



Laboratorio I

a.a. 2026/2027

TypeScript - grammatiche per letterali e controllo di flusso
condizionale

Contenuti

- Grammatiche per letterali
- Controllo del flusso
 - Comandi condizionali: `if`, `switch`
 - Comandi iterativi: `for`, `while` (... next lecture!)

Recap: Tipi di dato primitivo e valori letterali in JS/TS

JavaScript consente di denotare **valori letterali** di determinati tipi.

I seguenti sono i corrispondenti tipi in TypeScript:

- **boolean**: `true`, `false`
- **undefined**: `undefined` (→ **non è stato definito** nessun valore)
- **null**: `null` (→ so che **non c'è** un valore)
- **number**: `1`, `5`, `-53.38`, `12.3e4` (numeri “normali”, $-(2^{53}-1) \leq n \leq 2^{53}-1$)
- **bigint**: `90071992547434344169871610992n` (interi a precisione infinita)
- **string**: `“ciao ugo”`, `“हिन्दू”`, `‘hello’`, ``x vale ${x}`` (sequenze di caratteri)

Una grammatica per i letterali

Finora abbiamo fornito *esempi* di come si scrivono i letterali, ma ciò non è soddisfacente... vogliamo ottenere la massima precisione!

Nel corso di P&A avete visto il concetto di **grammatica**: usiamolo!

La **Backus-Naur Form** (BNF) è un modo di descrivere sinteticamente la grammatica di un linguaggio

La grammatica è data da un insieme di **produzioni**, ciascuna delle quali ha la forma:

$$classe ::= definizione_1 \mid definizione_2 \mid \dots \mid definizione_n$$

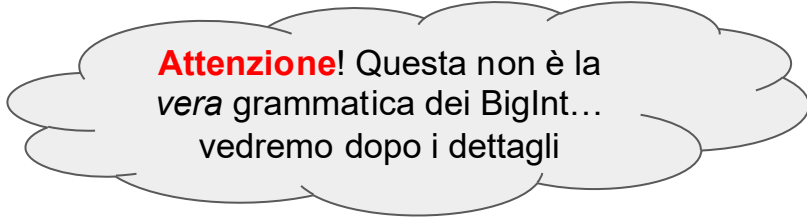
e ogni definizione è una sequenza di simboli terminali e *classi*

Esempio: grammatica per i BigInt (semplificata)

⇒ *letterale_bigint* ::= *intero* n

intero ::= *cifra* | *cifra intero*

cifra ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

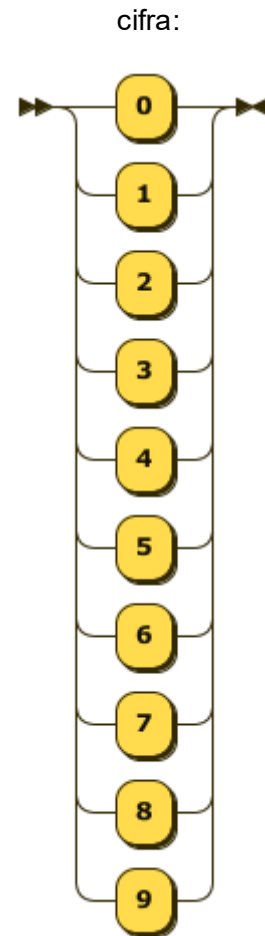
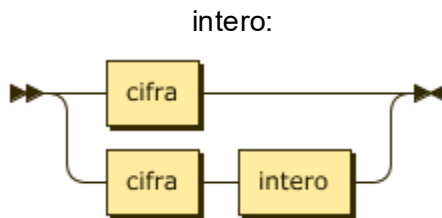


Attenzione! Questa non è la
vera grammatica dei BigInt...
vedremo dopo i dettagli

Partendo dal **simbolo iniziale** *letterale_bigint*, questa grammatica produce frasi come 8374n, 2n, 0n, 00038n, 9823410713087529813874221245345n

