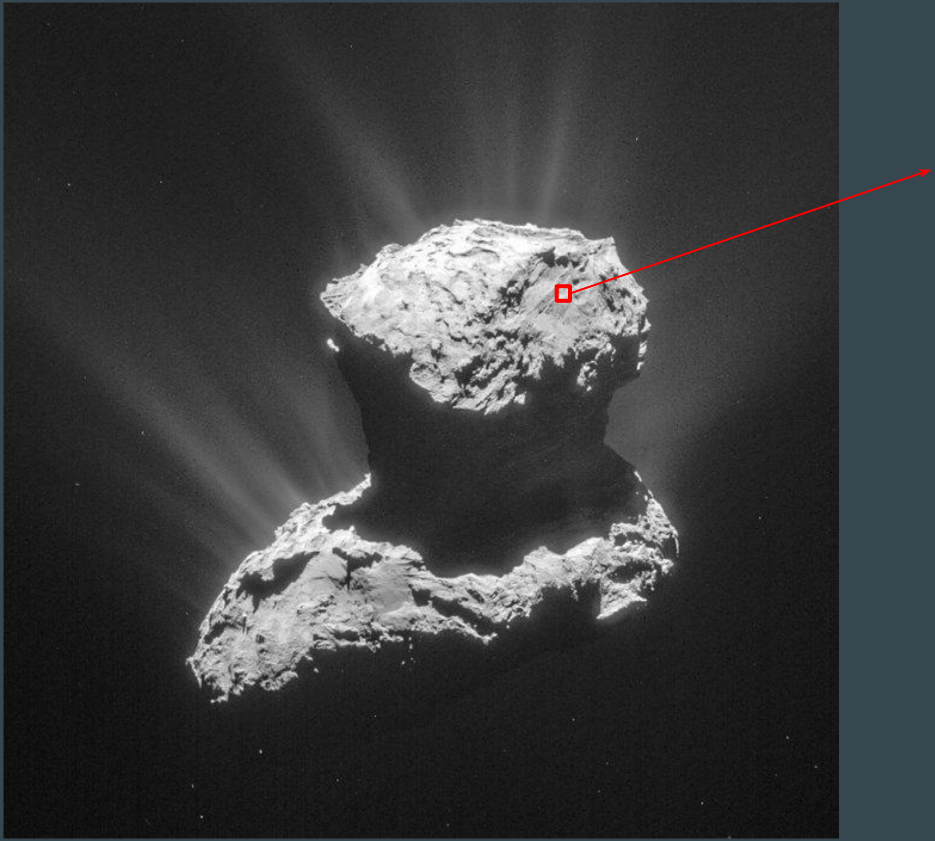


Which is the data?

The pixel is represented as a 32-bit floating-point number.

Does this have meaning?

Planetological Example



Which is the data?

The pixel is represented as a 32-bit floating-point number.

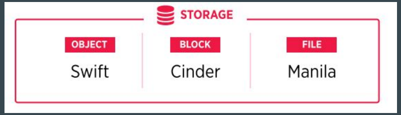
Does this have meaning?

The answer is **NO**.
It is necessary to know

- lighting,
- comet position,
- spacecraft position,
- exposure times,
- acquisition mode,
- pixel georeferencing.

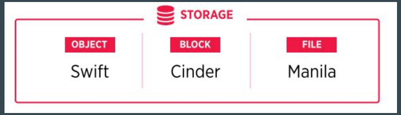
Data in the Cloud

Cloud Storage



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

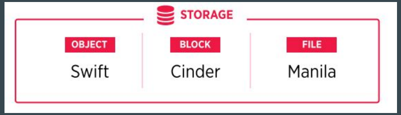
Cloud Storage



Block storage divides data into separate components made up of fixed-size data blocks, each with a unique identifier. Block storage allows the underlying storage system to retrieve it regardless of where it is stored.

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

Cloud Storage

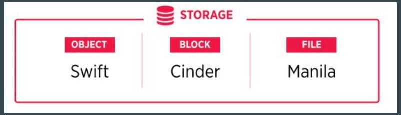


Block storage divides data into separate components made up of fixed-size data blocks, each with a unique identifier. Block storage allows the underlying storage system to retrieve it regardless of where it is stored.

File storage is the most well-known storage format: data is stored in files that can be interacted with, contained in folders within a hierarchical file directory.

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

Cloud Storage



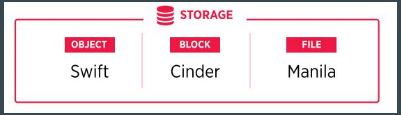
Block storage divides data into separate components made up of fixed-size data blocks, each with a unique identifier. Block storage allows the underlying storage system to retrieve it regardless of where it is stored.

File storage is the most well-known storage format: data is stored in files that can be interacted with, contained in folders within a hierarchical file directory.

Object storage is a storage format in which data is stored in separate units called objects. Each unit has a unique identifier, or key, that allows it to be located independently of where it is stored in a distributed system.

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

Cloud Storage



Block storage divides data into separate components made up of fixed-size data blocks, each with a unique identifier. Block storage allows the underlying storage system to retrieve it regardless of where it is stored.

File storage is the most well-known storage format: data is stored in files that can be interacted with, contained in folders within a hierarchical file directory.

Object storage is a storage format in which data is stored in separate units called objects. Each unit has a unique identifier, or key, that allows it to be located independently of where it is stored in a distributed system.

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

File Storage

Unix and Unix Like

- Name
- Path
- Type
- Size
- Owner (UID, GID)
- Permission
- Timestamps
  - creation
  - modify

Maxlength 255 characters

- All file names are "Case Sensitive." This means that vivek.txt, Vivek.txt, and VIVEK.txt are three different files.
- File names can use uppercase and lowercase letters as well as the symbols "." (dot) and "_" (underscore).
- Other special characters like " " (blank space) can also be used, but they require a more complex handling (they must be quoted) and are generally discouraged.
- In practice, a file name can contain any character except "/" (root folder), which is reserved as a separator between files and folders in the pathname.
- The ***null*** character cannot be used.
- Using a "." is not necessary but increases readability, especially if used to identify the file extension.
- The file name must be unique within a folder.
- A folder and a file with the same name cannot coexist within the same folder.

File Storage

- Name
- **Path**
- Type
- Size
- Owner (UID, GID)
- Permission
- Timestamps
  - creation
  - modify

The set of names required to specify a particular file in a hierarchy of folders is called the file path.
The path and the filename together form what is known as the pathname.

The path can be absolute or relative:

- In an absolute path, the entire path is specified starting from the beginning of the disk (/, root):
  /u/politi/projectb/plans/1dft
- In a relative path, the path can be indicated starting from the folder in which you are located:
  projectb/plans/1dft

A relative path cannot start with /.
Special symbols:

. indicates the current folder
.. indicates the parent folder

Maxlength 1024 characters

File Storage

Unix and Unix Like

- Name
- Path
- Type
- Size
- Owner (UID, GID)
- Permission
- Timestamps
  - creation
  - modify

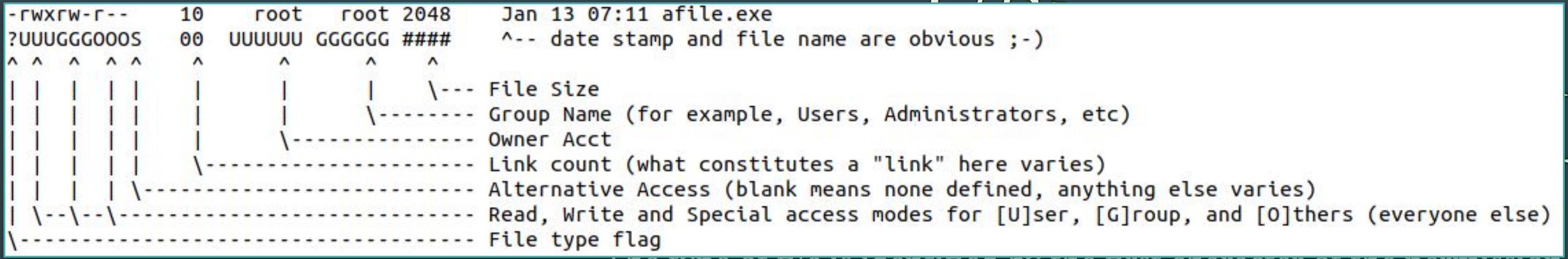
The type of file is identified by the first character of the permission string.

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

The types could be:

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

Uni Like



```
-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
```

- **Type**
- Size
- Owner (UID, GID)
- Permission
- Timestamps
  - creation
  - modify

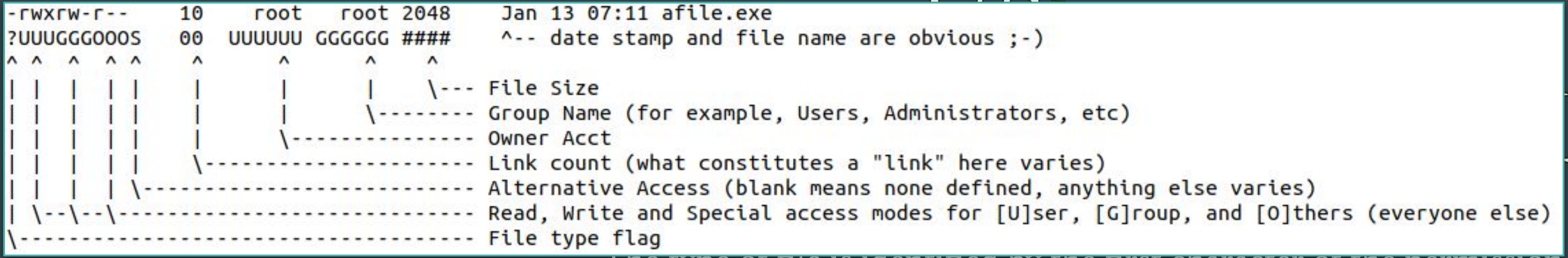
The type of file is identified by the first character of the permission string.

