Esercitazione 2 - 05/10

Argomenti:

- tipo `boolean` e operatori di confronto
- condizioni: `if`, `else if`, `else`
- operatori logici `&&`, `||`, `!`
- `&&` e `||` con valori non booleani (truthy e falsy)
- ciclo `for`: somma e prodotto dei primi n numeri
- ciclo `while`: come alternativa al `for` e per condizioni diverse dal contare

Indice degli esercizi

Esercizio	Argomento
Esercizio 1	Operatori logici
Esercizio 2	Operatori di confronto
Esercizio 3	Operatori logici
Esercizio 4	Ciclo for
Esercizio 5	Ciclo for
Esercizio 6	Accumulatori
Esercizio 7	Ciclo for
Esercizio 8	Accumulatori
Esercizio 9	Accumulatori
Esercizio 10	Ciclo for
Esercizio 11	Accumulatori
Esercizio 12	Ciclo for
Esercizio 13	Ciclo while
Esercizio 14	Ciclo while
Esercizio 15	Ciclo while
Esercizio 16	Condizionali
Esercizio 17	Condizionali
Esercizio 18	Condizionali
Esercizio 19	Condizionali
Esercizio 20	Condizionali
Esercizio 21	Cicli e condizionali

Esercizio	Argomento
Esercizio 22	Cicli annidati
Esercizio 23	Ciclo for
Esercizio 24	Cicli annidati
Esercizio 25	Ciclo while

Recap 1 · `bool` e operatori di confronto

Con i tipi `number` e `string` possiamo descrivere la maggior parte degli oggetti che ci interessano.

Esempio	
Chilometri percorsi nello scorso anno	`number` - numero intero
Media dei voti	`number` - decimale
Nome e cognome	`string`
Anno di nascita	`number` - numero intero

Nel mondo reale però esiste anche un altro tipo di valore che vogliamo poter rappresentare in un linguaggio di programmazione. Ad esempio noi sappiamo valutare le seguenti espressioni:

- "il numero 3 è minore del numero 5"
- "Andrea è più alto di Marta"

Questi valori (vero o falso) sono rappresentati nei linguaggi di programmazione dal tipo `boolean`

Possiamo ottenerli effettuando operazioni **di confronto** su altri valori:

Operatore	Significato
`<` `>`	minore, maggiore
`<=` `>=`	minore o uguale, ecc.
`==` `!=`	uguale, diverso

- l'espressione `3 < 5` vale `true`

- l'espressione `"casa" == "gatto"` vale `false`

Recap 2 · Confronti e operatori logici

Operatore	Significato	Esempio (`x = 5`)
`==` `!=`	uguale, diverso	`x == 5` → `true`
`<` `<=` `>` `>=`	confronti	`x >= 6` → `false`
`&&`	**e**: entrambe vere	`x > 0 && x < 10` → `true`
`||`	**o**: almeno una vera	`x < 0 || x > 3` → `true`
`!`	**non**: nega	`!(x > 3)` → `false`

Pattern utili:

```
n % 2 == 0                  // n è pari
n % k == 0                  // n è multiplo di k
n >= 0 && n <= 10           // n compreso tra 0 e 10 (estremi inclusi)
```

Attenzione: `=` **assegna**, `==` **confronta**.

`0 <= n <= 10` non funziona: si scrive `n >= 0 && n <= 10`.

Recap 3 · Strumenti utili

Strumento	Cosa fa	Esempio
`Math.max(a, b)` / `Math.min(a, b)`	massimo / minimo	`Math.max(3, 7)` → `7`
`Math.floor(x)`	parte intera (arrotonda per difetto)	`Math.floor(2.9)` → `2`
`Math.random()`	decimale casuale in [0, 1)	`Math.floor(Math.random() * 6) + 1` → da 1 a 6
`s += x`	abbreviazione di `s = s + x`	`s -= x`, `s *= x`, ...
`x++` / `x--`	aumenta / diminuisce il valore della variabile x di 1	

Esercizio 1 · Affermazioni su *a* e *b*

Leggete due numeri `a` e `b` dall'input e stampate, **ognuna su una riga**, il valore booleano di queste affermazioni:

```
a è maggiore di b: ...
a è uguale a b: ...
a è diverso da b: ...
a è pari: ...
b è multiplo di a: ...
```

Esempio con `a = 6` e `b = 12`:

```
a è maggiore di b: false
a è uguale a b: false
a è diverso da b: true
a è pari: true
b è multiplo di a: true
```

▼ **Soluzione**

```
let a_input = prompt("a:");
let a = Number(a_input);
let b_input = prompt("b:");
let b = Number(b_input);
console.log("a è maggiore di b:", a > b);
console.log("a è uguale a b:", a == b);
console.log("a è diverso da b:", a != b);
console.log("a è pari:", a % 2 == 0);              // resto 0 -> pari
console.log("b è multiplo di a:", b % a == 0);      // b diviso a senza resto
```

Pari e multiplo si esprimono con `%` e un confronto. Se `a` vale 0, `b % a` dà `NaN`: provate!

