DOPO AVERLO LETTO SARÀ FACILISSIMO
IMPARARE QUALSIASI LINGUAGGIO

• • •

IMPARARE

A PROGRAMMARE

• • •

LORENZO FOTI

Lorenzo Foti

IMPARARE
A PROGRAMMARE

Introduzione

Questo libro nasce dalla convinzione che la programmazione sia ormai accessibile a chiunque vi si voglia avvicinare.

Oggi viene troppo spesso vista come complicata perché, a mio parere, viene affrontata subito in modo troppo tecnico. Esistono tantissimi manuali, fatti anche molto bene, che sono però focalizzati sui singoli linguaggi di programmazione e non sulla logica in generale. Linguaggi che poi, come ti mostrerò in questo libro, sono in tantissimi casi veramente molto simili tra loro.

Questo libro è allora rivolto a tutte quelle persone che hanno provato a imparare a programmare, ma che si sono bloccate di fronte a concetti che sembravano troppo complessi.

È rivolto inoltre a chi non ha mai studiato nulla di programmazione e non sa bene da dove cominciare.

Infine è per quelle persone che magari non vorranno mai programmare, ma che sono curiose di capire come funzionano gli oggetti elettronici che sono ovunque intorno a noi.

Nel libro si parte da zero spiegando come si costruisce un programma e, un passo alla volta, capirai quali possono essere le difficoltà e come si possono risolvere.

Troverai esempi nei vari linguaggi di quanto spiegato e scoprirai come questi non siano lingue astruse e incomprensibili, ma qualcosa di molto logico che serve a dare una forma alle istruzioni che vogliamo dare al computer.

Imparerai quindi quali sono i mattoncini che costituiscono un programma e come, grazie a questi, sia possibile esprimere qualsiasi comando vogliamo dare a un computer.

Andando avanti arriverai a scoprire come i programmatori siano riusciti a organizzare il modo di scrivere le istruzioni in modo da

semplificare la programmazione e condividere tra loro quanto viene fatto.

Infine, nell'ultimo capitolo, troverai dei piccoli approfondimenti su argomenti strettamente collegati alla programmazione e che sono sempre più importanti nel mondo di oggi: banche dati, siti internet, programmare con Google, Facebook, etc. e, infine, la logica dietro la costruzione di un robot con Arduino.

È difficile programmare?

Come scoprirai leggendo questo libro, la risposta è No!

Se si procede nel giusto modo, imparare a programmare può essere facile e divertente. Il problema che si incontra di solito è che dopo 10 minuti da quando si è iniziato a studiare, si viene sommersi da termini tecnici e comandi del linguaggio di programmazione difficili da ricordare.

I manuali di programmazione in genere sorvolano completamente sullo spiegare come bisogna ragionare quando si scrive un programma; passano infatti direttamente a spiegare tutte le caratteristiche particolari del linguaggio di cui parlano.

In questo libro invece procederemo piano piano andando a capire come bisogna costruire un programma e facendo degli esempi usando più linguaggi di programmazione. Ti accorgerai di come in verità, almeno nella maggior parte dei casi, i diversi linguaggi non siano altro che "dialetti" diversi con cui dire la stessa cosa.

Se voglio dire che "mangio una mela" in italiano, inglese o tedesco, userò parole diverse, ma il concetto fondamentale è che sto mangiando una mela. In questo libro ci concentreremo quindi sul come dire al computer: "mangio una mela".

Una volta capito questo sarà molto facile imparare un linguaggio piuttosto che un altro.

Imparerai quindi a cosa serve programmare, come devi pensare mentre lo fai, come organizzare il tuo programma, come scriverlo e

come sfruttare quanto già esiste e non partire da zero.

Imparerai la logica che c'è dietro al Kindle che tieni in mano in questo momento, ai siti web, alle app e ai robot comandati con Arduino. Vedrai che a quel punto imparare un linguaggio non sarà più così difficile, ma sarà semplicemente un capire come un determinato concetto, che già conosci, si deve esprimere in quella particolare lingua.

Nel prossimo paragrafo vedremo perché è importante saper programmare e poi ci addentreremo nel vero e proprio mondo della programmazione.

Perché imparare a programmare?

La prima risposta a questa domanda è: perché l'informatica si continua a diffondere e nel futuro serviranno sempre più programmatori.

Prima i computer erano macchine a sé, poi sono entrati nei telefoni, nei televisori, nelle automobili e stanno cominciando a diffondersi ormai sempre più anche negli elettrodomestici.

Ci sarà quindi sempre più bisogno di persone in grado di scrivere programmi, correggerli e migliorarli.

La seconda risposta è che: sempre più programmi e strumenti permettono agli utenti di programmarli per ottenere più velocemente quello di cui hanno bisogno, senza dover fare tanti passaggi che rallentano enormemente il lavoro. Un esempio più noto sono le macro su excel che permettono di fare in un attimo delle operazioni sui dati che altrimenti richiederebbero ore di lavoro.

Saper programmare ci permette quindi di utilizzare al meglio i programmi in qualsiasi ambito: dalla scrittura, ai fogli di calcolo, ai programmi di disegno, etc.

La terza risposta è forse la meno scontata: capire la logica della programmazione ti aiuta a comprendere il funzionamento di ciò che ti circonda. Come stavo dicendo, i computer sono sempre più diffusi

e nel giro di pochissimo tempo dovremo interagire con uno di essi anche quando faremo partire una lavatrice.

Comprendere allora come è stato programmato l'oggetto che abbiamo davanti, ci permetterà di capire come ragiona e di essere quindi più pronti ad usarlo al meglio e a risolvere i problemi quando si presenteranno.

Sapendo programmare si perde quel senso di smarrimento che si prova ad esempio quando il computer ha un problema; naturalmente non imparerai automaticamente a risolverlo, ma almeno ti renderà più facile capire se sia un problema grave o meno, se vale la pena chiamare un tecnico, etc.

Le basi

I linguaggi di programmazione

I linguaggi di programmazione sono le “lingue” con cui sono scritti i programmi; ne esistono moltissimi, alcuni tra i più famosi che forse avrai sentito sono BASIC, PASCAL, FORTRAN, C, C++, C#, PHP, ASP, .NET, JAVA, JAVASCRIPT, PYTHON, VISUAL BASIC, DELPHI.

Sono centinaia e se sei curioso puoi trovare la lista completa qui su wikipedia:
https://it.wikipedia.org/wiki/Lista_dei_linguaggi_di_programmazione

Tanti dei linguaggi elencati sono molto vecchi e non più utilizzati, ma ciò che è più importante sapere è che, in generale, esistono linguaggi di programmazione diversi a seconda dell’ambito nel quale ci si trova a lavorare.

Se per esempio si ha a che fare con internet, si programmerà principalmente in PHP, .NET o, in casi più complessi, in JAVA. Se invece si ha a che fare con piccoli programmini per velocizzare il lavoro con Word o Excel, si lavorerà in Visual Basic. Se si sta scrivendo un software per Arduino, il cervello attraverso il quale puoi comandare il tuo robot, lo si farà in C e C++.

Un linguaggio sviluppato per un certo ambito ti permetterà di fare facilmente le operazioni utili per quel contesto e non per un altro. Per esempio il PHP, il linguaggio più diffuso per il web, ti permette di creare facilmente una pagina internet, di fare ricerche nelle banche dati, etc.

Quando si usa Arduino, attraverso il C è semplice dire ad esempio a un motore di cominciare a girare, a un led di accendersi o a un altoparlante di emettere un suono. Sarebbe impossibile fare lo stesso col PHP.

È come se ogni linguaggio, oltre alle parole generiche, avesse delle parole speciali per dire cose che gli altri linguaggi non sanno dire. Se quindi dovrò parlare con un robot fatto con Arduino, dovrò parlare con una lingua che sa dire “accendi il motore” e non con una che sa dire “scrivimi questo sulla pagina internet”.

Tutti i linguaggi moderni hanno però la stessa logica di base che è quella di cui parleremo in questo libro.

Per rendere ancora più evidente come questo sia vero, ogni esempio lo troverai in due o più linguaggi diversi; dopo un po’ ti renderai conto di quanto può essere semplice, una volta capita la logica, passare dall’uno all’altro.

Cominciamo ora ad addentrarci nella programmazione vera e propria.

Imparare a dare i comandi

Si scrive un programma per dare dei comandi a un “computer”; questo potrà essere quello che hai sulla scrivania, quello dentro al tuo smartphone, quello dell’automobile, etc.

I comandi sono il modo per dirgli di come si deve comportare in determinate situazioni, ecco alcuni esempi:

Videogioco: quando il giocatore preme la barra spaziatrice, fai fare un salto al personaggio del gioco

Sensore di parcheggio dell’automobile: quando la macchina fa retromarcia e c’è un muro a meno di un metro, dai un avvertimento sonoro

Frigorifero: quando il frigorifero è troppo caldo, fai suonare un allarme.

Sito internet: quando l’utente preme il tasto “cerca”, fagli vedere una pagina internet contenente una tabella con i risultati.

Lavatrice: quando la funzionalità “blocco bambino” è attivata, blocca tutti gli altri tasti.

Smartphone: quando l'utente preme sull'icona della posta, apri la casella di posta elettronica

Come vedi, attraverso un programma non gli stiamo dicendo solamente di fare qualcosa, ma di farlo quando succede qualcos'altro.

L'obiettivo della programmazione è infatti quello di insegnare a un computer come si dovrà comportare di fronte agli eventi esterni.

Chi programma un videogioco può modificarlo fino a quando è sul suo computer, ma una volta che questo è finito, è stato messo su un DVD e viene avviato dal giocatore sul suo computer di casa, il programmatore non potrà più intervenire.

Se il videogioco è stato programmato bene, il computer che lo esegue avrà le istruzioni su come comportarsi in ogni occasione. Saprà quindi cosa fare se il giocatore preme un tasto sulla tastiera, se usa il mouse, il joypad, etc.

Il programma del videogioco conterrà quindi tanti comandi, chiamati spesso istruzioni, di questo tipo:

- quando il giocatore preme la barra spaziatrice, fai fare un salto al personaggio del gioco
- quando il giocatore preme il tasto C, fai spostare a destra di un passo il personaggio del gioco
- quando il giocatore preme il tasto X, fai spostare a sinistra di un passo il personaggio del gioco
- quando il giocatore preme il tasto A, fai spostare in avanti di un passo il personaggio del gioco
- quando il giocatore preme il tasto Z, fai spostare indietro di un passo il personaggio del gioco
- quando il mouse si muove a destra, fai spostare a destra lo sguardo del personaggio del gioco
- quando il mouse si muove a sinistra, fai spostare a sinistra lo sguardo del personaggio del gioco

etc.

Per ogni evento che può accadere, il computer saprà cosa fare.

Tornando ora a vedere gli esempi di comandi che ti ho mostrato all'inizio del paragrafo, ti accorgerai di una cosa: sono stati scritti ad uno specifico livello di dettaglio.

Per quanto riguarda per esempio il comando del frigorifero

quando il frigorifero è troppo caldo, fai suonare un allarme

si vede subito come la stessa istruzione poteva essere scritta in molti modi diversi.

Eccone due di esempio:

- quando la temperatura del frigorifero è maggiore o uguale a 20° C, fai suonare un allarme
- quanto il termostato manda un impulso elettrico, manda ogni secondo all'altoparlante un impulso elettrico lungo mezzo secondo

Come puoi vedere, sia il comando dell'esempio che queste due versioni hanno senso; quello dell'esempio era molto generico, la seconda di queste versioni è invece è molto particolareggiata.

Qual è allora il modo giusto di dare il comando?

Per rispondere prova a pensare: **cos'è che cambia tra le varie versioni del comando?**

Quello che cambia sono le conoscenze che noi diamo per scontato.

