Simulazione compitino - 30/10

Commento ai risultati della **simulazione** svolta su Moodle: la parte di **conoscenze di base**.

- 9 domande a risposta multipla, 10 minuti
- punteggio: +1 esatta, −1 errata, 0 in bianco
- si passa con 5/9
- risultati su **59** studenti che hanno consegnato

Come è andata

- punteggio medio: **4.5** su 9
- hanno raggiunto la sufficienza (5/9): **34** su 59 (**58%**)

Domanda	Argomento	Risposte corrette
1	Introduzione: compilazione	81%
2	Funzioni: valore restituito	54%
3	Cicli: somma condizionata	71%
4	Condizionali: catena if / else if	56%
5	Operatori aritmetici: resto	85%
6	Variabili e tipi: stringhe e numeri	39%
7	Operatori di confronto e logici	80%
8	Variabili e tipi: Number()	86%
9	Condizionali: if separati	88%

Per ogni domanda: **soluzione**, **spiegazione** ed **errori più comuni**.

Domanda 1

Introduzione: compilazione. Per un linguaggio compilato, quale sequenza descrive correttamente il percorso dal programma al risultato?

Opzione	Risposte
`codice sorgente → compilatore → eseguibile → esecuzione`	**81%**
`codice sorgente → esecuzione riga per riga → eseguibile`	2%
`eseguibile → compilatore → codice sorgente → esecuzione`	2%
Nessuna delle risposte è corretta	2%
in bianco	14%

Soluzione: `codice sorgente → compilatore → eseguibile → esecuzione`

In un linguaggio compilato il compilatore traduce **prima tutto** il sorgente in un file eseguibile, e solo dopo si esegue quel file.

- «riga per riga» descrive un linguaggio **interpretato**
- la terza opzione inverte l'ordine delle fasi

Errori comuni. Domanda andata molto bene: nessun errore diffuso.

Domanda 2

Funzioni: valore restituito. Che cosa stampa il seguente codice?

```
function f(a, b) {
    if (a > b) return a - b;
    return b - a;
}
console.log(f(3, 8), f(8, 3));
```

Opzione	Risposte
`-5 5`	15%
`5 5`	**54%**
`5 -5`	8%
Nessuna delle risposte è corretta	3%
in bianco	19%

Soluzione: `5 5`

`f` restituisce sempre la differenza **positiva** tra i due numeri.

- `f(3, 8)`: `3 > 8` è falso, si arriva al secondo `return` e si calcola `8 - 3 = 5`
- `f(8, 3)`: `8 > 3` è vero, `return 8 - 3 = 5`

Il risultato non è mai negativo.

Il `return` fa uscire dalla funzione. Qui non c'è nessun `else`, e non serve: se l'`if` è vero si esegue `return a - b` e la funzione **finisce**, quindi il secondo `return` non viene mai letto. Se l'`if` è falso si salta direttamente al secondo `return`. È equivalente a scrivere `if (a > b) { return a - b; } else { return b - a; }`: sono due stili per la stessa cosa. Chi non ha familiarità con la forma senza `else` può essersi confuso nello scegliere la risposta.

Errori comuni. `-5 5` e `5 -5` si ottengono se si **ignora l'`if`** e si calcola sempre `a - b` (o sempre `b - a`). Bisogna invece seguire il flusso: quale `return` viene eseguito dipende dalla condizione.

Domanda 3

Cicli: somma condizionata. Che cosa stampa il seguente codice?

```
let s = 0;
for (let i = 1; i <= 5; i++) {
    if (i % 2 === 0) {
        s += i;
    }
}
console.log(s);
```

Opzione	Risposte
`6`	**71%**
`9`	3%
`15`	0%
Nessuna delle risposte è corretta	12%
in bianco	14%

Soluzione: `6`

Il ciclo scorre `i` da 1 a 5 e **somma solo i pari** (`i % 2 === 0`): 2 + 4 = **6**.

- 9 sarebbe la somma dei dispari (1 + 3 + 5)
- 15 sarebbe la somma di tutti i numeri

Errori comuni. Pochi hanno scelto `9` o `15`, ma più di uno su dieci ha scelto «Nessuna»: vale la pena **tracciare il ciclo** con una tabella (`i`, condizione, `s`) invece di fidarsi dell'intuizione.

