GIOVANNI PELLEGRINI

Guida all'uso di Mit App Inventor 2

Una semplice guida in italiano che illustra il funzionamento di MIT App Inventor ambiente di sviluppo per la creazione di App Android originariamente ideato all'interno dei laboratori di Google ed oggi sviluppato dal Massachusetts Institute of Technology di Boston

2024

Dispensa didattica

Indice

1. Introduzione a MIT App Inventor

Questa guida è orientata a offrire una sintetica panoramica riguardo le principali funzionalità della seconda edizione di **MIT App Inventor**. Lo scopo di questa trattazione sarà quello di guidare il lettore ad un immediato utilizzo della piattaforma mediante una serie di esempi pratici (tratti per lo più dalla documentazione ufficiale del progetto) che hanno lo scopo di illustrare le potenzialità e le metodiche proprie di questo semplice, ma altrettanto potente, ambiente di sviluppo.

Attraverso tale strumento software, infatti, i ricercatori del Massachusetts Institute of Technology si propongono di rendere il più semplice, intuitivo e rapido possibile il processo di realizzazione di una **app** (ovvero di un'applicazione destinata a essere eseguita su dispositivi di mobile computing, come smartphone o tablet) per sistemi operativi di tipo **Android** (prodotto da **Google**).

Grazie ad App Inventor, infatti, la creazione di App Android diventa un processo incredibilmente intuitivo e, come tale, non più riservato ad una stretta cerchia di professionisti ma aperto anche (e soprattutto) a chi è alle prime armi e non ha solide conoscenze di programmazione (normalmente lo sviluppo di App Android richiederebbe una buona conoscenza di Java quale pre-requisito per addentrarsi nel mondo dello sviluppo per Android).

Prima

Google poi il MIT

Originariamente nato all'interno dei laboratori di Big G, App Inventor è stato in seguito (10 Agosto 2011) ceduto alle amorevoli cure del MIT che attualmente si occupa della manutenzione e dello sviluppo della piattaforma che è stata ribattezzata **App Inventor EDU** o più comunemente **MIT App Inventor**.
L'impegno congiunto di due giganti come **MIT** e **Google** desta naturalmente grande interesse, tanto più che la sfida che MIT App Inventor si propone di raccogliere è senz'altro ardua.
Sempre più numerosi, infatti, sono gli ambienti di sviluppo dedicati specificamente alla realizzazione di **app** e molti consentono di adottare un approccio multipiattaforma, ovvero di poter creare sia una versione **Android** che una versione **iOS** delle proprie applicazioni (e, in alcuni casi, anche una versione per **Windows phone**). Diversamente da questi, **MIT App Inventor** si propone invece come un software dedicato esclusivamente alla piattaforma **Android** ma, nonostante questo "limite" (se così lo si può definire), il progetto dl MIT ha raggiunto un grandissimo successo potendo vantare quasi due milioni di utenti in 195 Paesi.

Le diverse versioni di MIT App Inventor

A partire dal 3 dicembre 2013 è disponibile all'interno del sito del MIT la versione 2 di App Inventor che è andata a sostituire la precedente versione Beta (in seguito ribattezzata App Inventor 1 o App Inventor Classic). Questa guida tratterà appunto di questa nuova versione del software caratterizzata da una serie di migliorie oltre, ovviamente, alla correzione di numerosi bug ed instabilità che affliggevano la prima release dell'ambiente di sviluppo.

Caratteristiche principali di MIT App Inventor

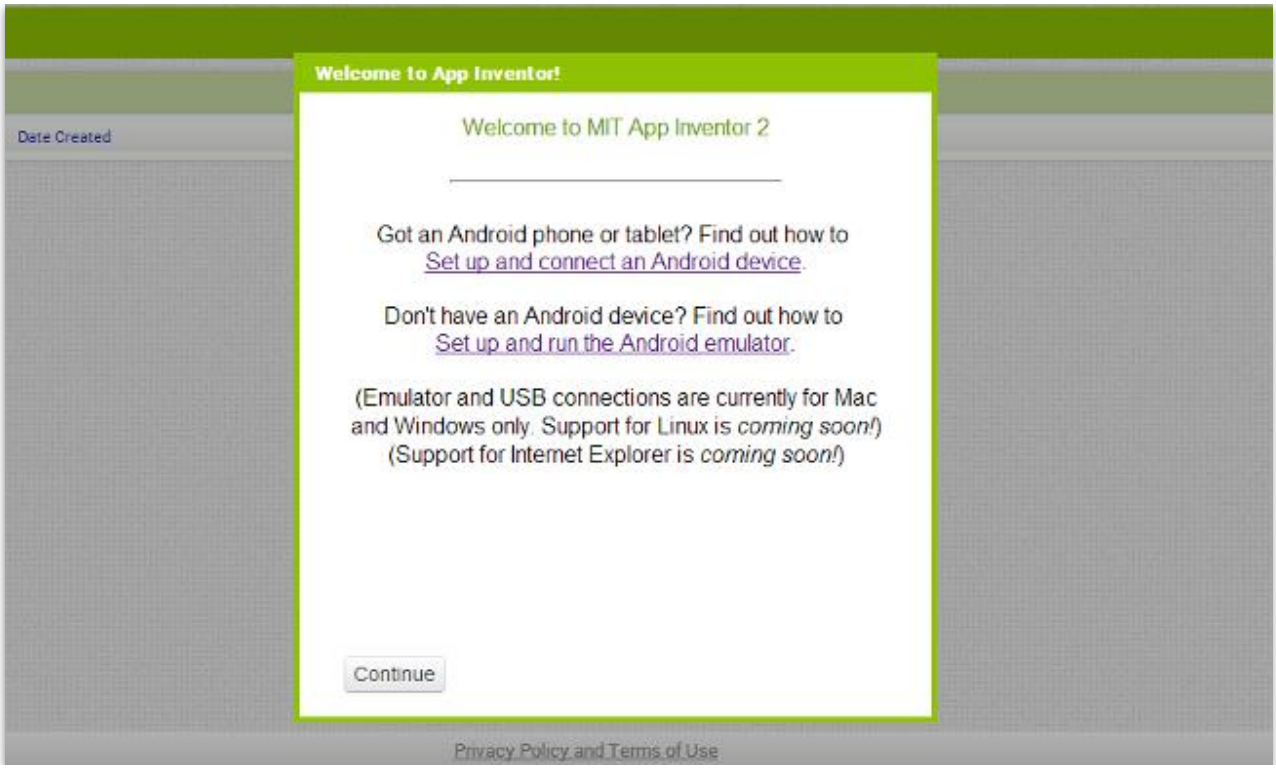
Prima di prendere confidenza con la creazione e la gestione di un progetto **MIT App Inventor**, destinato alla realizzazione di **app** descriviamo in modo sintetico ma puntuale le **feature** capaci di rendere tale software una scelta appetibile per i programmatori.
MIT App Inventor si propone come punto di riferimento soprattutto per chi desidera programmare la propria **app**, ma non è in possesso di una preparazione specifica dal punto di vista informatico. La potente interfaccia grafica di **MIT App Inventor** consente di organizzare ogni aspetto della propria **app** senza la necessità di scrivere codice, rimuovendo un importante ostacolo per i non addetti ai lavori.
Oltre alle funzioni base (ad esempio aggiunta di **pulsanti**, **caselle di testo**, **animazioni**), sono messe a disposizione, sempre tramite **interfaccia grafica**, anche funzionalità più complesse, come ad esempio l'integrazione con **social network** come **Twitter**.
Non si tratta però soltanto di questo: **MIT App Inventor** rappresenta un'opportunità importante anche per i programmatori più esperti. La potenza dell'interfaccia grafica può infatti essere molto utile anche per loro, per risparmiare tempo e velocizzare alcuni processi di sviluppo grazie all'integrazione della libreria Java **Open Blocks** mediante la quale è possibile gestire processi di programmazione in modo visuale attraverso semplici operazioni all'interno della GUI.

Nella prossima lezione della nostra guida vedremo come installare l'ambiente di lavoro e come avviare la creazione del nostro primo progetto.

2. Installazione di MIT App Inventor

Setup di MIT App Inventor

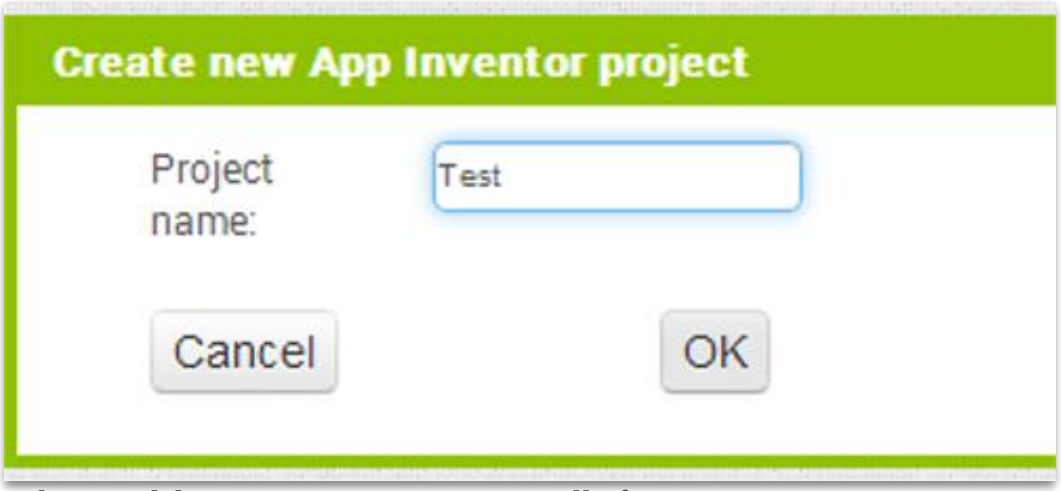
Come il sito ufficiale riporta, la soluzione ideale per il setup (facendo riferimento al sito, **Option One - RECOMMENDED**) coinvolge l'utilizzo di un dispositivo **Android** (ad esempio uno smartphone) su cui testare, a mano a mano che le si sviluppano, le **app**.
Per poter rendere fruibile questa guida anche a chi non fosse dotato di un dispositivo **Android** però, scegliamo invece la seconda opzione, tra quelle indicate. Si tratta di questo: scaricheremo e installeremo un software che ci consentirà di **emulare** il comportamento di un dispositivo **Android** e ci serviremo poi di tale **emulatore** per eseguire le operazioni di **test** e di **debug** dei nostri lavori
Ipotizzeremo, per descrivere nello specifico la procedura di installazione, di avere a disposizione un sistema operativo di tipo **Windows**, data l'ampia e capillare diffusione di tale OS. Tuttavia, segnaliamo che è possibile, sempre come indicato sulla pagina ufficiale di MIT App Inventor, utilizzare tale software anche con sistemi operativi di tipo **MAC OS**. A breve, l'elenco verrà aggiornato, e sarà possibile servirsi dell'**emulatore** anche con sistemi **GNU/Linux** (attualmente supportati soltanto se è disponibile una connessione con un dispositivo **Android**). Concludendo l'elenco dei requisiti, segnaliamo che, per servirsi di MIT App Inventor sarà necessario essere provvisti di un account **Google**.
La versione **Windows** del pacchetto software che contiene il già citato **emulatore** può essere scaricata a questa pagina. Una volta completata la procedura guidata di installazione, verrà automaticamente avviato il già citato emulatore **aistarter**.
Non resta quindi che recarsi all'indirizzo http://ai2.appinventor.mit.edu e indicare, quando richiesto, le credenziali del nostro account **Google**, per accedere alla schermata di benvenuto di **MIT App Inventor 2** (schermata riportata nella figura sottostante).



Creiamo il nostro primo progetto con MIT App Inventor

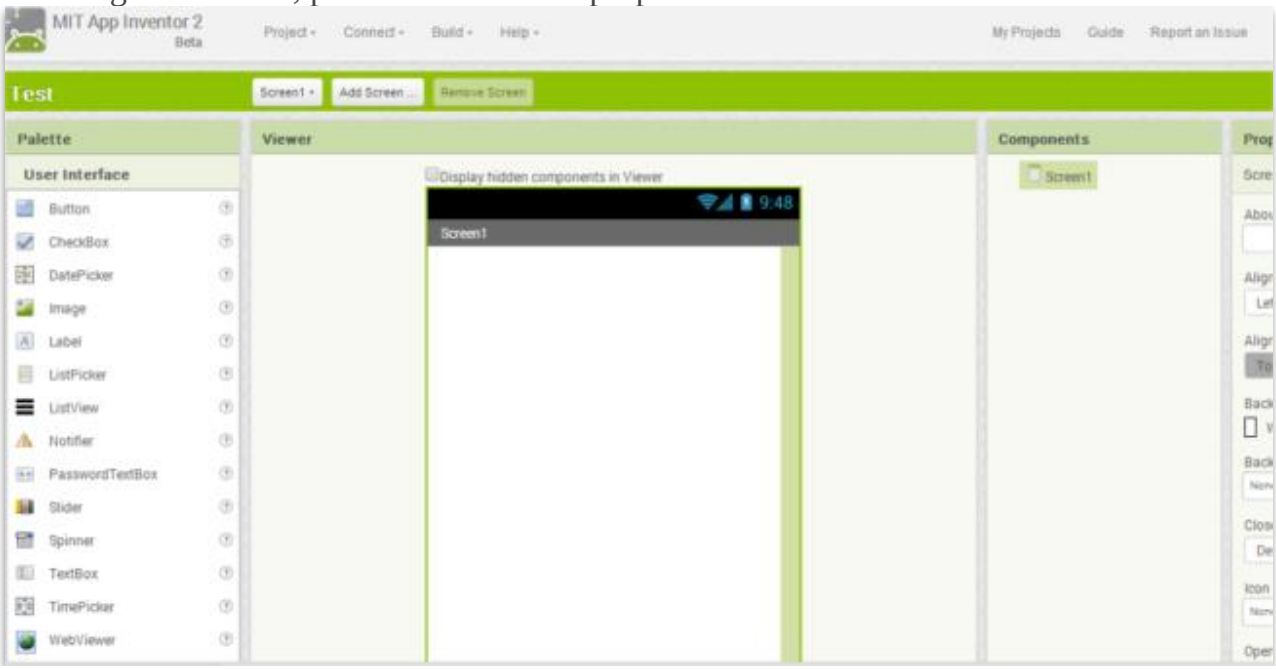
Una volta installato l'ambiente di lavoro possiamo dare uno sguardo alla schermata iniziale. Una volta preso atto del messaggio di benvenuto, potremo procedere (cliccando su continue) alla creazione del nostro primo progetto. A tale scopo clicchiamo sul pulsante New project (in alto a sinistra) e indichiamo Test come nome per il nostro nuovo progetto, come mostrato nella figura sottostante. Clicchiamo poi sul pulsante OK.

Accediamo ora alla **schermata principale** per il controllo del progetto **Test**, cliccando



proprio sul nome del progetto, come mostrato nella figura sottostante.

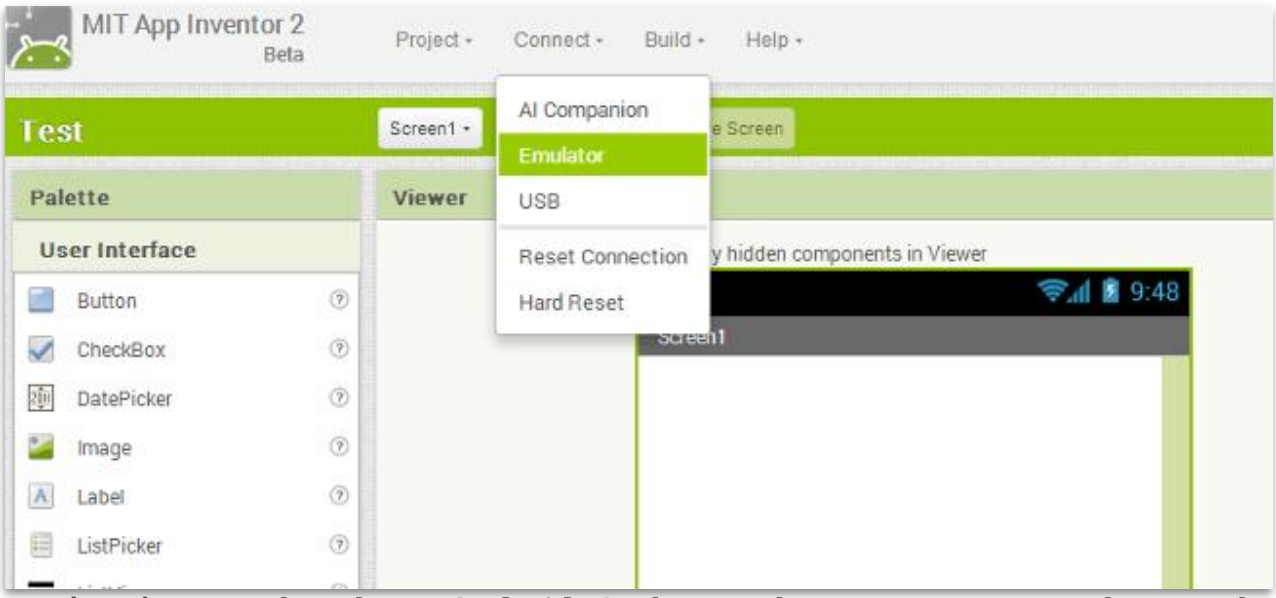
Nella figura in basso, possiamo osservare proprio tale schermata.



Come vedremo, sarà sufficiente trascinare col mouse gli elementi di interesse (ad esempio **Button** o **Image**, che possiamo vedere nel menu a sinistra) all'interno della schermata principale per portarli a far parte della nostra applicazione. Le **proprietà** di tali elementi (ad esempio, l'**allineamento**) potranno poi essere specificate tramite il menu a destra.

L'emulatore Android

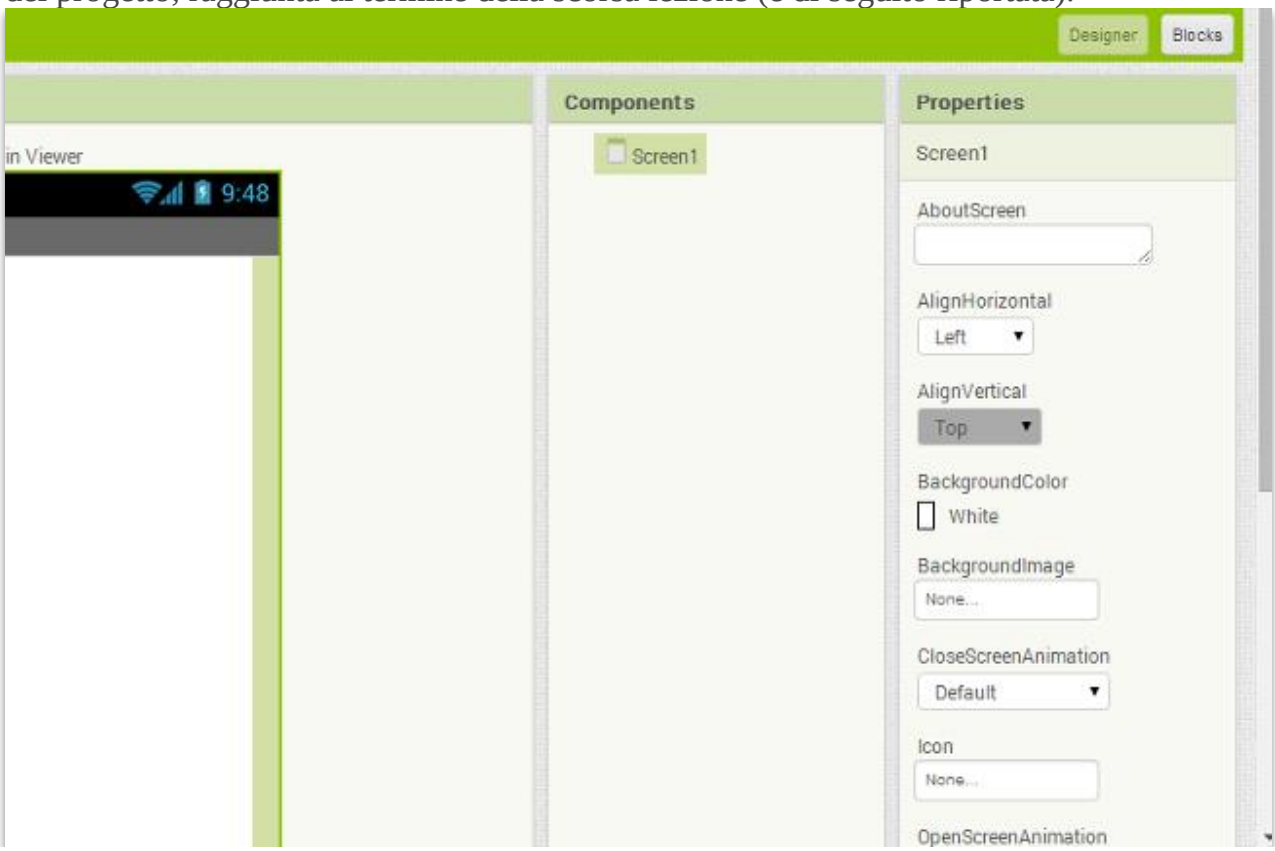
Prima di procedere ad analizzare, nel corso della prossima lezione, quale sia l'architettura di una **app** in **MIT App inventor**, eseguiamo una semplice verifica. Verifichiamo cioè che l'**emulatore** installato nel corso della scorsa lezione funzioni correttamente. A tal fine clicchiamo sull'opzione **Emulator** del menu **Connect**, come mostrato nella figura in basso.



