Programmazione - Sommario

Tutto sulla programmazione, il pensiero computazionale (? da rielaborare meglio ?)

Paradigmi

Paradigmi di Programmazione

Breve introduzione alla programmazione; paradigmi di programmazione con esempi

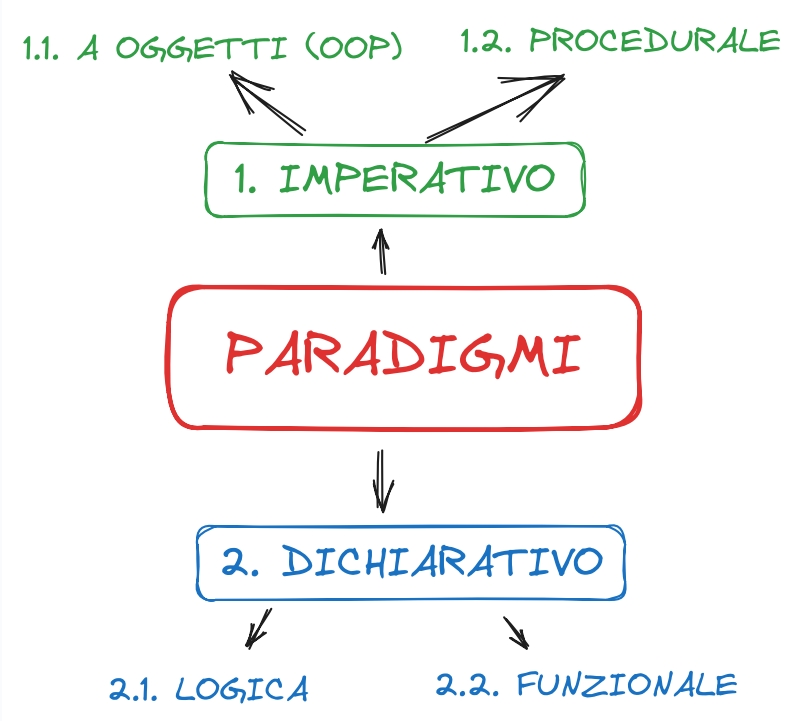
---

Paradigma: cos'è

Nella *programmazione* un **paradigma** (di programmazione) è una *macroarea*, uno *stile* in cui si sviluppa un *linguaggio di programmazione*; nei vari linguaggi di programmazione (soprattutto quelli moderni) si ha molteplici *paradigmi di programmazione*.
Ad esempio in *Python* v'è presente il paradigma *imperativo ad oggetti* con le *classi*, e anche il *paradigma dichiarativo funzionale* con la funzione *lambda*.

Quali sono? Quanti sono?

Generalmente si hanno i seguenti *paradigmi* rappresentati nel diagramma sottostante:



Principalmente i paradigmi sono le seguenti due:

1. **Paradigma IMPERATIVO**
   Il paradigma *imperativo* pone l'enfasi sullo *specificare* le istruzioni al fine di ottenere un risultato voluto. Un esempio di *paradigma imperativo procedurale* è il linguaggio *C*, oppure un esempio di *paradigma imperativo a oggetti (OOP)* è *Java*.

2. **Paradigma DICHIARATIVO**
   In questo caso il paradigma *dichiarativo* esprime la *logica* di un calcolo senza dover descrivere il flusso di controllo. Per esempio nel in un linguaggio dichiarativo *logico* si usa, appunto, la logica per rappresentare e/o elaborare delle informazioni, oppure con la programmazione *funzionale* si usa una serie di valutazioni matematiche.

Differenza tra imperativo e dichiarativo: esempi

La differenza tra il paradigma **IMPERATIVO** e **DICHIARATIVO** si illustra mediante il seguente esempio; due *"pseudocodici"* che rappresentano, da un lato il paradigma *imperativo*, e dall'altro il paradigma *dichiarativo*. Entrambi vogliono esprimere l'utilizzo di un ascensore.

IMPERATIVO	DICHIARATIVO
- ATTESA	- SE ARRIVATO e APERTO, ENTRA
- APRI	- SE PUOI ENTRARE, DEVI ASPETTARE ARRIVI
- CHIUDI	- ...
- BOTTONE	
- ...	

A sinistra si può vedere che impongo *una serie di istruzioni*, come quello di attendere, aprire, chiudere, et cetera ...; invece a destra impongo una *struttura logica*, per esempio SE l'ascensore è ARRIVATO e APERTO, allora posso entrare.

Un'altra analogia potrebbe essere quella di una *ricetta di cucina*, che solitamente esprime una serie di istruzioni (pertanto usa una struttura *imperativa*), come *"cucina per 10 minuti"*, *"sbatti le uova"* e via così ...
Invece se si vuole, per qualche motivo, scrivere una ricetta mediante il paradigma *dichiarativo*, allora si troverebbe scritto qualcosa del genere di *"se l'acqua bolle a 100 gradi °C, allora la pasta è pronta"*.