Esercizio 2 · Confronti

Scrivere un programma che legga due numeri `N` e `M` e visualizzi, ognuno su una riga:

- il massimo tra `N` e `M`;
- il minimo tra `N` e `M`;
- se `N` è minore o uguale di `M`;
- se `N` è compreso tra 0 ed `M` (estremi inclusi);
- se `N` è divisibile per `M`.

Quali risultati sono numeri? Quali sono booleani?

▼ Soluzione

```
let N_input = prompt("N:");
let N = Number(N_input);
let M_input = prompt("M:");
let M = Number(M_input);
console.log("massimo:", Math.max(N, M));
console.log("minimo:", Math.min(N, M));
console.log("N <= M:", N <= M);
console.log("N tra 0 e M:", N >= 0 && N <= M);   // due confronti uniti da &&
console.log("N divisibile per M:", N % M == 0);
```

I primi due risultati sono `number`, gli altri tre `boolean`. Provate `M = 0`: `N % 0` dà `NaN`, e `NaN == 0` è `false`.

Esercizio 3 · Lancio di due dadi

Simulate il lancio di **due dadi** a 6 facce: stampate i due valori, la loro somma e se è uscito un **doppio** (valore booleano).

Esempio:

```
Dado 1: 4
Dado 2: 4
Somma: 8
Doppio: true
```

▼ **Soluzione**

```
let dado1 = Math.floor(Math.random() * 6) + 1;    // intero tra 1 e 6
let dado2 = Math.floor(Math.random() * 6) + 1;
console.log("Dado 1:", dado1);
console.log("Dado 2:", dado2);
console.log("Somma:", dado1 + dado2);
console.log("Doppio:", dado1 == dado2);
```

Ogni dado va estratto separatamente: riusare lo stesso valore darebbe sempre un doppio.

Recap 4 · I cicli `for` e `while`

Entrambi **ripetono** il corpo `{ ... }` finché la condizione è vera. Questi due programmi stampano `Ciao!` **5 volte**:

```
let i;                                  // for: tutto nell'intestazione
for (i = 1; i <= 5; i = i + 1) {        // inizializzazione; condizione; aggiornamento
  console.log("Ciao!");
}
```

```
let i = 1;                              // while: i tre pezzi sono "sparsi"
while (i <= 5) {                        // condizione
  console.log("Ciao!");
  i = i + 1;                            // aggiornamento: se manca, ciclo infinito!
}
```

- `for` quando **so quante volte** ripetere; `while` quando dipende da cosa succede (es. input dell'utente);
- abbreviazione: `i++` significa `i = i + 1`, e nel `for` si può scrivere `for (let i = 1; ...)`.

Esercizio 4 · Completa la guardia (1)

Il corpo del ciclo c'è già. Completate la **condizione** (al posto di `______`) in modo che il programma stampi `Ciao!` **5 volte**.

```
let i;
for (i = 0; ______; i = i + 1) {
  console.log("Ciao!");
}
```

▼ **Soluzione**

```
let i;
for (i = 0; i < 5; i = i + 1) {
  console.log("Ciao!");
}
```

Partendo da 0, `i` vale 0, 1, 2, 3, 4: cinque giri. Con `i <= 5` i giri sarebbero 6! Regola pratica: partendo da 0 con `<` il numero di giri è il valore a destra.

Alternative. Con un `while` i tre pezzi del `for` si separano; e partendo da `1` la condizione deve diventare `<=`:

```
// con while
let i = 0;
while (i < 5) {
  console.log("Ciao!");
  i = i + 1;
}

// con for, partendo da 1
for (i = 1; i <= 5; i = i + 1) {
  console.log("Ciao!");
}
```

Sono tutte versioni da 5 giri: cambiano il **punto di partenza** e il **confronto** (`< 5` da 0, `<= 5` da 1).

Esercizio 5 · Completa l'input e la guardia

Il numero di ripetizioni lo sceglie l'utente. Completate il programma (al posto di `_______`) in modo che chieda `n` all'utente, lo converta in numero e stampi `Ciao!` **n volte**.

```
let n_input = ______("Quante volte?");     // chiedo il valore all'utente
let n = ______(n_input);                   // lo converto in numero
let i;
for (i = 1; ______; i = i + 1) {
  console.log("Ciao!");
}
```

▼ **Soluzione**

```
let n_input = prompt("Quante volte?");     // chiedo il valore all'utente: è una stringa
let n = Number(n_input);                   // lo converto in numero
let i;
for (i = 1; i <= n; i = i + 1) {
  console.log("Ciao!");
}
```

`prompt` restituisce sempre una stringa: senza `Number`, la condizione `i <= n` confronterebbe un numero con un testo. La condizione può contenere una **variabile**, non solo un numero fisso. Provate `n = 0`: il corpo non viene mai eseguito.