Verrà così avviato l'emulatore **Android**. Qualora si rilevasse un messaggio d'errore, che segnala l'impossibilità di individuare l'emulatore stesso, sarà sufficiente avviare **aiStarter** manualmente, con un semplice **doppio clic** proprio sul collegamento a **aiStarter** inserito sul nostro desktop a seguito dell'installazione operata poco fa.

3. Architettura di una app: components e behaviors

Prima di proseguire, nel corso della prossima lezione, alla realizzazione della nostra prima **app** con **MIT App Inventor**, ci proponiamo di comprendere quale sia la struttura logica di una **app**. Il nostro obiettivo cioè è quello di individuare le sotto-componenti principali che costituiscono una **app**, evidenziandone le **interazioni**.
Prendiamo come punto di partenza della nostra analisi la **schermata principale** di controllo del progetto, raggiunta al termine della scorsa lezione (e di seguito riportata).



Notiamo come, in alto a destra, siano presenti due pulsanti: **Designer** e **Blocks** e come l'opzione selezionata per default, tra le due, sia **Designer**. Quella che stiamo osservando, quindi, è la scheda **Designer** della schermata principale.
Da qui possiamo agire su una delle tipologie di elementi costitutivi della nostra **app**, ovvero le sue **componenti** (**components**).

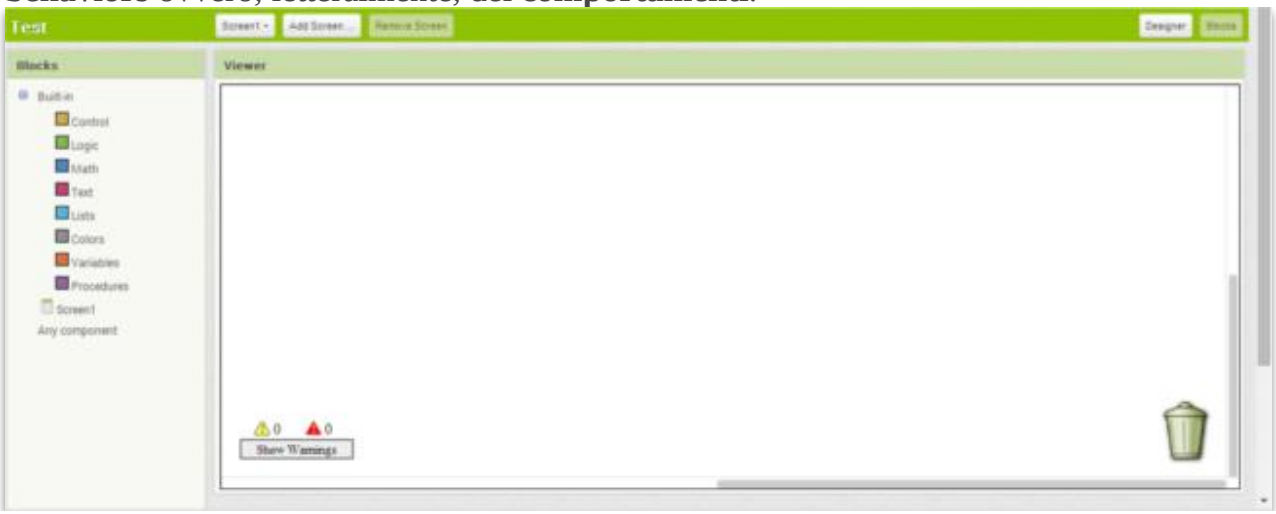
Components

Possiamo pensare alle **componenti** come alle unità minime coinvolte nella costruzione della nostra **app**. Ad esempio, appartengono alla categoria delle **componenti**: **pulsanti**, **caselle di testo**, **immagini**, **suoni**, **video**. Tali elementi sono presenti proprio nella scheda **Designer** che stiamo osservando, in particolare nel pannello a sinistra.

Fanno parte delle **componenti** anche unità (sempre indicate nel pannello a sinistra della schermata principale **Designer**) che non compaiono alla vista dell'utente, ma risultano rilevanti per il programmatore. Ad esempio, la componente **Texting**, dedicata all'elaborazione e all'invio di SMS, oppure la componente **Location Sensor**, utile per conoscere la posizione dell'utente. Faremo riferimento alle **componenti** del primo gruppo (pulsanti, immagini, ecc.) come **visibili** e a quelle della seconda categoria come **non visibili**.

Behaviors

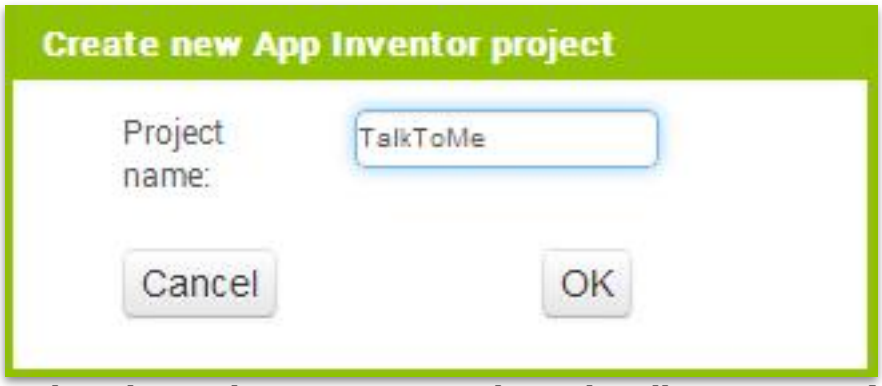
Passiamo ora alla scheda **Blocks** della schermata principale (riportata più in basso), per fare conoscenza con la seconda categoria di elementi che costituisce una **app**. Si tratta dei **behaviors** ovvero, letteralmente, dei **comportamenti**.



Prima di entrare nello specifico per quanto riguarda tale categoria di elementi, osserviamo che il paradigma moderno di programmazione tende a modellare un software non più in termini statici (ovvero come la procedura composta da una sequenza di istruzioni) ma come un'**entità reattiva**. Come un modulo capace, cioè, di qualificare e definire se stesso in funzione delle proprie reazioni a stimoli esterni.
La struttura di una **app** realizzata con **MIT App Inventor** (e non solo) ricalca tale modello. I **behavior** consentono proprio di definire come la nostra **app** dovrà comportarsi a seguito, ad esempio, di un particolare input dell'utente, o dopo un fissato lasso di tempo. Definiremo tali **behaviors** proprio dall'interno della schermata **Blocks** che stiamo osservando, facendo uso in particolare del pannello omonimo (**Blocks**), sulla sinistra.
Abbiamo a questo punto una panoramica più chiara circa la struttura di una **app**: si tratta di un insieme di **componenti** (che possono essere sia visibili che non visibili) che reagiscono, modificando il proprio comportamento, a particolari **eventi**, tramite i **behaviors**. Dopo questa breve introduzione teorica, passiamo senz'altro passare ad aspetti più pratici (e divertenti).

4. Creiamo la nostra prima App

Nel corso di questa lezione, ci dedicheremo alla realizzazione della nostra prima app con **MIT App Inventor**. Si tratta di una app molto semplice che ha lo scopo di introdurre il lettore alle potenzialità di questo ambiente di sviluppo. Rimandiamo, quindi, la teoria alle prossime lezioni e facciamo un tuffo nella pratica. Accediamo dunque per prima cosa alla schermata iniziale del nostro ambiente di sviluppo.
Il nostro obiettivo sarà realizzare una app in grado di riprodurre vocalmente un messaggio predefinito, al clic di un pulsante da parte dell'utente. Prenderemo in questo modo confidenza sia con elementi di base, come la realizzazione di un **pulsante** che con funzionalità più complesse, come la **riproduzione vocale**. Non solo: in questo modo ci sarà possibile utilizzare sia **componenti visibili** (**Button**) che **non visibili** (**TextToSpeech**). Non a caso, gli stessi autori di MIT App Inventor segnalano proprio questa app come una delle più indicate per un primo approccio.
Creiamo quindi un nuovo progetto e scegliamo, per chiarezza, un nome che rappresenti le funzionalità dell'app: ad esempio **TalkToMe**, ovvero "parlami" (v. figura in basso). La creazione di questo nuovo progetto ci offre l'occasione di far presente che è opportuno evitare spazi vuoti all'interno dei nomi dei progetti MIT App Inventor.



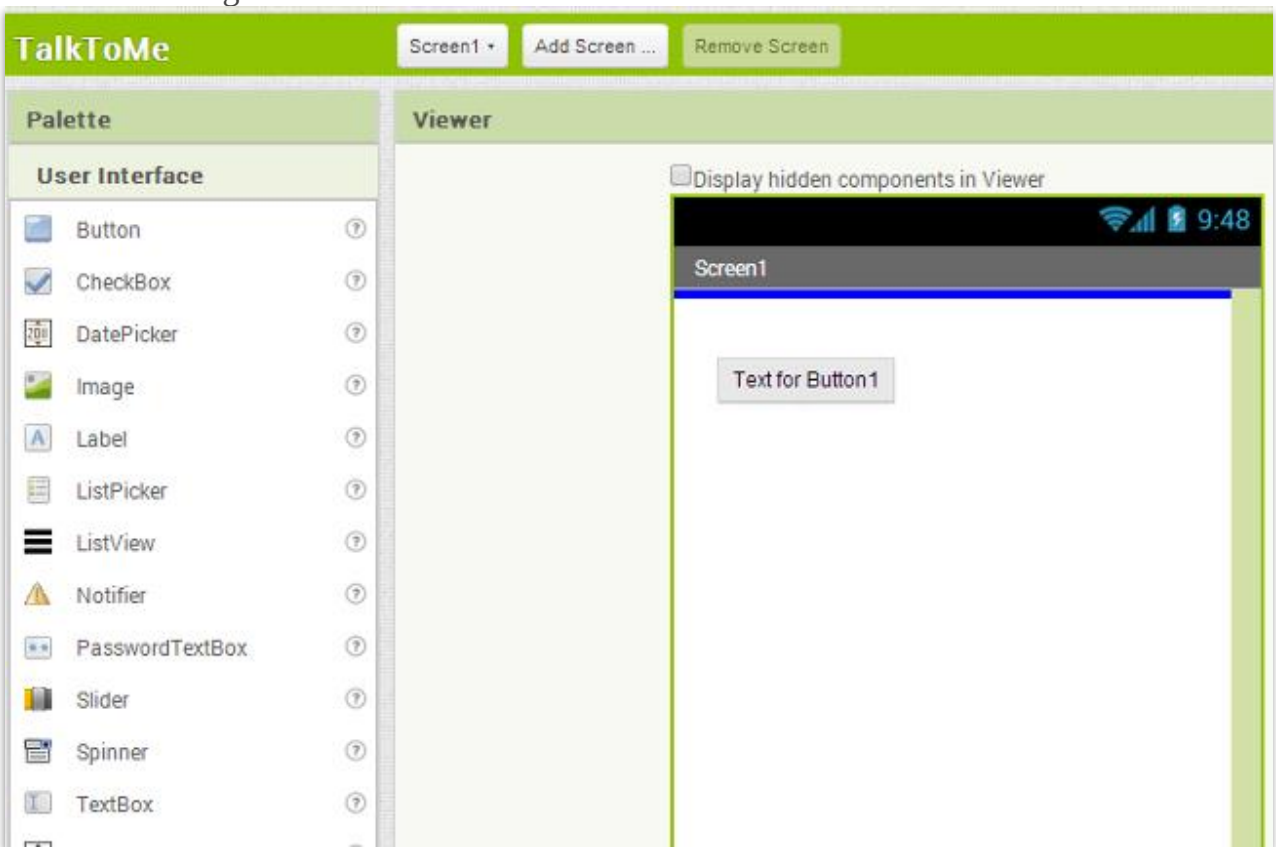
Connettiamo ora l'emulatore al nostro progetto, cliccando sull'opzione **Emulator** del menu **Connect**. Qualora **aiStarter** non fosse stato avviato in modo automatico, potrebbe esserci richiesto di provvedere manualmente (come visto nelle scorse lezioni, tramite un doppio clic sul collegamento ad **aiStarter** situato sul nostro desktop).
Una volta completata la connessione all'**emulatore** verrà aperta automaticamente un'altra finestra, che rappresenta lo schermo di uno smarpthone: utilizzeremo proprio tale finestra per testare il comportamento della nostra app.

Inseriamo i componenti all'interno della nostra App

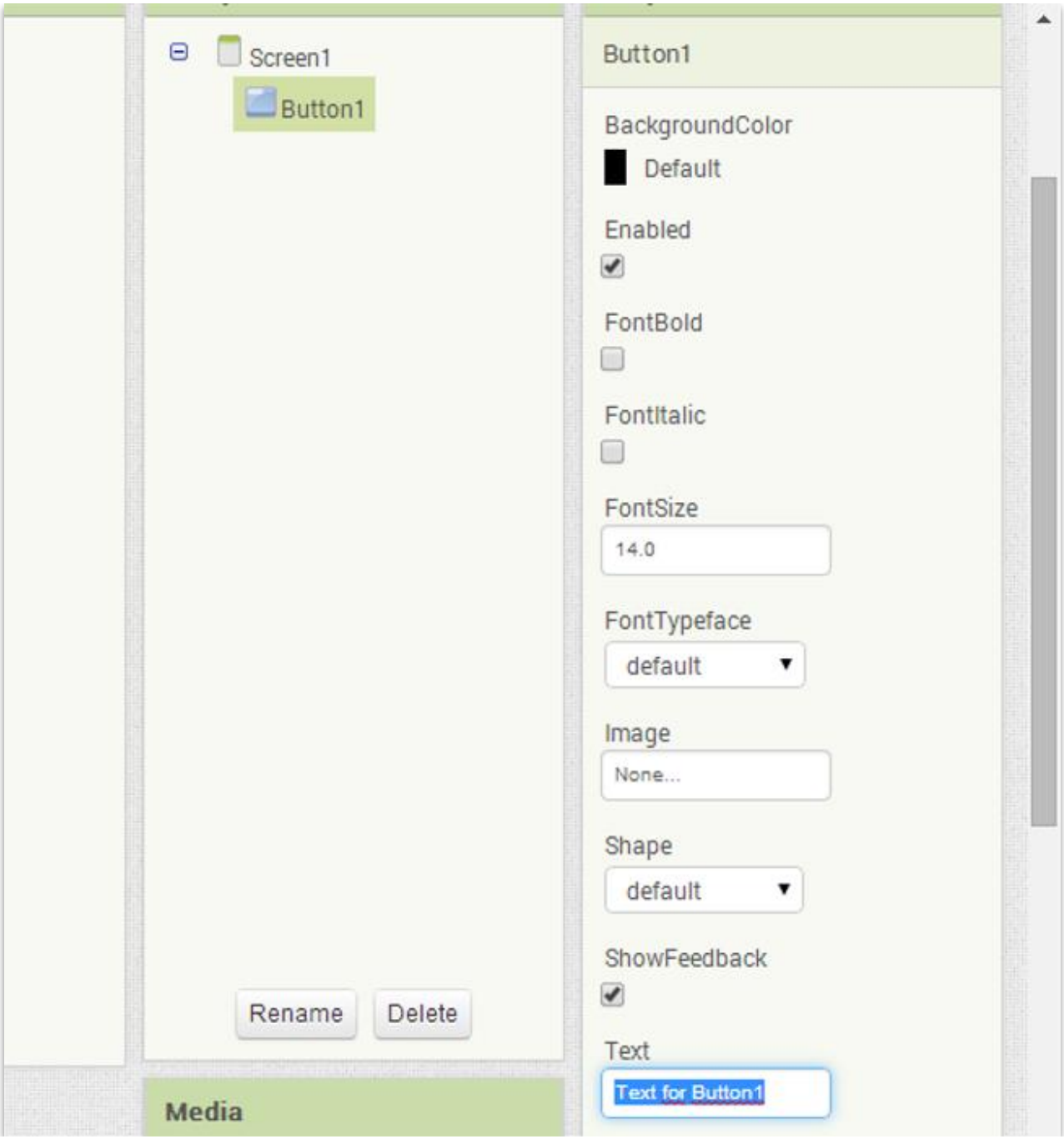
Inseriamo ora le **componenti** di cui abbiamo bisogno:

- un **pulsante** (visibile)
- un **TextToSpeech** (non visibile)

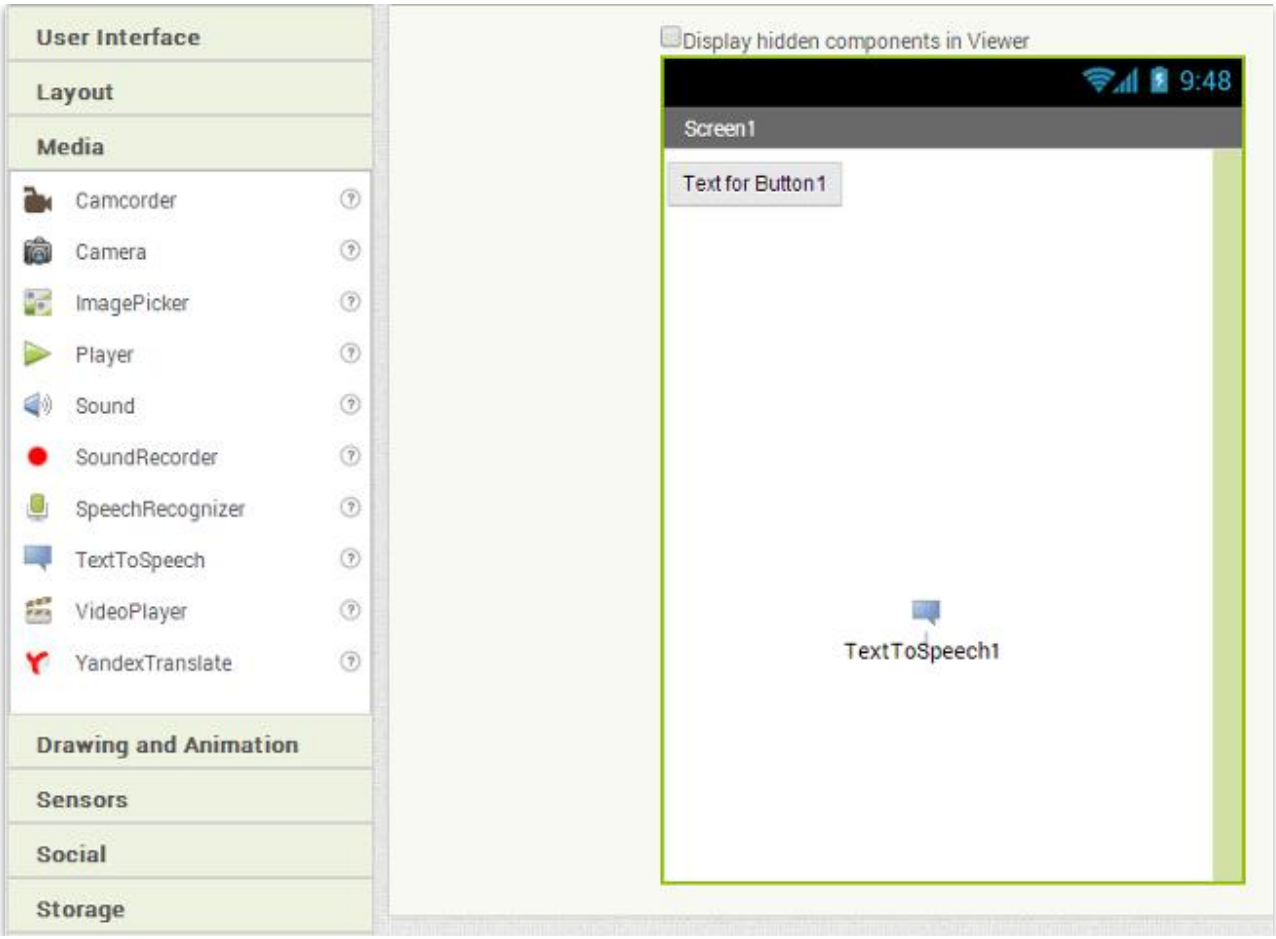
Cominciamo dal pulsante: dalla scheda **Designer** della schermata principale, che stiamo visualizzando, trasciniamo **Button** (menu a sinistra) sul pannello **Viewer** (al centro), come indicato nella figura in basso.



Sullo schermo apparirà un pulsante, con la scritta **TextforButton1**. Si tratta del valore di default per il testo del pulsante.
Sostituiamo tale scritta con **TalkToMe**, selezionando l'elemento **Button** appena aggiunto nel pannello **Viewer** e modificando opportunamente il campo **Text** nel menu **Properties** (a destra), evidenziato nella figura sottostante.