Nozioni fondamentali

Nozioni Fondamentali di Programmazione

Elenco di nozioni fondamentali di programmazione: programma, algoritmo, input/output, variabile, stato di programma. Assegnamento, sintassi.

NOZIONE 1. PROGRAMMA

PROGRAMMA: Un programma è una descrizione *eseguibile da un calcolatore* di un metodo (*algoritmo*) per il calcolo di un risultato voluto (*output*) a partire da un *input*.

CHIARIMENTI SU ALCUNI TERMINI

Ora vediamo di analizzare alcune parole sottolineate per poter comprendere i concetti;

- *Eseguibile da un calcolatore*; ciò vuol dire che esiste qualcosa, ovvero un *calcolatore* (come un *PC*) che può eseguire il programma.
  Un'analogia per illustrare questo concetto è quello del *DNA* e delle *proteine*; il *DNA* contiene il codice genetico, come il programma contiene la *descrizione di un algoritmo*; e le *proteine* trascrivono il codice genetico dal *DNA*, come il *calcolatore* esegue *l'algoritmo* del programma.
- *Algoritmo*; dal nome d'origine *al-Khuwārizmī*, è un procedimento che serve per fare un *calcolo preciso*. Quindi è una *serie di operazioni finite* e il numero di passi o calcoli o operazioni necessarie per ottenere l'output viene intuitivamente definita come *complessità*.
- *Input*: I dati, le variabili, le informazioni inserite.
  - **OSS.** Quando non c'è nessuna informazione o nessun dato inserito, allora si dice che l'input è *vuoto*.

ANALOGIA CON FUNZIONE MATEMATICA

Il concetto del *programma* è intuitivamente analoga al concetto della funzione nella matematica; ovvero

$$f(x) = y$$

ogni termine in quell'espressione equivale a:

- $f()$ = l'algoritmo
- x = l'input

- y = l'output
  ESEMPIO. Ad esempio, se si ha $f(x) = log(x) + 1$ e si inserisce $x = 10$, allora $log() + 1$ sarebbe l'algoritmo, 10 sarebbe l'input e 2 sarebbe l'output.

MOLTEPLICITA' DI PROGRAMMI

Normalmente, in una macchina molti programmi coesistono; infatti oggi si può addirittura parlare di migliaia di programmi in un PC moderno.
Ciò vuol dire che devono condividere uno *spazio di memoria*, ovvero la *RAM*; in questo caso si parla di *memoria virtuale*.

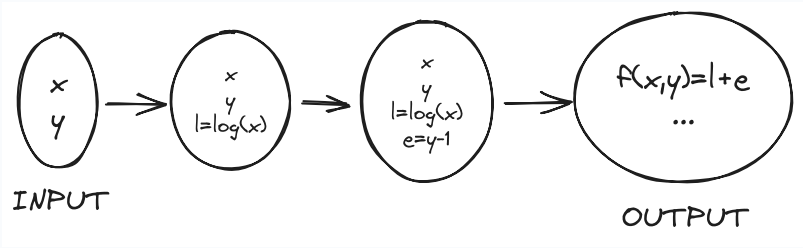
NOZIONE 2. VARIABILE

VARIABILE: Una variabile è un nome associato ad un valore, *modificabile*

NOZIONE 3. STATO

STATO: Lo stato di un *programma* è un insieme di *variabili* che rappresentano la *quantità d'interesse per il programma*.

Per rappresentare graficamente *lo stato interno* di un programma, si può usare dei *cerchi* in cui all'interno si inseriscono le *variabili*; quindi lo nello *stato iniziale* vi è l'input, invece nello *stato finale* vi è l'output desiderato.
ESEMPIO. Sia $f(x, y) = log(x) + (y - 1)$ un programma. Se voglio rappresentare lo stato interno del programma dallo stato iniziale fino a quello finale, devo fare il seguente:



OSS. Dal grafico sopra osservato possiamo vedere che abbiamo eseguito la cosiddetta *operazione di assegnamento*, che definisce la programmazione imperativa, in quanto si istruisce al calcolatore di assegnare un certo valore ad una certa variabile.

NOZIONE 4. ASSEGNAMENTO (SINTASSI)

Per rappresentare la sintassi di assegnamento si scrive il seguente.

`NOME = EXPR;`

- Notare che alcuni simboli sono *necessari*, ovvero `=` (per distinguere NOME ed EXPR) e `;` (per concludere l'operazione di assegnamento).
- Intuitivamente, il NOME rappresenta la denominazione della variabile;
- EXPR rappresenta tutte quelle combinazioni di simboli che mettono assieme *operatori* (aritmetici o logici), *costanti*, *variabili*, *funzioni*.
  - **OSS: DISAMBIGUIRE LE ESPRESSIONI.** Ogni tanto si nota che delle espressioni possono essere ambigue; per esempio $3 + 4 * 2$ per un calcolatore potrebbe significare due espressioni: o $(3 + 4) * 2$ o $3 + (4 * 2)$. Ovviamente queste due espressioni danni due risultati diversi.
    Allora un calcolatore usa un **albero di sintassi astratta**, che danno delle precise *precedenze* a degli operatori. Ad esempio, in questo caso l'operatore moltiplicazione $*$ ha la precedenza sull'addizione $+$.

