# KOTLIN

Linguaggio staticamente **tipato**.  
Sistema di inferenza di kotlin decide il tipo del valore nel caso in cui non lo si specifica.

* kotlin prova a capire automaticamente il tipo della variabile
  + Se voglio specificare il tipo 🡪 var i: Int = 5;

#### TIPI:

* Int, Short, Long, Byte, Float, Double, Char, Boolean
* Immagine che contiene diagramma, Piano, linea, schematico

  Descrizione generata automaticamenteClass **Any** → root della gerarchia
* Class **Nothing** → bottom della gerarchia
  + contiene null
* Class **Nothing?** → Non può contenere nulla, lì per completezza

**Se seguiti dal ? sono nullabili**, altrimenti no

**?.** → safe-call operator: se il valore non è nullo, si traduce in . ; Altrimenti ritorna null.  
**?:** → se quello di sx e null, valore dx

**NOTA**: Se dichiaro var b=null; lo legge come un Nothing?

Immagine che contiene testo, schermata, Carattere, documento

Descrizione generata automaticamente

##### Dichiarazione di variabili

* **var** i=5; → dichiaro variabile e il tipo viene dedotto
* **val** i=5; → value non riassegnabile 🡪 costante

**NOTE**:

* Le variabili sono salvate **sull’heap**.
* Posso cambiare il valore delle variabili ma non il tipo (Anche se non l’ho specificato)
  + Posso fare var i=5; i=2;
  + Non posso fare var i=4; i=2.2;
* Esiste un tipo **Unit** che viene usato per le funzioni che non tornano nulla

Immagine che contiene testo, schermata, Carattere, documento

Descrizione generata automaticamente

**NOTE ULTERIORI**

* **println**(i) 🡪 stampo valore della variabile i
* **val** equivale ad una costante ma lo Stringbuilder permette di fare operazioni particolari come l’append
* **val** s= StringBuilder("Hello"); s.append(" world"); println(s); -🡪 Hello world
* se dichiaro una variabile var al di fuori delle funzioni questa sarà **globale** e vivrà sempre.

#### Funzioni

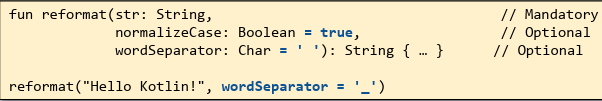
Immagine che contiene testo, Carattere, calligrafia, schermata

Descrizione generata automaticamente**Funzione** test a cui passo un intero e torna una stringa   
 Appende l’intero alla parola “test”

Immagine che contiene testo, Carattere, schermata, calligrafia

Descrizione generata automaticamente***chiamata*** *della funzione*

**valore opzionale**→ se non viene passato l, assume valore 0.0

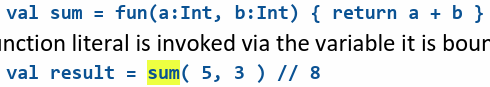
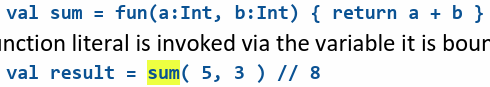


**NOTA**: in fase di chiamata della funzione, posso specificare i nomi dei parametri a cui sto assegnando il valore

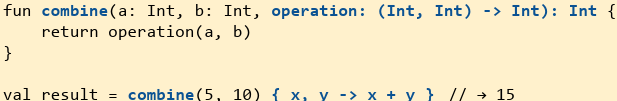
Posso **assegnare una funzione ad una variabile** (f) in base al valore di una variabile. Quindi, usando f, chiamo la funzione decisa nell’if   
  
  
**NOTA**: *assegnando un valore ad una variabile, perdo la possibilità di usare i valori opzionali*

*\ -slide 15, terzo punto*

**Function literal:** blocco fun anonimo, invocato tramite variabile



**LAMBDA expression** 🡪 *val f= {i:Int, d:Double -> "Test3{$i\*d}"};*

**NOTA**: Se ho un solo parametro per la lambda, posso non esprimere in modo esplicito il nome e il tipo del parametro e chiamarlo it