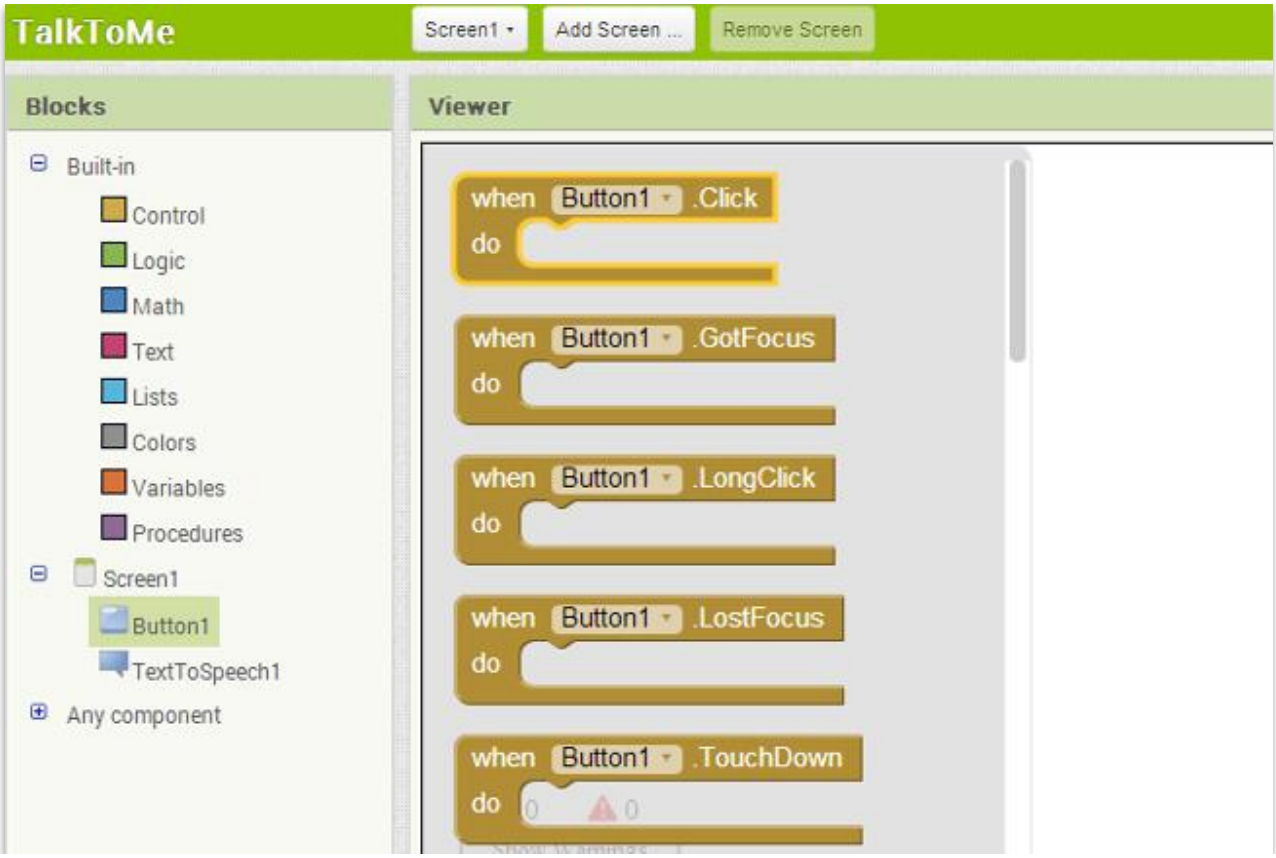
Trasciniamo ora nel pannello **Viewer** anche il componente **TextToSpeech**, individuandolo all'interno del gruppo **Media** nel pannello **Palette** (v. figura in basso).



Definiamo il comportamento della nostra App

Ora non resta che definire il comportamento della nostra app in risposta a eventi: ovvero specificarne i **behavior**. Nel nostro caso: riprodurre un messaggio vocale a seguito del clic sul pulsante **TalkToMe**.

Passiamo quindi alla scheda **Blocks** della finestra principale. Selezioniamo la componente **Button1**, all'interno del pannello **Blocks** (a sinistra). Selezioniamo poi e trasciniamo verso destra, sempre all'interno del pannello **Viewer**, il primo elemento tra quelli comparsi, ovvero **When Button1.Click do** (v. figura sottostante).



Trasciniamo poi, dopo aver cliccato su **TextToSpeech** all'interno del pannello **Blocks** (a sinistra), l'elemento **CallTextToSpeech1.speak message** sul pannello **Viewer**, in modo che la parola **"call"** segua la parola **do** dell'elemento **When Button1.Click do** precedentemente trascinato.

Il significato di tale operazione è creare un collegamento logico tra le due **componenti** ovvero far sì che, ogni volta che viene cliccato il pulsante, venga generato un messaggio vocale. Resta solo da definire il messaggio. A tal fine, trascineremo il primo elemento dell'elenco **Text** dal pannello **Blocks** nel pannello **Viewer**, in modo da collegarlo con **CallTextToSpeech1.speak message**, come mostrato nella figura in basso.

Non resta ora che effettuare un doppio clic sull'ultimo elemento inserito, e digitare poi il messaggio che si desidera ascoltare. La lingua di default è l'inglese: possiamo scegliere, ad esempio "Great job" (ovvero "ottimo lavoro"), per festeggiare la buona riuscita della nostra prima app creata con MIT App Inventor. Possiamo verificarne il corretto funzionamento servendoci della finestra emulatore, ovvero cliccando sul pulsante TalkToMe e ascoltando il messaggio impostato.
Nel corso delle prossime lezioni, decisamente più teoriche, prenderemo in esame vari elementi che possono permetterci di arricchire le nostre app.

5. I componenti di MIT App Inventor

Dopo aver visto, alla lezione precedente, un semplice esempio di una prima App creata con **MIT App Inventor**, passiamo ad approfondire alcuni aspetti teorici già precedentemente introdotti. Torniamo, cioè, sul "duopolio" di **components** e **behaviors**. In questa lezione, più precisamene, ci soffermeremo sulle differenti tipologie di componenti resi disponibili dagli sviluppatori del MIT all'interno della schermata **Designer**.
Prima di passare in rassegna i vari tipi di componenti supportati dalla piattaforma è opportuno precisare che la nostra trattazione non ha l'ambizione di voler essere esauriente ma, più modestamente, si propone l'obiettivo di offrire una panoramica generale, una visione d'insieme, sul vasto numero di components disponibili in MIT App Inventor.

User Interface Components

La famiglia più ricca di componenti è, certamente, quella dei componenti relativi all'interfaccia utente. Si tratta, in sostanza, dei componenti necessari a disegnare e far funzionare l'interfaccia utente. L'esempio tipico di questa tipologia di componenti è offerto dai bottoni. Ogni componente, come avremo modo di vedere, dispone di metodi, eventi e proprietà peculiari (nella prossima lezione approfondiremo questi concetti).
Di seguito l'elenco completo dei componenti di questa famiglia:

- Button - il classico bottone;
- CheckBox - consiste nella classica casella di scelta (true/false);
- DatePicker - apre un pop-up per la selezione di date;
- Image - mostra un'immagine;
- Label - mostra del testo;
- ListPicker - è un bottone che, una volta premuto, apre una lista di opzioni tra cui è possibile scegliere;
- ListView - consente di creare una lista di elementi testuali;
- Notifier - consente di creare delle notifiche;
- PasswordTextBox - campo per l'inserimento di password (al posto dei caratteri digitati vengono mostrati dei puntini);
- Screen - è l'elemento genitore (lo schermo) che contiene tutti gli altri componenti di questa famiglia;
- Slider - genera una barra con all'interno un cursore draggabile
- Spinner - apre un menu di scelta multipla;
- TextBox - campo per l'inserimento di testo;
- TimePicker - apre un pop-up per la selezione di un orario;
- WebViewer - consente di aprire una URL remota.

Layout Components

Offre una serie di componenti che consentono di gestire il layout dell'applicazione. Questa famiglia di componenti offre tre alternative:

- HorizontalArrangement - gestisce diversi elementi affiancandoli orizzontalmente (da sinistra verso destra);
- TableArrangement - consente di creare un layout tabellare;
- VerticalArrangement - gestisce diversi elementi impilandoli verticalmente (dall'alto verso il basso).

Media Components

Questa famiglia di componenti consente di gestire gli elementi multimediali all'interno della nostra applicazione creata con MIT App Inventor. Vediamo l'elenco dei componenti a nostra disposizione:

- Camcorder - apre la videocamera integrata nel device per la registrazione di un video;
- Camera - apre la videocamera integrata nel device per scattare una foto;
- ImagePicker - consente di selezionare un'immagine tra quelle presenti nella galleria del dispositivo;
- Player - consente di riprodurre un file audio e di controllare la vibrazione del device (consigliato per file audio di lunga durata);
- Sound - consente di riprodurre un file audio e di controllare la vibrazione del device (consigliato per file audio di breve durata);
- SoundRecorder - consente di accedere al microfono integrato nel device per effettuare una registrazione audio;
- SpeechRecognizer - consente di attivare la funzionalità di riconoscimento vocale integrata in Android al fine di convertire un parlato in testo;
- TextToSpeech - consente di trasformare un testo in un parlato attraverso un sintetizzatore vocale (tra i vari linguaggi è supportato anche l'italiano);
- VideoPlayer - consente di riprodurre un file video all'interno di un player dotato dei normali comandi attivabili al touch dell'utente;
- YandexTranslate - consente di effettuare traduzioni in tempo reale attraverso le API offerte dal traduttore automatico di Yandex.

Drawing and Animation Components

Si tratta di una serie di componenti per gestire il disegno e le animazioni. Il componente principale di questa tipologia e canvas il quale costituisce la "cornice" all'interno della quale è possibile effettuare disegni ed animazioni. Questo l'elenco dei componenti:

- Ball - crea uno sprite circolare che si muove secondo le proprietà ad esso assegnate;
- Canvas - pannello rettangolare bidimensionale sensibile al touch all'interno del quale è possibile disegnare e riprodurre animazioni;
- ImageSprite - crea uno sprite sulla base di un'immagine che si muove secondo le proprietà ad esso assegnate.

Sensor Components

Si tratta di una serie di componenti attraverso i quali è possibile accedere ai sensori del dispositivo.

- AccelerometerSensor - componente non visibile che intercetta lo shake del dispositivo e ne misura le accellerazioni;
- BarcodeScanner - consente di leggere un codice a barre;
- Clock - componente non visibile che consente di accedere all'orologio del dispositivo;
- LocationSensor - componente non visibile che consente di accedere alle informazioni di geolocalizzazione (latitudine, longitudine, altitudine e indirizzo)
- NearField - componente non visibile che consente di accedere alle funzionalità NFC;
- OrientationSensor - componente non visibile che consente di rilevare l'orientamento del device.

Social Components

Questa tipologia di componenti di MIT App Inventor consente di gestire alcune funzionalità sociali come, ad esempio, l'attivazione di pulsanti di chiamata o l'interazione con Twitter. Segue l'elenco dei componenti disponibili:

- ContactPicker - consente di effettuare la scelta di un contatto tra quelli disponibili nella rubrica del device;
- EmailPicker - consente di inserire una casella di testo con auto-completamento all'interno della quale può essere digitato il nome di un contatto o un'indirizzo email;
- PhoneCall - componente non visibile che consente di effettuare una chiamata ad un numero prestabilito;
- PhoneNumberPicker - mostra i numeri di telefono dei contatti presenti nella rubrica;
- Sharing - componente non visibile che consente di condividere file o messaggi tra l'applicazione ed altre app installate sul device;
- Texting - consente di inviare un messaggio di testo con un messaggio e ad un numero predefinito;
- Twitter - consente di comunicare con Twitter ed effettuare alcune operazioni come, ad esempio, inviare un tweet.

Storage Components

Questi componenti consentono di memorizzare dei dati all'interno del device al fine di poterli riutilizzare nei successivi accessi all'applicazione.

- File - componente non visibile che consente di scrivere e/o leggere un file all'interno del device;
- FusionTablesControl - componente non visibile che consente di comunicare con Google Fusion Tables (un servizio sperimentale di Google che consente di gestire e condividere data tables);
- TinyDB - componente non visibile che consente di memorizzare (e leggere) dati all'interno di una sorta di database (estremamente semplificato) mediante un meccanismo di tags;

- TinyWebDB - componente non visibile che consente di interagire con un web-service per la memorizzazione (e la lettura) dati in remoto.

Connectivity Components

Si tratta di una serie di componenti attraverso i quali gestire le possibilità di connessione dell'applicazione. Questi i componenti disponibili:

- ActivityStarter - consente di lanciare una nuova activity (ad esmepio è possibile aprire il browser con una pagina preselezionata o effettuare una ricerca su Google);
- BluetoothClient - consente di attivare un client Bluetooth;
- BluetoothServer - consente di attivare un server Bluetooth;
- Web - componente non visibile attraverso il quale è possibile inoltrare chiamate HTTP mediante GET, POST, PUT e DELETE.

Lego Mindstorms

Per finire ricordiamo che, tra i componenti di MIT App Inventor 2, si annovera anche una famiglia particolare. Si tratta dei componenti LEGO MINDSTORMS che consentono di interagire, attraverso Bluetooth, coi robot della famiglia MINDSTORMS prodotti dalla LEGO.

6. Eventi, metodi e proprietà

Dopo aver offerto una rapida panoramica sui componenti di **MIT App Inventor**, a partire da questa lezione ci occuperemo del funzionamento del pannello **Blocks** ovvero come gestire i cosidetti **behaviors**. In pratica ci occuperemo di **programmazione visuale**, cioè di come creare un programma utilizzando i **blocchi** colorati di **MIT App Inventor**.

Eventi

In una precedente lezione, anticipando un po' i tempi, abbiamo già visto come creare una prima app ed abbiamo già visto come gestire un evento (la pressione di un bottone) e come associare ad esso una determinata azione (una riproduzione vocale).

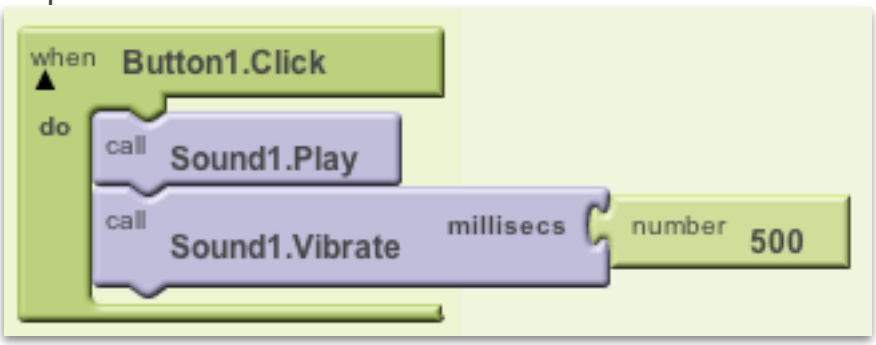
Quello che abbiamo visto, in pratica, è un esempio di **Event Handlers**. La sintassi di questo tipo di blocco si esaurisce nella parole **when ... do** che, tradotte in italiano, significano: "quando succede X fai Y".
E' bene precisare che ogni componente di MIT App Inventor ha una propria serie di eventi: questi ultimi, pertanto, cambieranno a seconda che il componente coinvolto sia, ad esempio, un Bottone o una TextBox.

Metodi e Proprietà

Abbiamo detto che ogni **componente** ha una propria serie di eventi associati, allo stesso modo possiamo dire di **metodi** e **proprietà**. Senza voler entrare nel dettaglio della programmazione ad oggetti, in questa sede basti sapere che col termine "metodo" s'intende solitamente un'azione che può essere compiuta da un dato componente, mentre col termine "proprietà" s'intende una sua caratteristica che può essere manipolata.

Chiamare un metodo

Per meglio comprendere il funzionamento dei metodi vediamo un altro esempio:



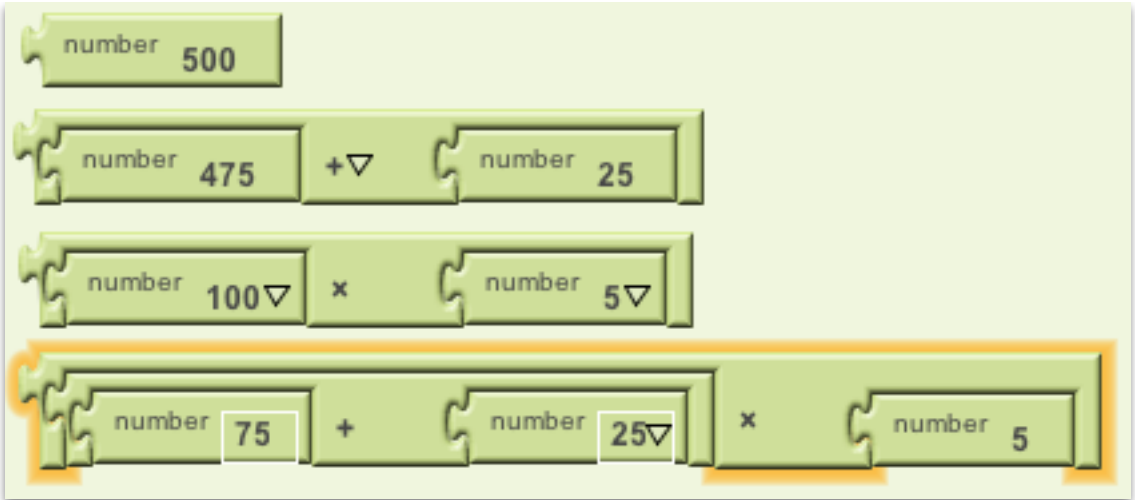
Nell'esempio qui sopra quando viene premuto un dato bottone (Button1) l'app riproduce un dato suono (Sound1) ed attiva la vibrazione per mezzo secondo.
Possiamo notare dal nostro esempio che ogni azione da eseguire è attivata dal blocco **call** e che di questi blocchi è possibile inserirne più di uno. Volendo usare un linguaggio maggiormente tecnico (e sicuramente più appropriato) possiamo dire di aver usato call per chiamare due **metodi** del componente Sound (rispettivamente Play e Vibrate).

Osservando il nostro esempio, ancora, possiamo notare che alcuni metodi, per essere eseguiti, necessitano di un **parametro** (è il caso della vibrazione che necessita dell'inidicazione esplicita della durata) mentre altri (come Play) no.
Nota: quando all'interno di un medesimo evento vengono racchiuse più azioni queste vengono eseguite in ordine dall'alto verso il basso. Nel nostro esempio, quindi, prima il telefono suonerà e poi vibrerà. Ovviamente potete cambiare l'ordine a vostro piacimento attraverso una semplice operazione di drag & drop.

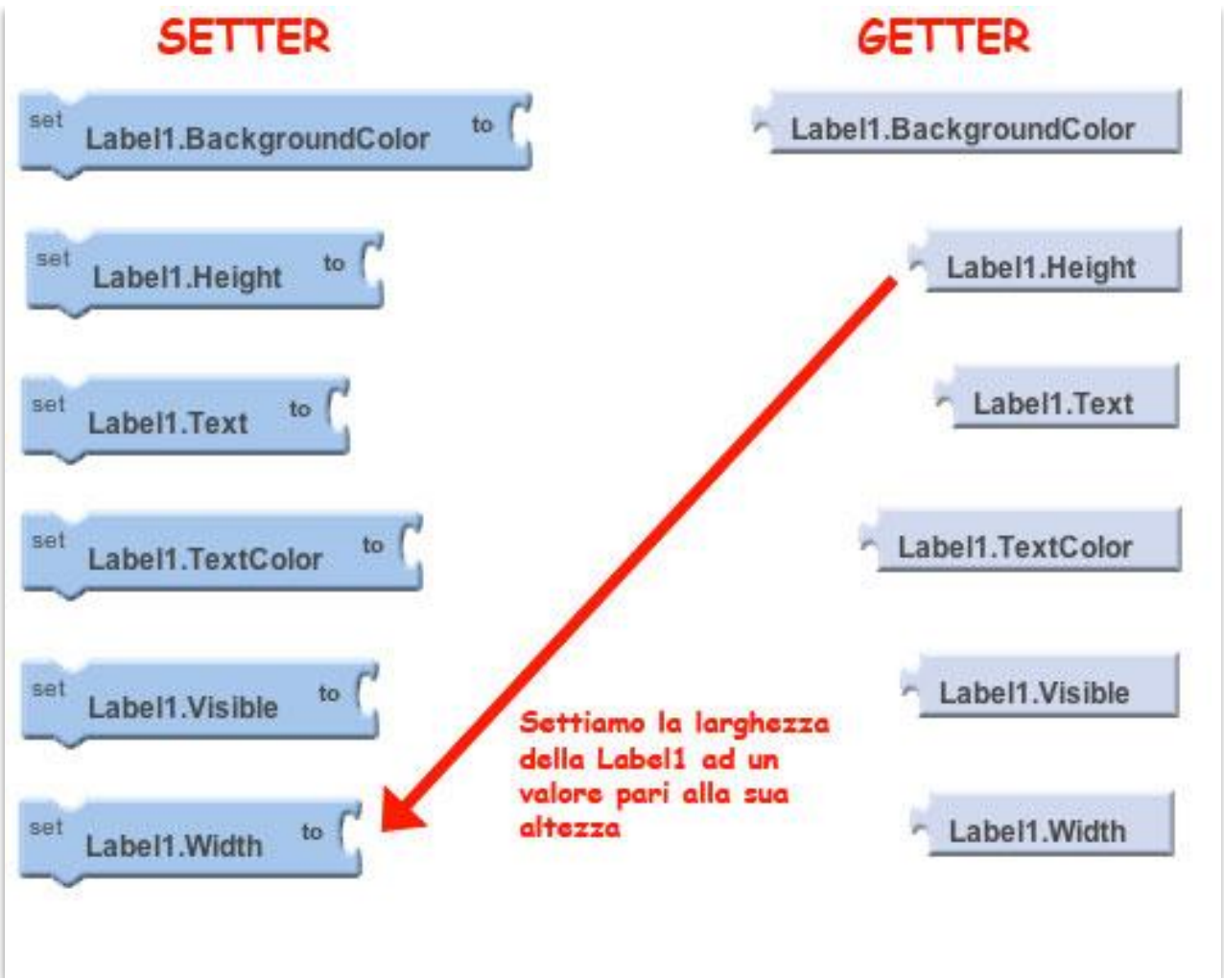
Lavorare con le proprietà: i blocchi setter e getter

Oltre a call, lo spazio do del blocco **when ... do**, può contenere anche un blocco di tipo **set** (cosidetti blocchi **setter**) attraverso il quale si assegna un valore ad una proprietà di un dato componente. In questo caso l'utilizzo di parametri è obbligatorio.
Come detto le proprietà corrispondono a delle caratteristiche dell'elemento, vediamo quindi, per fare un esempio pratico, come utilizzare set per effettuare una modifica sulla proprietà BackgroundColor di un componente Label:

Da notare che l'assegnazione di un valore, che nell'esempio qui sopra è avvenuta in modo diretto ed esplicito, può avvenire anche tramite un **espressione**. Ad esempio un'altezza o una larghezza possono essere calcolate con espressioni del tipo:



Oppure, ancora, è possibile assegnare un valore ricavato da un'altra proprietà dello stesso o di un altro componente mediante un **getter** (cioè un blocco col quale non si setta, ma si recupera il valore di una data proprietà). Di seguito alcuni esempi di setter e getter relativi ad una ipotetica Label:


SETTER
GETTER
set Label1.BackgroundColor to
Label1.BackgroundColor
set Label1.Height to
Label1.Height
set Label1.Text to
Label1.Text
set Label1.TextColor to
Label1.TextColor
set Label1.Visible to
Label1.Visible
set Label1.Width to
Label1.Width
Settiamo la larghezza della Label1 ad un valore pari alla sua altezza

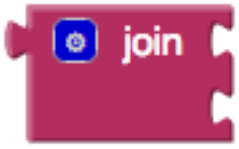
7. Gestire le stringhe (Texts Blocks)

In questa nuova lezione della nostra guida a **MIT App Inventor** vediamo come operare con le **stringhe di testo**. Anche in questo caso abbiamo a disposizione dei blocchi appositi che prendono il nome di **Text Blocks** o blocchi di testo.
Per prima cosa vediamo come definire una stringa:

Il blocco contrassegnato dai simboli " e " può contenere qualsiasi carattere: lettere, numeri, spazi, simboli speciali. Tutto questo blocco sarà considerato da MIT App Inventor una stringa.
Vediamo ora come compiere le principali operazioni sulle stringhe.