Nel primo esempio stiamo dando per scontato che il computer sappia che:

troppo caldo **vuol dire** temperatura maggiore o uguale a 20° C

Sempre nello stesso esempio, e in quello successivo, si da per scontato che:

suonare l'allarme **vuol dire** mandare ogni secondo all'altoparlante un impulso elettrico lungo mezzo secondo

I comandi quindi non sono giusti o sbagliati a priori, ma bisogna usare quelli che potranno essere compresi dal computer.

Lo stesso vale per noi uomini: se diciamo a un bambino piccolo di “fare ciao con la mano”, lui potrà eseguire questa azione solo se sa cosa vuol dire “fare ciao” e se cosa sia la “mano”.

Se conosce una delle due cose, ci guaderà senza sapere cosa fare.

Proviamo ora a scrivere gli stessi esempi di prima, ma a un livello di dettaglio differente (per comodità ripeto anche il comando originale):

Videogioco

Comando originale: quando il giocatore preme la barra spaziatrice, fai fare un salto al personaggio del gioco

Altro dettaglio: quando il giocatore preme la barra spaziatrice, fai fare due passi avanti al personaggio del gioco e contemporaneamente fallo andare prima su e poi giù

Sensore di parcheggio dell’automobile

Comando originale: quando la macchina fa retromarcia e c’è un muro a meno di un metro, dai un avvertimento sonoro

Altro dettaglio:quando le ruote girano all’indietro, se il segnale del sensore a ultrasuoni torna prima di 0,006 secondi, dai un avvertimento sonoro
(0,006 secondi è il tempo che il suono ci mette a fare un metro, a rimbalzare e a tornare indietro).

Sito internet

Comando originale: quando l’utente preme il tasto “cerca”, fagli vedere una tabella con i risultati in una pagina internet.

Altro dettaglio: quando l'utente preme il tasto "cerca", inserisci all'interno della pagina internet una tabella dove in ogni riga riporti le informazioni che hai trovato nel database (database = banca dati)

Lavatrice

Comando originale: quando la funzionalità "blocco bambino" è attivata, blocca tutti gli altri tasti.

Altro dettaglio: quando la funzionalità "blocco bambino" è attivata, non eseguire le istruzioni relative agli altri tasti.

Smartphone

Comando originale: quando l'utente preme sull'icona della posta, apri la casella di posta elettronica

Altro dettaglio: quando l'utente preme sull'icona della posta, apri il programma per la posta elettronica, controlla se ci sono nuovi messaggi e mostra all'utente la pagina della posta in arrivo.

Questi sono solo delle possibili varianti, potremmo scrivere ognuno degli esempi in tantissimi modi diversi.

Quello che è importante allora quando si programma è sapere a che livello di dettaglio è necessario dare le istruzioni al computer.

Come capirlo?

Questo problema viene risolto dai linguaggi di programmazione in quanto loro stessi contengono i comandi di base che verranno capiti dal computer.

In altre parole, i linguaggi di programmazione sono formati da delle istruzioni, al giusto livello di dettaglio, che spiegano al computer come fare le azioni di base.

Per esempio un comando comune a molti linguaggi è: **PRINT**

“Print” è una parola inglese che in italiano vuol dire “stampa”; PRINT viene utilizzato in generale per dire al computer di scrivere qualcosa su un file, ad esempio un file di testo.

Se io voglio quindi scrivere la parola “casa” in un file, scriverò:

    print casa

e non dovrò spiegare al computer cosa deve scrivere nella memoria del mio computer per riuscire a scrivere “casa” su un file, tutto questo è già contenuto all’interno del comando **print**.

Il giusto livello di dettaglio è allora quello del linguaggio di programmazione: se questo mi permette di usare una parola per scrivere su un file, allora userò quella e non avrò bisogno di dare più dettagli al computer.

Torneremo più avanti su questo concetto quando vedremo come insegnare noi, al computer, dei nuovi comandi per fare operazioni più complesse.

Per ora è importante ricordare che, se voglio dare un comando al computer, devo scomporlo in una serie di istruzioni semplici e vedere se il linguaggio di programmazione che sto usando va bene per quel livello di dettaglio.

Dove si scrivono i comandi

Prima di proseguire, in questo brevissimo paragrafo voglio fare un accenno a dove si dovranno scrivere i comandi per il computer.

Ogni linguaggio di programmazione differisce leggermente dagli altri, ma quello che è importante è che vengono scritti in su dei file di testo.

Le istruzioni verranno scritte una sotto l’altra e il computer le eseguirà dall’alto verso il basso.

Quello che segue è ad esempio una parte di programma in PHP. Non ti preoccupare di capirlo, te lo mostro solo per farti vedere il risultato finale:

```
    foreach ($view->result as $row) {

            $record_month_start = date("m", strtotime($row-
      >field_field_campaign_start[0]['raw']['value']));

                    $record_month_end = date("m", strtotime($row-
    >field_field_campaign_end[0]['raw']['value']));

                        $record_year_start = date("Y", strtotime($row-
    >field_field_campaign_start[0]['raw']['value']));

                        $record_year_end = date("Y", strtotime($row-
    >field_field_campaign_end[0]['raw']['value']));

        for ($i = 1; $i <= 12; $i++) {

            $ok_start = 0;

            $ok_end = 0;

            $month = $current_month + $i;

            $year = $current_year-2;

            if ($month > 12) {$month = $month - 12; $year = $year +
    1;}

            if($record_year_start >= $year) {

                    if($record_month_start<=$month) {

                        $ok_start = 1;

                    …           }
```

Potrà sembrarti incomprensibile, ma continuando a leggere arriverai a capire la logica con la quale è costruito.

Riprendiamo ora il discorso principale e cominciamo a vedere i primi comandi.

La principale struttura di controllo: IF

Come abbiamo appena visto, i linguaggi di programmazione ci danno la possibilità di dire al computer come deve comportarsi. Riprendiamo uno dei comandi e analizziamolo più a fondo:

> **Lavatrice**: quando la funzionalità “blocco bambino” è attivata, blocca tutti gli altri tasti.

Questo comando, come anche tutti gli altri mostrati fino a ora, è composto da due parti: nella prima prima si dice quale deve essere la condizione che deve verificarsi, nella seconda invece viene indicato cosa deve fare.

Condizione: quando la funzionalità “blocco bambino” è attivata

Cosa fare: blocca tutti gli altri tasti

La condizione è un elemento fondamentale, se nel comando quella parte non fosse esplicitata, allora non avrebbe senso. Il computer della lavatrice infatti leggerebbe solo “blocca tutti gli altri tasti” e bloccherebbe quindi immediatamente tutti i comandi.

La **condizione** è una così detta **struttura di controllo**, qualcosa cioè che non è un vero e proprio comando, ma che ci permette di dare una struttura logica al programma.

Quando parliamo le usiamo continuamente:

- Prendo l’ombrello **quando** piove
- **Quando** ho fame mangio
- Studio **perché** ho voglia di imparare

In verità, i tre esempi qui sopra e quello sulla lavatrice, possono essere espressi tutti in un’unica forma:

- **Se** piove, **allora** prendo l'ombrello
- **Se** la funzionalità "blocco bambino" è attivata, **allora** blocca tutti gli altri tasti
- **Se** ho fame, **allora** mangio
- **Se** ho voglia di imparare, **allora** studio

Prova a pensare ad altri esempi e ti accorgerai che riuscirai sempre a scomporli nella forma:

Se ……. **allora** ……

La struttura logica "Se – Allora" è la più importante in assoluto ed è infatti prevista in tutti i linguaggi di programmazione. In inglese la parola **Se** si traduce con **If** e **allora** si traduce con **then**.

Ecco come si scrive questa struttura di controllo in alcuni dei linguaggi più importanti:

Visual Basic:

If condizione **Then** cosa fare **End if**

C:

If (condizione) {cosa fare} Nota che in questo caso il then è sottinteso.

PHP:

If (condizione) {cosa fare} Nota che in questo caso il then è sottinteso.

JAVA:

If (condizione) {cosa fare} Nota che in questo caso il then è sottinteso.

Prima di tutto puoi notare come questa struttura di controllo si dica esattamente allo stesso modo in tre dei linguaggi più importanti e utilizzati al mondo:C, PHP e Java.

La seconda cosa che notiamo immediatamente è che il **cosa fare** è racchiuso tra parentesi o, nel caso del Visual Basic, tra le parole **Then** e **End if** (quest'ultimo può essere tradotto come "fine del se").

Questo accade perché il computer deve avere ben chiaro quando iniziano e finiscono le istruzioni che deve eseguire se si verifica la condizione.

Per esempio, se si dovesse scrivere un programma per spiegare a un bambino come si deve preparare la mattina, questo sarebbe del tipo:

> Prendi i vestiti se piove prendi l'ombrello prendi lo zaino

Scritto in questo modo però, non sarebbe chiaro se lo zaino debba essere preso comunque o solo se piove.

Proprio per questo motivo usiamo la punteggiatura:

> Prendi i vestiti**,** se piove prendi l'ombrello**,** prendi lo zaino

La virgola separa le istruzioni tra loro e ci fa capire in questo caso qual è l'unica istruzione che dipende dal fatto che piova o meno.

Quando parliamo facciamo la stessa cosa cambiando il tono di voce.

Programmando, a seconda del linguaggio, facciamo la stessa operazione utilizzando delle parentesi o mettendo delle parole che identificano in modo chiaro l'inizio e la fine delle istruzioni legate all'IF.

In alcuni casi la struttura di controllo **IF** potrebbe aver bisogno di qualche elemento in più; negli esempi precedenti abbiamo detto al computer cosa fare se si verifica qualcosa, ma non gli abbiamo detto come deve comportarsi se questo non si verifica.

Questo perché abbiamo preso tutti casi in cui l'alternativa era il "non fare nulla".

Esplicitiamo per esempio l'esempio dello smartphone mettendolo anche nella forma "Se – Allora"

> **Se** l'utente preme sull'icona della posta **allora** apri la casella di posta elettronica **altrimenti** non fare nulla

La parte da "altrimenti" in poi era sottintesa e non era stata scritta.

Si verifica però spesso il caso in cui c'è bisogno di dire cosa fare. Nel caso del videogioco per esempio:

> **Se** il giocatore preme la barra spaziatrice **allora** fai fare un salto al personaggio del gioco **altrimenti** continua a muoverlo nella direzione in cui stava andando.

In un programma che fa la divisione del numero A per il numero B si dovrebbe scrivere:

> **Se** B è uguale a zero **allora** rispondi che "non si può dividere un numero per zero" **altrimenti** fai la divisione tra A e B.

La parola **altrimenti** in inglese si traduce con la parola **else**. Nei linguaggi di prima le strutture si modificano quindi così:

Visual Basic:

If condizione **Then** cosa fare **Else** cos'altro fare **End if**

C:

If (condizione) {cosa fare} **else** {cos'altro fare}

PHP:

If (condizione) {cosa fare} **else** {cos'altro fare}

JAVA:

If (condizione) {cosa fare} **else** {cos'altro fare}

Per lo stesso motivo di prima, anche le istruzioni dopo **else** sono racchiuse tra parentesi.

So che ti sembrerà strano, ma con quanto fatto fino a questo punto sei già molto avanti nella programmazione!

Continuiamo ora con la seconda importante struttura di controllo.

La seconda struttura di controllo: il ciclo

Una seconda importantissima struttura di controllo è quella dei cicli; un ciclo non è altro che una serie di comandi che deve essere ripetuta per un certo numero di volte o fino a quando è verificata una certa condizione.

Nei due paragrafi che seguono ti mostrerò questi due casi.

Ciclo con condizione

Prendiamo l'esempio della macchina che fa retromarcia:

> **se** mentre la macchina fa retromarcia c'è un muro a meno di un metro **allora** dai un avvertimento sonoro

Se ci pensi bene qui è stato sottinteso un "ciclo"; noi infatti ci aspettiamo che il sensore di parcheggio faccia continuamente il controllo della distanza da eventuali muri e non lo esegua solo quando ingraniamo la retromarcia.

Se così non fosse, allora sarebbe totalmente inutile.