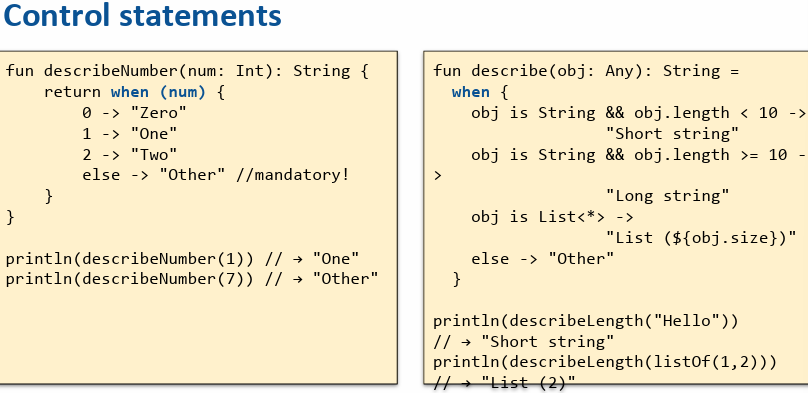
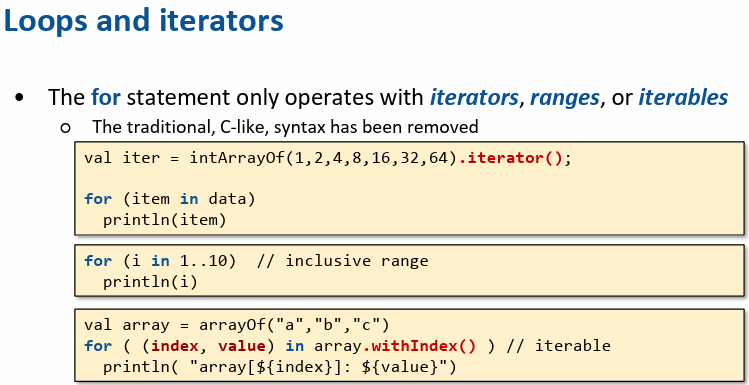
Posso specificare operazione in fase di chiamata

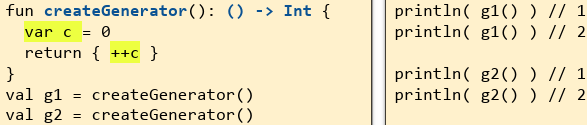
*lambda function hanno la capacità di tirare fuori gli elementi*

**funzione che riceve un int, un double e una funzione e torna una stringa**

* Immagine che contiene testo, Carattere, schermata

  Descrizione generata automaticamentedefinizione
* chiamata



**CLOSURE**: funzioni in grado di catturare lo stato dell’environment (incluse variabile e oggetti)

* Estende ciclo di vita delle variabili → continuano ad esistere anche quando funzione ritorna

### USER DEFINED TYPES

* Classi
* Interface

**NOTA**: se non si specifica la classe padre → assegnato Any

Non ci sono getter e setter 🡪 tutto ciò che abbiamo sono **proprietà**

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

**Immagine che contiene testo, Carattere, schermata, calligrafia

Descrizione generata automaticamentePrimary constructor**: dichiarazione delle minime cose che devo fornire per creare un qualcosa → parametri della funzione necessari

* Devo passare determinati valori per creare

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente*(in questo caso var brand: String)*

Se non specifico il tipo (var/val), quello che passo è un semplice parametro che sarà usato nell’init → body del costruttore

**Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamenteConstructor secondari:**

* Introdotti da constructor
* Delega il primo step al costruttore primario
* Svolge altre azioni secondarie