Alternativa con `while`:

```
let n_input = prompt("Quante volte?");
let n = Number(n_input);
let i = 1;
while (i <= n) {
  console.log("Ciao!");
  i = i + 1;
}
```

Esercizio 6 · Potenze di 2

Scrivere un programma che visualizzi le potenze di 2 partendo dall'esponente 0 fino all'esponente 8, una per riga.

▼ **Soluzione**

```
for (let i = 0; i <= 8; i++) {        // i è l'esponente
  console.log(2 ** i);
}
```

Stampa `1, 2, 4, 8, ..., 256`. Notate che il ciclo parte da `0`, non da `1`.

Alternative. Con un `while`, oppure facendo variare **direttamente la potenza** (che raddoppia a ogni giro) invece dell'esponente:

```
// con while
let i = 0;
while (i <= 8) {
  console.log(2 ** i);
  i++;
}

// con for: l'aggiornamento è p = p * 2, non p++
for (let p = 1; p <= 256; p = p * 2) {
  console.log(p);
}
```

Nell'ultima versione non serve `**`: il `for` non conta più gli esponenti, ma moltiplica per 2 a ogni giro (e `256` è `2 ** 8`).

Esercizio 7 · Completa la guardia (2)

Il programma deve chiedere all'utente quanti numeri vuole sommare, poi chiedere i numeri uno alla volta e stampare la loro **somma**. Il corpo del ciclo c'è già: completate la **condizione** (al posto di `______`).

```
let n_input = prompt("Quanti numeri vuoi sommare?");
let n = Number(n_input);
let somma = 0;                                  // accumulatore
let i;
for (i = 1; ______; i = i + 1) {
  let x_input = prompt("Numero " + i + ":");
  let x = Number(x_input);
  somma = somma + x;
}
console.log("Somma:", somma);
```

▼ **Soluzione**

```
let n_input = prompt("Quanti numeri vuoi sommare?");
let n = Number(n_input);
let somma = 0;                                    // accumulatore
let i;
for (i = 1; i <= n; i = i + 1) {
  let x_input = prompt("Numero " + i + ":");
  let x = Number(x_input);
  somma = somma + x;
}
console.log("Somma:", somma);
```

Con `n = 3` e numeri `4`, `5`, `6` la somma è 15. Con `i < n` l'utente verrebbe interrogato una volta di meno: è l'errore più comune. Qui `i` serve solo a contare i giri (e a numerare le domande), i numeri da sommare sono quelli inseriti.

Alternativa con `while`:

```
let n_input = prompt("Quanti numeri vuoi sommare?");
let n = Number(n_input);
let somma = 0;
let i = 1;
while (i <= n) {
  let x_input = prompt("Numero " + i + ":");
  let x = Number(x_input);
  somma = somma + x;
  i = i + 1;
}
console.log("Somma:", somma);
```

Esercizio 8 · Somma e prodotto dei primi *n* numeri pari

Leggere un numero naturale `n` e calcolare la **somma** e il **prodotto** dei primi `n` numeri pari positivi. Stampare i due risultati alla fine.

Esempio: con `n = 4` i numeri sono `2, 4, 6, 8`, quindi la somma è `20` e il prodotto è `384`.

▼ **Soluzione**

```
let n_input = prompt("n:");
let n = Number(n_input);
let somma = 0;                        // accumulatore della somma: parte da 0
let prodotto = 1;                     // accumulatore del prodotto: parte da 1 (non da 0!)
for (let i = 1; i <= n; i++) {
  let pari = 2 * i;
  somma = somma + pari;
  prodotto = prodotto * pari;
}
console.log("Somma:", somma);
console.log("Prodotto:", prodotto);
```

Due accumulatori nello stesso ciclo: ognuno parte dall'elemento neutro della sua operazione (`0` per la somma, `1` per il prodotto). Se partisse da `0`, il prodotto resterebbe sempre `0`.

Alternativa: far partire `pari` da `2` e incrementarlo di `2` a ogni giro (`pari = pari + 2`).

```
// il for conta direttamente i numeri pari: da 2 a 2 * n, di 2 in 2
for (let pari = 2; pari <= 2 * n; pari = pari + 2) {
  somma = somma + pari;
  prodotto = prodotto * pari;
}

// con while
let pari = 2;
while (pari <= 2 * n) {
  somma = somma + pari;
  prodotto = prodotto * pari;
  pari = pari + 2;
}
```

(Le altre righe del programma, compresa la lettura di `n`, restano uguali.) In questa versione `i` non esiste più: si conta con `pari`, e il ciclo si ferma quando `pari` supera l'ultimo pari che serve, cioè `2 * n`.

Cosa succede con `n = 0`? Il ciclo non viene mai eseguito: somma `0` e prodotto `1`.

Esercizio 9 · Media di *n* numeri

Leggere un numero `n` e poi `n` numeri, uno alla volta. Stampare la loro **media**.