Esplicitando quindi tutto il comando, avremmo qualcosa del tipo:

> **ogni** mezzo secondo controlla la distanza dal muro, **se** questo è a meno di un metro, **allora** dai un avvertimento sonoro.

In questo caso il ciclo deve continuare all'infinito, eseguendo la sua azione ogni mezzo secondo, fino a che è inserita la retromarcia.

Il ciclo continua cioè fino a che non si verifica una condizione.

Un altro esempio di questo tipo potrebbe essere quello di un robot che deve prendere un oggetto con una pinza:

> continua a stringere la pinza di 1mm alla volta **fino a quando** non tocchi un oggetto

Questo ciclo farà sì che il robot chiuda la pinza di 1mm alla volta fino a quando non incontrerà un oggetto e si fermerà.

Vediamo come si scrive questo tipo di ciclo nei vari linguaggi.

Visual Basic:

While condizione cosa fare **End While**

C:

While (condizione) {cosa fare}

PHP:

While (condizione) {cosa fare}

JAVA:

While (condizione) {cosa fare}

Prima di tutto vediamo che il ciclo è indicato dalla parola **While** che può essere tradotto in italiano con **Finché**.

Come nel caso della struttura di controllo **IF**, anche in questo caso è ben definito cosa deve essere fatto durante il ciclo attraverso l'uso delle parentesi graffe o della parola **End While**.

Infine si può notare come, di nuovo, la sintassi sia identica per gli ultimi tre linguaggi.

Ciclo per un numero determinato di volte

Nel paragrafo precedente abbiamo visto come è strutturato un ciclo che deve ripetersi fino a che non si verifica una condizione.

In alcuni casi si potrebbero voler fare invece dei cicli per un numero determinato di volte.

Immagina per esempio un robot che riempie degli scatoloni con delle lattine di pomodoro e che, una volta arrivato a 20 di queste, debba cambiare lo scatolone perché quello aperto è stato completamente riempito.

Il programma del robot sarà del tipo:

> **Per 20 volte** prendi una lattina di pomodoro e mettila nello scatolone

Nei linguaggi di programmazione, questo viene scritto in questo modo:

Visual Basic:

For contatore **= 1 to 5** cosa fare **Next**

C:

For (contatore **= 1**; contatore **<= 5**; contatore = contatore **+ 1**) {cosa fare}

PHP:

For (contatore **= 1**; contatore **<= 5**; contatore = contatore **+ 1**) {cosa fare}

JAVA:

For (contatore **= 1**; contatore **<= 5**; contatore = contatore **+ 1**) {cosa fare}

Il ciclo in questo caso è indicato dalla parola **For** che può essere tradotto in italiano con **Per**.

Anche qui il cosa fare è racchiuso tra parentesi o comunque, in Visual Basic, delimitato dalla parola **Next** che può essere tradotta con **Prossimo**.

C'è poi il **contatore** che non è altro che un qualcosa che cambierà di valore tenendo il conto di a quale giro del ciclo ci si trova.

Quando si definisce un ciclo di questo tipo bisogna dire allora al computer il valore iniziale del contatore contatore (**=1**) e a quanto dovrà arrivare (**to 5** oppure **<= 5**).

Oltre come al solito a notare che gli ultimi tre linguaggi hanno la stessa sintassi, puoi vedere come questi abbiano un elemento in più: **contatore = contatore + 1**.

Questa parte della definizione del ciclo serve a dire al computer di quanto deve aumentare il contatore dopo ogni giro; in alcuni casi potrei volere infatti un contatore che non procede di uno in uno, ma di due in due, tre in tre, etc.

Dovrò allora scrivere per esempio: **contatore = contatore + 2**.

Nel comando in Visual basic che ho riportato questa parte manca perché, in quel linguaggio, il computer da per scontato che il contatore cresca di un'unità alla volta.

Arrivati a questo punto sai come utilizzare le due strutture di controllo che sono alla base di ogni programma: **condizioni** (**IF**) e **cicli** (**While** e **for**).

Grazie a queste puoi dare una forma a qualsiasi istruzione complessa; come esercizio immagina di dover spiegare al computer come fare qualcosa e prova a scomporlo in cicli e condizioni, vedrai che sarà sempre possibile farlo.

Nel prossimo paragrafo procediamo andando a vedere su cosa vengono eseguiti i nostri comandi: le variabili.

Le variabili

Quando si scrive un programma, si ha continuamente bisogno di qualcosa per fare riferimento alle cose su cui stiamo lavorando.

Per esempio, se faccio un programma che fa la somma di due numeri, avrò bisogno di qualcosa che mi rappresenti i due numeri e il loro risultato.

Il modo più semplice di farlo è scrivere:

C = **A** + **B**

A, **B** e **C** sono delle lettere che il programma andrà a riempire con i numeri che gli darò io.

Se voglio quindi fare la somma 3 + 2, dovrò dire al computer:

A = 3 e **B = 2**

Lui farà il calcolo e metterà il risultato dentro **C**.

Noi possiamo fare con **C** quello che vogliamo, potremmo ad esempio fare questo programma:

C = A + B

Stampa C sullo schermo

Oppure quest'altro:

C = A + B

D = C + 1

Stampa D sullo schermo

Se diciamo quindi al computer che A = 3 e che B = 2, il primo programma farà vedere sullo schermo il numero 5, il secondo invece farà vedere sullo schermo il numero 6. Se invece uso A = 5 e B = 10, avrò nel primo caso 15 e nel secondo 16.

Lo stesso posso fare per qualsiasi valore di A e B; l'unica cosa importante è quella di inserire per A e B due numeri, due cose cioè che possono essere sommate.

Se do come valore A = pippo e B = topolino, il computer non saprà come sommarli e mi dirà che ho fatto un errore.

Dall'esempio fatto sopra si vede come A, B, C e D sono di fatto delle scatole all'interno delle quali noi e il computer possiamo mettere qualcosa.

Queste scatole vengono chiamate **Variabili**.

Facciamo un altro esempio: per entrare dentro Facebook devi inserire la tua email e la password; una volta entrato, in alto vedi a destra vedi il tuo nome.

Cosa ha fatto il programma che sta dietro Facebook? Ha eseguito queste istruzioni:

Trova nella banca dati se c'è un utente che ha come email e password quelle inserite

Se lo trovi, allora metti il suo nome dentro la variabile **nome_utente**

Scrivi **nome_utente** nella barra in alto

Il programma ha detto quindi al computer di fare prima di tutto una ricerca per vedere se l'utente esiste, poi attraverso una condizione

Se (Se lo trovi) gli ha detto cosa fare: prendere il nome dell'utente e metterlo dentro la **variabile** che il programmatore ha deciso di chiamare **nome_utente**.

A quel punto gli ha detto di mostrare il contenuto della variabile **nome_utente** sullo schermo.

Nota bene: essendo **nome_utente** una variabile, quando diciamo al computer di scrivere nome_utente sullo schermo, quello scriverà il contenuto della variabile (Lorenzo, Marco, Maria, etc.) e non "nome_utente".

Riprendiamo l'esempio del frigorifero:

> Se la temperatura del frigorifero è maggiore o uguale a 20° C, fai suonare un allarme

In questo caso il programma sarà:

> Misura la temperatura
>
> **T** = risultato della misurazione
>
> Se **T** >= 20 allora fai suonare un allarme

Come puoi vedere, in questo caso **T** è una variabile che viene riempita con il risultato della misurazione della temperatura.

Subito dopo averle dato un valore, si è detto al computer di verificare **Se** questo questo sia maggiore o uguale a 20.

Nel caso sia così, **allora** deve far suonare l'allarme.

Nel caso della macchina che fa retromarcia, una variabile che si potrebbe utilizzare è la distanza dal muro. In quello del videogioco, la variabile è il tasto premuto (se tasto_premuto = barra spaziatrice allora…), nel caso dello smartphone è l'icona che è stata premuta (se icona_premuta = posta allora…), etc.

Nota bene che le variabili possiamo chiamarle come vogliamo perché non sono dei comandi che il computer deve eseguire; sono solo scatole dove mettiamo quello che ci serve.

Quando si scrive un programma si potrebbe aver bisogno anche di un numero molto alto di variabili, quindi è importante dargli dei nomi che facciano capire immediatamente cosa contengono.

Se comincio a chiamare le variabili a, b, c, d, etc., dopo pochissimo non ricorderò più cosa contengono e diventerà difficilissimo programmare.

I programmatori usano generalmente uno di questi stili per farlo: **distanza_muro** oppure **distanzaMuro**. Nel primo caso si separano le parole con il carattere "_", nel secondo invece viene messa in maiuscolo l'iniziale di ogni parola ad esclusione della prima.

Ripeto che per il computer è assolutamente indifferente, l'importante è non chiamare la variabile **distanza muro** in quanto per il computer queste sarebbero due variabili diverse: **distanza** e **muro**.

Il computer infatti se vede uno spazio pensa che la variabile sia finita e che la parola che segue sia un comando o una nuova variabile.

Quando si scrive un programma si assegnano continuamente dei valori alle variabili, ci si fanno delle operazioni e infine si utilizzano i risultati per l'obiettivo che ci siamo prefissati.

Dare forma a un programma

Le funzioni

Fino ad ora abbiamo visto che un programma è costruito con strutture di controllo e variabili.

Questi elementi rappresentano lo scheletro del programma e gli oggetti da utilizzare al suo interno; quello che manca ancora però è come dire al computer cosa deve fare.

Negli esempi dei paragrafi precedenti era scritto per esempio:

While (condizione) {**cosa fare**}

In questo paragrafo vediamo proprio cosa scrivere al posto di **cosa fare**.

Naturalmente il **cosa fare** dipenderà dai nostri obiettivi; negli esempi precedenti facevamo suonare l'allarme del frigorifero, dare un avvertimento sonoro alla macchina, aprire il programma di posta elettronica allo smartphone, etc.

Come fare però?

A questo punto ci vengono in aiuto i linguaggi di programmazione; come ti dicevo in uno dei primi paragrafi, i linguaggi di programmazione contengono dei comandi, più o meno basilari, che ci permettono di interagire col computer.

In un esempio precedente ti ho parlato del comando **print** che serve a scrivere qualcosa in un file.

Ecco come si scrive "Casa" in un file nei vari linguaggi:

Visual Basic:

print(var_file, "Casa")

C:

fprintf (var_file, "Casa")

PHP:

fwrite ($var_file, “Casa”)

JAVA:

per capire come si dice in Java c’è bisogno di sapere concetti più avanzati quindi per ora non scendiamo nel dettaglio

---

Confrontando i primi tre linguaggi ci accorgiamo che, al di là del comando che si esprime in modo diverso (print, fprintf, fwrite), per il resto la struttura è identica:

comando (variabile che indica il file, cosa scrivere)

Il comando è seguito quindi da delle parentesi che danno al computer tutte le informazioni per eseguire il comando stesso.

Se infatti vogliamo che il computer scriva qualcosa su un file, dovremmo specificargli sia su quale file scrivere, sia che cosa scrivere.

Le informazioni che gli forniamo vengono chiamate **argomenti della funzione**.

In questo caso quindi, il primo argomento è la variabile che indica il file, il secondo è invece cosa scrivere.

Nota bene, anche **cosa scrivere** poteva essere espresso con una variabile, per esempio in PHP:

$cosa_scrivere=”casa”;

fwrite ($var_file, $cosa_scrivere);

N.B. in PHP tutte le variabili sono indicate col carattere “$”.

Arrivati a questo punto possiamo cominciare a chiamare questo tipo di comandi con il loro nome: **funzioni**.

Le **funzioni** sono quindi dei comandi che, a partire da delle informazioni che vengono loro fornite (argomenti della funzione), danno come risposta un valore che è il risultato della loro elaborazione.

Per rendere ancora più chiaro il concetto di funzione, ne seguono alcune espresse nei diversi linguaggi.

Trovare una parola in una frase

Visual Basic:

InStr (dove_cercare, cosa_cercare)

C:

strstr (dove_cercare, cosa_cercare)

PHP:

strpos ($dove_cercare, $cosa_cercare)

In tutti e tre i casi, le funzioni cercano il contenuto della variabile “cosa_cercare” all’interno del contenuto della variabile “dove_cercare” e danno come risultato un numero che indica la posizione.

