



Empowering Digital Skills For The Jobs Of The Future



by



# Dart pt 1

# Docente



Claudia Infante



[claudia.infante@bcsoft.net](mailto:claudia.infante@bcsoft.net)

# Sommario

- Introduzione
- Installazione
- Sintassi in dart
  - Variabili
  - Commenti
  - Tipi di dato
  - Operatori
  - Funzioni
  - Control flow
- Dart asincrono
- Hello world

# Introduzione

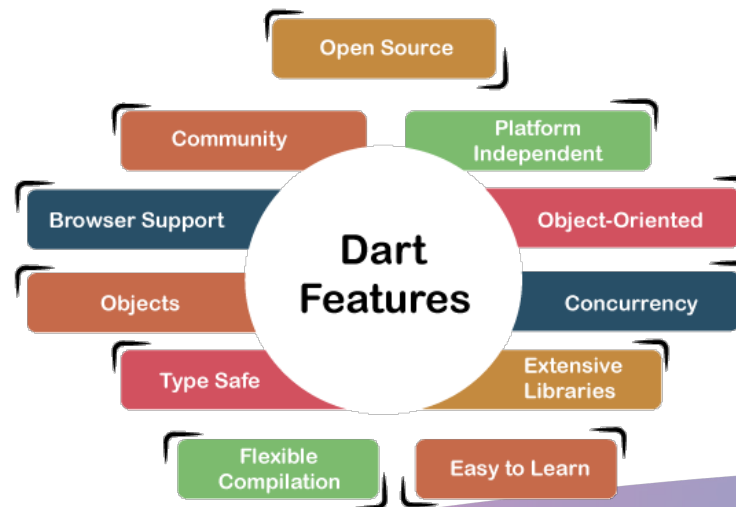
Dart è un linguaggio di programmazione strutturato e open source per la creazione di applicazioni web complesse, applicazioni web basate su browser.



# Dart

# Introduzione

Dart è un linguaggio di programmazione orientato agli oggetti e supporta tutti i concetti opzionali come classi, ereditarietà, interfacce e funzionalità di tipizzazione opzionale. Supporta anche concetti avanzati come mixin, abstract, classi, generici reificati e un sistema di tipi robusto.



# Installazione

Per utilizzare Dart è possibile sfruttare il playground messo a disposizione da Dart <https://dartpad.dev/?>.

# Installazione

Un approccio più globale prevede la sua installazione che prevede una serie di requisiti :

- Sistemi operativi supportati
- PowerShell v2+
- .NET Framework 4.8



# Installazione

Successivamente da powershell avviato come amministratori , lanciare il seguente script :

```
Set-ExecutionPolicy Bypass -Scope Process -Force;  
[System.Net.ServicePointManager]::SecurityProtocol =  
[System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-  
Object  
System.Net.WebClient).DownloadString('https://community.chocolatey.org/ins  
tall.ps1'))
```

# Installazione

Una volta eseguito, lanciare il comando choco per verificarne la versione e la corretta installazione.

```
PS C:\WINDOWS\system32> choco
Chocolatey v2.2.2
Please run 'choco -?' or 'choco <command> -?' for help menu.
PS C:\WINDOWS\system32> _
```

# Installazione

Procedere poi con l'installazione di Dart seguendo le indicazioni relative al proprio sistema operativo presenti sul sito <https://dart.dev/get-dart>

**Windows** Linux macOS

You can install the Dart SDK using [Chocolatey](#).

❗ **Important**

These commands require administrator rights. Here's one way to open a Command Prompt window that has admin rights:

1. Press **Windows+R** to open the **Run** window.
2. Type **cmd** into the box.
3. Press **Ctrl+Shift+Enter**.

To install the Dart SDK:

```
C:\> choco install dart-sdk
```

# Installazione

Lanciando il comando `dart --version` apparirà la versione attualmente presente.

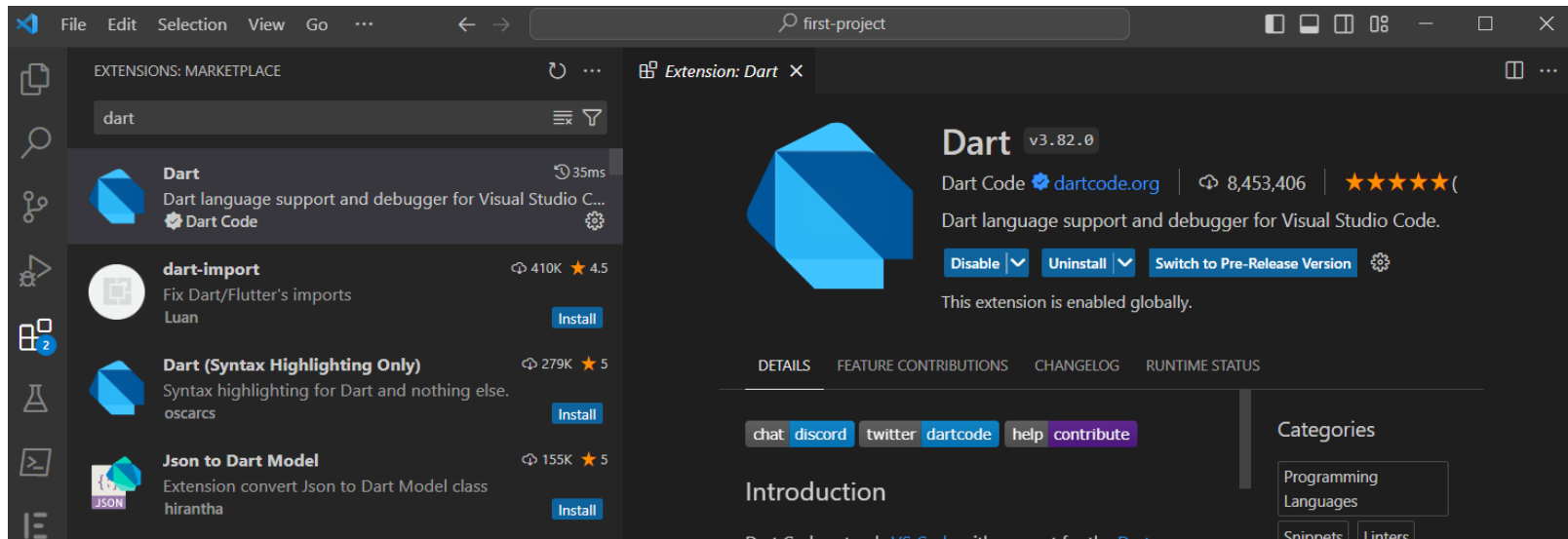
```
C:\Users\Claudia>dart --version  
Dart SDK version: 3.2.4 (stable) (Thu Dec 21 19:13:53 2023 +0000) on "windows_x64"
```

# Primo progetto in Dart

Posizionarsi tramite powershell nella directory in cui si desidera creare un progetto lanciare il comando  
`dart create --template console first-project`