▼ **Soluzione**

```
let n_input = prompt("Quanti numeri vuoi inserire?");
let n = Number(n_input);
let somma = 0;
for (let i = 1; i <= n; i++) {
  let x_input = prompt("Numero " + i + ":");
  let x = Number(x_input);
  somma = somma + x;
}
console.log("Media:", somma / n);
```

Il numero di ripetizioni è noto (`n`), quindi è un caso da `for` . Cosa succede con `n = 0` ? `0 / 0` dà `NaN` : come lo gestireste con un `if` ?

Alternativa con `while` :

```
let n_input = prompt("Quanti numeri vuoi inserire?");
let n = Number(n_input);
let somma = 0;
let i = 1;
while (i <= n) {
  let x_input = prompt("Numero " + i + ":");
  let x = Number(x_input);
  somma = somma + x;
  i++;
}
console.log("Media:", somma / n);
```

Esercizio 10 · Una riga di asterischi

Leggere un numero `n` e stampare **una sola riga** composta da `n` asterischi. Con `n = 5` :

```
*****
```

Il ciclo e la stampa ci sono già: completate il corpo (al posto di `______`) in modo da **aggiungere** un asterisco alla stringa `riga` a ogni giro.

```
let n_input = prompt("n:");
let n = Number(n_input);
let riga = "";                        // stringa vuota: l'accumulatore
let i;
for (i = 1; i <= n; i = i + 1) {
  riga = ______;
}
console.log(riga);
```

▼ **Soluzione**

```
let n_input = prompt("n:");
let n = Number(n_input);
let riga = "";                          // stringa vuota: l'accumulatore
let i;
for (i = 1; i <= n; i = i + 1) {
  riga = riga + "*";                    // riga di prima + un asterisco
}
console.log(riga);
```

Come per la somma, ma con le stringhe: `riga` parte da `""` e a ogni giro diventa `""`, poi `"*"`, `"**"`, `"***"`... Si stampa **una volta sola, dopo il ciclo**: un `console.log` dentro il ciclo stamperebbe `n` righe.

Alternativa con `while`:

```
let riga = "";
let i = 1;
while (i <= n) {
  riga = riga + "*";
  i = i + 1;
}
console.log(riga);
```

Esercizio 11 · I numeri da 1 a *n* su una riga

Leggere un numero `n` e stampare **su una sola riga** i numeri da 1 a `n`, separati da uno spazio. Con `n = 5`:

```
1 2 3 4 5
```

Suggerimento: partite dall'esercizio precedente!

▼ **Soluzione**

```
let n_input = prompt("n:");
let n = Number(n_input);
let riga = "";
let i;
for (i = 1; i <= n; i = i + 1) {
  riga = riga + i + " ";              // aggiungo il numero e uno spazio
}
console.log(riga);
```

Passo per passo, con `n = 3`:

1. `riga` vale `""`;
2. `"" + 1 + " "` → `"1 "`: il numero `1` viene trasformato in testo, perché a sinistra c'è una stringa;
3. `"1 " + 2 + " "` → `"1 2 "`;
4. `"1 2 " + 3 + " "` → `"1 2 3 "`.

`i` resta un numero, ma `riga` è una stringa: il `+` concatena. Alla fine c'è uno spazio in più, che a schermo non si vede.

Alternativa con `while`:

```
let riga = "";
let i = 1;
while (i <= n) {
  riga = riga + i + " ";
  i = i + 1;
}
console.log(riga);
```

Esercizio 12 · Conto alla rovescia

Completate (al posto di `______`) in modo che il programma stampi `5`, `4`, `3`, `2`, `1` e infine `Via!`.

```
let i = 5;                            // inizializzazione
while (______) {                      // condizione
  console.log(i);
  i = ______;                         // aggiornamento
}
console.log("Via!");
```

▼ **Soluzione**

```
let i = 5;                              // inizializzazione
while (i >= 1) {                        // condizione
  console.log(i);
  i = i - 1;                            // aggiornamento: si scende
}
console.log("Via!");
```

Se dimenticate l'aggiornamento, `i` resta sempre `5` e il ciclo non finisce mai. Con `i > 1` l'`1` non verrebbe stampato.

Alternativa con `for`. Qui il numero di giri è noto (5), quindi il `for` è adatto, con l'aggiornamento `i--` che fa scendere:

```
for (let i = 5; i >= 1; i--) {
  console.log(i);
}
console.log("Via!");
```

Esercizio 13 · Completa l'input valido

Il programma chiede un numero **positivo** e ripete la domanda finché il valore non è valido. Completate la **condizione** e la riga che **rilegge** il valore (al posto di `_____`).

```
let x_input = prompt("Inserisci un numero positivo:");
let x = Number(x_input);
while (_____) {                                    // finché NON è valido
  x_input = _____("Non valido. Riprova:");
  x = Number(x_input);
}
console.log("Hai inserito:", x);
```

▼ **Soluzione**

```
let x_input = prompt("Inserisci un numero positivo:");
let x = Number(x_input);
while (x <= 0) {                                    // finché NON è valido
  x_input = prompt("Non valido. Riprova:");
  x = Number(x_input);
}
console.log("Hai inserito:", x);
```

La condizione descrive il caso **sbagliato** (`x <= 0`), non quello giusto. Il valore si legge una volta prima del ciclo e di nuovo a ogni giro: senza la rilettura, `x` non cambierebbe mai.