Unire più stringhe

Per unire più stringhe in un unica stringa si usa il blocco **join**; questo blocco appartiene alla famiglia dei mutators e quindi può essere "ampliato" per collegare più delle due stringhe previste di default.



Operazioni comuni sulle stringhe

Vediamo di seguito i blocchi grazie ai quali è possibile effettuare le operazioni più comuni sulle stringhe:

- **trim**
- elimina eventuali spazi vuoti presenti prima o dopo la stringa
- **upcase** e **downcase**
- trasforma una stringa, rispettivamene, in minusco e maiuscolo

Misurare la lunghezza di una stringa

Per misurare la lunghezza di una stringa (spazi compresi) si utilizza il blocco **length**. Il valore restituito sarà, ovviamente, un numero.
Un altro blocco molto utile è **is empty** il quale consente di verificare se una data stringa è vuota: se la stringa ha lunghezza 0 questo blocco restituisce true, in caso contrario false.

Confrontare due stringhe

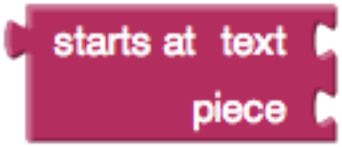
Per svolgere questa operazione, MIT App Inventor mette a disposizione il blocco **compare texts** il quale effettua la comparazione tra due stringhe (posizionate a sinistra e destra del blocco) attraverso gli operatori:

- =
- <
- >

Attraverso il simbolo dell'uguaglianza si verifica, in pratica, se le due stringhe sono identiche; con i simboli di maggiore e minore si verifica, da un punto di vista logico-alfabetico, qual'è la stringa maggiore e quella minore.

Inizia o contiene

Attraverso due appositi blocchi è anche possibile verificare se una data stringa inizia o contiene una data sotto-stringa. I blocchi in questione sono **starts at** e **contains**. A questi due bocchi vengono passati due argomenti: la stringa sulla quale effettuare la verifica e la sotto-stringa della quale si desidera verificare l'eventuale presenza.



Il blocco starts at restituirà 0 se la sotto-stringa non è presente, viceversa restituira la posizione del primo carattere della sottostinga all'interno della stringa. Ad esempio: "web" in "mrwebmaster" restituirà 3 perchè questo numero corrisponde alla posizione della lettera "w" all'interno della nostra stringa di riferimento.

Il blocco contains, invece, restituirà semplicemente true o false a seconda che sia stato possibile o meno ritrovare la sotto-stringa all'interno della stringa originale.

Dividere una stringa

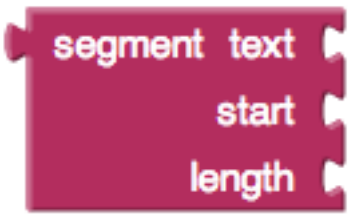
Se desideriamo dividere una stringa abbiamo a disposizione diversi blocchi, i pù utilizzati sono:

- **split**
- divide una stringa sulla base di uno specifico separatore (at); restituisce una lista di elementi;
- **split at first**
- divide una stringa sulla base di uno specifico separatore (at) ma questo viene considerato solo una volta, cioè alla prima occorrenza di sinistra verso destra; restituisce una lista di due elementi;

Oltre a questi deve essere menzionato **split at spaces** che, come è facile intendere, effettua la divisione della stringa sulla base degli spazi restituendo una lista di elementi.

Tagliare una stringa

Se desideriamo tagliare una stringa possiamo ricorrere al blocco **segment**.



Questo blocco richiede tre argomenti: la stringa che si desidera tagliare, il carattere di partenza (valore numerico corrispondente alla posizione) e la lunghezza attesa per la sotto-stringa che dovrà essere prodotta. Ad esempio:

- text: "mrwebmaster"
- start: 3

- lehgth: 3

produrrà: "web"

Effettuare sostituzioni

E' possibile effettuare sostituzioni all'interno di una stringa utilizzando il blocco **replace all**.

Grazie a questo blocco saranno sostituite all'interno della stringa specificata in text tutte le occorrenze di segment con replacement. Ad esempio:

- text: "mrwebmaster"
- segment: "mr"
- replacement: "mister"

produrrà: "misterwebmaster"

8. Numeri, operazioni e funzioni matematiche (Math Blocks)

Altri blocchi fondamentali nella programmazione con **MIT App Inventor** sono i **Math Blocks**, cioè i blocchi matematici. Attraverso questi blocchi possiamo compitere operazioni matematiche di vario tipo, dalle più semplici (come sottrazioni o addizioni) alle più complesse (logaritmi e radici quadrate).
In questa lezione vedremo i Math Blocks più importanti partendo, ovviamente, dai blocchi numerici (cd. **basic number block**).

Questo blocco può essere utilizzato per rappresentare qualsiasi numero, intero o decimale, positivo o negativo. Per cambiare il valore numerico è necessario fare click sullo 0 di default.

Semplici operazioni matematiche

I blocchi utilizzati più comunemente per effettuare semplici calcoli matematici sono:

- + (addizione)
- - (sottrazione)
- * (moltiplicazione)
- / (divisione)
- ^ (potenza)

Tra questi blocchi, due meritano particolare attenzione: si tratta dei blocchi + e * i quali appartengono alla categoria dei **mutators**. In pratica, attraverso questa funzionalità, è possibile "espandere" gi spazi previsti di default (2 numeri, a sinistra ed a destra dell'operatore matematico) e creare, rispettivamente, somme e moltiplicazioni di una pluralità di numeri.
Per attivare questa funzionalità bisogna cliccare sul simbolo blu in alto a sinistra:

Calcolo di min e max

MIT App Inventor ci mette a disposizione dei blocchi appositi per il calcolo del valore minimo e massimo all'interno di una pluralità di valori numerici. I due blocchi in questione sono, appunto, **min** e **max**. Come visto poco sopra relativamente agli operatori + e *, anche min e max appartengono alla famiglia dei mutators e possono essere ampliati per raggruppare una molteplicità di valori numerici.

Arrotondamenti

I blocchi destinati alla gestione degli arrotondamenti numerici sono tre:

- **round**
- Arrotonda un numero decimale all'intero più prossimo (es. 4.3 sarà arrotondato a 4, 4.7 sarà arrotondato a 5)
- **ceiling**
- Arrotonda un numero decimale all'intero inferiore (es. 4.7 sarà arrotondato a 4)
- **floor**
- Arrotonda un numero decimale all'intero maggiore (es. 4.3 sarà arrotondato a 5)

Comparazione

MIT App INventor 2 offre diversi blocchi per il confronto di due valori numerici, questi sono:

- = (uguale)
- ≠ (diverso)
- > (maggiore)
- ≥ (maggiore o uguale)
- < (minore)
- ≤ (minore o uguale)

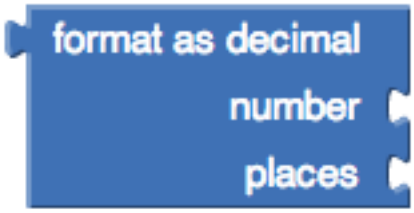
Numeri casuali

Attraverso il blocco **random integer** è possibile creare un numero intero casuale compreso tra un valore numerico A ed uno B.

Attraverso **random fraction**, invece, è possibile creare un valore casuale compreso tra 0 e 1.

Formattare un numero decimale

Attraverso il blocco **format as decimal** è possibile formattare un numero decimale specificando il livello di precisione (numero di decimali); il blocco provvederà automaticamente ad arrotondare il numero secondo il livello di precisione specificato.



Verificare se sia un numero

Tra i tanti Math Blocks messi a disposizione da MIT App Inventor c'è anche **is a number?**, si tratta di un blocco particolare che consente di verificare se un dato valore è numerico oppure no. Restituisce true o false a seconda che la verifica abbia avuto esito positivo o negativo.

9. Gestire le variabili (Variables Blocks)

Un elemento molto importante nella programmazione, a prescindere dal linguaggio o dall'ambiente utilizzato, sono le **variabili**. Queste non sono altre che delle porzioni di memoria all'interno delle quali vengono immagazzinate delle informazioni affinchè possano essere utilizzate nella fase di elaborazione del programma. Una caratteristica tipica delle variabili è, appunto, quello di essere "variabili"... cioè di poter cambiare valore nel corso del flusso del programma.
Una distinzione molto importante, quando si parla di variabili, è tra **variabili globali** e **variabili locali**. Le prime possono essere utilizzate all'interno di ogni punto della nostra applicazionea, mentre l'ambito di validità delle seconde è circoscritto a specifiche procedure. Anche in MIT App Inventor questa distinzione è di fondamentale rilevanza.

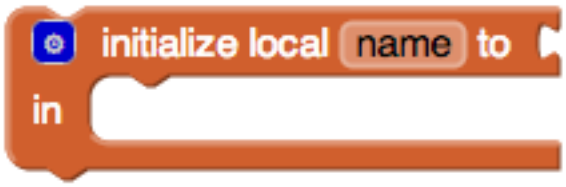
Definire una variabile globale

Grazie ai blocchi di MIT App Inventor, anche lavorare con le variabili diventa molto semplice. Per prima cosa vediamo come definire una variabile globale, cioè, come abbiamo già detto, un variabile che potrà essere utilizzata ovunque all'interno della nostra app.
Per definire una variabile globale si ricorre al blocco **initialize global**.

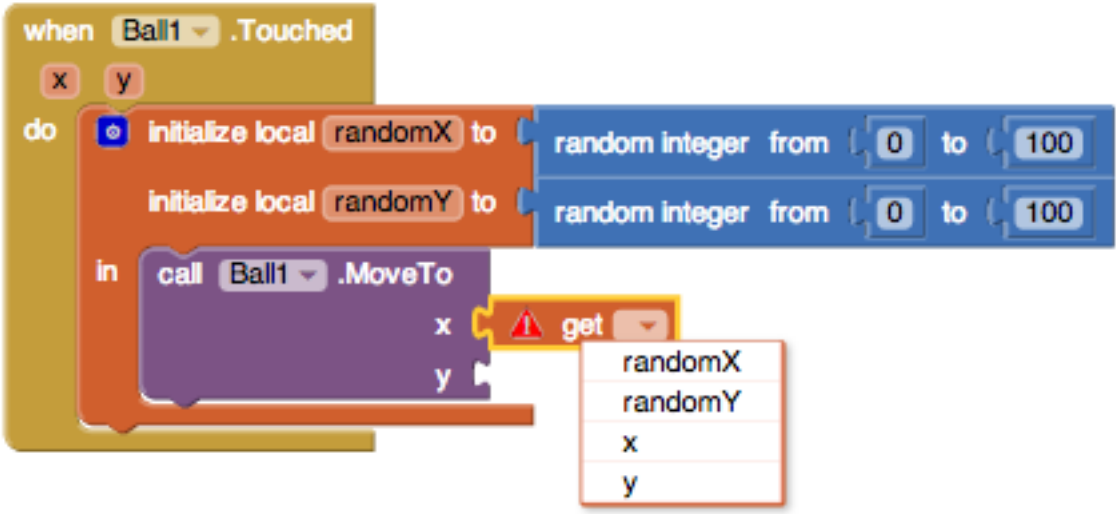
Al posto di name deve essere specificato il nome della variabile che si desidera creare, mentre to corrisponde al valore che le si vuole assegnare.

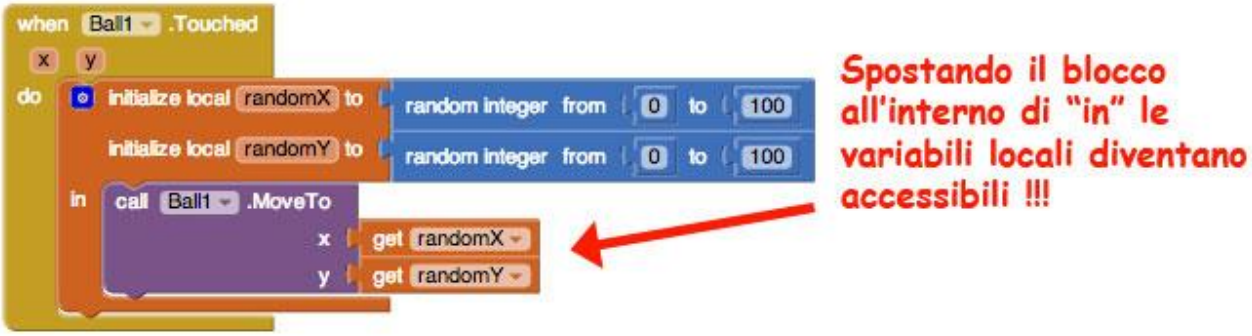
Definire variabili locali

Per definire una variabile locale si ricorre al blocco **initialize local** il quale può essere utilizzato al'interno di **do** o **return**.



Al posto di name deve essere specificato il nome della variabile che si desidera creare, mentre to corrisponde al valore che le si vuole assegnare e in corrisponde all'ambito di validità (scope) della variabile. Vediamo un esempio:





Assegnare un valore ad una variabile

Essendo variabile, come detto, è possibile modificarne il valore nel corso del programma. Per farlo utilizzeremo il blocco **set ... to**.

Dal menu dropdown si seleziona la variabile cui si desidera assegnare un nuovo valore che viene specificato in to. Si noti che all'interno del menu dropdown saranno presenti le sole variabili disponibili nello scope.

Recuperare il valore di una variabile

Per recuperare il valore di una variabile si ricorre al blocco **get**.

Come visto per set, anche in get è necessario selezionare dalla tendina il nome della variabile tra quelle disponibili nello scope

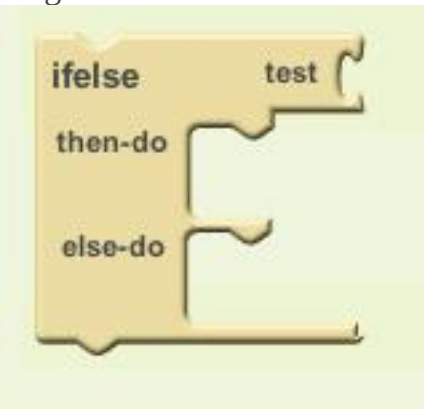
10. Strutture di controllo (Control Blocks)

Nel corso di questa lezione prenderemo confidenza con alcuni elementi di grande utilità per la progettazione delle nostre app: in particolare, approfondiremo la nostra conoscenza della categoria **Control** del pannello **Blocks** (accessibile dalla scheda **Blocks** della schermata iniziale). Ci concentreremo cioè sui principali **Control blocks**.
Per chiarezza espositiva andremo a distinguere i **control block** che ci accingiamo a esaminare in tre gruppi: **controllo di flusso**, **cicli**, **controllo dell'applicazione**.

Controllo di flusso

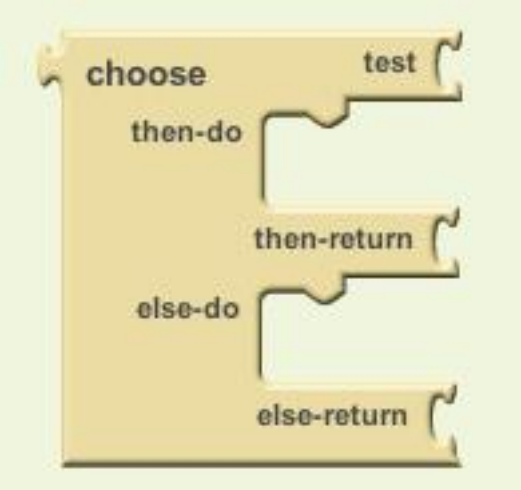
I **control block** che classifichiamo in questa categoria consentono fondamentalmente di operare una scelta circa il comportamento da tenere, in funzione di particolari condizioni. In sostanza, ci permettono, ad esempio di definire come l'app deve comportarsi in una forma del tipo: "se la condizione X è verificata esegui il blocco Y di istruzioni". Questo è esattamente il significato del blocco **if**, riportato nella figura sottostante.

Il blocco **if-else** riportato nella figura in basso è leggermente più elaborato e potrebbe suonare come: "se la condizione X è verificata esegui il blocco Y di istruzioni, altrimenti esegui il blocco Z".



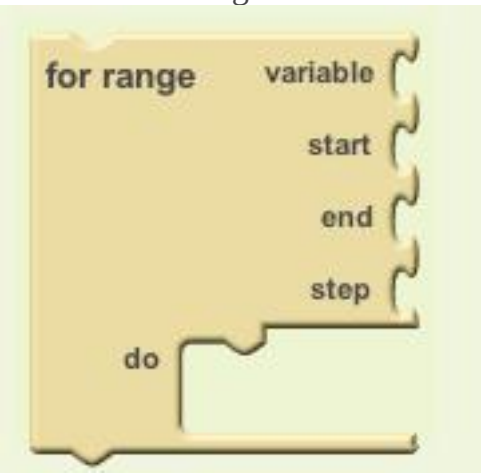
Il blocco più elaborato per quanto riguarda il controllo di flusso, è il blocco **choose** (v. figura in basso). La differenza rispetto al blocco if-else consiste ne fatto che, oltre a poter scegliere quale tra i due blocchi di istruzioni eseguire (a seconda del valore della condizione, vera o falsa), il blocco restituisce un valore (ad esempio un numero intero). Il valore restituito è diverso a seconda che sia stato eseguito il blocco di istruzioni 1 o il blocco di istruzioni 2,

identificando quindi qual è stato il percorso del programma fino a quel momento. Tale valore può essere usato a sua volta, ad esempio, come input per altri **control blocks**.

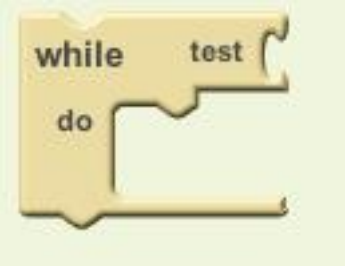


Cicli

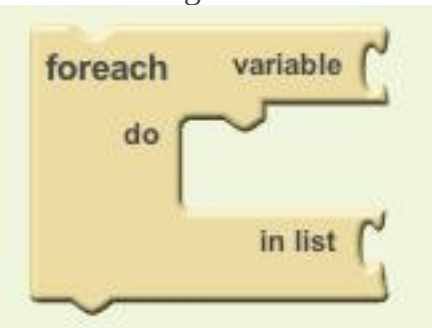
I **control block** che raggruppiamo nella categoria **cicli** sono utili per rappresentare in modo opportuno e conveniente l'esecuzione di operazioni ripetitive. Conviene infatti in tal caso raggruppare l'insieme delle operazioni da ripetere proprio all'interno di un **ciclo**. Se il numero delle ripetizioni da eseguire è noto, ci serviremo del blocco **for range** (v. figura in basso). In questo caso definiremo una variabile (**variable**) che svolge il ruolo di contatore: il conteggio parte da un valore iniziale (**start**) e viene incrementata a ogni esecuzione del ciclo (detta anche **iterazione**) di un valore di incremento fissato (**step**). Il ciclo continua a essere eseguito fino a che il valore della variabile contatore non raggiunge o supera una soglia (**end**). Specificando, ad esempio **1** come valore di **start** e di **step** e 10 come valore per **end**, il ciclo viene eseguito **10** volte.



