



ESTENDERE PYTHON CON FREEBASIC

Manipolare gli oggetti Python senza le Python Api

*Il Freebasic come alternativa
al C e al C++ per estendere
Python.*

Marco Salvati

Indice	Pag.
Cos'è il Freebasic.	3
Tipi di dato: Python, Ctypes, Freebasic a confronto.	4
Passaggio di parametri per valore e riferimento.	4
Convezione di chiamata delle procedure.	5
Un occhiata ad alcuni costrutti base, analogie.	5
Operatori differenti.	6
Operatori su bit.	6
Il modulo ctypes, nozioni di base.	6
Gli oggetti ctypes.	7
Prototipi di funzione.	8
Introduzione di base al Freebasic.	9
Piccoli esempi per iniziare.	10
• Passaggio parametri per valore: Esempio 1	10
• Passaggio parametri per riferimento: Esempio 2	11
• Le stringhe in Freebasic e come gestirle.	11
• Gestione delle substringhe in Freebasic, confronto con Python.	12
• Puntatori, confronto tra Freebasic e ctypes.	13
• Operatori Freebasic per puntatori	13
• Modalità di passaggio e ricevimento di passaggio delle stringhe.	15
• L'angolo delle chicche , Stringhe non troppo immutabili	16
Qualcosa di più complicato	18
• Array e matrici in Freebasic come crearle e accedervi da Python .	18
• Le matrici bidimensionali in Freebasic.	20
• fbDim per gestire array fino a 3 dimensioni in Python e Freebasic	23
• L'angolo delle chicche , Dim una classe Python per gli array e matrici con indici variabili	26
• Funzioni con puntatori anonimi.	28
• Strutture e Unioni	30
• Come chiamare Python da Freebasic.	31
• L'angolo delle chicche , Select Case per Python.	33
Le classi in Freebasic	35
• Esempio di classe in Freebasic e Python a confronto	36
• Metodi per inizializzare una classe Freebasic da Python	36
• Confrontiamo i due linguaggi nella creazione di classi	39
Gli oggetti Python in Freebasic, senza l'uso delle Python api.	40
• Cosa sono le Python api	42
• Progettiamo delle api in Python e ctypes per il Freebasic, e altri linguaggi.	43
• I files PyStruct.py e PyObj.bas, I descrittori di classe List, Tuple, Dict e Set	44
• Le classi wrapper, let assegnamento di un valore a un oggetto	49
• La classe isType cosa c'è in un oggetto	50
• Vediamo l'uso di isType in Freebasic	50
• ptr_convert la funzione per la conversione dei puntatori	51
• Le classi wrapper: getitem	52
• Getitem esempio	53
• La classe setitem	54
• Setitem esempio	55
	55
	56

• La classe delitem ed esempio di uso	56
• La classe copy_clear ed esempio di uso	57
• La classe contains ed esempio di uso	58
• La classe slice ed esempio di uso	58
• La classe newObj ed esempio di uso	59
• La classe toString ed esempio di uso , magic method __str__ e __repr__	59
• Le classe Del e Len, ed esempio di uso	60
• La classe count_index	61
• La classe buffer ed esempio di uso	62
• Le classi in list_method.py ed esempio di uso	63
• Il file dict_method.py la classe contiene che implementa i metodi dei dizionari	64
• Esempio di uso della classe dict_method	66
• Il file set_method.py contiene la classe che implementa i metodi dei set	66
• Le classi contenitori	69
• Il file PyTuple Ambiente per la creazione e persistenza delle tuple	72
• Il file pyDict Ambiente per la creazione e persistenza dei dizionari	73
• file pySet Ambiente per la creazione e persistenza dei set	74
• Il file pyList Ambiente per la creazione e persistenza delle liste	74
• Esempio completo	76
Mini framework per console	82
• Passo 01: Utilizziamo Freebasic per creare le primitive per la gestione video da console. Files fbConsole.bi e fbConsole.bas	82
• Testa alcune primitive	86
• Passo 02: Creazione di un wrapper per l'utilizzo delle primitive, file fbConsole.py	88
• Passo 03: Creazione della classe WinBase, base per la gestione delle finestre	99
• Passo 04: Creazione classe Screen derivata Winbase	100
• Passo 05: Creazione della classe Window derivata da WinBase	102
• Passo 06: Creazione della classe Write scrittura in una finestra	104
• Passo 07: Creazione della classe Draw disegno di box nella finestra	107
• Passo 08: Creazione della classe Shadow ombreggiatura all'interno della finestra	109
• Passo 09: Creazione della classe Widgets	111
• Passo 10: Il widgets Label	112
• Passo 11: Il widgets Buttons	117
• Passo 12: Il widgets Entry	119
• Passo 13: La classe Medit gestore di multi edit	123
• Passo 14: La classe Event gestore di eventi mouse e tastiera	124
• Passo 15: La classe Vmenu, crea dei menu verticali	128
• Passo 16: La classe Hmenu, crea dei menu orizzontali	131
Appendice	136
• fbDim, esempio completo	135
• Python/fbAPI: Dizionari, iteratori Python in Freebasic	137
• Metodi fbConsole non menzionati	140
• Considerazioni finali	141

Cos'è il Freebasic.

Freebasic è un dialetto del linguaggio basic, nato per essere completamente compatibile con lo storico Quick Basic della microsoft, si è poi evoluto in un linguaggio a se ma retrocompatibile. Freebasic ora è un linguaggio ad oggetti multiplatforma, (unica pecca riguarda l'eredità multipla non ancora implementata), con gestione dei puntatori stile C. Può utilizzare la libreria C standart in modo nativo, per i più avventurosi la possibilità di codice assembler inline.

Un preprocessore simile a quello del C che permette di avere una compilazione adattabile al sistema operativo ospite.

In questo tutorial si richiede una discreta conoscenza di Python, per quanto riguarda il Freebasic, non intendo farvi un corso, in rete troverete un'ottima documentazione gratuita, anche in italiano. Quindi procederò per esempi gradualmente, approfondendo man mano, alcuni argomenti non reperibili in rete. La versione Freebasic di riferimento che utilizzerò è la 1.08.1 a 64/32 bit ovviamente utilizzate la versione corrispondente al vostro sistema operativo.

Inizieremo con le analogie e le differenze tra Freebasic e con Python, l'uso di ctypes per colmare tali le differenze.

Copyright © Marco Salvati (2021)

Questo tutorial, eccetto dove diversamente specificato, è rilasciato nei termini della licenza Creative Commons Attribuzione 3.0 Italia (CC BY 3.0) il cui testo integrale è disponibile al sito

<http://creativecommons.org/licenses/by/3.0/deed.it>

Per maggiori informazioni su questo particolare regime di diritto d'autore si legga il materiale informativo pubblicato su www.copyleft-italia.it.

I programmi da me creati, utilizzati in questo studio, sono inclusi al tutorial in formato zip.

Rilascio i suddetti programmi sotto licenza GPL 3. Che Potranno essere scaricati dal mio repository su GitHub <https://github.com/marco-61>

Tipi di dato: Python, Ctypes, Freebasic a confronto

Tipi standart :

ctypes type	Freebasic	Python 3.x type
<u>c_bool</u>	Integer (0 = False 1 = True) Boolean dalla versione 1.08.1	bool
<u>c_char</u>	String*1 stringa a lunghezza fissa	1-bytes
<u>c_wchar</u>	1 carattere Wstring	1- str
<u>c_byte</u>	byte	int
<u>c_ubyte</u>	ubyte	int
<u>c_short</u>	short	int
<u>c_ushort</u>	ushort	int
<u>c_int</u>	Integer (c_long versione a 32 bit, c_lonlong versione a 64 bit)	int
<u>c_uint</u>	UInteger (c_ulong versione a 32 bit, c_ulonlong versione a 64 bit)	int
<u>c_long</u>	long	int
<u>c_ulong</u>	ulong	int
<u>c_longlong</u>	Longint (Integer versione a 64bit)	int
<u>c_ulonglong</u>	ulongint	int
<u>c_float</u>	float	float
<u>c_double</u>	double	float
<u>c_longdouble</u>	-	float
<u>c_char_p</u>	Zstring Ptr (NUL terminated)	bytes string or None
<u>c_wchar_p</u>	Wstring Ptr (NUL terminated)	str or None
<u>c_void_p</u>	Any Ptr	int or None
<u>FBSTRING</u> <u>**</u>	String	Bytes or None

I tipi Integer e UInteger si adattano alla versione del compilatore 32 o 64 bit, quindi in Freebasic conviene utilizzare Long e Ulong per un intero a 32 bit e Longint e ULongInt per quello a 64 bit.

** FBSTRING non è un tipo standart di ctypes ma una struttura creata appositamente.
String è il tipo di stringa storico del basic consiste in un descrittore di tipo, che vedremo più avanti.

Passaggio di parametri per valore e riferimento.

Passare un dato per valore ci permette di non compromettere il contenuto della variabile originale, questo metodo da preferire, a meno che sia necessario modificare i dati in questo caso siamo obbligati a passare le variabili che vogliamo modificare per riferimento.

Convezione di chiamata delle procedure e funzioni.

Le convenzioni per chiamare delle procedure o funzioni sono di tre tipo, sia Freebasic che le ctype di python possono utilizzarle indistintamente.

Stdcall è la convenzione di default usata nelle API su windows.

Pascal è la convenzione di default del linguaggio Pascal e del Quick basic.

Cdecl è la convenzione di default del linguaggio C, utilizzata su Linux e altri sistemi Unix.

Freebasic è stato compilato con FBCALL che assume una delle precedenti identità a seconda del sistema operativo o del tipo di processore 32/64. Windows 32 bit e Xbox stdcall, Windows 64 bit, Linux e altri Unix like (inoltre il vecchio msdos) cdecl.

La convenzione cdecl è quella che utilizzeremo, per creare i wrapper alle funzioni da esportare nelle librerie dinamiche (.dll per windows - .so per Linux e Unix like).

Un'occhiata ad alcuni costrutti base, analogie.

Python	Freebasic
if condizione : [statement block] [elif condizione:] [statement block] [else:] [statement block]	If condizione then [statement block] [Else If condizione] [statement block] [Else] [statement block] End If
while condizione: [statement block]	While condizione <i>oppure</i> Do While condizione [statement block] [statement block] Wend Loop
while True <i>ciclo infinito</i> [statement block]	Do <i>ciclo infinito</i> [statement block] Loop
while not condizione <i>ciclo until</i> [statement block]	Do Until condizione [statement block] Loop
while True <i>ciclo while testato</i> [statement block] <i>alla fine</i> If condizione : break	Do [statement block] Loop while condizione
while True <i>ciclo until testato</i> [statement block] <i>alla fine</i> If not condizione : break	Do [statement block] Loop Until condizione
for x in range (start,end+1 [,step]): [statement block]	For x =start to end [Step p] [statement block] Next [x]

Autore: Marco Salvati

Licenza Tutorial: CC BY 3.0

Email: marcosalvati61@gmail.com

Licenza Software: GPL V.3

Operatori differenti.

Descrizione	Python	Freebasic
Divisione intera	//	\
Elevamento a potenza	**	^
Scorrimento a sinistra	<<	shl
Scorrimento a destra	>>	shr
Modulo	%	mod
Divisione intera e assegna	//=	\=
Eleva a potenza e assegna	**=	^=
Scorri a sinistra e assegna	<<=	Shl=
Scorri a destra e assegna	>>=	Shr=
Modulo e assegna	%=	mod=
Diverso	!=	<>
Assegnamento	=	= oppure =>
Confronto	==	=

Operatori su bit.

Descrizione	Python	Freebasic
Bitwise or		Or
Bitwise and	&	And
Bitwise xor	^	Xor
Bitwise or con assegnamento	=	Or=
Bitwise and con assegnamento	&=	And=
Bitwise xor con assegnamento	^=	Xor=

Il modulo ctypes, nozioni di base.

Ctypes è una libreria di funzioni esterne a Python. Fornisce i tipi i dati compatibili con il C e consente di chiamare funzioni in DLL o librerie condivise.

Caricamento di librerie a collegamento dinamico.

Ctypes esporta *cdll*, e per gli oggetti Windows *windll* e *oledll* per il collegamento dinamico.

Caricate le librerie si può accedere ad esse come attributi di questi oggetti.

Cdll carica le librerie che utilizzano la convezione *cdecl*, mentre *windll* carica le librerie che utilizzano la convenzione *stdcall*, anche *oledll* utilizza le *stdcall* come condizione di chiamata, ma presuppone che le funzioni restituiscono HRESULT il codice di errore di Windows .

Anche se su Windows per convezione freebasic utilizza le *stdcall* lo forzeremo a utilizzare le *cdecl* per le funzioni da esportare, questo permetterà un riutilizzo della libreria su altri sistemi operativi.

Gli oggetti ctypes.

Ctypes implementa i dati C con degli oggetti che li inglobano. `A=c_int(6)` crea un oggetto con un dato di tipo integer di valore 6.

Per accedere fisicamente al dato utilizzare la proprietà **value** in lettura e in scrittura.

`C = A.value` C ora contiene il valore in A: `A.value+=1` incrementa A che ora vale 7, *tutti i tipi dato hanno la proprietà value.*

I tipi puntatore, un puntatore è un oggetto che non contiene effettivamente il dato, ma la locazione di memoria di un dato di un certo tipo o ad un'area dati di un certo, tipo o anche di una funzione.

`P=pointer(a)` crea un puntatore ad un oggetto ctypes non al suo valore, P non ha la proprietà value ma possiede la proprietà **contents** che ritorna l'oggetto contenuto.

`X=P.contents` ritorna l'oggetto `c_int(7)` (su molte piattaforme `c_int` e `c_size_t`, sono un alias di `c_long`)

Per accedere al valore di a potete utilizzare o la forma lunga `p.contents.value` (scomodo) o quella breve `p[0]` come se fosse un array o una lista.

Attenzione per i tipi semplici utilizzate solo l'indice 0 altri valori solo se P punta ad un array o un altro tipo di buffer dati.

Un indice fuori dall'area in considerazione, comporta in lettura un valore errato, in scrittura il crash di Python.

Puntatori di puntatori, un puntatore, può anche puntare ad altri puntatori, `p2=pointer(p)` avremo che `p2[0]` punta al puntatore e `p2[0][0]` al valore.

Questa doppia indicizzazione è utile per gestire matrici, array di puntatori a funzioni etc.