Diagrammi di sintassi

Può essere comoda anche una rappresentazione alternativa per la grammatica:

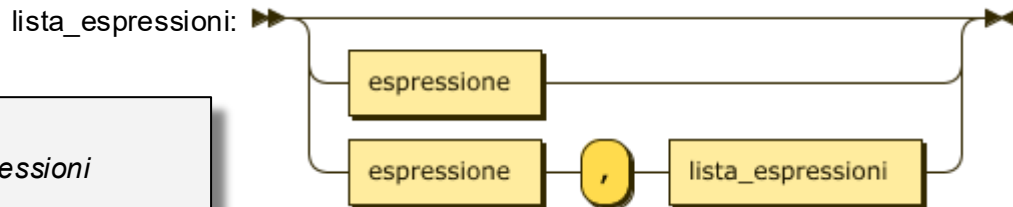


Diagrammi di sintassi

Alcuni altri esempi (semplificati rispetto alla realtà!)



```
letterale_array ::= [ lista_espressioni ]  
lista_espressioni ::= | espressione | espressione , lista_espressioni
```

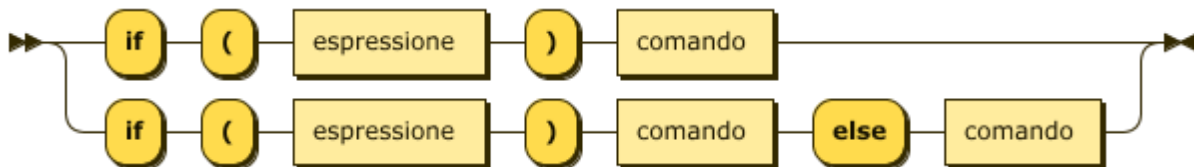


comando_while:



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

comando_if:



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

Dichiarazione di variabile e assegnamento

Una *variabile* è una locazione di memoria in cui è memorizzato un valore

Un *identificatore* è invece un nome, scelto a piacere ma con alcune restrizioni lessicali

- Per esempio, in JS/TS gli identificatori possono essere lunghi a piacere, essere composti di lettere, cifre, \$ e _ , e non possono iniziare con una cifra

Le **dichiarazioni** associano un identificatore a una variabile (operazione di **binding**)

- (**var** | **let** | **const**) Id [= Expr]



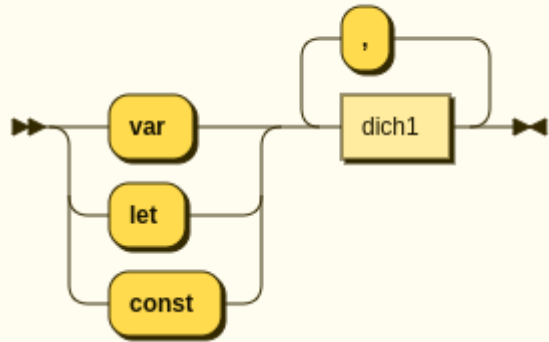
Per brevità, useremo spesso
“variabile” anziché “identificatore
associato a una variabile”

Gli **assegnamenti** memorizzano un nuovo valore in una locazione

- Id = Expr

Dichiarazione di variabile e assegnamento

dichiarazione:



dich1:



comando_ass:



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 - 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 fissato di volte - `for`
 - Un numero sconosciuto di volte - `while`