Esempio:

strpos (“La mia casa mi piace molto”, “casa”)

da come risultato 7.

La “c” di casa è infatti il settimo carattere della frase.

Sostituire una parola in una frase

Visual Basic:

Replace (dove_cercare, cosa_cercare, cosa_sostituire)

PHP:

str_replace ($cosa_cercare, $cosa_sostituire, $dove_sostituire)

In questo caso non esiste una funzione base in **C** che sa fare questo tipo di operazione.

La variabile “cosa_cercare” contiene la stringa che deve essere sostituita, “cosa_sostituire” la parola che bisogna mettere al posto di quella da sostituire.

“Dove_sostituire“ contiene invece la frase all’interno della quale deve avvenire la ricerca.

Esempio:

str_replace (“rosso”, “verde”, “mi piace molto il rosso”)

da come risultato “mi piace molto il verde”.

Arrotondare un numero decimale

Visual Basic:

Round (numero)

PHP:

round ($numero)

C:

round (numero)

Questa funzione ha la stessa sintassi in tutti e tre i linguaggi e arrotonda la variabile “numero” al numero intero più vicino.

Per esempio:

> Round(5,2)
>
> Da come risultato 5

Come puoi vedere, queste **funzioni** sono molto semplici e fanno operazioni basilari.

Insieme alle strutture di controllo e alle variabili sono i mattoncini base con cui puoi costruire il tuo programma.

Il passo successivo sarà allora quello imparare a scrivere le tue funzioni per far fare al computer operazioni più complesse.

Le tue funzioni

Nel paragrafo precedente abbiamo visto come le funzioni di base permettano di fare operazioni molto semplici.

In questo vedrai come puoi definire le tue funzioni personalizzate e realizzare così il tuo programma.

Da quanto detto prima, una funzione è un comando che a partire dagli argomenti produce un risultato.

Qualcosa quindi di questo tipo:

> Risultato = nome_funzione (argomento_1, argomento_2, …)

Dove gli argomenti possono essere anche molto numerosi.

Tutto questo, nei diversi linguaggi, si esprime in questo modo:

Visual Basic:

> **Function** nome_funzione **(**argomento_1 **As Integer,** argomento_2 **As Integer) As Integer**
>
> Istruzioni della funzione
>
> **End function**

PHP:

> **function** nome_funzione**(**$argomento_1, $argomento_2**)** **{**istruzioni della funzione **}**

C:

> **int** nome_funzione **(int** argomento_1**, int** argomento_2**)** **{**istruzioni della funzione**}**

Prima di tutto notiamo ancora una volta come, anche in questo caso, il C e il PHP utilizzino le parentesi graffe per delimitare qualcosa e come invece in visual basic venga usato un comando specifico: **End function**.

L'altra cosa che si può subito vedere è come sia il visual basic che il C dicano al computer che tipo di risultato verrà dato dalla funzione e che di che tipo devono essere gli argomenti; in questo caso in particolare, gli argomenti sono entrambi dei numeri interi e lo è anche il risultato.

Questa informazione è data attraverso l'uso di **int** e **integer**.

Potevano essere anche delle parole, dei numeri decimali o altro, ma il concetto non cambia; sarebbe solo cambiata la definizione.

Se fossero state ad esempio tutte parole, si sarebbe scritto:

Function nome_funzione **(**argomento_1 **As String,** argomento_2 **As String) As String**

Istruzioni della funzione

End function

In PHP invece non è necessario specificare che tipo di dato è contenuto nelle diverse variabili; come accennato precedentemente però, bisogna dire al computer quali siano le variabili e per farlo si usa il segno **$**.

Scrivendo allora “$argomento_1“ si sta dicendo al computer che argomento_1 è una variabile; se non lo si specifica, il php andrà a cercare una funzione chiamata “argomento_1” e darà errore.

Le istruzioni contenute all’interno di funzione non saranno altro che le funzioni di base o altre funzioni definite sempre da noi.

Se per esempio la nostra funzione deve fare la somma tra due numeri, sarà qualcosa di questo tipo:

Visual Basic:

Function somma_numeri **(**a **As Integer,** b **As Integer) As Integer**

Return a + b

End function

PHP:

function somma_numeri ($a, $b) {

return $a + $b;

}

N.B. in php bisogna mettere il ; alla fine di ogni istruzione

---

C:

```
int somma_numeri (int a, int b) {
    return a+b;
}
```

N.B. in php bisogna mettere il ; alla fine di ogni istruzione

---

Come puoi vedere, l'unica novità è il comando **return** che è quello che dice al computer qual è il risultato della funzione.

Se quindi voglio sapere, per esempio in PHP, quant'è la somma tra 7 e 3, scriverò:

```
$risultato = somma_numeri(7, 3);
```

E la variabile $risultato avrà come valore 10.

Aggiungiamo ora all'esempio una struttura di controllo in modo da vedere come è facile mettere insieme tutto quanto visto fino a ora.

Realizziamo per esempio una funzione che come prima ha due argomenti; se la loro somma è superiore a 10 ne fa la sottrazione, altrimenti ne la somma.

La funzione deve quindi comportarsi in questo modo:

Se a = 1 e b = 5 il risultato è c = a + b

(infatti a+b = 6 ed è quindi minore di 10)

Se invece a = 9 e b = 2 il risultato è c = a – b

(infatti a+b = 11 ed è quindi maggiore di 10)

Vediamo ora come si scrive nei diversi linguaggi:

Visual Basic:

```
Function somma_sottrai (a As Integer, b As Integer) As Integer
    Dim c as Integer
    c = a + b
    If c < 10 Then
        Return a + b
    Else
        Return a - b
    End if
End function
```

PHP:

```
function somma_sottrai ($a, $b) {
    $c = $a + $b;
    if($c < 10) {
        return $a + $b;
    } else {
        return $a - $b;
    }
}
```

C:

```
int somma_sottrai (int a, int b) {
    int c;
```

```
        c = a + b;
        if(c < 10) {
            return a + b;
        } else {
            return a - b;
        }
}
```

In Visual Basic e in C è stata definita la variabile “c” dicendo al computer che avrebbe contenuto un numero intero; in PHP invece non c’è bisogno invece di farlo.

Dopo averla definita gli si è assegnato il valore a + b.

A quel punto, con una semplice condizione IF, si è verificato se la somma dei due numeri fosse minore o maggiore di 10.

A seconda del risultato della verifica, la funzione esegue la somma o la sottrazione dei due numeri secondo quanto indicato dalle specifiche del programma.

Come hai visto, con quello che hai imparato fino a ora è molto semplice costruire semplici programmi.

Avrai anche notato come la differenza tra i linguaggi non sia molto grande una volta che si sia capita la logica che c’è dietro la programmazione.

Qualche altro esempio di funzione

Tornando agli esempi dell’inizio del libro, a questo punto diventa possibile immaginare quale sia il modo in cui possono essere scritti i loro programmi.

Macchina che fa retromarcia

Ipotizziamo che il programma sia scritto in C.

Avremo quindi:

```
while (retromarcia_inserita == true) {
    distanza = misura_distanza();
    if (distanza < 1) {
        suona_allarme();
    }
}
```

Vediamo ora un'istruzione alla volta:

```
while (retromarcia_inserita == true) {
```

Questo è l'inizio di un ciclo che si ripete fino a quando è vera la condizione marcia_indietro == true.

Probabilmente ti stai chiedendo perché c'è scritto == e non =; il motivo è che in quel punto stiamo verificando una condizione.

Stiamo cioè vedendo se la variabile "retromarcia_inserita" sia uguale a "true" oppure no.

Per fare questo confronto si usa il così detto **operatore** "==".

Se avessimo scritto "retromarcia_inserita = true", avremmo detto al computer che il valore della variabile retromarcia_inserita doveva essere uguale a true; gli avremmo quindi detto noi che valore dare e non chiesto a lui di verificare quale sia.

Altri operatori che si usano comunemente sono:

- != diverso
- > maggiore
- < minore
- >= maggiore o uguale
- <= uguale

La seconda novità di questa istruzione è la parola **true**. True in inglese vuol dire “vero” e il suo opposto è **False** che invece vuol dire “falso”.

Quando si programma, una variabile ha di solito come valore un numero, un testo oppure i valori true o false.

In questo caso quindi stiamo chiedendo al computer se la variabile retromarcia_inserita ha come valore **true**.

Forse ti chiederai: se il programma inizia in quel punto, chi ha dato il valore alla variabile?

È una domanda giustissima; quello che ho fatto è immaginare qui solo un pezzo di tutto il programma che gestisce la macchina.

Nel programma completo la variabile retromarcia_inserita verrà impostata su true nel momento in cui il guidatore mette la retromarcia e su false quando la toglie.

Quando è true viene eseguito questo pezzettino di programma che vedi qui.

Ricapitolando, questa prima riga dice al computer di eseguire quello che segue fino a quando la retromarcia è inserita.

La seconda istruzione è:

```
distanza = misura_distanza();
```

Qui abbiamo una variabile, che ho chiamato **distanza**, che è uguale al risultato della funzione **misura_distanza**.

Questa funzione sarà definita da un’altra parte e avrà a che fare probabilmente con un sensore a ultrasuoni che misura la distanza.

Per ora quello che ci interessa è che è una funzione che, una volta chiamata, ci da come risultato la distanza in metri.

Come vedi, una funzione come questa non ha bisogno di argomenti; non deve infatti fare delle operazioni su qualcosa che gli diciamo noi,

ma deve solo misurare una distanza.

Le parentesi però si mettono lo stesso perché così il computer capisce che è una funzione e non una variabile.

Se avessi scritto: distanza = misura_distanza, il computer avrebbe pensato di dover dare alla variabile “distanza” lo stesso valore della variabile “misura_distanza”.

Non avrebbe infatti potuto capire che la seconda è una funzione non una variabile.

Eseguita quindi questa istruzione, avremo la variabile “distanza” che contiene il valore della distanza in metri.

La riga successiva è:

```
if (distanza < 1) {
```

Questa è una semplice condizione che verifica se la distanza della macchina dal muro è inferiore a un metro. Se è vera, esegue quello che segue nelle parentesi.

In questo caso, la riga successiva è:

```
suona_allarme();
```

Anche in questo caso vale quanto abbiamo detto prima per la funzione “misura_distanza”: anche suona_allarme non ha bisogno di argomenti, ma si esprime con le parentesi per non confonderla con una variabile.

In questo caso verrà chiamata una funzione, definita da un’altra parte del programma macchina, che ha come effetto quello di far suonare un allarme.

Se la distanza è superiore invece a un metro, allora il computer non fa niente e ricomincia il ciclo da capo.

Allarme frigorifero per temperatura troppo alta

Nell'esempio avevamo un frigorifero che faceva suonare l'allarme se la temperatura superava i 20°C.

Come forse immaginerai, il programma in questo caso è praticamente identico a quello appena visto per la macchina che fa retromarcia.

L'unica differenza è che il frigorifero deve controllare sempre la temperatura e non solo quando si verifica qualcosa di particolare come, per l'esempio di prima, l'inserimento della retromarcia.

Il programma sarà:

```
while (frigorifero_acceso == true) {
    temperatura = misura_temperatura();
    if (temperatura >= 20) {
        suona_allarme();
    }
}
```

Come vedi le istruzioni sono identiche alle precedenti.

Per far sì che il ciclo continui senza sosta ho inserito la variabile frigorifero_acceso che viene impostata uguale a true appena questo si accende.

Salto nel videogioco

In questo esempio dobbiamo dire al computer di far fare un salto al personaggio del gioco quando il giocatore preme la barra spaziatrice.

Qui ipotizziamo che ci sia la variabile "tasto_premuto" che viene riempita con il valore del tasto premuto dal giocatore sulla tastiera.

```
If (tasto_premuto == "barra spaziatrice") {
    muovi_personaggio("salto");
```

```
}
```

In questo caso ho ipotizzato l'esistenza della funzione **muovi_personaggio** che ha come argomento il nome del movimento che deve far fare al personaggio.