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

The types could be:

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

Uni Like



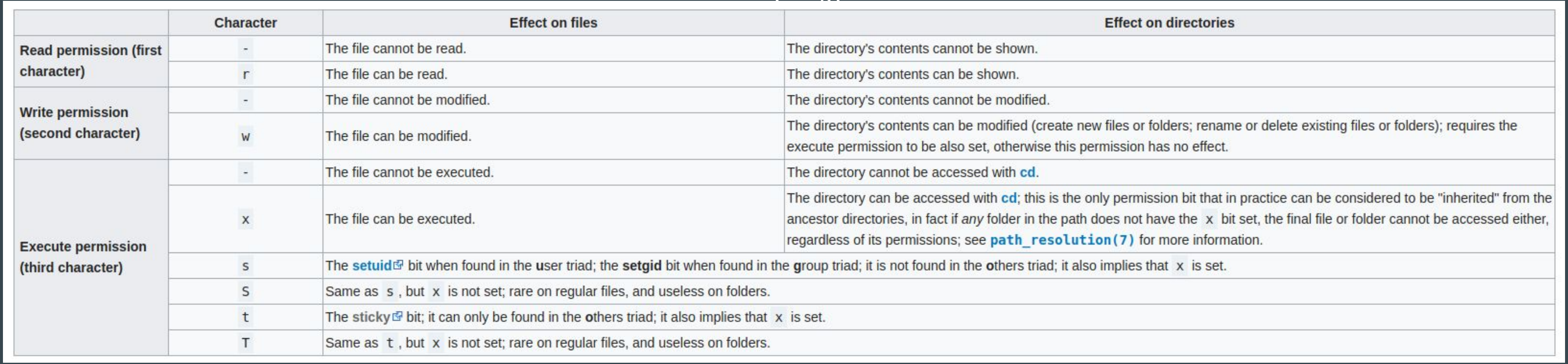
```
-rwxrw-r--    10    root   root 2048    Jan 13 07:11 afile.exe
?UUUGGGOOOS   00   UUUUUU GGGGGG ####    ^-- date stamp and file name are obvious ;-)
^ ^  ^  ^ ^    ^       ^      ^    ^
| |  |  | |    |       |      |     \--- File Size
| |  |  | |    |       |       \-------- Group Name (for example, 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
```

- **Type**
- Size
- Owner (UID, GID)
- Permission

The type of file is identified by the first character of the permission string.

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

The types could be:



	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)

File Storage

Unix and Unix Like

- Name
- Path
- Type
- Size
- Owner (UID, GID)
- Permission
- Timestamps
  - creation
  - modify

The type of file is identified by the first character of the permission string.

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

The types could be:

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

Object Storage

In object storage, data is broken down into discrete units called objects and stored in a single repository (Bucket) instead of as files within folders or as blocks on servers.

The volumes of object storage function as modular units: each one is an independent repository that contains the data, a unique identifier that allows an object to be located in a distributed system, and the metadata that describes the data.

Metadata is important and includes details such as age, privacy/security, and access restrictions.

Object Storage

In object storage, data is broken down into discrete units called objects and stored in a single repository (Bucket) instead of as files within folders or as blocks on servers.

The volum pendent repository located in a distrib

Object storage **metadata** can be extremely detailed and capable of storing information about where a video was filmed, the type of camera used, and the actors appearing in each frame.

Metadata is important and includes details such as age, privacy/security, and access restrictions.

The Data Preservation

In data management, Data Preservation is the act of safeguarding and maintaining both the security and integrity of data. Preservation is achieved through formal activities governed by policies, regulations, and strategies designed to protect and prolong the existence and authenticity of data and its associated metadata.

The Data Preservation

In data management, Data Preservation is the act of safeguarding and maintaining both the security and integrity of data. Preservation is achieved through formal activities governed by policies, regulations, and strategies designed to protect and prolong the existence and authenticity of data and its associated metadata.

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

The Data Preservation

In data management, Data Preservation is the act of safeguarding and maintaining both the security and integrity of data. Preservation is achieved through formal activities governed by policies, regulations, and strategies designed to protect and prolong the existence and authenticity of data and its associated metadata.

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

The Data Preservation

In data management, Data Preservation is the act of safeguarding and maintaining both the security and integrity of data. Preservation is achieved through formal activities governed by policies, regulations, and strategies designed to protect and prolong the existence and authenticity of data and its associated metadata.

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

Short-term Preservation.
Access to digital materials for a defined period during which use is expected, but which does not extend beyond the foreseeable future and/or until it becomes inaccessible due to technological changes.

The Data Preservation

In data management, Data Preservation is the act of safeguarding and maintaining both the security and integrity of data. Preservation is achieved through formal activities governed by policies, regulations, and strategies designed to protect and prolong the existence and authenticity of data and its associated metadata.

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

Long-term Preservation
Continuous access to digital materials, or at least to the information contained in them, indefinitely.

Version Control

Git

Git is a distributed version control software that can be used via the command line interface, created by Linus Torvalds in 2005.

A **Distributed Version Control System** (DVCS) is a type of version control that allows tracking changes and versions made to software source code without needing to use a central server, as in traditional cases.

With this system, developers can collaborate individually and in parallel, working on their own development branch, recording their changes (commits), and later sharing or merging them with others' changes—all without the need for a centralized server. This system enables various modes of collaboration, as the server is merely a support tool.

https://github.com/torvalds/linux

Glossary

repository: It is a "folder" that contains all the files needed for your project, including the files that track all versions of the project.

clone: It is the local version of the repository.

remote: It is the remote version of the repository that can be modified by anyone with access to the repository.

branch: "Branches" are used in Git to implement isolated features, that is, developed independently from each other but starting from the same root.

fork: A copy of a repository belonging to another user.

commit: A snapshot of the local repository compressed with SHA, ready to be transferred from the clone to the remote or vice versa.

tag: A marker used to highlight specific commits.

First Steps

Git can be downloaded from https://git-scm.com/downloads (all Linux distributions have Git among the available packages).

Once the software is installed, to "copy" a repository locally, you simply use the clone command.

For example, for the course repository:

```
git clone https://github.com/RomoloPoliti-INAF/PhDCourse2024.git
```

First steps

Git can be downloaded from https://git-scm.com/downloads (all Linux distributions have Git among the available packages).

Once the software is installed, to "copy" a repository locally, you simply use the clone command.

For example, for the course repository:

```
git clone https://github.com/RomoloPoliti-INAF/PhDCourse2024.git
```

Windows users need to install the Git application by downloading it from the GitHub website.
After installation, it will be necessary to restart the machine.

Fundamental Git commands

clone: Create a local copy of a remote repository

pull: Update the local copy of the repository

add: Add one or more files to the list of contents in the local repository

commit: Record changes to the repository

push: Update the remote repository

Fundamental Git commands

clone: Create a local copy of a remote repository

pull: Update the local copy of the repository

add: Add one or more files to the list of contents in the local repository

commit: Record changes to the repository

push: Update the remote repository

Examples of using git can be found in the '*Examples/01 - git*' folder of the course repository.

Course Overview

Cloud
- ~~Cloud structure~~
- ~~Data in the Cloud~~
- ~~Cloud Computing~~

Data
- ~~Data and Metadata~~
- ~~Archives~~
- Relational and not-relational Database

Computing
- Retrieval
- Manipulation
- Visualization

Environment
- Virtualization and Containers
- Microservices
- DevOps

Coding
- Fundamentals of Coding
- Python
- Versioning and Documentation

Relational Database

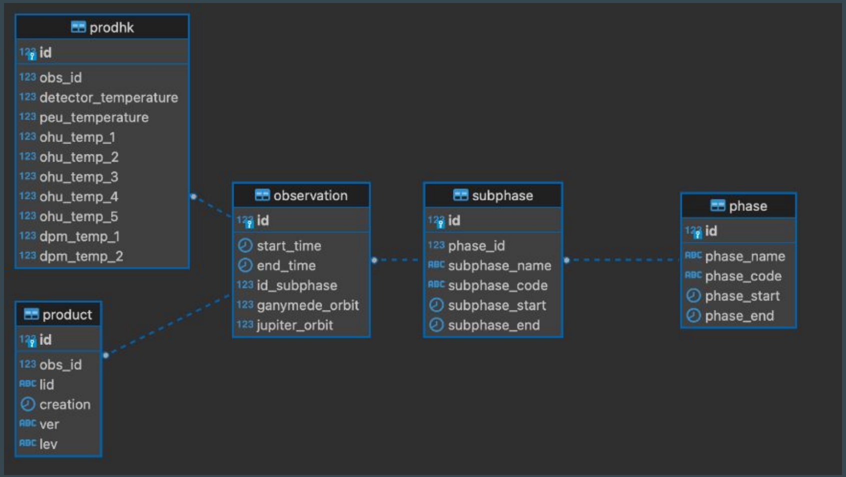
Introduction

The term "relational database management system" (**RDBMS**) refers to a database management system based on the relational model introduced by Edgar F. Codd.

In addition to these, although less commercially widespread, there are other database management systems that implement data models alternative to the relational one: hierarchical, network, and object-oriented models.

Among the various relational and object-oriented databases, PostgreSQL is the most widely used.

Entity Relationship Diagram



IDEF1X model

Entity Relationship Diagram - Glossary

Schema	A group of entities with their relationships.
Entity	They represent classes of objects (facts, things, people, ...) that have common properties and autonomous existence for the purposes of the application of interest. An occurrence of an entity is an object or instance of the class that the entity represents. This is not about the value that identifies the object, but about the object itself. An interesting consequence of this is that an occurrence of an entity has an existence independent of the properties associated with it. In a schema, each entity has a name that uniquely identifies it and is graphically represented by a rectangle with the entity's name inside it.
Relationship	They represent a relationship between two or more entities. The number of entities linked is indicated by the degree of the association: a good E-R schema is characterized by a prevalence of associations with a degree of two.
Tupla	A series of attributes that describe the entities. All objects of the same entity class have the same attributes: this is what is meant when talking about similar objects.
Attribute	A characteristic of the entity.

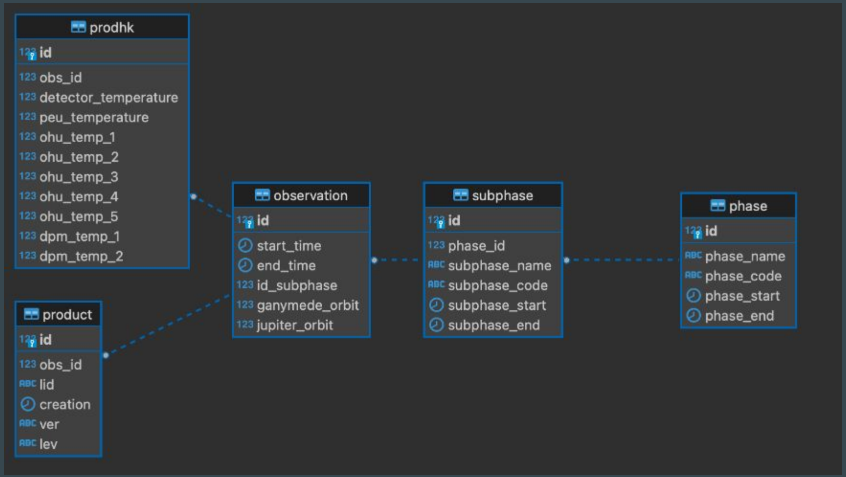
Entity Relationship Diagram - Glossary

The choice of attributes reflects the level of detail with which we want to represent the information of individual entities and relationships.

For each entity or association class, a key is defined.

The key is a minimal set of attributes that uniquely identifies an entity instance.

Entity Relationship Diagram



UML (Unified Modeling Language)

IDEF1X model

SQL Language

To give some examples of SQL language, we will use SQLite.

