Dart (autore: Vittorio Albertoni)

Premessa

Dart nasce in casa Google ad opera di Gilad Bracha e Lars Bak.

Il primo si autodefinisce un teologo computazionale a causa del lungo lavoro effettuato sulle specifiche del linguaggio Java e della Java Virtual Machine.

L'altro è pure un mago dell'informatica, esperto di Javascript, cui si deve lo sviluppo del velocissimo interprete Javascript del browser Chrome.

Dart, nelle intenzioni iniziali enunciate nel 2011, voleva essere un linguaggio per la programmazione web strutturato ma flessibile, che superasse i limiti di Javascript nella realizzazione di applicazioni capaci di offrire prestazioni elevate su qualsiasi browser moderno.

Divenuto da subito open source, col tempo le sue funzioni si sono espanse ed è diventato un linguaggio versatile adatto a vari tipi di applicazioni. Al punto che ci si è dimenticati dell'ambizione di rimpiazzare, con questo linguaggio, il linguaggio Javascript (anche perché gli sviluppatori non compresero mai il perché di tutto questo) e Dart, tra le tante cose che fa, può generare script in linguaggio Javascript da uno script Dart.
In questo modo è forse diventato addirittura un modo per generare script Javascript senza conoscere il relativo linguaggio. Altra cosa, comunque, che pare non interessi più di tanto gli sviluppatori.

Ma la fortuna di Dart inizia nel 2017, con la nascita di Flutter, un framework open source nato in casa Google, che, con la versione 2.0 del 2021, consente agli sviluppatori di generare in maniera stabile applicazioni multipiattaforma (per il Web, per Linux, Windows, macOS e per Android e iOS) utilizzando proprio il linguaggio Dart.

In questo manualetto presento le basi del linguaggio in modo che il principiante possa divertirsi a produrre qualche programmino facile facile.

A chi voglia approfondire suggerisco, dopo aver letto questo manualetto, di seguire le indicazioni che si trovano nel Capitolo 12.

Dart passa per un linguaggio facile da imparare. Personalmente ritengo ne esistano molti di più facili, soprattutto se pensiamo di creare un programma con GUI per desktop: tra fare qualche cosa con Python e Tkinter o con Pascal e Lazarus e farlo con Dart e Flutter ce ne passa.

Anche pensando allo sviluppo di app per dispositivi mobili, riconosciuta l'utilità di poter utilizzare lo stesso linguaggio sia per l'ambiente Android che per l'ambiente iOS, forse è più facile lavorare per Android utilizzando il linguaggio Kotlin, che semplifica Java in modo molto più deciso di quanto non faccia Dart (a partire dall'incubo di doversi ricordare il punto e virgola alla fine di ogni riga di codice).

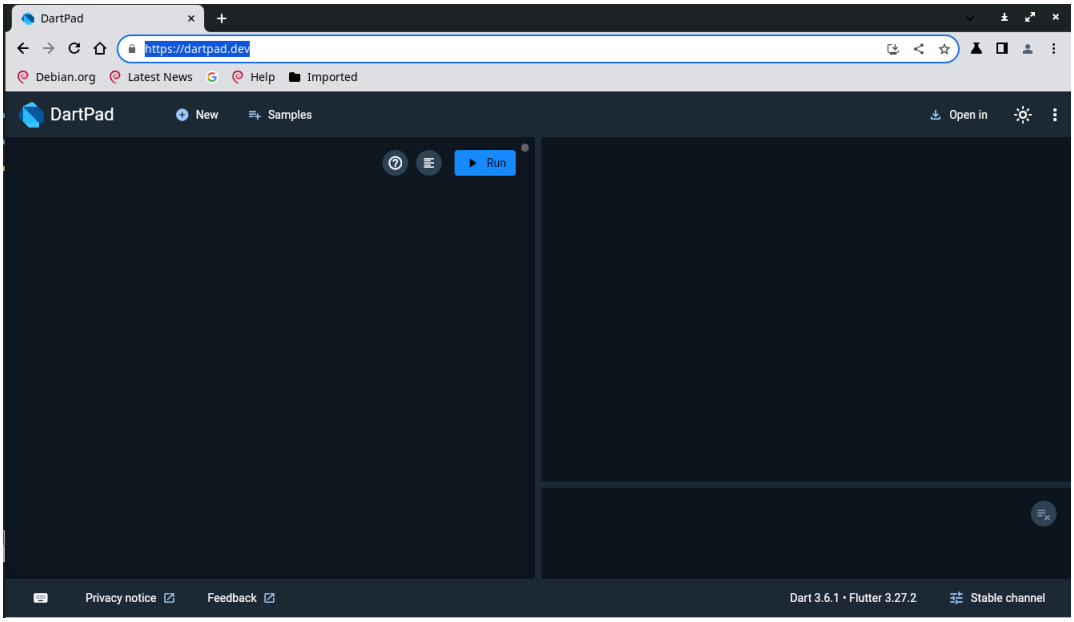
Nulla si sa dell'origine dei nomi Dart, che richiama il gioco delle freccette, e Flutter, che richiama un batter d'ali o di bandiera: misteri del software libero.

Indice

1 Installazione

Prima di parlare di installazione vera e propria è giusto avvisare che se la nostra intenzione è semplicemente di vedere come funziona il linguaggio Dart possiamo evitare l'installazione ed avvalerci, con collegamento internet attivo, di un editor online, che si chiama Dartpad.