Il blocco **while**, che possiamo osservare nella figura sottostante, ci sarà utile invece nei casi in cui il numero delle ripetizioni del ciclo non sia prefissato. Il ciclo verrà ripetuto fino a che una determinata condizione (**test**) è verificata.

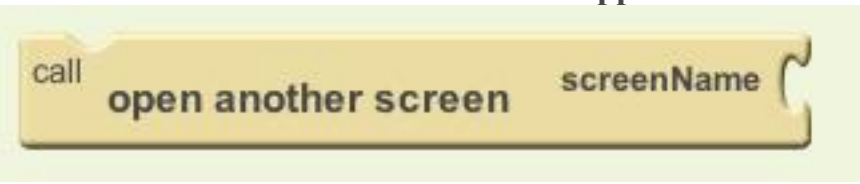


Il ciclo di tipo **foreach** ci tornerà utile qualora avessimo bisogno, per la nostra **app** di costruire **strutture dati** per ospitare informazioni di interesse. In particolare, il ciclo **foreach**, è adatto a operare su **liste** di dati, consentendo di eseguire le operazioni indicate nel ciclo su ogni elemento che fa parte della **lista**.



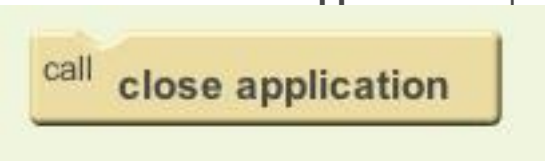
Controllo dell'applicazione

Ci serviremo di questi **control blocks** per regolare il comportamento dell'**app**. In particolare, il blocco **open another screen**, mostrato nella figura sottostante, ci consentirà di aprire un'altra schermata da dedicare alla nostra **app**.



Per chiudere la schermata corrente, potremo servirci del blocco **close screen**, che vediamo nella figura sottostante.

Infine il blocco **close application** ci permette di terminare l'esecuzione della **app**.



11. Operatori logici, di confronto e di aggregazione (Logic Blocks)

Nella lezione precedente abbiamo visto come gestire il flusso di un'app Android attraverso i **control block**. Abbiamo visto, più precisamente, come creare dei semplici costrutti condizionali e iterattivi sfruttando la semplicità dell'interfaccia grafica di **MIT App Inventor**.
In questa lezione vediamo il funzionamento dei **logic blocks**, o blocchi logici, ovvero:

- true
- false
- not
- =
- ≠
- and
- or

Come è facile intuire, per chi abbia già delle basi di programmazione, si tratta di blocchi che ci torneranno utili in fase di definizione di strutture condizionali. Vediamoli nel dettaglio.

Valori booleani

I valori boolenai **true** e **false** possono essere utilizzati in fase di assegnazione di valore ad un componente o una variabile.

Negazione

Il blocco **not** funge da negazione ed inverte il risultato di una condizione. Server, ad esempio, per verificare che un dato valore non sia uguale a qualcosa, non sia vero, ecc.

Uguale e diverso

Come è facile intuire, i blocchi = e ≠ sono utilizzati, rispettivamente, per verificare che gli argomenti collegati (a destra ed a sinistra) siano uguali oppure diversi.

Due argomenti s'intendono uguali:

- quando si tratta di due numeri aventi lo stesso valore numerico. Ad esempio: 1 = 1.0

- quando si tratta di due stringhe dallo stesso contenuto; si faccia attenzione perchè questa operazione è case-sensitive e quindi "bottone" è diverso da "Bottone"
- quando si tratta di due liste identiche, col medesimo numero di elementi e questi ultimi hanno lo stesso contenuto

And e Or

Si tratta di due operatori utilizzati per creare condizioni complesse. Perchè una condizione sia true è necessario che tutte le sotto-condizioni collegate con **and** siano vere; se l'operatore utilizzato è **or** è sufficiente che sia vera una sola delle sotto-condizioni.

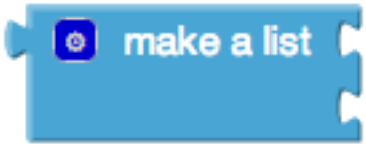
12. Gestire le liste (Lists Blocks)

Grazie ai blocchi di **MIT App Inventor** è possibile gestire delle **liste** di elementi, si tratta, per chi sa di programazione, di uno strumento analogo alle array.
In parole semplici, una lista è un insieme di elementi contraddistinti da un indice numerico che li identifica all'interno della lista stessa. Il primo elemento ha indice 1 (diversamente da molti linguaggi di programmazione dove il conteggio parte da 0). Si noti che una lista può contenere, al suo interno anche altre liste.
I blocchi resi disponibili da MIT App Inventor per la gestione delle liste sono paecchi, vediamo i più importanti.

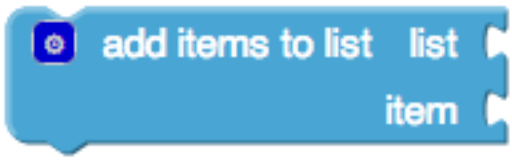
Creare una lista, aggiungere e rimuovere elementi

Per creare una lista si utilizza il blocco **create empty list** il quale, come lascia intendere il nome, crea una lista vuota (cioè priva di elementi).

Se si desidera creare una lista e specificare subito degli elementi si utilizzerà il blocco **make a list**. Cliccando sul pulsante bluè possibile espandere il blocco e prevedere spazi per maggiori elementi.

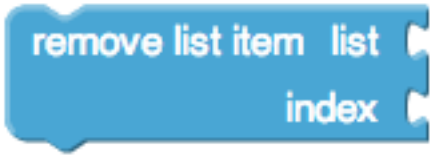


Una volta creata la lista sarà possibile aggiungere nuovi elementi attraverso il blocco **add items to list**. A questo blocco deve essere associata una lista e poi devono essere specificati il o gli elemento che si desidera aggiungere. Anche questo blocco, come il precedente. è un mutator e quindi può essere espanso.

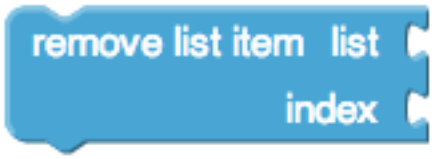


Con questo blocco i nuovi elementi vengono aggiunti in fondo alla lista.
Se si desidera inserire un nuovo elemento in una posizione specifica della lista si userà, invece, **insert list item** il quale consente di specificare la posizione (indice) al quale inserire l'elemento.

Se, invece, si vuole eliminare un elemento si farà ricorso al blocco **remove list item**: sarà sufficiente specificare l'indice dell'elemento che si desidera rimuovere dalla lista per ottenere quanto voluto.

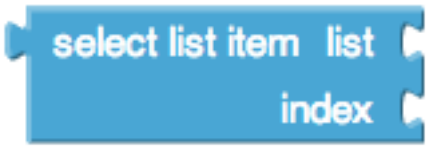


Ma se invece di rimuovere l'elemento desiderassimo, semplicemente, sostituirlo? A tal fine ci soccorre il blocco **replace list item**. Per effettuare la sostituzione non dovremo fare altro che indicare l'indice dell'elemento da sostituire ed il nuovo valore da inserire quale elemento con l'indice specificato.



Selezionare elementi della lista

Vediamo ora come selezionare un elemento della lista. Per farlo si utilizza il blocco **select list item** associando a questo, oltre alla lista, l'indice dell'elemento che si desidera selezionare.



E' anche possibile selezionare un elemento casuale all'interno della lista utilizzando **pick a random item**.

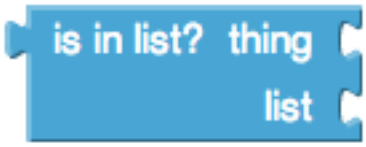
Verificare se sia una lista, se sia vuota e quanti elementi contenga

Tre blocchi molto utilizzati sono:

- **is a list?** - verifica se un dato elemento è o non è una lista;
- **is list empty?** - verifica se una data lista è o non è vuota;
- **length of list** - conteggia il numero di elementi presenti nella lista (restituisce 0 se la lista è vuota);

Verificare se una lista contenga un dato elemento

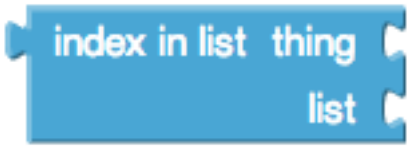
Può capitare spesso, in sede di sviluppo di una qualsivoglia applicazione, di avere la necessità di verificare se una data lista contiene o meno un dato elemento. Per farlo si ricorre al blocco **is in list?**.



A questo blocco deve essere passato come thing l'elemento di cui si desidera verificare la presenza nella lista specificata in list.

Conoscere l'indice di un elemento nella lista

Quando si gestisce dinamicamente una lista non è sempre possibile conoscere a priori l'indice di un dato elemento al suo interno. Per risalire a questo valore numerico è possibile ricorrere al blocco **index in list**.



A questo blocco deve essere passato come thing l'elemento di cui si desidera conoscere l'indice all'interno della lista specificata in list. Se le'emento non viene trovato restituisce 0.

13. Gestire i colori (Colors Blocks)

Gestire il colore di un elemento con MIT App Inventor è davvero molto semplice. Per farlo, ancora una volta, non è necessario scrivere codici o altro ma è sufficiente fare ricorso ai cosidetti **color blocks**. I blocchi di questa famiglia sono soltanto tre. Vediamoli uno ad uno.

Basic color block

Si tratta del blocco più semplice; è rappresentato da un quadrattino colorato che rappresenta il colore in uso il determinato elemento cui viene associato.

Per cambiare colore è sufficiente cliccare col mouse all'interno del quadrattino: così facendo comparirà una tavolozza di 70 diversi colori tra i quali è possibile scegliere. Cliccando su un colore all'interno della tavolozza questa si chiuderà ed il quadrattino cambierà colore in corrispondenza a quello che abbiamo scelto.

Creare un colore personalizzato

A chi ha maggiori esigenze grafiche, MIT App Inventor mette a disposizione il blocco **make color** grazie al quale è possibile creare colori personalizzati (quindi superare il limite dei 70 standard colori presenti nella tavolozza). Per creare un colore personalizzato si dovrà adottare la notazione RGB e quindi specificare tre valori numerici da 0 a 255 corrispondenti alle quantità di rosso, verde e blue. E' anche possibile estendere questo blocco per aggiungere

un quarto valore corrispondente alla saturazione (da 0 a 100). Se non settato è impostato a 100 di default.

Dividere un colore

E' anche possibile svolgere l'operazione inversa rispetto a quella gestita col blocco make color, per farlo si ricorre al blocco **split color**: questo blocco, dato un colore, ne restituisce il valore RGB.

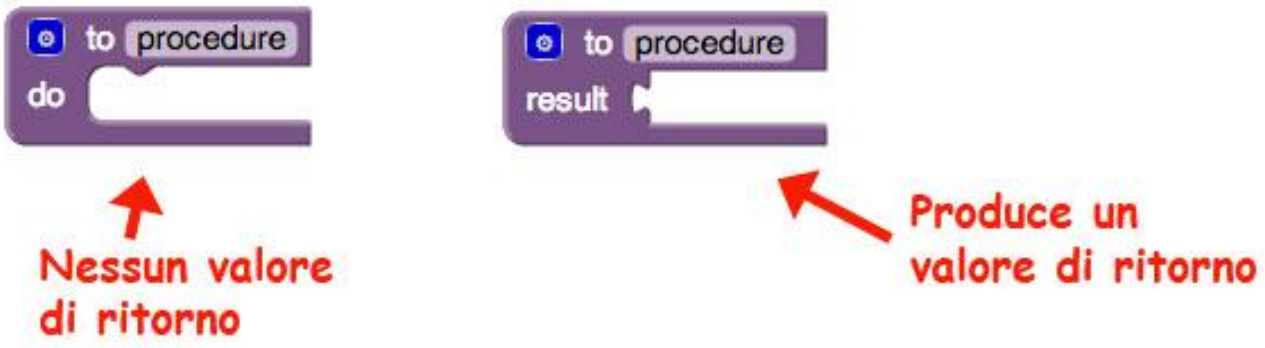
14. Le procedure

Una delle caratteristiche più interessanti di **MIT App Inventor** è la possibilità, offerta agli utilizzatori, di espandere il linguaggio mediante la definizione di nuovi blocchi (funzionalità) che prendono il nome di **procedure**.
Una procedura non è altro che un insieme di blocchi nativi che raggruppati godono di una certa autonomia ed assolvono ad una funzione specifica ed, in un certo senso, astraibile dal contesto generale. In pratica, attraverso la definizione di procedure, si assolve ad una buona prassi di programmazione attuando quel **riuso del codice** che è sintomo di ordine ed efficienza (viene eliminata la ridondanza nel codice) di ogni buona applicazione.
Una volta definita la procedura questa potrà essere richiamata all'interno dell'applicazione passando eventuali argomenti ed ottenendo in risposta l'elaborazione attesa.
Un esempio di procedura potrebbe essere data dal calcolo dell'IVA su un dato prezzo: si passando come argomenti un prezzo e l'aliquota da applicare e, mediante l'utilizzo di alcuni Math Blocks si calcola e poi restituisce il valore del prezzo ivato.
In MIT App Inventor si distinguono due tipi di procedure:

- procedura semplice
- procedura con valore di ritorno

Creare una procedura

La definizione di una nuova procedura avviene in modo simile a come viene definita una variabile utilizzando il blocco **procedure do** oppure **procedure result** a seconda che si desideri o meno un valore di ritorno.

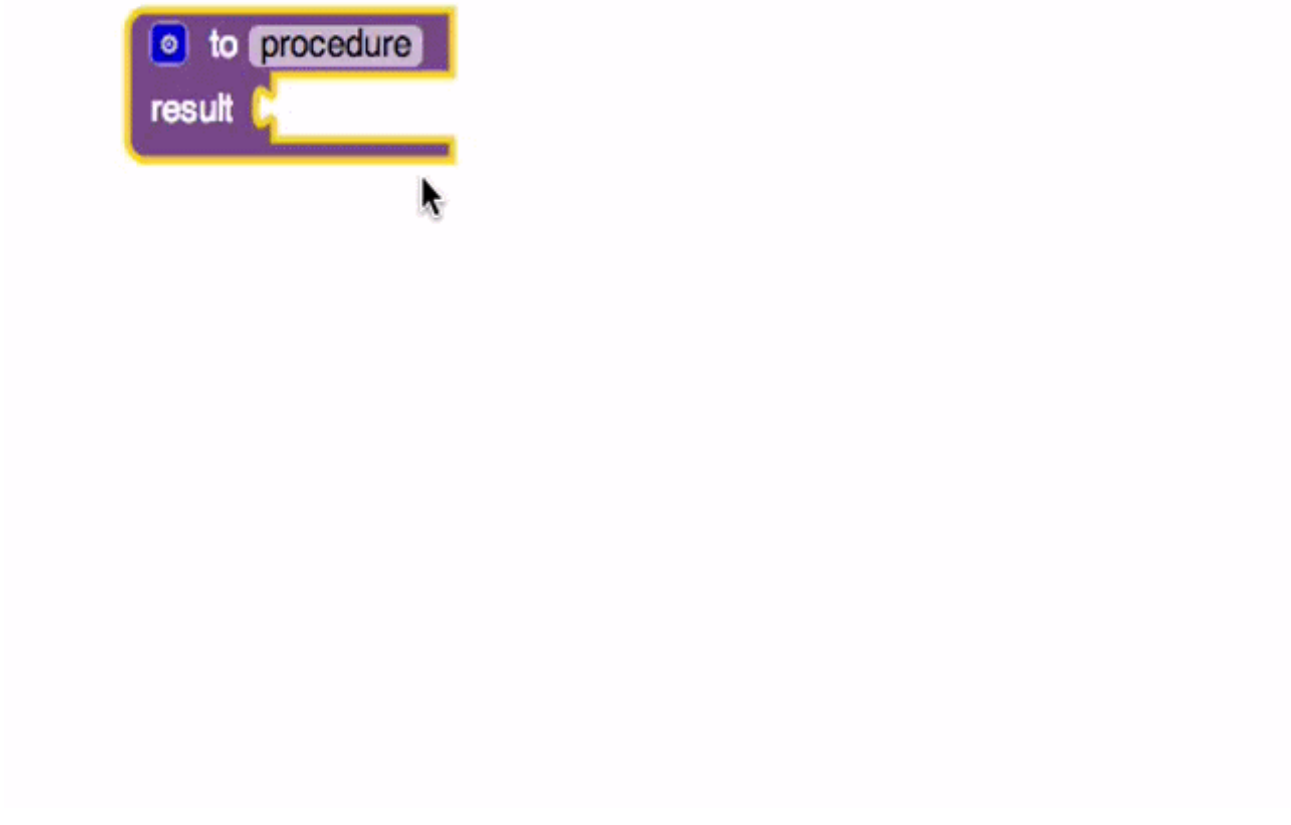


La fase di elaborazione è gestita, invece, all'interno di **do** dove, in pratica, andremo ad aggiungere tutti i blocchi necessari al compito che ci siamo prefissati. Se abbiamo scelto di utilizzare il blocco **procedure result** la fase di elaborazione sarà gestita all'interno di result.

Definire gli argomenti della procedura

Per definire eventuali argomenti per la nostra procedura dovremo utilizzare il pulsante blu mutator posizionato in alto a sinistra del nostro blocco.

Ad ogni argomento deve essere assegnato un nome univoco il quale sarà poi disponibile all'interno della procedura stessa.
Di seguito un immagine animata (tratta dalla documentazione ufficiale del MIT) mostra come aggiungere argomenti ad una procedura:



Per fare un esempio vediamo una semplice procedura, che abbiamo chiamato add, che esegue la somma dei due numeri passati in argomento:

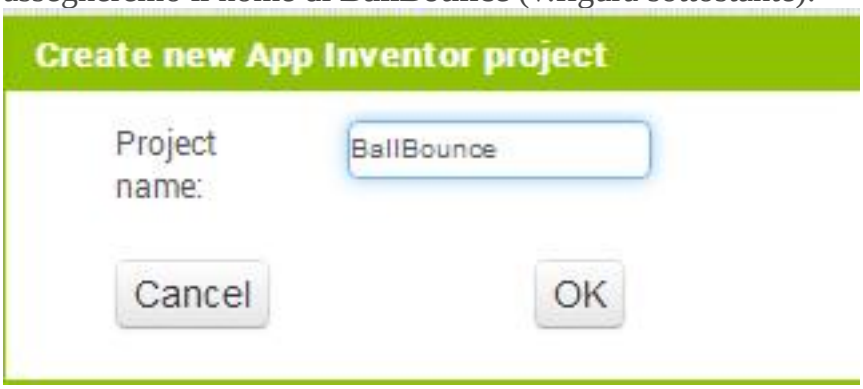
Richiamare una procedura

Una volta creata la procedura potrà essere richiamata in ogni punto della nostra app mediante **call** seguito dal nome della procedura e dall'indicazione degli eventuali argomenti richiesti.

Come potete notale il call di una procedura con valore di ritorno può essere "pluggato" ad altri blocchi cui verrà passato, appunto, tale valore.
Nota: è bene ricordare, infine, che ogni procedura deve avere un nome univoco. Una volta assegnato un nome alla procedura (cliccando sul nome di default) saraà possibile cambiarlo anche successivamente; in tal caso App Inventor si preoccuperà di aggiornare automaticamente tutti i richiami a quest'ultima aggiornandoli col nuovo nome ad essa assegnato.