Array, si tratta di un area contigua dello stesso elemento, `Array=(c_int *10)` (1,2,3,4,5,6,7,8,9,10) crea un array di 10 interi e lo inizializza. `Array10Int=(c_int*10)` crea un prototype di un array di 10 interi `V=Array10Int` (1,2,3,4,5,6,7,8,9,10) crea un array V di 10 interi.

Se utilizzate `c_char_p` per creare una stringa di bytes esempio: `c=c_char_p(b'ABCD')`

Avrete un stringa di bytes contenente A, B, C, D ma è a solo lettura. Se volete creare una stringa o un array di caratteri che sia anche modificabile, dovete utilizzare `create_string_buffer()`. E' possibile accedere (o modificare) al contenuto del blocco di memoria con la proprietà **raw**; se volete accedervi come stringa con la terminazione NULL usate proprietà **value**.


```

>>> from ctypes import *
>>> p=create_string_buffer(3) #crea un buffer di 3 bytes Null
>>> print(sizeof(p),repr(p.raw))
3 b'\x00\x00\x00'
>>> p=p=create_string_buffer(b'Hello') # crea un buffer contenente un Null alla fine di Hello
>>> print(sizeof(p),repr(p.raw))
6 b'Hello\x0'
>>> p=p=create_string_buffer(b'Hello',10) # crea un buffer di 10 bytes
>>> print(sizeof(p),repr(p.raw))
10 b'Hello\x00\x00\x00\x00\x00'
>>> p.value=b'Hi'
>>> print(sizeof(p),repr(p.raw))
10 b'Hi\x00\x00\x00\x00\x00'

```

Per creare un blocco di memoria mutabile contenente caratteri unicode (tipo `c_wchar`) utilizzate la `create_unicode_buffer()`.

Prototipi di funzione.

E' possibile specificare i tipi di argomenti richiesti delle funzioni esportate dalle DLL impostando l'argtypes attributo, tramite l'attributo restype il valore di ritorno (che per default è `c_int`).

Esempio: dobbiamo chiamare una ipotetica funzione di nome `sum_array()` che deve ricevere un puntatore ad un array di float e un intero che indica la lunghezza, ritornare la somma degli elementi.

Supponendo che la dll sia nel path corrente:

```

lib=CDLL('mydll.ext') # .ext estensione .dll Windows, .so Linux, .dylib MacOS
lib.sum_array.argtypes=[POINTER(c_float),c_int] #puntatore ad un array di float stile C e la sua lunghezza
lib.sum_array.restype = c_float # valore di ritorno
array=(c_float *5)(3.9,7.6,5.4,4.35,23.87) #array di 5 float
length=c_int(5) # lunghezza dell'array
result=lib.sum_array(array,length)

```

Introduzione di base al Freebasic

Freebasic distingue le funzioni che ritornano un valore da quelle che non ritornano nulla con due diversi costrutti, la sintassi di questi costrutti è molto elaborata con diverse opzioni, quindi andrò a mostrarvi solo quelle necessarie per l'utilizzo che ci riguarda:

```
Declare Function nome cdecl alias "nome esterno"([lista parametri])As type
Function nome cdecl alias "nome esterno"([lista parametri])As type Export
statements
...
Return return_value
End Function

Declare Sub nome cdecl alias "nome esterno"([lista parametri])
Sub nome cdecl alias "nome esterno"([lista parametri]) Export
statements
...
End Sub
```

Declare dichiara il prototipo di una function o di una sub, le dichiarazioni possono trovarsi in un file distinto da includere, estensione utilizzata .bi

Cdecl il passaggio di parametri stile C (obbligatorio per un corretto funzionamento)

Alias nome esterno che verrà memorizzato nella libreria identifica la funzione (obbligatorio)

Export specifica che la funzione deve essere visibile da altri programmi (obbligatorio).

Dim dichiara una Variabile:

```
Dim [Shared] symbolname As DataType [, ...]
Dim [Shared] As DataType symbolname [, ...]
```

Dichiara un' Arrays:

```
Dim [Shared] symbolname ( [lbound To] ubound [, ...] ) [As DataType] [,...]
Dim [Shared] As DataType symbolname ( [lbound To] ubound [, ...] ) [,...]
```

Dichiara e inizializza i valori:

```
Dim scalar_symbol As DataType = expression | Any (dichiarazione di una costante)
Dim array_symbol ([lbound To] ubound) [As DataType] => { expression [, ...] } |
Any
```

```
Dim udt_symbol As DataType = ( expression [, ...] ) | Any
shared indica che la variabile è globale
```

Gli indici degli array possono essere dichiarati nel range che si desidera.

Esempio: *Dim As String mesi(1 To 12)*

As indica il tipo di dati., *udt_symbol* nome di una struttura utente, *scalar_symbol* dichiara una costante.

Any tipo speciale puntatore permette di non inizializzare un puntatore o un array.

```
Dim As Integer A(0 To 9) = Any '' this variable is not initialized
```

Autore: **Marco Salvati**

Licenza Tutorial: **CC BY 3.0**

Email: marcosalvati61@gmail.com

Licenza Software: **GPL V.3**

Piccoli esempi per iniziare.

Vediamo in primis il passaggio di tipi semplici per riferimento e per valore.

Esempio 1:

Iniziamo con una semplice funzione che sommi due interi passati per valore e ritorna la somma.

```
' Dichiarazione della funzione – questo è un commento  
Rem anche questo è un commento, vecchio stile  
Declare Function addInt Cdecl Alias "addInt" (Byval a as Integer, Byval b as Integer) as Integer  
  
Function addInt Cdecl Alias "addInt" (Byval a as Integer, Byval b as Integer) as Integer export  
Return a + b  
End Function
```

Ora andiamo a creare una libreria dinamica. Negli esempi utilizzerò per default l'estensione .dll tipica di Windows, per altri sistemi cambiatela con quella relativa.

Per la compilazione apriamo il terminale, nella directory dove si trova il sorgente.

```
C:\pathname> pathFreebasic\fbcc -dll addInt.bas
```

oppure:

```
C:\pathname> pathFreebasic\fbcc -dylib addInt.bas
```

Se non ci sono errori verranno create due librerie una statica libaddInt.a, e una dinamica addInt.dll, l'estensione dipenderà dal sistema operativo.

Ora apriamo Python da console o da ide.

```
>>> from ctypes import *  
>>> lib=CDLL(r'Libpathname\addInt.dll') oppure su Linux  
lib=CDLL('Libpathname/addInt.so')
```

I parametri sono tutti interi e per valore. ctypes accetta direttamente interi e None come puntatore nullo. Quindi possiamo non utilizzare gli attributi argtypes e restype.

```
>>> lib.addInt(5,6)
```

```
>>> 11
```

Ma è sempre consigliabile impostare gli attributi di tipo anche in casi semplici come questo e utilizzare i tipi forniti da ctypes.

```
>>> lib.addInt.argtypes=[c_int,c_int]  
>>> lib.addInt.restype=c_int  
>>> lib.addInt(c_int(5),c_int(6))  
>>> 11
```

Questo permetterà un maggiore controllo di quello che la funzione riceverà.

È più comodo utilizzare una variabile come path per le VS librerie per poterlo cambiare in conformità alle VS necessità o sistema operativo

Ora vediamo un esempio di passaggio per riferimento:

Esempio 2:

Scrivere una subroutine di nome swapFloat, che prende due parametri di tipo float passati per riferimento e ne scambia i valori.

```
Declare Sub swapFloat Cdecl Alias "swapFloat (ByRef a as Single, ByRef b as Single)  
  
Sub swapFloat Cdecl Alias "swapFloat (ByRef a as Single, ByRef b as Single) export  
    Dim tmp as Single  
    tmp=a: a=b: b=tmp  
    ' Freebasic implementa la funzione Swap, che fa la stessa cosa Swap(a,b)  
End Sub
```

Dopo aver creato la dll apriamo python:

```
>>> from ctypes import *  
>>> LIBPATH=r'.\lib\ swapFloat.dll' #la libreria è nella cartella lib del path corrente  
>>> lib=CDLL(LIBPATH)  
>>> a=c_float(10.5);b=c_float(7.6) # creazione delle variabili  
>>> lib.swapFloat.argtypes=[POINTER(c_float), POINTER(c_float)]  
>>> lib.swapFloat(byref(a),byref(b))  
>>> print(a.value,b.value)  
>>> 7.6 10.5
```

Le stringhe in Freebasic e come gestirle.

In Freebasic esistono 3 tipi di stringhe:

Le Zstring equivalenti a c_char_p di ctypes, sono stringhe di bytes.

Le Wstring equivalenti a c_wchar_p di ctypes, sono stringhe di unicode.

Le String sono stringhe di bytes, come le Zstring, la differenza è che il puntatore non indica direttamente l'area di memoria ma una struttura descrittiva della stringa.

```
class FBSTRING(Structure):  
    _fields_=[('data',c_char_p),          # puntatore alla stringa  
              ('len', c_ssize_t),        # lunghezza della stringa  
              ('size', c_ssize_t)]       # blocchi occupati
```

Il tipo String in freebasic può avere una lunghezza da 0 a 2 gigabytes.

Gestione delle substringhe in Freebasic, confronto con Python.

Freebasic a 3 funzioni per gestire le substringhe.

Descrizione	Freebasic	Python
Ritorna la parte sinistra di una stringa s	$R = \text{Left}(s, Ncar)$	$R = s[:Ncar]$
Ritorna la parte destra di una stringa s	$R = \text{Right}(s, Ncar)$	$R = s[-Ncar:]$
Ritorna la parte centrale di una stringa s n caratteri a partire da start	$R = \text{Mid}(s, start, Ncar)$	$R = s[start:-Ncar]$
Modifica una stringa	$\text{Mid}(s, start, Ncar) = \text{news}$ Freebasic sovrappone la stringa	$R = s.\text{replace}(\text{old}, \text{newS})$ Python ritorna una nuova stringa non la sovrappone fisicamente

Puntatori confronto tra Freebasic e ctypes.

Dichiarazione di un puntatore generico.

Dim nome As DataType Ptr oppure **Dim nome As DataType Pointer**

Esempi di puntatori

Questi esempi Freebasic sono tratti dalla documentazione ufficiale, per scopo dimostrativo.

```
' Crea un puntatore ZString e assegna direttamente il valore
Dim As ZString * 14 str1 => "hello, world"
Print str1
Print Len(str1)      'ritorna 12, la grandezza del contenuto della stringa
Print SizeOf(str1)   ' ritorna 14, la grandezza della variabile
' Crea un puntatore ZString
Dim As ZString Ptr str2
str2 = Allocate( 14 ) ' alloca lo spazio dedicato
*str2 = "hello, world" ' assegna un contenuto
Print *str2
Print Len(*str2)      'ritorna 12, la grandezza della stringa contenuta
Print SizeOf(*str2)   'ritorna len(zstring), la grandezza della variabili
' Crea un puntatore a un Integer.
Dim p As Integer Ptr
' Crea un valore intero che verra utilizzato dal puntatore "p"
Dim num As Integer = 98845
p=@num ' assegna a p l'indirizzo di num
```

Operatori Freebasic per puntatori

VarPtr(nome)	Ritorna l'indirizzo di una variabile o di una struttura per il tipo String e per gli array ritorna il puntatore al descrittore.
@nome	Uguale a VarPtr ma si applica anche alle funzioni e alle sub
StrPtr(stringa) o SAdd(stringa)	Ritorna il puntatore effettivo di una stringa
CPtr(PtrDataType,p2)	Effettua il Cast di un puntatore da un tipo ad un altro

VarPtr esempio

```
Dim a As Integer, addr As Integer
a = 10

' place the address of a in addr
addr = VarPtr(a)

' change all 4 bytes (size of INTEGER) of a
Poke Integer, addr, -1000
Print a

' place the address of a in addr (same as above)
addr = @a

' print the least or most significant byte, depending on the CPU endianness
Print Peek( addr )
```

StrPtr esempio

```
' This example uses StrPtr to demonstrate using pointers with strings

Dim myString As String
Dim toMyStringDesc As Any Ptr
Dim toMyString As ZString Ptr
' Note that using standard VARPTR notation will return a pointer to the
' descriptor, not the string data itself

myString = "Improper method for Strings"
toMyStringDesc = @myString
Print myString
Print Hex( toMyStringDesc )
Print

' However, using StrPtr returns the proper pointer
myString = "Hello World Examples Are Silly"
toMyString = StrPtr(myString)
Print myString
Print *toMyString
Print

' And the pointer acts like pointers to other types
myString = "MyString has now changed"
Print myString
Print *toMyString
Print
```

@ esempio 1

```
Dim a As Integer
Dim b As Integer

Dim addr As Integer Ptr

a = 5      'Here we place the values 5 and 10 into a and b, respectively.
b = 10

'Here, we print the value of the variables, then where in memory they are stored
Print "The value in A is ";a;" but the pointer to a is ";@a
Print "The value in B is ";b;" but the pointer to b is ";@b

'Now, we will take the integer ptr above, and use @ to place a value into it.
'Note that the * will check the value in the ptr, just as @ checked the ptr
'for a normal variable.

addr = @a

Print "The pointer addr is now pointing at the memory address to a, value: ";*addr

addr = @b

Print "The pointer addr is now pointing at the memory address to b, value: ";*addr
```

@ esempio 2

```
'This program demonstrates how the @ symbol can be used
'to create pointers to subroutines.

Declare Sub mySubroutine()

Dim say Hello As Sub()

say_Hello = @mySubroutine      'We tell say_Hello to point to mySubroutine.
                               'The sub() datatype acts as a pointer here.

say_Hello() 'Now we can run say_Hello just like mySubroutine.

Sub mySubroutine
    Print "hi"
End Sub
```

CPtr Esempio

```
Dim intval As Integer
Dim intptr As Integer Ptr
intval = &h0080
intptr = @intval
'will print -
'128 and 128, as the first expression will be "seen" as an signed byte
Print *CPtr(Byte Ptr, intptr), *intptr
```

Confronto, Freebasic con Ctypes

Freebasic	Python
<i>Dim Pfoo as Any Ptr</i>	<i>Pfoo=c_void_p()</i>
<i>Dim Pi As Integer Ptr</i>	<i>Pi=POINTER(c_int)</i>
<i>*Pa</i>	<i>Pa[0]</i>
<i>Dim Pb As Byte Ptr</i> <i>Pb=CPrt(Byte Ptr, Pi)</i>	<i>Pb=cast(Pi,POINTER(c_byte))</i> <i>I parametri sono al contrario rispetto CPrt()</i>

I successivi 3 esempi hanno solo lo scopo di mostrare come inviare e ricevere una stringa di qualsiasi tipo.