Da un'altra parte del programma sarà ci saranno le istruzioni che servono a definirla che saranno qualcosa del tipo:

```
function muovi_personaggio(nome_movimento) {
    if (nome_movimento == "avanti") {
        …
    }
    if (nome_movimento == "indietro") {
        …
    }
    if (nome_movimento == "destra") {
        …
    }
    if (nome_movimento == "salto") {
        …
    }
    …
}
```

I puntini indicano che lì dove sono ci sarà ancora altro codice; non sono riempiti perché a noi in questo momento interessa solo la struttura di questa funzione.

Come vedi dalle istruzioni, il computer controllerà il nome del movimento e poi, grazie a una serie di condizioni IF, troverà qual è il codice che deve eseguire.

Questa funzione, come per esempio quella suona_allarme() degli esempi precedenti, non darà nessun risultato attraverso il comando return, ma eseguirà semplicemente un'azione.

Aprire la posta sullo smartphone

In questo esempio dobbiamo vedere qual è il pezzo di codice per dire al computer che deve aprire la posta elettronica quando viene premuta una determinata icona.

Dal punto di vista logico questa operazione è identica a quella del videogioco. Il computer deve controllare cosa è stato premuto e fare qualcosa di conseguenza.

Sarà quindi:

```
If (icona_premuta == "icona posta elettronica") {
    apri_app("Posta elettronica");
}
```

La funzione apri_programma sarà identica a quella muovi_personaggio dell'esempio precedente. Al posto di tutti i movimenti possibili avrà però tutte le app che sono presenti nel telefono.

Come hai visto in questi esempi, diventa abbastanza semplice programmare se si riescono a separare tra di loro le diverse azioni (misura_distanza, suona_allarme, muovi_personaggio, etc.) e a definire per ognuna di esse una specifica funzione.

Dove mettere le funzioni?

Più volte negli esempi precedenti ho scritto che le funzioni che nominavo erano scritte in altre parti del programma, in questo

paragrafo ti faccio vedere come viene fatta di solito questa operazione.

Quando scriviamo un programma, apriamo un file vuoto e cominciamo a scrivere le istruzioni; dopo poco ci troveremo di fronte alla necessità di scrivere una funzione.

Avremo cioè bisogno di interrompere il “ragionamento” principale del programma e di scrivere da qualche altra parte la funzione necessaria a fare quello che ci serve in quel momento.

La cosa migliore è allora aprire un altro file e scriverci dentro la funzione.

Il secondo file inizierà quindi con qualcosa del tipo:

```
function la_mia_prima_funzione() {

    …

}
```

Più avanti avremo bisogno di una nuova funzione e a quel punto apriremo di nuovo questo file e andremo ad aggiungerla sotto alla prima:

```
function la_mia_prima_funzione() {

    …

}

function la_mia_seconda_funzione(argomento1) {

    …

}
```

Scriveremo una sotto l’altra tutte le funzioni mano a mano che ci servono; non è importante in che ordine sono perché tanto, quando il computer esegue una funzione in particolare, legge solo le istruzioni

che sono al suo interno. Una volta lette, torna al programma principale e ignora tutte le altre funzioni del file.

A questo punto abbiamo il programma principale che sarà qualcosa del tipo:

```
…
a = la_mia_prima_funzione();
…
…
b = la_mia_seconda_funzione(x);
…
…
```

Una serie di istruzioni dove ogni tanto vengono chiamate anche le mie funzioni.

Il problema è: come fa il computer a sapere cosa deve fare quando trova scritto “la_mia_prima_funzione”?

È vero che la funzione è spiegata nel secondo file, ma dobbiamo dire al computer che questo file esiste e che deve usarlo durante l’esecuzione di questo programma.

In altre parole, dobbiamo dire al computer che i due file sono collegati tra loro.

Anche in questo caso è molto semplice, basterà usare quel comando che, in molti linguaggi, si chiama **include** (includi).

Il mio programma principale diventerà quindi:

```
include(file_con_funzioni.txt);
…
a = la_mia_prima_funzione();
```

```
…

…

b = la_mia_seconda_funzione(x);

…

…
```

Abbiamo semplicemente detto al computer di includere, e cioè inserire, il secondo file all'inizio del primo.

A questo punto, quando troverà le mie funzioni, saprà cosa fare in quanto le trova spiegate nel file incluso.

Se ci pensi bene, questo permette di fare qualcosa di molto importante: **riutilizzare le funzioni in diversi programmi**.

Immagina ad esempio di essere il programmatore che ha scritto il programma per la retromarcia della macchina e di trovarti ora a scrivere quello per l'allarme temperatura del frigorifero.

Ti rendi conto immediatamente che anche in questo caso hai bisogno di una funzione che faccia suonare un allarme.

Quello che puoi fare allora è andare a cercare la funzione suona_allarme() nel file collegato al programma principale della retromarcia della macchina e andarla a copiare nel file allegato al programma del frigorifero.

In questo modo non hai bisogno ogni volta di ricominciare da capo, ma potrai riutilizzare quello che hai già fatto.

In verità, potresti fare qualcosa di ancora più intelligente: se in un programma hai tante funzioni che hanno a che fare con lo stesso ambito, potresti fare un file dove definisci solo quelle funzioni lì.

Esempio: scrivendo un programma ti trovi a dover preparare delle funzioni che servono a fare delle operazioni sulle immagini. Le funzioni potrebbero essere ad esempio:

- Ridimensiona
- Trasforma_bianco_nero
- Sfoca
- Sgrana

Sai che nel prossimo lavoro avrai di nuovo bisogno di queste funzioni, ma non di tutte le altre che stai scrivendo per il programma che stai facendo in questo momento.

Quello che puoi fare allora è scrivere queste funzioni in un file a parte che poi potrai riutilizzare.

Il programma che stai scrivendo in questo momento avrà allora all'inizio due include in quanto le funzioni sulle immagini sono sul file che riutilizzi e quelle nuove le andrai a mettere su un altro file ancora:

include(file_con_funzioni.txt);

include(file_con_funzioni_immagini.txt);

Quando comincerai a scrivere il nuovo programma, dove ti servono le funzioni sulle immagini, non dovrai far altro allora che andare a includere sempre lo stesso file all'inizio del programma stesso:

include(file_con_funzioni_immagini.txt);

In questo modo non dovrai nemmeno fare copia e incolla delle funzioni da un file all'altro, ma dovrai semplicemente includere il file.

Un ulteriore vantaggio sta nel fatto che se trovi un errore in una delle funzioni che hai scritto, dovrai allora semplicemente correggerlo nel file "file_con_funzioni_immagini.txt" e automaticamente sarà corretto per tutti i programmi dove l'hai usato.

Se tu avessi fatto copia e incolla delle funzioni, avresti dovuto ricercarle in tutti i programmi e andare a correggerle manualmente una per una.

L’ultima importantissima cosa che voglio farti notare è che a questo punto diventa molto semplice anche **utilizzare funzioni scritte da altri**.

Se le funzioni sulle immagini sono già state preparate da un tuo amico, collega o da qualcuno che le ha poi rese disponibili da scaricare su internet, tu potrai prenderle e inserirle con semplicità nel tuo programma.

Non dovrai far altro che includere il file messo a disposizione dal programmatore all’inizio del file.

…

Abbiamo allora visto come, grazie all’utilizzo delle funzioni è possibile:

- Scrivere programmi in modo semplice e chiaro dividendolo nelle varie azioni
- Riutilizzare pezzi di programma che hai già scritto
- Utilizzare pezzi di programma scritti da altri

Nel prossimo capitolo faremo un’ulteriore passo in più sulla strada della semplificazione del codice e sulla possibilità di riutilizzarlo ogni volta.

Gli oggetti

Gli oggetti sono una grande evoluzione fatta nell'ambito della programmazione.

Inizialmente quando si programmava si scrivevano tutte le istruzioni una sotto l'altra e senza utilizzare funzioni; come ora puoi capire, venivano prodotti programmi lunghissimi e molto articolati.

Con l'utilizzo delle funzioni la programmazione si è semplificata moltissimo e si è potuto cominciare a spezzettare ogni programma in tante parti più semplici da gestire.

L'ultimo passo avanti è stato quello di arrivare a definire i così detti **oggetti**.

Un **oggetto** è un qualcosa che ha delle **proprietà** e dei **metodi**.

Le proprietà sono le caratteristiche dell'oggetto stesso, i metodi sono invece le operazioni che si possono fare sull'oggetto.

Ecco degli esempi per capire meglio:

Oggetto: immagine

Proprietà: larghezza, altezza, risoluzione, numero di colori, file, etc.

Metodi: ridimensiona, ritaglia, cambia_risoluzione, converti_bianco_nero, etc.

Oggetto: risultato di una ricerca su una banca dati

Proprietà: numero di risultati, comando_ricerca, etc.

Metodi: cancella, metti_ordine_alfabetico, metti_ordine_grandezza, vai_alla_fine, vai_all_inizio, etc.

Oggetto: allarme del frigorifero

Proprietà: tonalità, volume, frequenza

Metodi: accendi, spegni, cambia_volume, cambia_tono, cambia_frequenza, etc.

Oggetto: personaggio del videogioco

Proprietà: posizione, stato_salute, oggetti_raccolti, direzione_in_cui_guarda, etc.

Metodi: muovi, cambia_salute, aggiungi_oggetto_raccolto, togli_oggetto_raccolto, muovi_sguardo, etc.

L'oggetto in informatica è un'entità astratta che può far riferimento a qualsiasi cosa abbia delle caratteristiche e sulla quale si possa fare qualche azione.

Come hai visto si possono riferire sia agli oggetti fisici, come l'allarme del frigorifero, sia a cose immateriali come il risultato di una ricerca.

Una volta che ho definito le caratteristiche di una famiglia di oggetti, a quel punto posso cominciare a usarli nel mio programma. Prendiamo per esempio l'oggetto immagine, in PHP per ridimensionare l'immagine e farla diventare la metà in larghezza e in altezza, potrei fare:

```
$mia_immagine = new immagine;

$larghezza = $mia_immagine.larghezza;

$altezza = $mia_immagine.altezza;

$mia_immagine.ridimensiona($larghezza/2, $altezza/2);
```

Nella prima riga ho detto al computer che $mia_immagine è una copia dell'oggetto astratto "immagine"; è cioè un qualcosa che ha le

stesse caratteristiche, quindi metodi e proprietà, dell'oggetto immagine. In informatica questo è chiamato **istanza** dell'oggetto.

Una volta che il computer sa che $mia_immagine è un oggetto di tipo immagine, sa anche quali sono le proprietà e i metodi per quell'oggetto.

Nella seconda riga gli si chiede quindi di prendere il valore della **proprietà larghezza** e di metterlo all'interno della variabile $larghezza.

Nell'ultima riga infine si dice al computer di applicare il **metodo ridimensiona** e gli si passano la nuova larghezza e la nuova altezza che, per quanto detto all'inizio dell'esempio, sono state impostate alla metà di quelle iniziali.

Nei prossimi paragrafi vedrai altri esempi, ma è importante notare intanto come grazie agli oggetti sia possibile strutturare un programma in modo diverso e più semplice.

Come programmatore non devi più pensare in termini di funzioni che fanno cose, ma di oggetti che interagiscono.

Questo permette di separare completamente le complessità legate alla programmazione.

Immagina ad esempio il dover scrivere un programma per gestire un'automobile; sembra un compito difficilissimo in quanto la macchina è un qualcosa di molto complesso e che può fare tante cose.

Pensa però a suddividerla in **oggetti**:

- automobile nel suo insieme
- volante
- luci
- frecce
- cambio
- freni
- motore

- cruscotto
- display
- sensore olio
- sensore acqua

etc.