Alternativa con `do ... while`. La domanda va fatta almeno una volta, quindi si legge **una sola volta**, dentro il ciclo (con un solo messaggio, `"Inserisci un numero positivo:"`):

```
let x;
do {
  let x_input = prompt("Inserisci un numero positivo:");
  x = Number(x_input);
} while (x <= 0);
console.log("Hai inserito:", x);
```

Esercizio 14 · Somma fino a zero

Leggere numeri dall'input e sommarli, finché l'utente non inserisce `0`. Alla fine stampare la somma e quanti numeri sono stati inseriti (lo `0` finale escluso).

▼ **Soluzione**

```
let somma = 0;
let quanti = 0;
let x_input = prompt("Numero (0 per terminare):");
let x = Number(x_input);
while (x != 0) {
  somma = somma + x;
  quanti++;
  x_input = prompt("Numero (0 per terminare):");
  x = Number(x_input);    // si rilegge a ogni giro
}
console.log("Somma:", somma);
console.log("Numeri inseriti:", quanti);
```

Se l'utente inserisce subito `0`, il corpo non viene mai eseguito e il risultato è `0` e `0`.

Alternativa con `do ... while`. Il corpo parte sempre, quindi va protetto con un `if`: lo `0` finale viene letto ma **non** va contato né sommato.

```
let somma = 0;
let quanti = 0;
let x;
do {
  let x_input = prompt("Numero (0 per terminare):");
  x = Number(x_input);
  if (x != 0) {
    somma = somma + x;
    quanti++;
  }
} while (x != 0);
console.log("Somma:", somma);
console.log("Numeri inseriti:", quanti);
```

Si legge in un solo punto, ma la condizione `x != 0` compare **due volte**: la versione con `while` è più lineare.

Esercizio 15 · Indovina il numero

Il programma sceglie un numero casuale tra 1 e 10. L'utente prova a indovinarlo, finché non ci riesce. Alla fine stampare in quanti tentativi ci è riuscito.

▼ **Soluzione**

```
let segreto = Math.floor(Math.random() * 10) + 1;          // intero tra 1 e 10
let tentativo_input = prompt("Indovina il numero (1–10):");
let tentativo = Number(tentativo_input);
let tentativi = 1;
while (tentativo != segreto) {
  tentativo_input = prompt("Sbagliato, riprova:");
  tentativo = Number(tentativo_input);
  tentativi++;
}
console.log("Indovinato in", tentativi, "tentativi");
```

Non sappiamo quanti tentativi serviranno: è un caso da `while`. Come bonus, aggiungete dentro il ciclo un `if` che dica "più alto" o "più basso".

Alternativa con `do ... while`:

```
let segreto = Math.floor(Math.random() * 10) + 1;
let tentativi = 0;
let tentativo;
do {
  let tentativo_input = prompt("Indovina il numero (1–10):");
  tentativo = Number(tentativo_input);
  tentativi++;
} while (tentativo != segreto);
console.log("Indovinato in", tentativi, "tentativi");
```

Qui il primo tentativo è sempre necessario, quindi il `do ... while` è naturale; con un solo messaggio per tutti i tentativi.

Recap 5 · `if` / `else if` / `else`

```
if (condizione1) {
  // eseguito se condizione1 è true
} else if (condizione2) {
  // eseguito se condizione1 è false e condizione2 è true
} else {
  // eseguito se nessuna delle precedenti è true
}
```

- la condizione è un'espressione che vale `true` o `false` (ad esempio `eta >= 18`);
- viene eseguito **un solo** blocco: il primo la cui condizione è vera;
- `else if` e `else` sono **facoltativi**;
- l'ordine conta: le condizioni vengono controllate dall'alto verso il basso.

```
let voto = 24;
if (voto >= 18) {
  console.log("Esame superato");
} else {
  console.log("Esame non superato");
}
```

Esercizio 16 · Pari o dispari

Leggere un numero intero e stampare `"pari"` oppure `"dispari"`.

▼ Soluzione

```
let n_input = prompt("Numero intero:");
let n = Number(n_input);
if (n % 2 == 0) {
  console.log("pari");
} else {
  console.log("dispari");
}
```

Con un numero negativo, `-3 % 2` vale `-1`: il confronto con `0` funziona comunque, perché `-3 % 2 == 0` è `false`.

Esercizio 17 · Radice quadrata

Dato un numero reale in input, mostrare il valore della sua radice quadrata.

Fare inizialmente il controllo che il numero sia **non negativo**, dato che la radice quadrata è definita solo per argomenti reali maggiori o uguali a zero. Altrimenti stampare un messaggio di errore.

