

# Ripetizioni Alex

Marini Mattia

2025

*Ripetizioni Danny* is licensed under [CC BY 4.0](#) .

© 2023 [Mattia Marini](#)

## Indice

<b>1 Database</b>	<b>4</b>
1.1 Sql . . . . .	5
1.1.1 Sintassi base . . . . .	5
1.1.2 Join . . . . .	6
1.1.3 Outer join . . . . .	7
1.1.4 Select distinct . . . . .	7
1.1.5 Operatori di aggregazione . . . . .	8
1.1.6 Operazioni insiemistiche fra più tabelle . . . . .	9
1.1.7 Subquery . . . . .	9
1.1.8 Group by e having . . . . .	10
1.1.9 Order by . . . . .	10
1.2 Esercizi schema sailors . . . . .	11
1.3 Esercizi schema università . . . . .	12
1.4 Query varie . . . . .	12
1.5 Schemi per esercizi . . . . .	13
1.5.1 Schema sailors <code>sailors.sql</code> . . . . .	14
1.5.2 Schema università <code>università.sql</code> . . . . .	15
1.5.3 Schema studenti <code>studenti.sql</code> . . . . .	20
<b>2 Linux e terminale</b>	<b>22</b>
2.1 Filesystem . . . . .	22
2.2 Comandi principali . . . . .	23
2.3 Shell indirection . . . . .	29
2.4 Creazione di script . . . . .	30
2.5 Variabili di ambiente . . . . .	30
2.6 If statements . . . . .	31
2.7 Subshell . . . . .	32
2.8 Pipes . . . . .	32
2.9 Cose utili random . . . . .	32
2.10 Esercizi . . . . .	32
<b>3 Ricorsione</b>	<b>36</b>

3.1 Esercizi . . . . .	36
<b>4 Java</b>	<b>37</b>
4.1 Aspetti simili a c++ . . . . .	38
4.1.1 Dichiarazione delle variabili: . . . . .	38
4.1.2 Commenti: . . . . .	38
4.1.3 Cicli: . . . . .	38
4.1.4 Operatori . . . . .	38
4.1.5 If e switch . . . . .	39
4.1.6 Funzioni . . . . .	39
4.2 Differenze . . . . .	39
4.2.1 Stampa su terminale . . . . .	39
4.2.2 Linguaggio interpretato vs compilato . . . . .	40
4.2.3 Memoria e puntatori . . . . .	40
4.2.4 Object orientation . . . . .	41
4.3 Esempio programma . . . . .	41
4.4 Funzioni utili . . . . .	41
4.4.1 Stringhe . . . . .	41
4.4.2 Vettori . . . . .	42
4.5 Parti utili della libreria standard . . . . .	42
4.5.1 Math . . . . .	42
4.5.2 ArrayList . . . . .	43
4.5.3 Scanner . . . . .	43
4.5.4 Random . . . . .	43
4.5.5 LinkedList . . . . .	44
4.5.6 Altre strutture dati utili . . . . .	44
4.6 Lettura/scrittura su file . . . . .	44
4.7 Try catch ed eccezioni . . . . .	44
4.7.1 Checked vs unchecked . . . . .	45
4.8 Scanner . . . . .	45
4.9 File . . . . .	46
4.10 Esercizi . . . . .	47

## Esercizi

1 Compilazione base ( <code>compilazione_base</code> ) . . . . .	32
2 Compilazione e redirect ( <code>compilazione_redirect</code> ) . . . . .	32
3 Custom cat ( <code>custom_cat_1</code> ) . . . . .	33
4 Riordina workspace ( <code>tidy</code> ) . . . . .	33
5 Smista eseguibili ( <code>smista_eseguibili_v1</code> / <code>smista_eseguibili_v2</code> ) . . . . .	33
6 Custom mkdir ( <code>custom_mkdir</code> ) . . . . .	34
7 Menu ( <code>menu</code> ) . . . . .	34
8 Inventario ( <code>inventario</code> ) . . . . .	35
9 Login ( <code>login</code> ) . . . . .	35
10 Riordina csv ( <code>csv</code> ) . . . . .	36
11 Fold ( <code>fold</code> ) . . . . .	36
12 Fattoriale ( <code>fattoriale</code> ) . . . . .	36
13 Fibonacci ( <code>fibonacci</code> ) . . . . .	36

14 Ricerca binaria (bin_search) . . . . .	37
15 Ricerca binaria (insieme_delle_parti) . . . . .	37
16 Combinazioni con step (combinazioni_step) . . . . .	37
17 Scala ottimale (scala_ottimale) . . . . .	37
18 Hello World (hello_world) . . . . .	47
19 Hello classe (estrazione) . . . . .	48
20 Geometria 1 (geometry1) . . . . .	48
21 Geometria 1 (geometry1) . . . . .	49
22 Montecarlo (montecarlo) . . . . .	49
23 Sistema gestione prenotazioni centro sportivo . . . . .	51
24 ASCII art . . . . .	52

# 1

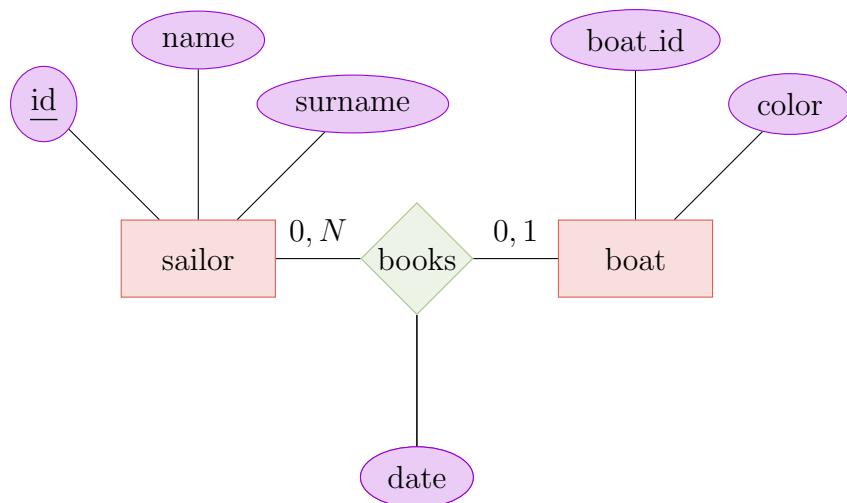
## Database

### 1.0.0 Quadro generale e introduzione ai database

Nella creazione di app e siti una componente fondamentale sta nel trovare strategie per salvare e accedere a dati in maniera efficace. Solitamente i dati vengono immagazzinati in un database e l'accesso viene effettuato tramite un linguaggio di query. Il più comune linguaggio di query è l'SQL (Structured Query Language).

### 1.0.0 Database relazionale vs database non relazionale

- DB relazionale: in un database relazionale i dati vengono salvati in strutture tabellari, che presentano relazioni tra di loro. I diagrammi entity relation forniscono una vista "ad alto livello" fra queste relazioni.
- DB non relazionale: in un database NON relazionale i dati sono salvati in strutture dati più flessibili. Ad esempio, i dati possono essere salvati tramite lo standard JSON (di base un dizionario che permette di accedere ai valori specificando una chiave)



Il seguente schema viene tradotto in tabelle come segue:

sailor		
id	name	surname
1	Bob	Bobson
2	Alice	Alicson
4	Charlie	Charlson

boat	
boat_id	color
1	red
2	blue
3	green

books		
id	date	boat_id
1	2025-01-01	1
2	2025-01-02	2

In un DB NON relazionale i dati hanno una forma che può assomigliare a qualcosa del genere:

```
{
  1 : {
    "name" : "Bob",
    "surname" : "Bobson"
    "booked_boats": [{1: {"color": "red"} }]
  }
  2: {
    "name" : "Alice",
    "surname" : "Alicson"
    "booked_boats": [{2: {"color": "blue"} }]
  }
  4: {
    "name" : "Charlie",
    "surname" : "Charlson"
    "booked_boats": []
  }
}
```

Nota come in questo caso i dati sono ”più collegati” tra di loro. Questo può creare problemi nel momento in cui i dati immagazzinati diventano più complessi. Pensa ad esempio cosa succederebbe se volessimo estrarre solo un marinaio (dovremmo estrarre necessariamente anche tutte le sue prenotazioni, il che diventa molto inefficiente su larga scala)

Noi ci concentreremo sulla prima tipologia (DB relazionali)

## 1.1 Sql

### 1.1.1 Sintassi base

Vediamo ora come utilizzare sql per estrarre i dati da un database relazionale. Una query generica ha struttura:

```
SELECT <colonna da selezionare> FROM <tabella da cui estrarre i dati>
WHERE <condizione>;
```

- All’interno di <condizione> dobbiamo possiamo fare riferimento ai nomi delle colonne
- All’interno di <tabella da cui estrarre i dati> possiamo rinominare con alias in modo da accedervi in modo più facile all’interno di <condizione>
- All’interno di <colonna da selezionare> possiamo:
  - Rinominare la colonna che verrà mostrata nell’output tramite l’operatore as

- Utilizzare il carattere \* per selezionare tutte le colonne
- Utilizzare l’ alias creato in <tabella da cui estrarre i dati> qualora vi fossero conflitti di nomi

Un esempio di una query completa:

```
SELECT * FROM sailor WHERE id = 1;

SELECT s.id FROM sailor s
WHERE s.name = 'Bob' AND s.surname = 'Johnson';
```

### 1.1.2 Join

L’operazione più importante (e anche più difficile concettualmente) è quella del join fra tabelle. In particolare, quanto in un diagramma ER abbiamo una relazione può essere necessario ”collegare” i dati opportunamente. Sempre in riferimento allo schema 1.5.1, immaginiamoci di voler fare quanto segue:

*Estrai i marinai che hanno prenotato una barca rossa*

Per fare ciò è necessario effettuare un join. La sintassi è quanto segue

```
SELECT <colonna da selezionare> FROM <tabella 1>, <tabella 2>
WHERE <id tab. 1> = <id tab. 2>;
```

Dunque per estrarre i marinai che hanno prenotato una barca rossa possiamo fare quanto segue:

```
SELECT b.sailor_id FROM books b, boat bo
WHERE AND b.boat_id = bo.boat_id AND bo.color = 'Red';
```

Nota che il join può essere fatto anche fra più tabelle. Ad esempio, se volessimo estrarre il nome e il cognome dei marinai che hanno prenotato una barca rossa possiamo fare quanto segue:

```
SELECT s.name, s.surname FROM sailor s, books b, boat bo
WHERE s.id = b.sailor_id AND b.boat_id = bo.boat_id AND bo.color = 'Red';
```

Di fatto, il join non fa altro che fare il prodotto cartesiano fra le tabelle e selezionare solo le righe che soddisfano la condizione. Il prodotto cartesiano è l’operazione che crea una nuova tabella nella quale per ogni riga della tabella a sinistra associamo tutte le righe della tabella a destra. Vediamo un esempio:

sailor			books		
id	name	surname	id	date	boat_id
1	Bob	Bobson	1	2025-01-01	1
2	Alice	Alicson	2	2025-01-02	2
4	Charlie	Charlson			

Facendo il prodotto cartesiano otteniamo:

SELECT * FROM sailors s, books b					
id	name	surname	id	date	boat_id
1	Bob	Bobson	1	2025-01-01	1
1	Bob	Bobson	2	2025-01-02	2
2	Alice	Alicson	1	2025-01-01	1
2	Alice	Alicson	2	2025-01-02	2
4	Charlie	Charlson	1	2025-01-01	1
4	Charlie	Charlson	2	2025-01-02	2

