Indovina il numero

Un gioco in Python

Con interfaccia grafica in Kivy

Di che tipo di gioco si tratta?

Il programma estrae un numero a caso tra 1 e 50 e chiede al giocatore di indovinarlo. Per ogni tentativo inserito il programma indica se il numero è troppo alto o troppo basso, oppure se il giocatore ha indovinato.

Questa versione ha un'interfaccia grafica creata con Kivy
http://www.kivy.org

Gli oggetti

Python fa parte di quella categoria di linguaggi che viene detta "orientata agli oggetti". Ma cos'è esattamente un oggetto in un linguaggio di programmazione?

Come gli oggetti fisici, anche quelli che creiamo programmando possono avere **attributi** (es.: colore, dimensioni) e sono in grado di **compiere azioni** (es.: aprirsi, poter essere premuti).

In Python tutto è un oggetto, anche le stringhe di caratteri e i numeri.

Per accedere a un "attributo" o per far compiere un'azione (detta tecnicamente "metodo") a un oggetto si usa la notazione con il punto "."

```
nome = "pippo"
print nome.upper()
```

Nell'esempio qui sopra si esegue il metodo upper() sulla stringa nome: il metodo restituisce la stringa con i caratteri minuscoli convertiti in maiuscolo.

Prova a eseguirlo nell'interprete.
Cosa farà il metodo lower()?

Un modo veloce per sapere quali sono gli attibuti e i metodi di un oggetto è la funzione dir(). Prova a eseguirla sulla stringa di prima:

```
dir( nome )
```

Ovviamente quello che compare a video non serve a molto per capire a cosa servono. Devi per forza guardare la documentazione sul sito di Python, oppure usare l'help dell'interprete:

```
help( nome.upper )
```

Provalo.

Le classi

Gli oggetti dei linguaggi di programmazione, sempre come gli oggetti reali, si possono raggruppare per tipo: ci sono le stringhe, i numeri interi, i numeri con la virgola, le liste, i file, ecc. Il "tipo" in informatica prende il nome di "classe".

Per esempio si dice che il tuo oggetto nome di prima appartiene alla classe "stringa" (sapevi già che era una stringa!), quindi poiché è di quel tipo possiede tutti i metodi della classe:

```
capitalize, center, count, upper, lower, startswith, endswith, find
replace, isalpha, isdigit, join, strip, split, ...
```

e ci si possono fare delle "operazioni":

```
len( nome )   # restituisce la lunghezza della stringa
nome[ 1 ]     # restituisce il secondo (!) carattere della stringa
```

Prova sempre nell'interprete, non ti fidare!

Ogni volta che crei una nuova stringa crei un oggetto della classe str. Ogni volta che crei un nuovo numero crei un oggetto di una delle classi: int, float, long o complex.

Python fa tutto questo senza dirti niente, quindi volendo non c'è bisogno di saperlo per utilizzare il linguaggio.

Quindi?

Quindi si scopre che il grosso del lavoro del programmatore è inventare nuove classi di oggetti (o "estendere", ovvero potenziare, classi esistenti) per fargli fare cose nuove.

Per costruire un gioco in Kivy occorre utilizzare le classi di Kivy ed estenderle per aggiungere quello che serve appunto al gioco per funzionare. Per questo si è resa necessaria questa introduzione alle classi.

La GUI

Altro nuovo termine: GUI sta per Graphical User Interface, ovvero interfaccia grafica per l'utente.

Iniziamo a costruirla.

Crea un file Python; chiamalo "indovina.py"

Questo sarà il file principale che verrà eseguito, quindi dobbiamo inserire gli elementi fondamentali per Kivy: la App e il Widget principale.

La App

Kivy si aspetta che il programmatore estenda la sua classe App e lì dentro costruisca l'interfaccia.

Quindi inizia a mettere l'intestazione al programma e a importare un po' di oggetti di Kivy:

```
#!/usr/bin/env python
# -*- coding: utf8 -*-

from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
```

Adesso introduciamo la sintassi per la creazione delle classi, una riga alla volta.

Crea la nuova classe IndovinaApp che estende la classe App di Kivy:

```
class IndovinaApp ( App ):
```

Non abbiamo idea di cosa sappia fare la classe App di Kivy, ma per ora l'importante è che la nostra nuova classe prenda tutte le funzioni di quella e aggiunga le proprie. Questo è quanto c'è scritto sul manuale e quanto ci serve sapere.

Convenzione da rispettare: i nomi delle classi iniziano con una lettera maiuscola.

Dentro la classe definisci il metodo build (occhio all'indentazione del codice):

```
class IndovinaApp ( App ):  # <- questo e' gia' scritto
        def build ( self ):
```

Il primo parametro self è obbligatorio per tutti i metodi in Python, non lo devi mai dimenticare nella creazione di un metodo nuovo.

Finalmente scriviamo il codice dentro nel metodo build (sempre occhio all'indentazione):

```
class IndovinaApp ( App ):  # <- questo e' gia' scritto
        def build ( self ): # <- questo e' gia' scritto
                game = IndovinaGame ()
                return game
```

Cos'è IndovinaGame()? È la creazione di un nuovo oggetto di una classe IndovinaGame che non abbiamo ancora definito, quindi per ora il nostro programma non funziona.

