Appunti di Python

Scritto da Giusti Gianni

giannigiusti@tiscali.it

Indice generale

Che cosa contiene questo documento

Questo documento contiene la mia esperienza personale con python. Il testo, in alcune parti, è scritto "a caldo", intendendo con questo termine, la scrittura di ciò che apprendo, immediatamente dopo averla appresa. Proprio per questo, il codice che troverete non risulterà ottimizzato e probabilmente farà inorridire i guru del python.
Sono ben accetti consigli sul codice, sulla stesura del documento ecc.. Sono ben accette pure le critiche negative, purchè siano fatte con uno scopo costruttivo e non distruttivo.

Qualche ringraziamento

Prima di iniziare vorrei fare alcuni ringraziamenti (in ordine sparso):
Stefano Pardini (http://www.compaiuto.it/) e **Comick** (http://www.comick.net/) per le pubblicazioni sul web.
Panther e Zengar Zambolt per le correzioni e gli utili consigli
Queste persone sono reperibili sul forum del LUG Versilia Lucca Massa e Carrara (come me tra l'altro) www.tivedotonico.it
redclay per il link a glade e pygtk per windows (redclay lo trovate sul NG di python)
Enrico Franchi per le utili informazioni su MacOS (la sezione riguardante MacOS è stata scritta da lui). Il suo indirizzo email è riko@anubics.org

Perchè ho deciso di scrivere il testo

Quando si inizia ad imparare un nuovo strumento di sviluppo, si ha sempre difficoltà a trovare documentazione, esempi, librerie, strumenti ecc... Poi c'è la difficoltà di iniziare... capire come si deve procedere, gli strumenti da usare, come avvengono le installazioni ecc... Tutte cose molto ovvie e banali per chi conosce lo strumento, talmente ovvie che poi ci si scorda di "comunicarle" ai novizi come me.
In questo testo voglio descrivere tutto ciò che incontrerò durante il mio cammino, le difficoltà, gli errori, i motivi delle mie scelte.
Cercherò di scrivere in modo semplice e comprensibile possibile, con un occhio di riguardo per chi non ha mai programmato.
Spero che sia utile a qualcuno. Buona lettura.

Gli strumenti di python scelti per lo sviluppo

Dovendo scrivere gui(interfacce grafiche), per prima cosa ho cercato il modo in cui python permette la scrittura di gui.
Ci sono diverse librerie che si possono utilizzare per tale scopo:
Tkinter, wxWindows, QT, GTK ecc...
Ho escluso Tkinter perchè produce interfacce esteticamente brutte.
Ho escluso QT perchè proprietarie e la versione windows non permette lo sviluppo di applicazioni commerciali.
Le wxWindows sono ottime librerie, ma ho optato per le GTK perchè offrono un ottimo strumento per il disegno delle gui, chiamato glade, inoltre funzionano anche su windows. I programmi,

ovviamente, sono perfettamente integrati con gnome (il mio desktop manager preferito).
Come database ho scelto MySql, ma solo per semplicità e portabilità, visto che sono all'inizio con Python, ho cercato gli strumenti più semplici, in modo da concentrarmi unicamente sullo sviluppo.
Ok, adesso che vi ho detto gli strumenti che userò, se volete seguire la mia strada, dovrete procurarvi:

1) Python 2.3 (http://www.python.org/2.3/)
2) MySql per windows e per Linux (www.mysql.com).
3) Glade (http://glade.gnome.org) (solo per gli utenti Linux). Disponibile su tutte le distribuzioni. Un ringraziamento particolare a **redclay** che mi ha postato questo link: http://www.pcpm.ucl.ac.be/~gustin/win32_ports/pygtk.html in cui, gli utenti windows, possono trovare Glade per windows (di cui ignoravo l'esistenza). Gli utenti windows che installano i pacchetti, si ricordino di inserire nel loro path il percorso delle librerie GTK: C:\Programmi\File comuni\GTK\2.0\bin
4) Il supporto per usare MySql con python: mysql-python (http://sourceforge.net/projects/mysql-python).
5) I binding pyGTK (http://www.daa.com.au/~james/software/pygtk/).
6) Le librerie GTK per windows (ovviamente solo su windows: http://www.dropline.net/gtk/download.php).
7) Un editor di testo a vostra scelta, meglio se supporta la sintassi colorata per python (io uso bluefish, ma potete usare vi, emacs, quanta o quello che vi piace di più).

Qualcuno si starà chiedendo “ci metterò una vita ad installare il tutto!”... non è così:

- I pacchetti non sono grandi (escluse le librerie GTK per windows: 5 Mb ca)
- Le installazioni sono estremamente semplici. Sotto windows è tutto automatico; sotto linux, probabilmente troverete tutti i pacchetti su i cd della vostra distribuzione.

Lo sviluppo si svolgerà tutto sotto linux (si mettano l'anima in pace gli utenti windows), ma le applicazioni generate funzioneranno su entrambe le piattaforme.
Tra i pacchetti installati, avrete notato Glade. Questo programma ci permetterà di “disegnare” le finestre. E' estremamente semplice da usare e ha tutto il necessario...in poche parole: semplice ed essenziale (ottimo per chi inizia).
Chi si aspetta di iniziare a disegnare form, rimmarrà deluso: il passo successivo sarà quello di iniziare a programmare piccoli esempi senza gui.
Questo modo di procedere ci servirà per imparare le poche istruzioni di python. Sì, avete capito bene, python ha poche istruzioni, il resto sono moduli aggiuntive che non tratteremo in questo tutorial.

Informazioni utili per gli utenti MacOS (di Enrico Franchi)

Un buon punto di partenza per Python con il MacOS e`:
http://docs.python.org/mac/mac.html

Per programmare interfacce grafiche sotto MacOS X, le possibilita` sono molteplici.
Le GUI 'native' sono due, Carbon e Cocoa.

Carbon e` il vecchio standard, che funziona anche sotto MacOS classico (in particolare 9 e 8).
Cocoa e` il 'nuovo' framework, fortemente consigliato.
Entrambi possono essere utilizzati con Python.
Per Carbon purtroppo non e` disponibile molta documentazione, per quanto ne so. Se si intende

programmare con Carbon conviene semplicemente leggere la documentazione Apple per il Framework e poi guardare qualche esempio scritto in Python. Ad ogni modo la 'traduzione' dovrebbe essere un compito abbastanza semplice.

Per Cocoa invece il sito di riferimento e`:
http://pyobjc.sourceforge.net/
Cocoa e` un framework molto ben fatto e comodo, originariamente scritto in ObjectiveC e accessibile anche da Java e ora da Python. Per scrivere applicazioni complete e 'mac-only' questa e` probabilmente la strada migliore.

Per MacOS X sono anche disponibili anche alcune librerie multipiattaforma.
C'e` Tkinter sia in versione nativa che in versione X11. Delle soluzioni per X11 parlero` in seguito.
Ad ogni modo qui viene spiegato come fare l'installazione
http://www.astro.washington.edu/owen/PythonOnMacOSX.html

Brevemente e` necessario scaricare il pacchetto Tcl/Tk aqua da qui:
http://tcltkaqua.sourceforge.net/

e in seguito MacPython:
http://homepages.cwi.nl/~jack/macpython/
(Quest'ultimo e` comunque consigliatissimo, poiche` va a 'completare' la distribizione Python di Apple, anche se non si intende usare Tkinter)

A questo punto si puo` usare PacketManager, incluso in MacPython per installare il pacchetto Tkinter.

Tkinter e` davvero facile da usare, tuttavia non e` estremamente accattivante e sembra sempre un po' alieno.

Una soluzione molto piu` 'bella' dal punto di vista visuale e` senza dubbio QT.
Trolltech rilascia una versione GPL delle sue librerie, a patto che il software che le usa sia a sua volta GPL.

http://www.trolltech.com/download/qt/mac.html

Su questo e` necessario installare i bindings per Python:
http://www.riverbankcomputing.co.uk/pyqt/index.php

Qt si appoggia a Carbon per la visualizzazione, in modo trasparente per lo sviluppatore.

Sotto MacOS X e` anche possibile lanciare X11.app. Questa e` una normale applicazione che e` un server X11. E` quindi possibile utilizzarla per utilizzare applicazioni grafiche remote o locali di provenienza *nix.
E` possibile usare Tkinter e Qt (anche se a mio avviso in questo caso sono da preferire, se appena possibile, le versioni native).

E` soprattutto l'unico modo per utilizzare applicazioni scritte con Gtk e Gtk2. E` anche possibile

svilupparle, e` in tutto e per tutto un server X11, con le stesse possibilita` di quelli che si trovano in altri *nix.
La principale differenza e` che le applicazioni X11 hanno un look and feel piuttosto diverso da quello delle classiche applicazioni MacOS (a partire dal menu dentro le finestre fino ad arrivare al tema... che comunque si puo` aquizzare).

Esiste anche un port di Gtk1:
http://sourceforge.net/projects/gtk-osx/

tuttavia non sono ben a conoscenza di come proceda lo sviluppo.

Da dove iniziamo

Per prima cosa, si crei una directory chiamata “progetti” dove metteremo i nostri script python.
Prima di iniziare ricordate che:

1. Gli script dovranno avere l'estensione .py

2. L'identazione del codice (spazi tra il bordo sinistro ed i comandi), fa parte della sintassi python. L'identazione DEVE essere fatta con il tasto tab (quel tasto vicino alla 'q', giusto per capirci) Sbagliare l'identazione significa generare errori
3. Le istruzioni python non hanno una punteggiatura che ne indica la fine.
4. Per eseguire il codice python basta scrivere:
5. python nomescript.py (in realtà non è l'unico modo, ma è quello che seguiremo per semplificarci la vita)

E adesso iniziamo!!

Ok, iniziamo con un semplice esempio:

1) Apriamo il nostro editor
2) Creamo un file chiamato test.py sotto la cartella “progetti” (precedentemente creata)
3) All'interno del file scriviamo:

```
print 'Hello World'
```

4) Adesso apriamo il terminale
5) Posizioniamoci sotto la cartella “progetti” e scriviamo: python test.py
6) Verrà visualizzata la scritta Hello World

L'esempio è semplice e il risultato abbastanza scontato: l'istruzione print stampa a video.
Adesso proviamo ad aggiungere:

```
print 'Hello World'
print “Oggi è una bella giornata”
print “La somma tra 2+2 è:”
print 2+2
```

Se eseguite lo scritp, vi restituirà:
Hello World
Oggi è una bella giornata
La somma tra 2+2 e:
4

Quindi si conclude che:
1) La print funziona anche con doppi apici
2) La print, se incontra numeri con operazioni matematiche, ne restituisce il risultato
3) Tutto ciò che è tra apici e doppi apici viene considerato come carattere generico. Avrete notato che 2+2 tra gli apici non ha prodotto il risultato di 4 ma seplicemente la stampa di 2+2.

Le Variabili

Per chi non sa cosa sono le variabili, possiamo dire che una variabile è un contenitore in cui noi andiamo a mettere un valore numerico, una serie di caratteri e numeri, un valore boleano ecc... Le variabile possono assumere diversi tipi di valore, per semplicità noi vedremo le stringhe e i numeri (intesi come tipo generale):

Stringa (una serie di lettere, numeri, punteggiatura ecc..)
Esempi di stringhe:
“Questa è una stringa”
“Questa, anche se contiene il numero 2 è comunque una stringa”
“2+2”
ecc..
Le stringhe sono scritte tra singoli apici o doppi apici. Se una stringa inizia con un doppio apice, DEVE finire con un doppio apice e viceversa per il singolo apice

ATTENZIONE!!
Notare che “2+2” è una stringa, ciò signifiva che i due numeri non sono considerati numeri ma è come se fossero lettere. Quindi “2+2” non ritorna il valore dell'operazione matematica!

Numerici (un numero intero o decimale)
I numerici si dividono a loro volta, secondo il tipo di valore (intero, doppio, reale ecc..).
Cercheremo di vedere solo i più comuni.
Esempi di numerici:
2
1000
2.2
2.00
ecc..
Sui numerici possono essere applicate le comuni operazioni matematiche di:
+ (somma)

- (differenza)
/ (divisione)
* (Moltiplicazione)
% (Modulo)
Si possono anche usare le parentesi tonde per determinare l'ordine di esecuzione dell'operazione.

Facciamo qualche esempio per capire la differenza tra numerici e stringhe:

```
print '5+2 ='
print 5+2
print “(3*4)/2 =”
print (3*4)/2
```

Il codice sopra produce:
5+2 =
7
(3*4)/2 =
6
Se non riuscite a capire questi esempi, dovete rileggervi il capitolo sopra, poichè significa che non avete capito la differenza tra una stringa ed un numero. Capire questa differenza è ESSENZIALE!!
Proviamo a variare l'esempio in questo modo:

```
print '5+2 =',
print 5+2
print “(3*4)/2 =”,
print (3*4)/2
```

Sono state aggiunte due virgole alla fine delle stringhe. Il risultato cambia in:
5+2 = 7
(3*4)/2 = 6
Si deduce che la virgola unisce, due stringhe o due risultati di due print. Notare che l'unione con la virgola lascia uno spazio tra le due stringhe.
Tornando alle variabili, esse possono essere scritte in con qualsiasi carattere esclusi i numeri se messi come primo carattere. Esempi di variabile sono:
Nome='Gianni'
Nome1='Gianni'
entrambe le variabili sono corrette. Nella variabile Nome ci va il valore stringa 'Gianni', stessa cosa per la variabile Nome1.
ATTENZIONE!!
Le assegnazioni avvengono sempre da destra a sinistra. Scrivere 'Gianni' = Nome, non è corretto!!
Adesso passiamo a vedere qualche esempio di variabile:

```
Nome = 'Gianni'
Cognome = 'Giusti'
Email = 'giannigiusti@tiscali.it'
print 'Io mi chiamo ',Nome,Cognome
print 'La mia Email è:',Email
```

Questo produce:
Io mi chiamo Gianni Giusti

Appunti di Python

La mia Email è: giannigiusti@tiscali.it
poichè:
viene stampato 'Io mi chiamo '
poi c'è una virgola e successivamente la variabile Nome, quindi viene stampato il contenuto di quella variabile. Idem per La variabile Cognome. La virgola unisce due stringhe e inserisce uno spazio tra le due
ATTENZIONE!!
Scrivere: print 'Io mi chiamo Nome Cognome'
sarebbe stato sbagliato perchè Nome e Cognome vengono considerate stringhe....provare per credere.
P.S.
Scrivere il codice sotto è corretto:

```
stringa = '2+2 ='
numero = 4
print stringa,numero
```

Adesso allenatevi un po' con le variabili e con la print. Magari fate qualche piccolo script che faccia qualche operazione di calcoli di volumi e superfici di figure geometriche (tipo: altezza=10 lunghezza=10, superfice =altezza*lunghezza stampate il risultato).

ATTENZIONE!!!
Le variabili sono sensibili al maiuscolo/minuscolo, ciò significa che scrivere ad esempio:

Nome= 'Gianni'
print nome

Non otterremo la scritta Gianni, ma un errore “NameError: name 'nome' is not defined”. Questo perchè la variabile 'Nome' e la variabile 'nome' non sono la stessa variabile. Quando faccio la print, giustamenete per python, la variabile 'nome' non risulta definita.
Quindi, nel caso sopra o cambio Nome in nome o cambio print nome in print Nome.

Nota personale
Nel mio personale modo di dichiarare le veriabili, in genere uso le maiuscole per le prime lettere che mi identificanio in qualche modo le variabili. Esempio:
Se devo creare una variabile contenente il nome ed il cognome di una persona, la chiamerò:
NomeCognome = 'Gianni'
Questo serve solo ad una lettura migliore del codice.

Input da tastiera

Fino ad ora tutto ok, ma se volessimo fare un piccolo programma che calcola l'area di un rettangolo con i valori immessi da un utente?
In questo caso ci vengono in aiuto dei comandi che 'catturano' i valori immessi tramite tastiera. (questa operazione è detta 'input da tastiera', cioè, prendere in ingresso i valori digitati dall'utente)
L'istruzione che permette di prendere dei valori numerici in input si chiama proprio input.
Valore=input('digita un numero')
Questo sopra è un esempio dell'istruzione input. Essa scrive a video 'digita un numero' e aspetta che l'utente immetta dei valori e prema su enter (invio). Fatto ciò assegna il valore immesso alla variabile Valore. Vediamo un esempio:

```
lunghezza = input('Dammi la lunghezza del rettangolo:')
larghezza = input('Dammi la larghezza del rettangolo:')
area = lunghezza * larghezza
print "L'area del rettangolo è",area
```

Eseguite il codice sopra e guardate gli effetti.
Facile vero?
Eseguite l'esempio sopra ma date in input una stringa, noterete che vi viene restituito un errore, questo perchè l'istruzione input si aspetta valori numerici. In realtà è possibile passare valori stringa alla input, se vengono racchiusi tra apici.
Se volete prendere in ingresso valori stringa (alfanumerici), si deve utilizzare l'istruzione raw_input.
La sintassi è la stessa della input.

Prima di procedere vi lascio con un piccolo esercizio da fare:
fate un programma che, sapendo le misure (altezza, lunghezza e spessore) di un blocco di marmo, mi determini quante lastre si possono ottenere, dato uno spessore lastra immesso dall'utente. Dovrete inoltre calcolare i metri quadrati totali delle lastre.
Esempio:

lunghezza blocco = 400 cm
altezza blocco = 150 cm
larghezza blocco (spessore) = 200 cm
spessore lastra = 2 cm
quindi....
N° lastre = 200/2
Area lastre = N°Lastre * (lunghezza * altezza)

beh, praticamente vi ho detto tutto..... :-)

Liste, dizionari e tuple

Liste

Una lista è una sequenza ordinata di oggetti. Detto in parole molto semplici, una lista va vista come una specie di variabile che può assumere più valori contemporaneamente.
Esempio di lista:
listagiorni = ['Lunedì','Martedì','Mercoledì','Giovedì','Sabato']
Nell'esempio, abbiamo creato una lista chiamata listagiorni, contenente i valori 'Lunedì', 'Martedì' ecc...
L'istruzione
print listagiorni[1]
restituisce 'Martedì'
questo perchè il primo elemento di una lista è l'elemento zero.
Esistono molte funzioni interessanti sulle liste. Tra le più importanti:

append = inserisce un nuovo elemento alla lista
esempio:
listagiorni.append ('Domenica')
aggiunge Domenica alla nostra lista

del = elimina un elemento in una lista
esempio:
del listagiorni[0]
elimina Lunedì dalla nostra lista

sort = ordina una lista
listagiorni.sort()
Mette in ordine alfabetico la nostra lista

range() = Essa prende in ingrasso un numero e restituisce una lista contenente una sequenza. Esempio:
listanumeri = range(5)
Questa crea una lista contenente una sequenza di numeri dallo 0 al 4
Se avessi usato l'istruzione range in questo modo:
listanumeri = range(2,5)
avrei orrenuto la lista con sequenza di numeri compresi tra il 2 compreso e il 5 escluso, poichè l'intervallo è di 5 elementi (quindi dallo [0,1,2,3,4]), ma si questi elementi voglio solo quelli dal numero 2 in poi, quindi [2,3,4]

Tuple

Le tuple sono delle speciali liste, il cui contenuto è immutabile. Quindi, non dispone delle istruzioni tipiche della lista.
Un esempio di tupla:
listagiorni = ('Lunedì','Martedì','Mercoledì','Giovedì','Sabato')
Notare che la tupla è definita con parentesi tonde!!

Dizionari

Il dizionario è una collezione, non ordinata, di oggetti. Questo significa che, a differenza di liste e

tuple, gli elementi che la compongono sono identificati da una chiave. Ogni elemento del dizionario è infatti composto da una chiave ed un valore. La chiave serve per il recupero del valore.
Vediamo un esempio:
agenda = {'nome':'Gianni','cognome':'Giusti','email':'giannigiusti@tiscali.it'}
Notare che i dizionari sono definiti tra le parentesi graffe.
Nel nostro esempio, l'istruzione
print agenda['nome']
restituisce 'Gianni'
Anche per i dizionari, come per le liste, esistono molte funzioni interessanti. Tra le più importanti:

len = ci indica quanti elementi compongono il dizionario
esempio:
print len(agenda)
l'istruzione sopra, nel nostro specifico esempio, restituisce 3

has_key = ci indica se esite una specifica chiave. Restituisce 1 se presente, altrimenti 0
esempio:
agenda.has_key('nome')
restituisce 1

keys = restituisce una lista contenente i nomi delle chiavi
esempio:
print agenda.keys()
restituisce ['nome','cognome','email']

values = restituisce una lista contenente i valori. Funziona come la keys vista sopra

del = elimina una chiave e il suo valore, dal dizionario
esempio:
del agenda['nome']

Per aggiungere una chiave e il suo valore, ad un dizionario esistente, vedere l'esempio sotto:
agenda['paginaweb'] = 'www.paginaweb.gia'
l'esempio sopra aggiunge la coppia chiave/valore 'paginaweb':'www.paginaweb.gia'

Per concludere vediamo un esempio simpatico.
Unendo una lista ed un dizionario, possiamo creare una struttura simile ad una tabella di un database. Ammettiamo che voglio fare una tabella in ram, chiamata agenda, contenente il nome, il cognome e gli indirizzi di amici. Basterà fare una lista di dizionari ed il gioco è fatto:

```
agenda = []
agenda.append({'nome':'Gianni','cognome':'Giusti','email':'giannigiusti@tiscali.it'})
agenda.append({'nome':'Comick','cognome':'','email':'info@comick.net'})
agenda.append({'nome':'Rossi','cognome':'Bianchi','email':'rossibianchi@abcde.it'})
```

Nell'esempio abbiamo la lista agenda, composta da 3 dizionari.
print agenda[0]

restitituisce il dizionario {'nome':'Gianni','cognome':'Giusti','email':'giannigiusti@tiscali.it'}

Adesso attenzione:
print agenda[0]['nome']
restituisce 'Gianni'. Questo perchè viene preso il primo elemento della lista agenda, poi, dal dizionario, il valore della chiave 'nome', in questo caso Gianni.
p.s.
Le parentesi graffe con linux, possono essere fatte con i tasti AltGr+7 e AltGr+0

Istruzioni di base

Istruzione if e while

In questa sezione studieremo le istruzioni più usate, quelle istruzioni che prima o poi tutti utilizzano. Iniziamo con le strutture di controllo.
Spesso si ha la necessità di verificare che certe condizioni siamo verificate. Se considerate l'esempio sotto,

```
lunghezza = input('Dammi la lunghezza del rettangolo:')
larghezza = input('Dammi la larghezza del rettangolo:')
area = lunghezza * larghezza
print "L'area del rettangolo è",area
```

noterete che, se l'utente immette numeri negativi, il programma li prende e li elabora, restituendo un risultato negativo. Ovviamente un area negativa non può esistere, quindi ci servirebbe un istruzione che controlla il risultato.... un qualcosa che possa verificare che la variabile area non sia minore di zero, nel caso lo fosse, dovrebbe avvisare l'utente. Quindi:
se l'area < oppure = a zero scrivi “attenzione i dati immessi non sono corretti!”
Listruzione che fa questo si chiama “if” quindi il codice risulta essere:

```
if area<=0:
        print “Attenzione, i numeri immessi non sono corretti!”
        print “Eseguire nuovamente il programma”
else:
        print "L'area del rettangolo è",area
```

Analizzando insieme il codice:
per prima cosa si scrive l'istruzione “if” poi la condizione (in questo caso se area è minore o uguale a zero). Fatto ciò si mette i due punti “:”, si va a capo e si preme sul tasto tab per spostarci di una tabulazione. Questo spostamento è INDISPENSABILE e tutti ciò che si trova spostato di una tabulazione, sotto la “if”, viene eseguita se la condizione è vera. Nel nostro caso specifico, se l'area è minore o uguale a zero, viene stampato a video:
Attenzione, i numeri immessi non sono corretti!
Eseguire nuovamente il programma
Cioè le due print immediatamente sotto. Capite adesso l'importanza della tabulazione.
Avete notato che c'è anche un istruzione chiamata “else”. Essa è strettamente legata alla “if”, in italiano potrebbe essere tradotta con “altrimenti”. Nel nostro caso specifico:
se l'area < oppure = a zero scrivi “attenzione i dati immessi non sono corretti!”
altrimenti scrivi "L'area del rettangolo è",area
Vediamo ora l'esempio completo:

```
lunghezza = input('Dammi la lunghezza del rettangolo:')
larghezza = input('Dammi la larghezza del rettangolo:')
area = lunghezza * larghezza
if area<=0:
        print “Attenzione, i numeri immessi non sono corretti!”
        print “Eseguire nuovamente il programma”
else:
        print "L'area del rettangolo è",area
```

provate ad eseguire il codice, immettendo valori negativi e positivi in input.
Forse alcuni di voi avranno notato che, nel metodo con cui abbiamo risulto il problema dei valori negativi, c'è un errore di concetto:
l'errore di immissione dati andrebbe segnalato immediatamente dopo l'istruzione input. Questo permetterebbe la correzzione immediata del dato, senza il bisogno di rieseguire il programma. Ci servirebbe un istruzione che permette di fare questa domanda:
finchè la lunghezza è <= a zero, non andare avanti e continua a chiedere i valori
Stessa cosa con la larghezza.
In questo caso ci viene in aiuto l'istruzione “while”. La sintassi è la seguente:

```
while lunghezza<=0:
        lunghezza = input('Dammi la lunghezza del rettangolo:')
```

L'istruzione è semplice ed intuitiva, essa ripete tutto quello che si trova sotto di essa, spostata di una tabulazione, fin quando non sarà soddisfatta la condizione. Nel nostro caso specifico, fin quando la variabile lunghezza non sarà minore o uguale a zero, continua a chiedere il valore di lunghezza.
Riguardando il nostro codice originale, possiamo variarlo in questo modo:

```
lunghezza = 0
larghezza = 0
while lunghezza<=0:
        lunghezza = input('Dammi la lunghezza del rettangolo:')
while larghezza<=0:
        larghezza = input('Dammi la larghezza del rettangolo:')
area = lunghezza * larghezza
print "L'area del rettangolo è",area
```

Adesso vi invito a provare il codice, immettendo valori negativi.
La differenza tra l'istruzione “if” e la “while” è evidente, entrambe verificamo una condizione, ma nel caso della “if”, al verificarsi o no di essa, il programma procede. Nel caso della “while”, il codice al suo interno viene eseguito FINO A CHE non si verifica la condizione stessa.
Detto questo, modificatemi il programma sopra, in questo modo:
voglio che il programma, dopo aver calcolato l'area del rettangolo, chieda se voglio calcolare una nuova area. In caso affermativo, mi deve richiedere le misure, ricalcolare l'area e richiedere se voglio calcolare una nuova area. Tutto questo, fin quando l'utente non risponde “N” all'ultima domanda (“Calcolare una nuova area?”). Un piccolo aiuto: attenzione al tipo di instruzione “input” da usare!!

Ciclo For

Il ciclo for, permette di iterare una lista. In poche parole esso permette l'estrazione di tutti i dati di una lista e l'esecuzione del codice identato sotto di esso.
Più facile vedere un esempio che spiegarlo:

```
giorni = ['Lunedì','Martedì','Mercoledì','Giovedì','Venerdì','Sabato','Domenica']
for giornisettimana in giorni:
        print giornisettimana
```

il codice sopra stampa a video i nomi dei giorni della settimana. Analizzando il codice notiamo che il comando for ha la seguente sintassi:
parola chiave **for** + nome **variabile** + parola chiave **in** + **lista** da cui estrarre i dati + **:**
Tutto questo può sembrare un po' strano da chi proviene da altri linguaggi, abituati ad usare il for principalmente per ripetere una parte di codice per n volte. In realtà questo si può fare anche con python, ma non è certo questo l'uso principale che ne viene fatto.
Esempio di ripetizione del codive per n volte:

```
for variabile in range(5):
        print 'Gianni'
```

il codice sopra stampa a video “Gianni” per 5 volte.

Un'altro uso che si può fare del for è l'iterazione di un dizionario:

```
db = {'nome':'Gianni','cognome':'Giusti','email':'giannigiusti@tiscali.it'}
for k, v in db.items():
        print k,'=',v
```

Dall'esempio sopra è stato usato items() che estrare la coppia chiave/valore del dizionario.
Il risultato di quel codice è:

```
cognome = Giusti
email = giannigiusti@tiscali.it
nome = Gianni
```

Funzioni e Moduli

Fino ad ora abbiamo visto i comandi di base che ci permettono di scrivere del codice. Adesso inizieremo a fare programmi un po' più complessi, quindi avremo bisogno di ampliare le nostre conoscenze. La prima cosa che vedremo sono le funzioni.
Le funzioni sono delle porzioni di codice autonomi, atte a svolgere determinati compiti. In ingresso possono prendere dei valori (parametri), elaborarli e restituire un risultato.
I vantaggi nell'uso delle funzioni sono molti, tra i quali:
permettono il riutilizzo del codice
eliminano il codice ridondante
il codice viene diviso in blocchi omogenei
ecc...
Una funzione deve essere scritta con la seguente sintassi:
parola chiave **def** + **nome** funzione + **(** + **parametri** divisi da virgola + **)** + **:**
    istruzioni
    parola chiave **return** + **risultato**
Esempio:
Se volessi creare una funzione che calcola l'area di un rettangolo (giusto per riprendere un vecchio esempio), potrei fare:

```
def AreaRettangolo(lunghezza,larghezza,altezza):
    area = lunghezza * larghezza * altezza
    return area
```

adesso, per utilizzare la funzione basterà richiamarla con il nome e passare i parametri:

```
altezza = 10
largezza = 5
lunghezza = 15
print 'Il rettangolo ha un area di', AreaRettangolo(lunghezza,largezza,altezza) , 'Metri'
```

bene, adesso provate ad eseguire tutto il codice, il risultato sarà la stampa di:
Il rettangolo ha un area di 750 Metri

Attenzione!!
Le variabili usate all'interno della funzione sono dette variabili locali in quanto hanno validità solo all'interno della funzione, ciò significa che sono viste solo all'interno della funzione stessa. Nel nostro esempio specifico, la variabile altezza del programma, NON è la stessa variabile di altezza usata all'interno della funzione. Per chiarezza del codice, io uso mettere una P prima dei parametri, una F prima delle variabili usate all'interno della funzione. Nel nostro esempio specifico avrei scritto:

```
def AreaRettangolo(Plunghezza,Plarghezza,Paltezza):
    Farea = Plunghezza * Plarghezza * Paltezza
    return Farea
altezza = 10
largezza = 5
lunghezza = 15
print 'Il rettangolo ha un area di', AreaRettangolo(lunghezza,largezza,altezza) , 'Metri'
```

Questa nomenclatura non è obbligatoria è solo un mio modo di procedere

Se si definisce un valore al parametro, il parametro diventa opzionale.
Esempio:

```
def AreaRettangolo(Plunghezza,Plarghezza,Paltezza=10):
        Farea = Plunghezza * Plarghezza * Paltezza
        return Farea
```

In questo caso Paltezza è un parametro opzionale, quindi se non lo passo alla funzione, prende il valore di 10

I Moduli sono file .py che possono essere richiamati da codice python.
Avevamo detto all'inizio, che python ha poche istruzioni, (non dovrebbero essere più di 20). Questo potrebbe far credere che python sia limitato, ma non è così. Si è scelto la semplicità, invece di centinaia di istruzioni si è scelto poche istruzioni (quindi maggior semplicità). Questo ha fatto nascere una miriade di moduli fatti da terzi, per la gestione delle problematiche più comuni:
accesso a database, gestione dei file, gestione delle date, stringhe ecc..
Questi moduli non sono altro che funzioni richiamabili dai nostri progetti, tramite l'instruzione import nomemodulo. Chiunque può farsi il proprio modulo ed importarlo nel progetto che vuole.
Adesso pensate alle funzioni e ai moduli, se si uniscono le due cose, vi sarà sicuramente venuta un idea: “allora posso farmi delle funzioni, quelle che più uso, metterle in un modulo e usarle su tutti i progetti che voglio senza riscriverle”.....ebbene sì.
Io mi sono fatto un modulo chiamato utility.py in cui inserisco le funzioni generali dei miei progetti.
Tornando al nostro esempio, possiamo mettere la funzione:

```
def AreaRettangolo(Plunghezza,Plarghezza,Paltezza):
        Farea = Plunghezza * Plarghezza * Paltezza
        return Farea
```

in un file chiamato aree.py
poi nel nostro progetto, chiamato ad esempio lezione.py
potremmo scrivere:

```
import aree
altezza = 10
largezza = 5
lunghezza = 15
print 'Il rettangolo ha un area di', aree.AreaRettangolo(lunghezza,largezza,altezza) , 'Metri'
```

Notare:
import aree = importa il modulo aree.py l'estensione NON serve scriverla
aree.AreaRettangolo(lunghezza,largezza,altezza) = per richiamare una funzione del file aree, si deve specificare il nome del modulo + la funzione da chiamare. (per chi ha già programmato con linguaggi ad oggetti, avrà già capito il perchè si utilizza questa sintassi).

Leggere e scrivere in un file

Adesso vedremo le istruzioni che permettono di legge e scrive in un file. Queste istruzioni, saranno utili, in seguito, per la creazione del nostro primo programma.
Per prima cosa, per operare su i file, è necessario il modulo sys, quindi nel nostro codice dovremmo inserire:
import sys
all'inizio dello script.
Le istruzioni per aprire un file è "open". La sua sintassi è la seguente:
open ('nomefile','modalità')
La modalità ci indica in che modo deve essere aperto:
w = scrittura (write)
r = lettura (read)
r+ = Lettura e scrittura
Se scrivo open('agenda.txt','r'), mi apre il file agenda.txt in modalità di sola lettura.
L'istruzione open restituisce un puntatore al file (per il momento lo chiameremo puntatore, ma in realtà non è il termine esatto). Esso è molto importante perchè ci consente le operazioni sul file stesso, se vedete le istruzioni sotto, capirete la sua importanza:
FileAgenda = open('agenda.txt','r')
Nel caso sopra viene aperto in lettura il file agenda.txt, FileAgenda diventa il puntatore al file.
Non ci soffermeremo molto sull'uso dei file, vediamo solo le istruzioni più usate:

- *Scrittura* - PuntatoreAlFile.write('stringa da scrivere') -
- *Scrittura di una lista* - PuntatoreAlFile.writelines('stringa da scrivere')
- *Lettura* – PuntatoreAlFile.read()
- *Lettura di una riga* – PuntatoreAlFile.readline()

Ovviamente dovete sostituire PuntatoreAlFile, con il nome che avete assegnato al puntatore, in fase di apertura del file.
Una cosa molto importante è chiudere il file dopo le operazioni che vogliamo fare (e comunque prima della chiusura del programma). L'istruzione che fa questa operazione è:
PuntatoreAlFile.close().
L'uso delle istruzioni in un programma, verranno viste in seguito. Se non avete ben chiaro come funzionino, non vi preoccupate, nella lezione sotto vedrete un esempio.

Facciamo il nostro primo "vero" programma, un agenda

Supponiamo di voler fare un agenda che memorizza il nome e il numero telefonico in un file. L'agenda dovrà essere in grado di memorizzare i date, ricercarli e cancellarli. Al momento non gestiremo i casi di omonimia. Il tutto dovrà funzionare dal terminale, quindi per il momento niente interfaccia grafica.
Il programma sarà composto da:
Un menù di scelta che permette di scegliere l'azione da intraprendere
Un interfaccia per l'inserimento, in cui si chiede all'utente, il nome e il telefono da inserire
Un interfaccia per la ricerca, in cui si chiede il nome da ricercare
Un interfaccia per la cancellazione, in cui si chiede il nome da cancellare
Tra i menù, inserirei pure un azione che mi permette la visualizzazione dell'intera agenda
Useremo un dizionario (trattato precedentemente) come archivio in ram, poi salveremo i dati, su

richiesta, in un file di testo chiamato agenda.txt.
Divideremo il progetto in due moduli:
FunzioniAgenda.py, contenente tutte le funzioni generiche di gestione dell'agenda (inserimento, ricerca, cancellazione)
GuiAgendaTerminale.py, contenente le istruzioni di input e output sul terminale.
Agendo in questo modo, in futuro, quando faremo l'interfaccia grafica su gnome, genereremo semplicemente un nuovo modulo chiamato GuiAgendaGTK.py e RIUTILIZZEREMO, senza nessuna modifica, il modulo FunzioniAgenda.py. In parole povere, dividiamo il "motore" vero e proprio della gestione dell'agenda (FunzioniAgenda.py), dalla sua rappresentazione grafica (GuiAgendaTerminale.py e GuiAgendaGTK.py). Ho evidenziato la voce "RIUTILIZZEREMO" perchè questo è un concetto che dovete aver ben chiaro: qualsiasi cosa fate, cercate di farla in modo che possa essere riutilizzata. Per far questo si devono sempre usare funzioni e moduli, ben studiati.

Di seguito, vedremo le funzioni per inserimento, ricerca e cancellazione dell'agenda.

Iniziamo dall' inserimento.
Per prima cosa si deve aver ben chiaro quali sono i dati importanti, coinvolti durante l'inserimento e come deve avvenire. Ci sono 2 dati importanti nell'inserimento: il nome e il numero telefonico. Questi dati, abbiamo detto che saranno memorizzati in un dizionario, la cui chiave sarà il nome il cui valore sarà il numero telefonico. Quindi, la nostra funzione dovrà prendere in ingresso il nome del dizionario a cui aggiungere i dati, il nome e il telefono. I nomi delle funzioni, devono essere rappresentative di ciò che fanno, quindi ho pensato di chiamare AggiungiNumero, la nostra funzione (notare che i nomi delle funzioni sono scritte SENZA spazi). Ok, adesso iniziamo a scrivere:

```
def AggiungiNumero(Agenda,Nome,Telefono):
```

Questa è la prima riga della funzione AggiungiNumero. Vediamola nel dettaglio:
il comando def (come abbiamo già visto), serve a definire una funzione

- AggiungiNumero è il nome che noi abbiamo dato alla funzione
- Quello che sta tra parentesi tonde, sono i parametri (i dati che noi daremo "in pasto" alla funzione)
- Il parametro Agenda è la lista che gli passeremo (la lista contiene tutta l'agenda), Nome il nome e Telefono il numero telefonico da inserire.
- I due punti finali, sono obbligatori e fanno parte della sintassi delle funzioni (e non solo)

Adesso scriviamo il resto:

```
def AggiungiNumero(Agenda,Nome,Telefono):
    Agenda[Nome] =  Telefono
```

Ok, ma come funziona?
Semplice, vengono passati 3 valori. Agenda è il nome della lista che contiene la lista di tutti i nomi e i numeri telefonici, Nome contiene il nome della persona che vogliamo inserire nell'agenda, Telefono contiene il numero telefonico che voglio inserire, associato al nome.
L'istruzione Agenda[Nome] = Telefono non fa altro che aggiungere al dizionario, la chiave Nome ed il valore Telefono. Vediamo un esempio:

```
def AggiungiNumero(Agenda,Nome,Telefono):
   Agenda[Nome] =  Telefono
Agenda = {}
nomedainserire = 'Gianni'
telefonodainserire = '1234567890'
AggiungiNumero(Agenda,nomedainserire,telefonodainserire)
```

L'esempio sopra funziona in questo modo:
Nelle prime due righe viene definita la funzione (e questo lo abbiamo già spiegato)
Poi viene definito il dizionario vuoto, chiamato Agenda (tramite Agenda = {})
Successivamente, vengono definite due variabili stringa: nomedainserire il cui valore è 'Gianni' e telefonodainserire il cui valore è '1234567890'.
A questo punto viene richiamata la funzione, passando i valori nomedainserire e telefonodainserire, quindi (se torniamo a vedere la definizione della funzione AggiungiNumero), di fatto, i valori dei parametri della funzione (Nome e Telefono), diventano rispettivamente, 'Gianni' e '1234567890'. Adesso viene eseguita l'istruzione Agenda[Nome] = Telefono, che si trova all'interno della funzione. Questa istruzione, viene 'tradotta' dal programma in questo modo: Agenda['Gianni'] = '1234567890', questo perchè Nome = 'Gianni' e Telefono = '098078897'. Quindi il programma aggiunge la chiave 'Gianni' al dizionario Agenda, ed associa alla chiave 'Gianni', il valore '1234567890'.
Questa parte sopra è molto importante capirla bene, quindi vi consiglio di leggerle più volte fin quando non vi è tutto chiaro.

Procedura di Cancellazione

Fatta la funzione di inserimento, procediamo col fare la procedura di cancellazione.
Per cancellare i dati (nome e numero), è sufficiente avere il nome (in quanto non gestiamo le omonimie). Quindi dovremo fare una funzione che, passando il dizionario ed il valore della chiave 'Nome', provveda a cancellare il nome ed il numero telefonico. La procedura è semplice:

```
def CancellaNumero(agenda,Nome):
   if agenda.has_key(Nome):
      del agenda[Nome]
```

Questa funzione usa l'istruzione has_key. Essa verifica se la chiave esiste, in caso affermativo cancella la chiave ed il valore, altrimenti avvisa l'utente che il nome non è presente nell'agenda.
Se, ad esempio, si vuol cancellare il nome 'Gianni' dal dizionario agenda, scriveremo;
CancellaNumero(Agenda,'Gianni')
La funzione sopra, prenderà come valore del parametro Nome, il valore 'Gianni'. Successivamente verifica se nel dizionario agenda è presente la chiave 'Gianni': if agenda.has_key('Gianni'):. In caso affermativo, cancella la chiave ed il suo valore: del agenda['Gianni']

Procedura di Ricerca

La ricerca è, in qualche modo, simile alla cancellazione. Anche in questo caso, dovremmo passare come parametri: il nome del dizionario e il nome della chiave che si vuol cancellare.

```
def CercaNumero(agenda,Nome):
   if agenda.has_key(Nome):
```

```
    return "Numero Telefonico "+ agenda[Nome]
```

Non è difficile capire che cosa fa la funzione....
1) verifica se c'è la chiave
2) Se presente restituisce la stringa contenente il numero telefonico

Funzione per visualizzare l'intera Agenda

Per fare questa funzione, sarà necessario passare alla funzione, unicamente il dizionario. Successivamente dovremmo scorrere le chiavi del dizionario e stamparne il nome ed il valore.
In pratica:

```
def MostraAgenda(agenda):
   print "Agenda Telefonica:"
   for x in agenda.keys():
      print "Nome: ",x," \tTelefono: ",agenda[x]
   print
```

La funzione inizia con la stampa a video della scritta “Agenda Telefonica”. Successivamente, con l'istruzione for x in agenda.keys(): , scorre e preleva tutte le chiavi del dizionario. Per ogni chiave ne stampa il valore corrispondente (agenda[x]).
Alcuni di voi avranno notato uno strano carattere (\t), nell' istruzione:
print "Nome: ",x," \tTelefono: ",agenda[x]
esso serve a stampare una tabulazione. In pratica, l'istruzione sopra stampa:
Nome: + chiave + tabulazione + Telefono + valore della chiave
Esempio:
Nome: Gianni Telefono: 098078897

Funzione per il salvataggio dei valori su un file

Questa è già una funzione più complessa. Per prima cosa, dobbiamo decidere in che modo (in quale formato), vogliamo salvare i nostri dati. La cosa più semplice da fare, è il salvataggio con il separatore di campo. Che cosa significa salvare dei dati con un separatore di campo? Semplicemente andremo a scrivere in un file i nostri dati (nome e numero telefonico), separati da un carattere a nostra scelta. In genere, quando si eseguono salvataggi di questo genere, come separatore di campo viene scelto un carattere non usato nei dati: è assurdo, ad esempio, scegliere come separatore una lettera, poiché creerebbe confusione e, in molte situazioni, non si riuscirebbe più a capire, dove finisce il campo stesso. Normalmente, come separatore, viene usato un carattere di punteggiatura (o la virgola o il punto e virgola ecc..). Qualcuno di voi, potrebbe obbiettare che anche questo può generare confusione nei casi in cui, la punteggiatura è presente anche nel campo. In casi come questi, si usa mettere i campi tra i doppi apici.
Nel nostro caso specifico, siamo di fronte a nomi e numeri telefonici. In entrambi i casi, il punto e virgola non sarà mai presente in nessuno dei due campi, quindi lo useremo come separatore di campo. Il formato del nostro file, sarà:
Nome;Telefono + ritorno a capo
Nome;Telefono + ritorno a capo
ecc...
Spiegato più dettagliatamente, abbiamo scelto (per convenzione nostra), di salvare i dati seguendo la seguente sintassi:

Nome + separatore (nel nostro caso il punto e virgola) + Numero telefonico + ritorno a capo
Nella funzione precedente, avevamo visto com' è possibile stampare una tabulazione. Nello stesso modo, per tornate a capo in un file si deve stampare un carattere particolare: \n
Quindi, per la precisione, nel nostro file dovremmo scrivere:
Nome + punto e virgola + Telefono + \n
Nome + punto e virgola + Telefono + \n
ecc...
Vedendo un esempio concreto, nel nostro file ci potrebbero essere dati di questo genere:
Gianni;12345678\n
Panther;7654321\n
....e via dicendo per ogni nome registrato in agenda.

Iniziamo a creare la nostra funzione di salvataggio dati. Per prima cosa scegliamo un nome adatto: quale miglio nome di SalvaAgenda?
Adesso pensiamo ai parametri da passarealla funzione. Sicuramente dovremmo passare tutti i dati, quindi, nel nostro caso, potremmo passare l'intero dizionario. Altro parametro importante e indispensabile è il nome del file in cui vogliamo salvare i dati.
Quindi:

```
def SalvaAgenda(agenda,NomeFile):
```

....e la nostra funzione è definita.

La prima cosa che dovremmo fare nella funzione è aprire il file in cui scrivere i dati. Se vi ricordate bene, per aprire un file sono necessarie due cose:
1) Includere il modulo sys, quindi, nel nostro script, dovremmo scrivere “import sys” all'inizio
2) dovremmo usare la funzione open per aprire il file
Quindi:

```
out_file = open(NomeFile,"w")
```

L'istruzione sopra (come abbiamo già visto), apre un file, il cui nome è contenuto nella variabile NomeFile, in modalità di scrittura (“w”) e ci riferiremo al file stesso, tramite out_file (quindi le istruzioni di scrittura riguardanti quel file, dovranno essere scritte in questo modo: out_file.write ecc..)

Adesso abbiamo aperto il file in scrittura. Prossimo passo è la scrittura dei dati. I nostri dati sono contenuti in un dizionario. Abbiamo detto che i dizionari sono una speciale lista contenente la chiave ed il suo valore. Se noi estraiamo le chiavi, saremmo anche in grado di avere il valore corrispondente. Se usiamo l'istruzione for abbinata alla funzione keys dei dizionari, riusciremmo ad estrarre tutte le chiavi. Ad ogni chiave estratta, saremo in grado di estrarre il valore dal dizionario Agenda, riferendoci alla chiave (es:agenda['Gianni']).
Quindi:

```
for x in agenda.keys():
    out_file.write(x+","+agenda[x]+"\n")
```

Proviamo a tradurre in linguaggio umano, le istrizioni sopra:

Linguaggio Python -->	***For***	***x in***	***agenda.keys()***
Traduzione Italiano ->	Per ogni	valore di x tra	le chiavi dell'agenda

Per ogni chiave dell'agenda trovata, la variabile x diventa uguale al nome della chiave.
L'istruzione successiva potrebbe essere tradotta in questo modo:

Linguaggio Python -->	*out_file.write*	*(x+";"+agenda[x]+"\n")*
Traduzione Italiano ->	Scrivi nel file	Il nome della chiave (x), poi aggiungi il punto e virgola, poi aggiungi il valore del dizionario agenda, la cui chiave è x, poi vai a capo (\n)

Il giochetto della traduzione mi è servito a farvi capire due cose:
1) Python usa delle istruzioni semplici e chiare
2) Cercate sempre di capire cosa fa il codice. All'inizio è difficile, ma poi riuscirete a capire esattamente come funzionano le varie istruzioni ed imparerete a leggere il codice.

Estratti tutti i dati dal dizionario, ci resta solo una cosa da fare: chiudere il file stesso.
Quindi:
out_file.close()

Adesso vediamo l'intera funzione di scrittura su file dei dati:

```
def SalvaAgenda(agenda,NomeFile):
    out_file = open(NomeFile,"w")
    for x in agenda.keys():
        out_file.write(x+","+agenda[x]+"\n")
    out_file.close()
```

Funzione per caricamento dei valori dal file

Se vogliamo caricare la nostra agenda, dovremmo:
1) Aprire il file contenente l'agenda
2) Leggere una riga
3) leggere la chiave ed il valore (es:Gianni,12345)
4) Creare una nuova chiave sul dizionario ed assegnargli il rispettivo valore (es:
agenda['Gianni']='12345')
5) Ripetere tutto dal punto 2, fin quando non è finito il file
Adesso partiamo in modo "traumatico" :-) Scriverò prima la funzione che fa il tutto e successivamente la spiegheremo.
La funzione che legge i dati è la seguente:

```
def CaricaAgenda(agenda,NomeFile):
    in_file = open(NomeFile,"r")
    while 1:
        in_line = in_file.readline()
        if in_line == "":
            break
        in_line = in_line[:-1]
        [Nome,Telefono] = string.split(in_line,";")
        agenda[Nome] = Telefono
    in_file.close()
```

Indubbiamente, in questa funzione, ci sono parecchie novità che meritano di essere spiegate in modo dettagliato.
La cosa che vi sarà sicuramente balzata agli occhi, è un uso strano dell'istruzione while. Se vi

ricordate, l'istruzione while, ripete una porzione di codice fin quando la condizione è avverata. Esempio:
while lunghezza<=0:
significa, ripeti il codice fin quando la lunghezza è minore o uguale a zero.
Nel nostro caso abbiamo scritto:
while 1:
qualcuno si sarà chiesto: “ma la condizione dov'è?”
In realtà quel while ha una condizione che è sempre vera, quindi il codice sottostante sarà sempre eseguito. Questo tipo di ripetizione infinita è detto “loop infinito”. Nella realtà, quello non è proprio infinito, in quanto viene interrotto, ad un certo punto, da un istruzione break (guardate qualche riga sotto). Quindi ne deduciamo che è possibile uscire da un loop tramite l'istruzione break che lo “spezza”.
Altra “strana” istruzione è questa: in_line = in_line[:-1]
Il suo significato è semplice: “assegna alla variabile in_line, il suo stesso valore meno l'ultimo carattere. Nel dettaglio, l'istruzione in_line[:-1] significa “tutti i caratteri dal primo al penultimo”.
Se avessi scritto in_line[2:5] sarebbe significato “tutti i caratteri dal secondo al quinto”. Per capire meglio, guardate gli esempi sotto:

```
s = "Pinco Pallino"
print s[:3]
scrive 'Pin'

print s[3:]
scrive 'co Pallino'

print s[:]
scrive 'Pinco Pallino'

print s[2:-2]
scrive 'nco Palli'
```

Tornando al nostro programma, l'istruzione in_line = in_line[:-1], di fatto elimina l'ultimo carattere. Vi chiederete perchè? Semplice, se vediamo il nostro formato:

```
nome+;+telefono+\n
```

risulta chiaro che il ritorno a capo (\n) ci serviva nel file, ma non lo vogliamo stampare, quindi lo eliminiamo.
E adesso vediamo l'ultima istruzione nuova: [Nome,Telefono] = string.split(in_line,";")
Sembra complicata, ma il suo significato è semplice:
la string.split è una funzione che ci permette di “catturare” delle porzioni di stringhe, separate da un carattere a nostra scelta. Nel dettaglio: string.split(in_line,";") significa “taglia la stringa nei punti separati dal punto e virgola e dammi i valori stringa”.
Se torniamo a vedere il nostro formato (senza il ritorno a capo): nome+;+telefono, avrete già notato che noi abbiamo bisogno di estrarre il nome e il telefono. I due valori (per convenzione nostra), sono separati da un punto e virgola. La funzione string.split(in_line,";") fa proprio ciò che serve a noi. La sintassi è la seguente:
string.split(variabile_contenente_i_valori,"carattere_separatore_di_testo"), nel nostro caso specifico: string.split(in_line,";")
La sintassi [Nome,Telefono] = string.split(in_line,";") significa: “dopo aver estratto i valori di nome e cognome, assegnali alle variabili Nome e Cognome.

ATTENZIONE!!
L'istruzione string.split, fa parte di una serie di funzioni dedicate alle stringhe. Per usare queste funzioni è necessario inserire import string all'inizio dello script

Spero che la spiegazione all'intera funzione, non vi abbia “demolito” :-)

Facciamo un riassunto:

```
def CaricaAgenda(agenda,NomeFile):      -> definisco la funzione ed i suoi parametri
   in_file = open(NomeFile,"r")         -> apro il file contenente l'agenda in lettura
   while 1:                             -> creo un loop, finquando ci sono dati da leggere
      in_line = in_file.readline()      -> leggo una linea del file e assegno il contenuto alla
                                           variabile in_line
      if in_line == "":                 -> controllo se la variabile in_line è vuota, allora il file
                                           è stato letto tutto
         break                          -> se la condizione sopra è vera esci dal loop
      in_line = in_line[:-1]            -> elimina dalla stringa, il ritorno a capo (\n)
      [Nome,Telefono] = string.split(in_line,";") -> leggi i dati nome e telefono ed assegna i valori
                                                     alle variabili Nome e Telefono
      agenda[Nome] = Telefono           -> inserisci la chiave ed il valore nel dizionario
   in_file.close()                      -> Chiudi il file
```

Bene, studiatevi bene le linee sopra e cercate di capire il significato di ogni linea.

Adesso non ci rimane che due cose da fare:
1. Creare un po' di codice che utilizzi le funzioni
2. Creare una sorta di menù che indichi all'utente, le azioni che può fare.

Il menù, possiamo farlo, creando una semplice funzione che stampi a video le possibili azioni che l'utente può fare. Non vi starò a spiegare nel dettaglio la funzione, ve la propongo sotto, ma è chiaro ciò che fa:

```
def print_menu():
   print '----------- AGENDA -------------'
   print '1. Mostra Agenda'
   print '2. Aggiungi un nuovo numero'
   print '3. Cancella un numero telefonico'
   print '4. Cerca un numero telefonico'
   print '5. Carica Agenda'
   print '6. Salva Agenda'
   print '7. Esci'
   print '--------------------------------'
```

Adesso uniamo il tutto con il vero e proprio programma:

Il nostro scopo è quello di permettere di eseguire delle azioni, alla pressione di certi tasti. Esempio: se l'utente preme su 1, dovrà essere mostrata l'agenda, se preme su 2, dovrà esser aggiunto un nuovo numero e via dicendo. Le azioni saranno inserite tramite l'istruzione input.

Con degli if, andremo a controllare ciò che l'utente ha premuto, e, secondo le scelte fatte, saranno intraprese delle azioni e richiamate delle funzioni. Nel codice useremo l'istruzione “elif” essa è utilizzata in abbinamento ad if, in Italiano, potrebbe essere tradotta in “altrimenti se”. L'istruzione if unita ad elif, permette di creare delle scelte annidiate.... strutture particolari che svolgono azioni tipo:
se “a” è uguale a “b” fai questo
altrimenti se “a” è uguale a “c” fai questo
altrimenti se “a” è uguale a “d” fai questo
.....e così via
Ricordate una cosa, i confronti si scrivono in questo modo:
in Italiano “se a è uguale a b” con python “if a==b” (notare i due uguali!!)
in Italiano “se a è maggiore di b” con python “if a>b”
in Italiano “se a minore di b” con python “if a<b”
in Italiano “se a è maggiore o uguale a b” con python “if a=>b”
in Italiano “se a è diverso da b” con python “if a!=b”

Adesso avete tutto il necessario per capire come funziona l'intero programma, quindi vi scrivo sotto l'intero script, ve lo guardate, lo provate e cercate di capire come funziona. Soprattuto il codice finale che non vi ho spiegato nel dettaglio, ma che dovreste essere in grado di capire, con un po' di sforzo, come funziona.
Il consiglio che vi do è quello di stamparvi il codice, eseguire il programma e studiare sul codice come funziona al verificarsi delle scelte dell'utente.
Nella prossima puntata inizieremo ad usare Glade, successivamente separeremo il codice dell'agenda dalla sua rappresentazione visiva e creeremo il programma per glade.
Buon lavoro e buon studio.

Codice completo (salvate con il nome di agenda.py ed eseguitelo con l'istruzione python agenda.py).
ATTENZIONE!!!
E' possibile che le tabulazioni vadano riscritte altrimenti python si arrabbia:

```
import sys
import string

def MostraAgenda(agenda):
   print "Agenda Telefonica:"
   for x in agenda.keys():
      print "Nome: ",x," \tTelefono: ",agenda[x]
   print

def AggiungiNumero(agenda,Nome,Telefono):
   agenda[Nome] = Telefono

def CercaNumero(agenda,Nome):
   if agenda.has_key(Nome):
      return "Numero Telefonico "+agenda[Nome]
   else:
      return Nome+" non trovato"
```

```
def CancellaNumero(agenda,Nome):
    if agenda.has_key(Nome):
        del agenda[Nome]
    else:
        print Nome," non trovato"


def CaricaAgenda(agenda,NomeFile):
    in_file = open(NomeFile,"r")
    while 1:
        in_line = in_file.readline()
        if in_line == "":
            break
        in_line = in_line[:-1]
        [Nome,Telefono] = string.split(in_line,";")
        agenda[Nome] = Telefono
    in_file.close()

def SalvaAgenda(agenda,NomeFile):
    out_file = open(NomeFile,"w")
    for x in agenda.keys():
        out_file.write(x+";"+agenda[x]+"\n")
    out_file.close()


def print_menu():
    print '----------- AGENDA -------------'
    print '1. Mostra Agenda'
    print '2. Aggiungi un nuovo numero'
    print '3. Cancella un numero telefonico'
    print '4. Cerca un numero telefonico'
    print '5. Carica Agenda'
    print '6. Salva Agenda'
    print '7. Esci'
    print '--------------------------------'


DizionarioNumeri = {}
SceltaMenu = 0
print_menu()
while SceltaMenu != 7:
    SceltaMenu = input("Menu (1-7):")
    if SceltaMenu == 1:
        MostraAgenda(DizionarioNumeri)
    elif SceltaMenu == 2:
        print "Aggiungi un nome e il numero telefonico"
        Nome = raw_input("Nome:")
```

```
    phone = raw_input("Telefono:")
    AggiungiNumero(DizionarioNumeri,Nome,phone)
elif SceltaMenu == 3:
    print "Rimuovi il Nome e il relativo telefono"
    Nome = raw_input("Nome:")
    CancellaNumero(DizionarioNumeri,Nome)
elif SceltaMenu == 4:
    print "Ricerca il numero telefonico"
    Nome = raw_input("Nome:")
    print CercaNumero(DizionarioNumeri,Nome)
elif SceltaMenu == 5:
    NomeFile = raw_input("Nome del file da caricare:")
    CaricaAgenda(DizionarioNumeri,NomeFile)
elif SceltaMenu == 6:
    NomeFile = raw_input("Nome del file da salvare:")
    SalvaAgenda(DizionarioNumeri,NomeFile)
elif SceltaMenu == 7:
    pass
else:
    print_menu()
```

Che cos'è Glade

Glade è uno strumento che permette di disegnare in modo visuale, le finestre e tutti i componenti contenuti in essa (caselle di testo, pulsanti, griglie ecc...). I progetti generati da Glade, sono fatti per funzionare con le librerie GTK. Queste librerie sono alla base di Gnome stesso. Esso è scritto basandosi su queste: ogni finestra, pulsante, label ecc... di ogni programma Gnome e lo stesso desktop manager, funzionano grazie alle GTK.
Glade **non** è un programma scritto per il linguaggio python; esso **non genera** codice python. Glade è fatto per generare codice C e C++, quindi è un ambiente di sviluppo nato per facilitare lo sviluppo con questi due linguaggi. In realtà, esso ha una caratteristica molto interessante che lo rende adatto per lo sviluppo di applicazioni con altri linguaggi quali: php, ruby, python ecc..: i progetti fatti con glade, vengono salvati in xml.
Non scenderemo nel dettaglio di che cos'è un file xml, diciamo solo, in modo molto molto approssimativo, che un file xml è una specie di htlm molto particolare (per chi volesse approfondire può facilmente cercare informazioni on-line). In questo file xml, Glade scrive tutte le proprietà della finestra o delle finestre disegnate, e di ogni elemento che lo compongono.
Abbiamo parlato di proprietà della finestra e di ogni elemento, ma che cosa sono le proprietà? Possiamo semplificare dicendo che le proprietà sono proprio le caratteristiche dell' elemento. Per fare un esempio, le finestre hanno molte proprietà tra cui: l'altezza, la larghezza, la posizione, il titolo, il nome, lo spessore del bordo ecc...
Python, non fa altro che leggere il file xml, creato da Glade, e generare la finestra e i suoi componenti, con le proprietà contenute nel file stesso. Tutto questo avviene grazie al modulo PyGTK e alla libreria Libglade. Se volete approfondire meglio l'argomento, circa il funzionamento tecnico del tutto, vi rimando a questo link: http://www.daa.com.au/~james/software/pygtk/
Fino ad ora abbiamo parlato di Python, GTK, Gnome e Glade, ma le applicazioni fatte con questi strumenti generano poi applicazioni multipiattaforma? La risposta è sì se si seguono certe condizioni (per lo meno su windows e linux....forse anche per mac OSX, ma non conoscendo questo SO, non ne sono sicuro, se qualcuno ha info in proposito, sarei grato se mi informasse. Stessa cosa per Beos).
Lo sviluppo verrà fatto su linux (ma nulla vieta che voi utilizziate Glade per windows), ma l'applicazione prodotta, girerà sia su linux che windows.

Primo incontro con Glade

Al suo primo avvio Glade presenta 3 finestre: Glade, Paletta e Proprietà.
Vediamo e spieghiamo che cosa sono e a cosa servono.

Finestra Glade

La finestra Glade è la finestra principale del programma, se si chiude, si chiude il programma stesso. Essa si presenta come nella figura sotto (ovviamente i pulsanti possono variare secondo il tema di Gnome usato):

La finestra è divisa in 4 parti:
1. Barra dei menu (quella in alto dove c'è scritto Project, Modifica, Visualizza ecc...)
2. Barra degli strumenti (Nuovo, Apri, Salva ecc...)
3. Un riquadro bianco (il cui utilizzo lo vedremo in seguito)
4. Una Barra di stato (quella in cui è scritto "Nuovo progetto creato.), in cui vengono stampati messaggi e avvisi

Adesso spiegheremo che cose significano alcuni menù.

ATTENZIONE!!!
NON premete su i pulsanti che spiegheremo ora: **per il momento limitatevi a seguite il testo**, successivamente inizieremo ad usare Glade e ciò che si è spiegato

Con il **pulsante Nuovo** (oppure dal menù Project/Nuovo) è possibile creare un nuovo progetto. Per progetto si intende una o più finestre e i suoi componenti (pulsanti, etichette(label), caselle di inserimento testo ecc...). Tornando all'esempio della nostra agenda, il progetto, potrebbe essere composto da una finestra al cui interno c'è una casella testo per le ricerche, un pulsante ed una griglia che mostra tutta l'agenda.

Il **pulsante Apri** (oppure il menù Project/Apri), ci permette di aprire un progetto precedentemente salvato.

Il **pulsante salva** (oppure il menù Project/Salva), ci permette di salvare un progetto, come abbiamo visto in precedenza, in xml

Il **pulsante opzioni** (oppure il menù Project/Options...), ci permette di impostare le opzioni per il nostro specifico progetto

Il **pulsante Crea eseguibile**, non lo utilizzeremo mai poiché è specifico per il linguaggio C e non serve nel caso di python

Nella barra dei menù è presente "**Modifica**", esso è il classico menù in cui sono presenti i comandi di copia, incolla, taglia e delete (elimina).

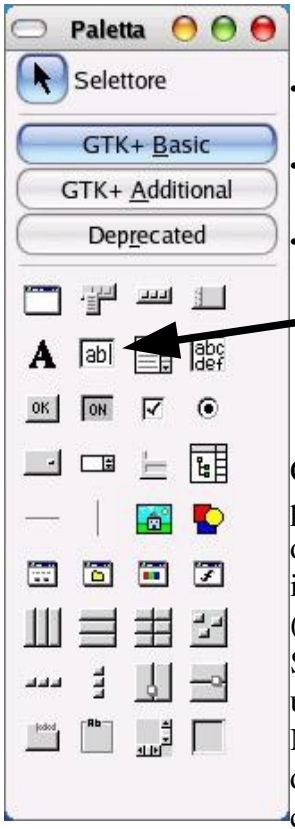
E' interessante vedere il **menù "Visualizza"**. Esso ci permette di scegliere gli strumenti che vogliamo visualizzare in glade, se lo aprite, noterete due caselle spuntate: Mostra Paletta e Mostra editor delle proprietà. Provate a deselezionare le voci per capire a che servono, poi rimetteteli come sono in modo da avere l'ambiente Glade uguale al mio e procedere insieme di pari passo.

Il **menù Impostazioni**, ci permette di impostare il nostro editor. Lo vedremo meglio nel dettaglio in seguito, sarebbe inutile spiegarlo ora

Per ultimo abbiamo un **menù "Aiuto"**...è facile intuire che questo menù contiene la guida a glade

Finestra Paletta

All'apertura di Glade, si sono aperte anche altre due finestre, una di queste è la finestra Paletta (vedi figura sotto).



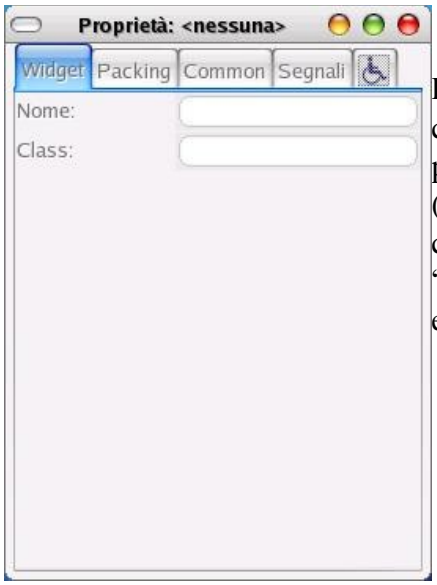
Come potete vedere, essa è divisa in tre parti:

- Nella parte alta, c'è un pulsante "Selettore", esso ci permette di passare in modalità selezione (capirete meglio in seguito che significa)
- Poi abbiamo 4 pulsanti: GTK+ Basic, GTK+ Additional, Gnome e Deprecated (nella figura ne sono presenti solo 3)
- Nella parte bassa, abbiamo tutta una serie di icone. Ogni icona rappresenta un tipo di controllo che si può aggiungere nella finestra. Esempio: l'icona rettangolare in cui è scritto abl (la seconda della seconda riga), ci permette di creare un campo di immissione nella finestra.

Come si può vedere dalla figura, in questo specifico caso è attivato il pulsante GTK+ Basic, quindi vengono mostrati tutti i controlli di base della libreria GTK.....in parole povere le icone relative ai controlli inseribili nella finestra (parte bassa della figura), sono quelli di base (Basic appunto).
Se premo su GTK+Additional, vengono mostrati i controlli addizionali, utilizzabili nelle finestre, stessa cosa per il pulsante gnome e deprecated.
Nel nostro caso specifico, abbiamo bisogno di compatibilità con windows, quindi eviteremo di usare controlli specifici di Gnome (pulsante Gnome) e controlli strani per l'ambiente windows (GTK+ Additional). I controlli Deprecated, non li utilizzeremo perchè, sotto questa voce, vengono inseriti controlli considerati vecchi. Ebbene sì, nelle nostre lezioni utilizzeremo solo controlli presenti sotto GTK+Basic. Comunque questi controlli sono più che sufficienti a creare tutte le applicazioni che desideriamo.

Finestra Proprietà

Adesso vediamo l'ultima finestra che abbiamo nel video, la finestra Proprietà, visibile in figura sotto.



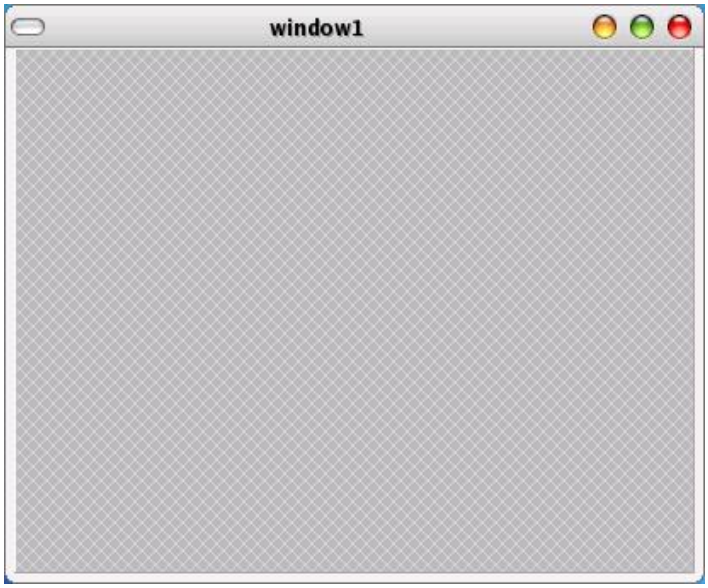
Chi è stato attento, avrà già intuito a cosa serve questa finestra. Essa ci permette di impostare le proprietà di ogni singolo componente della finestra del progetto e delle finestre stesse del progetto. Ricordiamo che ogni componente ha delle proprietà (es: lunghezza, altezza, bordo ecc...). Esso ha anche una sezione chiamata "Segnali" che è molto molto importante, direi che è il "motore" indispensabile al funzionamento del programma vero e proprio. In seguito vedremo meglio di che cosa si tratta.

Primi passi con Glade

Adesso la panoramica di Glade è stata fatta..... direi che possiamo passare all'azione, iniziando a fare qualcosa di veramente semplice: una finestra con 3 pulsanti (inserimento, cancellazione e ricerca), due caselle di inserimento testo (nome e telefono) e 2 label (Le label sono etichette, cioè scritte in genere statiche).
Primo passo da fare è premere sul pulsante Nuovo (o dal menù Project/Nuovo), apparirà una finestra uguale a quella sotto:

Primo problema: che cosa si deve scegliere?
La risposta è semplice, noi vogliamo creare un progetto funzionante con le librerie GTK+, sia su Gnome che su windows, quindi premiamo sul pulsante "New GTK+ Project", per creare un programma generico, funzionante con le librerie GTK+.
Adesso creiamo la finestra in cui inseriremo il pulsante:
dalla finestra "Paletta", premere sul pulsante "GTK+ Basic", poi sulla prima icona dei componenti (la prima icona in alto a destra, quella sopra l'icona **A**, giusto per essere chiari). Se ci andate sopra, vi verrà mostrata la guida con la scritta "Finestra".

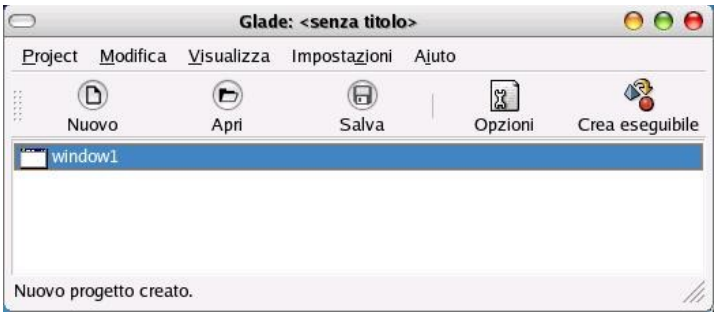
Appena avete premuto sull' icona “Finestra”, verrà mostrata una finestra come quella sotto:



Quella sarà la finestra del nostro progetto, quella in cui andremo a disegnare i pulsanti, le etichette ecc...

ATTENZIONE!!
Da adesso in avanti, chiameremo, le icone sulla finestra “Paletta”, con il nome di “Componenti” o widget. Quindi ciò che abbiamo fatto ora, è stato quello di premere sul componente (o widget) “Finestra” del gruppo “GTK+ Basic”

Notiamo subito alcuni particolari:
La finestra Glade è cambiata in questo modo:



In quella che prima era un area bianca, adesso c'è la scritta “window1”. In quest'area verranno elencate tutte le finestre del progetto. E' utile per passare da una finestra all'altra del progetto in fase di disegno, nel caso il nostro programma fosse composto da più finestre. Adesso fate un click proprio in quest'area, in corrispondenza della scritta window1 o della sua icona.

Guardiamo la finestra Proprietà com'è cambiata. Dovrebbe essere grosso modo come quella sotto:

Il titolo della finestra a fianco dice “Proprietà: window1”, infatti quelle che vedete sono le proprietà della finestra che avete creato. Cerchiamo di capire che cosa sono queste proprietà nel dettaglio (vedremo solo le proprietà più usate)

Nome -> è il nome della finestra (da non confondere con il titolo!!). E' il nome di riferimento che verrà usato da python per riferirsi a quella finestra.

Class -> è difficile spiegarvi che cosa significa Class, se non avete concetti di programmazione ad oggetti. Per farvela breve, diciamo che Class ci indica a che gruppo di componenti appartiene. Tutti i componenti che appartengono allo stesso gruppo, anche se hanno nomi diversi, hanno proprietà identiche. Se create una nuova finestra, essa apparterrà sempre alla stessa Classe. In seguito, vedremo che ciò che c'è scritto nella casella Class, potrà essere utile. **NON VARIATE CIO' CHE C'E' SCRITTO!!**

Spessore Bordo -> è lo spazio, in pixel, che intercorre tra il bordo ed i componenti (provate ad aumentarlo e guardate come varia la finestra, poi riportatelo su zero)

Titolo -> E' il titolo che compare sulla finestra, cambiamolo e scriviamo “La mia Prima Finestra”. Notate ciò che succede alla finestra

Posizione -> Indica in che posizione si deve trovare quando viene visualizzata per la prima volta (all'avvio del programma). Cambiamo il valore e scegliamo “Center”. L'effetto non lo vedete subito, ma durante l'esecuzione del programma python che la richiama.

Resizable -> Indica se vogliamo che la finestra possa essere ridimensionata dall'utente. Lasciamo Sì.

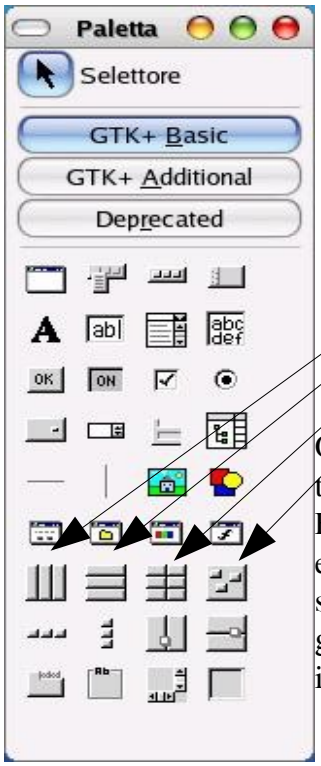
Icona -> Indica l'icona da associare alla finestra. Lasciamo vuoto

Adesso provate a passare sulla scritta Nome, Class, Titolo ecc... soffermandovi qualche secondo su ogni scritta. Noterete che vi viene restituito un suggerimento che vi aiuta a capire che cosa significa quella proprietà.

Vediamo anche alcune proprietà di window1, sulla linguetta Common:

Larghezza -> Indica la larghezza minima della finestra. Nel caso la finestra sia ridimensionabile, quello è il limite minimo relativo alla larghezza. Nel nostro caso è disattivato, quindi non esiste una larghezza minima.
Altezza -> Come sopra ma relativo all'altezza
Visibile -> Indica se la finestra dovrà essere visibile all' esecuzione del programma. Può sembrare una proprietà strana, ma è molto utile su progetti con più finestre (con l'esperienza poi capirete perchè)
Suggerimento -> è il suggerimento che volete mostrare, quando l'utente passa sulla finestra.

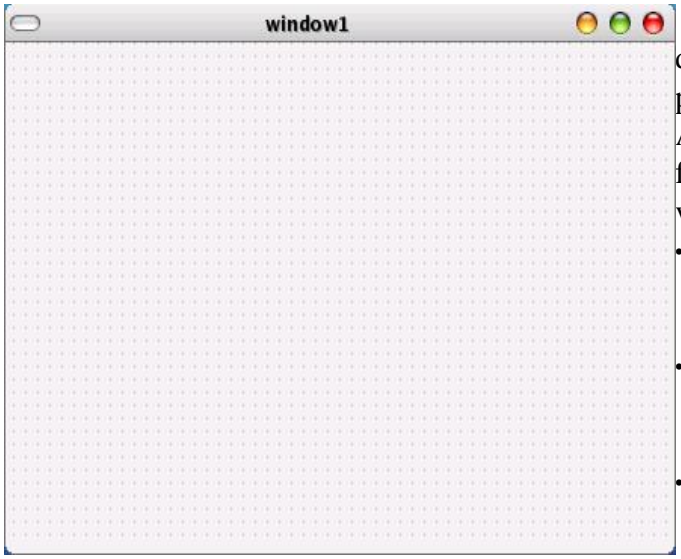
Dopo questa breve e noiosa lezione, passiamo a spiegare come inserire dei componenti della finestra. Per inserire i componenti, dobbiamo anche capire di cosa abbiamo bisogno. NON è possibile (o per lo meno NON è corretto), inserire i componenti direttamente sulla finestra, poiché la finestra accetta un solo componente (widget). Quindi, prima di tutto, dovremmo inserire uno o più speciali pannelli chiamati sizers, adibiti al posizionamento dei componenti. Attraverso i sizers è possibile creare una struttura simile a tabelle, in cui, ogni singola cella, può “ospitare” un solo widget (componente). E' buona norma pensare precedentemente a come deve essere la nostra interfaccia, questo permette una facile scelta dei sizer da utilizzare.
Quando realizzate un nuovo programma, prima di disegnare la finestra o le finestre che lo compongono, è buona norma disegnarle su un foglio, o meglio ancora attraverso dei programmi. Non è importante disegnare il dettaglio della finestra, ma un prototipo, in modo da aver chiaro che cosa si vuol ottenere in fase di disegno con glade.



I sizers che possiamo utilizzare sono quelli indicati dalle freccie

Ogni uno ha le proprie caratteristiche, i più “eleganti” sono i primi 3 (per tutta una serie di ragioni che non vi starà a dire), ma l'ultimo (Posizione Fissa) è il più semplice da usare. Esso supporta più componenti su di esso, ed un posizionamento tramite il trascinamento del componente stesso. Quindi useremo proprio quel componente. Prima di procedere, guardate la finestra, il suo interno è composto da righe oblique incrociate.

Premete sul componente “Posizione Fissa”, poi fate un click all'interno della finestra e notate com'è variata:



Adesso al suo interno compaiono una serie di puntini, quelli sono la nostra guida per posizionare gli oggetti.
Andate sul Menu “Impostazioni” della finestra Glade e vediamo i menù che ci vengono proposti:

- Mostra Griglia -> permette di attivare e disattivare la griglia (provate a farlo e guardate che succede)
- Snap to Griglia -> Significa se vogliamo che i componenti inseriti si spostino solo sui punti
- Impostazioni Opzioni Griglia -> Indica quanto fitta dev'essere la griglia

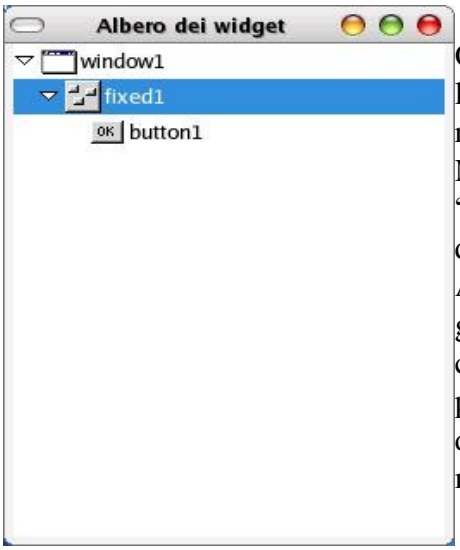
Adesso prendiamo un componente e posizioniamolo sulla finestra:

1. Premete sul componente Pulsante (Il primo della terza riga, quello con scritto **OK)**
2. Spostatevi su un punto all'interno della finestra e premete sul pulsante sinistro del mouse

Ciò che otterrete è questo:

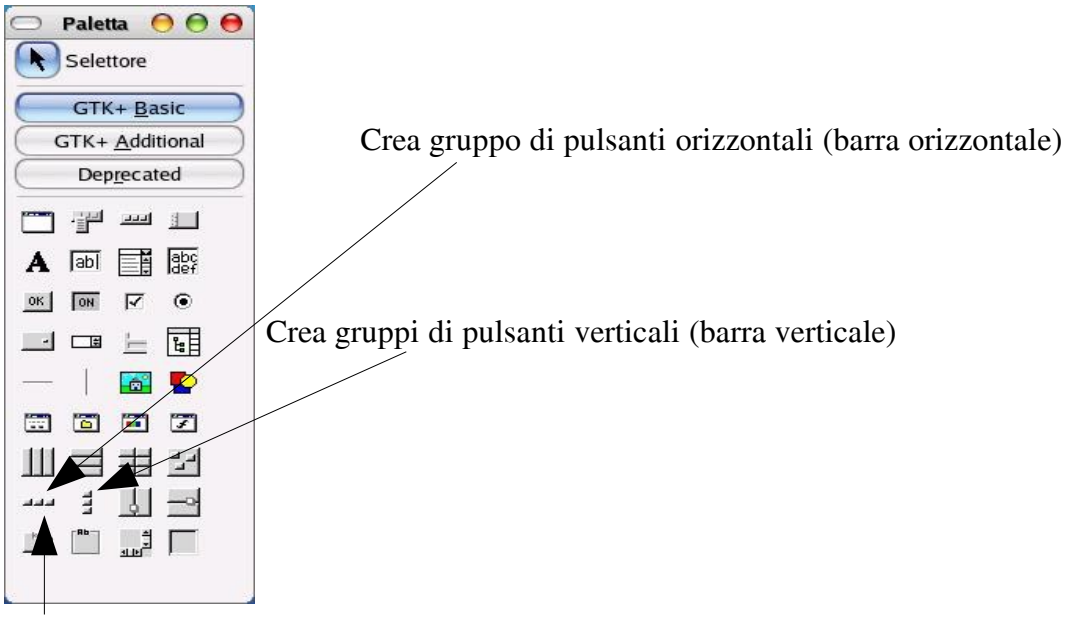
La vostra prima finestra con un pulsante. Adesso premete sul pulsante ed andate a vedere le proprietà di esso. Noterete che alcune proprietà sono uguali a quelle della finestra, mentre altre sono differenti o nuove. Da ciò si deduce che le proprietà variano da componente a componente.

Adesso andate sul menu “Visualizza” e premete su “Mostra albero dei widget”. Apparirà una finestra come questa:



Questa vi visualizza la lista dei componenti (widget) usati e la loro gerarchia, vi permette inoltre, di selezionarli in modo rapido.
Nel nostro caso, vediamo che abbiamo una finestra chiamata “window1”, all'interno di essa c'è un pannello “fixed1”, sul quale è presente un bottone “button1”.
Adesso provate a cliccare su ogni singolo componente e guardate che succede. Notate che alla selezione di un componente, corrisponde la visualizzazione delle sue proprietà e la selezione nella finestra. Avrete senza ombra di dubbio capito l'utilità di questa finestra. Soprattutto se ci sono molti componenti e finestre che compongono il progetto.

Se vogliamo cancellare un componente inserito, basta selezionarlo e premere sul tasto “Canc” o “Del”. Proviamolo subito cancellando il pulsante.
Lo scopo del nostro programma è quello di gestire la nostra agenda. Quindi dovremmo inserire almeno tre pulsanti per gestire la ricerca, l'inserimento e l'eliminazione di nomi e telefoni. Esistono due componenti che ci aiuta a gestire gruppi di pulsanti, uno per pulsanti disposti in orizzontale e uno per gruppi di pulsanti verticali. Essi sono molto comodi perchè raggruppano i pulsanti ad intervalli regolari e li allineano. Noi utilizzeremo pulsanti orizzontali.



Premiamo sul componente per gruppi pulsanti orizzontali, successivamente al centro della nostra finestra. Glade ci chiederà quanti pulsanti (colonne) si vuol inserire, inseriamo 3 e premiamo su OK.

Quello che otterremo è visibile nella figura sotto:

Adesso abbiamo inserito i tre pulsanti, ma dobbiamo cambiare la scritta sul pulsante:
il primo pulsante deve diventare “Ricerca”
il secondo pulsante deve diventare “Inserisci”
il terzo pulsante deve diventare “Elimina”
Per fare questo, dobbiamo selezionare ogni singolo pulsante ed andare sulle proprietà per cambiare la scritta.

Procediamo insieme:

1. fate un click sul primo pulsante.
2. andare sulla finestra “proprietà”.
3. nella proprietà “etichetta”, variate ciò che c'è scritto con la scritta “Ricerca”. Mentre variate la scritta, noterete che anche la scritta sul pulsante varia.
4. direi di assegnare anche un icona al pulsante, per far ciò, andate nella proprietà icona, premete sulla freccia per aprire la lista (combo box), vi apparirà una serie di icone da scegliere. Scegliete l'icona “Trova”.
5. a questo punto, c'è ancora una cosa molto importate da fare, assegnare un nome al pulsante. Questa operazione è molto importante, il nome del pulsante, ci servirà nel codice python, per riferirci a quel componente specifico. Il nome non deve essere mai casuale!!!. In genere io scelgo un nome che ne identifichi il tipo di componente e l'azione che deve svolgere. Nell'esempio specifico, il componente (witget) è un pulsante e l'azione che svolge è inserire un nome ed un numero telefonico nella nostra agenda; quindi un nome possibile potrebbe essere “Bt_Ricerca” (abbreviazione di Bottone di ricerca). Il nome deve essere messo nella proprietà “Nome”, quindi sostituite quello che c'è scritto, con “ Bt_Ricerca”.
6. Ultima cosa che possiamo fare per abbellire il tutto, è mettere un bel suggerimento per l'utente. Andate nella linguetta “Common”, nella sezione “suggerimento” inserire “Ricerca il nome

all'interno dell'achivio". Adesso andate col mouse sul pulsante, fermatevi e guardate ciò che appare.

7. Adesso ripetiamo i punti da 1 a 6, per il secondo e terzo. Essi dovranno avere rispettivamente la scritta "Inserisci" e "Elimina", l'icona "Aggiungi" e "Elimina", i loro nomi saranno "Bt_Inserisci" e "Bt_Elimina" e nei suggerimenti sarà inserito "Aggiunge il nuovo numero" e "Elimina l'utente dall'agenda" (guardate la nota sotto prima di procedere)

ATTENZIONE!!!

- Avrete notato che all'interno del pulsante è possibile selezionare ogni singolo elemento che lo compone (immagine e scritta), quello che noi vogliamo fare è considerare l'intero bottone. Se selezioniamo solo l'immagine, vedremo SOLO le proprietà dell' immagine stessa, ciò che noi vogliamo variare, sono le proprietà del pulsante. Per selezionare l'intero bottone dovete premere nel suo bordo o ancora meglio, utilizzate l'albero dei widget per selezionare i componenti, così non avrete problemi (menu visualizza, mostra albero dei widget)
- NON vi preoccupate se i pulsanti si sovrappongono

Adesso vi ritrovate con i tre pulsanti sovrapposti e forse non perfettamente centrati. Non vi preoccupate, a tutto c'è una soluzione :-) Noi abbiamo i pulsanti posizionati su una barra orizzontale, se spostiamo la barra, spostiamo tutti i pulsanti. Per quanto riguarda la sovrapposizione, è evidente che questa dipende dal fatto che la barra è troppo piccola. Correggiamo queste due cose, procedendo come segue:

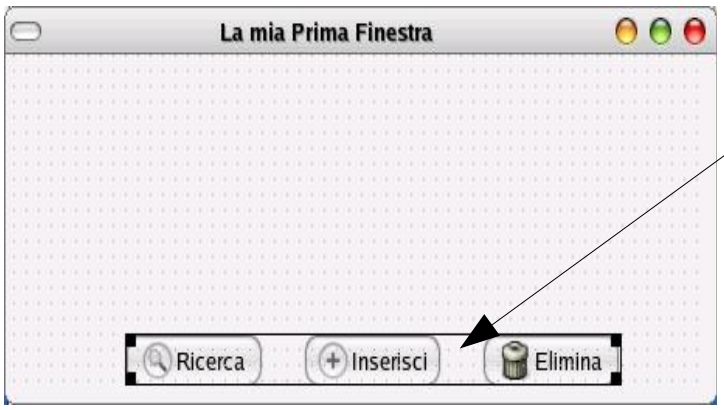
Visualizzate l'albero dei widget (dal menù visualizza).

Selezionare la barra (hbuttonbox1), come in figura.

p.s.
Notate la gerarchia degli oggetti, i pulsanti, come si può vedere, stanno all'interno della barra

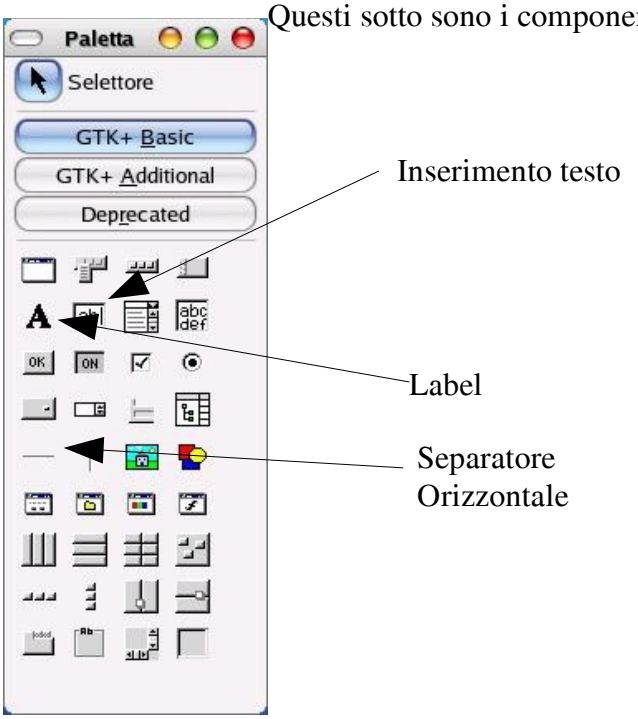
Adesso andiamo a vedere le proprietà della barra (PRESTATE ATTENZIONE): noi dobbiamo ampliare la barra, avrete sicuramente notato la proprietà "dimensione" in cui è scritto "3". La traduzione in Italiano di questa proprietà è un po' forviante, essa infatti, si riferisce al numero dei pulsanti presenti su di essa (tre appunto) e non alle dimensioni fisiche. Le dimensioni della barra, si trovano, selezionando la linguetta Common, in cui è presente la proprietà larghezza. Aumentiamo la larghezza, in modo da creare ampi spazi tra i pulsanti. Fatto questo, selezioniamo la barra, premendo in uno spazio vuoto tra un pulsante ed un' altro e trasciniamola al centro, in

basso della finestra. Vedi figura sotto:



Adesso dobbiamo inserire una casella di “inserimento di testo” in cui l'utente inserirà il nome, ed una in cui verranno inseriti o visualizzati i numeri telefonici.
Ormai siete in grado di farlo da soli, quindi, io vi elenco i componenti necessari, le loro proprietà, e vi faccio vedere come deve venire, voi costruirete il tutto in modo autonomo. Ok, iniziamo.....
Per la ricerca dovrete utilizzare un componente “Inserimento testo” la cui proprietà “nome” dovrà essere “Tx_Nome”.
Per visualizzare ed inserire il numero telefonico, dovrete utilizzare un componente “Inserimento testo” la cui proprietà nome dovrà essere “Tx_Telefono”.
Utilizzeremo inoltre 3 componenti “label” (etichette di testo) e un separatore orizzontale per migliorare l'aspetto.

Questi sotto sono i componenti che utilizzeremo:

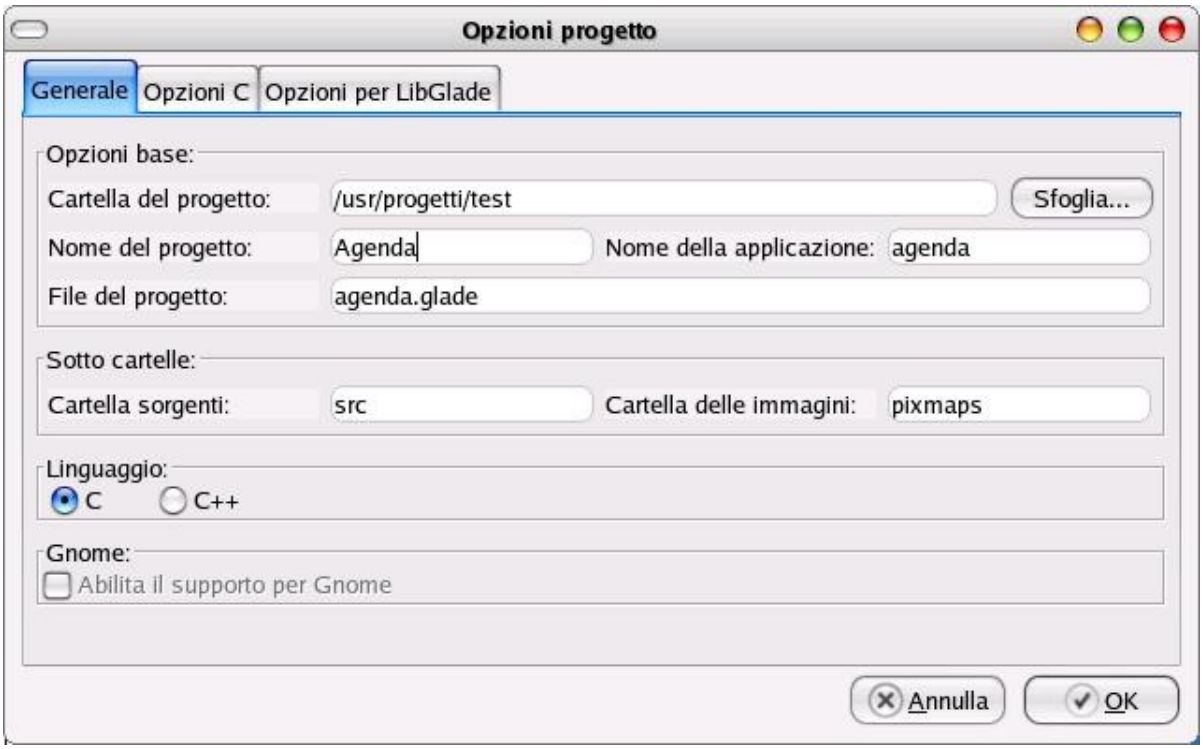


Il risultato che dobbiamo ottenere è visibile nell'immagine successiva:

Alla fine, dovete avere una finestra identica a quella sotto.....
beh, due variazioni rispetto alla finestra sotto ve le concedo, al posto del nome Gianni scrivete il

vostro nome e il titolo della finestra, se volete, variatelo. Che dire, buon lavoro.

Adesso che la nostra finestra è finita, non ci resta che salvare il nostro progetto. Per salvarlo, andate sul menù “project” e scegliere “Salva”. Vi appare la finestra sotto:



Selezionatevi una cartella in cui inserire il progetto, date un nome al progetto e al file che verrà salvato. Tutto il resto non è importante per noi.
Nel caso sopra, il progetto viene salvato nella cartella /usr/progetti/test/ con il nome di “agenda.glade”. Non vi fate ingannare dall'estensione, “agenda.glade” è un file xml.
Nella lezione seguente, vedremo come far funzionare l'applicazione con python.

“Agganciare“ python all'interfaccia creata da Glade

E' arrivato il momento di creare il programma per far funzionare la nostra finestra con python.

Provate a scrivere il testo sotto e salvatelo come agendagtk.py:

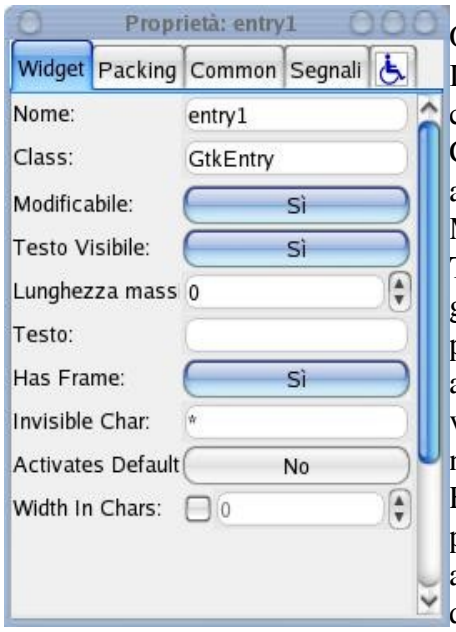
```
# import gtk
import gtk.glade

# Carica Finestra
ApplicazioneGlade = gtk.glade.XML("agenda.glade")
gtk.main()
```

Adesso andate sul terminale e scrivete python agendagtk.py
Vedrete apparire la gui disegnata in tutto il suo splendore ma.... se provate a premere i pulsanti non succede nulla, se provate a chiudere la finestra, si chiude ma il terminale rimane “congelato”. Usate ctrl+c per bloccare l'esecuzione del codice.
E' ovvio che manca ancora qualcosa. Quando noi eseguiamo il codice sopra, di fatto creiamo degli oggetti. Tutti i controlli (pulsanti, finestre, caselle di testo ecc...) inseriti nella nostra applicazione sono oggetti. Ogni oggetto ha delle proprietà e dei metodi. Per semplificare le cose possiamo dire che le proprietà sono tutte quelle caratteristiche che contraddistinguono il singolo oggetto. Un esempio è dato dal nome: tutti gli oggetti hanno un nome. Se consideriamo un campo di immissione testo ha, tra le proprietà, il valore (dato) contenuto in esso, posizione, grandezza ecc...
Quindi abbiamo capito cosa sono le proprietà, ci si poteva arrivare anche dal nome stesso. Ma i metodi cosa sono?
In modo molto semplicistico e rapportato unicamente agli oggetti dell' interfaccia grafica, potremmo chiamare i metodi con un nome molto più comprensibile: eventi. Già il nome “evento” evoca un qualche cosa che si verifica in un determinato attimo o ad una determinata azione, infatti è così. Per fare un esempio, noi vogliamo che alla pressione del tasto “Ricerca” (evento), il programma esegua la funzione relativa alla ricerca e visualizzazione del nome. Quindi gli eventi sono le azioni (funzioni) che vengono eseguite al verificarsi di un azione (evento). In glade, gli eventi vengono chiamati segnali. Ora, avrete capito quello che manca alla nostra applicazione: il legame tra l'evento e l'azione (funzione), quindi quando noi premiamo sul pulsante ricerca (evento), non c'è definita nessuna azione (funzione). Allo stesso modo, quando chiudiamo la finestra (evento), non viene interrotta l'applicazione.
Rileggete questi due paragrafi, cercate di capire il concetto, poi richiamate glade ed il progetto agenda e vediamo in pratica.

Proprietà e segnali (eventi)

Se richiamiamo il nostro progetto con glade, richiamando la nostra finestra e premendo

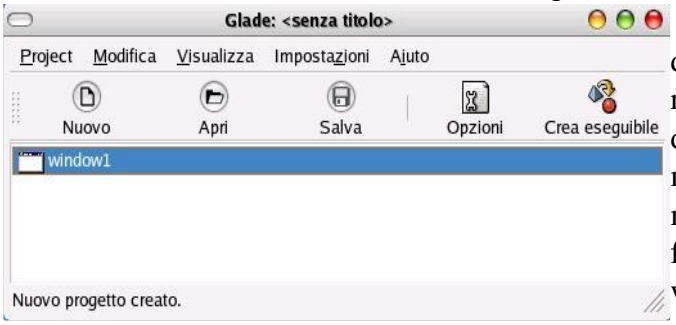
sull'oggetto di immissione testo relativo al nome, vedrete una finestra simile a quella sotto (se non la vedete potete richiamarla dal menu “visualizza/mostra editor delle proprietà”:



Quelle sono le proprietà dell'oggetto di immissione testo.
Il nome è la proprietà che identifica il singolo oggetto, cambiamo questa proprietà e scriviamo “nome”
Class ci indica il tipo di oggetto (da non cambiare assolutamente)
Modificabile, Testo visibile ecc.. si commentano da soli
Testo è il testo di default da visualizzare, provate a variarlo e guardate che cosa succede. Limitatevi a smanettare solo sulle prime 3 linguette (widget, packing e common). Dopo che avete fatto un po' di esperimenti rimettete tutto come prima, variando solo la proprietà Nome (al posto di entry1 scrivete nome).
Fatto questo richiamate (facendoci un click sopra l'oggetto) le proprietà dell'oggetto di immissione campo (da adesso in avanti chiamati gtkentry) relativo al numero di telefono. In questo caso vedrete che la proprietà class è la stessa, poiché sono oggetti uguali, con proprietà diverse ma appartenenti al solito tipo. Per questo oggetto cambiate la proprietà Nome da entry2 a telefono.
Detto in parole molto semplici, potremmo dire che le proprietà sono le caratteristiche dell'oggetto. Per capire meglio il concetto, faro un esempio rapportabile alla realtà di tutti i giorni:
immaginate di voler costruire un automobile. Per prima cosa progettate l'auto e costruite “lo stampo”, ovvero tutte quella catena di montaggio necessaria alla sua costruzione. E' ovvio che quella catena di montaggio è in grado di costruire quel particolare tipo di auto. La catena di montaggio si può paragonare alla classe. La classe è una specie di “stampo”, da cui viene generato l'oggetto. Se guardate la figura sopra, noterete che c'è una proprietà chiamata class (sotto la voce nome), quello ci indica da che classe deriva l'oggetto e quindi il tipo di oggetto. Tornando alla nostra catena di montaggio, con essa costruiremo auto, è vero che ogni auto sarà uguale all'altra, ma ogni una potrebbe avere caratteristiche diverse (colore, colore interni, cilindrata ecc..). Queste caratteristiche sono come le caratteristiche degli oggetti, tutti gli oggetti hanno proprietà che possiamo impostare a nostro piacimento (nel caso di un oggetti GtkEntry nome, lunghezza, testo ecc...). Facciamo un altro passo indietro e guardiamo la nostra automobile; quando l'abbiamo costruita, abbiamo deciso che: premendo sul pedale l'auto parte, spingendo un pulsante si azionano i tergicristalli, premendo un altro pedale l'auto si arresta e così via. Tutte queste azioni sono conseguenti ad un evento. Nel nostro oggetto, gli eventi si chiamano metodi, ma per maggior comprensione, continueremo a chiamarli eventi. Un evento può essere la pressione di un pulsante, la chiusura di una finestra, la scrittura di testo in un oggetto GtkEntry ecc.. Quindi gli eventi ci permettono di definire le azioni da intraprendere al verificarsi di certe azioni. Ora, pensate un secondo alla nostra applicazione, quando si chiudeva la finestra l'applicazione non si chiudeva (la finestra si chiudeva ma il programma python rimaneva “congelato”). Forse adesso sarete riusciti a capirne il motivo: non c'era un legame tra l'evento (chiusura della finestra) e l'azione (chiusura dell'applicazione). Prima di procedere, vi invito a rileggere bene quest' ultimo paragrafo.

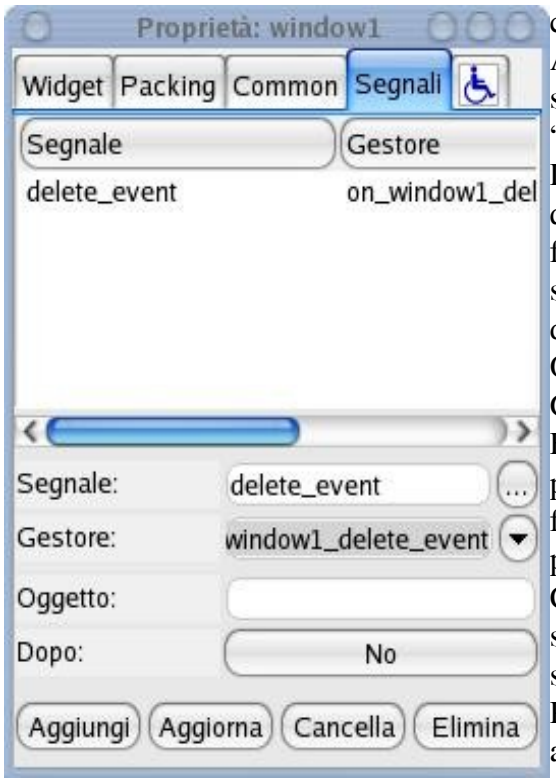
Creare gli eventi

Partiamo subito col dire che in Glade, gli eventi si trovano sotto la linguetta “segnali”. Per prima cosa leghiamo l'evento di chiusura della finestra, con l'azione di chiusura dell'applicazione. Per far ciò, selezioniamo la finestra. Procedete in questo modo:



Dalla finestra principale di Glade, fate doppio click sul nome della finestra (nel riquadro bianco), oppure nella finestra dell'applicazione, premere col tasto destro del mouse, successivamente scorrere la lista del menu contestuale fino alla voce del nome della finestra (nel nostro caso dovrebbe essere window1) poi premere sulla voce “seleziona”.

Adesso andate nella finestra “proprietà” e selezionate la linguetta “segnali”. E' da questa finestra che selezioneremo l'evento che ci interessa.



Adesso premete sul pulsante (...), di fianco alla voce segnale. Si apre una finestra dalla quale sceglierete “delete_event” poi premete sul pulsante “aggiungi”. L'evento sarà aggiunto all'elenco degli eventi per quell'oggetto (vedi figura a fianco). Segnatevi in un foglio ciò che sta scritto sotto gestore (nel caso specifico: on_window1_delete_event): quello è il nome del segnale relativo a quell'oggetto.

Ora salvate il progetto dall'apposito menu “project” di Glade.

Eseguito il salvataggio, richiamate, con il vostro editor preferito, il programma “agenda.py”. Dobbiamo ancora finire di collegare l'evento all'azione di chiusura del programma.

Collegare un evento ad un azione è relativamente semplice. Esso si svolge con due istruzioni: dic e signal_autoconnect(dic).

Passiamo subito a vedere come diventerà la nosta applicazione:

```
import gtk.glade
# Carica Finestra
ApplicazioneGlade = gtk.glade.XML("agenda.glade")
# Collega gli eventi
dic ={
        "on_window1_delete_event":gtk.mainquit
        }
#Connessione delle calllback
ApplicazioneGlade.signal_autoconnect(dic)
gtk.main()
```

Notare l'istruzione ApplicazioneGlade.signal_autoconnect(dic): ApplicazioneGlade è l'oggetto

definito alcune righe sopra con: ApplicazioneGlade = gtk.glade.XML("agenda.glade")

Adesso andate sul terminale, posizionatevi sulla cartella del vostro progetto e scrivete:
python agenda.py
Apparirà la finestra dell'agenda, ma adesso, se chiudete la finestra, verrà chiusa anche l'applicazione ed il terminale sarà rilasciato. Questo perchè alla chiusura della finestra, viene eseguita la funzione gtk.mainquit che termina il programma, come definito dall'istruzione:

```
dic ={
        "on_window1_delete_event":gtk.mainquit
        }
```

Rileggete la parte sopra fin quando non vi è chiaro al 100% il funzionamento. Quello che vedremo in seguito è la logica conseguenza di ciò che avete letto fino ad ora.

Ricerca/Inserisci/Elimina – Il cerchio si stringe

Dopo quest'ultimo capitolo, sarete sicuramente in grado di scrivere le vostre prime applicazioni python/gtk. Nel paragrafo precedente abbiamo visto come legare un evento ad un azione. In seguito vedremo come unire un evento ad una funzione. Per iniziare, richiamiamo la nostra finestra con glade e visualizziamo la finestra delle proprietà.
Ora è necessario selezionare il pulsante di inserimento per impostare l'evento. Questa operazione necessita di un poco di attenzione. I pulsanti sono composti da ben 4 oggetti distinti: l'immagine, l'etichetta, l'allineamento ed il pulsante vero e proprio. Per vedere i singoli oggetti, provate a premere sull'immagine del pulsante, noterete che solo l'immagine del pulsante è selezionata, infatti sulle proprietà potete vedere che sulla voce class c'è scritto GTKimage. Adesso provate a fare un click sulla scritta “inserisci” del pulsante e guardate cosa c'è scritto su class. Quello che noi vogliamo fare è che, alla pressione del pulsante inserisci, il numero telefonico sia inserito nell'agenda. E' chiaro quindi che noi dobbiamo selezionare l'intero pulsante ed assegnare l'evento a tutto il pulsante. Per fare questo si deve semplicemente cliccare in un lato estremo del pulsante, esso risulterà selezionato in tutta la sua interezza e sulle proprietà, alla voce class, sarà visualizzato GTKButton.
Adesso che avete selezionato il pulsante, passiamo alla selezione dell'azione. Premete sulla linguetta segnali della finestra proprietà, successivamente, premete sul pulsante segnale (...), selezionate l'evento “clicked” e poi su aggiungi. Come nel capitolo precedente, ricordatevi il nome del gestore (dovrebbe essere, nel caso del pulsante inserisci: on_Bt_Inserisci_clicked, ma questo ovviamente può variare se avete chiamato il pulsante con un' altro nome). Adesso fate la stessa cosa con gli altri due pulsanti (Ricerca ed Elimina). Fatto ciò, salvate il progetto e richiamate il codice agenda.py, con il vostro editor preferito.
Attualmente, nel nostro codice, abbiamo un solo evento e quindi questo codice:

```
# Collega gli eventi
dic ={
        "on_window1_delete_event":gtk.mainquit
        }
```

A questo codice, dobbiamo aggiungere anche gli altri 3 eventi dei pulsanti. Per far questo, è sufficiente inserire la virgola di separazione, il nome dell'evento e la funzione che vogliamo associare all'evento. Quindi possiamo scrivere:

```
# Collega gli eventi
dic ={
        "on_window1_delete_event":gtk.mainquit,
        "on_Bt_Inserisci_clicked":Inserisci,
        "on_Bt_Ricerca_clicked":Ricerca,
        "on_Bt_Elimina_clicked":Elimina
        }
```

Scriveremo ora le funzioni Inserisci, Ricerca ed Elimina.
Iniziamo con la funzione di Inserimento. Essa non sarà molto differente dalla funzione di inserimento che avevamo creato all'inizio di questo tutorial. Possiamo scriverla in questo modo:

```
def Inserisci(obj):
    DizionarioNumeri[ApplicazioneGlade.get_widget('entry1').get_text()]
                    =ApplicazioneGlade.get_widget('entry2').get_text()
```

Attenzione la riga dopo la definizione della funzione continua con la riga immediatamente sotto.
Spieghiamo un attimo l'istruzione apparentemente complessa.

DizionarioNumeri è il nostro dizionario definito con: DizionarioNumeri = {}

ApplicazioneGlade è la nostra finestra, definita con la riga:
ApplicazioneGlade = gtk.glade.XML("agenda.glade")

Dove ApplicazioneGlade diventa un oggetto generato dal file “agenda.glade” (il file salvato con glade)

ApplicazioneGlade.get_widget('entry1').get_text()
L'istruzione sopra significa:
prendi il testo contenuto nel controllo entry1 dell'oggetto ApplicationGlade, in poche parole prende la finestra (ApplicazioneGlade), all'interno della quale prende l'oggetto entry1 (get_widget('entry1')). Di quest'ultimo oggetto, prendi la proprietà get_text (cioè il suo contenuto).
Sembra complesso, ma se ne capite il funzionamento, vi risulterà naturale scrivere il codice.

Ora, qualcuno si dovrebbe essere chiesto: “ma come facevi a sapere che entry1 ha una proprietà chiamata get_text? Semplice, mi sono letto la documentazione. Prima di procedere, facciamo un secondo di pausa e riflettiamo su come dev'essere letta la documentazione.
Se andate sul sito delle pygtk, nella sezione della documentazione, troviamo una pagina in cui c'è scritto: PyGTK Class Hierarchy, cioè, gerarchia della classe pygtk.
http://www.pygtk.org/pygtk2reference/class-hierarchy.html
Essa mostra tutti i componenti e le loro derivazione delle classi GTK. Quindi abbiamo tutte le proprietà ed i metodi delle GTK (le proprietà ed i metodi dei pulsanti, delle griglie, degli entry ecc...). Ora, quello che noi abbiamo usato per l'immissione del nome e del numero telefonico è un oggetto gtk entry (guardatelo nella proprietà class su glade). Bene, adesso cercate, nella pagina della documentazione, la parola gtk.entry (solo gtk.entry !!!). Premeteci sopra per vedere la documentazione relativa a quell'oggetto.
Ora dovreste avere una pagina che vi elenca le proprietà, tra queste c'è anche get_text, se ne andate

a vedere il documento relativo, scoprirete che: “The `get_text`() method returns the value of the "text" property which is a string containing the contents of the entry”, praticamente questa proprietà contiene il valore inserito nell'oggetto entry. Tutto questo può sembrare complesso, ma una volta capito, è molto semplice da applicare. Non scordiamoci che python è un linguaggio ad oggetti e quello che avete visto fin'ora (proprietà e metodi), stanno alla base di questo tipo di programmazione. Ora vi invio il codice completo che inserisce e ricerca numeri telefonici del dizionario. Il programma funziona, ma necessita di molti aggiustamenti e migliorie che voi provvederete a fare. Le cose che andranno fatte sono:

1. Implementare un sistema di salvataggio dei dati in un file (come l'agenda da linea di comando del nostro primo esempio). Vi suggerisco di aggiungere un pulsante per il salvataggio, poi aggiungere un evento e collegarlo alla funzione di salvataggio
2. Implementare un sistema di caricamento dei dati dal file in memoria (vedi suggerimento sopra)
3. Ad ogni inserimento cancellare il contenuto dei due entry usati

Prima di iniziare a smanettare, cercate di capire ogni singola riga del programma ed andate a vedere le proprietà degli oggetti utilizzati, nella pagina di documentazione di pygtk. L'esercizio, per dei principianti, non è facilissimo. Non scoraggiatevi perchè capire tutto questo, significa aver capito come si può fare un'applicazione con python e le gtk.
Di seguito il codice python della nostra agenda:

```
import gtk.glade
# Carica Finestra
ApplicazioneGlade = gtk.glade.XML("agenda.glade")
DizionarioNumeri = {}
def Ricerca(obj):
        if DizionarioNumeri.has_key(ApplicazioneGlade.get_widget('entry1').get_text()):
                ApplicazioneGlade.get_widget('entry2').set_text
(DizionarioNumeri[ApplicazioneGlade.get_widget('entry1').get_text()])
def Inserisci(obj):
        DizionarioNumeri[ApplicazioneGlade.get_widget('entry1').get_text()]                    =
ApplicazioneGlade.get_widget('entry2').get_text()
def Elimina(obj):
        if DizionarioNumeri.has_key(ApplicazioneGlade.get_widget('entry1').get_text()):
                del DizionarioNumeri[ApplicazioneGlade.get_widget('entry1').get_text()]
                ApplicazioneGlade.get_widget('entry2').set_text('')
                ApplicazioneGlade.get_widget('entry1').set_text('')

# Collega gli eventi
dic ={
          "on_window1_delete_event":gtk.mainquit,
          "on_Bt_Inserisci_clicked":Inserisci,
          "on_Bt_Ricerca_clicked":Ricerca,
          "on_Bt_Elimina_clicked":Elimina
          }
#Connessione delle calllback
ApplicazioneGlade.signal_autoconnect(dic)
gtk.main()
```

Appunti di Python

Buon Divertimento :-)