è facile immaginare cosa possa fare ognuno di questi oggetti, eccone ad esempio qualcuno:

Volante

Proprietà: verso di rotazione, blocco_sterzo, posizione

Metodi: cambia_stato_blocco_sterzo, leggi_posizione

Luci

Proprietà: stato_acceso_spento

Metodi: accendi, spegni

Acceleratore

Proprietà: posizione pedale

Metodi: accelera, decelera

Motore

Proprietà: numero_di_giri

Metodi: aumenta_giri, diminuisci_giri

Come vedi la complessità si va riducendo moltissimo, ragionare per ogni singolo oggetto permette di non pensare contemporaneamente

a tutti gli altri elementi di un sistema.

Se mi concentro quindi ad esempio sul volante, dovrò scrivere solamente i due metodi “cambia_stato_blocco_sterzo” e “leggi_posizione”.

Il primo ad esempio sarà fatto così

cambia_stato_blocco_sterzo

```
if (mettere_blocco_sterzo == true) {
        blocco_sterzo = true;
} else {
        blocco_sterzo = false;
}
```

Dove “mettere_blocco_sterzo” sarà una variabile passata dal programma principale.

Il secondo metodo invece, “leggi_posizione“, interagirà con le parti meccaniche della macchina e avrà come risultato la posizione del volante. Questo tipo di interazioni le vedremo più avanti quando parleremo di Arduino.

Come hai visto, il metodo per il blocco sterzo è molto semplice da immaginare dal punto di vista logico e lo stesso avverrà per tutti gli altri metodi.

Una volta preparati tutti gli oggetti avremo già fatto gran parte del lavoro; il passo successivo sarà quello è di preparare il programma principale dove questi interagiscono.

Quando scriverò il programma principale, sarò molto facilitato perché avrò a disposizione tanti pezzi completamente separati dal punto di vista logico e delle funzioni da cui sono formati.

Nei prossimi paragrafi troverai altri esempi e tutto questo diventerà via via più chiaro.

Usare l'oggetto allarme_frigorifero

Ipotizziamo di voler fare il programma che gestisce l'accensione dell'allarme del frigorifero.

Nei paragrafi precedenti lo avevamo fatto con le funzioni, qui lo facciamo invece ragionando per oggetti.

Vogliamo però dire al computer che deve eseguire un compito leggermente più complesso: se la temperatura è sopra i 10 gradi deve dare un allarme a un certo volume, se invece sale sopra i 20, allora il volume deve essere maggiore.

Proviamo a scriverlo come sarebbe in C usando l'oggetto allarme_frigorifero:

```
allarme_frigorifero mio_allarme;
t = mio_termostato.misura_temperatura();
    if (t > 20) {
    if(t > 20) {
        mio_allarme.cambia_volume(200);
    } else {
        mio_allarme.cambia_volume(100);
    }
    mio_ allarme.accendi();
} else {
    mio_allarme.spegni();
}
```

Vediamo ora il codice spiegato riga per riga:

```
allarme_frigorifero mio_allarme;
```

In questa prima riga viene definita un'**istanza** dell'oggetto "allarme_frigorifero".

L'istanza si chiamerà mio_allarme e avrà tutte i metodi e le proprietà dell'oggetto allarme_frigorifero.

```
t = mio_termostato.misura_temperatura();
```

Qui viene misurata la temperatura utilizzando un'istanza dell'oggetto termostato che andrò a definire da un'altra parte.

Nota bene: io potrei anche non aver ancora programmato questo oggetto, ma mentre sto facendo questo programma non mi importa; cosa deve fare dal punto di vista logico è molto semplice, quindi mi dovrò solo annotare di preparare l'oggetto termostato e il un metodo "misura_temperatura".

```
if (t > 10) {
```

In questa riga c'è una semplice condizione che verifica se la temperatura sia superiore ai 10 gradi.

```
if(t > 20) {
```

Questa è un'altra condizione che, trovandosi all'interno di quella precedente, ci permette di prendere i casi in cui la temperatura non solo è superiore ai 10 gradi, ma lo è anche ai 20 gradi.

Se questo accade, il computer continuerà con l'istruzione che seguono

```
mio_allarme.cambia_volume(200);
```

Utilizzo il metodo cambia_volume per portare il volume a 200.

Nota bene, l'allarme ancora non è stato acceso. Gli sto dicendo però che il suo volume dovrà essere impostato a 200 nel momento in cui lo accenderò.

Stessa cosa faccio poco più avanti nel codice dove, se non è verificata la condizione dei 20 gradi, ma lo è quella dei 10, dico:

```
mio_allarme.cambia_volume(100);
```

Arrivato a questo punto del programma, il computer avrà impostato il volume al valore giusto in funzione della temperatura.

La riga che segue quindi:

```
mio_allarme.accendi();
```

Sfrutterà il metodo accendi per accendere l'allarme. Questo comincerà a suonare al volume che stato impostato impostato grazie al metodo "cambia_volume".

Come vedi, con pochissime righe di codice molto semplici da leggere e da scrivere, abbiamo gestito completamente l'allarme del frigorifero.

In questo piccolo pezzo di codice abbiamo anche sfruttato due oggetti diversi e li abbiamo fatti interagire tra loro.

Ricerca su un sito internet con gli oggetti

In questo caso useremo un oggetto che chiamiamo **ricerca** e un oggetto che chiamiamo **tabella**.

Il primo serve a fare le ricerche e a gestire i risultati, il secondo invece a far apparire delle tabelle sullo schermo.

Il programma principale dovrà quindi effettuare la ricerca e mostrare questi dati in forma di tabella.

Essendo un programma per internet, il codice è scritto in PHP:

```
$mia_ricerca = new ricerca;

$mia_tabella = new tabella;

$mia_ricerca.comando_di_ricerca = …;

$mia_ricerca.esegui();

if ($mia_ricerca.numero_risultati > 0) {
```

```
        for ($i = 1, $i <= $mia_ricerca.numero_risultati, $i = $i + 1)
{
                $mia_tabella.nuova_riga($mia_ricerca.leggi_risultato
($i));
        }
        $mia_tabella.mostra_tabella();
 }
```

Vediamolo ora in dettaglio:

```
$mia_ricerca = new ricerca;
$mia_tabella = new tabella;
```

In queste due righe sono state create le istanze dei due oggetti che abbiamo detto prima e che verranno usati nel programma.

```
$mia_ricerca.comando_di_ricerca = …;
```

Qui viene impostata la ricerca da effettuare. I puntini andranno naturalmente sostituiti con il comando per fare la ricerca che si vuole.

```
$mia_ricerca.esegui();
```

La ricerca viene eseguita sfruttando il metodo esegui.

```
if ($mia_ricerca.numero_risultati > 0) {
```

Si controlla se la ricerca ha prodotto dei risultati andando a guardare la proprietà “numero_risultati”.

Il valore a questa proprietà verrà dato all’interno del metodo “esegui” che si trova alla riga precedente; le sue istruzioni faranno la ricerca, conteranno i risultati e metteranno il risultato del conteggio all’interno della variabile “numero_risultati”.

In questo modo un programmatore, come in questo caso, potrà sapere subito se ci sono stati dei risultati oppure no.

Se il valore della variabile è maggiore di zero, allora vuol dire che è stato trovato qualcosa e si può procedere con la costruzione della tabella, altrimenti il computer non dovrà fare nulla.

```
for ($i = 1, $i <= $mia_ricerca.numero_risultati, $i = $i + 1) {
```

Questa riga è molto importante e serve a far sì che vengano letti tutti risultati della ricerca.

In questo caso è stato utilizzato un **ciclo For**, un ciclo quindi che esegue delle operazioni per un numero determinato di volte.

In questo caso particolare gli stiamo dicendo che il **contatore $i** deve partire da 1 e aumentare di uno in uno ($i = $i + 1) fino ad arrivare al numero dei risultati indicato dalla proprietà "numero_risultati".

Se quindi i risultati sono 5, il contatore comincerà da 1 e proseguirà fino ad arrivare a 5.

```
$mia_tabella.nuova_riga($mia_ricerca.leggi_risultato($i));
```

In questa riga sono stati usati due metodi dei due oggetti. Proviamo a scomporla per capirla meglio:

```
$mia_tabella.nuova_riga(……..);
```

Qui stiamo usando il metodo "nuova_riga" dell'oggetto tabella.

Questo metodo prende come argomento quello che deve inserire nella nuova riga.

L'azione del metodo sarà quindi quella di aggiungere nella tabella una nuova riga che conterrà quello che viene indicato tra parentesi.

Il contenuto in questo caso è il seguente:

```
$mia_ricerca.leggi_risultato($i)
```

Con questa istruzione si sta chiedendo all'oggetto "ricerca" di leggere il valore di uno dei risultati.

Quale risultato leggere lo si indica attraverso il numero che viene passato come argomento al metodo “leggi_risultato”.

Se quindi fosse leggi_risultato(1), leggerà il primo risultato, se fosse leggi_risultato(2), leggerà il secondo e così via.

In questo caso il numero del risultato sarà dato dalla variabile $i che è il contatore del ciclo.

Visto che nella definizione del ciclo abbiamo detto che il contatore deve aumentare di uno in uno fino ad arrivare al valore totale dei risultati, questo assumerà i valori: 1, 2, 3, …, $mia_ricerca.numero_risultati; di conseguenza, questa istruzione cambierà ad ogni giro e diventando:

```
$mia_ricerca.leggi_risultato(1)

$mia_ricerca.leggi_risultato(2)

$mia_ricerca.leggi_risultato(3)

$mia_ricerca.leggi_risultato(…)

$mia_ricerca.leggi_risultato($mia_ricerca.numero_risultati)
```

Ricapitolando, in questo **ciclo for** il computer riempirà una riga della tabella alla volta con il valore dei risultati.

Finito il ciclo si passa all’ultima riga:

```
$mia_tabella.mostra_tabella();
```

Dove non si fa altro che mostrare sullo schermo la tabella costruita all’interno del ciclo grazie al metodo “mostra_tabella” dell’oggetto “mia_tabella”.

C’è un imbroglio? Come definire gli oggetti

Interrompo momentaneamente la spiegazione sugli oggetti perché può essere che a questo punto ti sembri tutto troppo semplice e che ti sia venuto il sospetto che da qualche parte ci sia stato un imbroglio.

In questi esempi con gli oggetti abbiamo usato semplici strutture di controllo, IF e cicli, e per il resto ci siamo affidati a metodi e proprietà di questi “fantomatici” oggetti; grazie allora abbiamo potuto fare con semplicità ogni operazione.

Tutto troppo semplice, sembra quasi che abbiamo nascosto tutte le difficoltà all’interno degli oggetti e che ci siamo poi divertiti a comporre il programma principale facendogli fare ogni volta quello di cui avevamo bisogno.

La risposta è: **sì, è proprio quello che abbiamo fatto.**

Ci siamo **dimenticati** di tutte le difficoltà, le abbiamo chiuse in oggetti separati tra loro e abbiamo scritto il programma principale.

Ma quindi il nostro programma ora funzionerà?

La risposta è: **sì, ma solo quando avremo definito gli oggetti**.

Proviamo allora a vedere come dovrò definire un oggetto semplice che chiamiamo “quattro_operazioni”, un oggetto capace di fare le quattro operazioni tra due numeri.

Potrò quindi chiedergli di fare:

- a+b
- a-b
- a*b
- a/b

Decido che questo oggetto avrà quattro metodi:

- somma
- sottrazione
- moltiplicazione
- divisione

e nessuna proprietà.

(Nota bene: sono io che decido di farlo in questo modo, potrei anche decidere di costruirlo in modo differente e avere quindi un diverso

numero di metodi e proprietà)

Vediamo ora come si definisce questo oggetto per esempio in PHP:

```
class quattro_operazioni
{
    function somma ($a, $b) {
        $risultato = $a + $b;
        return $risultato;
    )
    function sottrazione ($a, $b) {
        $risultato = $a - $b;
        return $risultato;
    )
    function moltiplicazione ($a, $b) {
        $risultato = $a * $b;
        return $risultato;
    )
    function divisione ($a, $b) {
        if($b != 0) {
            $risultato = $a / $b;
            return $risultato;
        }
    )
}
```

Come vedi abbiamo fatto una cosa molto semplice: abbiamo prima di tutto detto al computer che stiamo definendo un oggetto, o come si dice in programmazione, una classe di oggetti attraverso il comando **class**.