Esempio 3:

Passaggio di una Zstring contenente 'Hello Word,' stampata da Freebasic, e ritorna un'altra con 'Ciao Mondo' stampata da poi da Python

```
Declare Function TestZstring cdecl alias "TestZstring" (Byval pzstr As Zstring Ptr) As Zstring  
Function TestZstring (Byval pzstr As Zstring Ptr) As Zstring Ptr export  
    dim s as zstring ptr  
    s=Allocate( 11 )  
    *s="Ciao Mondo"  
    print *pzstr  
    return s  
End Function
```

Effettuata la compilazione eseguire in console no tramite ide il Print di Freebasic scrive di default sullo schermo fisico non sullo standart output

```
>>> lib = CDLL('TestZstring.dll')  
>>> lib.TestZstring.argtypes =[c_char_p]  
>>> lib.TestZstring.restype =c_char_p  
>>> ptext =c_char(b'Hello Word')  
>>> ptext2=lib.TestZstring.(ptext)  
>>> print (ptext2.value)
```

Esempio 4

Passaggio di una Wstring contenente 'Hello Word,' stampata da freebasic, e ritorna un'altra con 'Ciao Mondo'

```
Declare Function TestWstring (Byval pWstr As Wstring Ptr) As Wstring Ptr  
Function TestWstring (Byval pzstr As Wstring Ptr) As Wstring Ptr export  
    dim s as Wstring Ptr  
    s=Allocate( 22 )  
    *s="Ciao Mondo"  
    print *pzstr  
    return s  
End Function
```



```

>>> LIBPATH=r'.\lib\ TestWstring.dll'
>>> lib = CDLL(LIBPATH)
>>> lib.TestWstring.argtypes =[c_char_p]
>>> lib.TestWstring.restype =c_char_p
>>> ptext =c_wchar('Hello Word')
>>> ptext2=lib.TestWstring.(ptext)
>>> print (ptext2.value)

```

Esempio 5

Passaggio di una stringa contenente 'Hello Word' stampata da Freebasic, e ritorna un'altra con 'Ciao Mondo'

```

Declare Function TestString (s As String) As String
Function TestString (s As String) As String export
    dim s2 as String
    s2="Ciao Mondo"
    print s
    return s2
End Function

```

Definire la struttura FBSTRING e inserirla in un file chiamato fbTypes.py per poterla importare più facilmente.

```

class FBSTRING(Structure):
    _fields_= [('data',c_char_p),          # puntatore alla stringa
               ('len', c_ssize_t),        # lunghezza della stringa
               ('size', c_ssize_t)]       # blocchi occupati

```

Lato Python:

```

>>> from ctypes import *
>>> from fbType import *
>>> LIBPATH=r'.\lib\TestString.dll'
>>> lib = CDLL(LIBPATH)
>>> lib.TestString.argtypes =[POINTER(FBSTRING)]
>>> lib.TestString.restype =[POINTER(FBSTRING)]
>>> t= b'Hello Word'
>>> text=FBSTRING(t, len(t), 1)
>>> text2=lib.TestString(text)
>>> print (text2.contents.data) # oppure print(text2[0].data)

```

Output:

```

Hello Word
b'Ciao Mondo'

```

L'angolo delle chicche

Stringhe non troppo immutabili

Abbiamo detto che le stringhe sono un oggetti immutabili, ma se si sa come fare non è del tutto vero, bisogna fare solo molta attenzione.

```
>>> S=c_char_p(b'ciao')           #prendiamo una stringa
>>> B=cast(S, POINTER(c_ubyte))    #convertiamo il puntatore da stringa a ubyte
>>> B[0]=65; B[0]=66; B[0]=67; B[0]=68; #nuova assegnazione
>>> S.value                         # Controlliamo la stringa
B'ABCD'
```

Maneggiare così una stringa può essere pericoloso, quindi ecco qui pronta una funzione che serve allo scopo.

```
def c_change(pchar,strepl,start):
    # controllo sulla coerenza dei tipi
    if not isinstance(pchar, c_char_p): raise TypeError('Era atteso c_char_p come argomento')
    if isinstance(strepl, bytes): mystr=strepl #stringa di bytes
    elif isinstance(strepl, str): mystr=strepl.encode() #unicode converti
    else:
        raise TypeError('Era atteso bytes o str come argomento') #tipo errato
    if not isinstance(start, int): TypeError('Era atteso int come argomento')
    pb=cast(pchar,POINTER(c_ubyte)) # conversione del puntatore da char a ubyte ( R/W)
    lc=len(pchar.value) # lunghezza della stringa
    mystr=mystr[:lc-start] # in caso che la stringa da sostituire sia maggiore del ricevente
    for i,x in enumerate(mystr):
        pb[start+i]=x
```

Ora è possibile sopra scrivere in sicurezza una stringa

Test di prova

```
pchar=c_char_p(b'Ciao Mondo'); s='Bellissima'
```

```
print(f'Prima { pchar.value }\n')
c_change(pchar,s,5)
r=pchar.value ; print(f'Dopo {r}\n')
b'Ciao Belli      questo è il risultato
```

In freebasic, grazie all'operatore [], è più semplice

Dim a As String="Ciao Mondo", b As String="Belli", l As Long

For l=5 to len(a)-1:

a[l]=b[l-5]

next l

Ciao Mondo

Ciao Belli

O semplicemente **Mid(a, 6, 5)="Belli"**

Qualcosa di più complicato

Array e matrici in Freebasic come crearle e accedervi da Python .

Freebasic può creare degli array con indici personalizzabili sia positivi che negativi o misti, in ordine crescente.

Dim my array(-7 to 7) ' Questo è un array di 15 elementi con indice inferiore -7 e superiore 7

Gli array in Freebasic possono avere fino ad un massimo di nove dimensioni.

Implementiamo massimo 3 dimensioni che sono quelle più utilizzate.

Il descrittore di un array in Freebasic. Estratto dal File fbTypes.py

```
FB_MAXDIMENSIONS=4 # Massimo numero di dimensioni, Freebasic supporta 9 dimensioni limitiamo a 3
# l'algoritmo richiamato da __getitem__ e __setitem__ andrà modificato per dimensioni superiori a 3
# In Freebasic invece FB_MAXDIMENSIONS=9 funziona perfettamente
FBARRAY_FLAGS_DIMENSIONS = 0x0000000f # number of entries allocated in arraydim()
FBARRAY_FLAGS_FIXED_DIM = 0x00000010 # array has fixed number of dimensions
FBARRAY_FLAGS_FIXED_LEN = 0x00000020 # array points to fixed-length memory
FBARRAY_FLAGS_RESERVED = 0xfffffc0 # reserved, do not use

class FBArrayDim(Structure):
    _fields_ = [('elements', c_size_t), #numero di elementi
                ('lbound', c_ssize_t), # limite inferiore della dimensione
                ('ubound', c_ssize_t)] # limite superiore della dimensione

class FBArray(Structure):
    #struttura di un array
    _fields_ = [('data', c_void_p), #prima struttura dati
                ('data2', c_void_p), #seconda struttura dati per allocazione in runtime
                ('size', c_size_t), #grandezza in byte dell'array
                ('element_len', c_size_t), #sizeof del tipo di dati es. sizeof(c_int)
                ('dimensions', c_size_t), #numero di dimensioni dell'array normalmente 1
                ('flags', c_ssize_t), # FBARRAY_FLAGS
                ('arraydim',(FBArrayDim*FB_MAXDIMENSIONS))] #caratteristiche delle varie dimensioni
```

Il descrittore è ben documentato, vediamo degli esempi.

Esempio 6

Andiamo a creare un funzione in Freebasic che riceva un array di interi e ne ritorni la somma

```
declare function IntArraySum cdecl alias "IntArraySum"(myarray() as Long) as Long
function IntArraySum cdecl alias "IntArraySum"(myarray() as Long) as Long export
    dim as Long conta=0,i=0
    for i=LBound(myarray) to UBound(myArray):
        conta+= myarray(i)
    next
    return conta
end function
```

Le funzioni LBound e UBound ritornano rispettivamente l'indice inferiore e superiore dell'array, il secondo parametro facoltativo (default=1) ritorna la dimensione di cui vogliamo testare i limiti. Il valore 0 ritorna il numero totale di dimensioni dell'array

Ora andiamo a creare un programma python che si interfacci alla libreria IntArraySum.dll
Per comodità mettiamo la struttura nel file di importazione fbTypes.py

```
from fblib.fbTypes import * # la chiamata a ctypes va inserita in questo file per prima
import os
LIBPATH=r'.\lib\ IntArraySum.dll' # libreria nel path corrente
a=(c_long*10)(1,2,3,4,5,6,7,8,9,10) # creo un array di interi di 10 elementi
p=pointer(a) # puntatore all'array
array=FBArray() # struttura array freebasic
array.data=cast(p,c_void_p) # effettua il cast a puntatore generico
array.ptr= None # la seconda struttura non è utilizzata
array.size=len(a)*sizeof(c_long) # grandezza in byte dell'area
array.element_len=sizeof(c_long) # grandezza in byte del tipo utilizzato
array.dimensions=1 # numero di dimensioni
array.flags= FBARRAY_FLAGS_FIXED_DIM
array.elements=len(a) # numero di elementi dell'array
array.arraydim[0].lbound=0 # limite inferiore
array.arraydim[0].ubound=9 # limite superiore
lib=CDLL(LIBPATH) #carica la libreria
IntArraySum=lib.IntArraySum
IntArraySum.argtypes=[POINTER(FBArray)]
IntArraySum.restype= c_long
r=IntArraySum(array)
print(f'La somma del contenuto dell'array={r}')
```

Lanciato il programma otteniamo:

La somma del contenuto dell'array=55

Tutto bene, ma l'indice parte da zero, se noi vogliamo un indice diverso magari negativo.

Purtroppo non basta cambiare gli indici bisogna anche spostare anche il puntatore, in modo che sommato a lbound punti all'area desiderata.

Questo comporta un po' più di lavoro ma ne vale la pena, vediamo come è possibile risolvere la cosa. Andiamo a creare una funzione che ci permetta di utilizzare indici di qualsiasi range, anche negativi.

Creiamo una funzione che dato l'indice inferiore ritorni il giusto puntatore e l'indici adeguati.

```
def CadrI(array,startIndex,Ctype):
    ind = -startIndex # indice inferiore desiderato
    by = byref(a,ind*sizeof(Ctype)) # calcola l'indirizzo corretto
    p = cast(by, c_void_p) # converte in c_void_p
    l=len(array)-1 # numero elementi dell'array
    return (p, startIndex,startIndex+l) # ritorna il puntatore e gli indici
```

Inseriamo nel file di definizioni la funzione CadrI e testiamo il tutto.

```
from fblib.fbTypes import * # la chiamata a ctypes va inserita in questo file per prima
import os
LIBPATH=r'.\lib\IntArraySum.dll'
a=(c_int*10)(1,2,3,4,5,6,7,8,9,10) # creo un array di interi di 10 elementi
r=CadrI(a, -20,c_int) # crea un puntatore adeguato all'indice inferiore, indici range(-20,-11)
array=FBArray() #struttura array freebasic
array.data=r[0] # associa il puntatore
array.ptr= None # la seconda struttura non è utilizzata
array.size=sizeof(r[0]) # grandezza in byte del puntatore
array.element_len=sizeof(c_int) # grandezza in byte del tipo utilizzato
array.dimensions=1 # numero di dimensioni
array.flags= FBARRAY_FLAGS_FIXED_DIM
array.elements=len(a) # numero di elementi dell'array
array.arraydim[0].lbound=r[1] # limite inferiore
array.arraydim[0].ubound=r[2] # limite superiore
lib=CDLL(LIBPATH) #carica la libreria
IntArraySum=lib.IntArraySum
IntArraySum.argtypes=[POINTER(FBArray)]
IntArraySum.restype=c_int
r=IntArraySum(array)
print(f'La somma del contenuto dell'array={r}')
```

Output:

La somma del contenuto dell'array=55

Matrici bidimensionali

Esempio 7: test di passaggio di un array bidimensionale

```
' test di passaggio di un array bidimensionale
' file IntArrayView2.bas
Declare sub IntArrayView cdecl alias " IntArrayView"(myarray() as Long)
Sub IntArrayView cdecl alias " IntArrayView"(myarray() as Long) export
Dim as Long i,j
For i=LBound(myarray,1) to UBound(myarray,1)
  Print
  Print ("Riga " & i) ' l'operatore & converte i e concatena in formato stringa
  For j=LBound(myarray,2) to UBound(myarray,2)
    Print myarray(i,j);
  Next j
Next i
Locate 15,5
Print "Premi un tasto per uscire"
Sleep
End sub
```

Vediamo come implementare il tutto in Python

```
Chiamante Python file IntArrayView.py
from fblib.fbTypes import * # la chiamata a ctypes è inserita in questo file
import os
LIBPATH=r'.\lib\IntArrayView.dll'
a=( c_long *20)(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20)
r1=CadrI(a, -20,c_long) # array con indici da -20 a 0
array=FByteArray2() # struttura array
array.data=r1[0] # puntatore all'area dati
array.ptr=None # Non utilizzato
array.size=len(a)*sizeof(c_long) # grandezza del puntatore in byte
array.element_len=sizeof(c_long) # grandezza dell'elemento in byte
array.dimensions=2 # numero di dimensioni
array.flags= FBARRAY_FLAGS_FIXED_DIM
array.arradim[0].elements=len(a)//2 # divide l'area dati in due
array.arradim[0].lbound=0 # riga iniziale 1 dimensione
array.arradim[0].ubound=1 # riga finale 1 dimensione
array.arradim[1].elements=len(a)//2 # assegno la meta dell'array
array.arradim[1].lbound=r1[1] # indice inferiore della colonna
array.arradim[1].ubound=r1[1]//2+r1[2] #indice superiore della colonna
lib=CDLL(LIBPATH) #carica la libreria
IntArrayView=lib. IntArrayView
IntArrayView.argtypes=[POINTER(FByteArray2)]
IntArrayView (array)
# questo programma va lanciato da console
```

Però *CadrI* funziona sugli indici delle colonne non per le righe che devono partire da zero vediamo come risolvere il problema.

La funzione che segue permette di calcolare lo spostamento per tutte le nove dimensioni possibili in Freebasic, (non utilizzeremo più di 3 dimensioni, vediamo poi il perchè).

```
Calcolo dello spostamento degli indici
def fb_ArrayCalcDiff(*args):
    diff=0
    dimensions=len(args)
    if dimensions<=0: return 0
    for i in range(dimensions-1):
        elements=(args[i+1][1]-args[i+1][0])+1
        diff=(diff+args[i][0])*elements
    diff +=args[dimensions-1][0]
    return -diff
```

Testiamo il funzionamento della funzione.

<i>fb_ArrayCalcDiff test</i>	<i>File IntArrayView2.py</i>
<pre> from fblib.fbTypes import * import os LIBPATH=r'.\lib\IntArrayView2_ca.dll' CTYPE=c_longlong a=(CTYPE*20)(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20) d=fb_ArrayCalcDiff((-64,-60),(-46,-43))*sizeof(CTYPE) r=byref(a,d) array=FBArray() array.data=cast(r,c_void_p) array.ptr=None array.size=20*sizeof(CTYPE) array.element_len=sizeof(CTYPE) array.dimensions=2 <i>#dichiaro un array tipo Dim array(-64 to -60,-46 to -43) as Long</i> array.arraydim[0].elements=5 array.arraydim[0].lbound=-64 array.arraydim[0].ubound=-60 array.arraydim[1].elements=4 array.arraydim[1].lbound=-46 array.arraydim[1].ubound=-43 array.flags=FBARRAY_FLAGS_FIXED_DIM lib=CDLL(LIBPATH) <i>#carica la libreria</i> IntArrayView=lib.IntArrayView IntArrayView.argtypes=[POINTER(FBArray)] IntArrayView(array) <i># questo programma va lanciato da console</i> </pre>	
<p>Output Programma</p> <pre> Riga -64 colonna -46:1 colonna -45:2 colonna -44:3 colonna -43:4 Riga -63 colonna -46:5 colonna -45:6 colonna -44:7 colonna -43:8 Riga -62 colonna -46:9 colonna -45:10 colonna -44:11 colonna -43:12 Riga -61 colonna -46:13 colonna -45:14 colonna -44:15 colonna -43:16 Riga -60 colonna -46:17 colonna -45:18 colonna -44:19 colonna -43:20 </pre>	

Gestire gli array in freebasic in questo modo è un po' pedante, andiamo a creare una classe che ci eviti tutto questo lavoro ripetitivo. E che ci aiuti a manipolare array o la matrici anche da Python. La classe fbDim permette di creare array fino a 3 dimensioni. Esempio:

ar=fbDim(LONG,(-3,3),data=(1,2,3,4,5,6,7)) crea un array di 7 elementi con indici da -3 a 3

E=ar[-1] legge l'elemento -1 *ar[0]=99* setta l'elemento 0

E=m[(2,3)] legge la posizione (2,3) della matrice m, *m[(3,5)]=7* setta l'elemento (3,5) della matrice m

ar.Lbound(d) ritorna il limite inferiore ar.Ubound(d) ritorna il limite superiore d=0 ritorna il numero di dimensioni dell'array.

fbDim come creare un descrittore di un array freebasic

class fbDim: *# questa classe è una parte del file fbTypes*

def __init__(self,Ctype,*args,data=None):

assert len(args)>=1 or len(args) <= FB_MAXDIMENSIONS,"Numero di dimensioni errato"

self._Ctype =Ctype *# tipo di dato esempio c_ulong*

self._fbarray=FBArray() *#Crea un array descrittore*

self._ctarray=None *#ptr area d'allocare*

self._fbarray.dimensions=len(args) *#dimensioni utilizzate*

self._fbarray.element_len=sizeof(self._Ctype) *#grandezza in byte del tipo di dati*

for i,j in enumerate(args): *#assegna i limiti delle dimensioni*

lb,ub=j

self._fbarray.arraydim[i].elements=abs(ub-lb)+1 *#calcolo del numero di elementi*

self._fbarray.arraydim[i].lbound=lb *#limite inferiore dell'array*

self._fbarray.arraydim[i].ubound=ub *#limite superiore dell'array*

self._elements=self.fb_ArrayCalcElement() *#calcola gli elementi dell'array*

self._fbarray.flags=FBARRAY_FLAGS_FIXED_DIM *#flags array fisso non ridimensionabile*

data2=(0,)*self._elements *# tupla con tutti zero*

self._ctarray=(self._Ctype*self._elements)(*data2) *#alloca l'area e la inilizza con zeri*

if data is not None: *#assegna senza problemi di index out se data è presente*

self.__call__(data) *# utilizza il magic method __call__ per settare i valori*

self._diff=self.fb_ArrayCalcDiff()*sizeof(self._Ctype) *# Calcolo dello spostamento*

r=byref(self._ctarray,self._diff) *#sposta il puntatore*

self._fbarray.data=cast(r,c_void_p) *#cast e assegna il puntatore*

self._myarray=cast(r,POINTER(self._Ctype)) *# puntatore per utilizzo da Python*

self._fbarray.ptr=None *# utilizzato da freebasic per una allocazione in runtime*

self._fbarray.size=self._elements*sizeof(Ctype) *#grandezza dell'array in byte*

@property

def getDesc(self):

return self._fbarray *#ritorna il descrittore per utilizzarlo con Freebasic*

def fb_ArrayCalcElement(self):

#calcolo del numero di elementi nell'array

TB=self._fbarray.arraydim

elements=(TB[0].ubound-TB[0].lbound)+1

for i in range(1,self._fbarray.dimensions):

elements*=(TB[i].ubound-TB[i].lbound)+1

return elements

def __call__(self,args): *# assegna i valori dell'array*

while (conta<largs and conta <lctarray):

v=args[conta]

self._ctarray[conta]=v

conta+=1

def fb_ArrayCalcDiff(self):

Calcolo dello spostamento degli indici

TB=self._fbarray.arraydim

diff=0

if self._fbarray.dimensions<=0: return 0

for i in range(self._fbarray.dimensions-1):

elements=(TB[i+1].ubound-TB[i+1].lbound)+1

diff=(diff+TB[i].lbound)*elements

Autore: Marco Salvati

Licenza Tutorial: CC BY 3.0

Email: marcosalvati61@gmail.com

Licenza Software: GPL V.3


```

diff +=TB[self._fbarray.dimensions-1].lbound
return -diff
def _fb_Element_(self, item): #calcolo degli indici in Python - Per dimensioni maggiori di 3 v  ampliata
elements=0; m=len(item)
TL=self._fbarray.arraydim
if self._fbarray.dimensions==1: #array monodimensionale
    el=-self._diff
    return item[1]
else: #matrici a due e tre dimensioni
    el=-self._diff
    for i in range(self._fbarray.dimensions):
        m-=1
        elements+=((item[i]-TB[i].lbound)*TB[i+1].ubound**m)
    elements+=item[-1]
    return el//sizeof(self._Ctype)+elements
def __getitem__(self,item): #legge un elemento dell'array
if isinstance(item,tuple): #Matrice a 2 e 3 dimensioni
    elements=self._fb_Element_(item)
    return self._myarray[elements]
elif isinstance(item,int) and self._fbarray.dimensions==1: #array 1 dimensione
    item2=(0,item)
    elements=self._fb_Element_(item2)
    return self._myarray[elements]
else:
    raise TypeError("L'indice deve essere una tupla tupla per le matrici e int per i vettori")
def __setitem__(self,item,value): #setta un valore nell'array
if isinstance(item,tuple): #Matrice a 2 e 3 dimensioni
    elements=self._fb_Element_(item)
    self._myarray[elements]=value
elif isinstance(item,int) and self._fbarray.dimensions==1: #array 1 dimensione
    item2=(0,item)
    elements=self._fb_Element_(item2)
    self._myarray[elements]=value
else:
    raise TypeError("L'indice deve essere una tupla per le matrici e int per i vettori")
def Lbound(self, d=1): #limite inferiore dell'array
if d==0: return self._fbarray.dimensions # d=0 ritorna il numero di dimensioni
assert d>=1 or d<=FB_MAXDIMENSIONS,"dimensione errata"
return self._fbarray.arraydim[d-1].lbound #ritorna il limite inferiore
def Ubound(self, d=1): #limite superiore dell'array
if d==0: return self._fbarray.dimensions # d=0 ritorna il numero di dimensioni
assert d>=1 or d<=(FB_MAXDIMENSIONS,"dimensione errata"
return self._fbarray.arraydim[d-1].ubound #ritorna il limite superiore

```

Esempio: di una matrice a 3 dimensioni.

File IntArrayView3.bas stampa una matrice a 3 dimensioni

```
' test di stampa di una matrici tridimensionale
declare sub IntArrayView cdecl alias "IntArrayView"(myarray() as long)

sub IntArrayView cdecl alias "IntArrayView"(myarray() as long) export
dim as long i,j,k
for i=Lbound(myarray) to Ubound(myarray)
  for j=Lbound(myarray,2) to Ubound(myarray,2)
    for k=Lbound(myarray,3) to Ubound(myarray,3)
      print i & ":" & j & ":" & k & "=" & myarray(i,j,k),
    next k
  next j
next i
sleep
end sub
```

File IntArrayView3.py stampa una matrice a 3 dimensioni

```
from fblib.fbTypes import *
import os
LIBPATH=r'.\lib\IntArrayView3'
Data=(1,2,3,4, 5,6,7,8, 9,10,11,12, 13,14,15,16, 17,18,19,20, 21,22,23,24, 25,26,27,28,\
      29,30,31,32, 33,34,35,36, 37,38,39,40, 41,42,43,44, 45,46,47,48)
fba=fbDim(LONG,(1,3),(1,4),(1,4),data=None) #Crea un array a 3 dimensioni
fba(data)
array=fba.getDesc #recupera il descrittore
lib=CDLL(LIBPATH)
IntArrayView=lib.IntArrayView
IntArrayView.argtypes=[POINTER(FBArray)]
IntArrayView(array)
# questo programma va lanciato da console
```

Output:

1:1:1=1	1:1:2=2	1:1:3=3	1:1:4=4	1:2:1=5	1:2:2=6	1:2:3=7	1:2:4=8
1:3:1=9	1:3:2=10	1:3:3=11	1:3:4=12	1:4:1=13	1:4:2=14	1:4:3=15	1:4:4=16
2:1:1=17	2:1:2=18	2:1:3=19	2:1:4=20	2:2:1=21	2:2:2=22	2:2:3=23	2:2:4=24
2:3:1=25	2:3:2=26	2:3:3=27	2:3:4=28	2:4:1=29	2:4:2=30	2:4:3=31	2:4:4=32
3:1:1=33	3:1:2=34	3:1:3=35	3:1:4=36	3:2:1=37	3:2:2=38	3:2:3=39	3:2:4=40
3:3:1=41	3:3:2=42	3:3:3=43	3:3:4=44	3:4:1=45	3:4:2=46	3:4:3=47	3:4:4=48

L'uso completo di questa classe in appendice.

La possibilità del Freebasic (dei Basic in generale) di avere gli indici variabili è unica, non conosco altri linguaggi che lo permettono, però Python non è un qualsiasi linguaggio di programmazione, e questa classe fornisce un ponte che unisce i due linguaggi ma può essere utilizzata anche senza

Autore: Marco Salvati
Licenza Tutorial: CC BY 3.0

Email: marcosalvati61@gmail.com
Licenza Software: GPL V.3

Freebasic, Nell'angolo delle chicche potrete trovare in alternativa la classe Dim in puro Python. La classe è molto versatile, è un ottimo esempio di classe contenitore.

L'angolo delle chicche

Dim una classe che implementa array e matrici con indici variabili

Implementiamo in Python degli array multidimensionali con indici variabili anche negativi, con una variante più pythonica, gli indici oltre che in ordine crescente potranno essere anche in ordine decrescente. L'array potranno essere ridimensionati, Slice e altro ancora.

```
class Dim:
    """ Scopo : array con indici personalizzabili
        Autore : Marco Salvati
        Email : salvatimarco61@gmail.com
        Data : 2019-11-28
        Licenza GPL v3"""
    def __init__(self, atype, start, end, data=[]):
        """ atype tipo di array: Qualsiasi tipo o classe python
            esempio d'uso: a=Dim(int,-7,7) crea un array di interi con indici che vanno da -7 a 7"""
        self.__atype=atype # tipo di oggetto
        self.__start=start # indice iniziale
        self.__end=end # indice finale
        step= 1 if end > start else -1 # passo positivo se crescente negative se decrescente
        self.__valori_iniziali={int:0,float:0.0,complex:(0+0j),str:'',\ # inizializza i valori di default degli oggetti
                               bool:False,list:[],tuple:(),dict:{},set:set()}
        if not self.__atype in self.__valori_iniziali: self.__valori_iniziali[self.__atype]=None #se è un tipo sconosciuto None
        self.__range=range(self.__start,self.__end+step,step) # indice range
        self.__posti=len(self.__range) # calcolo dei posti
        self.__array=[(self.__valori_iniziali[self.__atype])]* self.__posti #riserva l'area
        for i in range(len(data)): # sono presenti dei valori da settare
            self.__array[i] = data[i]
    def LBound(self):
        """ Ritorna il limite inferiore dell'array """
        return self.__start
    def UBound(self):
        """ Ritorna il limite superiore dell'array """
        return self.__end
    def __getitem__(self,item):
        if isinstance(item,slice): # ritorna una lista se l'item è uno slice altrimenti il suo valore
            return self.__array[item.start:item.stop:item.step] # fa uso degli indici standart
        elif isinstance(item,int):
            if not item in self.__range :
                raise IndexError("Indice fuori scala")
            return self.__array[self.__range.index(item)]
        else:
            raise IndexError("L'item deve essere intero o slice")
    def __setitem__(self,item,value): # setta una la posizione item con un valore
        if isinstance(item,int):
            if not item in self.__range :
                raise IndexError("Indice fuori scala")
            if self.__is_valid_value(value):
```

```

        self.__array[self.__range.index(item)]=value
    else:
        raise IndexError("il Valore richiesto deve essere {}".format(self.__atype))
def __is_valid_value(self,value): # uso interno
    """ Ritorna True se il valore è del tipo corretto """
    if isinstance(value,self.__atype):
        return True
    else:
        return False
def index(self,*args):
    """ritorna l'indice relativo a un valore dell'array:
    esempio su un array di interi:
    a.index(88) ritorna l'indice del valore 88
    a.index(88,3) ritorna l'indice del valore 88 a partire dall'indice 3
    a.index(88,3,5) ritorna l'indice del valore 88 nel range indici da 3 a """
    if len(args)==1:
        ir=self.__array.index(args[0]) # trova la posizione reale del valore
        return self.__range[ir] # ritorna l'indice relativo
    elif len(args)==2:
        s=self.__range.index(args[1])
        ir=self.__array.index(args[0],s)
        return self.__range[ir]
    elif len(args)==3:
        s=self.__range.index(args[1])
        e=self.__range.index(args[2])
        ir=self.__array.index(args[0],s,e)
        return self.__range[ir]
    else:
        raise ValueError('il valore non non è presente')
def clear(self):
    """ azzerra l'array """
    for i in range(len(self.__array)): self.__array[i]=(self.__valori_iniziali[self.__atype])
def __str__(self):
    """array to string"""
    return str(self.__array)
def __repr__(self):
    """implementazione metodo repr"""
    return "%s.%s('%s',%d,%d,data=%s)" % (self.__class__.__module__, self.__class__.__qualname__, self.__atype,\
self.__start, self.__end, repr(self.__array))
def __iter__(self):
    """Ritorna un iteratore su i valori"""
    return iter(self.__array)
def __contains__(self, key):
    """operatore in"""
    return key in self.__array
def sort(self, reverse=False):
    """Ordina l'array"""
    self.__array.sort(reverse=reverse)

def reverse(self):
    """ inverte l'array """
    self.__array.reverse()
def __reversed__(self):
    """ Ritorna un iteratore con l'array invertito """

```

```

return reversed(self.__array)
def count(self,value):
    """ Conta il numero di ripetizioni di un valore"""
    return self.__array.count(value)
def keys(self):
    """ Ritorna un iteratore sugli indici """
    return self.__range
def append(self,item=1): # aggiungi 1 o più indici
    """ Aggiunge altri posti nell'array default 1 posto"""
    for i in range(item):
        self.__array.append(self.__valori_iniziali[self.__atype])
        self.__end+=item
        self.__range=range(self.__start,self.__end+1)
def __len__self):
    """ Ritorna la lunghezza dell' array"""
    return len(self.__array)

```

La classe è ben documentata ed è un buono studio per le classi contenitore.

Esempio d'uso:

File test_Dim.py

```

from Dim import Dim
A=Dim(float,10,30,[x *3.3 for x in range(10,31)]) # crea un array di 20 float con indici che vanno da 10,30
B=Dim(int, 3,-3,data=[1,2,3,4,5,6,7]) # crea un array di 7 interi con indici misti invertiti e gli assegna dei
valori.

#Esempio di matrice
M=Dim(Dim,1,2) # crea la base della matrice
M[1]=A
M[2]=B
M[1][10]=5.67 #setta il primo indice dell'array A
M[2][-3]=45 #setta il primo indice dell'array B
S=M[2][-2] # ritorna il valore dell'indice -2 della secondo array
#Potete creare matrici miste ma non dimensioni miste ogni dimensione può avere un solo tipo
# stampa i valori in stile freebasic
for x in range(A.LBound(),A.UBound()): print(A[x] ,end=' ')
# stampa i valori in stile Python
for x in A: print(x,end=' ')
print(len(A)) # ritorna il numero di elementi in A
print(sum(A)) # somma il valore degli elementi e ne ritorna il totale

```

Funzioni con puntatori anonimi.

Funzioni come printf del C ricevono una stringa di formattazione, e dei puntatori anonimi che vengono convertiti in un secondo momento a secondo il contenuto della stringa.

Esempio 8: Creare una funzione chiamata *myadd* che riceva una stringa di formattazione e due puntatori anonimi, la stringa può contenere due direttive *INTEGER* o *SINGLE* si converta i puntatori nel tipo adeguato si effettui la somma dei parametri e si ritorni il puntatore in formato anonimo.

File myadd.bas

```
declare function myadd cdecl alias "myadd" (tipo as string,byval a as any ptr,byval b as any ptr) as any ptr

function myadd cdecl alias "myadd" (tipo as string,byval a as any ptr,byval b as any ptr) as any ptr export
    dim s as string=UCase(tipo) ' trasforma tutto in maiuscolo non sarebbe necessaria perché Python lo fa già
    select case s
    case "INTEGER"
        dim i as Long
        dim as Long r ptr p1=a,p2=b
        i=(*p1+*p2)
        return @i
    case "SINGLE"
        dim c as single
        dim as Single ptr p1=a,p2=b
        c=(*p1+*p2)
        return @c
    end select
End function
```

Dopo aver compilato la funzione, passiamo su python.

File myadd.py

```
from fblib.fbTypes import * # includiamo la struttura FBSTRING
import os
LIBPATH=r'.\lib\myadd.dll' #adeguare il path e l'estensione secondo il sistema operativo
lib=CDLL(LIBPATH)
def myadd(tipo,a,b):
    global lib
    tipo=tipo.upper() # Trasforma la stringa in maiuscolo
    tipo2=tipo.encode() # stringa di bytes no unicode
    s=FBSTRING(tipo2,len(tipo),1)
    if tipo=='INTEGER':
        a1=c_int(a)
        b1=c_int(b)
        lib.myadd.argtypes=[POINTER(FBSTRING),POINTER(c_int) ,POINTER(c_int) ]
        lib.myadd.restype=POINTER(c_int)
        r=lib.myadd(s,a1,b1)
        return r[0] # r.contents.value
    elif tipo=='SINGLE':
        a2=c_float(a)
        b2=c_float(b)
        lib.myadd.argtypes=[POINTER(FBSTRING),POINTER(c_float) ,POINTER(c_float) ]
        lib.myadd.restype=POINTER(c_float)
        r=lib.myadd(s,a2,b2)
        return r[0] # r.contents.value
    else:
        raise TypeError('Solo INTEGER o SINGLE')
print('Somma di 2 interi a+b={}'.format(myadd('Integer',5,7)))
```

```
print('Somma di 2 float a+b={}'.format(myadd('Single',5.7,7.8)))
```

Somma di 2 interi a+b= 11

Somma di 2 float a+b=13.5

Strutture e Unioni.

Definiamo una semplice struttura per un banale calcolo

Esempio 9 Struttura

Type myStruct

as Long x=0

as Long y=0

as Long risultato=0

End type

‘Usiamo la struttura come input e output

Declare sub structTest cdecl alias "structSum" (ByRef s1 as myStruct)

Sub structSum cdecl alias "structSum"(ByRef s1 as myStruct) export

s1.risultato = s1.x + s1.y

End Sub

La chiamante python è altrettanto semplice

Esempio 9

```
from ctypes import *
```

```
LIBPATH=r'.\lib\structSum.dll
```

```
class myStruct(Structure):
```

```
    _fields_=[('x',c_long),
```

```
              ('y',c_long),
```

```
              ('risultato',c_long)]
```

```
lib=CDLL(LIBPATH)
```

```
a=myStruct(13,76,0)
```

```
lib.structSum.argtypes=[POINTER(myStruct)]
```

```
lib.structSum(a)
```

```
print(f'{a.x} + {a.y} = {a.risultato}')
```

```
input('Premi enter per uscire')
```

Se invece mandiamo un puntatore alla struttura?

Esempio 9b versione puntatore a struttura

Type myStruct

as integer x=0

as integer y=0

as Integer risultato=0

End Type

‘Usiamo la struttura come input e output.

Declare sub structTest cdecl alias "structSum" (s1 as myStruct Ptr)

Autore: Marco Salvati

Licenza Tutorial: CC BY 3.0

Email: marcosalvati61@gmail.com

Licenza Software: GPL V.3

```
Sub structSum cdecl alias "structSum"( s1 as myStruct Ptr) export  
  s1.risultato = s1->x + s1->y  
End Sub
```

La parte in Python non cambia.

Le unioni sono simili alle strutture, eccetto che gli elementi di un unione occupano lo stesso spazio.

```
class HiLo(Union):  
  _fields_=[('a',c_ubyte*2),  
            ('b',c_uint)]  
>>> x=HiLo()  
>>> x.b=1234  
>>> print(f'LowByte={x.a[0]} - HiByte={x.a[1]}')  
LowByte=210 - HiByte=4
```

Valgono le stesse regole delle strutture, si possono avere combinazioni miste di strutture e unioni.

```
Type ComType  
  s As String * 20  
  ui As Byte 'Flag to tell us what to use in union.  
  Union  
    au As UByte  
    bu As Integer  
  End Union  
End Type
```

Come chiamare Python da Freebasic.

Ctypes consente di avere dei puntatori a funzioni richiamabili Freebasic da chiamabili Python

In primis, è necessario creare una classe per la funzione di callback. La classe conosce la convenzione di chiamata, il numero di argomenti che questa funzione riceverà e il tipo restituito.

La funzione CFUNCTYPE(), crea protipi per funzioni di callback utilizzando la cdecl convenzione di chiamata l'equivalente per le stcall è WINFUNCTYPE(), che non tratterò, ma funzionano allo stesso modo.

Esempio 10: callback.bas

```
Type operation As Function cdecl (As Long, As Long r) As Long 'crea un nuovo tipo da una funzione
Declare function test cdecl alias "test" (byval a as Long,b as Long r,op as operation) as Long
Function test cdecl alias "test" (byval a as Long,b as Long,op as operation) as Long Export
    dim r as Long
    r=(op(a,b))
    print "Freebasic dice "+str(a) + "+" + str(b)+ "=" + str(r)
    return r
End Function
```

Esempio 10 chiamante Python callback.py

```
from ctypes import * # ricevente python
LIBPATH=r'.\lib\callback.dll
lib=CDLL(LIBPATH)
test=lib.test # callback Python
@CFUNCTYPE(c_long,c_long,c_long) # definisce il prototype del funzione py_sum il primo parametro è quello
di ritorno (return, a, b)
def py_sum(a,b):
    return a+b
a,b=5,6
r=test(a,b,py_sum) # chiamo la funzione Freebasic che richiama Python
print(f'Python dice: {a}+{b}={r}')
input('premi enter per uscire')
```

Eseguendo da console il modulo Python.

Freebasic dice 5+6=11

Python dice 5+6=11

premi enter per uscire

L'angolo delle chicche, Select Case per Python.

Molti linguaggi hanno varie forme di scelte condizionale multiple dal switch del C e altri, al Select Case del Basic, Python non ha questo costrutto si può imitare con un dizionario, ma non la stessa eleganza ed elasticità. Vediamo la sintassi della struttura Select Case del Basic che andiamo a riprodurre.

<pre>Select Case expression [Case expressionlist] [statements] Case Else] [statements] End Select or Select Case As Const integer_expression [Case constant enumeration] [statements] [Case Else] [statements] End Select</pre>	<pre>Select Case As Const i Case 1, 3, 5, 7, 9 dummy += 1 Case 2, 4, 6, 8, 10 dummy += 1 Case 11 To 20 dummy += 1 Case 21 To 30 dummy += 1 Case 31 dummy += 1 Case 32 dummy += 1 Case 33 dummy += 1 Case Else If(i >= 34) Then dummy += 1 Else Print "can't happen" End If End Select</pre>
Case 1	constant
Case 5.4 To 10.1	range
Case Is > 3	bigger than-smaller than
Case 1, 3, 5, 7 to 9	match against a set of values
Case x	value of a variable

File pyCase.py

class Case:

```
""" Scopo : Implementare la struttura Select Case del basic
Autore : Marco Salvati
Email : salvatimarco61@gmail.com
Licenza : GPL 3
Data : 2019-11-04 """
```

def __init__(self):

```
self.__case=[] # lista dei casi
self.__else_case=None # opzione Case Else
self.__else_arg=() # Case Else argomenti
self.__label=None # label in select
```

def Select(self,label=None): # label opzionale

```
""" inizializza il costrutto """
```

```
self.__label=label
```

def __op(self,IS,label): #valuta il confronto

```
d={'==':self.__label==label,'<=':self.__label<=label,'>':self.__label>label \
'>=':self.__label>=label,'!=':self.__label!=label,'<':self.__label<label }
return d
```

def Range(self,start,stop, command=None, arg=()): # qui la label è obbligatoria

```
""" True se label è compreso tra start e stop """
```

```
if self.__label is not None: # se è presente la label
    cond=self.__label >=start and self.__label <= stop
    self.__case.append((cond, command, arg))
else: raise('label not present in Select')
```

def In(self,tupla, command=None, arg=()): # label in tupla o qualsiasi oggetto che supporti il metodo __contains__

```
cond=self.__label in tupla
self.__case.append((cond, command, arg))
```

def Is(self,label,cmp=='==', command=None, arg=()): # metodo di confronto con la label

```
if self.__label is not None: # se è presente la label
    cond=self.__op(cmp, label)
    self.__case.append((cond, command, arg))
else: raise('label not present in Select')
```

def Cmp(self,cond, command=None, arg=()): # non necessità di label testa qualsiasi condizione

```
"""testa qualsiasi condizione"""
self.__case.append((cond, command, arg))
```

def Else(self, command, arg=()):

```
"""viene eseguito se nessun caso è verificato"""
self.__else_case=command; self.__else_arg=arg
```

@property

def End(self):

```
"""Chiude la struttura ed esegue l'azione"""
```

```
eseguito=False
```

```
for t in self.__case:
```

```
    cond,command,arg=t
```

```
    if cond: # la clausola è vera
```

```
        if command : command(*arg) # se command è abilitato eseguillo
```

```
        eseguito=True
```

```
        break # esci dal ciclo
```

```
if not eseguito and self.__else_case is not None: # se esiste else case e nessuna altra opzione è stata eseguità
```

```
    self.__else_case(*self.__else_arg) # esegui
```

```
self.__init__() # ripristina i default per una nuova selezione
```

La classe si auto resetta a fine compito e pronta per un nuovo utilizzo.

Autore: Marco Salvati

Licenza Tutorial: CC BY 3.0

Email: marcosalvati61@gmail.com

Licenza Software: GPL V.3

Un semplice test, per semplificare utilizzerò solo le funzioni print ed exec.

```
from pyCase import Case
```

```
# _____ Esempio 1 _____
```

```
case=Case()
case.Select(10)
case.Is(1, command=print,arg=('voce 1',))
case.Is(2, command=print,arg=('voce 2',))
case.Range(3, 12, command=print,arg=('voci da 3 a 12','Ok'))
case.Is(8, cmp='>=', command=print,arg=('voce >= 8',))
case.Else(command=print,arg=('altro',))
case.End
```

```
# _____ Esempio 2 _____
```

```
case.Select('8b')
case.Is('1', command=print,arg=('voce 1',))
case.Is('2', command=print,arg=('voce 2',))
case.Range('A', 'Z', command=print,arg=('Lettera da A a Z',))
case.Is('8b', command=print,arg=('voce = 8b','Ok'))
case.Else(command=print,arg=('altro',))
case.End
```

```
# _____ Esempio 3 _____
```

```
case.Select('Ab')
case.Is('1', command=print,arg=('voce 1',))
case.Is('2', command=print,arg=('voce 2',))
case.Range('a', 'z', command=print,arg=('Lettera da a a z',))
case.Is('8b', cmp='!=', command=print,arg=('voce >= 8b',))
case.Else(command=exec,arg=('print('Ciao Ragazzi')',))
case.End
```

```
# _____ Esempio 4 _____
```

```
case.Select(5)
case.In((1,2,3,4,5,11,33),command=print,arg=('In Ok',))
case.Is(21, command=print,arg=('voce 1',))
case.Is(33, command=print,arg=('voce 2',))
case.Range(65, 77, command=print,arg=('range(65,77)',))
case.Is(9, cmp='!=', command=print,arg=('voce >= 8b',))
case.Else(command=exec,arg=('print('Ciao Ragazzi')',))
case.End
```

```
# _____ Esempio 5 _____
```

```
a="Pluto"
case.Select()
case.Cmp(a == "Pippo", print,arg=('voce 1',))
case.Cmp(a >= "Pippo", print,arg=('voce 2',))
case.Cmp(a == "Ciao", command=print,arg=('voce 3',))
case.Else(command=exec,arg=('print('Ciao Ragazzi')',))
case.End
```

Output dei 5 case:

voci da 3 a 12 Ok
voce = 8b Ok
Ciao Ragazzi
In Ok
voce 2

La classe è molto versatile e non ha nulla da invidiare all'originale o allo switch del C, o di Java.

Le classi in Freebasic.

Freebasic a una buona orientazione ad oggetti, overload delle funzioni e degli operatori, ereditarietà, metodi astratti e virtuali , etc., ma è mancante (almeno per il momento) per quanto riguarda l'ereditarietà multipla, ma visto che persino linguaggi come java non la supportano pienamente, ma si limitano a simularla con le interfacce, non è nel complesso una grave mancanza.

Gli oggetti utilizzano la stessa definizione delle strutture il costrutto Type ... End Type, è prevista la parola chiave Class per un'estensione futura ma non è ancora implementata.

Un semplice esempio è d'obbligo.

'Questo esempio è tratto dalla documentazione ufficiale

```
Type Vector2D
  As Single x, y
  Declare Operator Cast() As String
  Declare Property Length() As Single
  Declare Property Length( ByVal new_length As Single )
End Type

Operator Vector2D.cast () As String
  Return "(" + Str(x) + ", " + Str(y) + ")"
End Operator

Property Vector2D.Length() As Single
  return Sqr( x * x + y * y )
End Property

Property Vector2D.Length( ByVal new_length As Single )
  Dim m As Single = Length
  If m <> 0 Then
    ' new vector = old / length * new_length
    x *= new_length / m
    y *= new_length / m
  End If
End Property

Dim a As Vector2D = ( 3, 4 )

Print "a = "; a
Print "a.length = "; a.length
Print

a.length = 10

Print "a = "; a
Print "a.length = "; a.length
Sleep 'attesa di un tasto
```

Ricorda un pò il C++ .

Autore: **Marco Salvati**
Licenza Tutorial: **CC BY 3.0**

Email: marcosalvati61@gmail.com
Licenza Software: **GPL V.3**

Facciamo il confronto con Python

```
from math import sqrt

class Vector2D:

    def __init__(self,x,y): # costruttore
        self.x,self.y=0.0,0.0

    def __str__(self): # Operator Vector2D.cast () As String
        return f'({self.x},{self.y})'

    @property
    def length(self):
        return sqrt(self.x*self.x+self.y*self.y)

    @length.setter
    def lenght(self,new_lenght ):
        m= self.length
        if m !=0:
            self.x*=new_lenght / m
            self.y*=new_lenght / m

a = Vector2D(3.0,4.0)
print(f"a = {a}")
print(f"a.length = {a.length} \n")

a.length = 10

print(f"a = {a}")
print(f"a.length = {a.length} ")
```

Output:

```
a = (3.0, 4.0)
a.lenght = 5.0
a = (6.0, 8.0)
a.lenght = 10
```

Python è più compatto ed elegante, Freebasic come tutti i linguaggi compilati è più prolisso nelle dichiarazioni, ma rispetto ad altri linguaggi ad oggetti ad esempio il C++, a mio parere risulta molto più intuitivo, con una sintassi più chiara, (è solo un'opinione personale).

Python non può inizializzare direttamente una classe di un altro linguaggio, quindi dobbiamo fare in modo che sia Freebasic a farlo.

Metodo 1, la classe viene istanziata automaticamente al momento della creazione della libreria con dei valori di default che aggiorneremo in seguito.

Metodo 2, utilizziamo una classe intermedia per inizializzare le classi che ci servono, utile se abbiamo diverse classi ma non tutte ci interessano, si crea una classe che inizializzi individualmente le varie parti della libreria, o una classe con più costruttori.

Ogni membro della nuova classe dovrà essere circondato da un wrapper per potere accedere alle funzionalità.

Esempio 11 - Metodo 1 classe che somma i valori interni

```
Type mysum
  as Integer a0=0,b0=0,c=0
  declare constructor (byval a as integer, byval b as integer)
  declare function somma () as integer
  declare sub change (byval a as integer, byval b as integer)
End Type

Constructor mysum(byval a as integer, byval b as integer)
  this.a0 = a : this.b0 = b : this.c=0
end Constructor
function mysum.somma () as integer
  this.c = this.a0 + this.b0
  print("somma")
  return this.c
end function

sub mysum.change (byval a as integer, byval b as integer)
  this.a0 = a : this.b0 = b : this.c = 0
  print("change")
end sub

dim shared xc as mysum=mysum(3,5) 'inizializza la classe con valori di default
'wrapper
function somma cdecl alias "somma"() as integer export
  return xc.somma()
end function

sub change cdecl alias "change" (byval a as integer,byval b as integer) export
  xc.change(a,b)
end sub
```

Vediamo il lato Python.

```
from ctypes import *
import os
LIBPATH=r'.\lib\mysum.dll'
lib=CDLL(LIBPATH) #carica la libreria
lib.somma.restype=c_int
lib.change.argtypes=[c_int,c_int]
print('somma i valori di default di a+b={}'.format(lib.somma()))
lib.change(c_int(45),c_int(34))
#effettua la somma
print('la somma di 45+34={}'.format(lib.somma()))
lib.change(c_int(88),c_int(63))# cambia i valori
print('la somma di 88+63={}'.format(lib.somma()))
```

Output:

Somma i valori di default 3+5=8

La somma di 45+34=79

La somma di 88+63=151

Esempio 12 - Metodo 2

classe che somma i valori interni

classe che somma i valori interni Metodo 2

```
type mysum
  as Integer a0 = 0, b0 = 0, c = 0
  declare constructor (byval a as integer,byval b as integer)
  declare function somma () as integer
  declare sub change (byval a as integer,byval b as integer)
end type

Constructor mysum(byval a as integer,byval b as integer)
  this.a0 = a : this.b0 = b : this.c = 0
end Constructor

function mysum.somma () as integer
  this.c = this.a0 + this.b0
  print("somma")
  return this.c
end function

sub mysum.change (byval a as integer,byval b as integer)
  this.a0 = a : this.b0 = b :this.c = 0
  print("change")
end sub

type myInit ' classe attivatore
  dim ini as integer
  dim my as mysum=any
  declare Constructor()
  declare sub init (byval a as integer,byval b as integer)
end type

dim shared xc as mysum ptr=any

Constructor myInit()
  this.ini=0
end Constructor

sub myInit.init (byval a as integer,byval b as integer)
  if ini=0 then 'se la classe è già stata inizializzata esci
    this.my=mysum(a,b) 'inizializza la classe
    xc = @my 'associa il puntatore
    this.ini=1 ' una sola inizializzazione
  end if
end sub

dim shared cl as myInit=myInit()
' wrapper alle classi

declare sub init cdecl alias "init" (byval a as integer, byval b as integer)
declare function somma cdecl alias "somma"() as integer
declare sub change cdecl alias "change" (byval a as integer, byval b as integer)

sub init cdecl alias "init" (byval a as integer, byval b as integer) export
  cl.init(a,b) 'inizializza la classe
end sub

function somma cdecl alias "somma"() as integer export
  return xc->somma()
end function

sub change cdecl alias "change" (byval a as integer, byval b as integer) export
  xc->change(a,b)
end sub
```

Test della classe
<pre> from ctypes import * import os LIBPATH=r'.\lib\mysum2.dll' #carica la libreria lib=CDLL(LIBPATH) #inizializza la classe lib.somma.restype=c_int lib.change.argtypes=[c_int,c_int] lib.init.argtypes=[c_int,c_int] lib.init(c_int(3),c_int(5)) print('Somma i valori di default 3+5={}'.format(lib.somma())) lib.change(c_int(45),c_int(34)) # effettua la somma print('La somma di 45+34={}'.format(lib.somma())) lib.change(c_int(88),c_int(63))# cambia i valori print('La somma di 88+63={}'.format(lib.somma())) </pre>
<p>Output:</p> <p><i>Somma i valori di default 3+5=8</i></p> <p><i>La somma di 45+34=79</i></p> <p><i>La somma di 88+63=151</i></p>

Confrontiamo i due linguaggi nella creazione di classi.

Parole chiavi e sintassi a confronto:

Python	Freebasic
<pre> class A(object): def f1(self): pass def f2(self): pass class B(A): def f1(self): pass def f2(self): pass </pre>	<pre> Type A Extends Object Declare Virtual Sub f1() Declare Virtual Function f2() As Integer End Type Type B Extends A Declare Sub f1() Override Declare Function f2() As Integer Override End Type Sub A.f1() End Sub Function A.f2() As Integer End Function Sub B.f1() End Sub Function B.f2() As Integer End Function </pre>
Ereditarietà	

Extends: dichiara una classe derivata

Virtual method: sono quei metodi che possono essere sovrascritti.

Abstract: è una speciale forma di virtual, la differenza è che i metodi astratti non hanno corpo ma solo la dichiarazione

```
Type Hello Extends Object
  Declare Abstract Sub hi( )
End Type

Type HelloEnglish Extends Hello
  Declare Sub hi( )
End Type

Type HelloFrench Extends Hello
  Declare Sub hi( )
End Type

Type HelloGerman Extends Hello
  Declare Sub hi( )
End Type

Sub HelloEnglish.hi( )
  Print "hello!"
End Sub

Sub HelloFrench.hi( )
  Print "Salut!"
End Sub

Sub HelloGerman.hi( )
  Print "Hallo!"
End Sub
```

Override: un metodo dichiarato override, specifica che il metodo deve sovrascrivere uno virtuale.

```
Type A Extends Object
  Declare Virtual Sub f1( )
  Declare Virtual Function f2( ) As Integer
End Type

Type B Extends A
  Declare Sub f1( ) Override
  Declare Function f2( ) As Integer Override
End Type
.....
```

Parole chiavi e sintassi a confronto:

<i>Python</i>	<i>Freebasic</i>
<code>__eq__(self, other): # ==</code>	<code>Operator = (ByRef lhs As T, ByRef rhs As DataType) As DataType</code>
<code>__ne__(self, other): # !=</code>	<code>Operator <> (ByRef lhs As T, ByRef rhs As DataType) As DataType</code>
<code>__lt__(self, other): # <</code>	<code>Operator < (ByRef lhs As T, ByRef rhs As DataType) As DataType</code>
<code>__gt__(self, other): # ></code>	<code>Operator > (ByRef lhs As T, ByRef rhs As DataType) As DataType</code>
<code>__le__(self, other): # <=</code>	<code>Operator <= (ByRef lhs As T, ByRef rhs As DataType) As DataType</code>
<code>__ge__(self, other): # >=</code>	<code>Operator >= (ByRef lhs As T, ByRef rhs As DataType) As DataType</code>
<code>__add__(self, other): # +</code>	<code>Operator + (ByRef lhs As T, ByRef rhs As DataType) As DataType</code>
<code>__sub__(self, other): # -</code>	<code>Operator - (ByRef lhs As T, ByRef rhs As DataType) As DataType</code>

Autore: Marco Salvati
Licenza Tutorial: CC BY 3.0

Email: marcosalvati61@gmail.com
Licenza Software: GPL V.3

<pre> __mul__(self, other): # * __floordiv__(self, other): # // __truediv__(self, other): # / __mod__(self, other): # % __pow__(self, other): # ** __lshift__(self, other): # << __rshift__(self, other): # >> __and__(self, other): # & __or__(self, other): # __xor__(self, other): # ^ __iadd__(self, other): # += __isub__(self, other): # -= __imul__(self, other): # *= __ifloordiv__(self, other): # //= __itruediv__(self, other): # /= __imod__(self, other): # %= __ipow__(self, other): # **= __ilshift__(self, other): # <<= __irshift__(self, other): # >>= __iand__(self, other): # &= __ior__(self, other): # = __ixor__(self, other): # ^= </pre>	<pre> Operator * (ByRef lhs As T, ByRef rhs As DataType) As DataType Operator \ (ByRef lhs As T, ByRef rhs As DataType) As DataType Operator / (ByRef lhs As T, ByRef rhs As DataType) As DataType Operator Mod (ByRef lhs As T, ByRef rhs As DataType) As DataType Operator ^ (ByRef lhs As T, ByRef rhs As DataType) As DataType Operator Shl (ByRef lhs As T, ByRef rhs As DataType) As DataType Operator Shr (ByRef lhs As T, ByRef rhs As DataType) As DataType Operator And (ByRef lhs As T, ByRef rhs As DataType) As DataType Operator Or (ByRef lhs As T, ByRef rhs As DataType) As DataType Operator Xor (ByRef lhs As T, ByRef rhs As DataType) As DataType Operator += (ByRef lhs As T, ByRef rhs As DataType) As DataType Operator -= (ByRef lhs As T, ByRef rhs As DataType) As DataType Operator *= (ByRef lhs As T, ByRef rhs As DataType) As DataType Operator \= (ByRef lhs As T, ByRef rhs As DataType) As DataType Operator /= (ByRef lhs As T, ByRef rhs As DataType) As DataType Operator Mod= (ByRef lhs As T, ByRef rhs As DataType) As DataType Operator ^= (ByRef lhs As T, ByRef rhs As DataType) As DataType Operator Shl= (ByRef lhs As T, ByRef rhs As DataType) As DataType Operator Shr= (ByRef lhs As T, ByRef rhs As DataType) As DataType Operator And= (ByRef lhs As T, ByRef rhs As DataType) As DataType Operator Or= (ByRef lhs As T, ByRef rhs As DataType) As DataType Operator Xor= (ByRef lhs As T, ByRef rhs As DataType) As DataType </pre>
Overloading	

Gli oggetti Python in Freebasic, senza l'uso delle Python api.

Cosa sono le python api:

Le api C/Python sono probabilmente il metodo più utilizzato per creare estensioni, nonostante la loro complessità, per il fatto di poter manipolare gli oggetti Python nel codice C.

Questo metodo richiede che il codice C sia scritto specificatamente per l'interfaccia con il codice Python. Gli header delle Python Api sono stati scritti per il C, e a meno di non effettuarne una traslazione per il Freebasic, non possiamo utilizzarli.

Quello che intendo fare è mostrarvi, come creare delle api scritte completamente in Python con il modulo ctypes, che ci permettano di manipolare gli oggetti Python in Freebasic utilizzando le callback, passando al modulo in Freebasic, una struttura ctypes che simuli una classe nativa. ***Andiamo a creare quindi un ambiente virtuale in Python che permetta a Freebasic, o anche altri linguaggi, di interfacciarsi senza wrapper di alcun tipo.***

Vediamo i passi:

- *Passo 01: Andiamo a creare la struttura che descriverà la classe come una struttura di puntatori ai metodi Python e a metodi creati appositivamente per permettere la comunicazione.*
- *Passo 02: Creare una libreria di classi wrapper ai metodi.*
- *Passo 03: Creare la classe Contains che verrà ereditata da tutti i contenitori*
- *Passo 04: Creare i contenitori in cui assembleremo i vari wrapper.*

Anticipazione ai metodi non Python che utilizzeremo.

- **IsType:** Questa funzione si appoggia a `__getitem__` e controlla il tipo di oggetto all'indice o alla chiave indicata, ritorna un intero che indica il tipo di dato.
- **Let:** Assegna ad un oggetto un nuovo valore.
- **newObj:** Crea un nuovo oggetto dello stesso tipo.
- **Del:** Cancella l'oggetto rilasciando la memoria in Python permette così il riutilizzo dell'etichetta per un nuovo uso, si serve dell'identificatore *id* nella struttura come chiave per eliminare l'oggetto.
- **Id:** Identificatore dell'oggetto usato da Del
- **Wbuff:** Puntatore ad un unicode buffer
- **Zbuff:** Puntatore ad un bytes buffer

Per prima cosa definiamo le strutture (descrittori di classe) a cui i wrapper ai metodi Python si installeranno, e per ognuna la controparte in Freebasic.

Progettiamo delle api in Python e ctypes per il Freebasic, e altri linguaggi

I descrittori delle classi:

File PyStruct.py parte 1 List descrittore

```
from ctypes import *
# struttura che mappa i metodi per le liste
class py2fbList(Structure):
    _fields_=[('getitem', POINTER(c_uint)),
              ('setitem', POINTER(c_uint)),
              ('delitem', POINTER(c_uint)),
              ('append', POINTER(c_uint)),
              ('pop', POINTER(c_uint)),
              ('index', POINTER(c_uint)),
              ('count', POINTER(c_uint)),
              ('remove', POINTER(c_uint)),
              ('insert', POINTER(c_uint)),
              ('reverse', POINTER(c_uint)),
              ('clear', POINTER(c_uint)),
              ('In', POINTER(c_uint)),
              ('isType', POINTER(c_uint)), #controlla il tipo di dato presente (isinstance)
              ('len', POINTER(c_uint)),
              ('str', POINTER(c_uint)),
              ('repr', POINTER(c_uint)),
              ('sort', POINTER(c_uint)),
              ('slice', POINTER(c_uint)),
              ('let', POINTER(c_uint)), # riassegna il contenuto dell'oggetto
              ('copy', POINTER(c_uint)),
              ('newObj', POINTER(c_uint)), # crea una nuova lista
              ('zbuff', POINTER(c_uint)), # bytes buffer
              ('wbuff', POINTER(c_uint)), # str buffer
              ('Del', POINTER(c_uint)), # elimina l'oggetto python
              ('id ', c_int)] # indice chiave associate all'oggetto
```

Come si può notare, la struttura semplicemente mappa i vari metodi e i buffer, come puntatori a interi unsigned, tranne id che un intero e viene utilizzato da Del come identificativo per eliminare l'oggetto.

Questo ci permette di creare una struttura analoga per il linguaggio ospitato, in questo caso il Freebasic, dichiarando i vari puntatori come funzioni, che tramite una callback chiami il wrapper al metodo Python.

File pyobj.bas parte 1: List descrittore

```
type py2fbList # struttura che mappa I metodi per le liste
```

```

  getitem  as sub cdecl(byval tipo as Long,byval index as Long,byval pt as any ptr)
  setitem  as sub cdecl(byval tipo as Long,byval index as Long,byval pt as any ptr)
  delitem  as sub cdecl(byval index as Long)
  append   as sub cdecl(byval tipo as Long,byval pt as any ptr)
  pop      as sub cdecl(byval tipo as Long,byval index as Long,byval pt as any ptr)
  index    as function cdecl(byval tipo as Long,byval pt as any ptr,byval start as Long,byval stop_ as Long) as Long
  count    as function cdecl(byval tipo as Long,byval pt as any ptr) as Long
  remove   as sub cdecl(byval tipo as Long,byval pt as any ptr)
  insert   as sub cdecl(byval tipo as Long,byval index as Long,byval pt as any ptr)
  reverse  as sub cdecl()
  clear    as sub cdecl()
  in       as function cdecl(byval tipo as Long,byval key as any ptr) as Boolean
  isType   as function cdecl(byval index as Long) as Long
  len      as function cdecl() as Long
  str      as function cdecl() as wstring ptr
  repr     as function cdecl() as wstring ptr
  sort     as sub cdecl(byval reverse as Boolean)
  slice    as sub cdecl(byref buffer as py2fbList,byval start as Long,byval end as Long,byval Step as Long)
  let      as sub cdecl(byval t as wstring ptr)
  copy     as sub cdecl(byref buffer as py2fbList)
  newObj  as sub cdecl(byref buffer as py2fbList,byval lista as wstring ptr)
  zbuff    as zstring ptr
  wbuff    as wstring ptr
  del      as sub cdecl(byval t as Long)
  id       as Long
End Type
```

La struttura è sì auto commenta, procediamo con le altre strutture basi e poi passeremo allo sviluppo dell'ambiente.

File PyStruct.py parte 2 Tuple descrittore

```
# struttura che mappa I metodi per le tuple
```

```
class py2fbTuple(Structure):
```

```

    _fields_= [('getitem', POINTER(c_uint)),
                ('In', POINTER(c_uint)),
                ('isType', POINTER(c_uint)), #controlla il tipo di dato presente
                ('len', POINTER(c_uint)),
                ('str', POINTER(c_uint)),
                ('repr', POINTER(c_uint)),
                ('slice', POINTER(c_uint)),
                ('let', POINTER(c_uint)), #riassegna il contenuto dell'oggetto
                ('newObj', POINTER(c_uint)), #crea una nuova tupla
                ('zbuff', POINTER(c_uint)), #bytes buffer
                ('wbuff', POINTER(c_uint)), #str buffer
                ('Del', POINTER(c_uint)), #elimina l'oggetto Python
                ('id', c_int)] #id dell'oggetto Python
```

File pyobj.bas parte 2 : Tuple descrittore

```
type py2fbTuple
  getitem as sub cdecl(byval tipo as Long,byval index as Long ,byval pt as any ptr)
  in      as function cdecl(byval tipo as Long,byval key as any ptr) as Boolean
  isType  as function cdecl(byval index as Long) as Long
  len     as function cdecl() as Long
  index   as function cdecl(byval tipo as Long,byval pt as any ptr,byval start as Long,byval stop_ as Long) as Long
  count   as function cdecl(byval tipo as Long,byval pt as any ptr) as Long
  str     as sub cdecl()
  repr    as sub cdecl()
  slice   as sub cdecl(byref buffer as py2fbTuple,byval start as Long,byval end as Long,byval stop_ as Long)
  let     as sub cdecl(byval t as wstring ptr)
  newObj  as sub cdecl(byref buffer as py2fbTuple,byval tupla as wstring ptr)
  zbuff   as zstring ptr
  wbuff   as wstring ptr
  del     as sub cdecl(byval id as Long)
  id      as Long
End Type
```

File PyStruct.py parte 3 struttura che mappa I metodi per i dizionari

struttura che mappa I metodi per i dizionari

class py2fbDict(Structure):

```
  _fields_=[('getitem', POINTER(c_uint)),
            ('setitem',  POINTER(c_uint)),
            ('delitem',  POINTER(c_uint)),
            ('pop',      POINTER(c_uint)),
            ('clear',    POINTER(c_uint)),
            ('In',       POINTER(c_uint)),
            ('isType',   POINTER(c_uint)), #controlla il tipo di dato presente
            ('len',      POINTER(c_uint)),
            ('str',      POINTER(c_uint)),
            ('repr',     POINTER(c_uint)),
            ('let',      POINTER(c_uint)), #riassegna il contenuto dell'oggetto
            ('get',      POINTER(c_uint)),
            ('items',    POINTER(c_uint)),
            ('keys',     POINTER(c_uint)),
            ('values',   POINTER(c_uint)),
            ('setdefault', POINTER(c_uint)),
            ('popitem',  POINTER(c_uint)),
            ('update',   POINTER(c_uint)),
            ('fromkeys', POINTER(c_uint)),
            ('copy',     POINTER(c_uint)),
            ('newObj',   POINTER(c_uint)), # crea una nuovo dizionario
            ('zbuff',    POINTER(c_uint)), # bytes buffer
            ('wbuff',    POINTER(c_uint)), # str buffer
            ('Del',      POINTER(c_uint)), # elimina l'oggetto Python
            ('id',       c_int)] # id dell'oggetto Python
```


File pyobj.bas parte 3: Dizionario descrittore

type py2fbDict

```

getitem  as sub cdecl(byval tipo as Long,byval key as wstring ptr,byval pt as any ptr)
setitem  as sub cdecl(byval tipo as Long,byval key as wstring ptr,byval pt as any ptr)
delitem  as sub cdecl(byval key as wstring ptr)
pop      as sub cdecl(byval tipo as Long,byval key as wstring ptr,byval default as any ptr,byval pt as any ptr)
clear    as sub cdecl()
in       as function cdecl(key as wstring ptr) as Boolean
isType   as function cdecl(byval key as wstring ptr) as Long
len      as function cdecl() as Long
str      as sub cdecl()
repr     as sub cdecl()
let      as sub cdecl(byval t as wstring ptr)
get      as sub cdecl(byval tipo as Long,byval key as wstring ptr, default as any ptr ,byval pt as any ptr )
items    as sub cdecl(byref buffer as py2fbList)
keys     as sub cdecl(byref buffer as py2fbList)
values   as sub cdecl(byref buffer as py2fbList)
setdefault as sub(byval key as wstring ptr, tipo as Long,byval default as any ptr,pt as any ptr)
popitem  as sub cdecl()
update   as sub cdecl (byval iter as wstring ptr)
fromkeys as sub cdecl (byref buffer as py2fbDict, byval iter as wstring ptr)
copy     as sub cdecl(byref buffer as py2fbDict)
newObj   as sub cdecl(byref buffer as py2fbDict,byval dict as wstring ptr)
zbuff    as zstring ptr
wbuff    as wstring ptr
del      as sub cdecl(byval t as Long)
id       as Long

```

End Type

File PyStruct.py parte 4 Set descrittore**class py2fbSet(Structure):** *# struttura che mappa I metodi per i set*

```

_fields_=[('clear',          POINTER(c_uint)),
('In',          POINTER(c_uint)),
('len',         POINTER(c_uint)),
('str',         POINTER(c_uint)),
('repr',       POINTER(c_uint)),
('let',        POINTER(c_uint)),
('add',        POINTER(c_uint)),
('discard',    POINTER(c_uint)),
('remove',     POINTER(c_uint)),
('union',      POINTER(c_uint)),
('difference', POINTER(c_uint)),
('intersection',POINTER(c_uint)),
('symmetric_difference',POINTER(c_uint)),
('issubset',   POINTER(c_uint)),
('issuperset',POINTER(c_uint)),
('isdisjoint',POINTER(c_uint)),
('symmetric_difference_update',POINTER(c_uint)),
('difference_update',POINTER(c_uint)),
('intersection_update',POINTER(c_uint)),
('update',     POINTER(c_uint)),
('pop',        POINTER(c_uint)),
('copy',       POINTER(c_uint)),
('newObj',     POINTER(c_uint)),

```

#riassegna il contenuto dell'oggetto

crea un nuova set

Autore: Marco Salvati

Licenza Tutorial: CC BY 3.0

Email: marcosalvati61@gmail.com

Licenza Software: GPL V.3

('zbuff',	POINTER(c_uint)),	# bytes buffer
('wbuff',	POINTER(c_uint)),	# str buffer
('Del',	POINTER(c_uint)),	# elimina l'oggetto Python
('id',	c_int)]	# id dell'oggetto Python

