# ASTRAZIONE (del codice)

↳ ignorare dettagli non rilevanti

↳ identificare proprietà di interesse

→ variabili ≡ astrazione della cella di memoria

→ procedure ≡ astrazione del codice

⇒ All'astrazione si attribuisce un NOME

→ Parametri : permettono di rappresentare con unico codice una funzione dipendente da valori di input
unico codice ≡ più computazioni possibili

→ Corpo : insieme delle istruzioni che vengono eseguite quando si riferisce il nome della procedura

- Una procedura ha una singola entrata
- L'esecuzione del codice chiamante viene sospesa per passare il controllo alla procedura chiamata
- Al termine della procedura il controllo torna sempre al chiamante che riprende l'esecuzione (processo automatico)

---

P procedura →
- Specifica di P (dichiarazione) (interfaccia: parametri e/o risultato restituito e nome)
- Scrittura di P (codice)
- Uso di P (chiamata)

Abbiamo bisogno di strumenti sintattici per fare questi passaggi

parametri formali (tipo e nome)

```
double P (int x) {
    double z;
    /* corpo */
    return zexpr;
}
```

- specifica → informazioni necessarie per chiamare e usare correttamente P
- tipo del val. di ritorno
- z: unità locale definita nella procedura
- corpo della procedura → sono parte dell'ambiente locale di P
- dichiarazione

```
main ...
    double y;
    :
    y = P(3);
```

Uso di P → chiamata

parametri attuali

```
main
    :
    int x
    :
    P(x+1);
    Q(—)
```

```
void P (int y) {
    :
    z = x + y;
}
```

x non definita dentro P ⇒ Quale valore usare per x?

Regole di visibilità sono necessarie per risolvere i riferimenti non locali (non definiti nella procedura e non parametri formali)

⇒ rimangono le stesse viste: ogni dichiarazione è visibile nel blocco in cui è definita e nei blocchi annidati che non la sovrascrivono

- **Statico**: Annidamento della definizione del blocco
- **Dinamico**: Annidamento della chiamata e quindi dell'esecuzione

⇒ DOVE si trova la dichiarazione

Ambiente riferimento

dich. di P

⇓

regola di scoping statico

⇒ DOVE si trova la chiamata

Ambiente riferimento

Chiamata di P

⇓

regola di scoping dinamico

⇒ Regole per l'ambiente di riferimento

Regole di scope + Passaggio di parametri + Regole di binding

## Regole di scope

```
{ int x = 10;
  void foo () {
    x++;
  }
  void fie() {
    int x = 0;
    foo();
  }
  fie();
}
```

Ambiente che contiene la definizione (statico)

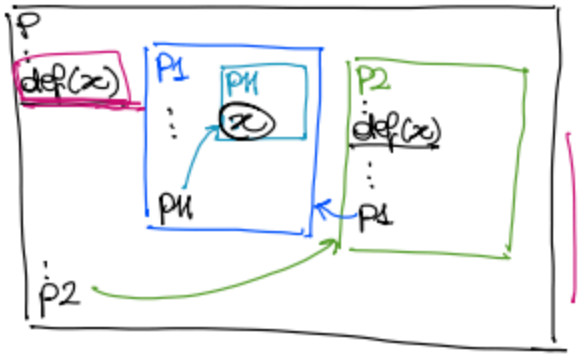
l'ambiente non dipende dalla chiam.

x non locale dove trova specificato?

Ambiente che contiene la chiamata (dinamico)

L'ambiente di riferimento combia (potenzialmente) per ogni chiamata

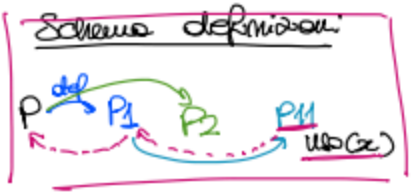
# Scoping Statico




Sequenza chiamate

p → P2 → P1 → P11

Schema definizioni




catena statica

```
{ int x=0;
  void pippo (int m){
    x = m+1;
  }
  pippo (3);
  write (x);
  {int x=0;
   pippo (2);
   write (x);
  }
  write(x);
}
```

x non locale

⇒ scoping statico ovvero x dentro pippo fa sempre riferimento alla x globale

modifico la x globale che ora vale 4

4 ←

assegno 3 alla x globale

→ 0

3 ←

Nello scoping statico l'ambiente di riferimento è indipendente dalla posizione della chiamata

NO:

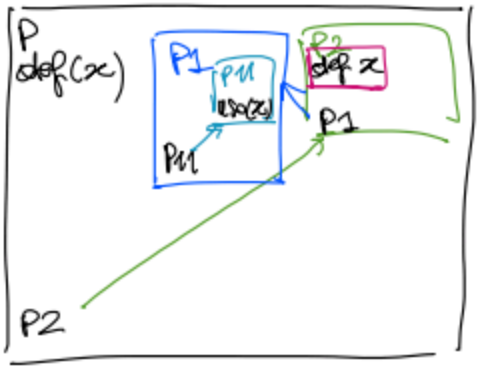
```
x=1;
int x
```

SI:

```
{void f(){
  g();
f();
```

4 5

## SCOPING DINAMICO




Sequenza di chiamate

P → P2 → P1 → P11

def x — uso (x)

catena dinamica

```
{int x=0;
 Void pippo (int n){
      x = n+1;
 }
 pippo (3);        porto la x globale a 4
 write(x);  ← 4
      {int x=0;
      pippo (2);   porto la x locale a 3
      write (x); → 3
      }
 write(x);  ← 4
```