All'interno delle parentesi che seguono class, abbiamo poi scritto delle funzioni come quelle che abbiamo visto nel capitolo precedente. Queste funzioni sono i metodi dell'oggetto.

A questo punto quindi, se nel nostro programma scriviamo:

```
$mio_quattro_operazioni = new quattro_operazioni;

$risultato_somma = $mio_quattro_operazioni.somma(3, 2);
```

Avremo che la variabile $risultato avrà come valore 5, quello cioè calcolato dal metodo "somma" che altro non è se non la **funzione somma** contenuta all'interno dell'oggetto **quattro_operazioni**.

In altre parole possiamo vedere a questo punto gli oggetti anche come contenitori ben strutturati di funzioni.

Rimane quindi valido tutto quello che abbiamo visto nei primi capitoli, ma grazie agli oggetti siamo ora in grado di organizzare ancora più facilmente il nostro programma separando e spezzettando tutte le difficoltà.

Un'altra caratteristica degli oggetti: gli eventi

Ora che abbiamo visto cosa sono gli oggetti, ti mostro una loro ultima importante caratteristica: gli **eventi**.

Gli eventi sono un qualcosa che si verifica quando "succede qualcosa" a un oggetto; se per esempio ho un oggetto che rappresenta il bottone "invia" in un modulo online, un evento potrebbe essere il click sul bottone stesso.

Un altro evento potrebbe essere quello "marcia_cambiata" dell'oggetto "cambio" della macchina.

Oppure “premuto_nuovo_tasto” dell’oggetto tastiera, “cambiata_temperatura” per l’oggetto sensore del frigorifero o della temperatura dell’acqua della macchina, “scorrimento_del_dito” per gli smartphone, etc.

Gli eventi sono quindi qualcosa che può accadere in qualsiasi momento e che è indipendente dal programma principale.

Nella programmazione a oggetti si possono associare delle istruzioni a ogni evento; è possibile quindi dire al computer cosa deve fare quando si vefica l’evento.

Per esempio l’evento “marcia_cambiata” potrebbe avere associato un codice tipo il seguente:

```
if (nuova_marcia == 1) {…}

if (nuova_marcia == 2) {…}

if (nuova_marcia == 3) {…}

if (nuova_marcia == 4) {…}

if (nuova_marcia == 5) {…}

if (nuova_marcia == -1) {…}
```

Dove l’ultima condizione è per la retromarcia.

Nel momento in cui la marcia viene cambiata, il computer va a vedere se c’è qualcosa che deve fare.

Ad esempio potrebbe andare a leggere il numero di giri, confrontarlo con una determinata tabella, e segnalare al guidatore se abbia messo una marcia non adatta all’andatura di quel momento (operazione che già molte automobili sanno fare).

L’ultima condizione, quella sulla retromarcia, conterrà al suo interno il codice che hai trovato negli esempi precedenti sul sensore di distanza.

Grazie all'evento il computer saprà quindi quando accenderlo e cominciare a misurare.

Con gli "eventi" è allora possibile suddividere ulteriormente il programma principale mettendo da parte tutte quelle azioni legate a degli eventi su degli oggetti.

L'ereditarietà, un altro vantaggio degli oggetti

Gli oggetti hanno un altro grande vantaggio, permettono di creare delle gerarchie grazie alle quali ogni oggetto può essere "figlio" di un altro oggetto e acquisirne, "ereditarne", i metodi e le proprietà.

Pensiamo ad esempio ai sensori della macchina: c'è il sensore per il livello dell'olio, quello per la pressione dei pneumatici, quello per la temperatura dell'acqua, etc.

Tutti questi sensori hanno delle caratteristiche in comune e altre invece che li contraddistinguono.

Per esempio tutti devono avere la capacità di accendere una spia sul cruscotto e magari di far suonare un allarme acustico.

Ognuno però dovrà però saper leggere i dati da delle sonde di tipo diverso.

Se noi programmassimo questi oggetti, dovremmo ripetere tutte le volte lo stesso codice per quelle funzionalità che hanno in comune.

Per esempio per il metodo accendi_spia; la differenza tra un sensore e l'altro è molto piccola ed è solo legata a quale spia accendere.

Grazie all'ereditarietà degli oggetti è possibile però evitare di ripetere questo lavoro.

Possiamo infatti definire un oggetto che chiamiamo "sensore" e poi i suoi oggetti figli: "sensore_temperatura", "sensore_pressione" e "sensore_ olio".

Tutti gli oggetti figli sapranno fare le stesse cose dell'oggetto padre e in più avranno delle loro funzionalità specifiche.

Avremo quindi:

Metodi sensore: accendere_spia, dare_allarme_acustico

Metodi sensore_olio:

Ereditati dal padre: accendere_spia, dare_allarme_acustico

Specifici: leggere_livello_olio

Metodi sensore_temperatura:

Ereditati dal padre: accendere_spia, dare_allarme_acustico

Specifici: leggere_temperatura_acqua

I metodi ereditati non andranno definiti nuovamente e quindi non avremo bisogno di riscriverli ogni volta.

Il comando con cui si creano i figli nei vari linguaggi è generalmente **extends**. Il senso è quello di dire che la nuova classe di oggetti "estende" quella precedente con nuovi metodi e proprietà.

Vediamo come si fa per esempio in **Java** e **PHP**

Java e PHP

```
class Sensore {
    …
}
class Sensore_olio extends Sensore {
    …
}
```

Nei due linguaggi si scrive nello stesso identico modo.

Qui è stata definita prima la classe “sensore” e poi la classe “sensore_olio” che **estende** la classe precedente. “sensore_olio” è quindi figlia della classe “sensore”.

A proposito del Java, non so se ricordi, ma nei capitoli iniziali ho smesso di fare esempi usando questo linguaggio; questo è stato dovuto al fatto che Java nasce come linguaggio a oggetti e c’era quindi bisogno di arrivare a conoscerli per poter di nuovo fare degli esempi.

Approfondimenti

Nei capitoli precedenti abbiamo visto quello che c'è da sapere per poter affrontare con tranquillità un libro di programmazione su un qualsiasi linguaggio.

Tutta la base logica è racchiusa nei capitoli che hai appena letto, quello che imparerai da un manuale di programmazione sono nuovi comandi, strutture logiche più complesse e soprattutto come fare le diverse operazioni nell'ambito del linguaggio di programmazione che sceglierai.

Nei prossimi paragrafi invece troverai una panoramica di argomenti che hanno a che fare con la programmazione, ma che non sono strettamente legati alla logica con la quale vengono scritte le istruzioni.

Le banche dati

Tantissimi tipi di applicazioni diverse utilizzano delle banche dati o, come si dice in inglese, **database**.

I database sono usati praticamente da qualsiasi sito web, sono presenti nei telefonini e in tantissimi programmi installati nel tuo computer.

La loro funzione è quella di immagazzinare dati e di saperli ritrovare velocemente quando gli viene richiesto dai programmi.

Dentro ai database vengono immagazzinati semplici dati come, per esempio per un rubrica del telefono:

- nome
- cognome
- indirizzo
- telefono
- email

etc.

Oppure anche dati che hanno a che fare con la configurazione o l'utilizzo del programma stesso.

Per esempio i siti internet costruiti con i CMS come Drupal e Wordpress, che vedremo più avanti, memorizzano all'interno del database il nome del sito, il sottotitolo, le voci di menu, i nomi utente e le password, etc.

Informazioni quindi che non sono quelle che poi il visitatore vede quando utilizza il sito, ma che servono al sito stesso per funzionare.

La terminologia

Quando si parla di database, questi sono i termini principali da conoscere:

tabella: una tabella è un qualcosa che racchiude dati omogenei di un certo tipo. Per esempio la tabella contenente le notizie di un sito

potrà contenere:

- titolo
- testo
- autore
- data di pubblicazione

campi: i campi sono le “colonne” di una tabella. Sono quindi i nomi dei tipi di informazione contenuti nella tabella stessa. “Titolo”, “testo”, “autore” e “data di pubblicazione” **sono i campi della tabella notizie**.

record: il record è una “riga” della tabella. Contiene quindi al suo interno i valori dei campi relativi a uno specifico elemento. Nella tabella notizie avremo, per ogni record (riga), le informazioni su una notizia (“titolo”, “testo”, etc.)

recordset: Il recordset è un gruppo di record. Quando viene fatta una ricerca, vengono estratti solo i record della tabella che soddisfano i criteri di selezione. Questi record, messi tutti insieme, formano il così detto recordset.

Per esempio, dalla tabella “notizie”, potrebbero essere estratte tutte quelle notizie pubblicate in una certa data. Quelle notizie insieme si chiameranno recordset.

query: una query è una ricerca. Per ogni query dovrà essere definito uno specifico comando secondo quello che vedremo nel prossimo paragrafo.

Scrivere le query

Quando si programma e si sta usando un database, c’è bisogno di dire al computer quali ricerche deve effettuare.

Questi comandi, che sono rivolti al database, non dipendono dal linguaggio di programmazione che si sta usando, ma dal tipo di database al quale ci si sta rivolgendo.

Il linguaggio con cui si danno istruzioni ai database è chiamato: **SQL**

L’SQL varia leggermente da un tipo di database a un altro, ma rimane comunque praticamente identico per la maggior parte delle istruzioni principali.

Le operazioni che si fanno più frequentemente su un database sono quelle che seguono. Il comandi che vedi in grassetto sono in linguaggio SQL:

SELECT

Attraverso questa istruzione è possibile effettuare una ricerca e selezionare solo quei record che corrispondono ai parametri che vengono forniti.

Esempio

SELECT titolo, testo **FROM** tabella_notizie **WHERE** data_pubblicazione = “01-09-2017”

Questa istruzione permette di estrarre dalla “tabella_notizie” il “titolo” e il “testo” delle notizie pubblicate il primo settembre 2017.

Naturalmente si possono fare ricerche anche molto più complicate andando a incrociare dati su tabelle diverse, ma per questo si rimanda ai manuali di SQL.

INSERT

Questa istruzione permette di inserire un nuovo record in una tabella.

Esempio

INSERT INTO tabella_notizie (titolo, testo, autore, data_pubblicazione) **VALUES** (“Pubblicato un manuale per imparare a programmare”, “Finalmente è stato pubblicato un manuale per imparare a programmare rivolto a chi si avvicina per la prima volta alla programmazione”, “Mario Rossi”, “01-09-2017”)

Questo comando dice quindi al database di inserire un nuovo record nella tabella “tabella_notizie” e di assegnare ai vari campi i valori

indicati dopo VALUES.

Nota bene che questi hanno lo stesso ordine di quelli indicati dopo “tabella_notizie”; in questo modo il database sa come associarli.

UPDATE

Questa istruzione serve a modificare uno o più record.

Esempio

UPDATE tabella_notizie **SET** autore = “anonimo” **WHERE** autore = “Mario Rossi”

In questo caso stiamo dicendo al database di sostituire “anonimo” a “Mario Rossi” in tutte le notizie che hanno “Mario Rossi” come autore.

DELETE

Comando per cancellare uno o più record

Esempio

DELETE FROM tabella_notizie **WHERE** autore = “Mario Rossi”

Questa istruzione dice al database di cancellare tutte le notizie che hanno come autore “Mario Rossi”.

Oltre ai comandi principali che ho appena elencato, ne esistono tanti altri che permettono di fare altre operazioni come ad esempio: limitare il numero di record che si selezionano (es. seleziona i primi 10 nomi che iniziano con la A), ordinare i dati in ordine alfabetico, unire i dati di più tabelle, etc.

Dove si trovano i database?

L’ultimo aspetto da conoscere riguardo ai database è la loro “posizione”. I database possono essere integrati all’interno di un programma o stare magari addirittura su un altro computer.