Es: viene usato nel caso in cui non si passa nulla alla funzione

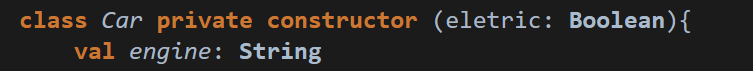
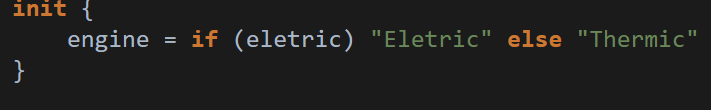
Volendo posso avere un costruttore primario privato

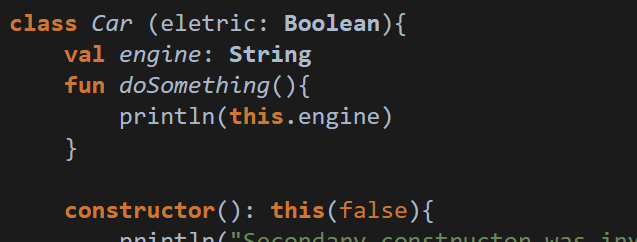
Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Se serve fare inizializzazioni all’interno, devo creare un blocco init

Metodi

Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamente

### PROPRIETÀ

* **Immagine che contiene testo, schermata, Carattere

  Descrizione generata automaticamenteval** → read/write
* **var** → read/only
* getter e setter possono essere esplicitamente overriden se necessario aggiungendo altri get() o set(…) dopo la dichiarazione

Immagine che contiene testo, schermata, Carattere, ricevuta

Descrizione generata automaticamenteVari tipo di **visibilità**:

* Se omessa → **public**
* **Private** → acceduta solo dentro lo stesso file/classe che l’ha definita
* **Protected** → classe o sottoclasse *(non fuori dalla gerarchia di classe)*
* **internal** → stesso modulo

*NOTA informale: per ogni classe max 7 attributi altrimenti impazzisco e non mi ricordo più niente*

**LATEINIT var** : usati quando il valore della variabile non è disponibile quando costruisco l’oggetto

* Classi creati dai framework (activity, etc)
* Costruttore di queste classi può essere senza parametri

Perché c’è un momento in cui esistono ma nessuno ancora le conosce, sa a cosa servono  
🡪 prima che siano completamente funzionali

Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamente*Diverso da null in quanto sono non disponibili all’inizio ma quando lo diventano, non potranno essere null   
🡪 perciò non posso dichiararle come nullabili 🡪 le dichiaro come lateinit*

Es: La mia attività ha sicuramente un bottone ma non posso ancora inizializzarlo

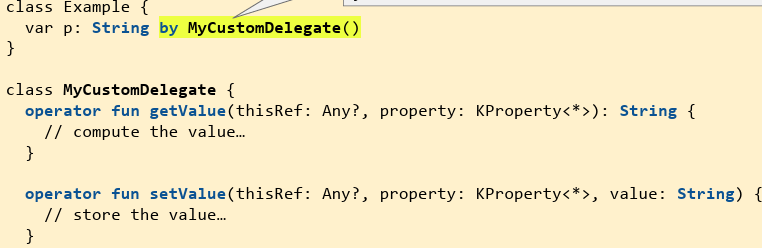
**DELEGATES**

*Le proprietà sono sofisticate in android, ogni istanza della classe car stora una stringa con il nome del brand.*  
Ci sono delle extra keyword che possono essere delegate 🡪 **mantenute da qualche altra parte.**

Comportamento completamente differente:

* Immagine che contiene testo, Carattere, schermata

  Descrizione generata automaticamentevariabile **lazy**: se store value è molto costoso da computing e nella maggior parte dei casi non è mai usato
  + funzione che accetta una lamda come parametro
    - non computa il tutto, fa solo se serve
    - la prima volta che provo ad accedere al contenuto della lazyValue, questa invoca la funzione lambda che riceve, stora il valore e poi mi da il valore
* **Immagine che contiene testo, schermata, Carattere