You can find a frontend here: https://sqlitebrowser.org

SQL Language

To give some examples of SQL language, we will use SQLite.

You can find a frontend here: https://sqlitebrowser.org

Schema	Entity	Tupla	Attribute	Relationship

Database	Table	Record	Field	Foreign Key

SQL - Basic Commands

CREATE	Create a database or a table
INSERT	Create one or more records in a table
DROP	Delete a database or a table
DELETE	delete one or more records
ALTER	Modify a database or a table
UPDATE	Modify a record
SELECT	Select a series of records.

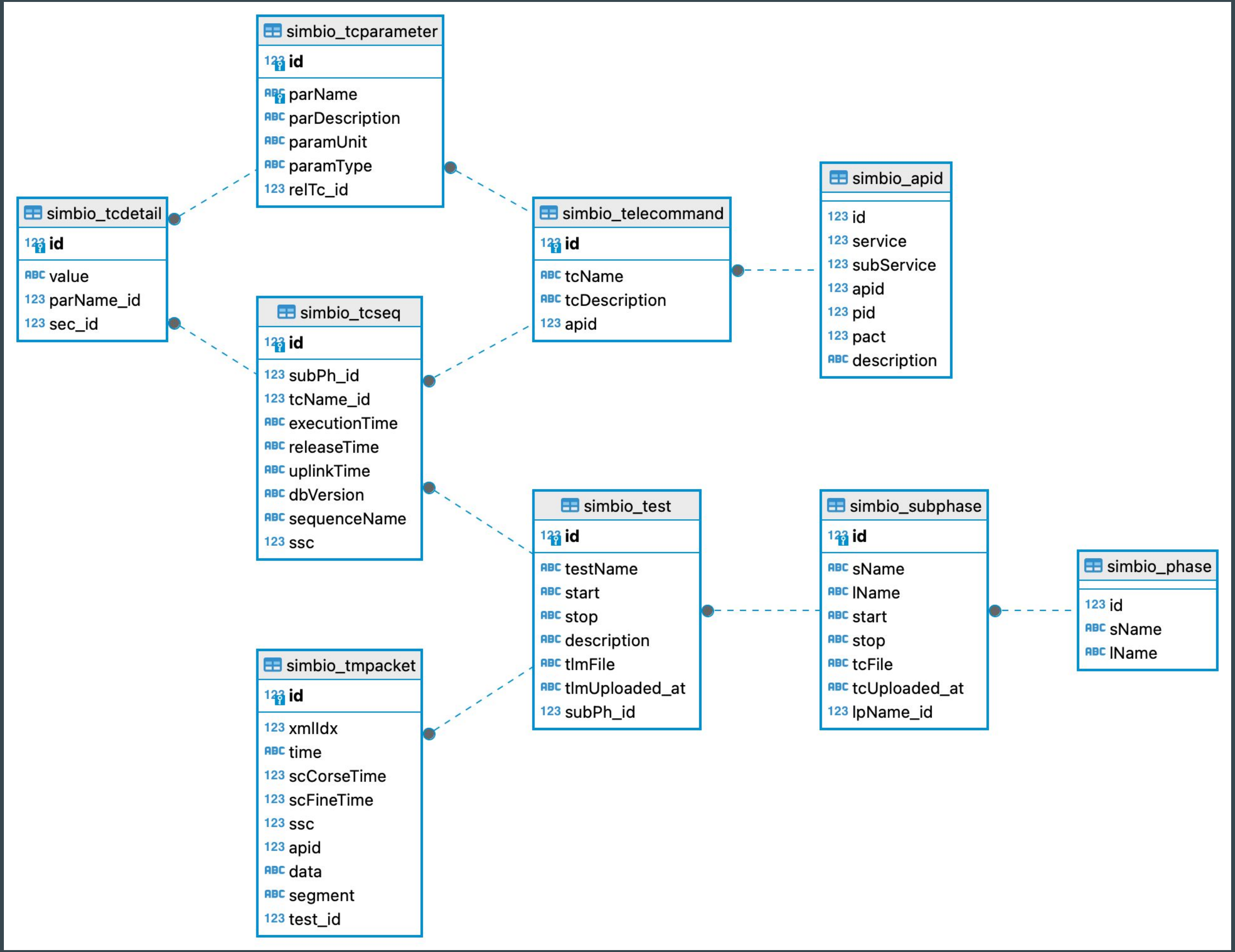
SQL - Basic Commands

CREATE	Create a database or a table
INSERT	Create one or more records in a table
DROP	Delete a database or a table
DELETE	delete one or more records
ALTER	Modify a database or a table
UPDATE	Modify a record
SELECT	Select a series of records.

Query

Examples

The examples use the database example.sqlite found in the *Examples/03 - sqlLite* folder of the repository. In the same folder is the file Script.sql with all the examples. On the side, you'll find the ER diagram of the database.



Select

```
SQL> SELECT table.field FROM table;
```

Select

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

Example 1: I want the list of all the tests (*simbio_test* table) present in my DB.

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

Select

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

Example 1: I want the list of all the tests (*simbio_test* table) present in my DB.

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

Example 2: From the previous list, I only want the name and start and end times.

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

Select

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

Example 1: I want the list of all the tests (*simbio_test* table) present in my DB.

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

Example 2: From the previous list, I only want the name and start and end times.

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

$\underbrace{\texttt{simbio\_test st}}$

Table alias. Equivalent to:
simbio_test AS *st*

Select

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

Example 1: I want the list of all the tests (*simbio_test* table) present in my DB.

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

Example 2: From the previous list, I only want the name and start and end times.

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

Example 3: the same info as in example 2 but only those performed on 11/12/2018.

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

Select

Example 4: I want all the fields of the sub-phases of the “CRUISE” phase ordered chronologically (knowing that phase and sub-phase are linked through an 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;
```

Exercise

Select the first VIHI science telemetry performed on 11/12/2018 and provide the time it was executed and the integration time.

Exercise - Solution

Select the first VIHI science telemetry performed on 11/12/2018 and provide the time it was executed and the integration time.

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

[OUT] 306

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

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

Exercise - Solution

Select the first VIHI science telemetry performed on 11/12/2018 and provide the time it was executed and the integration time.

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

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

[OUT] 578

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

[OUT] 3
```

Exercise - More elegant solution

Select the first VIHI science telemetry performed on 11/12/2018 and provide the time it was executed and the integration time.

```
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
```

Lecture September 18th 2024

eXtensible Markup Language - XML

What is l’XML

XML (short for eXtensible Markup Language) is a metalanguage for defining markup languages, meaning a language based on a syntactic mechanism that allows for defining and controlling the meaning of elements contained in a document or text.

What is l’XML

XML (short for eXtensible Markup Language) is a metalanguage for defining markup languages, meaning a language based on a syntactic mechanism that allows for defining and controlling the meaning of elements contained in a document or text.

In logic and the theory of formal languages, a **metalanguage** is understood as a formally defined language whose purpose is to define other artificial languages, known as target languages or object languages (in the context of SGML and XML, the term applications is also used). Such a definition tends to be formally rigorous and complete, so it can be used for the construction or validation of computer tools that support the target languages.

What is l'XML

XML (short for eXtensible Markup Language) is a metalanguage for defining markup languages, meaning a language based on a syntactic mechanism that allows for defining and controlling the meaning of elements contained in a document or text.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<users>
      <user years="20">
            <name>Ema</name>
            <surname>Princi</surname>
            <address>Torino</address>
      </user>
      <user years="54">
            <name>Max</name>
            <surname>Rossi</surname>
            <address>Roma</address>
      </user>
</users>
```

What is l’XML

XML (short for eXtensible Markup Language) is a metalanguage for defining markup languages, meaning a language based on a syntactic mechanism that allows for defining and controlling the meaning of elements contained in a document or text.

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

What is l'XML

XML (short for eXtensible Markup Language) is a metalanguage for defining markup languages, meaning a language based on a syntactic mechanism that allows for defining and controlling the meaning of elements contained in a document or text.

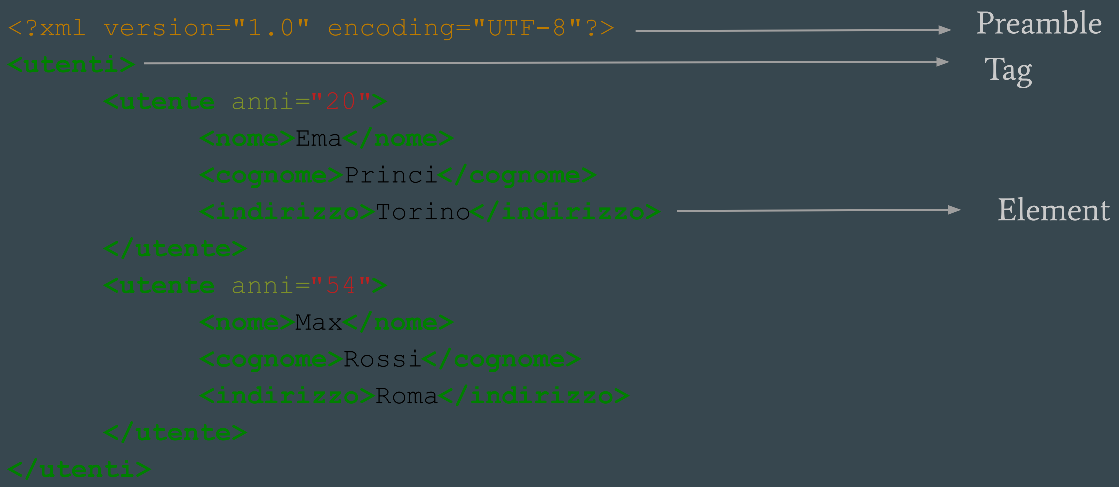
```
<?xml version="1.0" encoding="UTF-8"?>
<utenti>
      <utente anni="20">
            <nome>Ema</nome>
            <cognome>Princi</cognome>
            <indirizzo>Torino</indirizzo>
      </utente>
      <utente anni="54">
            <nome>Max</nome>
            <cognome>Rossi</cognome>
       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

Preamble

Tag

What is l’XML

XML (short for eXtensible Markup Language) is a metalanguage for defining markup languages, meaning a language based on a syntactic mechanism that allows for defining and controlling the meaning of elements contained in a document or text.



```
<?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>
```

Python Programming Fundamentals

Python Basics

Language: Python 3.12

Develop Environment: Microsoft Visual Studio Code

Topics

- Package e Modules
- Variables
- Class and Objects
- Software Versioning
- Conditional Statements
- Operators
- Loops
- Functions
- Decorators
- Namespace
- Lambda
- I/O
- Exceptions
- PyPI

Packages:

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

Basic elements

Line comment character: #

Multilines Comment :

’’’

…

’’’

Indentation

Basic elements

Line comment character: #

Multilines Comment :

’’’

’’’

Indentation

PEP

PEP stands for **Python Enhancement Proposal**. A PEP is a design document that provides information to the Python community or describes a new feature for Python or its processes or environment.
A PEP should provide a concise technical specification of the feature and a rationale for the feature.