Domanda 4

Condizionali: catena if / else if. Che cosa stampa il seguente codice?

```
let n = 15;
if (n % 3 === 0) {
    console.log("A");
} else if (n % 5 === 0) {
    console.log("B");
} else {
    console.log("C");
}
```

Opzione	Risposte
`A`	**56%**
`A e poi B`	27%
`B`	0%
Nessuna delle risposte è corretta	8%
in bianco	8%

Soluzione: `A`

`15 % 3 === 0` è vero, quindi si esegue `"A"`. In una catena `if / else if / else` i rami successivi vengono **saltati** appena uno è eseguito, anche se 15 è divisibile anche per 5.

Errori comuni. `A e poi B` è l'errore più diffuso (più di un quarto della classe): si dimentica che con `else if` **al massimo un ramo** viene eseguito. Confrontate con la domanda 9, dove i due `if` sono separati.

Domanda 5

Operatori aritmetici: resto. Che cosa stampa il seguente codice?

```
let n = 17;
console.log(n % 2);
```

Opzione	Risposte
`1`	**85%**
`0`	5%
`8.5`	7%
Nessuna delle risposte è corretta	3%
in bianco	0%

Soluzione: `1`

`%` è il **resto** della divisione: 17 = 2 · 8 + **1**.

- `8.5` sarebbe il risultato di `17 / 2`

Errori comuni. `8.5` è lo scambio tra `%` e `/`. Ricordate: `%` risponde alla domanda «che cosa avanza?», ed è l'operatore per testare pari/dispari e divisibilità.

Domanda 6

Variabili e tipi: stringhe e numeri. Che cosa stampa il seguente codice?

```
let a = "5";
let b = 3;
console.log(a + b, a - b);
```

Opzione	Risposte
`53 2`	**39%**
`8 2`	7%
`53 NaN`	32%
Nessuna delle risposte è corretta	15%
in bianco	7%

Soluzione: `53 2`

- `a + b` con `a` stringa: `+` fa la **concatenazione**, quindi `"5" + 3` è `"53"`
- `a - b` : `-` esiste solo per i numeri, quindi JavaScript **converte** `"5"` in `5` e ottiene `5 - 3 = 2`

Non si ottiene `NaN` perché `"5"` è convertibile in numero (`NaN` uscirebbe con, per esempio, `"ciao" - 3`).

Errori comuni. `53 NaN` è stata scelta da **quasi un terzo** della classe (32%), contro il 39% di risposte corrette. È la domanda con meno risposte esatte. Si pensa che, essendo `a` una stringa, anche `-` non funzioni. In realtà **solo `+` ha un doppio significato**; `-` , `*` , `/` provano sempre a convertire in numero.

`8 2` (stringa trattata sempre come numero) è l'errore opposto.

Domanda 7

Operatori di confronto e logici. Che cosa stampa il seguente codice?

```
console.log(5 > 3 && 2 === "2");
```

Opzione	Risposte
`false`	**80%**
`true`	7%
`"false"`	7%
`"true"`	3%
in bianco	3%

Soluzione: `false`

- `5 > 3` è `true`
- `2 === "2"` è `false`: `===` confronta **anche il tipo** (numero contro stringa)
- `true && false` dà `false`

`console.log` stampa il booleano **senza virgolette**.

Errori comuni. Le risposte con le virgolette (`"false"`, `"true"`) confondono il **valore** con la sua rappresentazione: un booleano non è una stringa. Chi ha scelto `true` ha probabilmente letto `2 === "2"` come `2 == "2"`.

Domanda 8

Variabili e tipi: Number(). Che cosa stampa il seguente codice?

```
let x = "10";
let y = Number(x) + 5;
console.log(y+1);
```

Opzione	Risposte
`16`	**86%**
`"16"`	5%
`"1051"`	2%
Nessuna delle risposte è corretta	3%
in bianco	3%

Soluzione: `16`

`Number("10")` dà il **numero** 10, quindi `y = 10 + 5 = 15`. Poi `y + 1` è una somma tra numeri: **16**.

- `"1051"` uscirebbe solo con una concatenazione (senza `Number`)

Errori comuni. Quasi tutti rispondono bene. Le poche risposte con le virgolette (`"16"`) confondono numero e stringa: `Number(...)` produce un numero, e `console.log` non mostra virgolette.