Lo troviamo all'indirizzo *https://dartpad.dev/* e si presenta così



La zona di sinistra è l'editor in cui scriviamo le istruzioni che compongono il programma. Nel corso della scrittura otterremo suggerimenti e verremo informati di eventuali errori con messaggi nella parte bassa dell'editor.

Terminato di scrivere il programma lo possiamo eseguire cliccando sul pulsante RUN e nella zona di destra otteniamo il risultato dell'elaborazione.

Giusto per imparare.

Ciò che facciamo non è possibile salvarlo per poterlo riutilizzare.

Da principianti probabilmente non ci interessano altre limitazioni, come quella di non poter organizzare progetti di una qualche complessità e di non poter accedere ai packages del repository ma solo alle core libraries.

Se decidiamo per l'installazione possiamo scegliere tra l'installazione del solo SDK (Software Development Kit) di Dart o del complessivo Framework Flutter, che ricomprende il SDK di Dart.

Nel primo caso non abbiamo bisogno di computer particolarmente dotati ma potremo sviluppare solo applicazioni eseguibili a terminale (senza GUI), server e web.

Nel secondo caso, per una dotazione minimale, servono almeno 2 GB di RAM e circa 2 GB liberi sul disco fisso ma per lavorare a regime, utilizzando gli ambienti di sviluppo adeguati (tipo Intellij Idea o Android Studio) servono almeno 8 GB di RAM e parecchi GB di spazio su disco. Ovviamente per sfruttare Flutter, oltre al linguaggio Dart dobbiamo imparare tutto quanto serve per costruire GUI (come dice lo slogan di Flutter, per «Build for any screen») ma potremo fare veramente di tutto, anche per dispositivi mobile.

Per ciò che vedremo in questo manuale basta il SDK di Dart, che troviamo all'indirizzo *https://dart.dev/*.

Apriamo la pagina GET DART e seguiamo le istruzioni per l'installazione sul sistema operativo che ci interessa (Windows, Linux o MacOS).

Purtroppo il SDK di Dart per Linux, come software specifico, esiste solo per le distribuzioni Debian e Ubuntu.

Se utilizziamo altre distro, per avere il SDK Dart dobbiamo installare Flutter - che, come ho detto prima, lo contiene - attraverso Snap. Ne accennerò nel Capitolo 12.

2 Come funziona: il primo programma

Le istruzioni che compongono il programma si scrivono con un qualsiasi editor di testo e si salva il file con estensione `.dart`.

Questo file può essere interpretato come uno script. A tale scopo si apre il terminale (in Windows si chiama prompt dei comandi), ci si posiziona nella directory dove è stato salvato il file e si scrive il comando

`dart run` <file>`.dart`.

Il più semplice script, nella tradizione del Ciao mondo, può essere questo:

```
main()
{
   print ('Ciao mondo');
}
```

che, salvato come `ciao_mondo.dart`, può essere eseguito con il comando

```
dart run ciao_mondo.dart
```

su un computer dove sia installato il SDK di Dart facendo comparire la scritta Ciao mondo nel terminale.

In questo modo, anche con uno script così semplice, notiamo una certa lentezza nell'esecuzione.

Il SDK di Dart contiene però un potentissimo compilatore che risolve il problema della lentezza di esecuzione e della portabilità del programma eseguibile.

Il problema della lentezza di esecuzione si può risolvere con una compilazione che genera istantanee Ahead of time o Just in time per eseguire le quali è sempre necessario ricorrere al SDK di Dart.

Il problema della portabilità e, almeno in parte, anche quello della lentezza di esecuzione si risolvono con una compilazione che genera un eseguibile adatto al sistema operativo su cui avviene la compilazione, che può girare su qualsiasi computer abbia installato quel sistema operativo indipendentemente dal fatto che vi sia installato il SDK di Dart.

Ciò si ottiene con il comando

`dart compile exe` <file>`.dart`

che genera un eseguibile con estensione `.exe`, anche se non siamo su Windows e tale estensione sarebbe fuori luogo e foriera di confusione: se siamo su Linux o su Mac l'eseguibile è per Linux o Mac e possiamo rinominare il file eliminando questa estensione.
Questo eseguibile, a differenza di quanto avviene per le istantanee, contiene quanto serve per il runtime e funziona semplicemente richiamandone il nome.

Può interessare una compilazione che genera un file binario che, indipendentemente dal sistema operativo su cui è stato prodotto, può essere eseguito molto velocemente su qualsiasi computer abbia installato il SDK di Dart.

Ciò si ottiene con il comando

`dart compile kernel` <file>`.dart`

che genera un file con estensione `.dill` da eseguirsi con il comando

`dart run` <file>`.dill`

3 Tipi

Il linguaggio Dart è fortemente tipizzato a tipizzazione statica ma può essere utilizzato a tipizzazione dinamica. Vedremo meglio questo aspetto quando parleremo delle variabili.

Per intanto vediamo quali sono i principali tipi supportati dal linguaggio.

3.1 Tipi base

3.1.1 Tipi numerici

I tipi che possiamo assegnare ad un valore numerico sono i seguenti.

Intero

Tipo identificato dalla parola chiave `int`, rappresenta numeri senza decimali a 64 bit.

Ciò significa che la dimensione del numero utilizzabile, con segno, è compresa tra -2^{63} e $2^{63} - 1$, cioè tra -9.223.372.036.854.775.808 e 9.223.372.036.854.775.807.