Aggiungendo una clausola **WHERE** che contiene una *chiave primaria* possiamo preservare solo le tuple che sono effettivamente legate dalla relazione in questione:

SELECT * FROM sailors s, books b WHERE s.id = books.id					
id	name	surname	id	date	boat_id
1	Bob	Bobson	1	2025-01-01	1
2	Alice	Alicson	2	2025-01-02	2

Nota che le 2 seguenti sintassi sono equivalenti:

```
SELECT * FROM sailor s JOIN books b ON s.id = b.sailor_id;
SELECT * FROM sailor s, books b WHERE s.id = b.sailor_id;
```

### 1.1.3 Outer join

Nota come negli esempi sopra, effettuando un join fra le due tabelle, nella tabella risultante non troviamo più una tupla della tabella a sinistra. A volte (anche se raramente), capita di voler mantenerla. Per questi casi esiste il cosiddetto outer join. In particolare:

- **LEFT OUTER JOIN**: mantiene le tuple della tabella a sinistra
- **RIGHT OUTER JOIN**: mantiene le tuple della tabella a destra

L'utilizzo è il medesimo del join tradizionale ma verranno inseriti valori NULL laddove necessario:

```
SELECT * FROM sailors s LEFT OUTER JOIN books b on s.id = books.id
```

id	name	surname	id	date	boat_id
1	Bob	Bobson	1	2025-01-01	1
2	Alice	Alicson	2	2025-01-02	2
4	Charlie	Charlson	NULL	NULL	NULL

### 1.1.4 Select distinct

La clausola **DISTINCT** ci permette di selezionare solo le righe distinte.

sailor		
id	name	surname
1	Bob	Bobson
2	Alice	Alicson
3	Bob	Winston
4	Charlie	Charlson

Se volessimo selezionare solo i nomi dei marinai potremmo fare quanto segue:

```
SELECT name FROM sailor;          -- Bob, Alice, Bob, Charlie  
SELECT DISTINCT name FROM sailor; -- Bob, Alice, Charlie
```

### 1.1.5 Operatori di aggregazione

#### 1.1.5 Select count

Per contare le righe di una tabella possiamo utilizzare la funzione COUNT. La sintassi è la seguente:

```
SELECT COUNT(<[DISTINCT] * | nome colonna>) FROM <tabella>;
```

Questo ritornerà il numero di tuple contenute nella tabella risultante. Se utilizziamo il DISTINCT verranno conteggiate solo le tuple distinte. Ad esempio, per contare il numero di marinai possiamo fare quanto segue:

```
SELECT COUNT(*) FROM sailor;
```

Supponendo di avere la seguente tabella: L'uso del DISTINCT cambierebbe il valore restituito:

```
SELECT COUNT(name) FROM sailor;      -- 4  
SELECT COUNT(DISTINCT name) FROM sailor; -- 3
```

#### 1.1.5 Select sum

In modo del tutto analogo a quanto descritto in 1.1.5, possiamo utilizzare la funzione SUM per sommare i valori di una colonna. La sintassi è la seguente:

```
SELECT SUM([DISTINCT]<colonna>) FROM <tabella>;
```

#### 1.1.5 Select avg

In modo del tutto analogo a quanto descritto in 1.1.5, possiamo utilizzare la funzione AVG per calcolare la media dei valori di una colonna. La sintassi è la seguente:

```
SELECT AVG([DISTINCT]<colonna>) FROM <tabella>;
```

#### 1.1.5 Select min/max

In modo del tutto analogo a quanto descritto in 1.1.5, possiamo utilizzare le funzioni MIN e MAX per calcolare il minimo e il massimo dei valori di una colonna. La sintassi è la seguente:

```
SELECT MIN([DISTINCT]<colonna>) FROM <tabella>;  
SELECT MAX([DISTINCT]<colonna>) FROM <tabella>;
```

### 1.1.6 Operazioni insiemistiche fra più tabelle

#### 1.1.6 Intersezione

Possiamo utilizzare operazioni insiemistiche fra tabelle derivanti da diverse query. Vedi ad esempio:

*Seleziona i nomi dei marinai che hanno prenotato sia una barca rossa e una verde*

```
SELECT s.name FROM sailor s, books b, boat bo
WHERE s.id = b.sailor_id AND b.boat_id = bo.boat_id AND bo.color = 'Red'
INTERSECT
SELECT s.name FROM sailor s, books b, boat bo
WHERE s.id = b.sailor_id AND b.boat_id = bo.boat_id AND bo.color =
'Green';
```

La clausola INTERSECT effettua l'intersezione insiemistica fra le due tabelle derivanti dalle due query

#### 1.1.6 Meno insiemistico

*Seleziona i nomi dei marinai che hanno prenotato una barca rossa ma non una verde*

```
SELECT s.name FROM sailor s, books b, boat bo
WHERE s.id = b.sailor_id AND b.boat_id = bo.boat_id AND bo.color = 'Red'
EXCEPT
SELECT s.name FROM sailor s, books b, boat bo
WHERE s.id = b.sailor_id AND b.boat_id = bo.boat_id AND bo.color =
'Green';
```

#### 1.1.6 Unione insiemistica

*Seleziona i nomi dei marinai che hanno prenotato una barca rossa o una verde*

```
SELECT s.name FROM sailor s, books b, boat bo
WHERE s.id = b.sailor_id AND b.boat_id = bo.boat_id AND bo.color = 'Red'
UNION
SELECT s.name FROM sailor s, books b, boat bo
WHERE s.id = b.sailor_id AND b.boat_id = bo.boat_id AND bo.color =
'Green';
```

### 1.1.7 Subquery

E' infine possibile utilizzare delle query all'interno di altre query. Questo può essere utile per eseguire operazioni più complesse. La sintassi generale è la seguente:

```
SELECT <colonna> FROM <tabella>
WHERE <operatore> (SELECT <colonna> FROM <tabella>);
```

<operatore> può avere la seguente forma:

- <colonna> IN (<subquery>): la tupla corrente viene selezionata solo se <colonna> è presente all'interno della tabella ritornata dalla subquery
- <colonna> NOT IN (<subquery>): la tupla corrente viene selezionata solo se <colonna> NON è presente all'interno della tabella ritornata dalla subquery
- EXISTS (<subquery>): la tupla corrente viene selezionata solo se la subquery ritorna almeno una tupla
- <colonna> < >|<|= > ANY : la tupla corrente viene selezionata solo se <colonna> è < >|<|= > di almeno 1 colonna della tabella ritornata dalla subquery
- <colonna> < >|<|= > ALL : la tupla corrente viene selezionata solo se <colonna> è < >|<|= > di tutte le colonna della tabella ritornata dalla subquery

#### 1.1.8 Group by e having

A volte è necessario raggruppare il risultato sulla base di un determinato attributo. Ciò è possibile utilizzando GROUP BY e HAVING. La sintassi è la seguente:

```
SELECT <colonna>, <funzione aggregazione> FROM <tabella>
GROUP BY <colonna>
HAVING <condizione>;
```

Tramite questo costrutto le tuple verranno raggruppate sulla base della colonna specificata. Per questa ragione è obbligatorio utilizzare un operatore di aggregazione. Ad esempio, se avessimo la seguente tabella:

boat	
id	color
1	red
2	blue
3	green
4	red
5	red
6	blue

*Selezione il numero di barche per ogni tipo di colore. Non considerare i colori per cui esistono meno di 2 barche*

Possiamo fare quanto segue:

```
SELECT color, COUNT(*) FROM boat
GROUP BY color;
HAVING COUNT(*) > 1
```

#### 1.1.9 Order by

Possiamo ordinare le tuple sulla base dei valori di una colonna utilizzando l'operatore ORDER BY. La sintassi è la seguente:

```
SELECT <colonna> FROM <tabella>
ORDER BY <colonna> [ASC|DESC];
```

Questo ordinerà le tuple secondo i valori della colonna specificata. Se non specificato, l'ordinamento sarà crescente. Possiamo specificare l'ordinamento decrescente utilizzando DESC.

### 1.1.9 Limit

Se siamo interessati solo a un numero limitato di tuple possiamo utilizzare l'operatore **LIMIT**. La sintassi è la seguente:

```
SELECT <colonna> FROM <tabella>
LIMIT <numero>;
```

Questo selezionerà solo le prime **<numero>** tuple risultanti dalla query.

## 1.2 Esercizi schema sailors

---

Vedi sezione 1.5.1 per lo schema

---

### 1.2.0 Query base

- Seleziona gli id dei marinai di cognome "Miller"
- Seleziona le barche rosse o blu
- Seleziona le barche che non siano rosse
- Seleziona le barche che non sono né blu né rosse

### 1.2.0 Join

- Seleziona gli id dei marinai che hanno prenotato una barca rossa
- Seleziona i nomi dei marinai che hanno prenotato una barca rossa
- Seleziona i nomi dei marinai che hanno prenotato almeno una barca
- Elenca tutti i marinai e le barche che hanno prenotato, includendo anche i marinai che non hanno mai prenotato una barca.
- Elenca tutte le barche e i marinai che le hanno prenotate, includendo anche le barche che non sono mai state prenotate.

## 1.2 Sottoquery

- Trova i marinai che hanno prenotato almeno una barca di colore rosso.
- Trova il nome del marinaio che ha prenotato il maggior numero di barche.
- Elenca i marinai che hanno prenotato almeno una barca ma non una di colore rosso.

## 1.2 Aggregazione, GROUP BY e HAVING

- Conta quante barche ha prenotato ciascun marinaio.
- Trova il colore della barca con più prenotazioni
- Elenca i marinai che hanno prenotato più di 2 barche.
- Trova il nome del marinaio che ha prenotato il maggior numero di barche.

## 1.2 Operazioni sugli Insiemi (UNION, INTERSECT, EXCEPT)

- Trova i marinai che hanno prenotato sia una barca rossa che una barca blu.
- Trova i marinai che hanno prenotato o una barca rossa o una barca blu.
- Trova i marinai che hanno prenotato almeno una barca ma non una di colore rosso.