Domanda 9

Condizionali: if separati. Che cosa stampa il seguente codice?

```
let n = 7;

if (n > 5) {
    console.log("A");
}

if (n > 3) {
    console.log("B");
} else {
    console.log("C");
}
```

Opzione	Risposte
`A`	7%
`A e poi B`	**88%**
`A e poi C`	2%
Nessuna delle risposte è corretta	0%
in bianco	3%

Soluzione: `A e poi B`

Qui ci sono **due `if` separati**, non una catena.

- primo `if`: `7 > 5` è vero, stampa `A`
- secondo `if`: `7 > 3` è vero, stampa `B` (l'`else` non viene eseguito)

Output: `A` e poi `B`.

Errori comuni. Domanda andata bene, ma ricordate il contrasto con la domanda 4: lì l'`else if` escludeva il ramo successivo, qui **no**, perché ogni `if` è indipendente.

Modulo A: la prova

Seconda parte della simulazione: **programmi JavaScript da completare** scegliendo, per ogni menu a tendina, il frammento di codice corretto.

- 3 esercizi da 10 punti (cicli, funzioni, `switch`), 30 minuti
- 4 menu a tendina per esercizio, da 2.5 punti ciascuno
- risposta errata −0.75, lasciata in bianco 0

Esercizio	Argomento	Menu	Punti per menu
1	Cicli: divisori e numeri primi	4	2.5
2	Funzioni: multipli	4	2.5
3	`switch`: giorno dell'anno e stagione	4	2.5

Modulo A: come è andata

- **53** studenti hanno svolto la prova
- punteggio medio: **17.7** su 30; mediana **20.2**

Esercizio	Argomento	Risposte corrette (media sui 4 menu)
1	Cicli	68%
2	Funzioni	72%
3	`switch`	62%

In ogni esercizio c'è **un menu** che ha fatto molta più fatica degli altri: li vediamo uno per uno.

Modulo A - Esercizio 1: cicli

Scrivere un programma che chieda un numero intero N maggiore di 1, **ripetendo la richiesta** finché non è valido. Poi deve:

- stampare tutti i divisori di N (1 ed N compresi), in ordine crescente
- stampare la somma dei divisori
- stampare "N è primo" se N ha esattamente due divisori, altrimenti "N non è primo"

Esempio (N = 12):

```
Divisori di 12:
1
2
3
4
6
12
Somma dei divisori: 28
12 non è primo
```

Modulo A - Esercizio 1: soluzione

```
let N;
do {
  N = Number(prompt("N > 1:"));
} while (!Number.isInteger(N) || N <= 1);
let conta = 0;
let somma = 0;
console.log("Divisori di " + N + ":");
for (let d = 1; d <= N; d++) {
  if (N % d === 0) {
    console.log(d);
    conta++;
    somma += d;
  }
}
console.log("Somma dei divisori: " + somma);
let esito = conta === 2 ? " è primo" : " non è primo";
console.log(N + esito);
```

Menu 1	Risposte
`!Number.isInteger(N) || N <= 1`	**45%**
`Number.isInteger(N) && N > 1`	23%
`!Number.isInteger(N) && N <= 1`	17%
`Number.isInteger(N) || N > 1`	11%

Il `while` dice quando **ripetere**: finché il valore **non** è valido, cioè non intero **oppure** ≤ 1.

- `Number.isInteger(N) && N > 1` (23%) è la condizione di un valore **valido**
- `!Number.isInteger(N) && N <= 1` (17%): con `&&` il ciclo si ferma per N = 0 o 2.5

Nel `do ... while` si esprime sempre ciò che **non** va bene (quando il valore è valido, il ciclo deve fermarsi). Si può arrivarci in due modi: scrivendo direttamente i casi da rifiutare (non intero **oppure** ≤ 1), oppure negando la condizione del valore valido: `!(Number.isInteger(N) && N > 1)`. Sono equivalenti per le leggi di De Morgan.

Per controllare che un numero sia intero si può usare anche `Math.floor(N) !== N`, cioè confrontare il numero con la sua parte intera (l'arrotondamento per difetto): se sono diversi, `N` ha una parte decimale. `!Number.isInteger(N)` è però più leggibile, e quindi preferibile.