Potremmo superare queste dimensioni ricorrendo alla classe BigInt, che Dart eredita quasi tale e quale dal linguaggio Java, ma se abbiamo esigenze di questo tipo e non ci bastano le approssimazioni ottenibili lavorando con il tipo double, che vediamo subito, è meglio che scegliamo un linguaggio che tratta gli interi a precisione arbitraria, come Python.

Per le applicazioni web il valore intero utilizzabile, come in JavaScript, è un floating point a 64 bit senza parte frazionaria, che può avere un valore compreso tra -2^{53} e $2^{53} - 1$, cioè tra -9.007.199.254.740.992 e 9.007.199.254.740.991.

Sicché se, per esempio, scriviamo un programma per calcolare il fattoriale di un numero scegliendo che il risultato sia espresso da un numero di tipo int, il numero più elevato di cui possiamo calcolare il fattoriale è 20, che vale 2.432.902.008.176.640.000 (il valore del fattoriale di 21 è superiore e $2^{63} - 1$).

Decimale

Tipo identificato dalla parola chiave `double`, rappresenta i numeri in informatica definiti «a virgola mobile» (floating point) in doppia precisione a 64 bit senza parte frazionaria (16 cifre buone, con l'ultima arrotondata, oltre alla virgola) e con possibilità di rappresentare, con questa approssimazione e in notazione scientifica, numeri di oltre 300 cifre.

Sicché se, per esempio scriviamo un programma per calcolare il fattoriale di un numero scegliendo che il risultato sia espresso da un numero di tipo `double`, il numero più elevato di cui possiamo calcolare il fattoriale diventa 170, che vale
72574156153079989673967282111292631147169916.....e avanti fino a 307 cifre, e il risultato viene espresso con
7.257415615307994e+306, come si vede sempre 307 cifre, di cui però solo le prime 15 sono esatte.

* * *

Esiste anche il tipo identificato dalla parola chiave `num` che ricomprende sia il tipo `int` sia il tipo `double`.

3.1.2 Tipo stringa

La stringa è una immutabile sequenza di caratteri UTF-16 e corrisponde all'identificativo `String`.

Si crea scrivendo i caratteri all'interno di una coppia di apici doppi (`"`) o semplici (`'`).

3.1.3 Conversioni tra tipi numerici e tipo stringa

Esistono metodi per convertire numeri in stringhe e viceversa.
<intero>`.toString()`
converte un numero intero in una stringa,
<double>`.toString()`
converte un numero decimale in una stringa,
<double>`.toStringAsFixed(n)`
converte un numero decimale in una stringa fissando i decimali a n,
`int.parse(`<stringa>`)`
converte una stringa in un numero intero,
`double.parse(`<stringa>`)`
converte una stringa in un numero decimale.
<intero>, <double> e <stringa> possono essere indicati direttamente o attraverso il nome della variabile che ne contiene i valori.

3.1.4 Tipo booleano

L'identificativo è `bool`. Solo due oggetti sono di questo tipo: `true` e `false`.

3.2 Tipi contenitore

Si tratta di collezioni di oggetti di tipo base.
Quelli più utilizzati sono i seguenti.

3.2.1 Tipo list

E' una collezione di elementi, anche eterogenei, separati da virgola (,) e racchiuso tra parentesi quadre (`[` e `]`). Gli elementi sono indicizzati a partire da 0.
`[45, 'pippo', 12.5]` è una lista.
I principali metodi ricollegabili ad un oggetto di tipo lista sono:
`.length`
per esprimere la lunghezza (numero degli elementi) della lista,
`.first`
per esprimere il primo elemento della lista,
`.last`
per esprimere l'ultimo elemento della lista,
`.elementAt(<n>)`
per esprimere l'elemento con indice n,
`.add(<elemento>)`
per aggiungere un elemento in fondo alla lista,
`.removeAt(<n>)`
per rimuovere dalla lista l'elemento con indice n.

3.2.2 Tipo set

In italiano si chiama insieme ed è una collezione di elementi, anche eterogenei, ma tutti diversi uno dall'altro, separati da virgola (,) e racchiuso tra parentesi graffe (`{` e `}`). Gli elementi sono indicizzati a partire da 0.
`{22,5, 'Luigi', 24}` è un set.
I principali metodi ricollegabili ad un oggetto di tipo set sono:
`.length`
per esprimere il numero degli elementi dell'insieme,
`.first`
per esprimere il primo elemento dell'insieme,
`.last`
per esprimere l'ultimo elemento dell'insieme,
`.elementAt(<n>)`
per esprimere l'elemento con indice n,
`.add(<elemento>)`
per aggiungere un elemento all'insieme,
`.removeAt(<elemento>)`
per rimuovere dall'insieme l'elemento indicato tra le parentesi,
`.union(<altro_set>)`
genera un insieme unione tra l'insieme cui è applicato il metodo e un altro insieme,
`.difference(<altro_set>)`
genera un insieme con elementi dell'insieme cui è applicato il metodo che non sono presenti in un altro insieme,
`.intersection(<altro_set>)`
genera un insieme con gli elementi in comune tra l'insieme cui è applicato il metodo e un altro insieme.

3.2.3 Tipo map

In altri linguaggi di programmazione si chiama dizionario ed è costituito da una collezione di abbinamenti tra una chiave e un elemento, separati dal simbolo due punti (`:`), abbinamenti separati da virgola (`,`). Il tutto tra parentesi graffe (`{` e `}`).

Chiave ed elementi possono essere di un tipo base qualsiasi.

`{1:'pera', 2:'mela', 3:'prugna'}` è un map.