Riassunto

- I nomi delle classi per convenzione iniziano con una lettera maiuscola: IndovinaGame.
- Quindi i nomi delle altre variabili iniziano con una lettera minuscola: game.
- Per creare un oggetto di una classe si usa il nome della classe seguito da due parentesi tonde, come se fosse la chiamata di una funzione: IndovinaGame().
- Quando si definisce un metodo c'è sempre un primo parametro del metodo ed è self.
- Un metodo si chiama come se fosse una funzione, quindi con le due parentesi tonde, ma senza specificare il self.

Il widget principale

Un widget è un componente grafico di una interfaccia utente di un programma. Il termine deriva dalla contrazione dei termini "window" e "gadget".

In un programma Kivy dobbiamo definire un widget principale "radice" ("root" in inglese) che conterrà tutti gli altri.

Per usare un widget di Kivy si può creare un oggetto di una classe già pronta oppure, se dobbiamo fare delle modifiche, creare una nostra classe ereditando una classe di Kivy e personalizzandola.

Siamo nel secondo caso: ti ricordi la classe IndovinaGame utilizzata in IndovinaApp? Devi creare quella. Inserisci questo codice prima di IndovinaApp:

```
class IndovinaGame ( BoxLayout ):
        pass
```

pass?!? pass non fa niente, ma dopo un ":" inizia per forza un blocco di codice, quindi devi mettere un'istruzione.

Per ora il nostro widget è piuttosto banale: è identico al BoxLayout di Kivy perché ereditiamo tutto da quella classe e non facciamo modifiche.

Il BoxLayout, come dice il nome, organizza gli altri widget come se fossero in una scatola, uno di fianco all'altro, oppure uno sopra all'altro se gli diremo che l'orientamento è verticale. Nella documentazione di Kivy ci sono dei disegni che spiegano molto bene cosa succede.

Il punto di ingresso del programma

Se avvii il programma adesso non fa niente: tutte le istruzioni che hai scritto definiscono delle classi o altri oggetti ma non eseguono nessuna azione che faccia "partire" qualcosa.

Aggiungi in fondo il pezzo mancante:

```
if __name__ == '__main__':
        IndovinaApp ().run ()
```

La riga con if è un'altra convenzione in Python, per ora lasciala così. L'importante è capire che la prima cosa che il tuo programma esegue è IndovinaApp().run().

In dettaglio:

- IndovinaApp() -> crea un oggetto della classe IndovinaApp (vedi le parentesi tonde dopo il nome della classe?);
- run() -> chiama (esegue) il metodo run dell'oggetto appena creato.

Nota che noi non abbiamo mai definito un metodo run nella nostra classe IndovinaApp, ma l'ha ereditato dalla classe "genitore" App di Kivy.

Cosa succeda quando chiamiamo il metodo run non lo sappiamo, però non ci interessa molto: l'importante è che si apra una finestra che, per ora, è completamente vuota.

Riempiamo la finestra

Per creare e aggiungere tutti i widget nella finestra potremmo usare dei comandi in Python, ma Kivy ci mette a disposizione un altro sistema che ci facilita la vita: il file "kv".

Crea un file che si chiama "indovina.kv". Importante: in questo caso il nome del file deve essere uguale al nome della classe ...App, ma senza la parte "App".

La sintassi del file "kv" è abbastanza simile a quella di Python: tutta con ":" e rientri.

La nostra finestra sarà composta da quattro righe:

1. inserimento numero
2. pulsante per controllare se è corretto
3. risposta del programma
4. messaggio di congratulazioni

Ecco il codice:

```
<IndovinaGame>:
	orientation: "vertical"

	BoxLayout:
		Label:
			text: "Il tuo tentativo"
		TextInput:
			id: numero

	BoxLayout:
		Button:
			text: "Controlla se hai indovinato"

	BoxLayout:
		Label:
			text: "numero"
		Label:
			id: ordine
			text: ""
		Label:
			text: "estratto"

	BoxLayout:
		Label:
			id: messaggio
			text: ""
```

Riesci a vedere come sono "inscatolati" i widget?

Ora avvia il programma e guarda cosa compare.

La finestra è decisamente un po' ingombrante.

Per ridimensionarla quando parte il programma devi inserire le seguenti istruzioni nel file Python prima della definizione delle classi:

```
from kivy.config import Config
Config.set ( "graphics", "width", "400" )
Config.set ( "graphics", "height", "120" )
```

width è la larghezza e viene impostata a 400 pixel, mentre height è l'altezza impostata a 120 pixel. Scegli le dimensioni che preferisci.

La logica del programma

In questa nuova struttura dove facciamo estrarre il numero da indovinare? Dove mettiamo la lettura del numero inserito dall'utente e la verifica se ha indovinato?

Per fare tutto ciò modifichiamo la nostra classe IndovinaGame aggiungendo i metodi che faranno i lavori richiesti.