  Descrizione generata automaticamenteobservable**: posso creare un qualcosa, devo fornire un valore iniziale e una lambda che sarà invocata automaticamente quando cambia il valore
  + reagisce facilmente al cambiamento
  + lambda ha 3 parametri:
    - proprietà
    - old value
    - new value
* **vetoable**: lambda invocata prima del cambiamento del valore
  + per esempio, può bloccare la modifica di un valore
* **custom delegate**
  + posso creare i miei delegati che fanno ciò che voglio io

**METODI**: funzioni definiti con lo scope di una classe

* definiscono comportamento delle istanze della classe
* invocabili con **dot notation**→ someObject.someMethod(param1, param2)
* possono avere una rappresentazione short → fun sum(A:Int, b:Int) = a+b

OPERATOR **OVELOADING**

Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamenteRidefinire il comportamento di un operatore

In questo caso cambia il significato dell’operatore +

**COMPARISON**

**Equity**

* **==** → valori
* **===** → stesso oggetto (pointer)
* Equality interagisce con la logica cìgenerale e con l’inheritance
* Si può implementare un **equals ad hoc**, facendo attenzione che rispetti le proprietà base
* Ogni classe che si crea deve essere in grado di essere comparata con Any

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Nota: questa classe non è estendibile, se voglio estenderla, devo dichiararla open

→dichiaro come open quindi estendibile

Immagine che contiene testo, Carattere, schermata, tipografia

Descrizione generata automaticamente→ classe ElectricCar dichiarata come estensione della classe Car

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamenteImmagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Usare JavaClass è possibile se si usa Java ma non Javascript

**HASHCODE**: ridurre il contenuto di un oggetto ad un numero

* Implementare containers come hashMap o hashSet che beneficiano del fatto che possiamo **ridurre lo spazio di ricerca** per scoprire se un elemento è presente controllando direttamente l’hashcode.
* Implementare hashcode causa la responsabilità di garantire che ogni volta che due elementi sono uguali, lo devono essere anche gli hashcode *(non è detto il reverse)*

**ORDER TESTING**

I tipi primitive possono essere comparati usando i normali operatori di ordine che effettuano un ordinamento:

* Numerico per numeri
* Alfabetico per stringhe

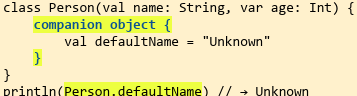
**Altri tipi possono essere comparati se implementano l’interfaccia Comparable:**

se due object sono equals → comparTo deve ritornare 0

**SINGLETON OBJECT**

In alcune situazioni, vogliamo essere sicuri che per **una data classe esiste una sola istanza**.  
Se non ha parametri e non mi interessa averlo Lazy, posso dichiarare come un **oggetto**.

*In alcuni casi vogliamo che singleton sia Lazy: non posso semplicemente usare object invece che class*

**COMPANION OBJECT** → definisce l’insieme di proprietà e metodi statici di una classe

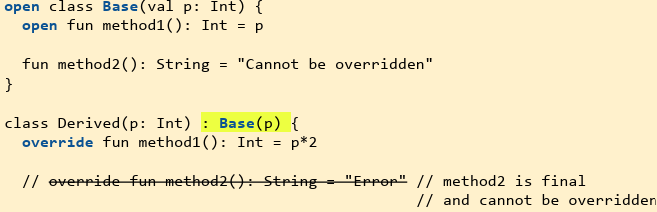
* Creato come viene creata la classe

**Immagine che contiene testo, schermata, Carattere

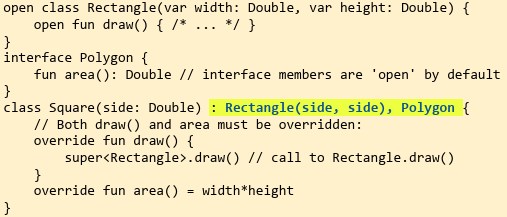
Descrizione generata automaticamenteEXTENSION FUNCTION** → possono essere usate per aggiungere funzionalità ad una classe esistente senza modificare il codice della classe sorgente.