L'identificazione di un elemento si ottiene facendo seguire al map la chiave dell'elemento tra parentesi quadre:

`{1:'pera', 2:'mela', 3:'prugna'}[3]` ritorna `prugna`.

Ovviamente al posto del map generalmente avremo il nome della variabile che lo contiene.

I principali metodi ricollegabili ad un oggetto di tipo map sono:

`.remove(`<chiave>`)`

per rimuovere l'elemento identificato da <chiave>,

`.addAll({`<chiave>:<elemento>, <chiave>:<elemento>, ...`})`

per aggiungere uno o più abbinamenti chiave:elemento.

4 Variabili e costanti

Le variabili e le costanti sono delle locazioni di memoria, delle scatole, alle quali diamo un nome, destinate a contenere valori di un certo tipo.

Nel primo programma visto nel Capitolo 2 abbiamo usato l'istruzione `print` per visualizzare il saluto Ciao mondo.

In quel caso abbiamo indicato all'istruzione una stringa da visualizzare.

Nello stesso modo potremmo indicare un numero da stampare o anche un'espressione numerica, scritta utilizzando gli operatori che vedremo, per visualizzarne il risultato.

Il tutto utilizzando elementi volanti.

Se ci limitassimo a questo non andremmo lontano con i nostri programmi.

In un programma che debba fare qualche cosa in più di quattro conticini, che debba sviluppare algoritmi complessi nel corso dei quali siano da prendere decisioni su cosa fare in presenza di un valore piuttosto che di un altro è necessario tenere a disposizione i valori, poterli modificare e richiamare quando serve. A questo servono le variabili e le costanti.

Variabili

La variabile è una locazione di memoria destinata a contenere un valore modificabile nel corso del programma.

Essa va creata prima di essere utilizzata.

Nei linguaggi a tipizzazione statica, come è Dart per default, la variabile va creata dichiarandone il nome e assegnandole un tipo.

Abbiamo due modi per fare questo:

. utilizzare la parola chiave `var` con la sintassi

`var` <nome> = <valore>

dove

<nome> è l'identificatore della variabile,

<valore> è il valore di inizializzazione della variabile.

Il tipo viene assegnato automaticamente in base al valore indicato.

. utilizzare la sintassi

<tipo> <nome> = <valore>

dove <valore> deve essere del <tipo> indicato.

In entrambi i casi, come si vede, la variabile va inizializzata.

Se vogliamo creare una variabile senza inizializzarla dobbiamo dichiararla facendo precedere alla dichiarazione la parola chiave `late`, con la sintassi

`late` <tipo> <nome>

Esempi:
`var nome = 'Vittorio'`
crea la variabile di tipo String nome contenente il valore Vittorio.
`var a = 12.5`
crea la variabile a di tipo double contenente il valore 12.5.
`late int x`
crea la variabile x destinata a contenere un valore di tipo numerico intero
e non la inizializza.
`double y = 14.75`
crea la variabile y destinata a contenere un valore di tipo numerico decimale
e la inizializza al valore 14.75.
`num n`
crea la variabile n destinata a contenere un valore di tipo numerico (intero o decimale)
e non la inizializza.
`String saluto = 'Ciao'`
crea la variabile saluto destinata a contenere un valore di tipo stringa
e la inizializza al valore Ciao.
In tutti questi casi le variabili create o create e inizializzate non potranno contenere in tutto il corso del programma un valore di tipo diverso da quello assunto o dichiarato al momento della creazione.

Tutto ciò perché Dart è un linguaggio a tipizzazione statica.

Esso però, attraverso la parola chiave `dynamic`, consente di creare variabili a tipizzazione dinamica, che, cioè, acquisiscono il tipo del valore che vi viene inserito durante le elaborazioni, indipendentemente dal tipo del valore precedentemente inserito.
Con
`dynamic x = 15`
creiamo la variabile x inizializzandola con il valore intero `15`
ma, successivamente, nel corso del programma, possiamo assegnarle il valore stringa `'ciao'`.

Costanti

La costante è praticamente una variabile il cui contenuto non può cambiare.

La sintassi per la dichiarazione di una costante è
`const` <nome> = <valore>
ove, per nome e valore vale quanto detto per le variabili nel precedente paragrafo.

Con questa sintassi il tipo viene automaticamente stabilito in base al valore indicato.

5 Operatori

Gli operatori collegano tra loro operandi di varia natura in espressioni che forniscono un risultato.

Quelli di uso più comune sono i seguenti.

5.1 Operatori aritmetici

In ordine di priorità sono:
`*` per la moltiplicazione tra numeri o la ripetizione di stringhe
`/` per la divisione tra numeri
`~/` per la divisione intera
`%` per il modulo (resto della divisione intera)
`+` per la somma tra numeri o il concatenamento di stringhe
`-` per la sottrazione tra numeri
`^` per l'elevamento a potenza

5.2 Operatori di confronto

Servono per confrontare due valori e il risultato che restituiscono è un valore booleano.

Sono i seguenti.

== uguale,
!= non uguale,
< minore,
<= minore o uguale,
> maggiore,
>= maggiore o uguale.

Gli operandi assoggettati al confronto devono avere lo stesso tipo.

5.3 Operatori logici

Forniscono come risultato un valore booleano e sono i seguenti.

`&&` per AND
`||` per OR

6 Funzioni

Nel Capitolo 2, esemplificando un primo semplicissimo programma scritto con il linguaggio Dart, abbiamo già utilizzato una funzione, la funzione `main()`, che è quella che si avvia con il programma e che contiene le istruzioni su ciò che il programma deve fare.