▼ **Soluzione**

```
let x_input = prompt("Numero reale:");
let x = Number(x_input);
if (x >= 0) {
  console.log("Radice quadrata:", Math.sqrt(x));
} else {
  console.log("Errore: il numero non può essere negativo");   // 0 invece è ammesso
}
```

`Math.sqrt` calcola la radice quadrata. Provate `x = 9`, `x = 2`, `x = 0` e `x = -4`: attenzione al `>=`, perché `0` è un argomento valido (con `>` verrebbe scartato per errore).

Esercizio 18 · Massimo di tre numeri

Leggere tre numeri e stampare il più grande, **senza** usare `Math.max`.

▼ **Soluzione**

```
let a_input = prompt("a:");
let a = Number(a_input);
let b_input = prompt("b:");
let b = Number(b_input);
let c_input = prompt("c:");
let c = Number(c_input);
if (a >= b && a >= c) {
  console.log("Il massimo è", a);
} else if (b >= c) {          // qui sappiamo già che a non è il massimo
  console.log("Il massimo è", b);
} else {
  console.log("Il massimo è", c);
}
```

Provate con numeri uguali (ad esempio `5, 5, 3`): il programma deve funzionare anche in questi casi.

Esercizio 19 · Voto e giudizio

Leggere un voto da 0 a 30 e stampare un giudizio:

Voto	Giudizio
minore di 18	insufficiente
da 18 a 23	sufficiente
da 24 a 27	buono
da 28 a 30	ottimo
fuori da 0–30	voto non valido

▼ **Soluzione**

```
let voto_input = prompt("Voto (0–30):");
let voto = Number(voto_input);
if (voto < 0 || voto > 30) {                // prima i casi non validi
  console.log("voto non valido");
} else if (voto < 18) {
  console.log("insufficiente");
} else if (voto < 24) {                     // qui sappiamo già che voto >= 18
  console.log("sufficiente");
} else if (voto < 28) {
  console.log("buono");
} else {
  console.log("ottimo");
}
```

Ogni `else if` "eredita" le condizioni false dei precedenti: per questo basta scrivere `voto < 24` e non `voto >= 18 && voto < 24`. L'ordine delle condizioni è parte della soluzione.

Esercizio 20 · Costo di una chiamata

Leggere la durata in minuti (numero reale) di una chiamata telefonica e visualizzare il costo, sapendo che:

- lo scatto alla risposta costa 0.18 €;
- il primo minuto è gratuito;
- ogni minuto successivo al primo costa 0.16 €.

Se la durata è negativa, stampare un messaggio di errore.

▼ Soluzione

```
let durata_input = prompt("Durata in minuti:");
let durata = Number(durata_input);
if (durata < 0) {
  console.log("Errore: durata negativa");
} else if (durata < 1) {
  console.log("Costo:", 0.18);                          // solo lo scatto
} else {
  console.log("Costo:", 0.18 + (durata - 1) * 0.16);    // scatto + minuti oltre il primo
}
```

Verificate con `0`, `0.5`, `1`, `10.3`, `-2`.

Esercizio 21 · Somma di multipli

Calcolare la somma di tutti i numeri da 1 a `n` che sono multipli di 3 **oppure** di 5.

Esempio: con `n = 10` si sommano `3, 5, 6, 9, 10` → `33`.

▼ **Soluzione**

```
let n_input = prompt("n:");
let n = Number(n_input);
let somma = 0;
for (let i = 1; i <= n; i++) {
  if (i % 3 == 0 || i % 5 == 0) {        // l'if sta dentro il corpo del for
    somma = somma + i;
  }
}
console.log("Somma:", somma);
```

Un numero come 15 è multiplo di entrambi, ma viene sommato **una sola volta**: l'`||` valuta la condizione, non conta i casi.

Alternative. Con un `while`; oppure **senza l'`if`**, facendo saltare il contatore di 3 in 3 e di 5 in 5 (e togliendo i multipli di 15, che sarebbero contati due volte):

```
// con while
let somma = 0;
let i = 1;
while (i <= n) {
  if (i % 3 == 0 || i % 5 == 0) {
    somma = somma + i;
  }
  i++;
}

// senza if: tre cicli, ognuno con il suo passo
let somma = 0;
for (let i = 3; i <= n; i = i + 3) {            // multipli di 3
  somma = somma + i;
}
for (let i = 5; i <= n; i = i + 5) {            // multipli di 5
  somma = somma + i;
}
for (let i = 15; i <= n; i = i + 15) {          // multipli di 15: contati due volte, si
tolgono una volta
  somma = somma - i;
}
```

Con `n = 10`: `3 + 6 + 9 = 18`, `5 + 10 = 15`, nessun multiplo di 15: `18 + 15 = 33`. La versione con `if` è più corta e più facile da cambiare; quella a tre cicli fa molti meno giri.

Esercizio 22 · Tabellina

Leggere un numero `n` e stampare la sua tabellina (da `n · 1` a `n · 10`) **in un'unica stringa**.

Se `n` non è un numero intero compreso tra 1 e 10, stampare un messaggio di errore.

