Laboratorio I
a.a. 2025/2026

Funzioni

Contenuti

- Analisi di funzioni matematiche
  - Calcolo del valore di una funzione in un punto o un dominio
- Funzioni in JavaScript

Funzioni in matematica

- Esempi:
  - $f : \mathbb{R} \to \mathbb{R}$, $f(x) = x^2$
  - $f : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, $f(x,y) = x+y$
  - $f : \mathbb{N} \to \{0,1\}$, $f(x) = 1$ se x è pari, 0 se x è dispari
  - ecc.
- Un dominio, un codominio, e *un'espressione* che definisce i valori in ogni punto del dominio
  - I punti del dominio - *parametri* della funzione
- Semplice calcolare il loro valore usando espressioni con variabili in JavaScript

Funzioni in matematica - calcolo valori

$f(x) = x^2$

```
let x = 10
console.log(`f(${x})=${x*x}`)
```

```
[LOG]: "f(10)=100"
```

f(x,y)=x+y

```
let x = 10, y = 20
console.log(`f(${x},${y})=${x+y}`)
```

```
[LOG]: "f(10,20)=30"
```

f(x) = 1 se x è pari, 0 se x è dispari

```
let x = 10
console.log(`f(${x})=${ x % 2 == 0 ? 1 : 0}`)
```

```
[LOG]: "f(10)=1"
```

Funzioni in matematica - calcolo valori

- Se invece voglio calcolare la funzione in tanti punti su un dominio?
  - Comandi iterativi - sempre dominio discreto, anche su funzioni su intervalli continui

```
for (let x = -5; x < 5; x++)
    console.log(`f(${x})=${x*x}`)
```

```
[LOG]: "f(-5)=25"
[LOG]: "f(-4)=16"
[LOG]: "f(-3)=9"
[LOG]: "f(-2)=4"
[LOG]: "f(-1)=1"
[LOG]: "f(0)=0"
[LOG]: "f(1)=1"
[LOG]: "f(2)=4"
[LOG]: "f(3)=9"
[LOG]: "f(4)=16"
```

```
for (let x = -5; x < 5; x++)
    console.log(`f(${x})=${ x % 2 == 0 ? 1 : 0 }`)
```

```
[LOG]: "f(-5)=0"
[LOG]: "f(-4)=1"
[LOG]: "f(-3)=0"
[LOG]: "f(-2)=1"
[LOG]: "f(-1)=0"
[LOG]: "f(0)=1"
[LOG]: "f(1)=0"
[LOG]: "f(2)=1"
[LOG]: "f(3)=0"
[LOG]: "f(4)=1"
```

Funzioni in matematica - calcolo valori

- Se invece voglio calcolare la funzione in tanti punti su un dominio?
  - Comandi iterativi - sempre dominio discreto, anche su funzioni su intervalli continui
    - Comandi iterativi annidati - parametri multipli

```
for (let x = -2; x < 2; x++)
    for (let y = -2; y < 2; y++)
        console.log(`f(${x},${y})=${x+y}`)
```

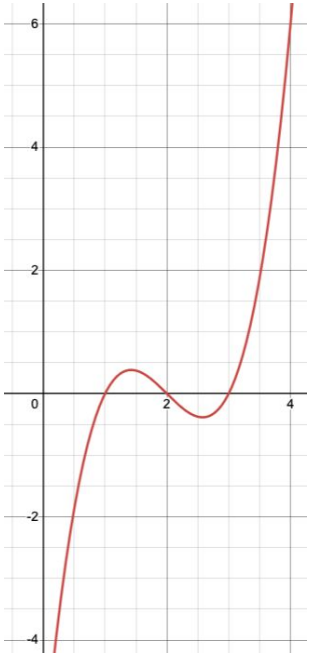
```
[LOG]: "f(-2,-2)=-4"
[LOG]: "f(-2,-1)=-3"
[LOG]: "f(-2,0)=-2"
[LOG]: "f(-2,1)=-1"
[LOG]: "f(-1,-2)=-3"
[LOG]: "f(-1,-1)=-2"
[LOG]: "f(-1,0)=-1"
[LOG]: "f(-1,1)=0"
[LOG]: "f(0,-2)=-2"
[LOG]: "f(0,-1)=-1"
[LOG]: "f(0,0)=0"
[LOG]: "f(0,1)=1"
[LOG]: "f(1,-2)=-1"
[LOG]: "f(1,-1)=0"
[LOG]: "f(1,0)=1"
[LOG]: "f(1,1)=2"
```

Esercizio: Ricerca zeri, minimo, e massimo

Si scriva un programma che analizzi la funzione $f(x) = x^3 - 6x^2 + 11x - 6$ per valori di x compresi tra 1 e 4, a passi di 0.01.