## 1.3 Esercizi schema università

---

Vedi sezione 1.5.2 per lo schema

---

### 1.3.0 Filtraggio e Aggregazione

- Conta quanti studenti sono iscritti a ciascun dipartimento.
- Conta la media dei salari degli istruttori per ogni dipartimento.
- Conta il numero totale di crediti acquisiti dagli studenti del dipartimento di "Fisica".
- Conta gli studenti che hanno ottenuto un voto di "A" in almeno un corso.
- Seleziona gli studenti che NON hanno frequentato corsi da più di 3 crediti

## 1.4 Query varie

- Trova gli studenti che hanno seguito un corso specifico in un determinato semestre e anno.
- Mostra tutti i corsi che non sono stati assegnati a nessun istruttore (prova sia con un join che con le operazioni insiemistiche)
- Elenca gli studenti che non hanno ancora completato alcun corso (ovvero non hanno un voto registrato).
- Trova gli studenti che hanno lo stesso ID dei professori
- Elenca tutti i nomi delle persone presenti nel database (studenti e istruttori)

### 1.4.0 Query più avanzate

- Trova gli studenti con il maggior numero di crediti totali.  
*(Mostra il nome dello studente e i suoi crediti, ordinati in ordine decrescente. Mostra massimo 5 valori.)*
- Trova i corsi con il numero massimo di studenti iscritti.  
*(Mostra l'ID del corso e il numero di studenti iscritti, ordinati in ordine decrescente. Mostra massimo 5 valori)*
- Trova gli istruttori con il salario più alto per ogni dipartimento.  
*(Mostra il nome dell'istruttore, il dipartimento e il salario.)*

- Trova gli edifici che ospitano il maggior numero di aule.  
*(Mostra il nome dell'edificio e il numero di aule presenti in esso.)*
- Trova il numero di aule in cui insegna ogni insegnante  
*(Mostra il nome dell'istruttore e il numero totale di aule in cui insegna )*
- Trova i corsi seguiti solo da studenti di un singolo dipartimento.  
*(Mostra il nome del corso e il dipartimento a cui appartengono tutti gli studenti iscritti.)*
- Trova i professori che insegnano solo un corso. *(Mostra il nome del professore e il corso)*
- Trova il numero medio di crediti dei corsi offerti per ogni dipartimento.  
*(Mostra il dipartimento e la media dei crediti dei corsi che offre.)*
- Trova gli studenti che hanno frequentato più di un corso nello stesso semestre e anno.  
*(Mostra il nome dello studente, il semestre e l'anno, insieme al numero di corsi seguiti in quel periodo.)*
- Trova gli istruttori che insegnano corsi in più di un semestre.  
*(Mostra il nome dell'istruttore e il numero di semestri diversi in cui ha insegnato.)*
- Trova gli studenti che hanno preso corsi solo da istruttori del loro stesso dipartimento.  
*(Mostra il nome dello studente e il suo dipartimento.)*

## 1.5 Schemi per esercizi

Qua sotto trovi gli schemi per poterti esercitare. Ti consiglio di usare il sito di [programiz](#). Nota che

- Il codice qui sotto è fatto per funzionare sul sito di [programiz](#). Se usi altri siti potrebbe non inizializzare lo schema correttamente
- Non è necessario capire le istruzioni. A noi basta capire come prelevare i dati, non ci interessa saperli inserire

### 1.5.1 Schema sailors sailors.sql

---

Clica per aprire file **sailors.sql**

---

```
PRAGMA writable_schema = 1;
DELETE FROM sqlite_master WHERE type IN ('table', 'index', 'trigger');
PRAGMA writable_schema = 0;
VACUUM;

CREATE TABLE sailor (
    id INTEGER PRIMARY KEY,
    name TEXT NOT NULL,
    surname TEXT NOT NULL
);

CREATE TABLE boat (
    id INTEGER PRIMARY KEY,
    color TEXT NOT NULL
);

CREATE TABLE books (
    sailor_id INTEGER,
    boat_id INTEGER,
    date TEXT,
    PRIMARY KEY (sailor_id, boat_id),
    FOREIGN KEY (sailor_id) REFERENCES sailor(id) ON DELETE CASCADE,
    FOREIGN KEY (boat_id) REFERENCES boat(id) ON DELETE CASCADE
);

-- Insert random sailors
INSERT INTO sailor (name, surname) VALUES
    ('John', 'Doe'),
    ('Jane', 'Smith'),
    ('Bob', 'Johnson'),
    ('Alice', 'Williams'),
    ('Charlie', 'Brown'),
    ('David', 'Jones'),
    ('Emma', 'Miller'),
    ('Ethan', 'Davis'),
    ('Sophia', 'Martinez'),
    ('Lucas', 'Garcia');

-- Insert random boats
INSERT INTO boat (color) VALUES
    ('Red'),
    ('Blue'),
    ('Green'),
    ('Red'),
    ('Red'),
    ('Blue'),
```

```

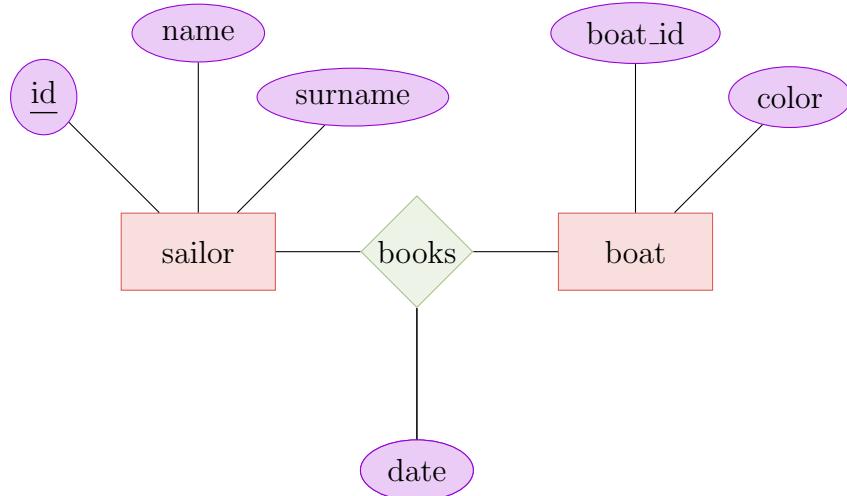
('Green'),
('Green'),
('Gray'),
('Gray');

-- Insert random books (sailor_id and boat_id as foreign keys)
INSERT INTO books (sailor_id, boat_id, date) VALUES
(1, 1, '2025-03-24'),
(1, 2, '2025-03-20'),
(2, 3, '2025-03-18'),
(2, 4, '2025-03-22'),
(3, 5, '2025-03-21'),
(3, 6, '2025-03-19'),
(4, 7, '2025-03-17'),
(4, 8, '2025-03-23'),
(5, 9, '2025-03-15'),
(5, 10, '2025-03-16');

PRAGMA INTEGRITY_CHECK;

```

Lo schema ha il seguente diagramma ER



### 1.5.2 Schema università [università.sql](#)

---

Clica per aprire file [università.sql](#)

---

```

-- Database: university (SQLite compatible)
PRAGMA writable_schema = 1;
DELETE FROM sqlite_master WHERE type IN ('table', 'index', 'trigger');
PRAGMA writable_schema = 0;
VACUUM;

PRAGMA foreign_keys = OFF;

-- Table: advisor
CREATE TABLE advisor (

```

```

    s_ID TEXT NOT NULL,
    i_ID TEXT NOT NULL,
    FOREIGN KEY (s_ID) REFERENCES student(ID),
    FOREIGN KEY (i_ID) REFERENCES instructor(ID)
);

-- Table: classroom
CREATE TABLE classroom (
    building TEXT NOT NULL,
    room_number TEXT NOT NULL,
    capacity INTEGER
);

-- Table: course
CREATE TABLE course (
    course_id TEXT PRIMARY KEY,
    title TEXT,
    dept_name TEXT NOT NULL,
    credits INTEGER
);

-- Table: department
CREATE TABLE department (
    dept_name TEXT PRIMARY KEY,
    building TEXT,
    budget REAL
);

-- Table: instructor
CREATE TABLE instructor (
    ID TEXT PRIMARY KEY,
    name TEXT NOT NULL,
    dept_name TEXT NOT NULL,
    salary REAL,
    FOREIGN KEY (dept_name) REFERENCES department(dept_name)
);

-- Table: student
CREATE TABLE student (
    ID TEXT PRIMARY KEY,
    name TEXT NOT NULL,
    dept_name TEXT,
    tot_cred INTEGER,
    FOREIGN KEY (dept_name) REFERENCES department(dept_name)
);

-- Table: takes
CREATE TABLE takes (
    ID TEXT NOT NULL,
    course_id TEXT NOT NULL,
    sec_id TEXT NOT NULL,

```

```

semester TEXT NOT NULL,
year INTEGER NOT NULL,
grade TEXT,
FOREIGN KEY (ID) REFERENCES student(ID),
FOREIGN KEY (course_id) REFERENCES course(course_id)
);

-- Table: teaches
CREATE TABLE teaches (
ID TEXT NOT NULL,
course_id TEXT NOT NULL,
sec_id TEXT NOT NULL,
semester TEXT NOT NULL,
year INTEGER NOT NULL,
FOREIGN KEY (ID) REFERENCES instructor(ID),
FOREIGN KEY (course_id) REFERENCES course(course_id)
);

-- Table: time_slot
CREATE TABLE time_slot (
time_slot_id TEXT NOT NULL,
day TEXT NOT NULL,
start_hr INTEGER NOT NULL,
start_min INTEGER NOT NULL,
end_hr INTEGER NOT NULL,
end_min INTEGER NOT NULL
);

INSERT INTO advisor (s_ID, i_ID) VALUES
('12345', '10101'),
('44553', '22222'),
('45678', '22222'),
('00128', '45565'),
('76543', '45565'),
('23121', '76543'),
('98988', '76766'),
('76653', '98345'),
('98765', '98345');

INSERT INTO classroom (building, room_number, capacity) VALUES
('Packard', '101', 500),
('Painter', '514', 10),
('Taylor', '3128', 70),
('Watson', '100', 30),
('Watson', '120', 50);

INSERT INTO course (course_id, title, dept_name, credits) VALUES
('BIO-101', 'Intro. to Biology', 'Biology', 4),
('BIO-301', 'Genetics', 'Biology', 4),