- **Standards Track PEP** describes a new feature or implementation for Python;
- **Informational PEP** describes the design of a new feature, sets general guidelines, or provides information to the Python community;
- **Process PEP** describes a Python process or proposes a change to a process.

(PEP 1)

The most important of all is PEP 8, **Style Guide for Python Code**, which standardizes how code should be written in Python.

Whenever information is derived from a PEP, we will indicate (PEP#) where # is the PEP number.

Dunder Methods or Magic Methods

The dunder methods or variables (starting and ending with double underscores '__'.) are defined by built-in classes in Python and commonly used for operator overloading

The most common are

__version__ variable with the version number of the software

__author__ variable with the author name

We will examine individual dunders as we encounter them in our code.

Shell or Script?

There are two main methods for running Python commands:

Shell or Script?

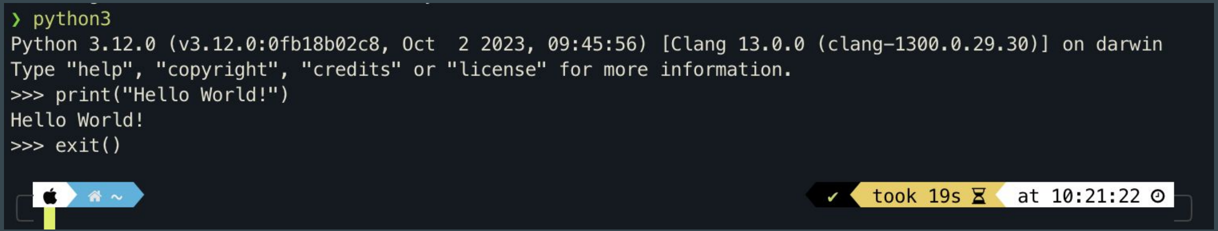
There are two main methods for running Python commands:

Using the python shell (python3)

Shell or Script?

There are two main methods for running Python commands:

Using the python shell (python3)



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

Shell or Script?

There are two main methods for running Python commands:

Using the python shell (python3)

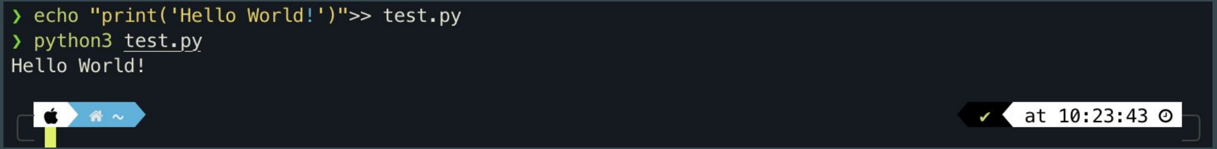
Using a Python script

Shell or Script?

There are two main methods for running Python commands:

Using the python shell (python3)

Using a Python script



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

                                                                              ✓  at 10:23:43
```

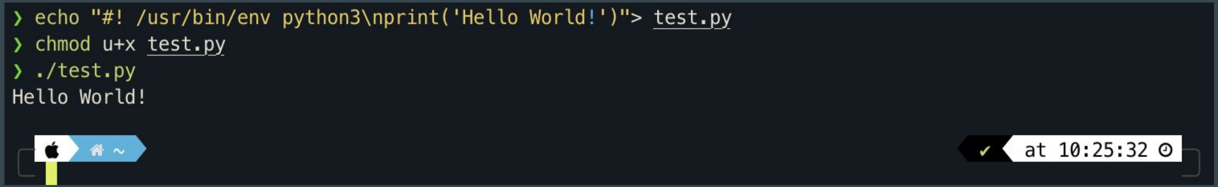
Shell o Script?

There are two main methods for running Python commands:

Using the python shell (python3)

Using a Python script

You can make a script executable.



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

Shell or Script?

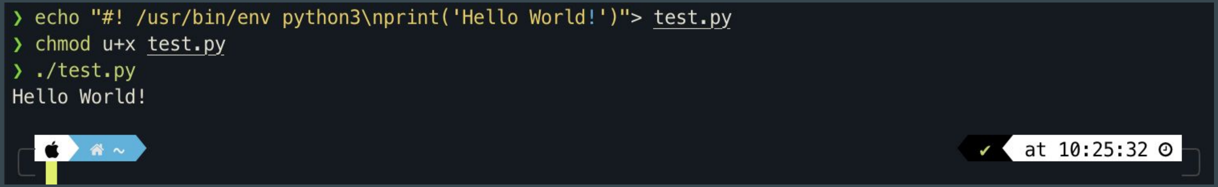
There are two main methods for running Python commands:

Using the python shell (python3)

Using a Python script

You can make a script executable.

In this case it is necessary to specify the command interpreter



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

Shell or Script?

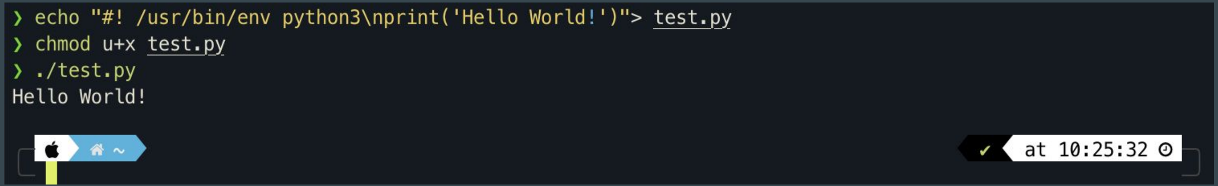
There are two main methods for running Python commands:

Using the python shell (python3)

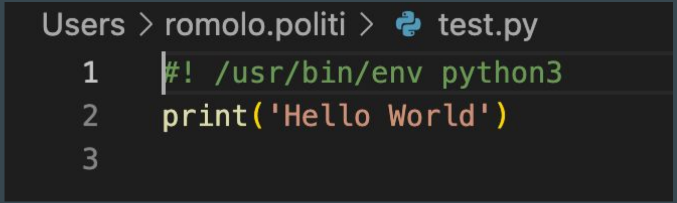
Using a Python script

You can make a script executable.

In this case it is necessary to specify the command interpreter



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



Users > romolo.politi > test.py

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

Shell or Script?

There are two main methods for running Python commands:

Using the python shell (python3)

Using a Python script

You can make a script executable.

In this case it is necessary to specify the command interpreter

We will mainly use scripts

Python Glossary

A **module** is a file containing Python definitions and statements.

- A module can define functions, classes, and variables.
- A module can also include executable code.
- Grouping related code into a module makes it easier to understand and use.
- It also makes the code logically organized.

Variables are containers for data.

- Python does not have commands to initialize variables. They are initialized upon first assignment.
- They are case-sensitive.
- They are characterized by their type.

Variable Types

To find out the type of a variable, the command type(var) is used.

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

Tipi di variabili

To find out the type of a variable, the command type(var) is used.

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 this case, we used a special type of string called a **binary string.**
Types of 'special' strings

- **b** binary string
- **f** formatted string
- **r** raw string

Class and Objects definition

Object-Oriented Programming (OOP) Vocabulary

Class: a project that consists of defining methods and attributes

Object: It’s an instance of a class. You can think of an object as something from the real world, like a yellow pen, a small dog, etc. In any case, an object can be much more abstract.

Attribute: a descriptor or a characteristic. For example, length, color, etc."

Method: an action that the class or object can receive.

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

Class and Objects

Let's see a practical example.

Versioning

Versioning:

MAJOR.MINOR.PATCH

MAJOR version when you make incompatible API changes,

MINOR version when you add functionality in a backward-compatible manner,

PATCH version when you make backward-compatible bug fixes.

Semantic Versioning 2.0.0

Versioning

Type:

- **devel** : In development
- **alpha** : In the first testing phase
- **beta** : Final testing phase
- Release Candidate: Ready for release
- final: Release version (according to semantic versioning, this type is not indicated)

A number is added to these to indicate the build, i.e., the progress of the type.

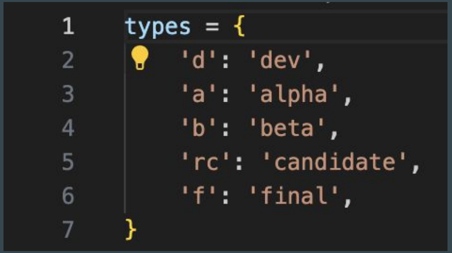
Example

In the folder *Examples/04 - Class_and_Object* you can find a jupyter notebook with some examples and a module called *version.py*

Module version (file version.py)

At the beginning of the module, you will find the initialization of a dictionary called types.

Since it is defined outside of any function or class, the dictionary has a global scope, meaning it is accessible by all objects within the module.

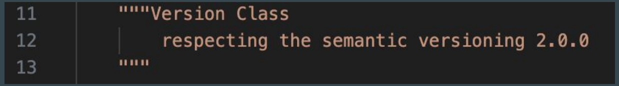


```python
types = {
    'd': 'dev',
    'a': 'alpha',
    'b': 'beta',
    'rc': 'candidate',
    'f': 'final',
}
```

Class Version

Immediately after the class definition, we find a multiline comment.

All comments placed right after the definition of a class or function are stored in the dunder variable **__doc__** . It is good practice to always include a comment after the declaration of an object or a function.



```
    """Version Class
       respecting the semantic versioning 2.0.0
    """
```

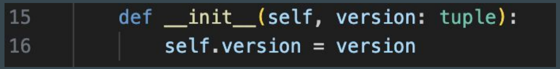
Class Version

Immediately after the class definition, we find a multiline comment.

All comments placed right after the definition of a class or function are stored in the dunder variable **__doc__** . It is good practice to always include a comment after the declaration of an object or a function.

In a class, the predefined methods are dunders. If the original methods are redefined, they are overwritten. The dunder **__init__** is called every time an object is created.

All methods of a class have the class itself as their first argument, which is usually represented by the variable **self**.



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

Classe Version

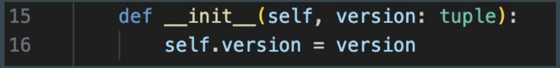
Immediately after the class definition, we find a multiline comment.

All comments placed right after the definition of a class or function are stored in the dunder variable __doc__ . It is good practice to always include a comment after the declaration of an object or a function.

In a class, the predefined methods are dunders. If the original methods are redefined, they are overwritten. The dunder __init__ is called every time an object is created.

All methods of a class have the class itself as their first argument, which is usually represented

In the definition of a function, it is good practice to indicate the type of the input variable. This allows for better code readability and, consequently, easier debugging.



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

Class Version

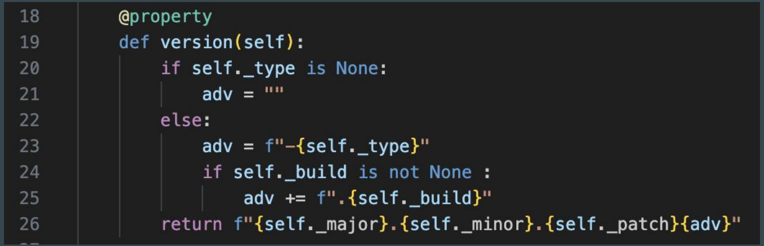
Before the definition of a function, we can find strings that start with the character @. These strings are called decorators. They are functions that allow us to manipulate the subsequent function by applying standard blocks of code before and/or after its execution. We will see later how to create a decorator.

The @property decorator in Python is a concise and clean way to define getters, setters, and deleters for class attributes.

Class Version

The function with the **@property** decorator is the **getter** , meaning it is the function that is called every time the attribute (or property) of the class is accessed.

In our case, every time we access the version attribute, it returns a string composed of the version numbers.