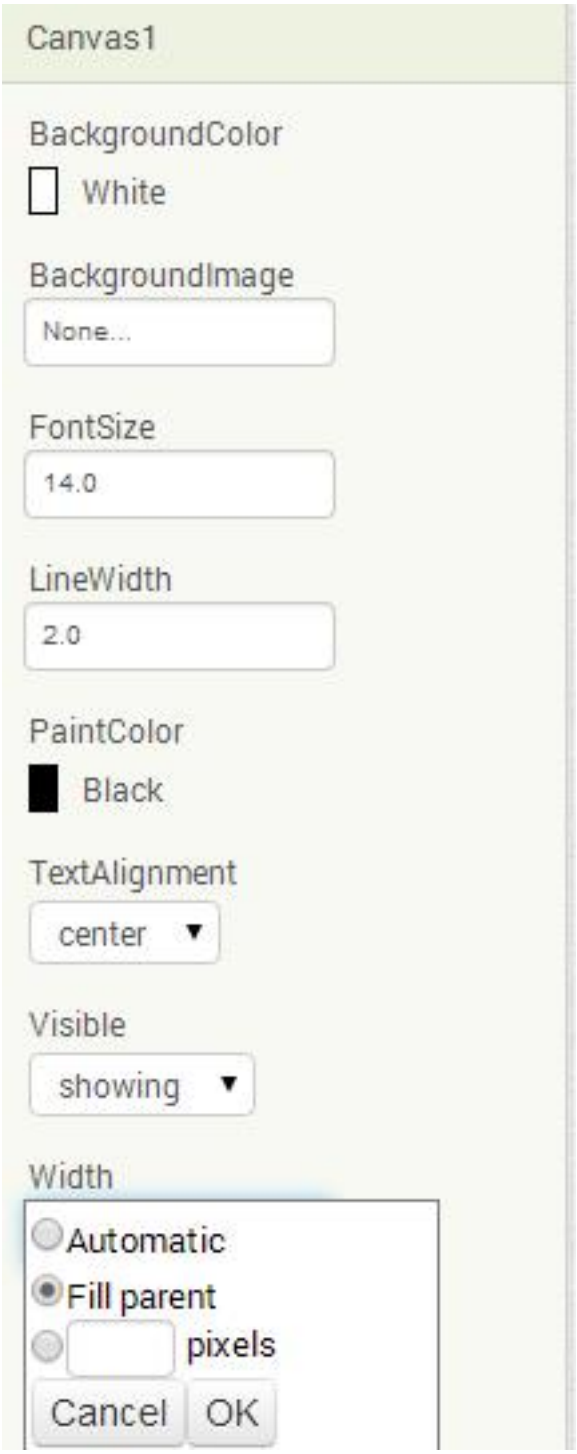
15. Creare una semplice animazione con Mit App Inventor

A partire da questa lezione cercheremo di offrire al lettore degli esempi pratici su specifiche tematiche. Lo scopo di questa lezione, nello specifico, è costruire una nuova app che faccia uso di **componenti** e **behavior** finalizzati alla realizzazione di una semplice **animazione**.
In particolare, la nostra app ci consentirà di muovere col dito una sorta di palla (rappresentata da un'immagine a forma di cerchio) che continuerà a correre sullo schermo nella direzione impressale. Faremo in modo inoltre, che la palla possa "rimbalzare" una volta raggiunto il limitare destro, sinistro, inferiore o superiore del nostro schermo.
La scelta dell'**app** è motivata dal fatto che gli stessi sviluppatori di **MIT App Inventor** indicano questo progetto come un modo semplice ed efficace per prendere confidenza con le **animazioni** e con i componenti della famiglia Drawing and Animation già accennati nella lezione dedicata ia componenti di MIT App Inventor.
Per prima cosa accediamo alla schermata iniziale e creiamo un nuovo progetto, a cui assegneremo il nome di **BallBounce** (v.figura sottostante).



Inseriamo i componenti

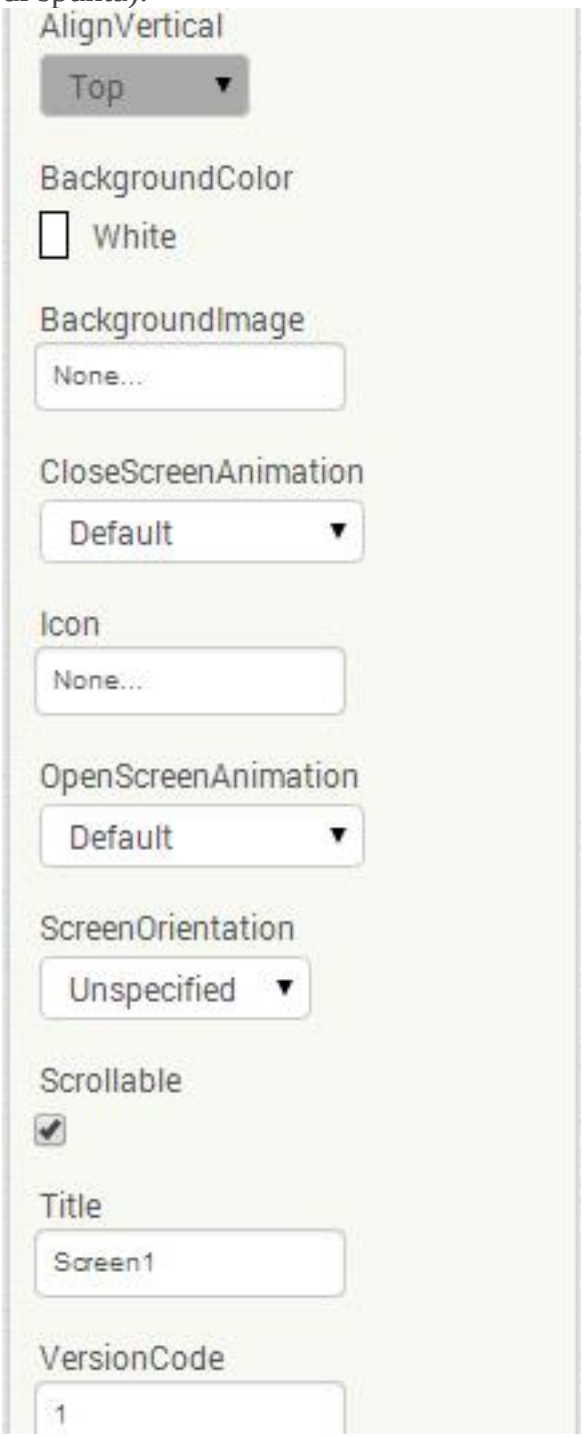
Poi, dalla scheda **Designer**, inseriamo le **componenti** di cui avremo bisogno.
Per poter assicurare il **rendering dinamico** dell'immagine, avremo bisogno di un elemento **Canvas**, primo elemento del menu **Drawing and Animation** del pannello **Palette**.
Facciamo poi in modo che l'elemento **Canvas** appena inserito si estenda per tutta la lunghezza e la larghezza dello schermo. Per ottenere ciò impostiamo, come possiamo vedere nella figura in basso, la proprietà **Width** (larghezza) della componente Canvas1 appena inserita, dal menu **Properties** selezionando il valore **Fill parent**. Grazie a questa operazione il nostro elemento "riempirà" tutta la larghezza a disposizione.



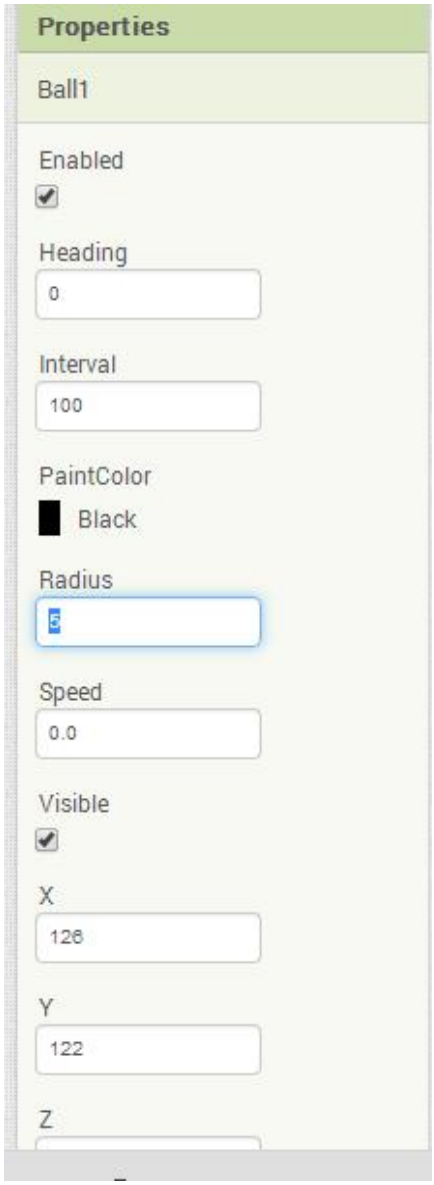
Analogamente, dovremo assegnare anche alla proprietà **Height** (altezza) il valore **Fill parent**.

Successivamente, delimitiamo l'area in cui la nostra pallina potrà muoversi, impedendo cioè allo schermo di "scorrere" (sarebbe altrimenti impossibile definire, come desideriamo, meccaniche di "rimbalzo"). Per ottenere ciò, selezioniamo la componente Screen1 e

priviamola della proprietà **Scrollable** (v.figura in basso: rimuoviamo, con un clic, il segno di spunta).



Aggiungiamo poi la componente Ball, secondo elemento del menu **Drawing and Animation** del pannello **Palette**. Per renderla maggiormente visibile, impostiamo, tramite il pannello **Properties** (sulla destra), un buon valore di **radius** (ovvero del raggio della nostra sfera), ad esempio **15**.



Gestiamo i behaviors

Non resta ora che definire i **behavior** propri della nostra **app**, dalla scheda **Blocks** della schermata principale. Tali behavior dovranno modellare il movimento della palla a seguito del trascinamento e rilascio (movimento che viene detto di **flung**) e il rimbalzo. Troveremo tutti i blocchi di nostro interesse tra quelli associati all'oggetto **Ball1**. Cominciamo dalla gestione del movimento di **flung**.

Trasciniamo a destra, nel pannello **Viewer**, il blocco **"When Ball1.Flung do"**. A questo punto colleghiamo a **"do"** sia **"set Ball1.speed to"** che **"setBall1.heading to"**, in modo da impostare, rispettivamente, **velocità** e **direzione** della palla.

Ma a quali valori vogliamo impostare **speed** e **heading**? Ovvero: quanto veloce e in che direzione la palla deve cominciare a muoversi sullo schermo? La velocità, così come la

direzione, sono determinate dalla natura del movimento di **flung** (che può essere verso l'alto, verso il basso, verso destra o verso sinistra, più o meno deciso). Desideriamo quindi leggere i valori di **speed** e **heading** associati al movimento di **flung** e trasferirli al movimento della palla, ovvero a **setBall1.speed** e a **setBall1.heading**.
MIT App Inventor, fortunatamente per noi sviluppatori, rende tale operazione molto semplice: posizioniamo il mouse, all'interno di **"When Ball1.Flung do"**, su **"speed"**. Appariranno due nuovi blocchi: **"get speed"** e **"set speed to"**. Trasciniamo il blocco **"get speed"** e rilasciamolo in modo che segua**set Ball1.speed to**, in modo da collegare i due blocchi. Svolgiamo un'operazione analoga anche per quanto riguarda la direzione, ovvero l'**heading**. A seguito di tali operazioni, il blocco risultante dovrà essere uguale a quello riportato nella figura sottostante.

Seguiremo una procedura simile anche per modellare il behavior associato al rimbalzo della palla, assemblando però in questo caso i blocchi: **"When Ball1.EdgeReached"**, **call Ball1.bounce** e **"get edge"** (il blocco **"get edge"** apparirà, in analogia al caso precedentemente trattato, stazionando col mouse su "edge", in alto nel blocco **"When Ball1.EdgeReached"**). Il risultato finale è mostrato nella figura in basso.

Una volta terminato, possiamo verificare il corretto funzionamento dell'app tramite l'**emulatore**.

16. Interazione con database

Fino a questo momento, abbiamo considerato app i cui dati non sono **persistenti**: ovvero rimangono in memoria soltanto durante l'esecuzione, per poi essere dimenticati. Sulla base delle informazioni prese in esame fino ad ora non sarebbe possibile, ad esempio, tenere traccia degli high scrores di un videogame oppure tenere in memoria le preferenze dell'utente circa determinati servizi.
In questa lezione vedremo, appunto, come soddisfare queste esigenze tramite una componente di **MIT App Inventor** che rappresenta un **database**.
Sono in realtà disponibili due opzioni per l'inclusione di un database tra le componenti della nostra app, queste sono:

- TinyDB
- TinyWebDB

Con il componente **TinyDB** i dati saranno salvati sulla memoria interna del dispositivo su cui la app stessa viene eseguita (ad esempio, uno smartphone), mentre nel caso di **TinyWebDB** i dati sono archiviati sul web e, pertanto, sono condivisibili da più dispositivi.
In questa lezione sceglieremo, per semplicità, la prima opzione.

TinyDB

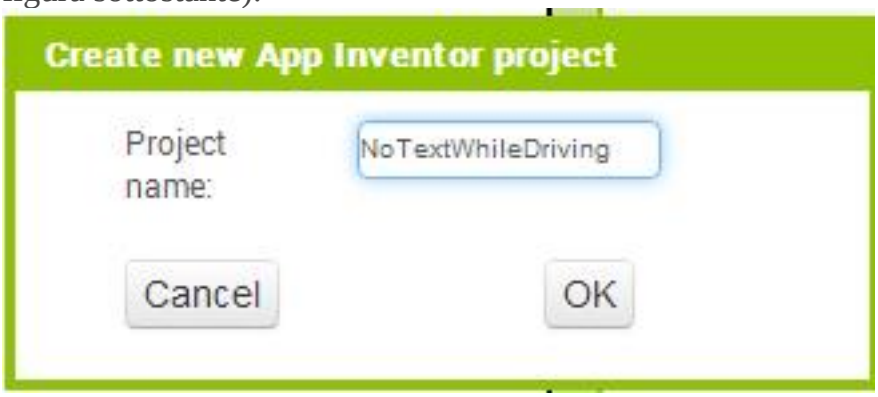
Grazie al componente "invisibile" TinyDB è possibile savare dati stringa mediante l'associazione a dei tags: in pratica quando si salvano dei deti gli si assegna un tag che a sua voltà verrà utilizzato per richiamare tali dati una volta salvati.
TinyDB dispone unicamente di cinque metodi, vediamoli:

- ClearAll() - cancella tutti i dati dell'app;
- ClearTag(tag) - cancella i dati associati allo specifico tag;
- GetTags() - restituisce un elenco di tutti i tag presenti nel data store;
- GetValue(tag, valueIfTagNotThere) - restituisce il valore corrispondente ad uno specifico tag o, qualora questo sia vuoto, il valore di default specificato in valueIfTagNotThere;
- StoreValue(tag, valueToStore) - salva un dato valore (valueToStore) associandolo ad un dato tag.

Un esempio pratico di app che utilizza TinyDB

L'app che intendiamo sviluppare ha una funzionalità molto semplice: mentre è in esecuzione, se si riceve un SMS viene generata una risposta automatica. Può essere utile, ad esempio, mentre si sta guidando, per segnalare a chi ci ha cercato che richiameremo appena possibile.
Il messaggio di risposta automatica viene memorizzato su un database e da lì prelevato all'avvio dell'applicazione. Tale messaggio può essere anche reimpostato, al clic dell'utente su un opportuno pulsante (**componente Button**).

Anche in questo caso, così come per alcune delle scorse lezioni, la scelta di questa app per impratichirsi con la componente TinyDB è suggerita dagli sviluppatori di **MIT App Inventor**. Creiamo per prima cosa un nuovo progetto, di nome **NoTextWhileDriving** (v. figura sottostante).

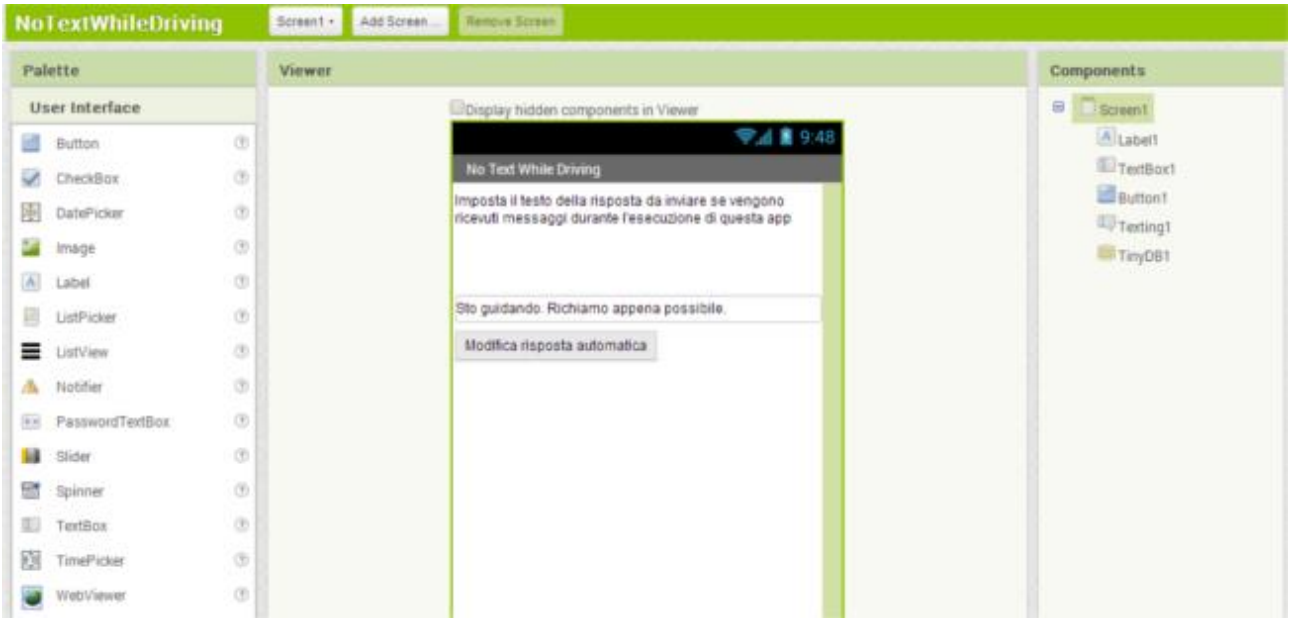


Inseriamo ora, dalla scheda **Designer**, le componenti, impostandone opportunamente le **proprietà**. Inseriamo tre **componenti visibili**: una **Label**, una **Textbox** e un **Button**.

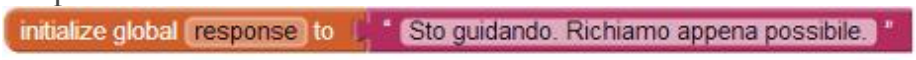
- Indichiamo all'interno della **Label** le istruzioni per l'utente, impostando la proprietà Text con un messaggio del genere: "Imposta il testo della risposta da inviare se vengono ricevuti messaggi durante l'esecuzione di questa app";
- Impostiamo poi la proprietà Text della componente **Button** a: "Modifica risposta automatica";
- Impostiamo la proprietà Text della componente **Textbox** a un messaggio di risposta di default, ad esempio: "Sto guidando. Richiamo appena possibile".

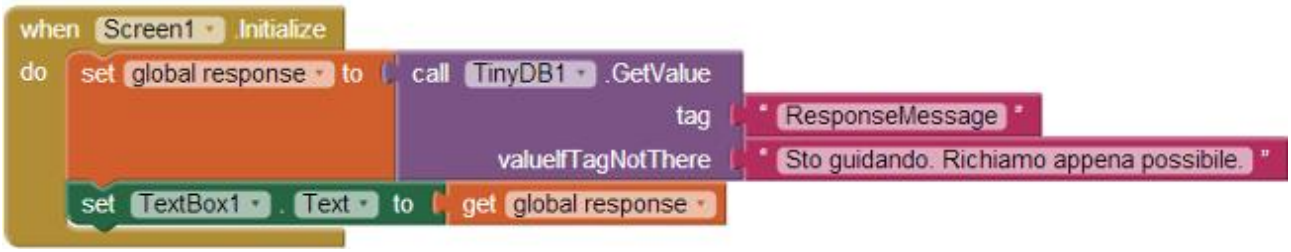
Impostiamo infine la proprietà **Title** della componente Screen1 (che rappresenta la schermata principale della nostra app ed è inserita nel progetto per default) a "NoTextWhileDriving".

Inseriamo inoltre due **componenti non visibili**: **Texting** (per la gestione degli SMS) e, ovviamente, TinyDB. Nella figura in basso, possiamo osservare la scheda **Designer** della schermata principale, come risulta dopo aver eseguito le operazioni indicate.



Passiamo ora alla scheda **Blocks** e ai **behavior** della nostra app, cominciando con lo specificare le operazioni da eseguire all'avvio.
Per prima cosa (tramite il blocco **initialize global**) definiamo una variabile a cui assegniamo il nome response ed un valore di default che tornerà utile qualora ci siano problemi con i passaggi successivi.
Questa variabile deve contenere il messaggio di risposta automatica, tale valore, pertanto, dovrà essere estratto dal **database**, tramite il metodo **TinyDB1.GetValue**, all'avvio della schermata principale (**"When Screen1.Initialize"**). Osserviamo, facendo riferimento alla figura in basso, che il messaggio viene cercato all'interno del database con lo specifico **tag** ResponseMessage.
Per garantire che in ogni caso tale variabile assuma un valore, dovremo definire un messaggio da assegnare in caso non sia possibile trovare il messaggio nel **database** (che potrebbe, ad esempio, essere ancora vuoto; il caso è gestito da **"valueIfTagNotThere"**).
Infine assegniamo, come valore predefinito della TextBox1, il valore della nostra variabile Response.