Esempio con `n = 5`: `5 10 15 20 25 30 35 40 45 50`

Scrivere due versioni: una con `for`, una con `while`.

▼ **Soluzione con `for`**

```
// versione con for
let n_input = prompt("n (da 1 a 10):");
let n = Number(n_input);
if (Number.isInteger(n) && n >= 1 && n <= 10) {
  let riga = "";
  for (let i = 1; i <= 10; i++) {
    riga = riga + (n * i) + " ";
  }
  console.log(riga);
} else {
  console.log("Errore: n deve essere un intero tra 1 e 10");
}
```

▼ **Soluzione con `while`**

```
// versione con while
let n_input = prompt("n (da 1 a 10):");
let n = Number(n_input);
if (Number.isInteger(n) && n >= 1 && n <= 10) {
  let riga = "";
  let i = 1;
  while (i <= 10) {
    riga = riga + (n * i) + " ";
    i++;
  }
  console.log(riga);
} else {
  console.log("Errore: n deve essere un intero tra 1 e 10");
}
```

Le parentesi in `(n * i)` non sono obbligatorie, ma rendono chiaro che prima si moltiplica e poi si concatena.

Alternativa con `for` senza il prodotto. I numeri della tabellina sono `n, 2n, 3n, ...`: invece di moltiplicare `n * i`, si può far partire il contatore da `n` e **aumentarlo di `n`** a ogni giro (`m = m + n`):

```
let riga = "";
for (let m = n; m <= 10 * n; m = m + n) {
  riga = riga + m + " ";
}
console.log(riga);
```

Esercizio 23 · Polinomio

Leggere un numero reale `x` e un numero naturale `n` (con `n ≥ 1`) e calcolare

$x + x^2 + x^3 + ... + x^n$

▼ **Soluzione**

```
let x_input = prompt("x:");
let x = Number(x_input);
let n_input = prompt("n (>= 1):");
let n = Number(n_input);
if (n >= 1 && Number.isInteger(n)) {
  let risultato = 0;
  for (let i = 1; i <= n; i++) {
    risultato = risultato + x ** i;      // ** ha precedenza su +
  }
  console.log("Risultato:", risultato);
} else {
  console.log("Errore: n deve essere un intero >= 1");
}
```

Controllo: con `x = 2` e `n = 3` il risultato è `2 + 4 + 8 = 14`.

Alternative. Con un `while`; oppure senza `**`, tenendo in una variabile la **potenza corrente**, che a ogni giro si moltiplica per `x`:

```
// con while
let risultato = 0;
let i = 1;
while (i <= n) {
  risultato = risultato + x ** i;
  i++;
}

// senza **: potenza = x, x², x³, ...
let risultato = 0;
let potenza = 1;
for (let i = 1; i <= n; i++) {
  potenza = potenza * x;                        // x^i, calcolata dalla potenza del giro
prima
  risultato = risultato + potenza;
}
```

Con `x = 2` e `n = 3` la `potenza` vale `2`, `4`, `8` e il risultato è `14`, come prima.

Esercizio 24 · Coppie con somma *n*

Realizzare un programma che stampi tutte le coppie di numeri interi positivi (diversi tra loro) la cui somma è un numero naturale `n` a scelta, maggiore di 2.

Ad esempio, con `n = 3` le coppie da visualizzare sono:

```
1, 2
2, 1
```

Accertarsi inizialmente che `n` sia effettivamente un numero naturale maggiore di 2. Fare uso del ciclo `for`.

▼ **Soluzione**

```
let n_input = prompt("n (intero > 2):");
let n = Number(n_input);
while (n <= 2 || !Number.isInteger(n)) {            // si richiede finché non è valido
  n_input = prompt("Non valido. n (intero > 2):");
  n = Number(n_input);
}
for (let i = 1; i < n; i++) {
  let j = n - i;                                    // il secondo numero è determinato dal
primo
  if (i != j) {
    console.log(i + ", " + j);
  }
}
```

Non servono due cicli: fissato `i`, l'unico `j` possibile è `n - i`. Con `n = 4` la coppia `2, 2` viene scartata dall'`if`.

Alternative. Il problema chiede il `for`, ma lo stesso ciclo si scrive con un `while`; e il controllo dell'input si può scrivere con un `do ... while`:

```
// controllo dell'input con do ... while
let n;
do {
  let n_input = prompt("n (intero > 2):");
  n = Number(n_input);
} while (n <= 2 || !Number.isInteger(n));

// coppie con while
let i = 1;
while (i < n) {
  let j = n - i;
  if (i != j) {
    console.log(i + ", " + j);
  }
  i++;
}
```

Esercizio 25 · Primi *N* pari dopo *A*

Calcolare la somma e la media dei primi `N` numeri interi positivi **pari** successivi a un numero intero positivo `A`. Sia `N` che `A` devono essere letti da tastiera.

Accertarsi inizialmente che `N` ed `A` siano numeri interi positivi.

Esempio: con `A = 5` e `N = 3` i numeri sono `6, 8, 10`: somma `24`, media `8`.