```
    @property
    def version(self):
        if self._type is None:
            adv = ""
        else:
            adv = f"-{self._type}"
            if self._build is not None :
                adv += f".{self._build}"
        return f"{self._major}.{self._minor}.{self._patch}{adv}"
```

Class Version - setter

The setter of the attribute version (@version.setter) parses the string used to initialize the class, splits it into the 5 main fields, and checks that the fourth field is a letter and is among the allowed ones, while the others are integers.

These validated values are then assigned to private attributes (which start with the character _) using the reflective function setattr.

Reflective Programming

Reflective programming is a programming paradigm that allows code to interact with itself at the metadata level. This means that the code can access and manipulate information about its own type, structure, and behavior.

In Python, reflective programming can be performed using a series of built-in functions and methods. For example, the **type()** function can be used to obtain the type of an object, the **dir()** function can be used to get a list of an object's attributes and methods, the **getattr()** function can be used to access an attribute of an object, and the **setattr()** function can be used to create an attribute of an object.

Reflective programming can be used for a variety of purposes, including:

- **Metadata manipulation:** Reflective programming can be used to access and manipulate information about the types, structures, and behavior of code.
- **Code testing:** Reflective programming can be used to write unit tests that verify the behavior of code at the metadata level.
- **Code generation:** Reflective programming can be used to generate new code or modify existing code.

Reflective programming can be a powerful tool for Python developers, but it is important to use it with caution. Reflective programming can make code more complex and difficult to maintain.

See Examples/05 - Reflective Programming

Class Version - setter

If the validation is not passed, an exception is raised using the **raise** command.

Exceptions

In Python, exceptions are events that indicate an error has occurred during the execution of a program. Exceptions can be raised by a variety of causes, including:

- **Invalid operations:** For example, dividing a number by zero or accessing a non-existent attribute of an object.
- **Runtime errors:** For example, a memory error or an I/O error.
- **Syntax errors:** For example, a punctuation error or a type error.

When an exception occurs, the normal flow of execution of the program is interrupted. The program then transfers control to an exception handler, which is a block of code designed to handle the exception.

In Python, exceptions are handled using the try-except syntax. The **try-except** syntax allows you to execute a block of code and handle any exceptions that are raised.

There are various types of exceptions in Python. Each type of exception is represented by an exception class. The Exception class is the base class for all exceptions.
Here are some examples of exception types in Python:

- ArithmeticError: Raised for arithmetic errors, such as division by zero.
- AssertionError: Raised when an assertion fails.
- AttributeError: Raised when an attempt to access an attribute of an object fails.
- EOFError: Raised when the end of a file is reached.
- ImportError: Raised when a module or package cannot be imported.
- KeyError: Raised when trying to access a key that does not exist in a dictionary.
- LookupError: Raised when trying to access an element in a sequence that does not exist.
- NameError: Raised when trying to use a name that has not been defined.
- TypeError: Raised when using an invalid data type.
- ValueError: Raised when using an invalid value.

Exception handling is an important aspect of Python programming. It helps make programs more robust and manage errors effectively.

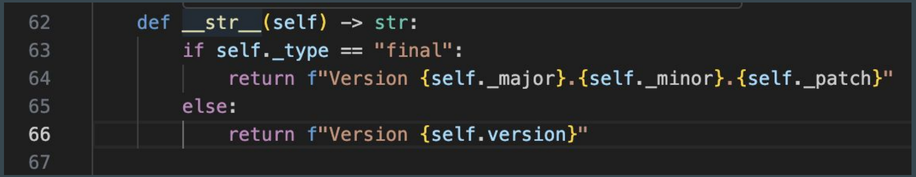
Class Version

The dunder method **__str__** is a method called when we try to convert the object to a string, e.g. try to print it.

The default is <version.Version object at 0x10d0739b0>

<module.Class object memory address>

In our case we obtain the string: Version 1.2.3



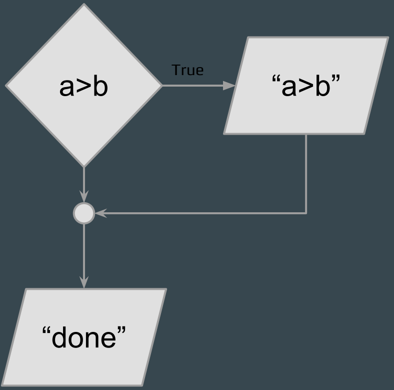
```
    def __str__(self) -> str:
        if self._type == "final":
            return f"Version {self._major}.{self._minor}.{self._patch}"
        else:
            return f"Version {self.version}"
```

Conditional Statements

The simplest is if.

It allows us to exclude a part of the code if a logical condition is not met.

```
if a > b:
   print(“a>b”)
print(“done”)
```



Conditional Statements

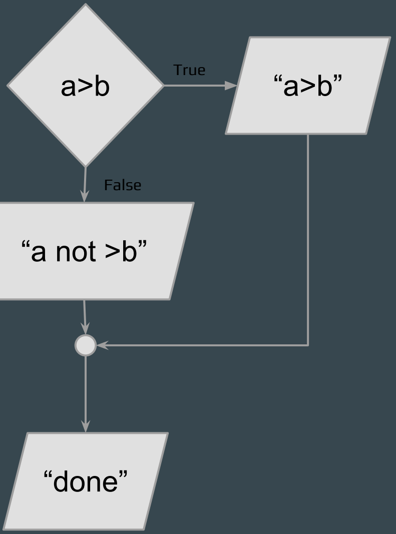
The simplest is if.

It allows us to exclude a part of the code if a logical condition is not met.

if...else

It allows us to choose the block of code to execute based on a logical condition.

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



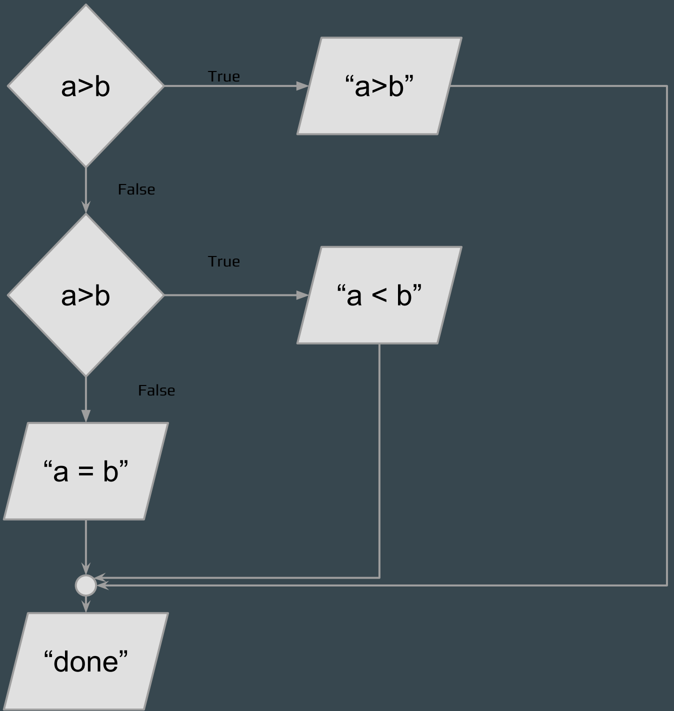
Conditional Statements

To perform multiple conditional operation we can use if…elif…else

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

Conditional Statements

To perform multiple conditional operation we can use if...elif...else



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

Conditional Statements

Python 3.10 introduced the statement match:

Conditional Statements

Python 3.10 introduced the statement match:

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

Conditional Statements

We can ***nesting*** the statements or create a composite condition

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

We can explore all the four states

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

Or all **True** or all **False**

Operators

Math

Operators

Math

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
**	Exponentiation
//	Floor division

Operators

Math

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
**	Exponentiation
//	Floor division

Modulus:

it finds the remainder or signed remainder after the division

5%2 = 1

Operators

Math

+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
**	Exponentiation
//	Floor division

Modulus:

it finds the remainder or signed remainder after the division

5%2 = 1

Floor division:

divide two numbers and return a quotient

5//2 = 2

Operators

Math

Comparison

==	Equal
!=	Not equal
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Operators

Math

Comparison

Logical

```
and
or
not
```

Operators

Math

Comparison

Logical

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

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

not True = False
not False = True

Operatori

Aritmetici

Confronto

Logici

Assignment

```
=
+= increment
-=  decrement
*=  multiplicator
/=
%=
//=
**=
^=
```

Operatori

Aritmetici

Confronto

Logici

Assignment

Identity

is
is not

Operatori

Aritmetici

Confronto

Logici

Assignment

Identity

is
is not

It allows us to choose the block of code to execute depending on a logical condition. Identity operators are used to check if two operands are equal (i.e., if they refer to the same object), meaning if they point to the same memory location

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

Operatori

Aritmetici

Confronto

Logici

Assignment

Identity

Membership

Operatori

Aritmetici

Confronto

Logici

Assignment

Identity

Membership

```
in
not in
```

Operatori

Aritmetici

Confronto

Logici

Assignment

Identity

Membership

in
not in

```
x='casa'

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

Operatori

Aritmetici

Confronto

Logici

Assignment

Identity

Membership

in
not in

```
x='casa'

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

Remember that:
'casa' == ['c','a','s','a']
```

Operatori

Aritmetici

Confronto

Logici

Assignment

Identity

Membership

Bitwise

Operatori

Aritmetici

Confronto

Logici

Assignment

Identity

Membership

Bitwise

&	**AND**	Sets each bit to 1 if both bits are 1
|	**OR**	Sets each bit to 1 if one of two bits is 1
^	**XOR**	Sets each bit to 1 if only one of two bits is 1
~	**NOT**	Inverts all the bits
<<	**Zero fill left shift**	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	**Signed right shift**	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

Operatori

Aritmetici

Confronto

Logici

Assignment

Identity

Membership

Bitwise

&	**AND**	Sets each bit to 1 if both bits are 1
|	**OR**	Sets each bit to 1 if one of two bits is 1
^	**XOR**	Sets each bit to 1 if only one of two bits is 1
~	**NOT**	Inverts all the bits
<<	**Zero fill left shift**	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	**Signed right shift**	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

```
0b110 & 0b010 = 0b010 (2)
0b100 & 0b001 = 0b000 (0)

0b110 | 0b011 = 0b111 (7)
0b110 ^ 0b011 = 0b101 (5)
```

Loops

Python has two primitive loop commands:

While a set of statements will be executed as long as a condition is true.

