PROCEDURE:

ASTRAZIONE DEL CODICE: (Variabili lo Sono Già)

- Ignorare dettagli NON rilevanti
- Identificare PROPRIETÀ D'INTERESSE

astrazione della Cella di MEMORIA

INTERFACCIA ⟹ Input Richiesto e OUTPUT dato

Anche La Classe è un Astrazione (delle dichiarazioni)

All' Astrazione si da un Nome, per poi Richiamarla

Parametri rendono il CODICE in Grado di EFFETTUARE molte Computazioni diverse.

Permettono di Rappresentare con un UNICO CODICE una funzione dipendente da dei valori di INPUT

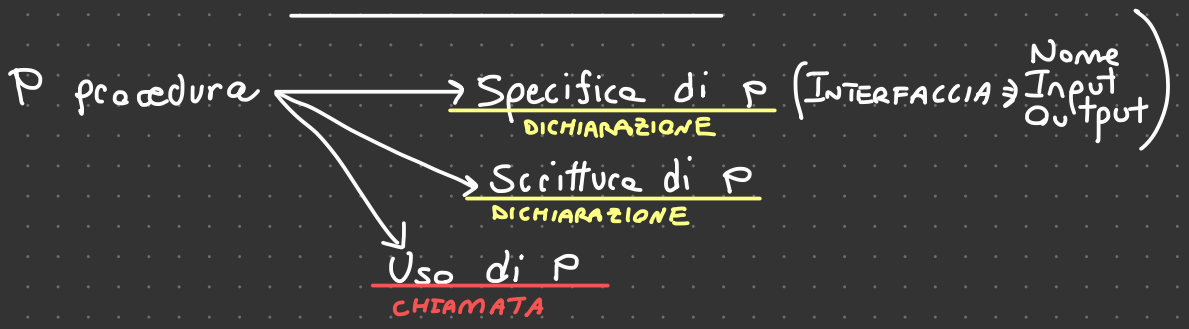
Corpo che è l'INSIEME delle ISTRUZIONI che Vengono eseguite quando si RIFERISCE il NOME della procedura.

È la parte INCAPSULATA Nella PROCEDURA... non Serve a chi La Chiama.

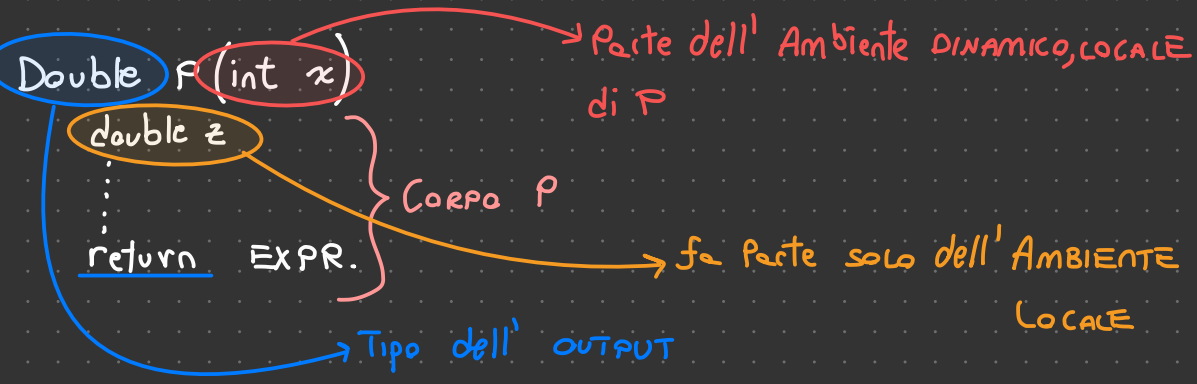
In Genere (a PROCEDURA ha una Sola ENTRATA (una Sola di Attivazione Annidata ma da un Altra parte con le Regole di Visibilità dei BLOCCHI ANNIDATI)

l'EXEC 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'EXEC. (UN PROCESSO AUTOMATICO)



Mi Servono STRUMENTI SINTATTICI per Questi STEP



Interfaccia è: DOUBLE P (int x) che Raccoglie TUTTE le info utili per Sfruttare la Procedura

Uso Di P ⇒ è La chiamata

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

Parametro Attuale

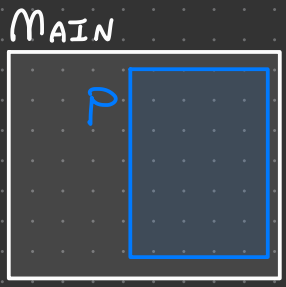
VISIBILITÀ

```
main
int x
P(x+1)
```

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

NON definita DENTRO P

Quale Valore uso per x ?