ESEMPI VARI

ESEMPIO 1. IL PROBLEMA DELLA MACCHINA

Abbiamo il seguente problema:
"Con 30.000€ voglio coprire il costo dei miei spostamenti in auto svolti nell'arco di un anno."
Vogliamo quindi formalizzare un *ragionamento* preciso per risolvere questo problema.

1. Prima di tutto ragioniamo su ciò che possono essere le *variabili* (nella linea generale, senza dover entrare nei minimi dettagli); quindi suppongo le seguenti variabili/input.
   1. Il costo dell'auto C = 20.000€
   2. Il costo della benzina (prima dei rincari prezzi) $B = 0.2\frac{€}{km}$
   3. La distanza percorsa in un anno $K = 10.000\frac{km}{A}$
      Abbiamo fatto dunque tre assegnamenti; ovvero `C = 20000;`, `B = 0.2;` e `K = 10000;`.
2. Ora congegniamo l'algoritmo per calcolare l'output `TOT = ?;` spento all'anno.
   1. $B * K = 20.000 * 0.2 = 2000\frac{€}{A}$ (Totale spento sulla benzina); `TOT = B*K;`
   2. $C + (B * K) = 20000 + 2000 = 22000€$ (Il totale) `TOT = TOT+C`

Ora, ragionando sullo *stato interno del programma*, si ha il seguente diagramma:

ESEMPIO 2. L'ALGORITMO DI MOLTIPLICAZIONE DEL CONTADINO RUSSO

ALGORITMO. Supponiamo di voler moltiplicare due numeri 146 e 37; per farlo useremo l'*algoritmo del contadino russo*, che consiste nel seguente.

1. Vogliamo calcolare 146×37; costruiamo quindi una tabella dove si posizione 146 a destra e 37 a sinistra; compiliamo man mano la tabella dividendo la colonna sinistra per due (arrotondato per difetto) e moltiplicando la colonna sinistra per due, fino a quando il numero nella colonna sinistra diventa 0.
2. La tabella risulta così:

146	37
73	74
36	148
18	296
9	592
4	1184
2	2368
1	4736

3. Ora eliminiamo le righe, dove a sinistra compaiono i numeri pari. Quindi ora la tabella diventa così:

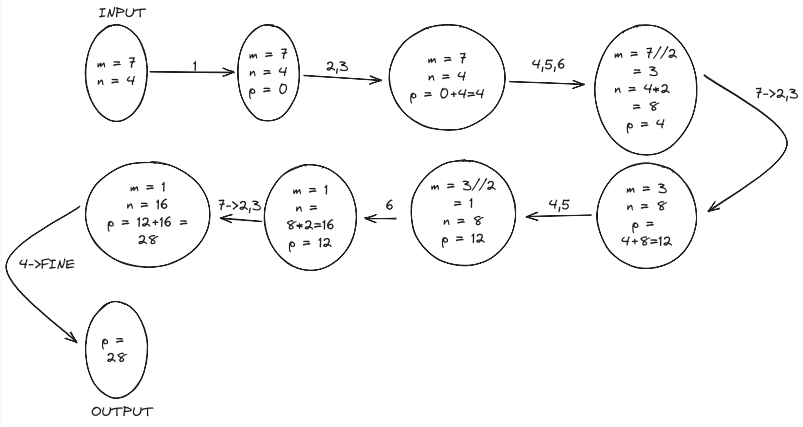
DESTRA	SINISTRA
73	74
9	592
1	4736

4. Ora per ottenere il risultato $p = 146 \times 37$ si sommano tutti i numeri sulla colonna sinistra, ovvero $p = 146 \times 37 = 74 + 592 + 4796 = 5402$.

PROGRAMMA. Ora vogliamo trasformare questo algoritmo in un programma, con il seguente *pseudocodice*.

```
1. CREA p
2. SE m è PARI, SALTA A (5)
3. ASSEGNA p = p+n
4. SE m=1, FINE
5. ASSEGNA m = m//2
6. ASSEGNA n = n*2
7. SALTA A (2)
```

ESERCIZIO-ESEMPIO. Con il seguente programma, disegnare lo *stato interno del programma* quando abbiamo gli input `m = 7; n =4;`.



NOZIONE 5. AMBIENTE E MEMORIA

Riprendiamo la **NOZIONE 3.**, in quanto tutto ciò che abbiamo detto in precedenza non è totalmente accurata; infatti bisogna introdurre le nozioni di *ambiente* e *memoria*.

NOZIONE 6. OPERAZIONE DI DICHIARAZIONE DI VARIABILE

Controllo del flusso di esecuzione

Controlli del flusso di esecuzione

Istruzioni che servono per controllare il flusso di esecuzione di un programma; istruzione condizionale if-else, espressioni logiche; istruzioni iterative while, for; blocchi di codice e pile di frame

---