▼ **Soluzione**

```
let N_input = prompt("N (intero positivo):");
let N = Number(N_input);
while (N < 1 || !Number.isInteger(N)) {
  N_input = prompt("Non valido. N (intero positivo):");
  N = Number(N_input);
}
let A_input = prompt("A (intero positivo):");
let A = Number(A_input);
while (A < 1 || !Number.isInteger(A)) {
  A_input = prompt("Non valido. A (intero positivo):");
  A = Number(A_input);
}

let somma = 0;
let trovati = 0;                    // quanti pari abbiamo già sommato
let numero = A;
while (trovati < N) {
  numero = numero + 1;              // passo al numero successivo
  if (numero % 2 == 0) {
    somma = somma + numero;
    trovati++;                      // conta solo quando troviamo un pari
  }
}
console.log("Somma:", somma);
console.log("Media:", somma / N);
```

Invece di bloccare il programma, qui si richiede il valore finché è valido. Il contatore `trovati` cresce solo dentro l'`if`: un `for` da 1 a `N` non basterebbe, perché non tutti i numeri visitati sono pari.

Alternativa con `for`. Se si parte direttamente dal **primo pari dopo `A`** e si salta di 2 in 2, tutti i numeri visitati sono pari, e il numero di giri è noto (`N`): l'`if` non serve più e si può usare un `for`.

```
let primo = A + 1;                  // il primo candidato
if (primo % 2 != 0) {
  primo = primo + 1;                // se è dispari, il primo pari è il successivo
}

let somma = 0;
let numero = primo;
for (let k = 1; k <= N; k++) {
  somma = somma + numero;
  numero = numero + 2;              // il prossimo pari
}
console.log("Somma:", somma);
console.log("Media:", somma / N);
```

Con `A = 5` e `N = 3`: `primo` vale `6`, e si sommano `6, 8, 10`. Quest'idea (saltare di 2) dà un ciclo più semplice, ma richiede di sapere **da dove cominciano** i pari: la versione con `while` e `if` è più generale.

Anche il controllo dell'input si può scrivere con un `do ... while`:

```
let N;
do {
  let N_input = prompt("N (intero positivo):");
  N = Number(N_input);
} while (N < 1 || !Number.isInteger(N));
```

Recap 6 · && e || con valori non booleani

In JavaScript `&&` e `||` non restituiscono sempre `true` o `false`: restituiscono **uno dei due operandi**. Per decidere, ogni valore viene letto come vero o falso:

- **falsi** (*falsy*): `false`, `0`, `""` (stringa vuota), `NaN`, `null`, `undefined`
- **veri** (*truthy*): tutto il resto, compresi `5`, `-1`, `"0"` e `"false"` (stringhe non vuote)

Le regole, con valutazione da sinistra a destra:

Operatore	Se il primo operando è…	Risultato
`a || b`	vero	`a` (b non viene nemmeno valutato)
`a || b`	falso	`b`
`a && b`	falso	`a` (b non viene nemmeno valutato)
`a && b`	vero	`b`

In pratica, `||` restituisce il **primo operando vero** (o l'ultimo, se sono tutti falsi), mentre `&&` restituisce il **primo operando falso** (o l'ultimo, se sono tutti veri).

Cosa stampano queste istruzioni?

```
console.log(5 || true);
console.log(5 || false);
console.log(0 || false);
console.log(0 || true);
console.log("0" || true);
console.log("0" || false);
console.log(5 && true);
console.log(5 && false);
console.log(0 && true);
console.log(0 && false);
console.log("0" && true);
```

▼ **Soluzione**

Istruzione	**Stampa**	**Perché**
`5 || true`	`5`	`5` è vero: `||` si ferma e restituisce il primo operando
`5 || false`	`5`	come sopra
`0 || false`	`false`	`0` è falso, si passa al secondo, che viene restituito
`0 || true`	`true`	`0` è falso, si restituisce il secondo
`"0" || true`	`"0"`	`"0"` è una stringa non vuota, quindi **vera**: si ferma lì
`"0" || false`	`"0"`	come sopra
`5 && true`	`true`	`5` è vero: `&&` passa al secondo e lo restituisce
`5 && false`	`false`	`5` è vero, si restituisce il secondo
`0 && true`	`0`	`0` è falso: `&&` si ferma e restituisce il primo operando
`0 && false`	`0`	come sopra (il secondo non viene valutato)
`"0" && true`	`true`	`"0"` è vera, si restituisce il secondo

Attenzione al caso `"0"`: il **numero** `0` è falso, ma la **stringa** `"0"` è vera, perché non è vuota. È la stessa differenza che c'è tra `Number("0")` e `"0"` quando leggiamo un valore con `prompt`.

Dentro un `if` o un `while` questa differenza si vede poco, perché la condizione viene comunque letta come vera o falsa. Si vede quando stampiamo o salviamo il risultato in una variabile.


Ludovica Pannitto - Università di Pisa - CC BY-NC-SA 4.0