Menu 2	Risposte
`%`	**87%**
`/`	9%
`*`	2%
`+`	0%

`%` dà il **resto**: `d` è un divisore di N se `N % d === 0`. Andato bene; `/` (9%) darebbe il quoziente.

Menu 3	Risposte
`conta++`	**66%**
`d++`	8%
`N++`	6%
`conta = d`	13%

`conta++` aumenta il contatore di 1 a ogni divisore trovato.

- `conta = d` (13%) assegna il divisore, non conta
- `d++` e `N++` (8%, 6%) alterano il ciclo o il numero

Menu 4	Risposte
`=== 2`	**75%**
`=== 1`	9%
`=== N`	4%
`=== 0`	2%

Un numero è primo se ha **esattamente due** divisori (1 e se stesso): `conta === 2`. `=== 1` (9%) vale solo per N = 1, escluso dalla validazione.

Modulo A - Esercizio 2: funzioni

Definire due funzioni:

- `isMultiplo(a, b)`: `true` se a è multiplo di b, `false` altrimenti
- `sommaMultipli(N, k)`: somma dei multipli di k tra 1 ed N (estremi inclusi), **usando `isMultiplo`**

Nel programma principale: leggere N e k, chiamare `sommaMultipli`, stampare il risultato e stampare se è pari o dispari (usando ancora `isMultiplo`).

Esempio (N = 10, k = 3):

```
Somma dei multipli di 3 fino a 10: 18
Il risultato è pari
```

Modulo A - Esercizio 2: soluzione

```
function isMultiplo(a, b) {
  return a % b === 0;
}

function sommaMultipli(N, k) {
  let somma = 0;
  for (let i = 1; i <= N; i++) {
    if (isMultiplo(i, k)) {
      somma += i;
    }
  }
  return somma;
}

let N = Number(prompt("N:"));
let k = Number(prompt("k:"));
let risultato = sommaMultipli(N, k);
console.log("Somma dei multipli di " + k +
  " fino a " + N + ": " + risultato);
let tipo = isMultiplo(risultato, 2) ? "pari" : "dispari";
console.log("Il risultato è " + tipo);
```

Menu 1	Risposte
`N, k`	**94%**
`k, N`	2%
`N`	0%
`i, k`	2%

La funzione si chiama `sommaMultipli(N, k)` e il corpo usa `N` e `k`. `i` è la variabile del ciclo, non un parametro.

Menu 2	Risposte
`isMultiplo(i, k)`	**62%**
`isMultiplo(k, i)`	9%
`isMultiplo(N, k)`	17%
`isMultiplo(somma, i)`	6%

Si sommano gli `i` che sono multipli di `k`: `isMultiplo(i, k)`. Nella **chiamata** si passano come argomenti, nell'ordine, **ciò che deve essere il multiplo** (`i`) e **ciò che deve essere il divisore** (`k`). Con `isMultiplo(k, i)` si controllerebbe il contrario, cioè se `k` è multiplo di `i`; con `isMultiplo(n, k)` si controllerebbe sempre `n`, giro dopo giro, invece di guardare i numeri da `1` a `n` uno per uno.

- `isMultiplo(N, k)` (17%) non dipende da `i`: sempre uguale a ogni giro
- `isMultiplo(k, i)` (9%): argomenti **scambiati**

Menu 3	Risposte
`sommaMultipli(N, k)`	**77%**
`sommaMultipli(k, N)`	8%
`sommaMultipli`	9%
`isMultiplo(N, k)`	2%

Si chiama la funzione con gli argomenti nell'ordine della firma. `sommaMultipli` senza parentesi (9%) è la funzione stessa, non il suo risultato.

Menu 4	Risposte
`isMultiplo(risultato, 2)`	**53%**
`isMultiplo(2, risultato)`	8%
`risultato % 2`	30%
`sommaMultipli(risultato, 2)`	6%

Pari = multiplo di 2: `isMultiplo(risultato, 2)`. `risultato % 2` (30%) vale 1 per i **dispari**, cioè "vero": il significato si **inverte**. Inoltre l'operatore ternario vuole una condizione **booleana**, e `%` è un operatore **aritmetico**: restituisce un numero, non `true` o `false`. Un confronto, `risultato % 2 === 0`, sarebbe stato invece corretto (e dà lo stesso risultato di `isMultiplo(risultato, 2)`).