È comune ad esempio che il codice di programmazione di un sito web sia su un server e che il database sia invece su un altro.

Grazie ai linguaggi di programmazione è poi possibile stabilire una **connessione** attraverso la quale inviare i comandi al database e ricevere poi i risultati.

Uno stesso programma potrebbe collegarsi anche a più database diversi. Questo avviene per esempio quando un programma, con una sua banca dati, sfrutta le informazioni provenienti da un altro programma.

Il secondo programma avrà la sua banca dati e, per rendere disponibili le informazioni anche al primo programma, non dovrà far altro che permettergli di stabilire una **connessione** con il suo database.

Come abbiamo visto in questa breve panoramica, quando si programma e si ha bisogno di immagazzinare dati, lo si può fare attraverso i database.

Questi permettono di gestire in modo efficiente i dati e, se necessario, di esportali o renderli disponibili anche ad altre applicazioni.

I CMS

L'acronimo CMS vuol dire "Content Management System" che significa "Sistema di Gestione del Contenuto"; la quasi totalità dei siti web è ormai gestita attraverso questi sistemi. I più famosi sono: Wordpress, Joomla e Drupal.

Tutti e tre questi CMS sono dei programmi scritti in PHP, il linguaggio più usato per il web.

I CMS sono nati per far sì che anche utenti non esperti potessero gestire un sito internet.

Per inserire una notizia su un sito, grazie ai CMS l'utente non ha più bisogno di conoscere l'HTML o il PHP e l'SQL, ma gli basta utilizzare le semplici interfacce fornite dai CMS stessi.

Dietro a quelle interfacce ci sono oggetti e funzioni in PHP che pensano a fare tutto il lavoro necessario a memorizzare la notizia sul database e poi a mostrarla all'interno del sito.

Come hai letto precedentemente, i CMS immagazzinano nei database non solo i dati veri e propri, come ad esempio le notizie, ma anche la maggior parte delle informazioni legate alla loro configurazione.

Dal punto di vista della programmazione è importante sapere che i CMS, essendo scritti in PHP, hanno il codice di programmazione completamente accessibile.

Nel caso quindi si voglia andare a capire il loro funzionamento a livello di programma, lo si potrà fare andando a leggere le istruzioni che li compongono.

L'altra cosa importante è che, sapendo programmare, si possono costruire delle estensioni personalizzate di questi CMS; è possibile cioè modificarli in modo da aggiungere delle funzionalità di cui si ha bisogno e che non sono fornite con il programma di base.

Se si programma una propria estensione, si avrà però la possibilità di sfruttare numerosi oggetti e funzioni messi a disposizione dai CMS stessi.

Per esempio un oggetto messo a disposizione potrebbe essere un pezzo di contenuto pubblicato sul sito che, ad esempio su Drupal, viene chiamato **Node** (nodo). Un nodo può essere quindi una pagina del sito, una notizia pubblicata, etc.

Drupal mette a disposizione l'oggetto Node e nella sua documentazione sono spiegati tutti i metodi e le proprietà che lo riguardano.

Esempio:

- $node->type: tipo (notizia, pagina, etc.)
- $node->title: titolo
- $node->uid: identificativo dell'autore
- $node->body: testo

etc.

E le funzioni:

- node_access: per controllare se l'utente può accedere al nodo.
- node_delete: per cancellare il nodo
- node_last_changed: per vedere l'ultima data in cui è stato modificato il nodo
- node_save: per salvare il nodo

etc.

In questo modo diventa semplice per il programmatore andare a scrivere delle estensioni che hanno a che fare con i contenuti pubblicati; non dovrà infatti andare ad analizzare tutto il database, per capire come sono organizzati i dati, e a riscrivere da zero tutte le funzioni che servono.

Proprio grazie a queste facilitazioni fornite dal sistema, sono stati sviluppati nel tempo tantissime estensioni che permettono di

aggiungere moltissime funzionalità al sistema di base.

Programmare con Google e Facebook

Oggi grazie alla programmazione è possibile fare programmi collegati alle applicazioni Google (es. google map), a Facebook, etc.

Queste piattaforme mettono a disposizione quelle che, in linguaggio tecnico, vengono chiamate API (Application Programming Interface).

Attraverso le API è possibile realizzare delle applicazioni che sfruttano i servizi messi a disposizione da chi fornisce le API.

Prendiamo per esempio Google Map, Google mette a disposizione delle API per fare tantissime operazioni.

Lo fa rendendo disponibili degli oggetti e fornendo la documentazione sui loro metodi e le loro proprietà.

Per esempio l'oggetto principale è quello definito dalla **classe Map**, alcuni dei suoi metodi sono:

getMapTypeId(): leggi il tipo di mappa (satellite, stradale, etc.)

setMapTypeId(type): imposta il tipo di mappa (satellite, stradale, etc.)

panTo(latLng): scorre la mappa fino al punto indicato dalla variabile latlng (latitudine e longitudine)

setCenter(latLng): imposta il centro della mappa

setZoom(zoomlevel): imposta il valore dello zoom

Ci sono poi tanti altri oggetti, tra questi ad esempio l'oggetto relativo ai punti segnati sulla mappa.

Questi sono definiti dalla **classe Marker**. Ecco alcune sue caratteristiche:

setClickable(flag): rende il punto cliccabile o meno, a seconda del valore true o false dato alla variabile flag

setPosition(latlng): imposta la posizione del marker

setTitle(shape): imposta il titolo del marker

È poi possibile, attraverso altri oggetti, cambiarne la forma, la dimensione, il colore, etc.

Il programmatore quindi, quando ha bisogno di inserire ad esempio una mappa personalizzata nel suo sito internet, non dovrà far altro che “collegare” il proprio programma a google e poi utilizzare metodi e proprietà forniti da google stesso.

Come fare il collegamento è spiegato di volta in volta nella documentazione. Nel caso dei siti internet è sufficiente aggiungere questa riga nella propria pagina internet:

<script src=https://maps.googleapis.com/maps/api/js?key=YOUR_API_KEY&callback=initMap async defer></script>

Si segue lo stesso tipo di approccio se si vogliono sfruttare le api di Facebook, Twitter, etc.

Si andrà a leggere la documentazione e si vedrà quali sono gli oggetti messi a disposizione e le possibilità di integrazione degli stessi all’interno delle proprie applicazioni.

Programmare un robot con Arduino

Fino a un po' di anni fa era abbastanza complicato far dialogare il software e l'hardware. Era cioè semplice fare programmi anche molto complicati che girassero sul computer, ma se si voleva far sì che un programma accendesse anche solo una lampadina, allora le cose diventavano molto più complesse.

Oggi invece, grazie ad Arduino (https://www.arduino.cc/), è diventato molto più facile far interagire l'hardware e il software.

È possibile quindi per esempio collegare anche al computer di casa dei sensori per fare delle misurazioni, oppure un braccio meccanico per fargli fare delle cose, etc.

Arduino è una scheda elettronica, contenente un microcontrollore, che fa da tramite tra il computer, al quale è collegata attraverso una normale porta usb, e componenti elettronici di vario tipo.

Arduino è poi possibile programmabile nel senso che può eseguire delle istruzioni anche quando è scollegato dal computer. Prima lo si programma tenendolo attaccato e po, una volta staccato, continua ad eseguire il programma.

Permette quindi ad esempio di fare dei robot che agiscono senza essere collegati al computer.

Arduino viene programmato con un "dialetto" dato da un misto di C e C++; facciamo ora una panoramica su come debba essere strutturato un programma.

Quando si programma Arduino si parte da due funzioni principali: setup() e loop().

La funzione **setup()** deve contenere al suo interno tutte le istruzioni di inizializzazione del sistema.

Attraverso le istruzioni contenute in questa funzione, si dice ad esempio al computer come prepararsi a usare i componenti

elettronici che sono collegati alla scheda principale.

La funzione **loop()** è quella che contiene invece tutte le istruzioni che devono essere ripetute continuamente. Loop non è altro quindi che un ciclo che si ripete all'infinito.

Dentro la funzione loop si scrivono allora tutte le istruzioni che si vuole che vengano eseguite.

Ti potrai chiedere: perché eseguirle continuamente?

La risposta è che se il sistema deve essere indipendente, allora deve continuamente controllare cosa sta succedendo per decidere cosa fare.

Pensa al nostro cervello: anche quando non stiamo facendo nulla, il cervello impiega una parte delle sue energie per analizzare gli stimoli che vengono da fuori (input).

È come se ogni istante fossero eseguite istruzioni come quelle che seguono:

- ascolta l'orecchio destro
- ascolta l'orecchio sinistro
- senti se ci sono vibrazioni
- senti la temperatura
- guarda con gli occhi

Di fatto è come se anche noi avessimo un ciclo che si ripete continuamente e che analizza quello che ci accade intorno.

In un robot è necessario qualcosa di simile; immaginiamo ad esempio una macchina che cammina da sola evitando gli ostacoli.

Avrà un sensore simile a quello della retromarcia dell'automobile, ma montato sulla parte frontale.

Ogni istante dovrà misurare la distanza per capire se si sta avvicinando troppo a un muro e se deve quindi girare o rallentare.

Questa operazione di “controllo della distanza” dovrà essere eseguita continuamente e senza sosta. Dal punto di vista della programmazione c’è quindi bisogno di un ciclo senza fine come quello della funzione “loop”.

All’interno di “loop” vengono poi scritte le istruzioni per interagire con i componenti elettronici; Arduino rende tutto questo molto semplice in quanto per ognuno di essi esistono gruppi di funzioni, metodi e proprietà che possono essere usati dal programmatore.

Ecco alcuni esempi:

digitalWrite(LED, HIGH): accende un led

digitalWrite(LED, LOW): spegne un led

analogRead(sensore): legge il valore misurato dal “sensore“ collegato

pulseIn(SONAR, HIGH): misura dopo quanto arriva un impulsto a ultrasuoni

etc.

Tutti i comandi sono spiegati nella documentazione di Arduino o contenuti nelle funzioni che vengono fornite al programmatore insieme ai componenti elettronici.

Anche in questo caso quindi, non è necessario capire in dettaglio come fa ogni singola funzione a fare quello che fa, ma sarà sufficiente usarla così come è spiegato nelle istruzioni.

Per esempio la funzione

digitalWrite(LED, HIGH)

dice ad Arduino di mandare la corrente a ciò che è collegato all’uscita dove è collegato il LED.

Come faccia a fare questo non ci interessa, di questo si è preoccupato chi ha realizzato Arduino; per noi l’importante sarà che

quel LED si accenda e, grazie alla funzione, sarà molto semplice farlo.

Quello che voglio sottolineare ancora una volta è che grazie a funzioni e oggetti, e con un minimo di basi di programmazione è ormai possibile realizzare cose che fino a poco tempo fa erano inimmaginabili.

Conclusioni

Siamo arrivati alla fine del libro e spero vivamente che ti sia piaciuto. Spero che le tue curiosità sulla programmazione siano state soddisfatte e che, se lo desideri, tu ti senta pronto a imparare un linguaggio che ti permetta di scrivere i programmi per fare quello che vuoi.

Non sono sceso troppo nel tecnico, ma come avrai capito l'obiettivo era quello di riuscire a capire la logica senza "impantanarsi" nei dettagli.

Ognuno degli argomenti dell'ultimo capitolo richiederebbe un libro a sé, ma penso sia stato importante fare comunque una panoramica su qualcosa di reale e concreto con il quale si può "giocare" oggi.

Se pensi che sia utile scrivere un manuale su quegli argomenti, se vuoi darmi suggerimenti o mandarmi dei commenti, questo è il mio indirizzo: lorenzo.foti@gmail.com

Se ti è piaciuto questo libro, prima di salutarti ti chiederei di votarlo su Amazon e magari di lasciare anche un piccolo commento. Questo farà sì che il libro sia più visibile e anche altri possano leggerlo.

Grazie e ti auguro una buona programmazione!

Lorenzo