Il programma deve stampare:

- **tutti gli zeri** della funzione nell'intervallo, considerando un valore pari a zero se minore in valore assoluto di `1e-6`,
- il **valore minimo** e il **valore massimo** assunti dalla funzione nell'intervallo.



Apri grafico

E se proviamo un'altra funzione?

Funzioni in Javascript: valori & letterali

Abbiamo già visto come le *funzioni* siano uno dei tipi base di JavaScript

Come tali, possono essere denotate da costanti letterali
(ovvero: è possibile esprimere un valore di tipo funzione scrivendolo nel testo di un programma)

Un valore di tipo funzione è pur sempre un valore
(quindi, può essere usato nelle espressioni, può essere il risultato di una espressione, può essere assegnato a una variabile, inserito in un array, ecc.)

Si può esprimere una costante letterale di tipo funzione in due modi:

1. Con la notazione "freccia" `=>` (anche detta *lambda espressione*)
2. Con la parola chiave `function`

Funzioni: costanti letterali

Alcuni esempi…

Le parti azzurre sono **costanti letterali di tipo funzione**, e dunque denotano dei **valori**, esattamente come 5, 3.1419826 o “pippo”

① `let raddoppia = x => 2*x`

① `var somma = (a,b) => a+b`

② `let prossimopari = (x) => { if (x%2==0) return x+2 else return x+1 }`

① `const cinque = () => 5`

③ `somma = function(a,b) { return a+b }`

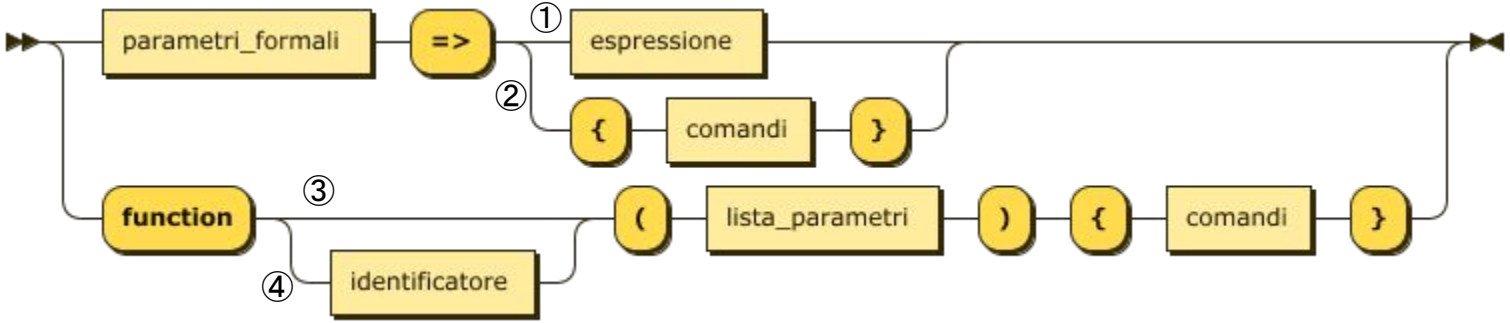
④ `f = function somma(a,b) { return a+b }`

Funzioni: costanti letterali

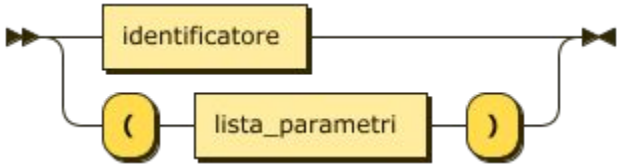
letterale_funzione ::=
parametri_formali '=>' *espressione*
| *parametri_formali* '=>' '{' *comandi* '}'
| 'function' '(' *lista_parametri* ')' '{' *comandi* '}'
| 'function' *identificatore* '(' *lista_parametri* ')' '{' *comandi* '}'

parametri_formali ::= identificatore | '(' *lista_parametri* ')'
lista_parametri ::= | *identificatore* | *identificatore* ',' *lista_parametri*