Comando condizionale - if

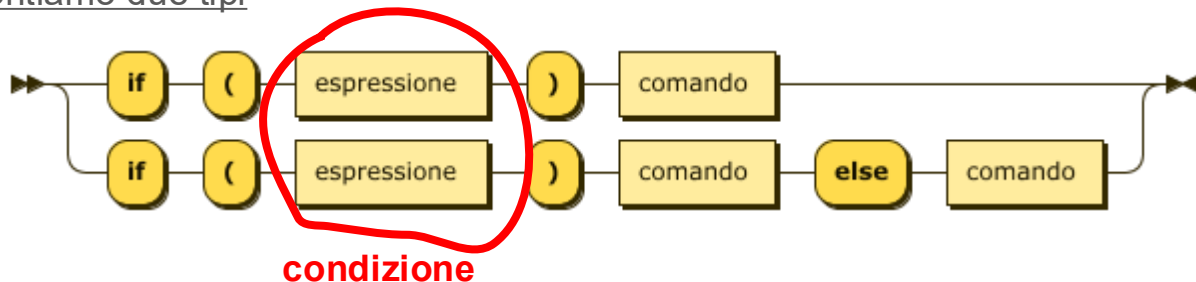


- Comando `if..else`

- 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). TS segnala **errore** se confrontiamo due tipi incompatibili!

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

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



- Operatore logico `?:`

- Simile a `if` però da usare in espressioni

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

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



Approfondimento: **falsy** e **truthy** values

- Ci sono 8 valori **falsy**:
 - **false** — boolean false
 - **0** — numero zero
 - **""** — stringa vuota
 - **null**
 - **undefined**
 - **NaN** — Not A Number
 - **-0** — numero zero negativo
 - **0n** — numero zero come BigInt
- Gli altri valori sono **truthy**, e.g.
 - **true** — boolean true
 - **-3,100** — numeri diversi da zero
 - **"ciao"** — stringa non vuota
 - gli oggetti vuoti **{}**, array vuoti **[]**, la stringa **"false"** o la stringa **"0"**.

Comando condizionale - switch

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

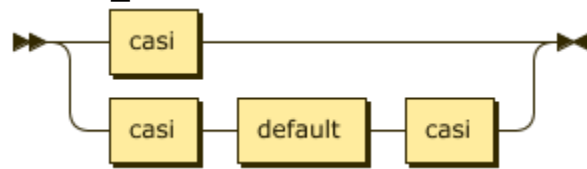
```
switch(espressione) {  
    case x:  
        // comando  
        break;  
    case y:  
        // comando  
        break;  
    default:  
        // comando  
}
```

Comando condizionale - switch

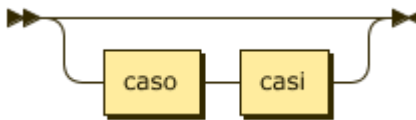
comando_switch:



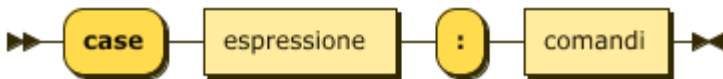
elenco_casi:



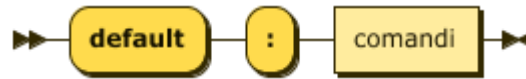
casi:



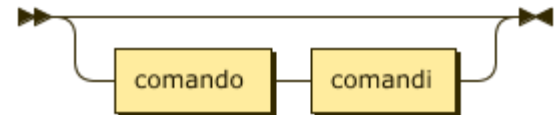
caso:



default:



comandi:



```
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
```

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. Perciò, 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, ma 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, corretti in JS ma non in TS

Caso tipico:

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

TS segnalerebbe come errore di compilazione il seguente confronto:

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

In TS il confronto tra tipi diversi come qui una stringa "2" e il numero 2 è un **errore**. In generale, 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). In JS il caso sopra, "2"===2 è falso, quindi si dispera.

Q & A

Esercizi (1)

- (**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.

Esercizi (2)

- (**Media**) Si scriva un programma che legga da tastiera 4 interi e stampi la media aritmetica di tutti i valori diversi da zero e di segno uguale all'ultimo valore della sequenza.
- (**Che tipo di dato?**) Scrivere un programma che, dato un valore, utilizzi l'operatore `typeof` insieme a uno switch per stampare a video una frase che descriva il tipo del valore (es. "è un numero", "è una stringa", "è un valore booleano", ecc.).
- (**Accesso**) Scrivere un programma che legga da tastiera una stringa che rappresenta una password, e usi l'operatore ternario per stampare "accesso negato" se la stringa è vuota, altrimenti "accesso consentito".