Di per sé non restituisce valori ma esegue istruzioni per cui viene comunemente utilizzata con la dizione `void main() {`<istruzioni>`}`.

Può accadere che, nel corso del programma, vi siano istruzioni che vanno eseguite parecchie volte oppure che, per organizzare un programma complesso, convenga raggruppare istruzioni che fanno una certa cosa in gruppi separati.
In questi casi si raggruppano in blocchi le istruzioni che conviene raggruppare in modo che, tutte le volte che serve, si possano eseguire le istruzioni raggruppate semplicemente richiamando il blocco anziché scriverle ogni volta tutte.

Questi blocchi di istruzioni sono le funzioni.

Dal momento che Dart è un linguaggio a oggetti, anche la funzione è un oggetto, di tipo `Function`, che può essere assegnato ad una variabile, ma possiamo anche creare la funzione in quanto tale come avviene nei linguaggi funzionali.

La creazione di una funzione in quanto tale si ottiene con la sintassi

<tipo_ritorno> <nome_funzione> (<tipo_parametro> <parametro>, ...)
{
    <istruzioni>
    return <valore_ritorno>
}

Per esempio, un funzione che raddoppia un numero si crea con

```
num raddoppia (num n)
{
   return 2 * n;
}
```

Se completiamo il programma con

```
void main()
{
   print(raddoppia(5.7));
}
```

e lo eseguiamo otteniamo sul terminale il risultato `11.4`.

Possiamo fare la stessa cosa assegnando l'oggetto funzione ad una variabile, utilizzando poi la variabile come fosse una funzione.

Nel caso dell'esempio:

```
var raddoppia = (num n) {return(2 * n);};
```

oppure, meglio

```
Function raddoppia = (num n) {return(2 * n);};
```

Completando il programma con la stessa funzione main() di prima otteniamo lo stesso risultato.

Questo procedimento è anche chiamato della funzione anonima. Infatti la funzione non ha un nome ma il nome l'ha la variabile che la contiene.

7 Funzioni precostituite

Il SDK di Dart che installiamo secondo quanto abbiamo visto nel Capitolo 1 contiene un'ampia serie di funzioni precostituite raggruppate nelle così dette Core libraries ed inoltre esiste un repository di altre librerie, denominate Packages, per affrontare ulteriori e più rare esigenze.

7.1 Core libraries

Dalla documentazione ufficiale possiamo vedere quali sono tutte le librerie disponibili e a cosa servono.

Qui non le esamineremo tutte ma solo quelle più ricorrenti per predisporre programmi alla portata di dilettanti.

Per utilizzare una libreria dobbiamo importarla scrivendo nella prima riga del programma il comando

```
import "<nome_libreria>";
```

dart:core

E' la libreria base e non c'è bisogno di importarla in quanto viene importata per default.

Contiene tutte le funzioni che ho presentato nei precedenti capitoli, dalla funzione `print()` alle funzioni per la conversione dei tipi, a quelle per manipolare liste, set, ecc.

dart:math

Si importa con

```
import "dart:math";
```

Contiene le costanti matematiche

`e` base dei logaritmi naturali,

`pi` numero π.

e le seguenti funzioni

`Random().nextInt(<n>)`

che genera un numero casuale intero compreso tra 0 e n (n escluso),

`Random().nextDouble()`

che genera un numero casuale di tipo double compreso tra 0 o 1 (1 escluso).

`pow(x, n)`

ritorna x elevato a n come double,

`sqrt(x)`

ritorna la radice quadrata di x come double,

`exp(n)`

ritorna e elevato a n come double,

`log(x)`

ritorna il logaritmo naturale di x come double,

oltre alle funzioni trigonometriche dirette (`sin()`, `cos()` e `tan()` con argomento in radianti) e inverse (`asin()`, `acos()` e `atan()` restituenti dati di tipo double in radianti).

dart:io

Si importa con
`import "dart:io";`

Questo package non può essere usato per applicazioni web e contiene strumenti per l'input e l'output di un programma. In particolare:

input e output standard Per input standard intendiamo quello dalla tastiera del computer e per output standard intendiamo quello verso lo schermo del computer.

Per l'input standard abbiamo la funzione
`stdin.readLineSync()!`
che legge quanto scriviamo sulla tastiera e lo ritorna come stringa.

Per l'outoput standard, per il quale, come ormai sappiamo, possiamo utilizzare la funzione `print()` contenuta nel package dart:core, abbiamo le funzioni
`stdout.write(`<stringa>`)` che scrive e non va a capo,
`stdout.writeln(`<stringa>`)` che scrive e va a capo.

input e output su file L'input e l'output può avvenire anche leggendo o scrivendo su file.

Qui ci occupiamo di leggere o scrivere su file di testo.

Per lavorare con un file dobbiamo innanzi tutto creare una variabile che lo contenga e lo facciamo con
`var f = File(`<percorso e nome del file>`)`
dove al posto di `f` possiamo indicare qualsiasi altro nome e <percorso e nome del file> indica non solo il nome del file ma la sua posizione nel file system.

Per leggere dal file abbiamo la funzione
`f.readAsStringSync()`

che ritorna il contenuto del file come stringa.

Per scrivere nel file abbiamo la funzione
`f.writeAsStringSync(`<stringa>`)`

che scrive la stringa indicata nel file, creando il file se non c'è o sovrascrivendolo.