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

For

Loops

Python has two primitive loop commands:

While	a set of statements will be executed as long as a condition is true.
For	a set of statements will executed over a sequence.

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

Loops - while

There are statements that have the ability to control the loop.

break can stop the loop even if the while condition is true:

Loops - while

There are statements that have the ability to control the loop.

break can stop the loop even if the while condition is true:

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

Loops - while

There are statements that have the ability to control the loop.

break	can stop the loop even if the while condition is true.
continue	can stop the current iteration, and continue with the next

Loops - while

There are statements that have the ability to control the loop.

break	can stop the loop even if the while condition is true.
continue	can stop the current iteration, and continue with the next

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

Loops - while

There are statements that have the ability to control the loop.

break	can stop the loop even if the while condition is true.
continue	can stop the current iteration, and continue with the next
else	can run a block of code once when the condition no longer is true

Loops - while

There are statements that have the ability to control the loop.

break	can stop the loop even if the while condition is true.
continue	can stop the current iteration, and continue with the next
else	can run a block of code once when the condition no longer is true

```
i = 1
while i < 6:
  print(i)
  i += 1
else:
  print("i is no longer less than 6")
```

Loops - for

There are statements that have the ability to control the loop.

break	can stop the loop even if the while condition is true.
continue	can stop the current iteration, and continue with the next
else	can run a block of code once when the condition no longer is true

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

Loops - for

There are statements that have the ability to control the loop.

break	can stop the loop even if the while condition is true.
continue	can stop the current iteration, and continue with the next
else	can run a block of code once when the condition no longer is true

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

Loops - for

There are statements that have the ability to control the loop.

break	can stop the loop even if the while condition is true.
continue	can stop the current iteration, and continue with the next
else	can run a block of code once when the condition no longer is true

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

Loops - for

There are statements that have the ability to control the loop.

break	can stop the loop even if the while condition is true.
continue	can stop the current iteration, and continue with the next
else	can run a block of code once when the condition no longer is true

Remember that the strings are list

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

Loops - for

With for often are used two functions:

range	return a list of integer

Loops - for

With for often are used two functions:

range	return a list of integer

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

Loops - for

With for often are used two functions:

range	return a list of integer
enumerate	convert a collection in a enumerate list

Loops - for

With for often are used two functions:

range	return a list of integer
enumerate	convert a collection in a enumerate list

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

Lecture October 7th 2024

Topics

- ~~Package e Modules~~
- ~~Variables~~
- ~~Class and Objects~~
- ~~Software Versioning~~
- ~~Conditional Statements~~
- ~~Operators~~
- ~~Loops~~
- Functions
- Decorators
- Namespace
- Lambda
- I/O
- Exceptions
- PyPI

Packages:

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

The Functions

A function is a block of code which only runs when it is called.

The Functions

A function is a block of code which only runs when it is called.

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

 myFunct()
```

The Functions

A function is a block of code which only runs when it is called.

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

 myFunct()
```

You can pass data, known as parameters, into a function.

The Functions

A function is a block of code which only runs when it is called.

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

myFunct()
```

You can pass data, known as parameters, into a function.

```
def myFunct(arg, par1:bool = False):
   print(f"Hello {arg}")
   if par1:
        print("Today is a beautiful day")

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

The functions

If you do not know how many arguments that will be passed into your function, add a * before the parameter name in the function definition.

This way the function will receive a tuple of arguments, and can access the items accordingly.

If you do not know how many keyword arguments that will be passed into your function, add two asterisk: ** before the parameter name in the function definition.

This way the function will receive a dictionary of arguments, and can access the items accordingly:

It is possible to "extend" a function using special functions called decorators, which allow executing code before and/or after the execution of the base function.

These are applied to the function by adding @ + the name of the decorator immediately before the function definition.

Lambda Functions

Python Lambda Functions are anonymous functions means that the function is without a name. As we already know the def keyword is used to define a normal function in Python. Similarly, the lambda keyword is used to define an anonymous function in Python.

Lambda Functions

Python Lambda Functions are anonymous functions means that the function is without a name. As we already know the def keyword is used to define a normal function in Python. Similarly, the lambda keyword is used to define an anonymous function in Python.

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

print(x(5))
```

Lambda Functions

Python Lambda Functions are anonymous functions means that the function is without a name. As we already know the def keyword is used to define a normal function in Python. Similarly, the lambda keyword is used to define an anonymous function in Python.

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

print(x(5))
```

```
x = lambda a: a+10

print(x(5))
```

Lambda Functions

Python Lambda Functions are anonymous functions means that the function is without a name. As we already know the def keyword is used to define a normal function in Python. Similarly, the lambda keyword is used to define an anonymous function in Python.

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

print(x(5))
```

```
x = lambda a: a+10

print(x(5))
```

The power of lambda is better shown when you use them as an anonymous function inside another function.

Lambda Functions

Python Lambda Functions are anonymous functions means that the function is without a name. As we already know the def keyword is used to define a normal function in Python. Similarly, the lambda keyword is used to define an anonymous function in Python.

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

print(x(5))
```

```
x = lambda a: a+10

print(x(5))
```

The power of lambda is better shown when you use them as an anonymous function inside another function.

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

Lambda Functions

Python Lambda Functions are anonymous functions means that the function is without a name. As we already know the def keyword is used to define a normal function in Python. Similarly, the lambda keyword is used to define an anonymous function in Python.

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

print(x(5))
```

```
x = lambda a: a+10

print(x(5))
```

The power of lambda is better shown when you use them as an anonymous function inside another function.

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

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

When Not to Use Lambdas

1. If a name is assigned to a lambda function

When Not to Use Lambdas

1. If a name is assigned to a lambda function

```
#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

When Not to Use Lambdas

1. If a name is assigned to a lambda function
2. If a function needs to be used inside a lambda

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

#Good
map(abs, list_3)

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

When Not to Use Lambdas

1. If a name is assigned to a lambda function
2. If a function needs to be used inside a lambda

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

#Good
map(abs, list_3)

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

The *map()* function applies a specific function to each element of an iterable.

```
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))
```

When Not to Use Lambdas

1. If a name is assigned to a lambda function
2. If a function needs to be used inside a lambda
3. When using multiple lines of code makes the code more readable

Zen of 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

Finite State Machine

A finite state machine (sometimes called a finite state automaton) is a computation model that can be implemented with hardware or software and can be used to simulate sequential logic and some computer programs. Finite state automata generate regular languages. Finite state machines can be used to model problems in many fields including mathematics, artificial intelligence, games, and linguistics.

A deterministic finite automaton (DFA) is described by a five-element tuple: (Q,Σ,δ,q0,F)

- **Q** = a finite set of states
- **Σ** = a finite, nonempty input alphabet
- **δ** = a series of transition functions
- **q0** = the starting state
- **F** = the set of accepting states

There must be exactly one transition function for every input symbol in Σ from each state.

FSM - Intro

file Examples/08 - StateMachine/state.py

Argparse

To pass parameters from the command line, we use the argparse module.

```
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 this case, we introduce an optional parameter to specify the command file of the automaton and two flags to set the debug mode and the verbose mode.

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

FSM - Intro - Logging

To perform logging, we use the logging module and initialize it as follows:

```
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

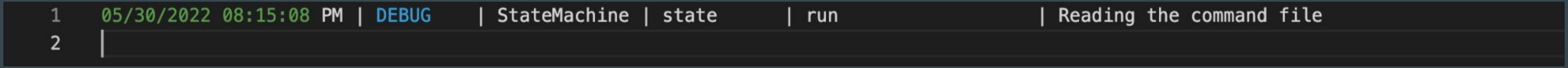
FSM - Intro - Logging

setup of Visual Studio Code:

Install the extension **Log Viewer** .

Add the following lines to the workspace file:

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

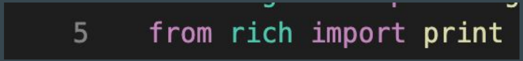


```
05/30/2022 08:15:08 PM | DEBUG    | StateMachine | state      | run                   | Reading the command file
```

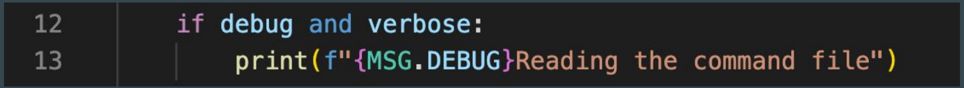
FSM - Intro - Verbosity

Verbose mode is an option available in many programming languages that would produce detailed output for diagnostic purposes thus makes a program easier to debug.

To increase the readability we will use the **rich** module



```
from rich import print
```

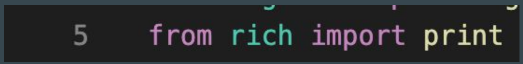


```
  if debug and verbose:
      print(f"{MSG.DEBUG}Reading the command file")
```

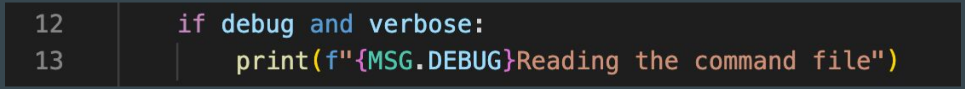
FSM - Intro - Verbosity

Verbose mode is an option available in many programming languages that would produce detailed output for diagnostic purposes thus makes a program easier to debug.

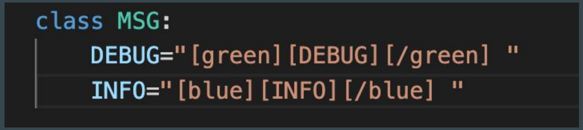
To increase the readability we will use the **rich** module



```python
from rich import print
```



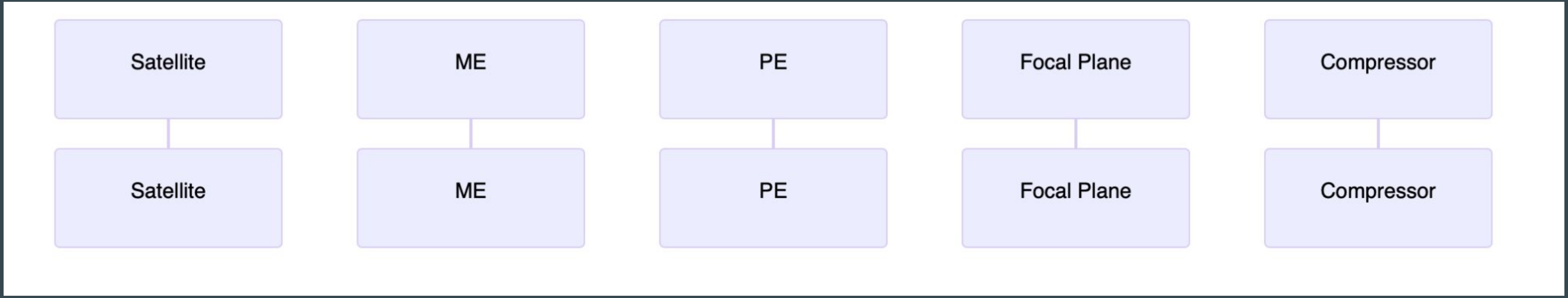
```python
    if debug and verbose:
        print(f"{MSG.DEBUG}Reading the command file")
```



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

FSM - Schema

https://mermaid.live/



ME: Main Electronics

PE: Proximity Electronics

They are the only "active" components (have a CPU).