**INHERITANCE**

Ogni cosa deriva dalla classe Any se non nullable; dalla classe Any? Se nullable.  
Di default, inheritance è disabilitata → una classe non può essere estesa se non lo dico esplicitamente dichiarandola come open.

Quando faccio comparazione, devo controllare se l’altro ha la mia stessa classe. Nel caso in cui lui sia una sottoclasse della mia, ho problemi perché restituirebbe vero nel controllo se è nella mia classe ma in realtà non siamo proprio nella stessa classe. (io sono un frutto generico, lui è una mela).  
Questo problema si ha solo con le classi dichiarate open, perché nel caso opposto non puoi creare sottoclassi.

Classe derivata può rimpiazzare l’implementazione dei metodi dichiarati come open, usando la keyword ovverride. Se un metodo non è dichiarato come open, non si può ovveridare.

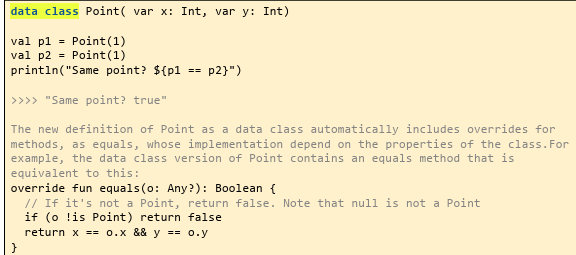
Ogni classe ha una e una sola superclasse e 0+ interfacce implementate:

* Superclassi e interfacce vanno specificate dopo il costruttore primario

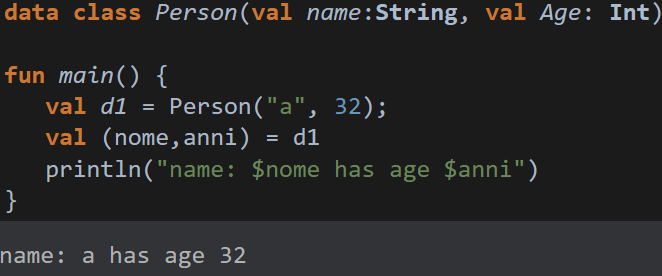
Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamente**DataClasses:**

classe designata a **storare dati**

* Deve avere almeno (at least) 1 parametro
* Tutti i parametri dei costruttori primari devono essere marcati con val o var
* Non possono essere abstract, open, sealed, ineer

Forniscono molte **funzionalità**:

* Equals(…) basato sui valori effettivi e non su indirizzi
* hashCode()
* toString
* component1(), … , componentN()
* copy(…)

**Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamenteEnum Class**

Per definire un insieme di possibili valori

Immagine che contiene testo, schermata, Carattere, algebra

Descrizione generata automaticamenteFittano molto bene con le istruzioni when

Nota: se accio un when con un enum e successivamente aggiungo un elemento ad enum e non al when 🡪 compile error finchè non riallineo

**Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamenteSEALED** **CLASS**: classi che possono essere estese solo da un’altra classe definita nello stesso file sorgente

* Open di default per le classi definite nel file sorgente corrente
* Può avere delle sottoclassi (Success, failure)

Immagine che contiene testo, Carattere, schermata

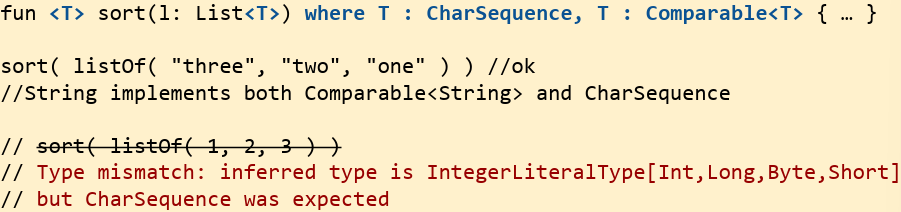
Descrizione generata automaticamenteUna sealed class è un’unione dei domini delle sue sottoclassi. Se una sottoclasse non ha attributi, può essere rimpiazzata dai data object.