Per aggiungere testo ad un file esistente la funzione di scrittura è
`f.writeAsStringSync(`<stringa>`, mode: FileMode.append)`

che aggiunge una nuova stringa sulla stessa ultima riga del file esistente;

per aggiungere una nuova stringa su una nuova riga iniziare la nuova stringa con `\n`.

7.2 Packages

Troviamo i packages per Dart e Flutter all'indirizzo *https://pub.dev/*.

Per utilizzare un package dobbiamo scaricarlo da questo sito.

Da principianti dilettanti non abbiamo certamente bisogno di ricorrere a queste librerie. Ci basti sapere che esistono e che, visitando il citato sito, possiamo vedere di che cosa si tratta.

8 Interattività con l'utente

Nel piccolo esempio «ciao mondo» che abbiamo visto nel Capitolo 2 il programma contiene già il dato da elaborare (la scritta `"Ciao mondo"`) e, una volta lanciato, fornisce il risultato secondo quella scritta. Se ci accontentassimo di questo, per salutare qualche cosa che non sia il mondo ma sia un nostro amico, dovremmo prendere il programma che abbiamo scritto per salutare il mondo, ricopiarlo sostituendo alla parola `"mondo"` il nome del nostro amico, ricompilarlo e rilanciarlo. Così dovremmo fare per tutti gli amici che volessimo salutare.

Ma con quanto abbiamo visto nel precedente Capitolo 7 sull'input standard possiamo ora fare meglio e modificare il programma in modo che, una volta lanciato, chieda all'utente chi voglia salutare e il programma diventi finalizzato a rivolgere il saluto a chicchessia senza bisogno di essere riscritto tutte le volte.

Il programma diventa il seguente

```
import "dart:io";
main() {
print("Come ti chiami?");
String nome = stdin.readLineSync()!;
print("Ciao, $nome!");
}
```

Lanciandolo saremo richiesti di indicare il nome di chi salutare e in un attimo vedremo il saluto rivolto a chi abbiamo indicato.

Così, un piccolo programma per sommare due numeri indicati dall'utente potrebbe essere così concepito

```
import "dart:io";
main() {
print("primo addendo");
num a = num.parse(stdin.readLineSync()!);
print("secondo addendo");
num b = num.parse(stdin.readLineSync()!);
stdout.writeln(a+b);
}
```

Qui vediamo l'uso di `stdout.writeln()`in luogo di `print()` e il casting con `parse()` per tramutare al volo le stringhe lette da `readLineSync()` in numeri, come indicato nel paragrafo 3.1.3.

9 Strutture di controllo

Come ogni altro linguaggio di programmazione, Dart ha dei comandi per condizionare l'esecuzione di certe istruzioni al verificarsi di determinate condizioni oppure per la ripetizione dell'esecuzione di una o più istruzioni.

9.1 Esecuzione condizionale

if

L'istruzione `if`, chiamata istruzione di esecuzione condizionale (in inglese if è il nostro se), ci dà modo di assoggettare l'esecuzione di un blocco di istruzioni al verificarsi di una determinata condizione: se la condizione è vera viene eseguito il blocco di istruzioni, altrimenti si prosegue l'esecuzione del programma saltando il blocco stesso.

La sintassi è la seguente

```
if (<condizione>)
{
   <istruzioni>
}
```

dove <condizione> è una qualsiasi espressione che relaziona due valori attraverso operatori di confronto: se la condizione si verifica vengono eseguite la o le istruzioni indicate, altrimenti si passa oltre.

L'istruzione `if` si presta anche all'esecuzione condizionale a due rami. Per ottenere questo dobbiamo abbinarla all'istruzione `else` con questa sintassi

```
if (<condizione>)
{
   <istruzioni>
}
else
{
   <istruzioni>
}
```

In questo caso se la condizione si verifica vengono eseguite le istruzioni contenute nel primo blocco altrimenti vengono eseguite quelle contenute nel secondo blocco dopo else (altrimenti). In ogni caso proseguendo poi nell'esecuzione del resto del programma.

Possiamo infine gestire l'esecuzione condizionale a più rami abbinando a `if` l'istruzione `else if` con questa sintassi

```
if (<condizione>)
{
   <istruzioni>
}
else if (<condizione>)
{
   <istruzioni>
}
else if (<condizione>)
{
   <istruzioni>
}
......
```

e proseguire quanto vogliamo.

switch

Abbiamo appena visto come l'istruzione `if` possa essere applicata a una ramificazione multipla attraverso le istruzioni aggiuntive `elseif`.

L'istruzione `switch` può convenientemente essere usata per semplificare la realizzazione della ramificazione multipla nel caso essa dipenda dal confronto tra diverse alternative che una variabile può assumere.

La sintassi è la seguente

```
switch (<variabile>)
{
   case <valore>:  {<istruzione>;}
   case <valore>:  {<istruzione>;}
   case <valore>:  {<istruzione>;}
   ....
   default <valore>:  {<istruzione>;}
}
```

dove <variabile> può essere di tipo numerico o di tipo stringa e <valore> è uno dei valori che la variabile può assumere in corrispondenza al quale eseguire una certa istruzione.

Propongo un esempio nel quale il confronto è basato sul valore di una variabile stringa.

```
import "dart:io";
main(){
print("Inserisci il nome di una moneta");
String moneta = stdin.readLineSync()!;
switch (moneta)
```

```
{
case "euro":  {print("Europa");}
case "dollaro":  {print("U.S.A.");}
case "sterlina":  {print("Regno Unito");}
case "yen":  {print("Giappone");}
case "yuan":  {print("Cina");}
default:  {print("non so");}
}
}
```

Il programma risponde indicando la zona geografica dove si usa una certa moneta indicata dall'utente e, anche se il programmatore si sforza di prevederle tutte, può sempre succedere che l'utente ne indichi una non prevista: al che il programma risponde «non so».