# Primo progetto in Dart

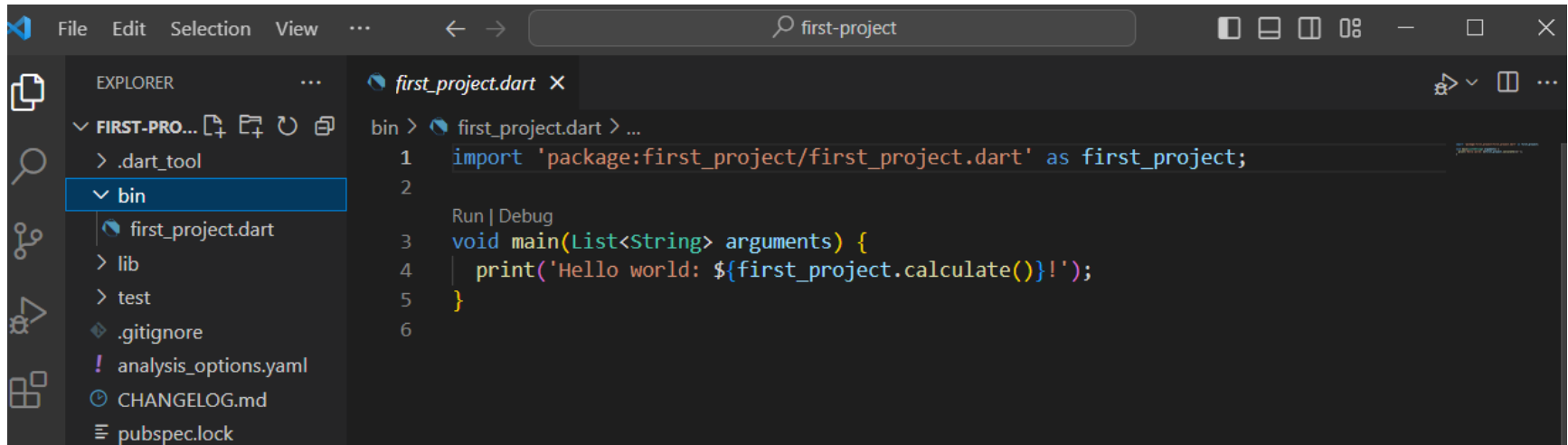
L'ide consigliato per sviluppare in Dart è Visual Studio Code, abbinando l'estensione Dart.



# Primo progetto in Dart

Il comando creerà una folder contenente una serie di file con estensione .dart.

In particolare il file presente nella folder bin :

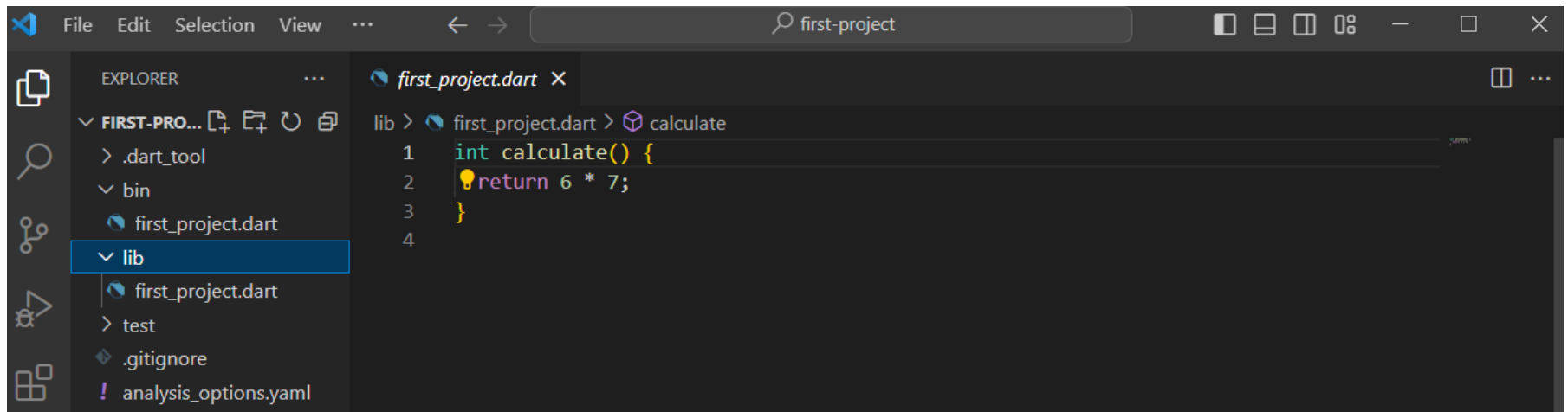


The screenshot shows an IDE window with the Explorer on the left and the editor on the right. The Explorer shows a project named 'FIRST-PRO...' with a 'bin' folder selected. The editor shows the file 'first\_project.dart' with the following code:

```
bin > first_project.dart > ...
1  import 'package:first_project/first_project.dart' as first_project;
2
   Run | Debug
3  void main(List<String> arguments) {
4      print('Hello world: ${first_project.calculate()}!');
5  }
6
```

# Primo progetto in Dart

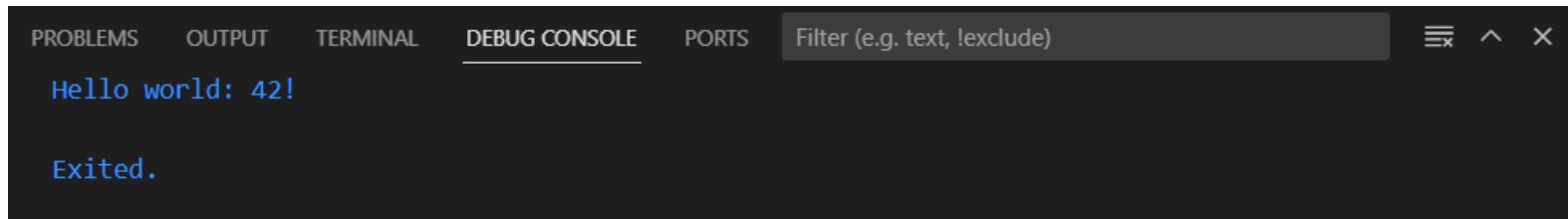
E il file presente nella folder lib.





# Primo progetto in Dart

Con il comando `dart run` o utilizzando il terminale dell'ide di riferimento, il progetto verrà compilato mostrando il risultato dell'operazione in console.

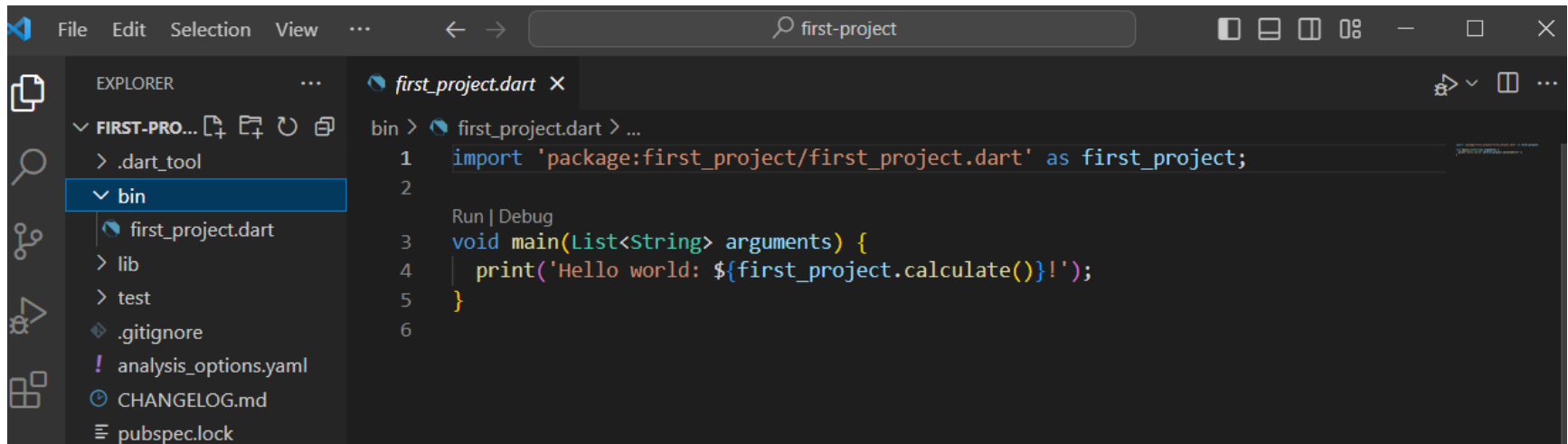


The image shows a screenshot of an IDE's Debug Console window. The window has a dark background and a light gray header bar. The header bar contains several tabs: 'PROBLEMS', 'OUTPUT', 'TERMINAL', 'DEBUG CONSOLE' (which is selected and underlined), and 'PORTS'. To the right of the tabs is a search bar with the placeholder text 'Filter (e.g. text, !exclude)'. Below the header bar, the console displays two lines of output in a light blue font: 'Hello world: 42!' and 'Exited.'.

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE  PORTS  Filter (e.g. text, !exclude)
Hello world: 42!
Exited.
```

# Funzioni

Tutti i programmi in Dart presentano una funzione obbligatoria e speciale, indicata come `main`, che rappresenta il punto di avvio dell'intera applicazione.



```
bin > first_project.dart > ...
1 import 'package:first_project/first_project.dart' as first_project;
2
3 Run | Debug
4 void main(List<String> arguments) {
5   print('Hello world: ${first_project.calculate()}!');
6 }
```

# Funzioni

La funzione `print()` consente di stampare il testo nella console e può utilizzare `${espressione}` per l'interpolazione.

```
print('Hello world: ${first_project.calculate()}!');
```

# Funzioni

È inoltre fondamentale che ogni dichiarazione termini con un punto e virgola affinché possa essere compilata. Pertanto è possibile scrivere più dichiarazioni su una stessa linea utilizzando il punto e virgola come delimitatore.

# Funzioni

Una funzione rappresenta un blocco di istruzioni. In dart ogni funzione deve definire un tipo di ritorno, il nome della funzione, eventuali parametri e il blocco delle istruzioni.

```
tipo di ritorno nome funzione (parametri) {  
  corpo della funzione }  
}
```

# Funzioni

È importante notare che Dart è un vero linguaggio orientato agli oggetti. Anche le funzioni sono oggetti, con il tipo `Function`. È possibile passare le funzioni e assegnarle a variabili.

```
void printHello() {  
  print('Hello!');  
}  
  
Run | Debug  
void main() {  
  var myFunction = printHello;  
  myFunction(); // prints "Hello!"  
}
```

# Funzioni

Dart supporta anche la sintassi abbreviata per la rappresentazione di funzioni con un solo tipo di ritorno, utilizzando le arrow functions :

```
void main() {  
  int a = 5;  
  int b = 10;  
  sum(int n, int n2) => n + n2;  
  print(sum(a, b)); // prints 15  
}
```

```
5 void main() {  
6   int a = 5;  
7   int b = 10;  
8   sum(int n, int n2) {  
9     return a + b;  
10  }  
11  print(sum(a, b)); // prints 15  
12 }
```

# Funzioni

Le funzioni Dart consentono parametri posizionali, parametri denominati, parametri posizionali e denominati opzionali o una combinazione di tutti.

I **parametri posizionali** sono quelli mostrati nell'esempio precedente in cui al richiamo della funzione vengono passati dei parametri seguendo l'ordine esatto dei due valori interi.



# Funzioni

Dart supporta i **parametri denominati**. Con il nome si intende che quando si chiama una funzione, si collega l'argomento a un'etichetta. Questo esempio richiama una funzione con due parametri denominati:

```
void main() {  
  int a = 5;  
  int b = 10;  
  double divi = divNumbers(op1: a, op2: b);  
  print(divi); // prints 15  
}  
  
double divNumbers({required int op1, required int op2}) {  
  return op1 / op2;  
}
```

# Funzioni

I **parametri opzionali** sono quei parametri non obbligatori il cui valore può dunque essere null.

```
void main() {  
  print(addSomeNums(5, 4));  
  print(addSomeNums(5, 4, 3));  
}  
  
int addSomeNums(int x, int y, [int? z]) {  
  int sum = x + y;  
  if (z != null) {  
    sum += z;  
  }  
  return sum;  
}
```

# Scope

Dart ha uno scope lessicale. Ogni blocco di codice ha accesso alle variabili dichiarate prima di esso. Per scope si intende l'ambito di visibilità definito dalla struttura del codice.

```
void main() {  
  int a = 5;  
  int b = 10;  
  
  void printSum() {  
    print(a + b);  
  }  
  
  printSum(); // prints 15  
  sum2(); //prints 30  
}  
  
sum2 () {  
  int a = 20; //can be redeclared; b doesn't exist  
  print(a +10);  
}
```

# Commenti

I commenti rappresentano un insieme di dichiarazioni che saranno ignorate dal programma. Sono utili per migliorare la leggibilità del codice e per descrivere i ruoli di variabili e funzioni.

# Commenti

In Dart esistono tre tipi di commenti :

- Su linea singola;
- Multilinea;
- Documentazione;

# Commenti

Commento su linea singola :

```
Run | Debug  
3 void main(List<String> arguments) {  
4  
5 //Questo rappresenta un esempio di commento in Dart  
6 }  
7
```

# Commenti

Commento multilinea:

```
3 void main(List<String> arguments) {  
4  
5  /* Esempio di commento  
6  su più linee */  
7 }
```

# Commenti

I commenti al documento sono usati per generare documentazione o riferimenti per un progetto/pacchetto software. Può essere un commento a una o più righe che inizia con `///` o `/*`.

Si può usare `///` su righe consecutive, alternativa al commento multilinea. Queste righe vengono ignorate dal compilatore Dart, tranne quelle scritte all'interno delle parentesi graffe.

Si possono definire classi, funzioni, parametri e variabili.



# Commenti

Esempio di commento di documento :

```
1  // Copyright (c) 2012, the Dart project authors. Please see the AUTHORS file
2  // for details. All rights reserved. Use of this source code is governed by a
3  // BSD-style license that can be found in the LICENSE file.
4
5  part of dart.core;
6
7  /// A sequence of UTF-16 code units.
8  ///
9  /// Strings are mainly used to represent text. A character may be represented by
10 /// multiple code points, each code point consisting of one or two code
11 /// units. For example, the Papua New Guinea flag character requires four code
```

# Identificatori

Regole per gli identificatori validi in Dart :

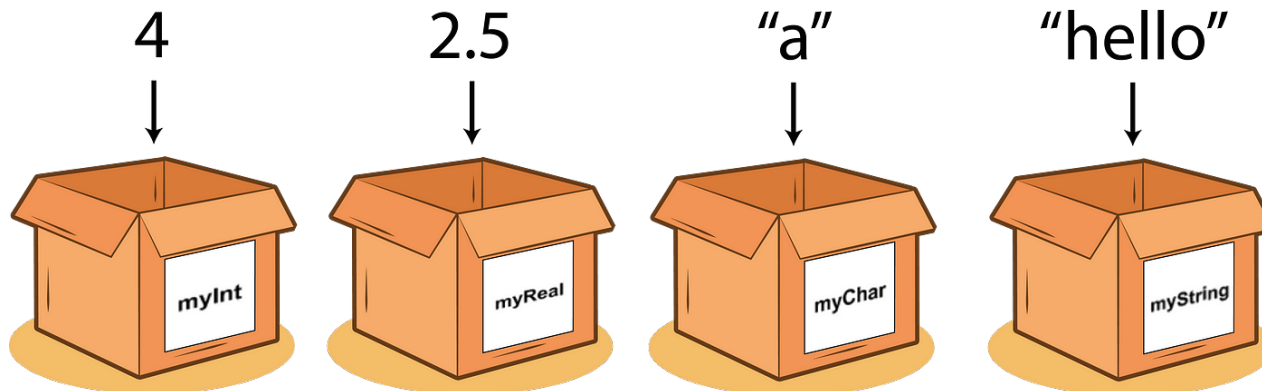
- Devono sempre iniziare con una lettera;
- Non sono ammessi caratteri speciali ad eccezione dell'underscore e del dollaro (\_ \$);
- Il primo carattere deve essere una lettera o un underscore;
- Gli identificatori devono essere unici;
- Non sono ammessi spazi;
- Sono case sensitive : una variabile indicata come A è diversa dalla variabile a

# Identificatori

Valid	Invalid
firstname	__firstname
firstName	first name
var1	V5ar
\$count	first-name
_firstname	1result
First_name	@var

# Dart parole chiave

I dati sono memorizzati in variabili, delle allocazioni in memoria che utilizzano degli identificatori per accedere alle informazioni associate.



# Dart parole chiave

Le variabili possono essere inizializzate con una serie di parole chiave, delle parole riservate che hanno un significato speciale per il compilatore.

# Dart parole chiave

Sono case sensitive e devono essere scritte così come sono definite. In Dart esistono 61 parole chiave, alcune classiche dei linguaggi di programmazione e altre speciali.

# Variabili

La creazione di una variabile rappresenta un aspetto essenziale per ogni linguaggio di programmazione.

# Variabili

Dart mette a disposizione delle regole per una corretta dichiarazione :

- Una variabile non può contenere caratteri speciali, simboli matematici, rune e parole riservate;
- Devono cominciare con una lettera dell'alfabeto;
- Sono case sensitive;
- Gli unici simboli consentiti sono l'underscore e il dollaro;
- Il nome deve essere conforme al ruolo e leggibile



# Variabili

Esistono vari modi per dichiarare una variabile in Dart :  
utilizzando delle parole chiave o facendo riferimento al tipo.

# Variabili

Var è una variabile di **tipo inferito** che consente al compilatore Dart di dedurre il tipo di una variabile in base al valore ad essa assegnato.

```
var name = "John "; // Il tipo dedotto è string
```

# Variabili

dynamic è una variabile che consente di memorizzare valori di qualsiasi tipo, senza specificare esplicitamente il tipo.

```
dynamic age = 25; // Il tipo non è esplicitato nè dedotto  
age = '30'; // Il tipo è ora indicato come stringa
```

# Variabili

Sebbene `var` sia comoda e possa rendere il codice più conciso, può anche portare a potenziali problemi sulla sicurezza dei tipi.

# Variabili

Per esempio, se per sbaglio si assegna un valore di tipo sbagliato a una variabile `var`, il compilatore Dart non coglierà l'errore fino al runtime.

D'altra parte, `dynamic` consente di rendere il codice più flessibile e più facile da mantenere, ma può anche rendere più difficile la cattura degli errori in fase di compilazione.

In generale, si consiglia di usare `var`, a meno che non ci sia una ragione specifica per usare `dynamic`.

# Variabili

`final` si usa per dichiarare una variabile che non può essere riassegnata una volta inizializzata. Quando viene dichiarata il suo valore non può essere modificato in seguito. Può essere dichiarata sostituendo la parola chiave `var` o in abbinamento al tipo.

```
final age;  
age = 25;  
age = 30; // Il valore non può essere cambiato e si avrà errore  
final String msg= 'Ricky';
```

# Variabili

Una variabile indicata con `const` deve essere inizializzata quando viene dichiarata e il suo valore non può essere modificato in seguito.

```
const weekdays = 7;
```

```
weekdays = 7 ; // Il valore non può essere cambiato e deve essere  
immediatamente inizializzato.
```

# Variabili

Sia le variabili dichiarate con `final` che con `const` non possono essere riassegnate.

Tuttavia esistono delle differenze.

In `final` il valore è modificabile mentre le variabili indicate con `const` no.



# Variabili

Ad esempio, provando ad utilizzare i metodi delle liste su due variabili indicate con `const` e con `final`, si noterà che l'esecuzione su `final` non darà errori.

```
3 void main(List<String> arguments) {  
4  
5   final giorni = ['Lunedì', 'Martedì'];  
6   giorni.add('Mercoledì');  
7   giorni.add('Giovedì');  
8   print(giorni);  
9  
10  const weekend = ['Sabato', 'Domenica'];  
11  weekend.add('Venerdì');  
12  print(weekend);  
13  
14 }
```

```
[Lunedì, Martedì, Mercoledì, Giovedì]  
Unhandled exception:  
Unsupported operation: Cannot add to an unmodifiable list  
#0      UnmodifiableListMixin.add (dart:_internal/list.dart:114:5)  
#1      main (file:///C:/Users/Claudia/Desktop/Flutter/Dart/first-pr  
ect.dart:11:11)  
#2      _delayEntrypointInvocation.<anonymous closure> (dart:isolate  
h.dart:295:33)  
#3      _RawReceivePort._handleMessage (dart:isolate-patch/isolate_p
```

# Variabili

`static` viene utilizzato per dichiarare una variabile o un metodo a livello di classe che appartiene alla classe stessa piuttosto che a una particolare istanza della classe. È possibile accedere a una variabile o a un metodo statico senza creare un'istanza della classe.

# Variabili

```
class MyClass {  
  static int count = 0; // variabile di classe  
  static void incrementCount() { //metodo di classe  
    count++;  
  }  
}
```

# Variabili

Nell' esempio precedente, `count` e `incrementCount()` sono entrambi dichiarati come statici. Ciò significa che appartengono alla classe `MyClass` e non a una particolare istanza della classe.

È possibile accedervi utilizzando il nome della classe, in questo modo:

```
MyClass.count = 10;  
MyClass.incrementCount();
```

# Variabili

Sebbene le variabili e i metodi statici possano essere utili in alcune situazioni, devono essere usati con cautela.

Possono rendere il codice più difficile da capire e da mantenere e possono introdurre bug se non vengono usati correttamente.

In generale, si raccomanda di usare le variabili `final` per dichiarare le costanti e le variabili e i metodi statici per dichiarare i dati e il comportamento a livello di classe.

# Tipi di dato

Dart è un linguaggio tipato.

I tipi di dato built in in dart sono : numeri, stringhe, booleani, collections, rune, null, oggetti generici, future, stream, iterable, never, dynamic, void e simboli.

Tutti i tipi di dato in Dart sono oggetti.

# Tipi di dato

**Numeri** : interi e decimali. Dart mette a disposizione 3 modi per dichiarare un numero : num, int, double

```
num x = 1;
```

```
int y = 2;
```

```
double j = 1.42;
```

```
var m = 3;
```

# Tipi di dato

```
Run/Debug
3 void main() {
4   num x = 1;
5   int y = 2;
6   double j = 1.42;
7   var m = 3;
8   // print(x.isOdd); //avremo errore poiché è troppo generico
9   print(y.isOdd); // funziona poiché è un intero;
10  // print(j.isOdd); //avremo errore poiché è un decimale
11  print(m.isOdd);
12 }
```



# Tipi di dato

Alcuni metodi per i numeri, oltre ad isEven e isOdd sono il ceil, il floor e il round.

```
void main() {  
  num x = 5.1;  
  print(x.ceil()); //Il successivo minimo intero non più piccolo di questo  
  print(x.floor()); //il numero maggiore non superiore all'attuale  
  print(x.round()); // valore arrotondato  
}
```

# Tipi di dato

**Stringhe** : qualsiasi sequenza di unità di codice UTF-16. Per creare una stringa si possono usare sia apici singoli che doppi:

```
var s1 = 'Single quotes work well for string literals.';
```

```
var s2 = "Double quotes work just as well.";
```

```
String s3 = "esempio stringa";
```

# Tipi di dato

Dart mette a disposizione una serie di metodi per le stringhe come il `toString` che legge il valore ricevuto e lo trasforma in stringa. Viceversa, per trasformare un valore stringa in numerico si utilizzerà il `parse`:

```
void main() {  
  num x = 1;  
  print('valore numero ' + x.toString());  
}
```

```
3 void main() {  
4   String x = '1';  
5   print(2+ int.parse(x));  
6 }
```

# Tipi di dato

Le stringhe non sono altro che una sequenza di caratteri, pertanto è possibile contare la lunghezza utilizzando il metodo `length()` che ritorna il numero di caratteri che costituisce la stringa, di cui sarà possibile estrarre le lettere da cui è costituita tramite l'indicizzazione dei caratteri.

```
3 void main() {  
4   String name = 'Claudia';  
5   print(name.length); //7  
6   print('prima lettera ${name[0]}'); //C  
7 }
```

# Tipi di dato

Utilizzando il metodo `split` la stringa potrà essere scomposta e definire una lista costituita dagli elementi della stringa .

```
3 void main() {  
4   String name = 'Nome: Claudia';  
5   print(name.split(':'));  
6 }
```



```
[Nome, Claudia]  
  
Exited.
```

# Tipi di dato

**Booleani** : accettano valori come true o false.

```
var fullName = "";  
assert(fullName.isEmpty);  
bool isAlive = true;
```

# Tipi di dato

**Array:** una collezione ordinata di dati o oggetti rappresentata dall'uso delle parentesi quadre e indicati come liste.

```
var list = [ 'Car', 'Boat', 'Plane', ];
```

```
List<String> mialista = [ 'Car', 'Boat', 'Plane', ];
```

Le liste utilizzano un'indicizzazione basata su zero, dove 0 è l'indice del primo valore e `list.length - 1` è l'indice dell'ultimo valore.

# Tipi di dato

È possibile valorizzare una lista anche in base a dei cicli o a delle condizioni.

```
3 void main() {  
4   var list = [  
5     'Car',  
6     'Boat',  
7     'Plane',  
8   ];  
9  
10  
11 List<String> negozio = [  
12   for(var i in list) "vehicle : $i"  
13 ];  
14 print(negozio);  
15 }  
16
```



# Tipi di dato

**Sets:** una collezione disordinata di dati o oggetti rappresentata dall'uso delle parentesi graffe. I valori dei set sono unici e non ordinati.

Un set non ammette duplicati:

`var` set1 = {'Car', 'Boat', 'Plane', 'Boat'}; //Boat verrà preso una sola volta

`Set<String>` set2 = {'Car', 'Boat', 'Plane', 'Boat'};

# Tipi di dato

I set possono essere riempiti tramite metodi :

```
var set3 = Set();  
set3.add(1);  
set3.add(2);  
set3.addAll({3, 4, 5});  
print(set3); // {1, 2, 3, 4, 5}  
set3.removeWhere((item) => item % 2 == 0);  
print(set3); // {1, 3, 5}
```

# Tipi di dato

È possibile anche confrontare i valori di più set tramite i metodi intersection, union e difference:

- Intersect mostra gli elementi comuni;
- Union unisce i set;
- Difference mostra i valori presenti nel primo set che sono assenti nel secondo set

# Tipi di dato

```
void main() {  
  var set1 = {1,2,3,4};  
  var set2 = {3,4,5};  
  print(set1.intersection(set2)); //{3, 4}  
  print(set1.union(set2)); //{1, 2, 3, 4, 5}  
  print(set1.difference(set2)); //{1, 2}  
  print(set2.difference(set1)); //{5}  
}
```

# Tipi di dato

Le mappe forniscono anche dei modi per accedere alle chiavi o ai valori, ad esempio utilizzando il `forEach` che scorrerà tutti gli elementi della mappa e recupererà le chiavi e/o i valori.

```
void main() {  
  var mappa = {"nome" : "Mario", "cognome" : "Rossi", "anni" : 36};  
  Map<String, Object> mappa2 = {...mappa, "professione" : "avvocato"};  
  mappa2.forEach((key, value) {  
    |   print("$key : $value");  
    | });  
  }  
}
```

# Tipi di dato

Dart è un linguaggio tipizzato. In fase di dichiarazione di una variabile, il tipo può essere espresso per ridurre il codice duplicato e definire il codice in modo più rigoroso:

```
var names = <String>['Seth', 'Kathy', 'Lars'];  
var uniqueNames = <String>{'Seth', 'Kathy', 'Lars'};  
var pages = <String, String>{  
  'index.html': 'Homepage',  
  'robots.txt': 'Hints for web robots',  
  'humans.txt': 'We are people, not machines'  
};
```

# Tipi di dato

**Rune** : Le rune di Dart sono stringhe speciali di unità Unicode UTF-32. Vengono utilizzate per rappresentare una sintassi speciale.

Ad esempio, il carattere speciale cuore ♥ equivale al codice Unicode `\u2665`, dove `\u` significa Unicode e i numeri sono interi esadecimali.

Una lista di emoji con il relativo codice è disponibile [qui](#).

```
Run | Debug
5  void main () {
6      var heartsymbol = '\u2665';
7      print(heartsymbol);
8  }
9
```

# Operatori

Un operatore rappresenta un simbolo utilizzato per manipolare dei valori o eseguire delle operazioni.



# Operatori

In Dart sono presenti i seguenti operatori:

- Aritmetici 1
- Assegnazione 2
- Confronto 3
- Logici 4
- Bitwise 5
- Condizionali 6
- A cascata 7

# Operatori

Si supponga che  $a$  sia valorizzato a 20 e  $b$  sia valorizzato a 10.

# Operatori

## Aritmetici

1

Sono gli operatori utilizzati per eseguire addizioni, sottrazioni, moltiplicazioni ecc.

Operatore	Descrizione	Esempio
Addizione (+)	Aggiunge l'operando di sinistra all'operando di destra	$a+b$ ritorna 30
Sottrazione (-)	Sottrae all'operando di destra l'operando di sinistra.	$a-b$ ritorna 10
Divisione (/)	Divide il primo operando per il secondo e ritorna il quoziente.	$a/b$ ritorna 2.0
Moltiplicazione (*)	Moltiplica gli operandi tra loro	$a*b$ ritorna 200
Modulo(%)	Ritorna il resto della divisione tra due operandi	$a\%b$ ritorna 0
Divisione(~/)	Divide il primo operando per il secondo operando e restituisce il quoziente intero.	$a/b$ ritorna 2
Unary Minus(-expr)	Utilizzato con un singolo operando ne cambia il segno.	$-(a-b)$ ritorna -10

# Operatori

Appartengono a questa categoria gli operatori unari in condizione di post e pre

Operatore	Descrizione	Esempio
++(Prefix)	Incrementa il valore dell'operando.	++x
++(Postfix)	Ritorna il valore attuale dell'operando prima dell'incremento.	x++
--(Prefix)	Decrementa il valore dell'operando.	--x
--(Postfix)	Ritorna il valore attuale dell'operando prima del decremento	x--

# Operatori

Assegnazione

2

Gli operatori di assegnazione vengono utilizzati per assegnare un valore alle variabili. Possono essere utilizzati anche in combinazione con gli operatori aritmetici.

# Operatori

Operatore	Descrizione
= (Operatore di assegnazione)	Assegna l'espressione destra all'operando sinistro.
+=(Aggiungi e assegna)	Aggiunge il valore dell'operando destro all'operando sinistro e l'assegnazione risultante all'operando sinistro. Ad esempio: $a+=b \rightarrow a = a+b \rightarrow 30$
-= (Sottrai e assegna)	Sottrae il valore dell'operando destro dall'operando sinistro e assegna nuovamente l'operando sinistro. Ad esempio: $a-=b \rightarrow a = a-b \rightarrow 10$
*=(Moltiplica e assegna)	Moltiplica gli operandi e assegna il risultato all'operando sinistro. Ad esempio: $a*=b \rightarrow a = a*b \rightarrow 200$
/=(Dividi e assegna)	Divide il valore dell'operando sinistro per l'operando destro e assegna il risultato all'operando sinistro. Ad esempio: $a/=b \rightarrow a = a/b \rightarrow 2.0$
~/=(Dividi e assegna)	Divide il valore dell'operando sinistro per l'operando destro e il quoziente intero del resto torna all'operando sinistro. Ad esempio: $a~/=b \rightarrow a = a/b \rightarrow 2$
%=(Modifica e assegna)	Divide il valore dell'operando sinistro per l'operando destro e assegna il resto all'operando sinistro. Ad esempio: $a%=b \rightarrow a = a \% b \rightarrow 0$

# Operatori

Operatore	Descrizione
$\ll$ =(Sposta a sinistra E assegna)	L'espressione $a \ll 3$ è uguale a $a = a \ll 3$
$\gg$ =(Sposta a destra E assegna)	L'espressione $a \gg 3$ è uguale a $a = a \gg 3$
$\&$ =(Assegna AND bit a bit)	L'espressione $a \& 3$ è uguale a $a = a \& 3$
$\wedge$ =(OR esclusivo bit per bit e assegnazione)	L'espressione $a \wedge 3$ è uguale a $a = a \wedge 3$
$ $ =(OR incluso bit per bit e assegnazione)	L'espressione $a   3$ è uguale a $a = a   3$

# Operatori

## Confronto

3

Gli operatori di confronto vengono utilizzati per effettuare un confronto tra due espressioni e operandi. Il confronto di due espressioni restituisce il valore booleano vero o falso.

Operatore	Descrizione
>(maggiore di)	$a > b$ restituirà VERO.
<(minore di)	$a < b$ restituirà FALSO.
>=(maggiore o uguale a)	$a \geq b$ restituirà VERO.
<=(minore o uguale a)	$a \leq b$ restituirà FALSO.
==(è uguale a)	$a == b$ restituirà FALSO.
!=(diverso da)	$a != b$ restituirà VERO



# Operatori

Il confronto può anche essere più rigoroso facendo un test sul tipo per testare i tipi di espressione in fase di runtime

Operatore	Descrizione
as	Utilizzato per il typecast
is	Ritorna vero se l'oggetto è del tipo specificato
is!	Ritorna vero se l'oggetto NON è del tipo specificato

# Operatori

Logici

4

Gli operatori logici vengono utilizzati per valutare le espressioni e prendere la decisione. Dart supporta i seguenti operatori logici.

Operatore	Descrizione
&&(AND)	Restituisce se tutte le espressioni sono vere.
(OR)	Restituisce TRUE se almeno un' espressione è vera.
!(NOT)	Restituisce il valore complementare dell' espressione.

# Operatori

Bitwise

5

Gli operatori Bitwise eseguono operazioni bit per bit sul valore dei due operandi.

Si supponga per esempio:

If  $a = 7$

$b = 6$

then  $\text{binary}(a) = 0111$

$\text{binary}(b) = 0011$

Hence  $a \& b = 0011$ ,  $a|b = 0111$  and  $a^b = 0100$

# Operatori

Operatore	Descrizione
&(AND binario)	Restituisce 1 se entrambi i bit sono 1.
(OR binario)	Restituisce 1 se uno qualsiasi dei bit è 1.
^(XOR binario)	Restituisce 1 se entrambi i bit sono diversi.
~(Complemento)	Restituisce il contrario del bit. Se il bit è 0 il complemento sarà 1.
<<(Shift a sinistra)	Il valore dell'operando sinistro si sposta a sinistra del numero di bit presenti nell'operando destro
>>(Shift a destra)	Il valore dell'operando destro si sposta a sinistra del numero di bit presenti nell'operando sinistro.

# Operatori

Condizionali

5

Gli operatori condizionali consentono di eseguire controlli di tipo if-else utilizzando dei simboli.

- Se la condizione è vera restituisce la prima espressione, altrimenti restituisce la seconda.

condizione ? esp1 : esp2

- Se l'espressione NON è nulla restituisce il suo valore, altrimenti restituisce la seconda espressione.

esp1?? esp2

# Operatori

A cascata

7

Viene utilizzato per valutare una serie di operazioni sullo stesso oggetto.

```
var list = [1, 2, 3];  
var list2 = [0, ...list];  
assert(list2.length == 4)
```

# Control Flow Statement

Le **istruzioni di controllo** vengono utilizzate per controllare il flusso del programma. Definiscono delle fasi di decisione in cui sarà stabilito se procedere con l'esecuzione o meno. L'istruzione viene eseguita generalmente in modo sequenziale.

# Control Flow Statement

In dart esistono tre tipi di flussi :

- Istruzioni decisionali;
- Istruzioni iterative;
- Istruzioni di salto;



# Control Flow Statement

Le **istruzioni decisionali** stabiliscono quale codice sarà eseguito in base alla valutazione di un'espressione. Nella programmazione con Dart, le espressioni di test singole o multiple ritornano un valore booleano di true o false che determinerà l'esecuzione del codice.

# Control Flow Statement

- If e if-else

```
3 void main() {  
4     int age = 25;  
5  
6     if (age >= 18) {  
7         print("You are an adult.");  
8     } else {  
9         print("You are a minor.");  
10    }  
11  
12    int temperature = 25;  
13  
14    if (temperature > 30) {  
15        print("It's hot outside.");  
16    } else if (temperature < 10) {  
17        print("It's cold outside.");  
18    } else {  
19        print("It's nice outside.");  
20    }  
21 }
```

# Control Flow Statement

- Switch-case

```
3 void main() {  
4     var dayOfWeek = 3;  
5  
6     switch (dayOfWeek) {  
7         case 1:  
8             print("Monday");  
9             break;  
10        case 2:  
11            print("Tuesday");  
12            break;  
13        case 3:  
14            print("Wednesday");  
15            break;  
16        case 4:  
17            print("Thursday");  
18            break;  
19        case 5:  
20            print("Friday");  
21            break;  
22        case 6:  
23            print("Saturday");  
24            break;  
25        case 7:  
26            print("Sunday");  
27            break;  
28        default:  
29            print("Invalid day of week");  
30        }  
31    }  
32 }
```

# Control Flow Statement

Le istruzioni iterative, anche indicate come cicli o **loop**, sono utilizzate per eseguire più volte uno stesso blocco di codice fin quando non è incontrata una condizione limite.

# Control Flow Statement

Dart fornisce i seguenti tipi di cicli :

- Dart for loop
- Dart for....in loop
- Dart while loop
- Dart do while loop

# Control Flow Statement

- for

```
Run | Debug
3 void main() {
4     // Loop from 0 to 9
5     for (int i = 0; i < 10; i++) {
6         print(i);
7     }
8 }
```

- for in

```
void main() {
    // Loop over a list of numbers
    List<int> numbers = [1, 2, 3, 4, 5];
    for (int num in numbers) {
        print(num);
    }

    // Loop over a map
    Map<String, int> scores = {"Alice": 80, "Bob": 90, "Charlie": 100};
    for (String name in scores.keys) {
        print("$name: ${scores[name]}");
    }
}
```

# Control Flow Statement

- while

```
3  void main() {  
4      // Example of a while loop  
5      var i = 0;  
6      while (i < 5) {  
7          print(i);  
8          i++;  
9      }  
10 }
```

- do while

```
3  void main() {  
4      // Example of a do-while loop  
5      var j = 0;  
6      do {  
7          print(j);  
8          j++;  
9      } while (j < 5);  
10 }
```

# Nullable

In Dart il nullable è supportato dalla versione 2.12. Durante l'esecuzione è possibile ricevere dei valori nulli. Si parte dal presupposto che di default ogni variabile non può essere null.



# Nullable

In Dart, le variabili nullable sono variabili che possono avere un valore del tipo specificato o null. Ciò significa che una variabile nullable può contenere un valore del tipo specificato oppure nessun valore.

Per dichiarare una variabile nullable, si può usare il simbolo ? dopo il nome del tipo.

```
String? name;  
print(name); //null
```

# Eccezioni

In Dart, le eccezioni sono errori che si verificano durante l'esecuzione di un programma e che non sono causati da errori di programmazione.

Le eccezioni vengono utilizzate per gestire situazioni inaspettate che possono verificarsi durante l'esecuzione di un programma, come la divisione per zero, un file non trovato o un errore di rete.

# Eccezioni

Dart utilizza un sistema strutturato di gestione delle eccezioni per la loro gestione. Quando si verifica un'eccezione, Dart cerca un blocco catch in grado di gestirla. Se viene trovato, il codice in esso contenuto viene eseguito per gestire l'eccezione. Se non viene trovato, l'eccezione viene nuovamente lanciata, causando il blocco del programma.

# Eccezioni

```
Run | Debug
5 void main() {
6   Random random = Random();
7   try{
8     int randomNumber = random.nextInt(2);
9     if(randomNumber ==0) {
10      throw Exception(['numero non valido' , randomNumber]);
11    } else {
12      print('numero valido, $randomNumber');
13    }
14  } catch(e) {
15    print(e);
16  }
17
18 }
```

# Classi

Dart è un linguaggio di programmazione orientato agli oggetti e supporta tutti i concetti della programmazione orientata agli oggetti come classi, oggetti, ereditarietà, mixin e classi astratte.

# Classi

I concetti degli Oops sono :

- Classe
- Oggetto
- Eredità
- Polimorfismo
- Interfacce
- Classe astratta

# Classi

In Dart, le classi sono utilizzate per definire oggetti che hanno proprietà e metodi. Una classe viene definita utilizzando la parola chiave `class` seguita dal nome della classe, dai suoi campi, costruttore e metodi.

# Classi

Ecco un esempio di come definire una semplice classe in Dart:

```
class Person {  
  String name;  
  int age;  
  
  Person(this.name, this.age);  
  
  void sayHello() {  
    print('Hello, my name is $name and I am $age years old.');  }  
}
```



# Classi

Grazie al modello della classe è possibile rappresentare un'istanza di quella classe. L'oggetto ha due caratteristiche: stato e comportamento definiti dagli attributi e dai metodi della classe .

# Classi

Si può accedere alle proprietà della classe creando un oggetto di quella classe.

```
Run | Debug  
void main() {  
  var p1 = Person('Luca', 23);  
  p1.sayHello();  
  var p2 = Person('Martina', 34);  
  p2.sayHello();  
}
```

# Classi

Dart fornisce dei parametri e dei metodi chiamati costruttori nominali per definire un costruttore personalizzato.

```
5 void main() {  
6   var p1 = Person(name : 'Luca', age : 23);  
7   p1.sayHello();  
8   var p2 = Person(age : 34, name : 'Martina');  
9   p2.sayHello();  
10  var p3 = Person.standard();  
11  p3.sayHello();  
12 }  
13  
14 class Person {  
15   late String name;  
16   late int age;  
17  
18   Person({required this.name, required this.age});  
19   Person.standard() {  
20     name = 'Mario';  
21     age = 18;  
22   }  
23   void sayHello() {  
24     print('Hello, my name is $name and I am $age years old.');25   }  
26  
27 }  
28
```

# Classi

Dart supporta l'ereditarietà, che viene utilizzata per creare nuove classi da una classe esistente. La classe da estendere si chiama genitore/superclasse e la classe appena creata si chiama figlia/sottoclasse. Dart fornisce la parola chiave **extends** per ereditare le proprietà della classe genitore nella classe figlio.

# Classi

La sintassi prevede:

class nome\_classe\_figlia extends nome\_classe\_genitore e il riferimento al costruttore della classe padre a cui si punta con la parola chiave super.

```
class Student extends Person {  
    String major;  
    Student(String name, int age, this.major) : super(name, age);  
    void study() {  
        print('I am studying $major.');    }  
}
```

# Classi

La nuova istanza della classe figlia potrà accedere ai metodi e agli attributi di entrambe le classi.

```
Run | Debug
void main() {
    Student student = Student('John', 20, 'Computer Science');
    student.sayHello();
    student.study();
}

class Person {
    String name;
    int age;
    Person(this.name, this.age);
    void sayHello() {
        print('Hello, my name is $name and I am $age years old.');
```

```
}
```

```
}
```

```
class Student extends Person {
```

```
    String major;
```

```
    Student(String name, int age, this.major) : super(name, age);
```

```
    void study() {
```

```
        print('I am studying $major.');
```

```
    }
```

```
}
```

# Classi

I metodi, oltre ad essere richiamati, possono anche essere sovrascritti grazie alla notazione `@override`.

Questa notazione dichiara esplicitamente la volontà di cambiare il comportamento del metodo ricevuto dalla classe padre.

```
class Student extends Person {  
  String major;  
  Student(String name, int age, this.major) : super(name, age);  
  void study() {  
    print('I am studying $major.');  }  
  @override  
  void sayHello() {  
    print('Goodbye, my name is $name and I am $age years old.');  }  
}
```

# Dart Asincrono

Dart fornisce anche una serie di metodi per la gestione del codice asincrono. L'operazione asincrona viene eseguita separatamente dal thread principale dell'operazione.

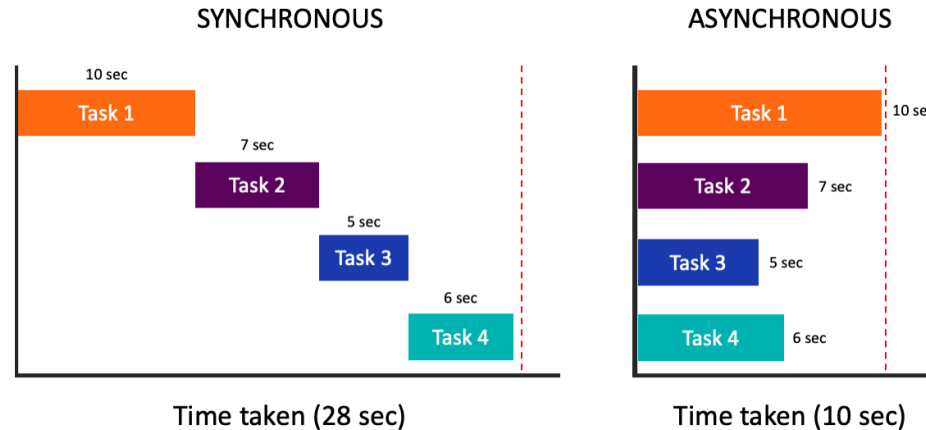


# Dart Asincrono

In informatica, quando si dice che un particolare programma è sincrono, significa che attende un evento per proseguire l'esecuzione. Questo approccio ha uno svantaggio: se una parte del codice impiega molto tempo per essere eseguita, i blocchi successivi vengono bloccati da un blocco non correlato.

# Dart Asincrono

Questo è il problema principale dell'approccio sincrono. È possibile che una parte del programma debba essere eseguita prima della parte corrente, ma l'approccio sincrono non lo consente.



# Dart Asincrono

La programmazione asincrona si concentra generalmente su un modello di programmazione senza attesa o non bloccante. L'opzione `dart: async` facilita l'implementazione del blocco di programmazione asincrona in uno script Dart.

```
Run | Debug
void main() async {
  print('First');
  String result = await fetchData();
  print(result); // prints "Second" after 2 seconds
}

Future<String> fetchData() {
  Completer<String> completer = Completer<String>();
  Timer(Duration(seconds: 2), () {
    completer.complete('Second');
  }); // Timer
  return completer.future;
}
```

# Dart Asincrono

**Future** è un oggetto utilizzato per facilitare la programmazione asincrona. Denotano che l'espressione ritornerà un valore a seguito di un'esecuzione che prima o poi risulterà completata. Affinchè possa essere utilizzato va abbinato ad **async** e **await**.

# Dart Asincrono

Le parole chiave **async** e **await** possono implementare la programmazione asincrona. La parola chiave **async** è necessaria per eseguire la funzione in modo asincrono; bisogna aggiungere **async** dopo il nome della funzione.

# Dart Asincrono

La parola chiave **await** viene utilizzata anche per eseguire la funzione in modo asincrono. Sospende la funzione attualmente in esecuzione finché il risultato non è pronto. Quando restituisce il risultato, continua con la riga di codice successiva. La parola chiave **await** può essere utilizzata solo con funzioni asincrone.

```
Future<void> greetings() async {  
  | print("Hello");  
}  
  
Run | Debug  
void main() async {  
  | await greetings(); // Using await keyword  
  | print("Task Complete");  
}
```

# Hello world

In base a quanto definito, si provi a configurare il progetto in modo da stampare un elenco di saluti.

# Hello world

Prima di tutto si definisce un tipo di collezione, che ha al suo interno oggetti con il relativo tipo: in questo caso, un elenco pieno di stringhe. Un altro esempio potrebbe essere `Map<int, String>`.

```
3 void main(List<String> arguments) {  
4     List<String> greetings = [  
5         'Marco',  
6         'Lucia',  
7         'Andrea',  
8         'Anna',  
9     ];  
0  
1 }
```



# Hello world

Utilizzare il loop for per ciclare gli elementi della collection e invocare la funzione che si occuperà di stampare la lista dei nomi con il relativo saluto.

```
3 void main(List<String> arguments) {  
4   List<String> greetings = [  
5     'Marco',  
6     'Lucia',  
7     'Andrea',  
8     'Anna',  
9   ];  
10  for (var name in greetings) {  
11    helloUser(name);  
12  }  
13 }  
14  
15 void helloUser(String name) {  
16   print('Hello $name!');  
17 }
```

# Dart console

Utilizzando il template console si realizzano dei programmi da eseguire in terminale.

Affinchè ciò sia possibile, si utilizza la libreria stdio.

# Dart console

In questo esempio si realizza un programma che consente di inserire dati in console come nome e età, mostrando poi un messaggio contenente i dati inseriti.

```
void main() {  
  stdout.write('Enter your name: ');  
  String name = stdin.readLineSync()!;  
  print('Hello, $name!');  
  
  stdout.write('Enter your age: ');  
  int age = int.parse(stdin.readLineSync()!);  
  print('You are $age years old.');
```

The background of the slide features a series of overlapping, wavy bands in various shades of blue and purple. These bands flow horizontally across the frame, creating a sense of movement and depth. The colors range from light, airy blues to deep, rich purples and blues. The central area of the slide is a plain white space where the text is located.

# Domande e approfondimenti

The background features a series of overlapping, wavy bands in various shades of blue, purple, and pink, creating a sense of movement and depth. The colors transition from deep blues and purples at the top and bottom to lighter, more ethereal tones in the center.

# Attività

# Attività

Creare un'applicazione con template console che, peschi casualmente un numero. Si dia la possibilità di inserire un numero nella console e si mostrino dei messaggi in relazione al valore del numero. Se è maggiore o inferiore rispetto al numero random si avranno dei messaggi che comunicano tale informazione. Se il numero è corretto mostrare un messaggio di congratulazioni.

```
Built first_project:first_project.  
Indovina il numero tra 1 e 100:  
2  
Complimenti! Numero vincente: 2
```

```
Indovina il numero tra 1 e 100:  
1  
Spiacente, il numero scelto è troppo basso - 2
```