Quindi il primo metodo che ci serve è quello che estrarrà il numero da indovinare. Nella classe IndovinaGame elimina il pass (era lì solo per occupare spazio) e crea il primo metodo vero:

```
def estrai_numero ( self ):
        self.estratto = randint ( 0, 50 )
```

Ricordati di importare la funzione randint da qualche parte all'inizio del programma:

```
from random import randint
```

Ma cos'è questo self?

"self" in inglese vuol dire "stesso", come "myself" -> "me stesso", quindi in Python self indica l'oggetto della classe IndovinaGame su cui stiamo lavorando. La riga

```
self.estratto = randint ( 0, 50 )
```

prende il valore del numero sorteggiato da randint e lo salva nell'attributo estratto dell'oggetto.

Perché in un attributo e non in una variabile? Un attributo è una variabile, ma non viene persa quando il metodo finisce perché rimane nell'oggetto e la potremo leggere in seguito. Questo ci servirà.

Perché il metodo estrai_numero venga eseguito deve essere richiamato da qualche parte. Quindi chiamiamolo dentro build di IndovinaApp. Aggiungilo prima del return:

```
game.estrai_numero ()
```

game è il nome del nostro oggetto della classe IndovinaGame, quindi avrà il nuovo metodo. Torna tutto?

Se esegui il programma non vedi nessuna differenza, ma adesso il numero da indovinare viene calcolato e memorizzato.

Facciamo funzionare il pulsante

Per far svolgere un'azione al pulsante quando viene premuto occorre collegarlo a un metodo che contiene il codice da eseguire.

Nel file "kv", come nuovo attributo del Button, aggiungi:

```
on_press: root.controlla_numero ( numero )
```

on_press significa "quando viene premuto", root è il widget principale, quindi la nostra classe IndovinaGame e controlla_numero sarà il nuovo metodo che dovrai

scrivere. A quel metodo viene passato, come parametro, il contenuto del widget che ha come id "numero". Cercalo: qual è?

Intanto che ci sei aggiungi altre due righe nel file "kv" che servono per fare in modo che alcuni widget definiti lì dentro siano visibili anche nel file Python. Nella seconda riga, dopo <IndovinaGame>, inserisci:

```
ordine: ordine
messaggio: messaggio
```

Sembra strano, ma prima e dopo il ":" c'è lo stesso nome perché per comodità usiamo lo stesso nome sia nel file "kv" che nel file Python. Non è obbligatorio.

Nel file Python aggiungi due righe simili a quelle subito sotto class IndovinaGame:

```
ordine = ObjectProperty ( None )
messaggio = ObjectProperty ( None )
```

Queste quattro righe (2 nel file "kv" e 2 nel file Python) permettono di modificare tramite il codice Python i widget che hanno come id "ordine" e "messaggio".

ObjectProperty è una classe definita da Kivy, quindi devi importarla all'**inizio** del programma, altrimenti Python segnala un errore:

```
from kivy.properties import ObjectProperty
```

Adesso puoi aggiungere il nuovo metodo alla classe IndovinaGame:

```
def controlla_numero ( self, w_numero ):
	try:
		numero = int ( w_numero.text )
	except ValueError:
		numero = 0

	if numero < self.estratto:
		self.ordine.text = "<"

	if numero > self.estratto:
		self.ordine.text = ">"

	if numero == self.estratto:
		self.ordine.text = "="
		self.messaggio.text = "Bravo hai indovinato!"
```

Prova a eseguire il programma e controlla se tutto funziona.

Come esercizio cerca di individuare dove nel codice:

1. viene letto il numero inserito dall'utente. Il valore inserito non viene usato così com'è. Perché? Cosa viene fatto?
2. Il numero inserito dall'utente viene confrontato con quello estratto. Vedi dove viene utilizzato l'attributo dell'oggetto che è stato memorizzato in precedenza?
3. I testi dei widget "ordine" e "messaggio" vengono modificati per comunicare il risultato all'utente.

Risposte

1. Il valore inserito dall'utente viene letto dal widget w_numero che è passato come parametro al metodo controlla_numero. Questo widget ha un attributo text in cui è contenuto il testo scritto dall'utente. Il valore, però, non può essere usato così com'è perché è una stringa, quindi deve essere convertito in un numero intero tramite la funzione int:

   ```
   numero = int ( w_numero.text )
   ```

2. In precedenza hai memorizzato il numero da indovinare nell'attributo estratto dell'oggetto della classe IndovinaGame, quindi per "ripescarlo" per confrontarlo con quello inserito dall'utente devi usare la sintassi self.estratto; la trovi nelle righe tipo questa:

   ```
   if numero < self.estratto:
               ...
   ```

3. I widget "ordine" e "messaggio" sono memorizzati negli attributi della classe IndovinaGame che hanno lo stesso nome. I widget di tipo Label hanno un attributo text che contiene il testo visualizzato, quindi copiando una nuova stringa in quell'attributo si cambia il testo mostrato a video:

   ```
   self.messaggio.text = "Bravo hai indovinato!"
   ```