9.2 Ripetizione

for

Si usa quando si vuole ripetere l'esecuzione di una istruzione o di un blocco di istruzioni per un numero definito di volte.

La sintassi per l'uso di questa istruzione è

```
for (var i = 0; i < n; i = i+1)
{
    <istruzioni>
}
```

La variabile `i` è un contatore che si inizializza a 0, si stabilisce il suo valore limite `n` e la si incrementa di una unità per ogni ciclo, in modo che il ciclo si blocchi al raggiungimento del valore n. In questo modo le <istruzioni> vengono eseguite n volte.

Questo piccolo programma

```
main() {
for (var i = 0; i < 5; i = i+1)
{
print("Ciao");
}
}
```

scrive 5 volte la parola Ciao.

while

Si usa per ripetere istruzioni fino a quando si verifica una certa condizione.

La sintassi è:

```
while (<condizione>)
{
    <istruzioni>
}
```

Se la condizione è espressa attraverso l'uso di un contatore otteniamo gli stessi risultati che otteniamo con la funzione `for` vista prima

```
main() {
var i = 0;
while (i < 5)
{
print("Ciao");
i = i + 1;
}
}
```

scrive la parola Ciao 5 volte.

Altro esempio un tantino diverso

```
main() {
var frutti = ["uva", "pera", "mela", "ananas", "arancia"];
var i = 0;
while (i < frutti.length)
{
print(frutti.elementAt(i));
i = i + 1;
}
}
```

stampa l'elenco degli elementi che compongono la lista `frutti`.

10 Classi e oggetti

Se ci basassimo su quanto visto finora e sugli esempi che abbiamo incontrato potremmo pensare che Dart sia un linguaggio funzionale e Dart ci consente questo modo di programmare.

Per la verità abbiamo anche incontrato passaggi di programmazione del tipo
<tipo_numerico>`.to String()` per convertire un numero in una stringa,
<tipo_numerico>`.parse(`<tipo_stringa>`)` per convertire una stringa in un numero,
<tipo_lista>`.length` per conoscere il numero di elementi di una lista,
nei quali le funzioni (in questo caso chiamate propriamente metodi), si ricollegano, attraverso un punto (`.`), a un oggetto di un certo tipo.

Questo avviene perché, in realtà, il linguaggio Dart è un linguaggio di programmazione a oggetti che ci consente anche di programmare con le funzioni: in realtà in Dart tutto è un oggetto.
La funzione `.to String()` può essere richiamata subito dopo aver scritto un qualsiasi numero, in quanto, come scriviamo quel numero, esso diventa un oggetto, dotato, tra i tanti, del metodo `.to String()` per essere convertito in stringa.
Così se scriviamo `13.toString()` non creiamo un numero ma una stringa contenente i caratteri 1 e 3.

Il linguaggio Dart dà a noi stessi la possibilità di creare oggetti dotati di funzioni membro, dette metodi.

Facciamo questo creando le classi, che sono stampi che si utilizzano per creare oggetti.

La sintassi per dichiarare una classe è la seguente

```
class <nome_classe>
{
    <elenco_variabili_di_istanza>
    <nome_classe>(<nome_variabile>, <nome_variabile>...)
        {
        this.<nome_variabile> = <nome_variabile>
        this.<nome_variabile> = <nome_variabile>
        ...
        }
    <metodi>
}
```

Definito il nome della classe elenchiamo le variabili che caratterizzeranno l'oggetto da creare, scriviamo il metodo costruttore dell'oggetto da creare attribuendogli le relative variabili e scriviamo i metodi, cioè le funzioni membro dell'oggetto da creare.

La creazione dell'oggetto, del tipo corrispondente al nome della classe, avviene poi richiamando il nome della classe con, come argomento, l'elenco dei valori delle variabili che lo caratterizzano.

Così possiamo, per esempio, definire una classe Cerchio per creare un oggetto cerchio dotato di metodi per calcolarne area e circonferenza.

L'uso dell'indentazione non è necessario ma me ne servo per chiarezza.

```
import "dart:math";
class Cerchio
{
    late num raggio;
    Cerchio(raggio)
    {
        this.raggio = raggio;
    }
    num area()
    {
        return raggio * raggio * pi;
    }
     num circonferenza()
    {
        return raggio * 2 * pi;
    }
}
```

L'importazione della libreria `dart:math` è stata necessaria per disporre del valore ad alta precisione di π (`pi`) che essa fornisce.

L'unica variabile che caratterizza un cerchio per determinarne area e circonferenza è il raggio.

Le due funzioni membro ritornano i valori di area e circonferenza applicando le note formule.

Possiamo utilizzare questa classe per creare un cerchio di raggio 5 in un programma:

```
main()
{
    Cerchio mioCerchio = Cerchio(5);
    print("Area:  " + mioCerchio.area().toStringAsFixed(3));
    print("Circonferenza:  " + mioCerchio.circonferenza().toStringAsFixed(3));
}
```

nel quale costruiamo un oggetto mioCerchio con raggio 5 e ne scriviamo area e circonferenza utilizzando le sue funzioni membro accontentandoci di tre decimali.