Modulo A - Esercizio 3: switch

Chiedere un mese M (da 1 a 12, **ripetendo** finché non è valido) e un giorno G. Anno **non bisestile**. Stampare:

- il numero d'ordine del giorno nell'anno, con **un solo** `switch` sul mese che sfrutta la mancanza di `break` per sommare i giorni dei mesi precedenti
- la stagione, con un secondo `switch` a `case` raggruppati
- "Primo semestre" se il giorno cade entro il 30 giugno, "Secondo semestre" altrimenti

Esempio (M = 3, G = 15):

```
Il 15/3 è il giorno n. 74 dell'anno
Stagione: primavera
Primo semestre
```

Modulo A - Esercizio 3: soluzione

```
let giorni = G;
switch (M) {
  case 12: giorni += 30;
  case 11: giorni += 31;
  case 10: giorni += 30;
  case 9: giorni += 31;
  case 8: giorni += 31;
  case 7: giorni += 30;
  case 6: giorni += 31;
  case 5: giorni += 30;
  case 4: giorni += 31;
  case 3: giorni += 28;
  case 2: giorni += 31;
  case 1: break;
}

let stagione;
switch (M) {
  case 12:
  case 1:
  case 2:
    stagione = "inverno";
    break;
  case 3:
  case 4:
  case 5:
    stagione = "primavera";
    break;
  case 6:
  case 7:
  case 8:
    stagione = "estate";
    break;
  default:
    stagione = "autunno";
}
```

Menu 1	Risposte
`M`	**40%**
`G`	19%
`giorni`	30%
`M - 1`	9%

I `case` (da 1 a 12) confrontano il **mese**: `switch (M)`. `giorni` (30%) e `G` (19%) sono valori diversi dal mese.

Un modo per orientarsi: `giorni` è la variabile che lo `switch` sta **costruendo** (accumula la somma), e non si può fare lo `switch` proprio sulla variabile che si modifica dentro. A guidare le somme è il **mese**: per sapere quanti giorni aggiungere basta sapere in che mese si è.

Menu 2	Risposte
`28`	**57%**
`29`	9%
`30`	13%
`31`	17%

Per marzo si aggiunge febbraio: **28** giorni, perché l'anno è **non** bisestile. Le altre opzioni: 31 (17%), 30 (13%), 29 (9%).

Menu 3	Risposte
`break;`	**98%**
`continue;`	0%
`return;`	0%
`stop;`	0%

`break;` chiude il gruppo `case 12 / 1 / 2` e impedisce di cadere in "primavera". `continue` e `return` servono in cicli e funzioni, `stop` non esiste.

Menu 4	Risposte
`default:`	**55%**
`case 0:`	4%
`else:`	2%
`case 9, 10, 11:`	38%

I mesi restanti (9, 10, 11) sono il caso `default:`. `case 9, 10, 11:` (38%) non è sintassi valida in JavaScript: a livello logico la scelta è sensata (si sono considerati 12, 1, 2, 3, …, 8 e mancano 9, 10, 11), ma ogni `case` vuole **un solo** valore. Per dare lo stesso risultato a più valori si scrivono più `case` di fila (`case 9: case 10: case 11:`), oppure si usa `default:`.

Un consiglio per il compitino vero: sono ammessi gli appunti, quindi conviene prepararsi uno **schemino per ogni costrutto** (`if / else`, `switch`, `for`, `while`, `do ... while`, funzioni): la sintassi (le parole chiave, cosa deve essere booleano, cosa un numero, cosa una stringa), la semantica in poche parole, e un esempio significativo. La sintassi non si ragiona, si deve ricordare: se ce l'avete sotto mano, la parte di sintassi di molte domande è risolta.

Modulo B: la prova

Terza parte della simulazione: **domande con due menu a tendina** sulla prima metà del Modulo B.

- 3 domande da 10 punti, 20 minuti
- 2 menu a tendina per domanda, da 5 punti ciascuno
- risposta errata −1.5, lasciata in bianco 0

Domanda	Argomento	Menu	Punti per menu
1	Basi di numerazione	2	5
2	Insiemi	2	5
3	Relazioni	2	5

Modulo B: come è andata

- **46** studenti hanno svolto la prova
- punteggio medio: **18.0** su 30; mediana **19.0**

Domanda	Argomento	Risposte corrette (media sui 2 menu)
1	Basi di numerazione	68%
2	Insiemi	63%
3	Relazioni	65%

Modulo B - Domanda 1: basi di numerazione

Sia dato il numero binario **101101**$_2$, che in base 10 vale 45.

- Sommando in binario **1111**$_2$ (15 in decimale) a 101101$_2$ si ottiene: `111100`, `111010`, `110100`, `1011100`
- Il numero 101101$_2$, scritto in esadecimale, è: `2D`, `D2`, `2B`, `4D`

Modulo B - Domanda 1: soluzione

Menu 1	Risposte
`111100`	**74%**
`111010`	0%
`110100`	7%
`1011100`	4%

Si somma **colonna per colonna, da destra a sinistra**, come in decimale, ma con una sola regola in più: `1 + 1 = 10` (scrivo `0`, riporto `1`).

Somma della colonna	Scrivo	Riporto
`0 + 0`	0	0
`0 + 1`	1	0
`1 + 1`	0	1
`1 + 1 + 1` (col riporto)	1	1

```
riporti   0 1 1 1 1 0
            1 0 1 1 0 1
          + 0 0 1 1 1 1
          -------------
            1 1 1 1 0 0
```

Colonna per colonna, da destra:

1. `1 + 1 = 10`: scrivo 0, riporto 1
2. `0 + 1 + 1 (riporto) = 10`: scrivo 0, riporto 1
3. `1 + 1 + 1 (riporto) = 11`: scrivo 1, riporto 1
4. `1 + 1 + 1 (riporto) = 11`: scrivo 1, riporto 1
5. `0 + 0 + 1 (riporto) = 1`: scrivo 1
6. `1 + 0 = 1`: scrivo 1

Verifica in decimale: 45 + 15 = 60 = 32 + 16 + 8 + 4 = `111100`$_2$.

Si può anche procedere **senza sommare in binario**: il testo già dice che `101101`$_2$ vale 45 e `1111`$_2$ vale 15, quindi si somma in decimale (45 + 15 = 60) e poi si **converte 60 in binario** con le divisioni successive per 2 (leggendo i resti dal basso

Menu 2	Risposte
`2D`	**63%**
`D2`	4%
`2B`	4%
`4D`	4%

verso l'alto). Le due strade portano allo stesso risultato: quella che vi sembra più comoda va benissimo.

Ogni cifra esadecimale (base 16) corrisponde a **esattamente 4 bit**, quindi:

1. si raggruppano i bit **a quattro a quattro, partendo da destra**; se il gruppo a sinistra è incompleto si aggiungono zeri davanti: `101101` → `10 1101` → `0010 1101`
2. si converte ogni gruppo in una cifra, con i pesi 8, 4, 2, 1
3. si riscrivono le cifre **nello stesso ordine** dei gruppi

Gruppo	Calcolo	Cifra
`0010`	0·8 + 0·4 + 1·2 + 0·1 = 2	2
`1101`	1·8 + 1·4 + 0·2 + 1·1 = 13	D

Le cifre oltre il 9 sono lettere: `10 = A`, `11 = B`, `12 = C`, `13 = D`, `14 = E`, `15 = F`.

Risultato: `2D` $_{16}$. Verifica: 2·16 + 13 = 45. `D2` ha i gruppi nell'ordine sbagliato.

Modulo B - Domanda 2: insiemi

In una scuola sono iscritte 90 persone. Corsi: Matematica (M), Fisica (F), Informatica (I).

Si sa che |M| = 42, |F| = 35, |I| = 30, |M∩F| = 15, |M∩I| = 12, |F∩I| = 10, |M∩F∩I| = 6.

- Chi frequenta almeno un corso, |M∪F∪I|, sono: `76`, `82`, `70`, `94`
- Chi frequenta esattamente due corsi sono: `19`, `25`, `37`, `13`

Modulo B - Domanda 2: soluzione

Menu 1	Risposte
`76`	**65%**
`82`	4%
`70`	13%
`94`	0%

Per contare le persone in **almeno un corso** non basta sommare le dimensioni: chi segue più corsi viene contato più volte. Si usa il **principio di inclusione-esclusione**:

$|M \cup F \cup I| = |M| + |F| + |I| - |M \cap F| - |M \cap I| - |F \cap I| + |M \cap F \cap I|$

1. si sommano i tre insiemi: 42 + 35 + 30 = 107
2. si tolgono le intersezioni a coppie (chi è in due corsi era contato due volte): 107 − 15 − 12 − 10 = 70
3. si **riaggiunge** la tripla intersezione: chi segue tutti e tre i corsi è stato contato 3 volte al passo 1 e tolto 3 volte al passo 2, quindi ora non compare più: 70 + 6 = **76**

`70` (13%) è l'errore tipico: ci si ferma al passo 2. Il totale di 90 iscritti non serve.

Un consiglio pratico: poiché si ha a disposizione un foglio, conviene **disegnare** i tre insiemi (e le loro intersezioni) e controllare visivamente quali regioni sono state contate: se dopo aver tolto le coppie manca la zona centrale, si capisce subito che va riaggiunta.

Menu 2	Risposte
19	**61%**
25	4%
37	24%
13	0%

Si vuole contare chi è in **esattamente due** corsi, cioè chi sta in una intersezione a coppie ma **non** nella terza.

Ogni intersezione a coppie comprende anche chi segue tutti e tre i corsi (la tripla, 6 persone): va tolta.

Solo questi due corsi	Calcolo	Persone
M e F (non I)	15 – 6	9
M e I (non F)	12 – 6	6
F e I (non M)	10 – 6	4

Totale: 9 + 6 + 4 = **19**.

Controllo: 15 + 12 + 10 = 37 conta la tripla **3 volte**; togliendola: 37 – 3·6 = 19. `37` (24%) è proprio la somma senza togliere la tripla.

Modulo B - Domanda 3: relazioni

A = {1, 2, 3, 4}, R = {(1, 2), (2, 3), (3, 3), (4, 1)}.

- La relazione R è: `né riflessiva né simmetrica`, `riflessiva ma non simmetrica`, `simmetrica ma non riflessiva`, `sia riflessiva sia simmetrica`
- Per renderla transitiva bisogna aggiungere almeno: `3`, `1`, `2`, `4` coppie

Modulo B - Domanda 3: soluzione

Menu 1	Risposte
`né riflessiva né simmetrica`	**78%**
`riflessiva ma non simmetrica`	13%
`simmetrica ma non riflessiva`	0%
`sia riflessiva sia simmetrica`	2%

Si controllano le definizioni, una alla volta:

- **riflessiva**: per **ogni** elemento a di A deve esserci (a, a)
- **simmetrica**: per **ogni** coppia (a, b) deve esserci anche (b, a)

Proprietà	Cosa serve	In R	Esito
riflessiva	(1,1), (2,2), (3,3), (4,4)	c'è solo (3,3)	no
simmetrica	(2,1) per (1,2)	(2,1) non c'è	no

Per escludere una proprietà **basta un controesempio**. `riflessiva ma non simmetrica` (13%) confonde la presenza di (3,3) con la riflessività: servono tutte e quattro le coppie (a, a).

Menu 2	Risposte
3	**52%**
1	13%
2	13%
4	9%

Transitiva: se ci sono (a, b) e (b, c), deve esserci anche (a, c).

Metodo: per ogni coppia (a, b) si guardano le coppie che **iniziano con b**, e si controlla che (a, c) ci sia.

Coppia	Coppie che iniziano con l'ultimo elemento	Serve	C'è?
(1,2)	(2,3)	(1,3)	no
(2,3)	(3,3)	(2,3)	sì
(3,3)	(3,3)	(3,3)	sì
(4,1)	(1,2)	(4,2)	no

Si aggiungono (1,3) e (4,2). Ma le **nuove coppie creano nuove catene**, quindi si ricontrolla:

- (1,3) con (3,3) richiede (1,3): c'è
- (4,2) con (2,3) richiede **(4,3)**: manca, si aggiunge
- (4,3) con (3,3) richiede (4,3): c'è

Fine: servono **3** coppie, (1,3), (4,2) e (4,3). Chi si ferma al primo giro risponde 2 (13%).

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