**GENERICS:**

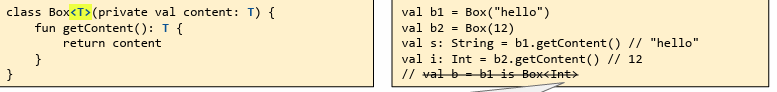
Paradigma di programmazione che permette di definire una classe o una funzione che può lavorare con diversi tipi di dato:

* Non cambia se ho numeri, stringhe o altro
* Functions, classes, interfaces fanno riferimento ai loro parametri/proprietà senza far riferimento al tipo specifico ma a <T>.

Due implementazioni:

* Qualche nome 🡪 come fa C
  + Compilatore genera una versione della mia classe generica, specializzata per il tipo che ho appena dichiarato
  + Se uso una lista di 3 diversi tipi, il compilatore genera 3 copie della funzione indipendenti tra loro
* **Type erasure** → cosa fa kotlin
  + Kotlin genera codice per Any?
    - Quando compilo posso prendere nota del tipo specifico per controllare validità del programma
    - Quando faccio runtime non posso far nulla a riguardo
      * Non riesco a capire se è una lista di numeri o char

Posso specificare che voglio Char Sequence → funzione accetta solo charSequence

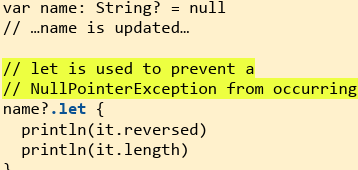
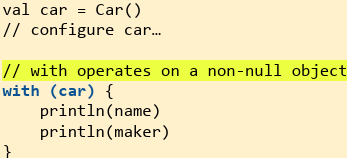
Tipo per classe.

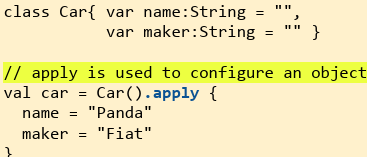
**Variance** in classe generiche, volendo, si può specificare l’uso del tipo

* **In** → classe prende T come un input
* **Out** → classe sputa fuori T come un ouput

*Nota: se non specificato, può essere usato in entrambi i modi*

**SCOPED FUNCTION**

Funzioni che eseguono un blocco di codice nel contesto dell’oggetto

* Let …
* Run …
* With

Eseguono un blocco di codice e modificano l’oggetto

* **Immagine che contiene testo, schermata, Carattere

  Descrizione generata automaticamenteAlso**
* Immagine che contiene testo, Carattere, schermata

  Descrizione generata automaticamenteImmagine che contiene testo, Carattere, schermata, linea

  Descrizione generata automaticamente**Apply**

Immagine che contiene diagramma, schizzo, Piano, disegno

Descrizione generata automaticamente

**Checking e casting** →

In alcuni casi è necessario accedere al tipo del valore salvato in una variabile

* Operatore **is**/**!is**

Se si vuole castare un valore ad un tipo differente si può usare l’operatore **as**

* Se possibile, trasforma nel cast richiesto; altrimenti throw an exception.

Se si vuole usare un modo più safe, si può usare **AS?:**

* ritorna un puntatore del tipo corretto se ok; altrimenti ritorna null

**SMART CAST:**

applicato solo in qualche situazione, casta automaticamente.