```

```
('BIO-399', 'Computational Biology', 'Biology', 3),
('CS-101', 'Intro. to Computer Science', 'Comp. Sci.', 4),
('CS-190', 'Game Design', 'Comp. Sci.', 4),
('CS-315', 'Robotics', 'Comp. Sci.', 3),
('CS-319', 'Image Processing', 'Comp. Sci.', 3),
('CS-347', 'Database System Concepts', 'Comp. Sci.', 3),
('EE-181', 'Intro. to Digital Systems', 'Elec. Eng.', 3),
('FIN-201', 'Investment Banking', 'Finance', 3),
('HIS-351', 'World History', 'History', 3),
('MU-199', 'Music Video Production', 'Music', 3),
('PHY-101', 'Physical Principles', 'Physics', 4);
```

```
INSERT INTO department (dept_name, building, budget) VALUES
('Biology', 'Watson', 90000.00),
('Comp. Sci.', 'Taylor', 100000.00),
('Elec. Eng.', 'Taylor', 85000.00),
('Finance', 'Painter', 120000.00),
('History', 'Painter', 50000.00),
('Music', 'Packard', 80000.00),
('Physics', 'Watson', 70000.00);
```

```
INSERT INTO instructor (ID, name, dept_name, salary) VALUES
('10101', 'Srinivasan', 'Comp. Sci.', 65000.00),
('12121', 'Wu', 'Finance', 90000.00),
('15151', 'Mozart', 'Music', 40000.00),
('22222', 'Einstein', 'Physics', 95000.00),
('32343', 'El Said', 'History', 60000.00),
('33333', 'Jackson', 'Biology', 120000.00),
('33456', 'Gold', 'Physics', 87000.00),
('45565', 'Katz', 'Comp. Sci.', 75000.00),
('58583', 'Califieri', 'History', 62000.00),
('76543', 'Singh', 'Finance', 80000.00),
('76766', 'Crick', 'Biology', 72000.00),
('83821', 'Brandt', 'Comp. Sci.', 92000.00),
('98345', 'Kim', 'Elec. Eng.', 80000.00);
```

```
INSERT INTO student (ID, name, dept_name, tot_cred) VALUES
('00000', 'ShinHwan Kang', 'Comp. Sci.', 100),
('00001', 'HoeHoon Jung', 'Comp. Sci.', 100),
('00128', 'Zhang', 'Comp. Sci.', 102),
('12345', 'Shankar', 'Comp. Sci.', 32),
('19991', 'Brandt', 'History', 80),
('23121', 'Chavez', 'Finance', 110),
('44553', 'Peltier', 'Physics', 56),
('45678', 'Levy', 'Physics', 46),
('54321', 'Williams', 'Comp. Sci.', 54),
('55739', 'Sanchez', 'Music', 38),
('70557', 'Snow', 'Physics', 0),
```

```

( '76543', 'Brown', 'Comp. Sci.', 58),
( '76653', 'Aoi', 'Elec. Eng.', 60),
( '98765', 'Bourikas', 'Elec. Eng.', 98),
( '98988', 'Tanaka', 'Biology', 120);

INSERT INTO takes (ID, course_id, sec_id, semester, year, grade) VALUES
( '00128', 'CS-101', '1', 'Fall', 2021, 'A'),
( '12345', 'CS-101', '1', 'Fall', 2021, 'A-'),
( '54321', 'CS-101', '1', 'Fall', 2021, 'B'),
( '19991', 'HIS-351', '1', 'Spring', 2022, 'B+'),
( '23121', 'FIN-201', '1', 'Fall', 2021, 'C'),
( '44553', 'PHY-101', '1', 'Fall', 2021, 'B'),
( '45678', 'PHY-101', '1', 'Fall', 2021, 'A'),
( '98765', 'EE-181', '1', 'Spring', 2022, 'A'),
( '98988', 'BIO-101', '1', 'Fall', 2021, 'A'),
( '76543', 'CS-315', '1', 'Spring', 2022, 'B+');

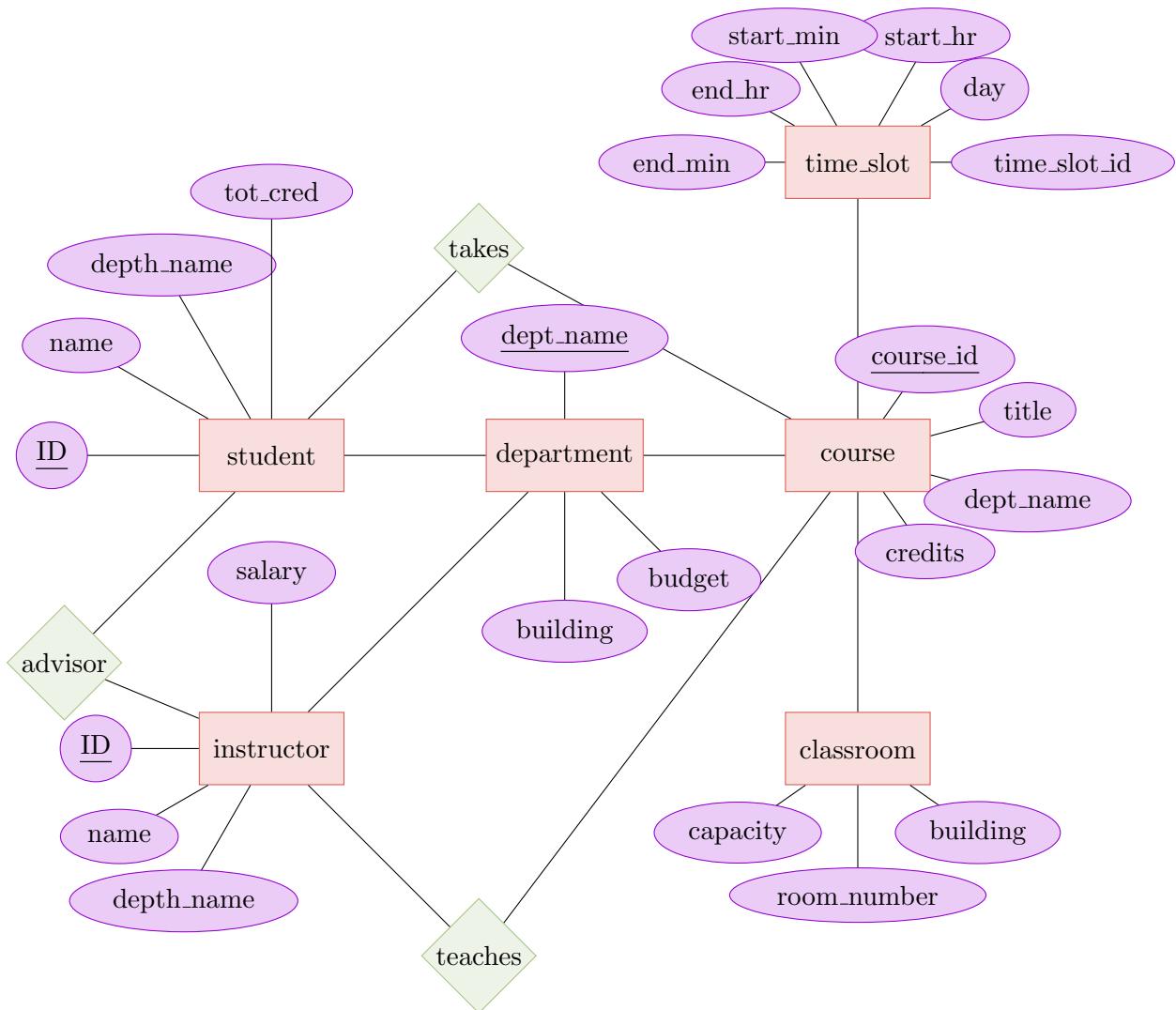
INSERT INTO teaches (ID, course_id, sec_id, semester, year) VALUES
( '10101', 'CS-101', '1', 'Fall', 2021),
( '45565', 'CS-315', '1', 'Spring', 2022),
( '76766', 'BIO-101', '1', 'Fall', 2021),
( '22222', 'PHY-101', '1', 'Fall', 2021),
( '98345', 'EE-181', '1', 'Spring', 2022),
( '58583', 'HIS-351', '1', 'Spring', 2022),
( '12121', 'FIN-201', '1', 'Fall', 2021);

INSERT INTO time_slot (time_slot_id, day, start_hr, start_min, end_hr,
end_min) VALUES
( 'A', 'Monday', 9, 0, 10, 30),
( 'A', 'Wednesday', 9, 0, 10, 30),
( 'A', 'Friday', 9, 0, 10, 30),
( 'B', 'Tuesday', 10, 0, 11, 30),
( 'B', 'Thursday', 10, 0, 11, 30),
( 'C', 'Monday', 13, 0, 14, 30),
( 'C', 'Wednesday', 13, 0, 14, 30),
( 'C', 'Friday', 13, 0, 14, 30),
( 'D', 'Tuesday', 15, 0, 16, 30),
( 'D', 'Thursday', 15, 0, 16, 30);

PRAGMA foreign_keys = ON;

PRAGMA INTEGRITY_CHECK;

```



### 1.5.3 Schema studenti [studenti.sql](#)

Clica per aprire file [studenti.sql](#)

```

PRAGMA writable_schema = 1;
DELETE FROM sqlite_master WHERE type IN ('table', 'index', 'trigger');
PRAGMA writable_schema = 0;
VACUUM;

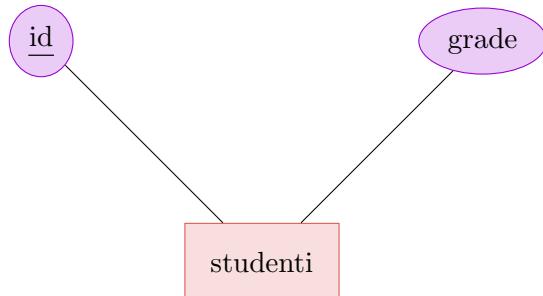
PRAGMA foreign_keys = OFF;
CREATE TABLE studenti (
    id INTEGER PRIMARY KEY,
    grade INTEGER
);

INSERT INTO studenti (id, grade) VALUES
(1, 85),
(2, 90),
(3, 78),
(4, 92),

```

```
(5, 88),  
(6, 76),  
(7, 94),  
(8, 80),  
(9, 82),  
(10, 89);
```

```
PRAGMA foreign_keys = ON;  
PRAGMA INTEGRITY_CHECK;
```



## 2

# Linux e terminale

Siamo abituati ad utilizzare un computer attraverso un'interfaccia grafica o *GUI*. Tuttavia è possibile fare tutto ciò che facciamo tramite gui attraverso comandi che possiamo digitare all'interno del *terminale*

Il vantaggio principale nell'utilizzare la riga di comando è la possibilità di creare dei file contententi codice (chiamati *script*) che ci permettono di automatizzare azioni ripetitive, ad esempio la compilazione di un codice sorgente

## 2.1 Filesystem

In ogni sistema operativo il filesystem è la struttura che ci permette di immagazzinare informazioni sul disco. Siamo abituati a navigarlo tramire un explorer (Windows file Explorer, MacOs Finder, ...). La cosa più importante da notare è che:

Il filesystem è strutturato ad albero. Ogni file o cartella ha un percorso che ci permette di trovarlo. Ad esempio, `/home/user/documents/file.txt` indica il file `file.txt` all'interno della cartella `documents` che si trova nella cartella `user` che si trova nella cartella `home`

E' importante imparare a specificare percorsi in modo corretto

- In linux e macos i percorsi sono specificati come stringhe in cui ogni cartella è separata da uno slash, ad esempio:

`/home/user/documents/file.txt`

- Se il nome di un file o una cartella contiene uno spazio questo va "escaped" tramite un backslash (\)

`/home/user/documents/file\ con\ spazio.txt`

o in alternativa il percorso va racchiuso tra apici o virgolette

`'/home/user/documents/file con spazio.txt'`  
`"/home/user/documents/file con spazio.txt"`

- Esistono percorsi speciali, in particolare:

- Percorso ".": indica la directory corrente
- Percorso "..": indica la directory padre
- Percorso "~": indica la home directory dell'utente corrente
- Percorso "/": indica la directory root

ad esempio, immaginiamo di avere aperto una sessione nel terminale nella directory `/home/user/desktop`, allora

- `./file.txt` indica `/home/user/desktop/file.txt`
- Percorso `"../documents"`: indica `/home/user/documents`
- Percorso `"~"`: indica `/home/user/mattia` (o qualunque user sia loggato)
- Percorso `"/"`: indica la directory root

## 2.2 Comandi principali

---

### 1. **ls** [directory]

stampa una lista delle directory e file nella directory corrente

---

Parametri:

- [directory]: se specificata, stampa informazioni sulla directory specificata, altrimenti ritorna ("No such file or directory (os error 2)")

Opzioni:

- **-a --all**: stampa anche i file nascosti (ossia i file il cui nome comincia per ".")
- **-l**: stampa più informazioni riguardo i file. In particolare stampa i permessi del file nel seguente formato:

**rwxrwxrwx**

ogni carattere può essere uno fra **rwx** o **-** nel caso il corrispondente permesso non sia concesso. In particolare:

- In pratica tre gruppi di **rwx**
  - \* **r** = può leggere
  - \* **w** = può scrivere
  - \* **x** = può eseguire
- I 3 gruppi hanno i seguenti significati
  - \* Primo gruppo: permessi per il proprietario del file
  - \* Secondo gruppo: permessi per il gruppo di utenti a cui appartiene il file
  - \* Terzo gruppo: permessi per tutti gli altri utenti

ad esempio

**rwxr-xr-x**

Esempi:

- Lista i file nella directory corrente:

**ls**

- Lista i file, compresi quelli nascosti nella directory corrente:

**ls --all**

- Stampa informazioni su **package.json**, se esiste nella directory corrente:

**ls package.json**

---

## 2. `mkdir <directory_name>`

Crea una directory nuova con il nome specificato

---

Parametri:

- `<directory_name>`: nome della directory da creare.