Questa è la REGOLA di Visibilità, come se fosse un BLOCCO Annidato

Regole di Visibilità Sono NECESSARIE per risolvere i Riferimenti NON Locali ⇒ Rimangono le STESSE: ogni dichiarazione è VISIBILE Nel Blocco in cui è DEFINITA e nei BLOCCHI

annidati che NON la Sovrascrivono

Apparentemente non cambia ma così abbiamo 2 tipi di AMBIENTI

- **STATICO** ⇒ Annidamento della definizione del BLOCCO, Quindi dove si Trova la DICHIARAZIONE

AMBIENTE DI DICHIARAZIONE

- **DINAMICO** ⇒ Annidamento della chiamata e Quindi dell' EXEC del Blocco quindi dove si Trova la chiamata

AMBIENTE DI CHIAMATA

Programmatore deve conoscere Queste 2 Regole di SCOPING (dinamico/Statico)

REGOLE PER DETERMINARE L'AMBIENTE DI RIFERIMENTO

- Di Scope ⟶ Quando l'AMBIENTE NON è LOCALE
- Di Parametri ⟶
- Di Binding ⟶

Regole di Scope:

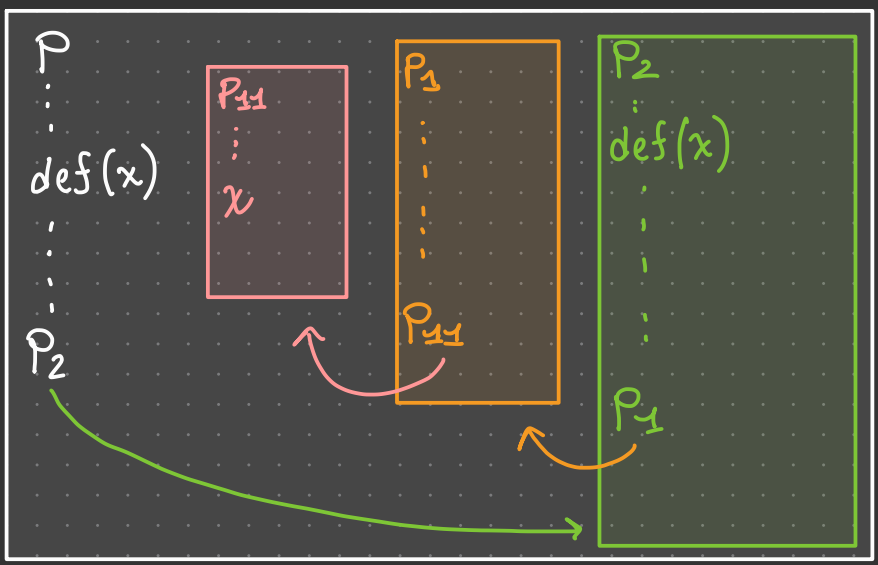
```
{int x = 10
    void foo(){
        x++;
    }
    void fee(){
        int x = 0;
        foo();
    }
    fee();
}
```

x non Locale, Ma dove Trova Significato?
Alla Riga Sopra, dove contiene la definizione che è Statico o Dinamico, dove contiene la Chiamata

→ Regola di Scoping
Mi definisce Quale dei 2 devo Usare

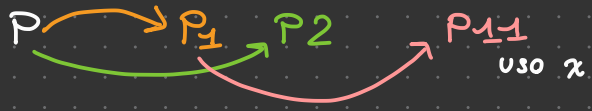
Con Ambiente Dinamico può Potenzialmente CAMBIARE per ogni CHIAMATA, altrimenti è SEMPRE UGUALE visto che NON dipende dalla CHIAMATA

Scoping Statico:



SEQUENZA CHIAMATE: P → P2 → P1 → P11

SCHEMA DEFINIZIONI:



detta CATENA Statica

→ potrei risalire all' INDIETRO da P11 raccogliendo delle INFO a Tempo di Compilazione per Non Sprecare Tempo POI

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

NON LOCALE, fa Sempre Riferimento alla x Globale della Riga Sopra (*)

pippo(3); → Modifica la x Globale (*) a 4

write(x); → Scrive 4 riferendosi a (*)

pippo(3); → Viene Assegnato il val 4 alla Variabile Globale

write(x); → Scrive Ø riferendosi Alla x del Blocco

write(x); → 4

Se fosse Stato un Valore ≠ Allora anche la WRITE di Sopra Sarebbe ≠

Nello SCOPING Statico l'AMBIENTE è INDIPENDENTE dalla Posizione delle Chiamata

NO

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

SI

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

INDIPENDENZA DALLA POSIZIONE DELLA CHIAMATA, È MENO ADATTABILE/FLESSIBILE

Python e Javascript sono degli esempi

Scoping Dinamico:

Stabilisce l'ambiente a partire dalla Seq. di Chiamate:

Sequenza Chiamate: P → P2 → P1 → P11 dove uso x

Che Viene detta Catena dinamica

```
{int * x = 0;
 void pippo(int m){
      x := m+1
 }
 pippo(3); → Porta x Globale * a 4
 write(x); → 4
   {int ^x = 0;
      pippo(2); → Porta la x Locale ^ a 3
      write(x); → 3
   }
write(x); → 4
}
```