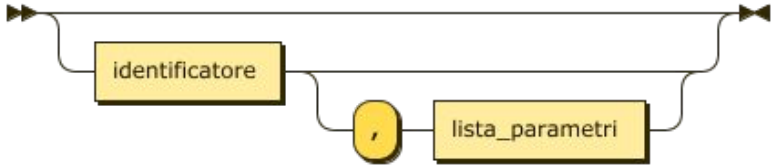
letterale_funzione:



parametri_formali:



lista_parametri:



Sembra complesso ma sono solo due forme (con `=>` e con `function`), ciascuna con due varianti.
① ② ③ ④

Funzioni: dichiarazione

Con una notazione molto comune, si può scrivere

```
function f(a,b) { return a+b }
```

Al posto di

```
var f = (a,b)=> { return a+b }
```

E' molto comune che un programma di grandi dimensioni consti quasi esclusivamente di moltissime dichiarazioni di tipo **function *nome*(...) {...}**, seguite magari da un solo comando che fa "partire" la computazione.

Dichiara una funzione di nome *f* con i parametri formali e il corpo indicati.

Equivalente a dichiarare una variabile di nome *f* e assegnare il valore-funzione corrispondente.

Altri linguaggi consentono solo la forma dichiarazione, e le funzioni non sono valori come gli altri ma "cose a parte" del linguaggio

Funzioni: operazioni

Come per i Number esistono + * - / e simili, e per String esiste +, anche per il tipo Function esistono degli operatori dedicati.

Il più importante si chiama **applicazione** (detta anche **invocazione** o **chiamata**), ed è indicato da una coppia di parentesi posta a destra del valore.

Operatori **infissi**	Operatori **prefissi**	Operatori **postfissi**	Operatori **circonfissi**
2+3	-5	f()	(a)

Cosa accade durante una chiamata

```
let f = (x => 2*x)
```

corpo

parametro formale

parametro attuale
o **argomento**

```
f(3) → 2*3 → 6
```

Il parametro attuale viene sostituito al parametro formale nel corpo della funzione, e l'espressione risultante viene valutata.
Il risultato di questa valutazione è il valore restituito dalla chiamata

Casi strani ma corretti
Non è necessario dare un nome al letterale funzione

```
(x=>2*x) (1+2) → 6
```

Valore di tipo funzione

Operatore di applicazione
(con argomento 3)

Risultato

Cosa accade durante una chiamata

Un altro modo per esprimere il processo di valutazione di una chiamata è il seguente:

Il corpo della funzione viene valutato in un ambiente in cui i nomi dei parametri formali sono associati ai valori dei parametri attuali

Questa definizione si applica sia al caso in cui il corpo sia una espressione, sia a quello in cui il corpo sia un blocco di comandi fra { … }

… tutti i dettagli nel corso di P&A!

Qualche esempio

```
function somma(a, b) {
    return a + b
}

console.log(somma(2, 5))

console.log(somma)

let add = somma

console.log(add(2, 5))
```

1. Il corpo di somma è `{ return a+b }`
2. Viene valutato in un ambiente in cui a=2, b=5
3. Si valuta `return 2+5`
4. Si valuta l'espressione del return, `2+5` → 7
5. Si valuta `return 7`
6. L'esecuzione della chiamata termina con valore 7
7. Si valuta `console.log(7)`

```
[LOG]: 7
```

```
[LOG]: function somma(a, b) {
    return a + b
}
```

```
[LOG]: 7
```

Esercizi

1. Definire una funzione `replace(arr, target, replacement)` che, dato un array arr, ritorni un nuovo array dove tutte le istanze di target sono sostituite da replacement. Non modificare l'array originale.
2. Definire una funzione `contamaggioredi(arr, threshold)` che ritorni il numero di elementi di arr maggiori strettamente di threshold.
3. Dati due vettori x e y ad n componenti, definire `prodotto_scalare(x, y)` che ritorna la somma di `x[i]*y[i]`. Se le lunghezze differiscono, ritorna `undefined`. Formula: `x · y = somma_{i=1..n} x_i * y_i`
4. Definire `clip(arr, threshold, replacement)` che ritorni un nuovo array dove ogni elemento > `threshold` viene sostituito con `replacement`. Se replacement è `undefined`, usare la soglia stessa.
5. Scrivere `norma(v)` che, dato un oggetto v con chiavi x e y, calcoli la norma euclidea `sqrt(x^2 + y^2)` e la aggiunga a v come proprietà 'norma'. Ritorna l'oggetto v.
6. Definire `creaFiltro(threshold)` che ritorna una nuova funzione. La funzione restituita, data una lista di numeri, produce un nuovo array contenente solo i valori <= `threshold`.

Q & A