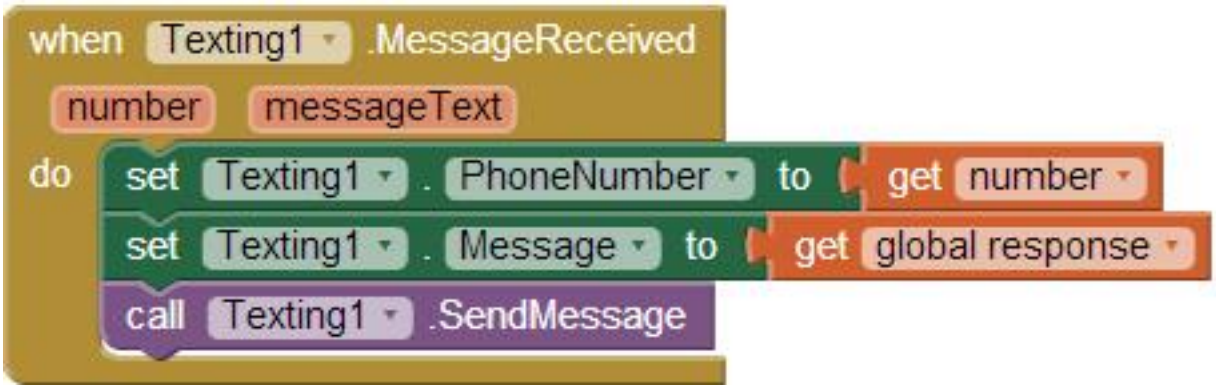




Sarà nostra cura salvare, a seguito del **clic** dell'utente sul pulsante "Modifica risposta automatica", il contenuto della Textbox all'interno del database contrassegnandolo proprio con questo tag. Per farlo utilizzeremo il metodo **TinyDB1.StoreValue**, seguito dal

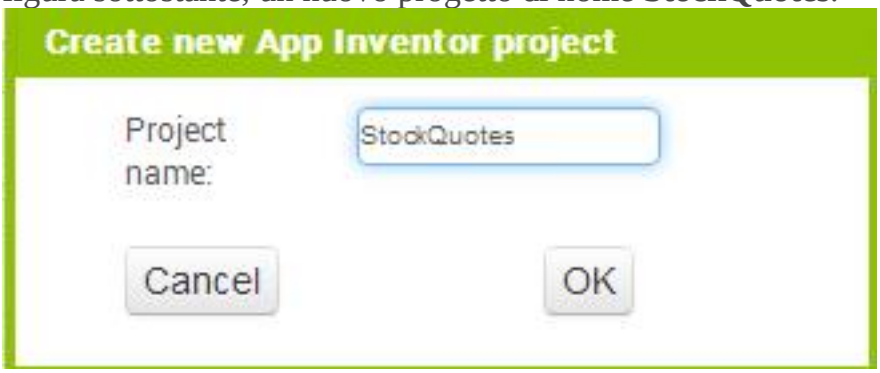
riferimento al tag di archiviazione (ResponseMessage) e dall'indicazione del valore da scriverci dentro (**valueToStore**) che viene recuperato, appunto, dalla nostra TextBox.
Possiamo osservare tale comportamento nella figura sottostante:

In conclusione, dobbiamo gestire la risposta automatica. Le azioni devono essere svolte a seguito della ricezione di un SMS, quindi ci serviremo del blocco **"when Texting1.MessageReceived"**. Eseguiremo, a seguito della ricezione del messaggio, l'azione **"Texting1.SendMessage"**, per inviare la risposta automatica.
Per eseguire correttamente **"Texting1.SendMessage"** però, abbiamo bisogno di impostare il numero telefonico di destinazione (**"Texting1.PhoneNumber"**) e il messaggio di risposta (**"Texting1.Message"**). Per quanto riguarda il numero di destinazione, corrisponde a quello da cui abbiamo ricevuto la chiamata: lo recuperiamo quindi per mezzo di **"get number"**. Il testo del messaggio da inviare, invece, è contenuto nella variabile (globale) **response** che abbiamo definito in precedenza.



17. Interagire con risorse remote API

Nel corso di questa lezione, vedremo come utilizzare nella nostra **app** una **API** (Application Programming Interface) esterna. Si tratta di una risorsa remota, raggiungibile tramite una connessione di Rete, che assolve ad uno specifico compito e ne restituisce il risultato all'applicazione stessa che lo potrà utilizzare come se l'intero processo fosse stato gestito in loco.
Per assolvere a questo compito si fa ricorso al componente "invisibile" **Web** il quale è ricco di proprietà e metodi interessanti per gestire connessioni con risorse remote. Ai fini della nostra lezione vedremo le principali di queste facendo ricorso ad un esempio pratico.
Sempre seguendo i consigli degli sviluppatori di **MIT App Inventor**, infatti, ci concentriamo su un semplice esempio: ci proponiamo cioè di utilizzare le **API** fornite da **Yahoo Finance** per sviluppare una **app** che consenta, al clic su un pulsante (**componente Button**), di visualizzare il prezzo attuale di un'azione, il cui codice è specificato all'interno di una casella di testo (**componente TextBox**). Creiamo quindi, come possiamo vedere nella figura sottostante, un nuovo progetto di nome **StockQuotes**.

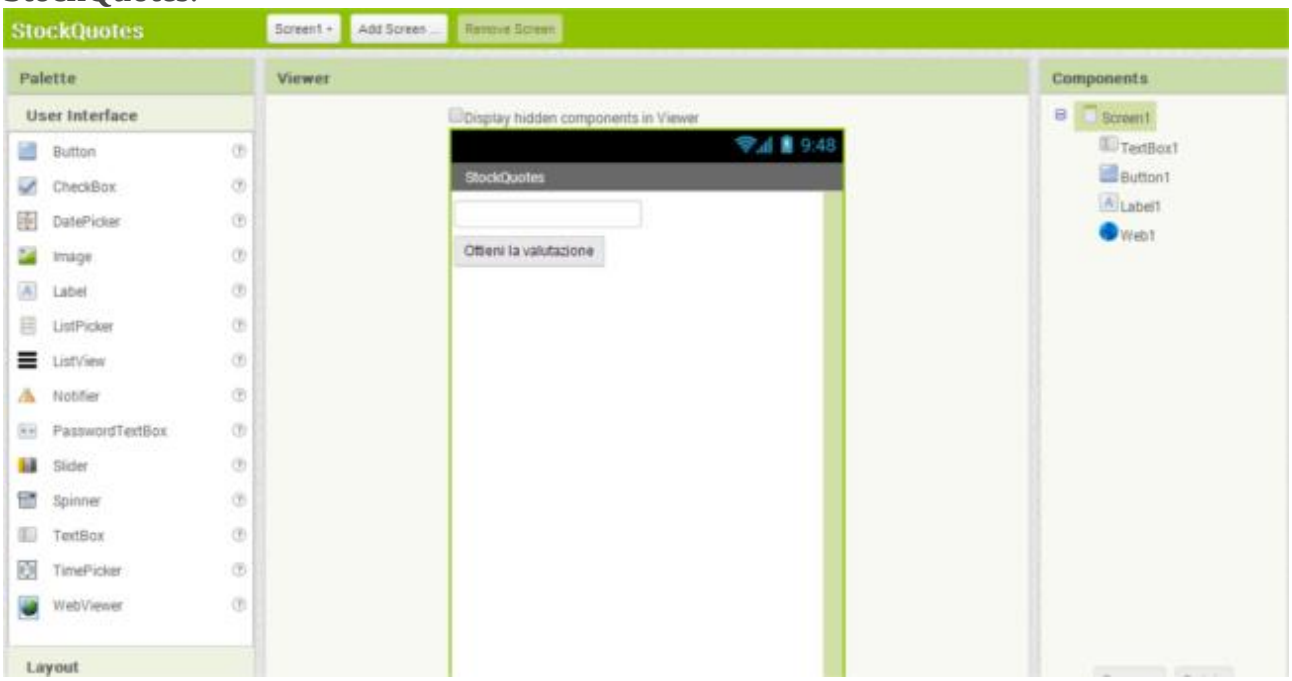


Le **componenti visibili** da inserire sono tre: una di tipo **Textbox** (che ci permette di indicare il codice della quotazione richiesta), una di tipo **Button** (per inviare la richiesta) e una di tipo **Label** (per visualizzare la quotazione, una volta recuperata tramite le API fornite da Yahoo Finance).
Modifichiamo le **proprietà** di queste tre **componenti** come segue:

- impostiamo la proprietà **Text** di **Button1** a "Ottieni la valutazione";
- impostiamo la proprietà **Title** di **Screen1** a "StockQuote";
- impostiamo la proprietà **Hint** (il suggerimento che indica all'utente come riempire la casella, che viene visualizzato quando ci si posiziona su di essa col cursore) di **TextBox1** a "Codice dell'azione da monitorare?";
- impostiamo la proprietà **Text** di **Label1** a ""; tale **Label** dovrà infatti risultare inizialmente vuota, per poi riempirsi con le quotazioni richieste dall'utente.

Inseriremo inoltre la **componente non visibile** denominata **web** (dal pannello **Palette**, categoria **Connectivity**). Tale **componente**, come già detto, consente alla nostra **app** di

gestire le **HTTP Request** di cui avremo bisogno per interagire con il web service di Yahoo Finance.
Nella figura sottostante, possiamo osservare la scheda **Designer** della nostra nuova app **StockQuotes**.



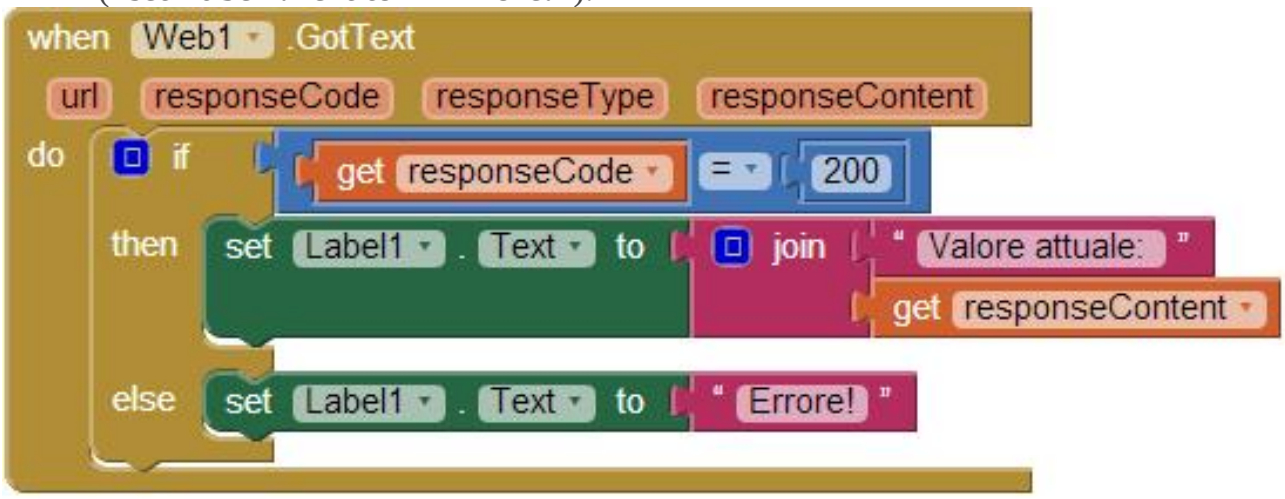
Passiamo ora alla realizzazione dei **behavior**: uno per sfruttare le **API** di Yahoo Finance tramite una richiesta HTTP a seguito del clic sul pulsante e l'altro per il conseguente aggiornamento della label.

Creare una richiesta HTTP attraverso il componente Web

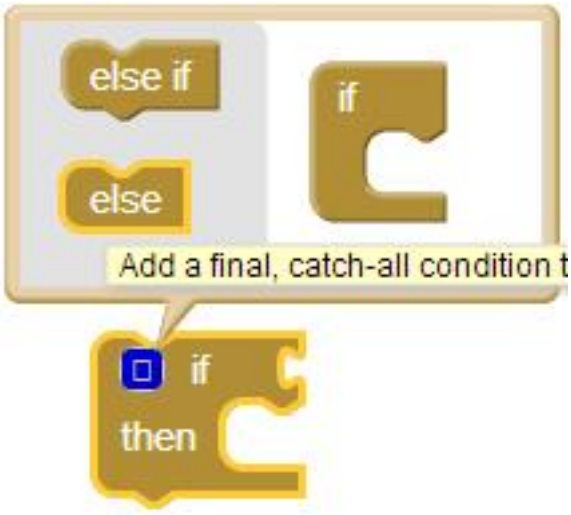
Cominciando dal **behavior** che porta la nostra app a generare una **HTTP Request** prendiamo in esame innanzitutto come ciò ci consenta di ottenere le informazioni di interesse (il valore di una specifica azione). Ciò è possibile perchè gli sviluppatori di Yahoo Finance hanno fatto in modo che tale informazione sia restituita a seguito di una richiesta all'indirizzo web:
http://finance.yahoo.com/d/quotes.csv?f=l1&s=CODICE
sostituendo a "CODICE" l'identificativo dell'azione di interesse (ad esempio, "GOOG" per Google).
Sviluppiamo quindi il **behavior** associando a un blocco **"When Button1.Click"** (al clic sul pulsante), una **HTTP Request** che realizzeremo mediante il componente **Web**: come prima cosa dobbiamo specificare la URL della risorsa remota settando la proprietà Url (**set Web1.Url**) ed in seguito invocare il metodo Get (**call Web1.Get**).
E' bene precisare che otteniamo la URL concatenando (**"join"**, dalla categoria **Text** del pannello **Blocks**) l'indirizzo visto prima (senza CODICE) al codice contenuto nella **TextBox** (**TextBox1.Text**). Il **behavior** appena descritto è rappresentato dai blocchi nella figura in basso:

Resta ora da descrivere, tramite un altro **behavior** il processo di aggiornamento della **Label**. Lo facciamo in questo modo:

- per prima cosa attendo la risposta della API remota mediante l'evento GotText del componente Web ("**When Web1.GotText**");
- creo un costrutto condizionale per verificare se la risposta della API sia di successo (HTTP CODE 200) oppure no;
- in caso affermativo ("**if**" "**get response code**" "**=**" "**200**") aggiorniamo il testo della **Label** col contenuto della risposta, cioè con la quotazione richiesta ("**then**" "**set Label1.Text to**" "**join**" "**Valore attuale: **" "**get responseContent**")
- in caso contrario ("**else**") inseriamo all'interno della **Label** un messaggio di errore ("**set Label1.Text to**" "**Errore!**").



Possiamo osservare che il blocco **if** in cima alla lista **Control** del pannello **Blocks** non prevede una clausola **else**, ovvero "**altrimenti**"). Possiamo però inserire tale blocco e poi modificarlo opportunamente tramite uno degli strumenti più potenti e flessibili messi a disposizione da **MIT App Inventor**, ovvero i **mutator**. Sarà sufficiente cliccare sul piccolo quadrato blu a sinistra di "**if**" nel blocco per poterlo trasformare in un blocco più elaborato, comprendente anche la clausola "**else**" (v.figura sottostante).



L'elenco completo delle proprietà, degli eventi e dei metodi del componente Web sono disponibili in appendice a questa guida di MIT App Inventor.

18. Interagire con sensori

Nel corso di questa lezione creeremo una **app** con **MIT App Inventor** che sarà in grado di regolare il proprio comportamento in funzione delle misure ricevute da **sensori**.
In particolare, ipotizziamo che il dispositivo su cui sarà eseguita sia dotato di un sensore **GPS** per il rilevamento della posizione (tale sensore è presente, ad esempio, su quasi tutti gli smartphone attualmente in commercio): la nostra app stamperà a video la nostra posizione (ovvero Latitudine, Longitudine e Indirizzo) a seguito di un click su un opportuno pulsante. Faremo inoltre in modo, tramite la componente TextToSpeech (già utilizzata nel corso di una precedente lezione) che la posizione corrente sia descritta anche da un messaggio audio.
E' bene precisare che il sensore per il GPS non è l'unico gestito da MIT App Inventor. Segue l'elenco dei componenti della categoria **Sensor**:

- AccelerometerSensor - individua lo shaking e l'accellerazione;
- BarcodeScanner - legge ed interpreta codici a barre;
- Clock - restituisce il momento attuale sulla base delle impostazioni dell'orologio del device e consente di impostare timer ed intervalli di tempo;
- LocationSensor - interagisce col GPS del dispositivo;
- NearField - interagisce con le funzionalità NFC eventualmente integrate nel device;
- OrientationSensor - intercetta l'orientamento del device ed i suoi mutamenti.

Come potete vedere, MIT App Inventor è in grado di "collegarsi" con le principali funzionalità di un moderno smartphone o tablet e di utilizzarle all'interno delle proprie app. In questa sede non approfondiremo in modo didascalico metodi e proprietà di ciascuno dei componenti supportati, ma ci limiteremo a proseguire nel nostro percorso attraverso un esempio significativo che mostra come integrare un sensore all'interno di una nostra app.

Creare una semplice app di geolocalizzazione

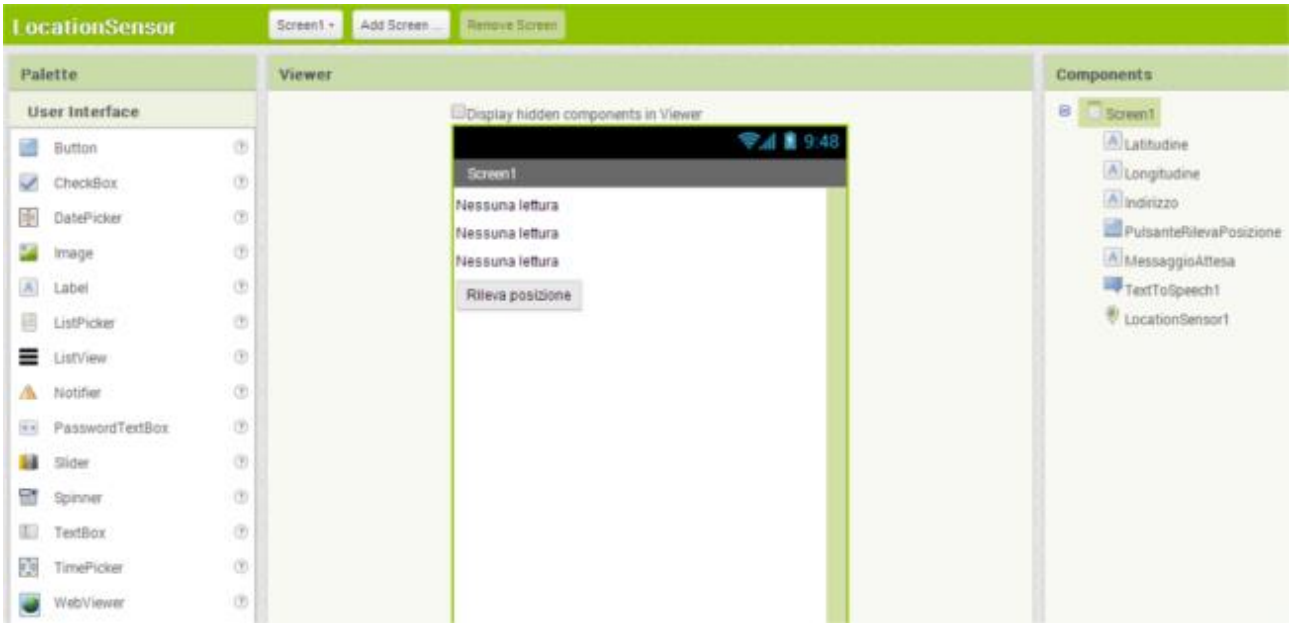
Così come nei casi trattati nelle scorse lezioni, seguiamo il consiglio degli sviluppatori di MIT App Inventor che fanno riferimento proprio a questa app per prendere confidenza con le **componenti** di tipo **Sensor** e i **behavior** a esse associate. Accediamo quindi alla schermata iniziale e creiamo un nuovo progetto, di nome **LocationSensor**, come vediamo nella figura in basso.

La nostra app necessita di cinque **componenti visibili** e due **componenti non visibili**. Per quanto riguarda le **componenti visibili**: quattro **Label** (destinate a Latitudine, Longitudine, Indirizzo e a un **messaggio di attesa**, da visualizzare mentre il **sensore GPS** sta rilevando i dati richiesti) e un **Button** (su cui cliccare per ottenere i dati del **GPS**).
Assegniamo ora a ciascuna delle appena citate **componenti** un nome significativo, in modo da poter distinguere chiaramente, ad esempio, la **Label** destinata a ospitare la Latitudine da quella della Longitudine o dell'Indirizzo, quando ci troveremo in seguito a formulare i **behavior**. Per **rinominare** le **componenti**, possiamo servirci del **pulsante Rename**, in basso nel **pannello Components**. Assegniamo alle quattro **Label** i nomi Latitudine, Longitudine, Indirizzo e **MessaggioAttesa**. Assegniamo al **Button** il nome PulsanteRilevaPosizione.
Impostiamo ora come segue le **proprietà** di queste cinque **componenti**:

- impostiamo la proprietà **Text** delle **Label** Latitudine, Longitudine e Indirizzo a "Nessuna lettura";
- impostiamo la proprietà **Text** di PulsanteRilevaPosizione a "Rileva posizione";
- impostiamo la proprietà **Text** della **Label MessaggioAttesa** a "Attendere che la posizione venga determinata...". Selezioniamo infine, sempre per quanto riguarda la **Label MessaggioAttesa** l'opzione **Hidden** (ovvero nascosta) dal menu a tendina **Visible**, nel pannello **Properties**. In questo modo, nascondiamo, all'avvio dell'app il messaggio di attesa alla vista dell'utente. Tale messaggio dovrà infatti essere visualizzato soltanto nell'intervallo di attesa tra il clic su PulsanteRilevaPosizione e l'aggiornamento delle **Label** Latitudine, Longitudine e Indirizzo. Ci occuperemo in seguito di gestire tale comportamento realizzando i **behavior** della nostra **app**.