Solo se il compilatore può garantire che il campo non può essere mutato dopo il suo check

* Val local variables → sempre fatto
* Val properties → se la proprietà è provata o interna o il check è svolto nello stesso modulo in cui è dichiarata tale proprietà
* Val local variables → se la variabile non è modificata tra il check e l’uso, e non è catturata da una lambda che la modifichi
* Var properties → mai svolto

**Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamenteCollection library**:

Si basa principalmente su 3 classi java:

* **Map**
* **List**
* **Set**

Top dela gerarchia → **Iterable** → fornisce gli elementi uno alla volta (valore successivo o null se finito)

* Può vedere il contenuto ma non modificarlo

**Mutable iterable** → può modificare l’oggetto su cui sto iterando

**Collection** → posso vedere più cose, dettagli (es: size)

**Mutable collection** → posso rimuovere/Aggiungere elementi alla collezione senza iterarci

Tipi di collection

* **List**: garantisce l’ordine, posso avere duplicati
  + Mutable se modificabile (può crescere o dimunuire)
* **Set** : non ho controllo sull’ordine degli elementi, non possono esistere duplicati
  + Mutable se modificabile (può crescere o dimunuire)

**Map**: associa chiave a valore

* Mutable se modificabile (può crescere o dimunuire)

**CREAZIONE**

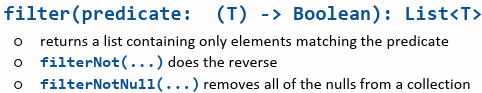
* Immutabili
  + Per map 🡪 Pamela to 26
* Mutabili

**Metodi di manipolazione** → non cambiano l’originale ma ce ne **forniscono una copia** modificata

* Immagine che contiene testo, Carattere, schermata, tipografia

  Descrizione generata automaticamenteMetodo per filtrare

**FILTER**

****

****Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamente **FIND**

* **Immagine che contiene testo, Carattere, schermata

  Descrizione generata automaticamenteImmagine che contiene testo, Carattere, schermata

  Descrizione generata automaticamenteMIN**
* **MAX**

Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamente

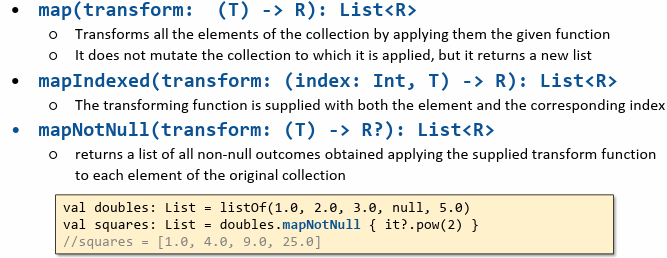
* **FOLD** → valori intermedi andando avanti fino alla fine della collezione quando li avrò letti tutti

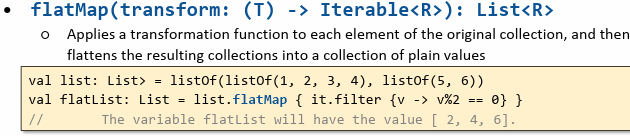
Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamente

Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamente

**MAP**

**FLAT MAP**

MAP:

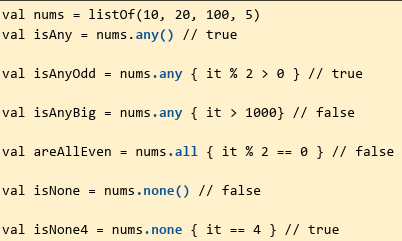
* **Map**
* **mapIndexed**:
  + index → numero
  + element
* **mapNotNull** → applica una funziona, se la funziona ritorna null → elimina elemento

**GROUP BY**: opera sulla collezione, ritorna un Map

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

**PARTITION** 🡪 divide in maniera booleana → 2 sottogruppi

**METODI BOOLEANI:**

* any:
* all:
* none(…)
* none() 🡪 ritorna vero solo se la collezione è completamente vuota

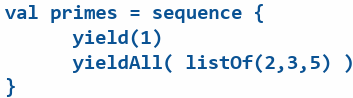


**SEQUENCE**

Container iterabile che opera Lazy, potenzialmente infiniti elementi

* quando una sequence va sotto un processo multi-step, gli elementi sono estratti uno alla volta

**CREARE SEQUENZA:**

* Da elementi
* Da ogni iterable
* Da una funzione
* Da chunk

→ lavora solo quando arriva a toList(). Finchè non capisce che serve non fa niente.

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente