

Laboratorio I

a.a. 2021/2022

JavaScript controllo del flusso

Contenuti

- Soluzione esercizio 'secondi'
- Controllo del flusso
 - Comandi condizionali: `if`, `switch`
 - Comandi iterativi: `for`, `while`

Soluzione esercizio 'secondi'

- Scrivere un programma che, dato un numero di secondi dalla tastiera, calcola e mostra a video il numero di ore, minuti e secondi inclusi.
- Modifica: Stampa ore e minuti solo se diverso da 0

Controllo del flusso

- Programma - serie di comandi da eseguire in ordine
- A volte serve cambiare il codice in base ai risultati dei calcoli, o in base all'input dell'utente
 - Comandi condizionali - una parte del codice viene eseguita solo se una condizione è soddisfatta
- A volte serve ripetere la stessa operazione un numero di volte
 - Comandi iterativi
 - Un numero fissate di volte - `for`
 - Un numero sconosciuto di volte - `while`



Comando condizionale - `if`

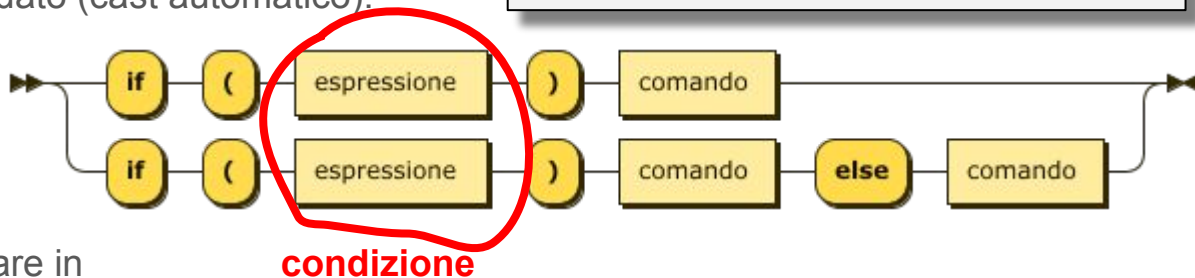


- Comando `if..else`

- Esegue primo *comando* solo se *espressione* vale `true`, altrimenti esegue secondo *comando*
- *espressione* ha di solito un valore Booleano - può essere qualsiasi tipo di dato (cast automatico).

```
comando_if ::=  
if ( espressione ) comando |  
if ( espressione ) comando else comando
```

```
if (m>0)  
  console.log(`${m} minuti`);
```



- Operatore logico `?:`

- Simile al `if` però da usare in espressioni

```
console.log(m>0?`${m} minutes`:``);
```

```
exp_logica_terziaria ::= espressione ? espressione : espressione
```



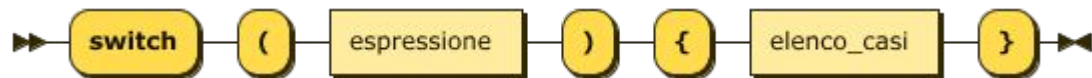
Comando condizionale - switch

- Permette di controllare più valori possibili della stessa espressione
- Esempio: stampare il nome del giorno della settimana in base all'input.

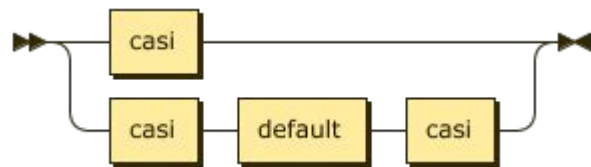
```
switch(espressione) {  
    case x:  
        // comando  
        break;  
    case y:  
        // comando  
        break;  
    default:  
        // comando  
}
```

Comando condizionale - switch

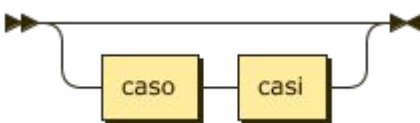
comando_switch:



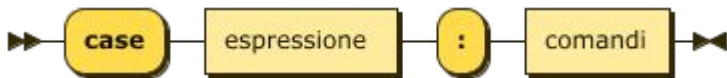
elenco_casi:



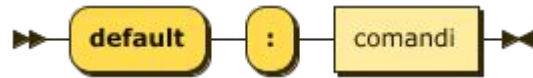
casi:



caso:



default:



comando_switch ::=

'switch' '(' espressione ')' '{' elenco_casi '}'

elenco_casi ::= casi | casi default casi

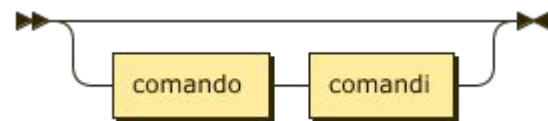
casi ::= | caso casi

caso ::= 'case' espressione ':' comandi

default ::= 'default' ':' comandi

comandi ::= | comando comandi

comandi:



Casi strani ma corretti

Caso tipico:

```
switch (n) {  
  case 1: /* fa qualcosa */  
    | | | | break;  
  case 2: /* fa qualcos'altro */  
    | | | | break;  
  default: /* disperati */  
    | | | | break  
}
```

Però è anche valido se...

```
switch (n) {  
  case 1: /* fa qualcosa */  
  case 2: /* fa qualcos'altro */  
    | | | | break;  
  default: /* disperati */  
    | | | | break  
}
```

La grammatica non dice che il **break** dopo ogni caso è obbligatorio, quindi si può omettere. In questo caso, se n vale 1, fa prima qualcosa, e poi continua con qualcos'altro. Se n vale 2, fa solo qualcos'altro. Se n è diverso da 1 e 2, si dispera.

Casi strani ma corretti

Caso tipico:

```
switch (n) {  
  case 1: /* fa qualcosa */  
    break;  
  case 2: /* fa qualcos'altro */  
    break;  
  default: /* disperati */  
    break;  
}
```

Però è anche valido se...

```
switch (1) {  
  case n/2: /* fa qualcosa */  
    break;  
  case n+1: /* fa qualcos'altro */  
    break;  
  default: /* disperati */  
    break;  
}
```

La grammatica dice che sia la parte dopo **switch** che quella dopo **case** sono *espressioni*. Non è detto che la prima sia una condizione e la seconda una costante. In questo caso, se $n/2=1$ si fa qualcosa, altrimenti se $n+1=1$ si fa qualcos'altro, altrimenti si dispera.

Casi strani ma corretti

Caso tipico:

```
switch (n) {  
  case 1: /* fa qualcosa */  
    break;  
  case 2: /* fa qualcos'altro */  
    break;  
  default: /* disperati */  
    break;  
}
```

Però è anche valido se...

```
switch (n) {  
  case 1: /* fa qualcosa */  
    break;  
  default: /* disperati */  
    break;  
  case 2: /* fa qualcos'altro */  
    break;  
}
```

Il **default** può anche essere in mezzo alla lista di casi, e in questo caso viene saltato finché non vengono prima controllati tutti i **case** (ma i vostri colleghi informatici di tutto il mondo non vorranno più parlarvi se lo fate: si tratta di un idioma universale!)

Casi strani ma corretti

Caso tipico:

```
switch (n) {  
  case 1: /* fa qualcosa */  
    break;  
  case 2: /* fa qualcos'altro */  
    break;  
  default: /* disperati */  
    break;  
}
```

Però è anche valido se...

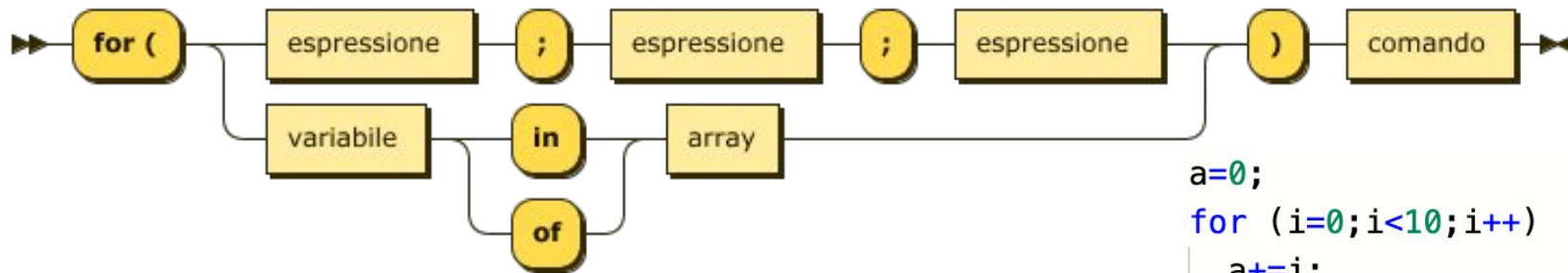
```
switch ("2") {  
  case 1: /* fa qualcosa */  
    break;  
  case 2: /* fa qualcos'altro */  
    break;  
  default: /* disperati */  
    break;  
}
```

Le espressioni possono essere qualunque, però la verifica di uguaglianza fra l'espressione dello switch e quella del case è fatta con `===` (quindi, sia valore che tipo). Nel caso sopra, `"2"===2` è falso, quindi si dispera.

Comandi condizionali

- Esercizi:
 - (Pari) Scrivere un programma che legga da tastiera un numero e stabilisca se pari o meno, stampando in uscita rispettivamente 1 o 0.
 - (Max) Scrivere un programma che legga 3 numeri da tastiera e mostri a video il loro massimo.
 - (Calcolatore) Scrivere un programma che legga due numeri e un operatore tra +, -, * e / da tastiera e mostri a video il risultato dell'operazione sui due numeri.

Comando iterativo - for



```
a=0;  
for (i=0;i<10;i++)  
| a+=i;
```

```
comando_for ::= 'for' '('  
  ( espressione ';' espressione ';' espressione  
  | variabile ( 'in' | 'of' ) array )  
  ')' comando
```

```
let studenti=['Matt','Alice','Bob'];  
for (i in studenti)  
| console.log(studenti[i]);
```

- Permette di ripetere un comando o set di comandi un numero di volte
- Di solito usato quando si conosce il numero di iterazioni
- Esempio: somma dei primi n numeri naturali

```
let studenti=['Matt','Alice','Bob'];  
for (s of studenti)  
| console.log(s);
```

Comando iterativo - while, do while



- Un altro tipo di comando per ripetere istruzioni
- Esegue *comando* fino a quando *espressione* diventa *false*
- *espressione* ha di solito un valore Booleano
 - Può essere qualsiasi tipo di dato (cast automatico).
- Di solito usato quando non si conosce il numero di iterazioni
- Esempio: sommare numeri in input finché l'utente inserisce -1

```
comando_while ::= while ( espressione ) comando
```



```
comando_do ::= do comando while ( espressione )
```



```
while(condizione){  
    //comandi  
}  
  
do{  
    //comandi  
} while (condizione);
```

Comandi iterativi

- A volte bisogna fermare una o tutte le iterazioni
- Comandi :
 - `break` - esce dal comando iterativo (`for`, `while`) fermandolo completamente e continua il programma
 - `continue` - salta l'iterazione corrente e passa alla prossima, continuando l'iterazione

Misurare il tempo di esecuzione

- Comando `Date.now()` - tempo - *timestamp* in millisecondi
- Chiedere il tempo all'inizio e alla fine del programma

```
let start=Date.now();
```

```
//programma...
```

```
let end=Date.now();
```

```
console.log(`Tempo di esecuzione: ${end-start} ms`);
```


Comandi iterativi - Esercizi

(Sommatoria) Si scriva un programma che dato un intero n calcoli e stampi la somma dei numeri da 1 a n .

(Accumulatore) Si scriva un programma che legge numeri finché la somma non supera 101, quindi stampi la somma.

(Primo) Si scriva un programma che legge un intero n e valuti se è primo (non esiste un numero d in $[2, n-1]$ tale che n sia divisibile per d). Si stampi il tempo di esecuzione del programma.

(MultiPrimo) Si modifichi il programma di sopra per leggere 10 numeri n e calcoli per ciascuno se si tratta di un numero primo.

Q & A

Esercizi Extra

(Fattoriale) Si scriva un programma che dato un intero n calcoli e stampi il suo fattoriale. Si ricorda che il fattoriale di n è $n! = 1*2*....*(n-1)*n$

(Asterischi) Si scriva un programma che dato un intero n stampi n asterischi sulla prima linea, $n - 2$ asterischi sulla seconda linea, $n - 4$ sulla terza e così via, fino ad arrivare a stampare uno o due asterischi sull'ultima linea

(Stampa selettiva) Si scriva un programma che legga da tastiera di interi e stampi l'elemento letto se rispetta una delle seguenti proprietà: (i) E' positivo e pari, oppure (ii) è negativo e preceduto (nell'ordine di inserimento) da un intero con valore maggiore o uguale. Terminare la l'acquisizione alla lettura di uno zero.

(Media) Si scriva un programma che legga da tastiera 10 interi e stampi la media aritmetica di tutti i valori diversi da zero e di segno uguale all'ultimo valore della sequenza.