Inseriamo infine le **componenti non visibili LocationSensor** (dall'elenco **Sensors** nel pannello **Palette**) e **TextToSpeech**.

Nella figura in basso, possiamo osservare la scheda **Designer** dell'**app LocationSensor**, così come l'abbiamo appena costruita:



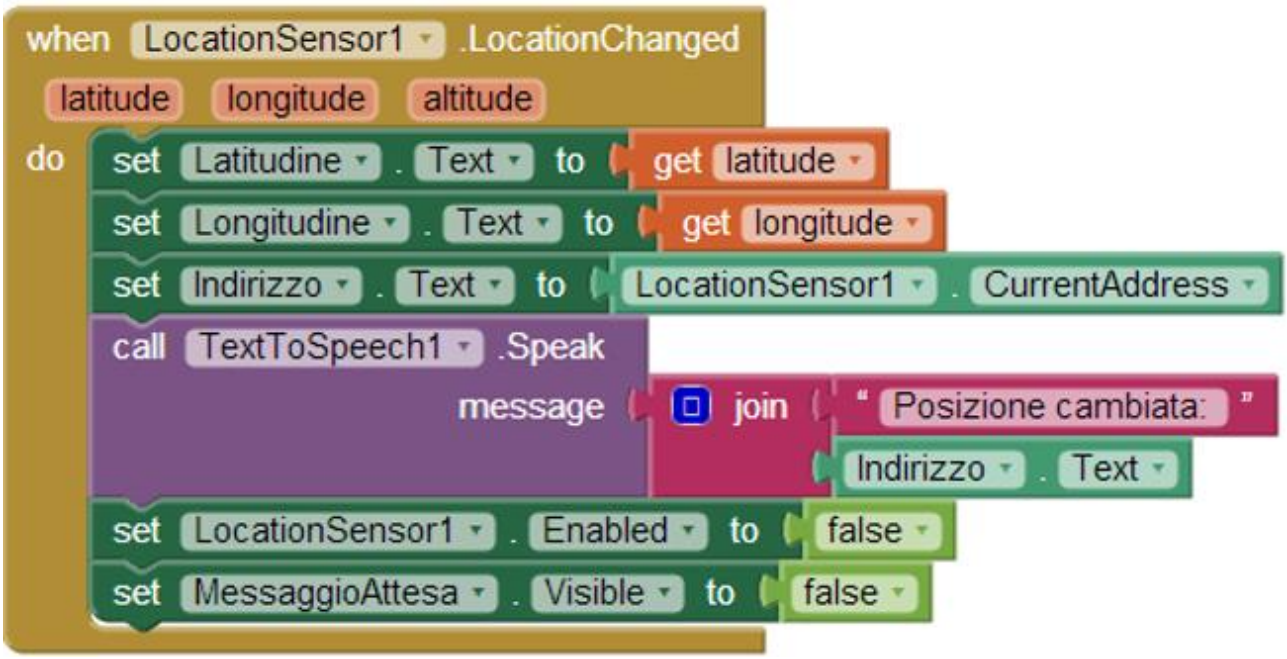
Passiamo ora alla scheda **Blocks** e così alla realizzazione dei **behavior**. Ne sono necessari due: la gestione del clic su PulsanteRilevaPosizione, con conseguente rilevamento della posizione corrente, e l'aggiornamento delle **label** con i nuovi dati.

Per quanto riguarda il primo **behavior**, ci serviamo del blocco **"when PulsanteRilevaPosizione.Click"** e associamo a esso le azioni **"set LocationSensor1.Enabled to" "true"** (ovvero: abilita il sensore GPS) e **"set MessaggioAttesa.Visible to" "true"** (ovvero: visualizza il messaggio di attesa). Possiamo osservare la realizzazione di tale **behavior** nella figura in basso:

Concentriamoci ora sul **behavior** che regola l'aggiornamento delle **Label**. Partiamo, a tale scopo, dal blocco **"When LocationSensor1.LocationChanged"** (attivato a seguito dell'abilitazione del sensore GPS o quando l'utente cambia posizione). A tale evento è associata una sequenza di azioni per l'aggiornamento delle Label Latitudine, Longitudine e Indirizzo:

- **"set Latitudine.Text to", "get latitude"**
- **"set Longitudine.Text to", "get longitude"**
- **"set Indirizzo.Text to", "LocationSensor1.CurrentAddress"**

Trasmettiamo poi un messaggio vocale circa l'indirizzo corrente, tramite **TextToSpeech**, tramite **"TextToSpeech1.Speak"**, costruendo il messaggio assemblando **"Posizione cambiata: "** e **"Indirizzo.Text"** tramite **"join"**.
Infine disabilitiamo il LocationSensor tramite **"set LocationSensor1.Enabled to"**, **"false"** e nascondiamo il contenuto della **Label MessaggioAttesa** con **"set MessaggioAttesa.Visible to"**, **"false"**.
Possiamo osservare la realizzazione di questo secondo **behavior** nella figura in basso:

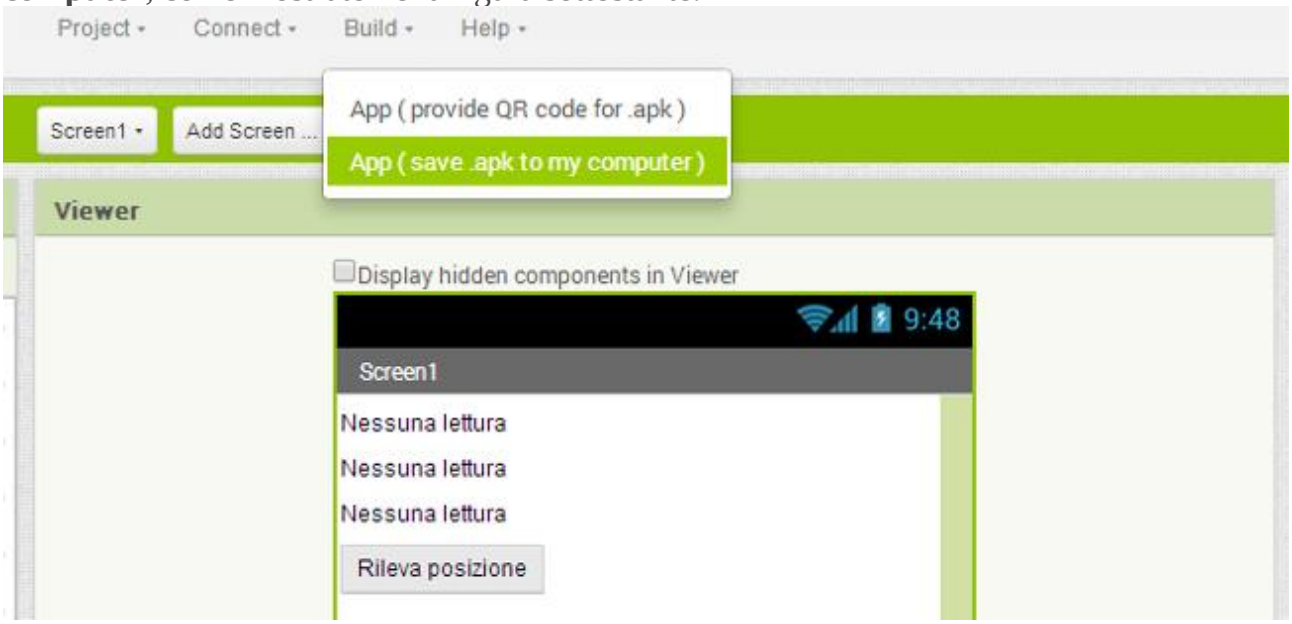


19. Condividere e/o distribuire una app di Mit App Inventor

A conclusione di questa nostra guida allo sviluppo di **app** con la seconda versione di **MIT App Inventor**, esaminiamo le modalità tramite cui è possibile condividere le nostre creazioni. Cominciamo da opzioni utili per una condivisione "diretta", ad esempio con amici, per poi valutare come sia possibile invece pubblicare le nostre **app** su **Google Play** ovvero portarle a far parte dell'archivio globale di app per sistemi operativi **Android**.

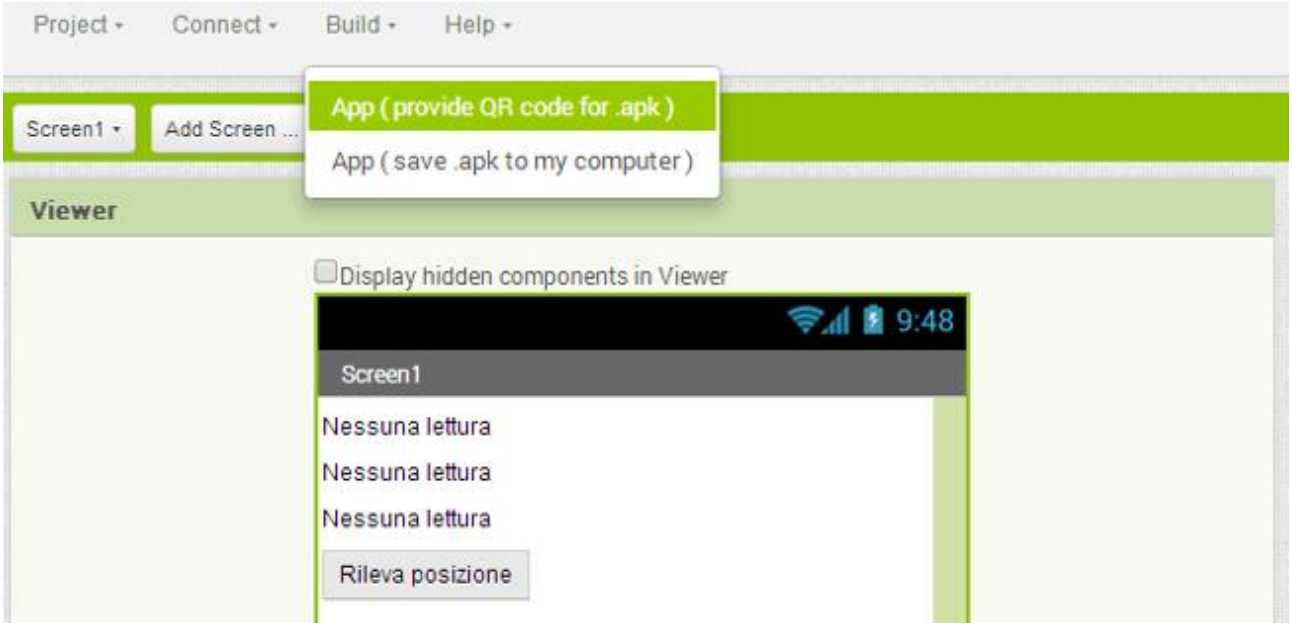
Condividere un file eseguibile

Prima di tutto, esaminiamo come sia possibile garantire ad alcuni amici l'utilizzo della nostra app. MIT App Inventor mette a disposizione una procedura molto semplice destinata a tale scopo. Sarà sufficiente infatti, dalla schermata iniziale, accedere al **progetto** di interesse e selezionare dal menu **Build** (in alto), selezionando poi l'opzione **App save .apk to my computer**, come mostrato nella figura sottostante.



Sul nostro computer verrà scaricato un nuovo file, con estensione **.apk**. Tale file non è altro che la nostra app, che può essere, ad esempio, inviata via mail e poi installata ed eseguita su dispositivi **Android** (avendo cura, per le versioni di**Android** 4 e successive, di abilitare l'installazione e l'esecuzione di app non provenienti da **Google Play**, tramite il menu **Impostazioni**, voce **Applicazioni**).

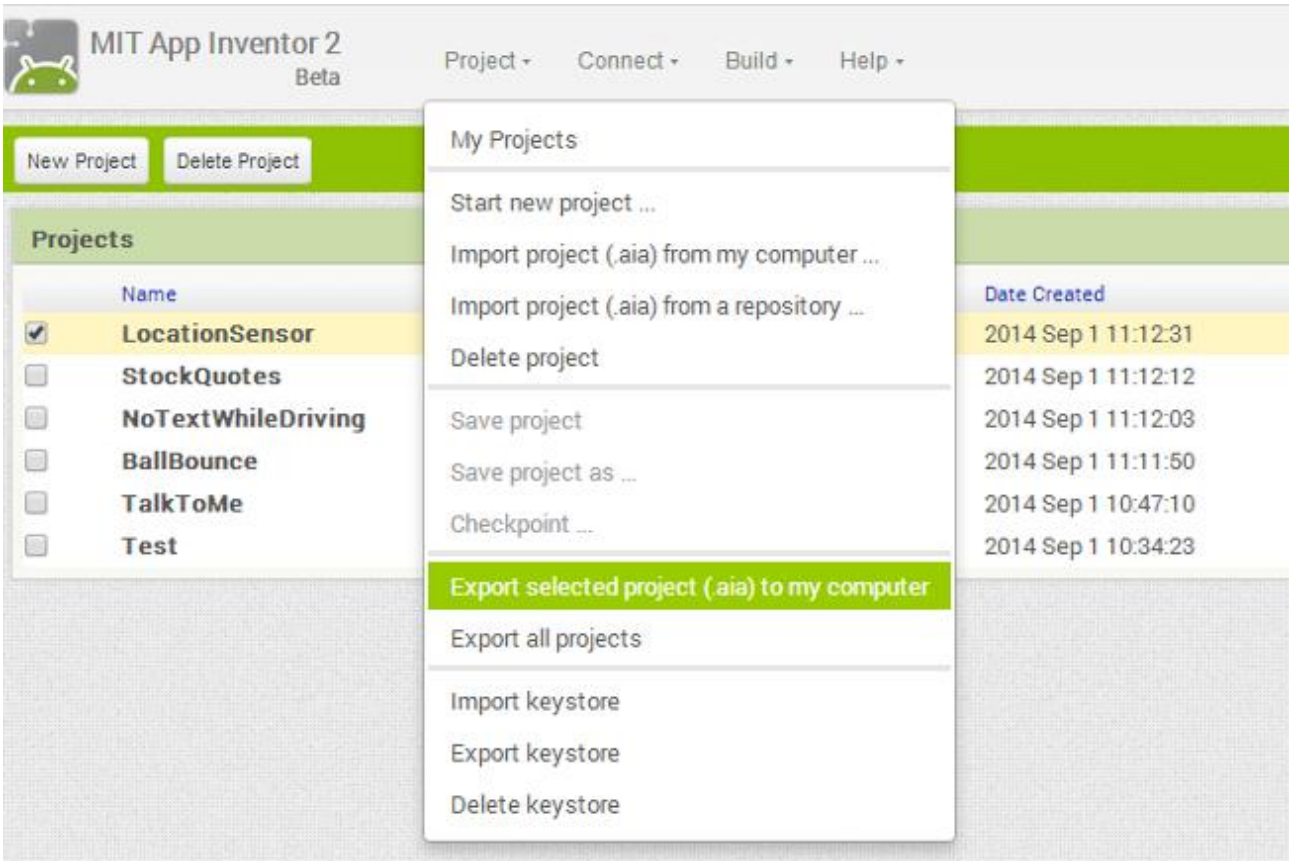
Un'altra opzione è quella di associare la procedura di installazione ed esecuzione della nostra app a un **QR code**: la procedura prenderà il via a seguito dell'osservazione di questo codice, ad esempio con la fotocamera di uno smartphone. Per generare il QR code associato alla nostra app non dovremo fare altro che selezionare l'opzione **App provide QR code for .apk** dal menu **Build**, come vediamo nella figura in basso. Teniamo presente però che il QR code sarà attivo per sole 2 ore.



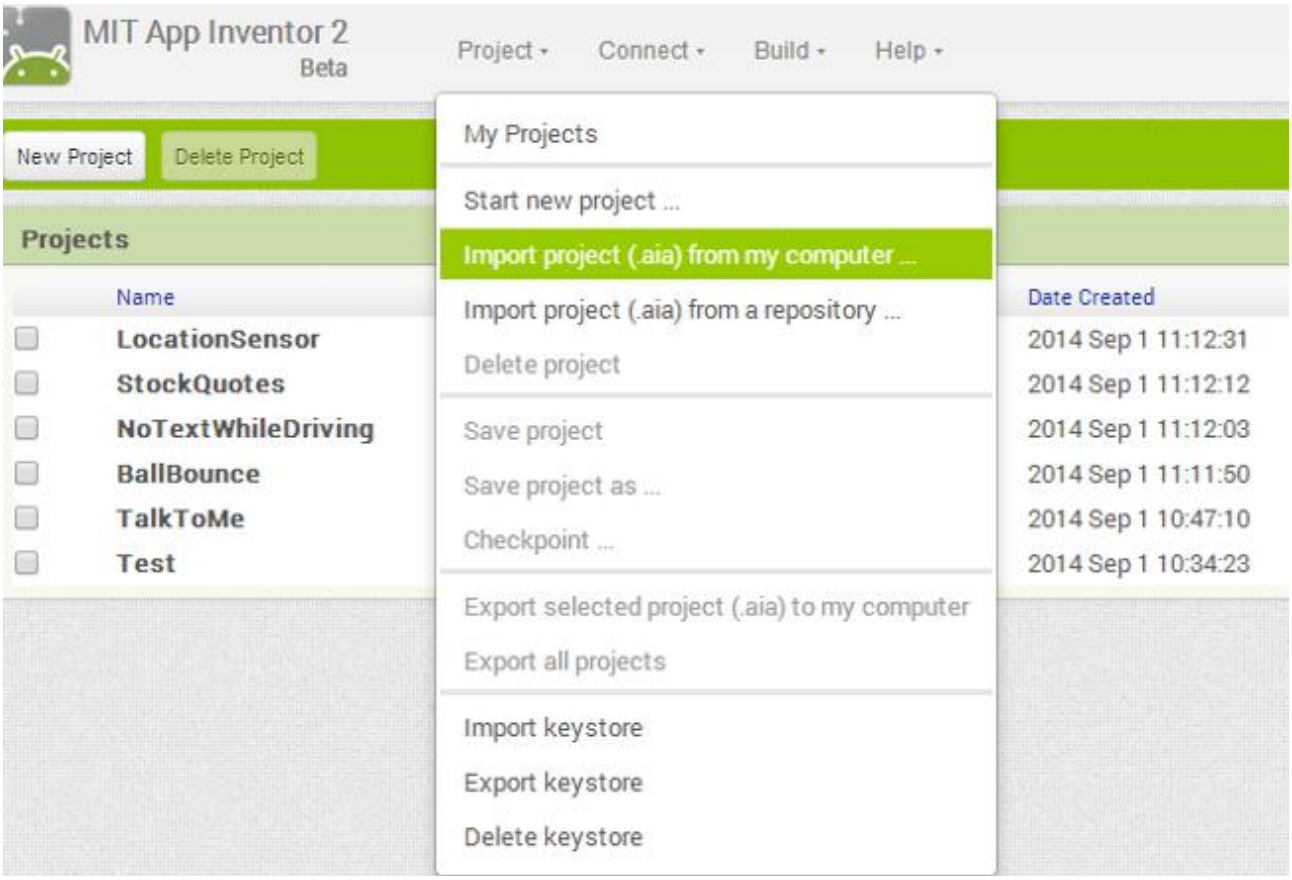
Condividere un progetto

In questa sezione prendiamo in esame una semplice procedura che può consentirci di coinvolgere altre persone nel progetto rappresentato dalla nostra app non come utenti passivi, ma come **sviluppatori**. Ci occuperemo quindi di condividere non una app, ma il **progetto** di una **app**.

Anche in questo caso, potremo generare un file, che rappresenta il progetto, e inviarlo ai nostri collaboratori. I file di questo tipo saranno contraddistinti non dall'estensione **apk**, ma **aia**. Per generare il file di estensione **aia** relativo a un progetto, selezioniamolo dalla schermata **My Projects** e scegliamo poi l'opzione **Export selected project (.aia) to my computer** dal menu **Projects** (v.figura in basso).



Una volta ricevuto il nostro file, il nostro collaboratore non dovrà far altro che **importare** il progetto per averlo a disposizione. Potrà ottenere tale risultato selezionando l'opzione **Import project (.aia) from my computer**, sempre dal menu **Projects** (v.figura in basso) e indicando, quando richiesto, che desidera importare il progetto associato al file da noi inviato.



Pubblicazione su Google Play

Se desideriamo pubblicare la nostra app su **Google Play**, sarà necessario dotarla di un **VersionCode**, numero che indica la versione dell'app stessa (versione 1,2,3 ecc. a mano a mano che si propongono aggiornamenti e migliorie) e di un **VersionName** ovvero di un nome generico. In genere si sceglie anche in questo caso un numero, ma stavolta decimale: ad esempio 1.1 oppure 2.3. A ogni **update** della nostra **app** su **Google Play**, il **VersionCode** dovrà essere incrementato e il **VersionName** modificato. Potremo farlo semplicemente cambiando le proprietà corrispondenti della **componente Screen** della nostra app, dalla scheda **Designer** della schermata principale, menu **Properties**.
Generiamo poi il file corrispondente alla nostra app (con estensione **.apk**), come visto nella prima parte di questa lezione.
Per procedere alla pubblicazione dovremo poi seguire la procedura guidata proposta da Google, come indicato a questa pagina.
Ulteriori dettagli e informazioni, utili specialmente per i neo-sviluppatori possono essere ritrovati agli indirizzi:

- http://developer.android.com/distribute/tools/launch-checklist.html
- https://support.google.com/googleplay/android-developer/answer/113469?hl=en&topic=2365624&ctx=topic