The PE drives the HW. The ME is responsible for validating and sorting the remote controls and organizing the packages.

The compressor is usually an **FPGA** (Field Programmable Gate Array), which is an electronic hardware device made up of an integrated circuit whose processing logic functions are specifically programmable.

FSM - Database dei Comandi

The command database is where all the commands we can give to our machine are stored, along with information about the destination sub-module, the input state, and the transient and output states.

FSM - Database dei Comandi

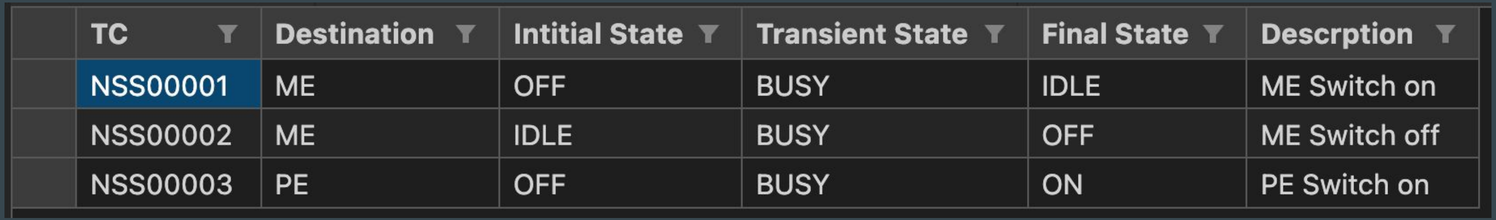
The command database is where all the commands we can give to our machine are stored, along with information about the destination sub-module, the input state, and the transient and output states.

This database in space missions is called the **Mission Information Database** (**MIB**).

FSM - Database dei Comandi

The command database is where all the commands we can give to our machine are stored, along with information about the destination sub-module, the input state, and the transient and output states.

In our case, we use a CSV file named *commandsTable.csv.*

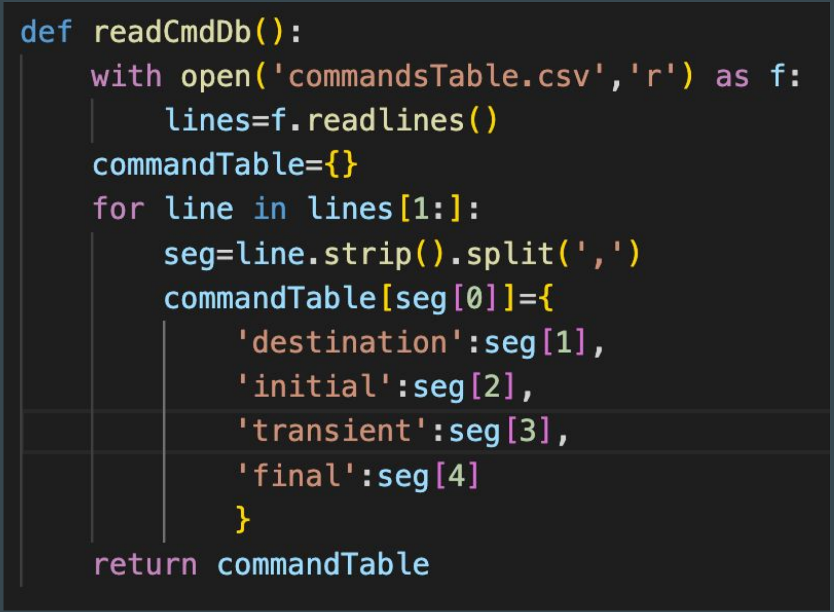


TC	Destination	Intitial State	Transient State	Final State	Descrption
NSS00001	ME	OFF	BUSY	IDLE	ME Switch on
NSS00002	ME	IDLE	BUSY	OFF	ME Switch off
NSS00003	PE	OFF	BUSY	ON	PE Switch on

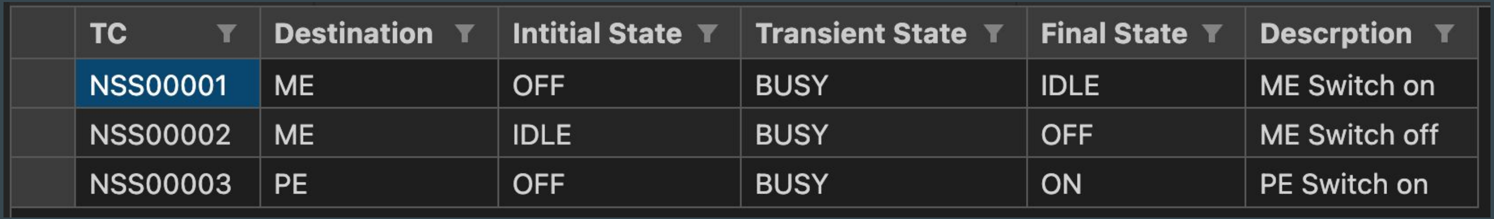
FSM - Database dei Comandi

The command database is where all the commands we can give to our machine are stored, along with information about the destination sub-module, the input state, and the transient and output states.

In our case, we use a CSV file named *commandsTable.csv.*



```python
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

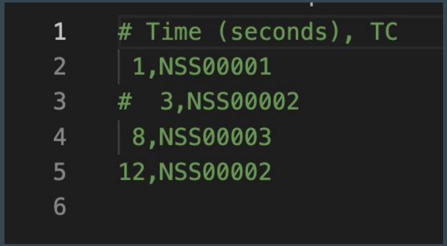
The command stack is the sequence of commands, along with their respective delays, that must be executed by the automaton.

As far as we are concerned, it is simply a text file.

FSM - Command Stack

The command stack is the sequence of commands, along with their respective delays, that must be executed by the automaton.

As far as we are concerned, it is simply a text file.



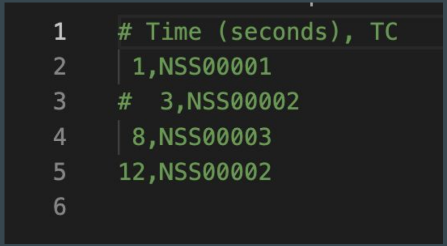
```
# Time (seconds), TC
 1,NSS00001
#  3,NSS00002
 8,NSS00003
12,NSS00002

```

FSM - Command Stack

The command stack is the sequence of commands, along with their respective delays, that must be executed by the automaton.

As far as we are concerned, it is simply a text file.

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

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

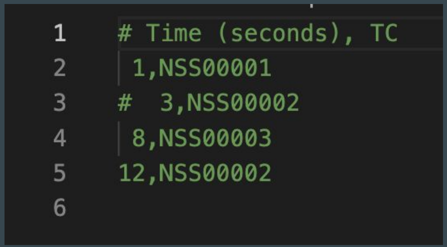


```
# Time (seconds), TC
 1,NSS00001
#  3,NSS00002
 8,NSS00003
12,NSS00002

```

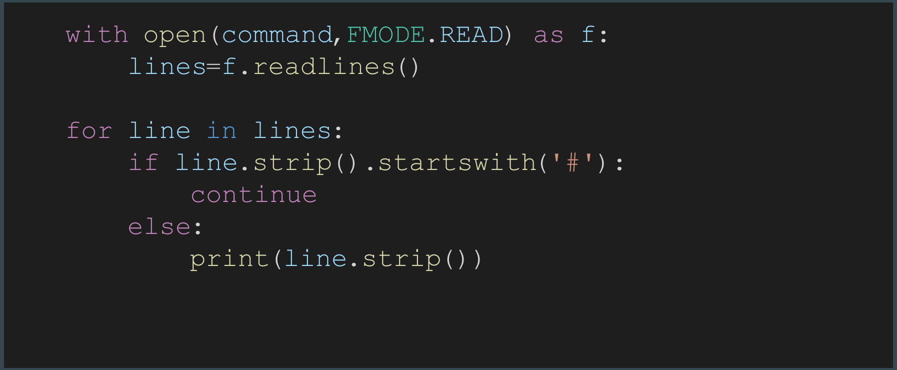
FSM - Command Stack

The command stack is the sequence of commands, along with their respective delays, that must be executed by the automaton.

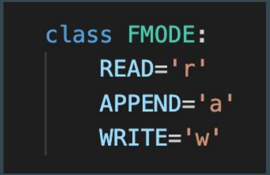
As far as we are concerned, it is simply a text file.



```
# 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())
```

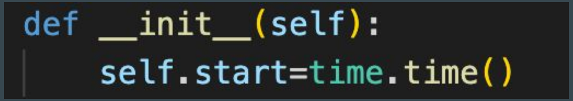


```
class FMODE:
    READ='r'
    APPEND='a'
    WRITE='w'
```

FSM - System Clock

Since commands are given in relative time, we need to create a system clock.

First, we record the moment when the machine is initialized:

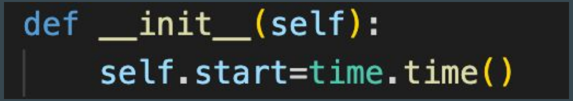


```python
def __init__(self):
    self.start=time.time()
```

FSM - System Clock

Since commands are given in relative time, we need to create a system clock.

First, we record the moment when the machine is initialized:



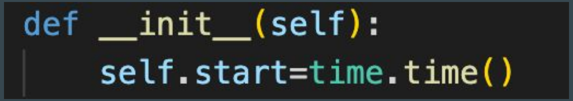
```
def __init__(self):
    self.start=time.time()
```

Then we create a method to read the time:

FSM - System Clock

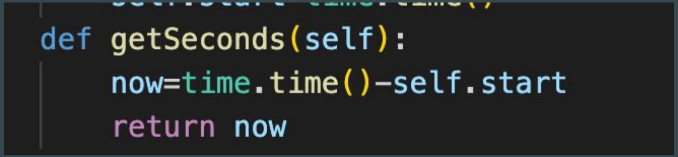
Since commands are given in relative time, we need to create a system clock.

First, we record the moment when the machine is initialized:



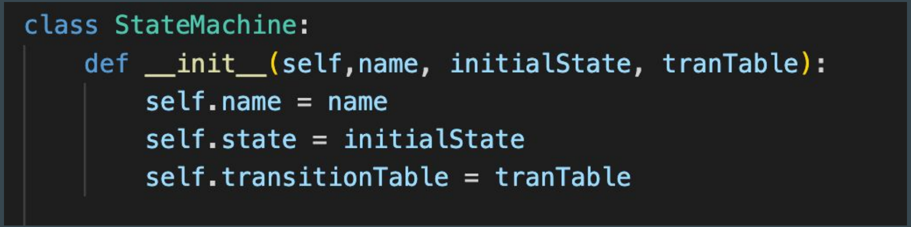
```
def __init__(self):
    self.start=time.time()
```

Then we create a method to read the time:



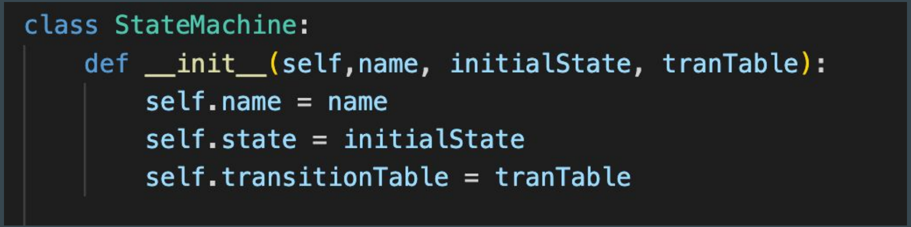
```
def getSeconds(self):
    now=time.time()-self.start
    return now
```

FSM - The Main Automaton

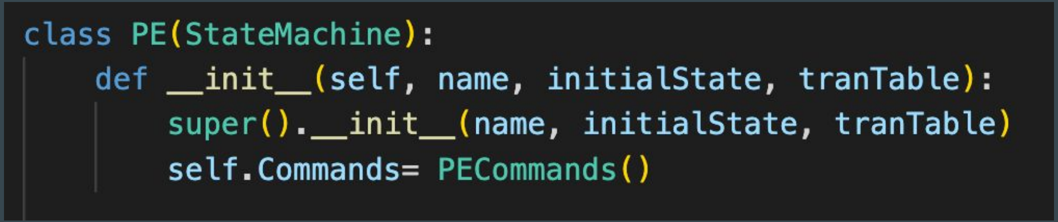


```python
class StateMachine:
    def __init__(self,name, initialState, tranTable):
        self.name = name
        self.state = initialState
        self.transitionTable = tranTable
```

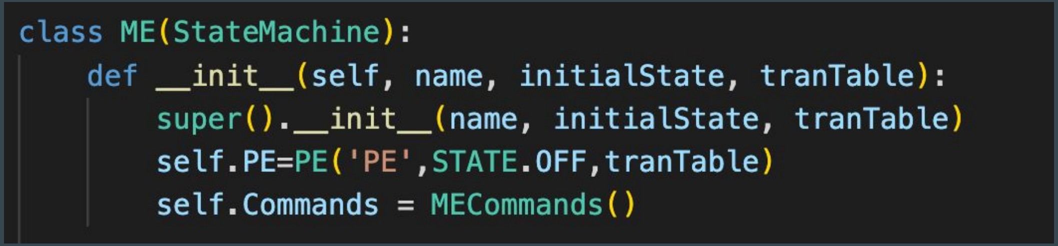
FSM - The Main Automaton



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

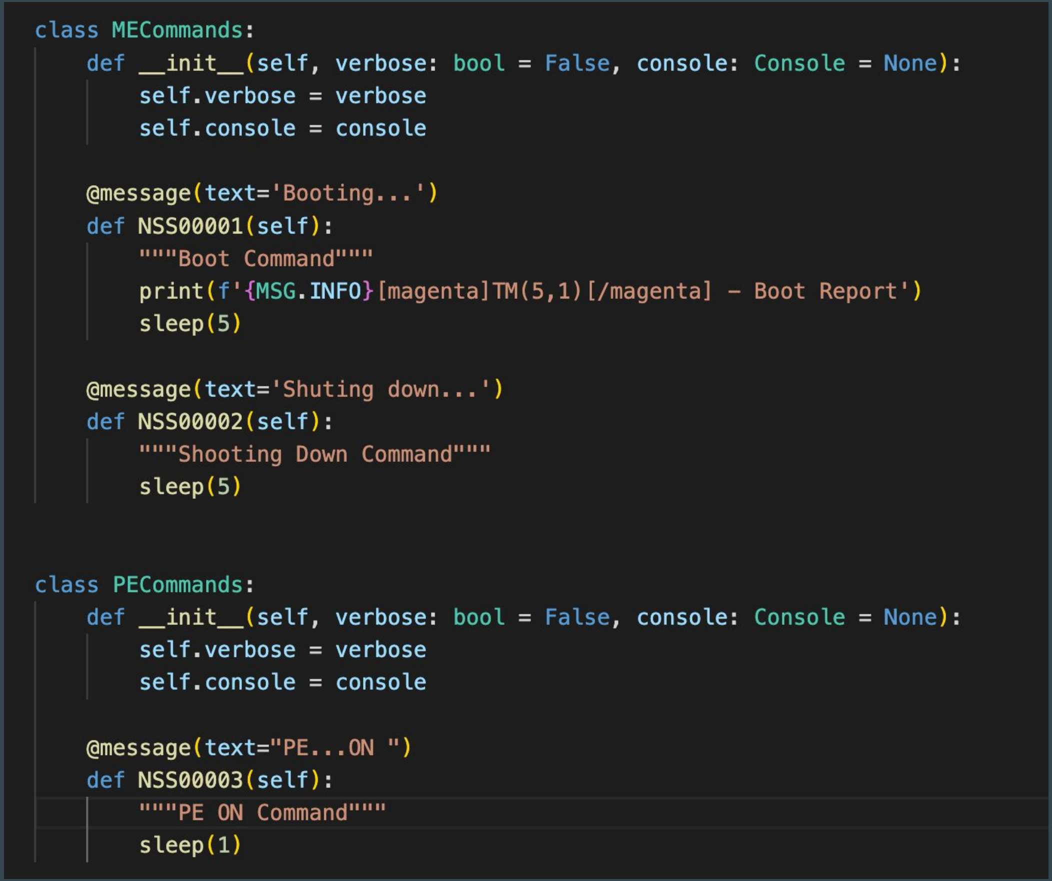


```python
class PE(StateMachine):
    def __init__(self, name, initialState, tranTable):
        super().__init__(name, initialState, tranTable)
        self.Commands= PECommands()
```



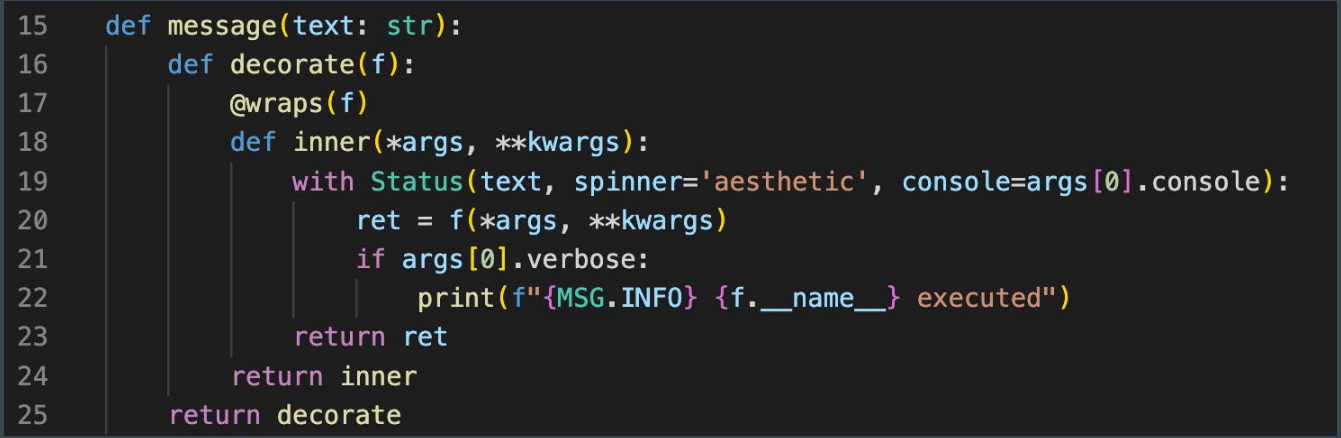
```python
class ME(StateMachine):
    def __init__(self, name, initialState, tranTable):
        super().__init__(name, initialState, tranTable)
        self.PE=PE('PE',STATE.OFF,tranTable)
        self.Commands = MECommands()
```

FSM - The Commands



```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 - The commands - Decorator



```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
```

Packages - rich

Rich is a Python library for writing "rich text" (with color and style) in the terminal and for displaying advanced content such as tables, markdown, and syntax-highlighted code.

Rich allows for visually appealing command-line applications and presents data in a more readable way. Rich can also be a useful aid for debugging through beautiful printing and syntax highlighting of data structures.

Main Modules:

- Console
- Prompt
- Progress
- Table
- Panel

https://rich.readthedocs.io/en/stable/introduction.html

Packages - rich - console

For complete control over terminal formatting, Rich offers a Console class.

It allows for managing status messages, separators, formatting, spinners, etc., in addition to providing the ability to save everything that has been printed to the screen.

Packages - rich - prompt

Rich has a series of Prompt classes that ask the user for input in a loop until a valid response is received.

An example of the functionalities can be obtained with the command:

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

Packages - rich - progress

Rich can display continuously updated information on the progress of long-running tasks/file copies, etc.

The displayed information is configurable; the default setting will show a description of the task, a progress bar, the percentage of completion, and the estimated remaining time.

An example of the functionality can be obtained with the command:

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

Packages - rich - table

The Table class in Rich offers a variety of ways to render tabular data in the terminal.

An example of the functionalities can be obtained with the command:

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

Packages - rich - panel

To draw a border around text or other renderables, you can use Panel with the renderable object as the first positional argument.

An example of the functionality can be obtained with the command:

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

FSM

Let’s go back to our car and look at the code in the module newState2.

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

Finite State Machine

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

FSM

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

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

Finite State Machine

options:

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

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

  -i, --interactive                enable the interactive mode

  -C FILE, --configure FILE        Configuration file

  -d, --debug                      Debug Mode

  -v, --verbose                    Verbose Mode
```

FSM - Configuration File

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

Writen in YAML format

YAML (pronounced ˈjæməl, rhyming with camel) is a format for data serialization that is human-readable. The language utilizes concepts from other languages such as C, Perl, and Python, as well as ideas from the XML format and the email format (RFC2822).

https://yaml.org/

FSM - Interactive Mode

Use prompts to take commands from the command line and execute them.

It transcribes the execution timeline to a file.

FSM - Final

The final code with image acquisition and compression is in the folder:

Examples/12 - StateMachine2.

In this code, the writing of packets has been suppressed to make the code more readable.