Esempi:

- Crea una directory chiamata `OSExercises` nella directory corrente:

```
mkdir OSExercises
```

---

## 3. `cd <directory_name>`

Naviga verso la directory specificata

---

Parametri:

- `<directory_name>`: nome della directory da raggiungere.

Esempi:

- Naviga verso la directory `OS_Ex`:

```
cd OS_Ex
```

- Torna alla directory padre:

```
cd ..
```

- Naviga alla directory home dell'utente corrente:

```
cd ~ oppure cd
```

---

## 4. `pwd`

Mostra il percorso completo della directory corrente

---

Esempi:

- Mostra il percorso della directory corrente:

```
pwd
```

---

## 5. `echo <string>`

Stampa una stringa o un messaggio specificato nella terminale

---

Parametri:

- `<string>`: Il messaggio o stringa da stampare.

Opzioni:

- **-e**: interpreta le escape sequences. Normalmente verranno stampate come sono
  - **\n**: nuova linea
  - **\t**: tabulazione
- **-n**: non aggiungere una nuova linea alla fine del comando

Esempi:

- Stampa il tuo nome:

```
echo "Il mio nome"
```

- Stampa due righe:

```
echo -e "Il mio nome\nIl mio cognome"
```

---

## 6. **pico <file\_name>**

Editor di testo semplice utilizzabile nella terminale

---

Parametri:

- **<file\_name>**: Nome del file da creare o aprire.

Esempi:

- Crea o modifica il file **name.txt**:

```
pico name.txt
```

---

## 7. **cat <file\_name>**

Visualizza il contenuto di uno o più file nella terminale

---

Parametri:

- **<file\_name>**: Nome del file o dei file da visualizzare.

Esempi:

- Mostra il contenuto del file **name.txt**:

```
cat name.txt
```

---

## 8. **man <command>**

Mostra il manuale di utilizzo per un comando specifico

---

Parametri:

- **<command>**: Nome del comando per cui si vuole visualizzare il manuale.

Esempi:

- Mostra il manuale del comando `cd`:

```
man cd
```

- Mostra il manuale del comando `cat`:

```
man cat
```

---

## 9. `chmod <mode> <file_name>`

Cambia i permessi di accesso di un file o directory

---

Parametri:

- `<mode>`: Specifica i nuovi permessi in base 8 (es. `644`) o come modifiche (`+w`, `-r`, ecc.).

Nota che una stringa di 3 caratteri in base 8 corrisponde ad una stringa di 9 caratteri in base 2, ad esempio

755 in base 8 → 111 101 101 in base 2  
il che corrisponde ai permessi `rwxr-xr-x`

- `<file_name>`: Nome del file o directory di cui si vogliono cambiare i permessi.

Esempi:

- Imposta i permessi di lettura e scrittura per il proprietario, solo lettura per gli altri:

```
chmod 755 executable
```

- Permetti a tutti di scrivere sul file:

```
chmod +w file.txt
```

---

## 10. `rm <file_name>`

Rimuove file o directory

---

Parametri:

- `<file_name>`: Nome del file o directory da rimuovere.

Opzioni:

- `-i`: Chiede conferma prima di rimuovere ogni file.
- `-r`: Rimuove ricorsivamente directory e il loro contenuto.

Esempi:

- Rimuove il file `BigBrother`:

```
rm BigBrother
```

- Rimuove tutti i file in una directory:

```
rm -r OSLab
```

---

## 11. cp <sourcefile> <destinationfile>

Copia un file o directory in un'altra posizione

---

Parametri:

- <sourcefile>: File o directory da copiare.
- <destinationfile>: Destinazione della copia.

Opzioni:

- -r: Copia ricorsivamente directory e il loro contenuto.

Esempi:

- Crea una copia del file **BigBrother** in /tmp con un nome diverso:

```
cp BigBrother /tmp/BigB
```

- Copia tutta la directory **OSEExercises** in **ExercisesOS**:

```
cp -r OSEExercises ExercisesOS
```

---

## 12. mv <sourcefile> <destinationfile>

Sposta o rinomina un file o directory

---

Parametri:

- <sourcefile>: File o directory da spostare o rinominare.
- <destinationfile>: Nuova posizione o nuovo nome.

Esempi:

- Sposta il file **BigBrother** nella directory **OSEExercises**:

```
mv BigBrother OSEExercises
```

- Rinomina il file **BigBrother** in **BigSister**:

```
mv BigBrother BigSister
```

---

## 13. more <file\_name>

Mostra il contenuto di un file una pagina alla volta (dall'inizio)

---

Parametri:

- <file\_name>: Nome del file da visualizzare.

Esempi:

- Visualizza il contenuto del file `ls_output` una pagina alla volta:

```
more ls_output
```

---

#### 14. `less <file_name>`

Mostra il contenuto di un file una pagina alla volta (dalla fine)

---

Parametri:

- <file\_name>: Nome del file da visualizzare.

Esempi:

- Visualizza il contenuto del file `ls_output` una pagina alla volta:

```
less ls_output
```

---

#### 15. `grep <string> <file_name>`

Cerca una stringa all'interno di un file

---

Parametri:

- <string>: Stringa da cercare.
- <file\_name>: Nome del file dove cercare.

Opzioni:

- `-v`: Mostra le righe che NON contengono la stringa.
- `-i`: Non distinguere tra maiuscole e minuscole.
- `-E`: Extended grep. Utilizzabile per regex
- `-r`: Recursive. Analizza ricorsivamente ogni file nella directory specificata

Esempi:

- Cerca la stringa "x" nel file `ls_output`:

```
grep "x" ls_output
```

- Mostra le righe che NON contengono "x" nel file `ls_output`:

```
grep -v "x" ls_output
```

Nota bene: `grep` in realtà è un comando molto avanzato e flessibile che supporta pattern matching e `regex` ([cheatsheet qui](#)).

---

## 16. `cut` [opzioni] [file]

Ritorna una porzione specificata da una riga

---

- `-d`: specifica il delimitatore (es. `-d ','` per CSV)
- `-f`: indica i campi da estrarre (usato insieme a `-d`)
- `-c`: estrae specifici caratteri

Ad esempio `echo "nome,cognome,email" | cut -d ',' -f 2` ritorna *cognome*

### 2.3 Shell indirection

Spesso può venire utile salvare ciò che il file stampa a terminale in un file. Questo può essere fatto utilizzando l'operatore di *shell indirection*. In particolare l'output di un qualsiasi comando può essere reindirizzato ad un file con questa sintassi:

<comando> > <file>

In realtà ci sono molte varianti di questo comando. La sintassi generica è

[file\_descriptor] [>|>>]

in particolare

- [file\_descriptor] indica lo stream da direzionare:
  - 1: `stdout`
  - 2: `stderr`
  - &: entrambi `stdout` e `stderr`
- > oppure >>:
  - >: crea un file nuovo e ci scrive dentro. Se il file esiste già lo sovrascrive
  - >>: appende l'output del comando in fondo file. Se il file non esiste lo crea

In modo simile può essere anche rendirizzato lo *standard input* con la seguente sintassi

<file> < <comando>

Inoltre è anche possibile reindirizzare `stderr` su `stdout` o viceversa, ad esempio:

```
echo "Hello from stdout"
echo "Hello from stderr" >&2          # Reindirizza tutto su stderr
echo "Hello from stdout" 2>&1          # Reindirizza stderr su stdout
echo "Hello from stderr" 1>&2          # Reindirizza stdout su stderr
```

## 2.4 Creazione di script

Tutto ciò che abbiamo visto fino ad ora può essere utilizzato all'interno di uno script. In particolare possiamo creare un file `script.sh`. La prima linea del file deve contenere il cosiddetto *shebang*. In particolare, se utilizziamo `bash`, dobbiamo mettere

```
#!/usr/bin/env bash
```

Ricordati di rendere eseguibile il file con il comando `chmod +x script.sh`

### 2.4.0 Argomenti

Quando eseguiamo uno script possiamo passare degli argomenti da riga di comando, elencandoli dopo il nome dell'eseguibile separati da spazi: `./script.sh arg1 arg2 ...`

Per accedere a questi all'interno dello script sono disponibili le variabili di ambiente `$1`, `$2` .... La variabile `$0` contiene il percorso dell'eseguibile

## 2.5 Variabili di ambiente

All'interno del terminale possiamo creare delle variabili che contengono informazioni utili. La sintassi è la seguente:

```
[export] <variabile>=<valore>
```

Se si include la keyword `export` la variabile è accessibile da ogni script runnato nella shell corrente. Ad esempio creando uno script `script.sh`

```
#!/usr/bin/env bash  
echo $MY_VAR
```

se da terminale eseguiamo le seguenti righe, avremmo un comportamento di questo tipo:

```
MY_VAR=42  
. ./script.sh # non stampa nulla  
  
export MY_VAR=42  
. ./script.sh # stampa 42
```

Alcune variabili d'ambiente speciali sono:

- `$USER`: nome dell'utente corrente
- `$HOME`: directory home dell'utente corrente
- `$PATH`: elenco dei percorsi di ricerca per i comandi
- `$SHELL`: shell predefinita dell'utente
- `$PWD`: directory di lavoro corrente
- `$LANG`: impostazioni della lingua e della localizzazione

## 2.6 If statements

In Bash, le istruzioni `if` compatibili con POSIX usano le parentesi quadre `[ ... ]` per eseguire test. La struttura di base è:

```
if [ condizione ]; then
    # comandi
elif [ altra_condizione ]; then
    # altri comandi
else
    # comandi di fallback
fi
```

- o 1. Confronti tra stringhe

```
[ "$str1" = "$str2" ]      # Uguali
[ "$str1" != "$str2" ]      # Diverse
[ -n "$str" ]                # Stringa non vuota (lunghezza > 0)
[ -z "$str" ]                # Stringa vuota (lunghezza == 0)
```

- o 2. Confronti numerici

```
[ "$a" -eq "$b" ]      # Uguali
[ "$a" -ne "$b" ]      # Diversi
[ "$a" -lt "$b" ]      # Minore di
[ "$a" -le "$b" ]      # Minore o uguale a
[ "$a" -gt "$b" ]      # Maggiore di
[ "$a" -ge "$b" ]      # Maggiore o uguale a
```

- o 3. Test su file

```
[ -f "file" ]      # Esiste un file normale
[ -d "dir" ]        # Esiste una directory
[ -e "file" ]        # Esiste un file qualunque
[ -r "file" ]        # È leggibile
[ -w "file" ]        # È scrivibile
[ -x "file" ]        # È eseguibile
[ -s "file" ]        # Il file non è vuoto
```

- o 4. Condizioni composte

```
[ "$a" -gt 0 ] && [ "$b" -lt 5 ]      # AND logico
[ "$a" -eq 0 ] || [ "$b" -eq 1 ]       # OR logico
! [ "$a" -eq 0 ]                      # NOT logico
```

Nota bene! Attenzione agli spazi. Ad esempio, `[ "$a" = "$b" ]` è corretto, ma `["$a"="$b"]` darà errore.

Nota come è anche possibile sfruttare la eager evaluation per avere degli statement che funzionano come operatore ternario. Ad esempio

```
[ -f script.sh ] && echo presente || echo non presente
```

questo stampa `presente` se `script.sh` esiste, non `presente` altrimenti

## 2.7 Subshell

E' possibile eseguire un comando all'interno di un altro comando. Questo può essere fatto utilizzando le parentesi quadre (). Ad esempio

```
pwd; (cd .. && pwd); pwd
```

E' possibile inoltre catturare l'output di un comando eseguito in una subshell usando il prefisso \$ ad esempio:

```
echo $(echo "Hello from subshell")
```

## 2.8 Pipes

E' spesso utile utilizzare l'output di un comando come input per un altro comando. Questo può essere fatto utilizzando il *pipe* >. Ad esempio

```
ls -l | grep "directory"
```

`grep` prenderà in input l'output del comando `ls -l`. Il comportamento è molto simile al *ridirezionamento su file* visto in precedenza, ma qui l'output viene ridirezionato da un comando all'input di un altro comando, anziche ad un file

## 2.9 Cose utili random

### 2.9.0 Terminare esecuzione di un programma

Spesso ci capita di far partire un programma e di non saperlo fare terminare. In questo caso possiamo premere `Ctrl + C` per terminare il processo corrente.

Se questo non funziona, a volte è possibile premere `Ctrl + D` due volte

## 2.10 Esercizi

### Esercizio 1: Compilazione base (*compilazione\_base*)

Creare uno script che compili e runni uno singolo file sorgente di cpp

### Esercizio 2: Compilazione e redirect (*compilazione\_redirect*)

Creare uno script `compile.sh` che compili uno singolo file sorgente di cpp, come in es. 2.10. Creare un secondo script `run.sh` che prenda in input in nome di un eseguibile e lo runni. Questo script ha due flags opzionali:

- `--suppress`: sopprime i messaggi stampati su `stdout` e `stderr`
- `--redirect`: redirrecta i messaggi stampati su `stdout` e `stderr` su due file nella directory corrente

### Esercizio 3: Custom cat (*custom\_cat\_1*)

Creare un eseguibile in c++ che stampi a video il contenuto del file `input.txt` nella cartella corrente. Creare uno script che faccia le seguenti cose:

- Compila il file `main.cpp` in un eseguibile `foo`
- Crea il file `input.txt` con contenuto "*new file*". Se il file è già presente il suo contenuto non deve essere toccato
- Runna l'eseguibile

### Esercizio 4: Riordina workspace (*tidy*)

Data una directory contenente file con estensioni `.txt` e `.cpp`, creare uno script `tidy.sh` che crei due sotto cartelle `input` e `src` nella directory corrente. Lo script accetta le seguenti opzioni:

- `--move` sposta i file
- `--copy` copia i file. Default
- `--cleanup` rimuove le cartelle `input` e `src`

in cui vengono copiati/spostati rispettivamente tutti i file.

## 2.10.0 Mockup verifica

### Esercizio 5: Smista eseguibili (*smista\_eseguibili\_v1 / smista\_eseguibili\_v2*)

Creare uno script `smista.sh` in una cartella in cui sono presenti altre 3 directories: `tmp`, `home` e `root`. Nella cartella `home` e `root` sono presenti dei file. Lo script deve creare un file `output.txt` nella cartella `tmp` in cui vengono inserite informazioni sui file nelle altre cartelle nel seguente formato:

```
File eseguibili cartella home:  
    lista dei file eseguibili della cartella home  
File eseguibili cartella root:  
    lista dei file eseguibili della cartella root  
File non eseguibili  
    lista dei file eseguibili della cartella home  
File non eseguibili cartella root:  
    lista dei file non eseguibili della cartella root
```

Stampare *un file per riga*

### Esercizio 6: *Custom mkdir (custom\_mkdir)*

Creare uno script che prenda due parametri **p1** e **p2** come input. **p1** rappresenta il percorso verso una directory, **p2** un nome sotto forma di stringa.

- Testare l'esistenza dei due parametri, altrimenti stampare un appropriato messaggio di errore e terminare la script
- Testare che il primo parametro rappresenti una directory altrimenti stampare un appropriato messaggio di errore e terminare la script
- Se tutti i test hanno dato esito positivo, creare una nuova directory all'interno della directory specificata dal primo parametro con il nome indicato nel secondo

### Esercizio 7: *Menu (menu)*

Creare un file di testo chiamato **menu.txt** contenente almeno quattro righe che elencano pizze tipiche con gli ingredienti, una pizza per riga. Ad esempio:

**Pizza pomodoro**  
**Pizza formaggio olive**  
**Pizza origano**  
**Pizza prosciutto funghi**

Scrivere uno script che stampi tutte le pizze nel menu che contengono l'ingrediente specificato.

- Creare una versione **v1** che accetti un solo argomento e stampi le pizze che contengono l'ingrediente specificato
- Creare una versione **v2** che accetti un numero arbitrario di argomenti (uno o più) e stampi le pizze che contengono *tutti* gli ingredienti specificati

### Esercizio 8: *Inventario (inventario)*

Creare uno script `inventario.sh` che esegua le seguenti operazioni:

- Riceve come parametro una directory (es. `./inventario`).
- Crea un file di output chiamato `/tmp/inventario.txt`, o `./inventario.txt`.
- Scrive tre sezioni nel file di output:
  - **FILE TESTO** – tutti i file `.txt` presenti nella directory.
  - **ALTRI** – tutti i file che non rientrano nelle due categorie precedenti.
  - **FILE AUDIO** – tutti i file `.mp3` presenti.
- Mostra il contenuto del file `inventario.txt` a schermo.
- Elimina il file temporaneo.

Ricorda bene di

- Ignorare eventuali directory presenti
- Controllare che il parametro esista e che sia una directory
- Se una sezione non contiene file, deve scrivere **Nessun file trovato**

Puoi utilizzare il comando `cut`

Puoi utilizzare il comando `file -b --mime-type` per controllare efficacemente il tipo del file

### Esercizio 9: *Login (login)*

Si ha un file `users.txt` che contiene una lista di utenti, uno per riga, nel formato:

`username-nome-cognome-password`

Lo script prende come argomenti 2 stringhe `username` e `password` e cerca un utente nel file che abbia username e password specificati.

### Esercizio 10: *Riordina csv (csv)*

Si ha un file `dati.txt` che contiene una lista di dati, uno per riga, nel formato:

`nome;cognome;email;classe`

Si stampino tutte le righe presenti nel file in un nuovo file `out.txt`, modificando il formato come segue:

`cognome;nome;classe;email`

Nella conversione ignorare le righe che iniziano con il carattere `#` (*hashtag*)

Ad esempio, se il file `dati.txt` contiene:

```
#nome;cognome;email;classe  
Luca;Rossi;luca.rossi@example.com;3  
Giulia;Bianchi;giulia.bianchi@example.com;2  
Marco;Verdi;marco.verdi@example.com;4  
Sara;Conti;sara.conti@example.com;1  
Francesco;Esposito;francesco.esposito@example.com;5
```

il file `out.txt` deve contenere:

```
Rossi;Luca;3;luca.rossi@example.com  
Bianchi;Giulia;2;giulia.bianchi@example.com  
Verdi;Marco;4;marco.verdi@example.com  
Conti;Sara;1;sara.conti@example.com  
Esposito;Francesco;5;francesco.esposito@example.com
```

## 3

## Ricorsione

Si parla di ricorsione ogni qualvolta che una funzione chiama se stessa. Questo permette di risolvere in maniera molto elegante problemi complessi.

### 3.1 Esercizi

#### Esercizio 11: *Fold (fold)*

Scrivere una procedura ricorsiva `fold` che prenda in input un array e ritorni la somma di ogni suo elemento in modo ricorsivo.

#### Esercizio 12: *Fattoriale (fattoriale)*

Scrivere un programma che calcoli il fattoriale di un numero in maniera ricorsiva. Scrivine una versione iterativa e compara la velocità di esecuzione

#### Esercizio 13: *Fibonacci (fibonacci)*

Scrivere un programma che calcoli  $n$ -esimo numero nella sequenza di Fibonacci in maniera ricorsiva. Ricorda che il numero  $i$  nella sequenza di fibonacci è dato dalla somma dei due numeri precedenti:

$$F(i) = F(i - 1) + F(i - 2)$$

una sequenza valida di fibonacci è: 1 1 2 3 5 8 13 ...

Scrivi una versione iterativa e compara la velocità di esecuzione. Scrivi una versione ricorsiva che implementi *memoization*

### Esercizio 14: Ricerca binaria (*bin\_search*)

Vi è dato un vettore ordinato di interi. Scrivere una funzione ricorsiva che prenda in input un intero e ritorni la posizione in cui si trova nel vettore, oppure -1 se non è presente. Eseguire la versione di ricerca lineare in  $O(n)$  e la versione di ricerca binaria  $O(\log(n))$ . Quest'ultima si basa sulla seguente logica:

- Controllo posizione centrale del vettore
- Se  $v[mid] > n$  allora ricerco ricorsivamente nel sottovettore sinistro
- Se  $v[mid] \leq n$  allora ricerco ricorsivamente nel sottovettore destro

Comparare i tempi di esecuzione delle due versioni

### Esercizio 15: Ricerca binaria (*insieme\_delle\_parti*)

Dato un array  $v$ , scrivere un algoritmo che generi il suo insieme delle parti

$$2^v$$

ossia un insieme contenente ogni possibile sottoinsieme di  $v$ . Ad esempio dato  $v = [1, 2, 3]$ , avrò:

$$2^v = \{[], [1], [2], [3], [1, 2], [2, 3], [1, 3], [1, 2, 3]\}$$

Occhio alla complessità!

### Esercizio 16: Combinazioni con step (*combinazioni\_step*)

Dato un intero  $n$ , un intero `upper_bound` e un intero `step`, generare tutte le combinazioni di lunghezza  $n$  in cui i valori possono variare da 0 a `upper_bound` a intervalli di `step`. Ad esempio, dato  $n=2$ , `upper_bound=4` e `step=2`, allora il programma deve stampare

`[0,0] [2,0] [0,2] [2,2] [4,0] [0,4] [4,2] [2,4] [4,4]`

### Esercizio 17: Scala ottimale (*scala\_ottimale*)

Dato un numero  $n$ , trovare il numero minimo di operazioni per arrivare a  $n$  partendo da 0. Le due operazioni possibili sono:

`+1, *2`

4

Java

Java è molto simile dal punto di vista della sintassi al c++. Non sarà molto complicato il passaggio

## 4.1 Aspetti simili a c++

La sintassi di Java è molto simile a quella di c++, ecco gli aspetti che rimangono invariati o quasi:

### 4.1.1 Dichiarazione delle variabili:

Sintassi	Differenze rispetto al c++
<code>int x = 15</code>	invariato
<code>long x = 15</code>	invariato
<code>float x = 15.0f</code>	nota il <code>15.0f</code> , dove <code>f</code> sta per float
<code>double x = 15.0</code>	float ma con precision maggiore, 64 bit
<code>boolean x = true</code>	<code>boolean</code> anzichè <code>bool</code>
<code>String s = "stringa"</code>	dato <code>String</code> sarebbe una classe ma è trattato come tipo primitivo, dato che è usato molto frequentemente
<code>String v[] = new String[15]</code>	vettore di stringhe di dimensione 15
<code>int v[] = new int[15]</code>	vettore di interi di dimensione 15

### 4.1.2 Commenti:

- Commento riga singola: `// commento`
- Commento righe multiple: `/* commento */`

### 4.1.3 Cicli:

- Ciclo for:

```
for(int i = 0; i<15; i++){
    // qualcosa
}
```

- Ciclo while:

```
while(i < 15){
    // qualcosa
}
```

### 4.1.4 Operatori

Operatore	Descrizione
<code>+</code>	operatori matematici
<code>-</code>	
<code>*</code>	
<code>/</code>	
<code>&amp;&amp;</code>	and logico
<code>  </code>	or logico
<code>!</code>	not logico