Con la stessa classe possiamo creare altre istanze di cerchi con raggi diversi per calcolarne area e circonferenza.

Secondo il paradigma funzionale possiamo fare le stesse cose in questo modo

```
import "dart:math";
num areaCerchio (num raggio)
{
    return raggio * raggio * pi;
}
num circonferenzaCerchio (num raggio)
{
    return raggio * 2 * pi;
}
main()
{
    print("Area:  " + areaCerchio(5).toStringAsFixed(3));
    print("Circonferenza:  " + circonferenzaCerchio(5).toStringAsFixed(3));
}
```

11 Scrivere il programma

Il linguaggio Dart è sensibile a minuscole e maiuscole, pertanto `a` è diverso da `A`.

Nello scrivere il programma possiamo introdurre commenti in questo modo:
. dopo il simbolo `//` per commenti su una sola riga, anche di seguito all'istruzione in codice
. tra i simboli `/*` e `*/` per commenti su più righe.

L'importazione di librerie con l'istruzione `import` va scritta nelle prima righe del programma.

Il programma deve contenere il metodo `main()` che è dove inizia l'esecuzione.

Al di fuori del metodo `main()`, che contiene le vere e proprie istruzioni su che cosa deve fare il programma, possiamo scrivere il codice per le variabili, che in questo modo sono variabili globali, richiamabili in qualsiasi funzione, il codice per le funzioni e il codice delle classi.

Le variabili create all'interno di funzioni sono variabili locali e non possono essere richiamate fuori dalla funzione di appartenenza.

L'ordine con cui scrivere il codice non è regolamentato. Possiamo scrivere le funzioni e le variabili richiamate nel metodo `main()` prima o dopo aver scritto il codice per questo.

Per le classi create da noi potremmo salvare il codice in file con estensione `.dart` e costituire una libreria archiviata in una certa posizione del file system.
Queste classi sono richiamabili nel programma con l'istruzione
`import "<percorso al file della classe>";`

Ricordare sempre il punto e virgola (`;`) alla fine della riga contenente un'istruzione.

Possiamo creare anche file con estensione `.dart` contenenti codice di funzioni e costituire con essi una libreria. Questi file sono trattabili come quelli appena visti e riferiti a codice per classi.

12 Alternativa Flutter

Come ho accennato nel Capitolo 1 il SDK di Dart funziona solo su Debian e Ubuntu, per l'esattezza da Debian 11 e da Ubuntu 20.04.

Lo stesso avviene per il SDK di Flutter, che contiene il SDK di Dart, se vogliamo installarlo direttamente scaricando il relativo file da *https://docs.flutter.dev/get-started/install*, dove troviamo tutte le istruzioni per l'installazione su Windows, Linux e Mac.

Per il sistema Linux il SDK di Flutter è però disponibile anche su Snap. Il che lo rende utilizzabile, ovviamente oltre che per Debian e Ubuntu in questa modalità, anche per Fedora, CentOS, Arch Linux e Gentoo, che hanno adottato Snap.

Attraverso il SDK di Flutter possiamo così avere Dart anche su queste distribuzioni di Linux.

Se installiamo con Snap, per disporre dei comandi dobbiamo inserire nel path la directory `/snap/bin`.

I comandi che ci mette a disposizione Flutter sono
`dart`, in tutto simile allo stesso del SDK Dart, che ci elenca tutti i relativi sottocomandi,
`flutter`, che elenca tutti i sottocomandi del SDK di Flutter.

Il funzionamento di Flutter richiede che
. su Windows sia installato Visual Studio,
. su Mac sia installato x code,
. su Linux siano installati `clang`, `cmake`, `ninja-build`, `pkg-config`, `liblzma-dev` e `libgtk-3-dev` (se usiamo la versione Snap ciò non è necessario).

Se vogliamo sviluppare applicazioni web con Flutter dobbiamo avere installato anche Chrome.

Per usare Flatter come si deve occorre anche un IDE che si rispetti, a livello di Android Studio o Intellij Idea.

Il tutto senza dimenticare che Flatter esiste per creare GUI adatte a qualsiasi tipo di schermo e, a questo scopo, l'installazione del suo SDK porta con sé due package, `material.dart` e `cupertino.dart`.

Per sviluppare GUI più belle possiamo procurarci su *https://pub.dev/* i packages `fluent_ui`, `libadwaida` e `macos_ui`, rispettivamente adatti per Windows, Linux e Mac.

Tutto ciò, ovviamente, comporta la conoscenza di come funzionano queste belle cose e i widget che ci mettono a disposizione.

In questa sede accontentiamoci di sapere che Flutter porta con sé anche il SDK di Dart, il cui funzionamento abbiamo visto in questo manualetto.

Per il resto ricordo un corso a cura di Antonio Tedeschi all'indirizzo
https://www.html.it/guide/flutter-creare-app-mobile-multipiattaforma/

Interessanti, per lo sviluppo di applicazioni desktop e web i filmati ai seguenti indirizzi:
https://www.youtube.com/watch?v=abitI1uRelw&t=232s
https://www.youtube.com/watch?v=7lBwVSo1_gg
che gentilmente ci offre la Fudeo, tra i tanti corsi che rende disponibili su Flutter.

Ma con queste cose ci allontaniamo dalle capacità di un dilettante ed entriamo veramente nel professionale.

Per averne un'idea possiamo aprire su Dartpad, di cui ho parlato nel Capitolo 1, gli esempi che troviamo aprendo SAMPLES, sotto le voci FLUTTER e ECOSYSTEM.