#### 4.1.5 If e switch

```
if(a<b) {  
    //codice  
}  
else(if a>b) {  
    //codice  
}  
else {  
    //codice  
}  
  
switch(espressione) {  
case x:  
    // codice  
    break;  
case y:  
    // codice  
    break;  
default:  
    // codice  
}
```

#### 4.1.6 Funzioni

```
void nome_funzione1 (int arg_1, String arg_2){  
    //corpo funzione  
}  
  
int nome_funzione2 (int arg_1, String arg_2){  
    //corpo funzione  
    return 5;  
}
```

Una funzione è quindi definita indicando nel seguente ordine, esattamente come in c++:

1. Tipo di ritorno (o `void` se non ritorna nulla)
2. Nome della funzione
3. Parametri, racchiusi fra parentesi tonde e separati da virgole
4. Corpo della funzione fra graffe

### 4.2 Differenze

#### 4.2.1 Stampa su terminale

Una delle feature usata moltissimo, ma completamente diversa dal c++ è la stampa su terminale:

```
System.out.println("Stampa questa cosa");  
//stampa andando a capo prima di stampare
```

```
System.out.print("Stampa questa cosa");  
//stampa SENZA andando a capo
```

nota che le stringhe si possono concatenare con l'operatore +:

```

String a = "Hello";
String b = "world";

System.out.println(a + b);
//stampa "Hello world"

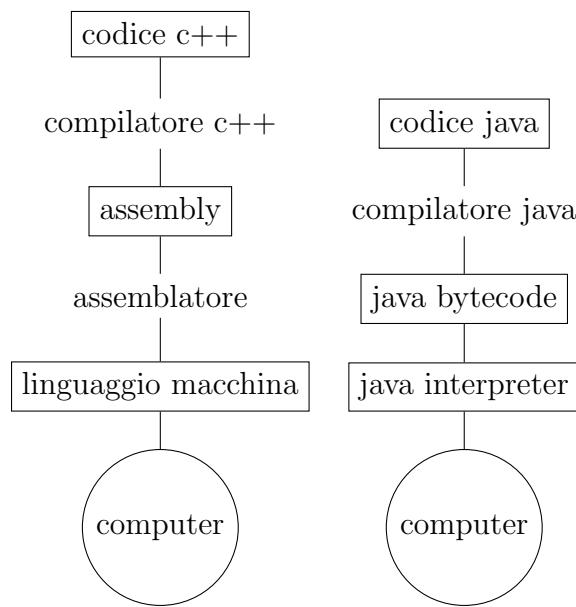
String c = a + b;
//Inizializza c a "Hello world"

```

#### 4.2.2 Linguaggio interpretato vs compilato

A differenza di c++, java è un linguaggio interpretato

- *Linguaggio compilato*: il codice è "dato in pasto" a un compilatore, il quale lo converte in linguaggio macchina (di fatto in una sequenza di, 0 ed 1)
- *Linguaggio interpretato*: il codice è "dato in pasto" ad un compilatore, il quale lo converte però in bytecode, ossia un linguaggio di basso livello (molto difficile da leggere e scrivere), il quale è in grado di essere letto da un *interprete*



#### 4.2.3 Memoria e puntatori

Java è un linguaggio ad alto livello che gestisce la memoria in maniera diversa rispetto al c++:

- c++: il compito di allocare e deallocate la memoria non più utilizzata è del programmatore
- java: la memoria viene deallocata in maniera automatica tramite un meccanismo chiamato garbage collection

Visto che in java la memoria è gestita in maniera automatica, il programmatore non ne ha accesso diretto tramite puntatori: al contrario, i puntatori non esistono

#### 4.2.4 Object orientation

Sebbene c++ sia un linguaggio che permette di utilizzare classi ed oggetti, in java l'object orientation è forzata: ogni parte del programma deve essere contenuta all'interno di una classe

#### 4.3 Esempio programma

```
class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```

Cose da notare:

- La funzione main è contenuta all'interno di una classe "HelloWorld", il cui nome è arbitrario
- La funzione main è marcata come **static**, ciò vuol dire che la funzione esiste anche se non esiste un oggetto di tipo "HelloWorld", affroteremo meglio il modificatore **static** più avanti, per ora possiamo ignorarlo
- La funzione main è marcata come **public**, ciò vuol dire che la funzione è accessibile ovunque. Affronteremo meglio questo modificatore più avanti, per ora possiamo ignorarlo
- La funzione main prende in ingresso un vettore di stringhe. Nel caso si avviasse l'applicazione da terminale e possibile passare al main dei parametri nel seguente modo:

```
cd cartella_applicazione  
. /nome_applicazione parametro_1 parametro_2 ...
```

in questo caso il vettore di stringhe **args** conterrà **parametro\_1** e **parametro\_2**.  
Penso non lo userete mai ma è buono saperlo

#### 4.4 Funzioni utili

In java sono definite alcune funzioni utilissime. Qui una lista (non esaustiva) delle più comuni:

##### 4.4.1 Stringhe

Supponiamo di avere **String s = "stringa stringa";**

Funzione	Descrizione
<code>s.length()</code>	Ritorna il numero di caratteri contenuti nella stringa (7 nel caso d'esempio)
<code>s.charAt(int index)</code>	Ritorna il carattere in posizione <code>index</code>
<code>s.indexOf(char carattere)</code>	Ritorna l'indice della prima occorrenza di <code>carattere</code> in <code>s</code>
<code>s.indexOf(String stringa)</code>	Ritorna l'indice della prima occorrenza della sottostringa <code>stringa</code> in <code>s</code>

```
String s = "stringa stringa";
s.length(); // 15
s.charAt(2); // 'r'
s.indexOf('r'); // 2
s.indexOf("ga"); // 5
```

#### 4.4.2 Vettori

Supponiamo di avere `int v[] = new int[15];`

Funzione	Descrizione
<code>v.length</code>	ritorna il numero di elementi contenuti nel vettore

### 4.5 Parti utili della libreria standard

La libreria standard di java offre moltissime classi utili. Vediamo qui le più comuni

#### 4.5.1 Math

Funzione	Descrizione
<code>Math.exp(float n)</code>	Ritorna $e^n$
<code>Math.log(float n)</code>	Ritorna $\ln(n)$
<code>Math.abs(float x)</code>	Ritorna $ x $ (valore assoluto di $x$ )
<code>Math.sin(float x)</code>	Ritorna $\sin(x)$
<code>Math.cos(float x)</code>	Ritorna $\cos(x)$
<code>Math.tan(float x)</code>	Ritorna $\tan(x)$
<code>Math.asin(float x)</code>	Ritorna $\arcsin(x)$
<code>Math.acos(float x)</code>	Ritorna $\arccos(x)$
<code>Math.atan(float x)</code>	Ritorna $\arctan(x)$
<code>Math.max(float a, float b)</code>	Ritorna l'elemento maggiore fra $a$ e $b$
<code>Math.min(float a, float b)</code>	Ritorna l'elemento minore fra $a$ e $b$
<code>Math.floor(float x)</code>	Arrotonda per difetto $x$
<code>Math.ceil(float x)</code>	Arrotonda per eccesso $x$
<code>Math.round(float x)</code>	Arrotonda $x$

#### 4.5.2 ArrayList

Utile per avere un vettore di lunghezza variabile. Si inizializza nel seguente modo:

```
ArrayList< tipo > nomeArray = new ArrayList< tipo >(dimensione)
```

Nota che:

- **dimensione** si può omettere, ottenendo un vettore con dimensione nulla
- **tipo** deve essere una classe. Se voglio un arrayList di tipi primitivi devo utilizzare le classi wrapper (**Integer**, **Boolean**, **Double** ...)

Metodo	Descrizione
v.get( index )	ritorna l'elemento all'indice <b>index</b>
v.set( index , element)	setta l'elemento a indice <b>index</b>
v.add( element )	inserisce <b>element</b> in fondo
v.remove ( index )	rimuove l'elemento a indice <b>index</b>
v.size()	ritorna il numero di elementi contenuti
v.clear()	rimuove tutti gli elementi

#### 4.5.3 Scanner

La classe scanner ci permette di leggere input utente da terminale (e anche da file, ma non ci servirà). Uno scanner si inizializza così:

```
Scanner nomeScanner = new Scanner(System.in);
```

Per leggere l'input da terminale ci sono i seguenti comandi:

Metodo	Descrizione
nextBoolean( )	Legge un boolean da terminale
nextByte()	Legge un byte da terminale
nextDouble()	Legge un double da terminale
nextFloat ()	Legge un float da terminale
nextInt()	Legge un int da terminale
nextLine()	Legge una String da terminale
nextLong( )	Legge un long da terminale
nextShort()	Legge uno short da terminale

Il metodo più comune è **nextLine()**, dato che ci restituisce l'intera riga come stringa, anche se contiene numeri

#### 4.5.4 Random

Random fornisce un modo comodo per generare numeri casuali. Inizializza con

```
Random nome = new Random()
```

Metodo	Descrizione
nextInt( range )	genera un numero casuale nel range [0, range)
nextFloat()	genera un float in range [0.0, 1.0]
nextDouble()	genera un double in range [0.0, 1.0]

#### 4.5.5 LinkedList

Lista linkata. Utilizzabile in modo molto simile al `ArrayList`. L'accesso agli elementi è molto più lento, rimozioni inserimenti sono molto più veloci

#### 4.5.6 Altre strutture dati utili

- `HashSet`: insieme matematico, possibile vedere se un elemento è contenuto in esso in maniera efficiente
- `Map`: utile poter collegare ad ogni valore un altro valore detto chiave. Si può risalire al valore tramite la chiave in maniera efficiente
- `Stack`
- `Queue`
- `PriorityQueue`: struttura nella quale è possibile accedere all'elemento maggiore in maniera efficiente
- `SortedSet`: struttura nella quale i dati mantengono sempre un ordinamento crescente. Non ammette duplicati

### 4.6 Lettura/scrittura su file

### 4.7 Try catch ed eccezioni

In java e in molti linguaggi di programmazione moderni vengono forniti meccanismi standard per gestire situazioni inaspettate che non possono però essere previste con certezza (es. un file non esiste, l'utente ha inserito un dato non valido etc etc). Per queste situazioni esistono oggetti dette *eccezioni*, fatte per descrivere l'errore che si è verificato.

L'idea generale è che qualunque pezzo di codice può sollevare o tirare un'eccezione e, nel caso in cui questa non venga gestita (ossia circondata da un blocco try catch nella porzione di programma chiamante), allora l'applicazione termina, stampando una descrizione della eccezione avvenuta

Il codice che tira eccezioni va circondato da un blocco try catch come segue:

```
try {
    // codice che potrebbe sollevare un'eccezione
} catch (TipoEccezione1 e1) {
    // codice per gestire l'eccezione di tipo TipoEccezione1
} catch (TipoEccezione2 e2) {
    // codice per gestire l'eccezione di tipo TipoEccezione2
} finally {
    // codice che viene eseguito sempre, sia che si sia verificata
    // un'eccezione sia che non si sia verificata
}
```

Ad esempio, se avessimo

```

try {
    f1() // solleva eccezione TipoEccezione2
} catch (TipoEccezione1 e1) {
    System.out.println("e1");
} catch (TipoEccezione2 e2) {
    System.out.println("e2");
} finally {
    System.out.println("finally");
}
// stampa:
// e2
// finally

```

Se invece facessimo qualcosa di questo tipo all'interno del main:

```
f1() // solleva eccezione TipoEccezione2
```

allora il programma si interromperebbe stampando informazioni riguardo l'eccezione sollevata

#### 4.7.1 Checked vs unchecked

Esistono due tipi di eccezioni: *checked* e *unchecked*. Dal punto di vista logico le prime rappresentano tutte le condizioni che non NON abbiamo modo di verificare non accadano, mentre nel secondo, utilizzando determinate accortezze possiamo evitare succedano. Per questa ragione, siamo *obbligati* a gestire le prime, mentre possiamo ignorare le seconde.

Checked exceptions	Unchecked exceptions
IOException	ArithmaticException
ClassNotFoundException	NullPointerException
FileNotFoundException	ArrayIndexOutOfBoundsException
SQLException	IllegalArgumentException

L'idea è che le unchecked exceptions possono essere ignorate proprio in virtù del fatto che tramite determinate accortezze possiamo evitare che si verifichino (es. controllare che un indice sia valido prima di accedere ad un array, controllare che un divisore non sia 0 prima di effettuare una divisione etc etc), mentre ciò non è vero per quanto riguarda le checked

Ciò che cambia nel pratico è che quando utilizziamo una porzione di codice che può sollevare una checked exception, il compilatore ci obbliga a circondarla con un blocco try catch, oppure a dichiarare che la funzione che la contiene può sollevare quell'eccezione (usando la parola chiave **throws**):

```
public f1() throws CheckedException{}
```

Si noti che in realtà è possibile dichiarare una funzione con la keyword throw anche per le unchecked exceptions, ma non è obbligatorio

#### 4.8 Scanner

Scanner fornisce il modo più semplice (ma più lento) per leggere da file o da **stdin**:

```

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

try (Scanner scanner = new Scanner(new File("path/to/file.txt"))) {
    while (scanner.hasNextLine()) {
        String line = scanner.nextLine();
        // Process the line
    }
} catch (FileNotFoundException e) {
    e.printStackTrace();
}

```

per la scrittura invece il metodo preferito è il seguente:

```

import java.io.PrintWriter;
import java.io.FileWriter;
import java.io.IOException;

try (PrintWriter writer = new PrintWriter(new FileWriter("esempio.txt",
    true))) {
    writer.println("Riga aggiunta!");
} catch (IOException e) {
    e.printStackTrace();
}

```

## 4.9 File

Il package nio.file un modo più rapido e moderno per leggere da file:

```

package com.mycompany.app;

import java.nio.file.Files;
import java.nio.file.Paths;
import java.io.IOException;

String content=new
    String(Files.readAllBytes(Paths.get("path/to/file.txt")));
List<String>lines=Files.readAllLines(Paths.get("path/to/file.txt"));

```

per la scrittura invece il metodo preferito è il seguente:

```

import java.nio.file.Files;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;
import java.io.IOException;

public class AppendFileNio {
    public static void main(String[] args) {
        String nuovaRiga = "Nuova riga aggiunta!";
        try {
            Files.write(

```

```

        Paths.get("esempio.txt"),
        Arrays.asList(nuovaRiga),
        StandardOpenOption.APPEND
    );
} catch (IOException e) {
    e.printStackTrace();
}
}
}
}

```

## 4.10 Esercizi

Collezione di esercizi, divisi per categoria

### Esercizio 18: Hello World (*hello\_world*)

Scrivere un programma in Java che prenda in input una stringa **s** e un numero **n** e stampi **s n** volte, accompagnata dal numero di riga. Ad esempio, dati in input **userin** e **4**, l'output sarà:

1. userin
2. userin
3. userin
4. userin

### Esercizio 19: Hello classe (estrazione)

Crea un programma per giocare a una sorta di roulette modificata. Le regole sono le seguenti:

- Vengono estratte delle palline che hanno un colore (**rosso**, **giallo** o **verde**) e un numero da 1 a 9 estremi compresi
- Il giocatore deve indicare un numero e un colore
- Il punteggio viene assegnato così:
  - Colore giusto: **+1**
  - Un punteggio che varia da 0 a 4 in base alla distanza del numero previsto da quello estratto, ad esempio, detto  $n_e$  il numero *estratto* e  $n_p$  il numero *previsto*
$$\left(1 - \frac{|n_e - n_p|}{8}\right) \cdot 4$$
  - Se colore e numero sono uguali: **+1**

Il programma deve chiedere in input all'utente un numero e un colore ed estrarre una pallina casualmente, per poi stampare in output il punteggio ottenuto. Si usi una classe **ball** e una classe **game** per gestire le partite.

Il programma deve partire chiedendo all'utente il numero  $n$  di round che vuole giocare. Dopodiché, verranno chieste  $n$  previsione all'utente; per ognuna deve essere stampato il punteggio e alla fine degli  $n$  round deve essere visualizzato il punteggio totale

### Esercizio 20: Geometria 1 (geometry1)

Creare una classe per ciascuno dei seguenti oggetti geometrici: **Punto**, **Rettangolo**, **Cerchio**. Creare una classe **PianoCartesiano** che possa contenere le tre classi elencate precedentemente. Una volta inserita una classe nel piano cartesiano verrà ritornato un **id**, tramite il quale possiamo accedervi (l'istanza dell'oggetto aggiunto rimane scollegata da quella inserita nel piano cartesiano). Creare un metodo **closer** che prenda un **id** esistente e ritorni l'id della forma geometrica più vicina (usare il centro delle figure per confrontare la distanza)

### Esercizio 21: Geometria 1 (geometry1)

Creare una classe per ciascuno dei seguenti oggetti geometrici: **Punto**, **Rettangolo**, **Cerchio**. Creare una classe **PianoCartesiano** che possa contenere le tre classi elencate precedentemente. Una volta inserita una classe nel piano cartesiano verrà ritornato un **id**, tramite il quale possiamo accedervi (l'istanza dell'oggetto aggiunto rimane scollegata da quella inserita nel piano cartesiano). Creare un metodo **closer** che prenda un **id** esistente e ritorni l'id della forma geometrica più vicina (usare il centro delle figure per confrontare la distanza)

### Esercizio 22: Montecarlo (*montecarlo*)

Il casinò di Montecarlo ha richiesto una nuova slot machine; lo scopo del gioco è quello di simulare la pesca di *cinque carte* da una mazzo standard (con quattro semi, le carte da 2 a 10, l'asso(1) le tre figure (11,12,13) e niente jolly)

Le opportunità di vincita (con quattro delle 5 carte estratte) sono le seguenti:

Combinazione	vincita
Poker di figure	1000 €
Poker d'assi	2000 €
<sup>a</sup> Colore	Somma delle carte uscite × 8
<sup>b</sup> Scala colore	Somma delle carte uscite × 10

La presenza di un'altra carta non intacca la vincita.

Contare quante slotmachine sono presenti nel casino

Si scrivano le classi opportune per implementare il gioco e un classe **Casino** dove creare diverse **Slot**.

Fare in modo che la giocata possa essere effettuata solo dopo aver inserito un certo importo;

Introdurre successivamente anche i *jolly* tra le carte e fare in modo che la presenza del jolly come carta estratta raddoppi la vincita;

<sup>a</sup>5 carte dello stesso seme, ad es: 1-4-5-10-13, *tutti cuori* → vincita = 264 €

<sup>b</sup>5 carte dello stesso seme in sequenza, ad esempio: 3-4-5-6-7 *tutte picche* → vincita = 250 €

### Esercizio 23: Sistema gestione prenotazioni centro sportivo

Il Centro Sportivo *ActiveWorld* offre diverse attività (*nuoto, palestra, yoga, arrampicata*) e vuole un software che gestisca le prenotazioni giornaliere per i vari corsi e spazi. Tutte le informazioni devono essere salvate e gestite tramite file di testo, letti all'avvio dell'applicazione e aggiornati durante l'esecuzione.

File coinvolti: l'applicazione utilizza tre file:

- *Utenti registrati*: ogni utente è caratterizzato da:

- nome
- cognome
- data di nascita
- telefono
- email

- *Attività disponibili*

- nome dell'attività

- capacità massima
- fascia oraria (es. 18:00--19:30)
- *Prenotazioni attive:* ogni riga del file contiene:
  - email dell'utente
  - nome attività
  - data della prenotazione (gg/mm/aaaa)

### Funzionalità richieste

1. *Caricamento dati dai 3 file:* all'avvio, il programma deve:
  - leggere tutti gli utenti registrati
  - leggere tutte le attività disponibili
  - leggere le prenotazioni attive
  - verificare che ogni prenotazione faccia riferimento a un utente e a un'attività validi
  - segnalare eventuali incongruenze (es. utente inesistente → prenotazione ignorata)
2. *Visualizzare le attività disponibili:* mostrare
  - nome attività
  - fascia oraria
  - numero prenotazioni attuali
  - posti disponibili rimanenti
3. *Visualizzare tutte le prenotazioni:* raggruppate per attività, ordinate per data.
4. *Cercare le prenotazioni di un utente:* richiedere *email* e mostrare tutte le sue prenotazioni.
5. *Aggiungere una nuova prenotazione:* il programma deve:
  - chiedere: email utente, attività, data
  - verificare che l'utente esista
  - verificare che l'attività esista
  - controllare che la data non sia nel passato
  - controllare che l'utente non abbia già una prenotazione per la stessa attività nella stessa giornata
  - controllare che l'attività non sia al completo per quella data
  - se tutto ok:
    - aggiungere la prenotazione in memoria

- aggiungerla al file in modalità *append*
6. *Cancellare una prenotazione*  
Richiedere:
- email utente
  - nome attività
  - data
- Controllare che esista, poi:
- rimuoverla dall'elenco
  - aggiornare il file riscrivendolo completamente
7. *Report automatico delle attività quasi piene*: il programma deve offrire una funzione *Genera Report* che:
- mostra tutte le attività che hanno meno del 10% dei posti liberi nella data selezionata
  - genera anche una versione salvata in un file `report.txt` con formattazione ordinata
8. *Statistiche settimanali (opzionale)*: dato un range di date, il programma deve indicare:
- quale attività è stata prenotata di più
  - quale di meno
  - il numero totale di prenotazioni
  - percentuale di riempimento media delle attività

#### Requisiti e vincoli tecnici

- Il formato dei file deve rimanere sempre uniforme.
- Se vengono inseriti dati mancanti o non validi, il programma deve richiederli di nuovo.
- Ogni modifica deve aggiornare coerentemente i file.
- L'applicazione deve funzionare finché il gestore non sceglie di uscire.

### Esercizio 24: ASCII art

Creare una applicazione che, dato in input un file *in formato .ppm p3* crei una ascii art di quanto rappresentato dalla figura. Il main deve definire due parametri:

- `inputFile` il file `.ppm` da convertire in ascii art
- `outputFile` il file `.txt` di output contenente la ascii art

Il formato `.ppm p3` rappresenta una immagine tramite una matrice di pixel, ognuno dei quali è rappresentato da tre numeri interi (rosso, verde, blu) che vanno da 0 a 255. Il file inizia con una intestazione composta da:

```
P3
<WIDTH> <HEIGHT>
<MAX_COLOR>
<R> <G> <B> ... <R> <G> <B>
```

Ad esempio, un file contenente 3 pixel in orizzontale, in ordine, uno rosso, uno verde e uno blu, sarebbe così codificato:

```
P3
3 1
255
255 0 0 0 255 0 0 0 255
```

dove

255 0 0 | 0 255 0 | 0 0 255  
pixel 1    pixel 2    pixel 3

hint: si può usare la seguente formula per convertire un pixel in bianco e nero:

```
double grayscale = 0.299 * r + 0.587 * g + 0.114 * b;
```