File pyobj.bas parte 4: Set descrittore

```

type py2fbDict
    getitem as sub cdecl(byval tipo as Long,byval key as wstring ptr,byval pt as any ptr)
    setitem as sub cdecl(byval tipo as Long,byval key as wstring ptr,byval pt as any ptr)
    delitem as sub cdecl(byval key as wstring ptr)
    pop as sub cdecl(byval tipo as Long,byval key as wstring ptr,byval default as any ptr,byval pt as any ptr)
    clear as sub cdecl()
    in as function cdecl(key as wstring ptr) as Boolean
    isType as function cdecl(byval key as wstring ptr) as Long
    len as function cdecl() as Long
    str as sub cdecl()
    repr as sub cdecl()
    let as sub cdecl(byval t as wstring ptr)
    get as sub cdecl(byval tipo as Long,byval key as wstring ptr, default as any ptr ,byval pt as any ptr )
    items as sub cdecl(byref buffer as py2fbList)
    keys as sub cdecl(byref buffer as py2fbList)
    values as sub cdecl(byref buffer as py2fbList)
    setdefault as sub(byval key as wstring ptr, tipo as Long,byval default as any ptr,pt as any ptr)
    popitem as sub cdecl()
    update as sub cdecl (byval iter as wstring ptr)
    fromkeys as sub cdecl (byref buffer as py2fbDict, byval iter as wstring ptr)
    copy as sub cdecl(byref buffer as py2fbDict)
    newObj as sub cdecl(byref buffer as py2fbDict,byval dict as wstring ptr)
    zbuff as zstring ptr
    wbuff as wstring ptr
    del as sub cdecl(byval t as Long)
    id as Long
End Type

```

File PyStruct.py parte 5 definisce l'unicode string buffer di comunicazione Python Freebasic

```

class pyStrBuffer:
    """ Classe ad una sola istanza """
    _stato=dict()
    def __init__(self,dim=500):
        self._stato=self.__dict__ # nessuna altra istanza è possibile
        if 'flag' not in self._stato: # controllo se prima avvio
            self.__dict__ ['flag']=True
            self.__dict__ ['buff']=create_unicode_buffer(dim) # creo il wstring buffer
            self.__dict__ ['pointer']=pointer(self.__dict__ ['buff']) #puntatore al buffer
            self._stato=self.__dict__ # nessuna altra istanza è possibile
    @property
    def pBuffer(self):
        return self.__dict__['pointer']
    @property
    def buffer(self):
        return self.__dict__['buff'].value
    @buffer.setter
    def buffer(self,value):
        self.__dict__['buff'].value=value

```

File pyobj.bas parte 5 Costanti

' tipi di dati N.B] PyInt e PyUint vengono considerati sempre a 32 bit alias di PyLong e PyULong

```
Const PyBool = 0
Const PyInt = 1
Const PyFloat = 2
Const PyBytes = 3
Const PyStr = 4
Const PyList = 5
Const PyTuple = 6
Const PyDict = 7
Const PySet = 8
Const PyComplex = 9
Const PyDouble = 10
Const PyUint = 11
Const PyLong = 1
Const PyULong = 11
Const PyShort = 12
Const PyUShort = 13
Const PyByte = 14
Const PyUByte = 15
Const PyLongint = 16
Const PYULongint= 17
Dim shared None as any ptr=0 'Puntatore nullo
```

File PyStruct.py parte 6 definisce lo bytes buffer di comunicazione Python Freebasic

```
class pyBytesBuffer:
    """ Classe ad una sola istanza """
    _stato=dict()
    def __init__(self,dim=500):
        self._stato=self.__dict__ # nessuna altra istanza è possibile
        if 'flag' not in self._stato: # controllo se prima avvio
            self.__dict__ ['flag']=True
            self.__dict__ ['buff']=create_string_buffer(dim) # creo il zstring buffer
            self.__dict__ ['pointer']=pointer(self.__dict__ ['buff']) #puntatore al buffer
            self._stato=self.__dict__ # nessuna altra istanza è possibile
    @property
    def pBuffer(self):
        return self.__dict__['pointer']
    @property
    def buffer(self):
        return self.__dict__['buff'].value
    @buffer.setter
    def buffer(self,value):
        self.__dict__['buff'].value=value
```

Ho suddiviso i due file per migliorare il confronto tra i due linguaggi, prima di passare a creare l'ambiente, osserviamo le 4 strutture che rappresentano altrettanti oggetti Python, liste, tuple, dizionari e set.

Sono facilmente riconoscibili i metodi Python, ho commentato quei metodi che non fanno parte dell'oggetto Python e che ritroviamo in tutte le strutture, si tratta di metodi creati per permettere a Freebasic di utilizzare gli oggetti.

Una rapida occhiata alle classi pyByteBuffer e pyStrBuffer, sono un'implementazione Singleton di due buffer, bytes e unicode, tutti gli oggetti avranno in condivisione questi buffer per ritornare stringhe di bytes e unicode, questi buffer verranno utilizzati oltre che per semplici stringhe anche per il ritorno di oggetti che Freebasic non può trattare direttamente liste, tuple, numeri complessi etc., ovviamente come stringhe in formato repr.

Le classi wrapper:

File pybase.py: Classe let assegnamento di un valore a un oggetto
metodi comuni
<pre>from ctypes import * from fblib.PyObj.PyStruct import * # Implementa il metodo Freebasic di riassegnamento class let: def __init__(self,obj,py2fbStruct): """ Wrapper assegnamento nuovo valore """ self.obj=obj # contenuto dell'oggetto (lista, tupla etc..) self.py2fbStruct=py2fbStruct # struttura a cui agganciare il wrapper let=CFUNCTYPE(None,c_wchar_p) # riceve una stringa unicode self.py2fbStruct.let=cast(let(self._let_),POINTER(c_uint)) #aggancia il metodo _let_ def _let_(self,t): # metodo Freebasic t2=eval(t) # valuta la stringa self.obj=t2 #assegna il nuovo valore @property def value(self): # ritorna il valore utilizzato da Python return self.obj</pre>

Tutti gli oggetti
let as sub cdecl(byval s as wstring ptr)
Mytuple.let(("11,13,17,19")) ' associa alla tupla un nuovo valore

File pybase.py: Questa classe implementa il metodo isType per il Freebasic:

```
class is_type: # wrapper al metodo Freebasic isType
    def __init__(self,obj,py2fbStruct,indexInt):
        """ Wrapper per Controllo del tipo in un contenitore """
        self.let=obj # metodo let
        self.obj=self.let.value # oggetto contenuto in let
        self.maxFloat=3.4*10**38
        self.minFloat=-3.4*10**(-38)
        self.py2fbStruct=py2fbStruct # struttura a cui agganciare il Freebasic wrapper
        # True indice numerico (liste e tuple) False stringa(dizionario)
        if indexInt: # True lista o tupla
            isType=CFUNCTYPE(c_int,c_int)
        else: # False dizionario
            isType=CFUNCTYPE(c_int,c_wchar_p)
        self.py2fbStruct.isType=cast(isType(self._is_type_),POINTER(c_uint)) # aggancio il puntatore
    def _is_type_(self,index): # funzione wrapper
        self.obj=self.let.value # valore corrente dell'oggetto
        if isinstance(self.obj[index],bool): # controllo il tipo dell'oggetto via __getitem__
            return 0 # bool
        elif isinstance(self.obj[index],int): #tipo intero
            return self._intBit_(self.obj[index]) #controlla che tipo di intero
        elif isinstance(self.obj[index],float): # float
            if self.obj[index]> self.maxFloat or self.obj[index]<self.minFloat : # controllo sul tipo di float
                return 10 #double
            return 2 #single
        elif isinstance(self.obj[index],bytes): #zstring
            return 3
        elif isinstance(self.obj[index],str): #wstring
            return 4
        elif isinstance(self.obj[index],list): #lista
            return 5
        elif isinstance(self.obj[index],tuple): #tupla
            return 6
        elif isinstance(self.obj[index],dict): #dizionario
            return 7
        elif isinstance(self.obj[index],set): #set
            return 8
        elif isinstance(self.obj[index],complex): #complex
            return 9
        else: # tipo non implementato
            return -1 # non riconosciuto

    def _intBit_(self,t): # controllo per i vari interi stile Freebasic
        if t.bit_length()<=7: return 14 #byte
        elif t.bit_length()==8: return 15 #ubyte
        elif t.bit_length() in range(9,16): return 12 # short
        elif t.bit_length()==16: return 13 # Ushort
        elif t.bit_length() in range(17,32): return 1 #pyInt PyLong
        elif t.bit_length()==32: return 11 #Pyuint PyuLong
        elif t.bit_length() in range(33,64): return 16 #Pylongint
        elif t.bit_length()==64: return 17 #Pyulongint
```

Il metodo utilizzato per adeguare un intero Python a quelli di Freebasic è forse la parte più interessante, l'uso della funzione bit_length(), al contrario del Freebasic, un intero in Python 3.x può avere una qualsiasi dimensione limitata solo dalla memoria del computer, la suddetta funzione ritorna il numero di bit utilizzato dal quel numero, quindi rapportiamo il numero di bit alla fascia equivalente per Freebasic.

Autore: Marco Salvati

Licenza Tutorial: CC BY 3.0

Email: marcosalvati61@gmail.com

Licenza Software: GPL V.3

Vediamo l'uso di *isType* in *Freebasic*:

IsType si appoggia al magic method `__getitem__` di Python per cui non può essere implementata se l'oggetto Python non supporta questo metodo, vedi `set`.

Tuple liste e simili	Dizionari
<code>isType as function cdecl(byval index as integer) as integer</code>	<code>isType as function cdecl(byval key as wstring ptr) as integer</code>
<pre>Tipo=myObj.isType(index) Select Case tipo Case PyInt, pyByte, pyUbyte, pyshort, pyUshort, pyUint ' uno qualsiasi di questi tipi ' istruzioni Case PyFloat ' Single ' istruzioni Case PyBytes ' Zstring ' istruzioni Case End Select</pre>	

Facciamo un controllo del tipo di dato contenuto nella posizione indicata da `index`, ed eseguiamo le istruzioni adeguate.

File *pybase.py*: La funzione *ptr_convert* è di uso interno, ma funge da pilastro nel passaggio dei parametri.

```
def ptr_convert(tipo, ptr): #funzione che converte il puntatore a secondo il tipo richiesto
    if tipo==0: #Bool
        p=cast(ptr,POINTER(c_bool)) #cast puntatori
    elif tipo==1: #int/long
        p=cast(ptr,POINTER(c_int))
    elif tipo==2: #float
        p=cast(ptr,POINTER(c_float))
    elif tipo==3: #zstring
        p=cast(ptr,c_char_p)
    elif tipo in (4,5,6,7,8,9): #wstring
        p=cast(ptr,c_wchar_p)
    elif tipo==10: #double
        p=cast(ptr,POINTER(c_double))
    elif tipo==11: #uint/ulong
        p=cast(ptr,POINTER(c_uint))
    elif tipo==12: #Short
        p=cast(ptr,POINTER(c_short))
    elif tipo==13: #UShort
        p=cast(ptr,POINTER(c_ushort))
    elif tipo==14: #Byte
        p=cast(ptr,POINTER(c_byte))
    elif tipo==15: #Ubyte
        p=cast(ptr,POINTER(c_ubyte))
    elif tipo==16: #longInt
        p=cast(ptr,POINTER(c_longlong))
    elif tipo==17: #uLongInt
        p=cast(ptr,POINTER(c_ulonglong))
    return p
```

File pybase.py: Questa classe implementa il magic method `__getitem__` per il Freebasic:

```
class getitem:
def __init__(self,obj,py2fbStruct,Buffer,tipoint):
    """ Wrapper al magic method __getitem__ """
    self.let=obj # classe let
    self.obj=self.let.value #oggetto in let
    self.py2fbStruct=py2fbStruct #struttura a cui agganciare il metodo
    self.buffer=Buffer #wstring buffer e zstring buffer
    if tipoint:
        ass=CFUNCTYPE(None,c_int,c_int,c_void_p) #Liste,Tuple e simili
    else:
        ass=CFUNCTYPE(None,c_int,c_wchar_p,c_void_p) #Dizionari
    self.py2fbStruct.getitem=cast(ass(self.__getitem__),POINTER(c_uint))
def __getitem__(self,tipoint,index,ptr):
    self.obj=self.let.value
    v=self.obj[index]
    p=ptr_convert(tipo,ptr) #converte il puntatore nel tipo adeguato
    if tipo in (0,1,2,10,11,12,13,14,15,16,17):
        p[0]=v
    elif tipo==3: #Zstring
        self.buffer.strbuff=v.decode() #wstring ritorna in wbuff
        self.buffer.bytbuff=v          #zstring ritorna in zbuff
    elif tipo==4: #Wstring
        self.buffer.strbuff=v          #wstring ritorna in wbuff
        self.buffer.bytbuff=v.encode() #zstring ritorna in zbuff
    elif tipo in (5,6,7,8,9):
        #list, tuple, dict, set, complex in formato string
        v=repr(v)
        self.buffer.strbuff=v          #wstring ritorna in wbuff
        self.buffer.bytbuff=v.encode() #wstring ritorna in wbuff
```

Le due forme di questa funzione

<i>Tuple liste e simili</i>	<i>Dizionari</i>
Getitem as function cdecl(byval tipo,byval index as long,byval Ptrvar) as Long	Getitem as function cdecl(byval,tipoint,byval key as wstring ptr,byval Ptrvar) as Long

Vediamo un esempio dell'uso di getitem in Freebasic:

```
#include once "pyObj.bi"

dim shared myTuple as py2fbTuple ' struttura che ospiterà quella inviata da Python
declare sub init cdecl alias "init" (byref t as py2fbTuple)
declare sub test cdecl alias "test"()

sub init cdecl alias "init" (byref t as py2fbTuple) export
    myTuple=t ' associa la struttura ctypes a quella di Freebasic
end sub

sub test cdecl alias "test"() export
    print"-----Len e Str test-----"
    myTuple.str(): print *myTuple.Wbuff ' stampa in contenuto in formato wstring
    print"Numero di elementi nella Tupla "&myTuple.len()
    print
    print"-----Getitem test-----"
    print "Mostro tutti gli elementi della tupla"
    dim i as Long, tipo as integer, sh as short, ush as ushort
    dim b as Boolean, iv as Long, f as single, li as longint, uli as ulongint
    dim by as byte, uby as ubyte, d as double, uiv as ULong
    dim z as zstring ptr, w as wstring ptr
    print "Nella Tupla ci sono "&myTuple.len() &" elementi"
    for i=0 to myTuple.len()-1
        tipo=myTuple.isType(i)
        select case tipo
            case PyBool ' Boolean
                myTuple.getitem(tipo,i,@b)
                print i & " Boolean "& b
            case PyInt ' Integer/Long
                myTuple.getitem(tipo,i,@iv)
                print i & " Long "& iv
            case PyUInt ' UInteger/ULong
                myTuple.getitem(tipo,i,@uiv)
                print i & " ULong "& uiv
            case PyByte ' byte
                myTuple.getitem(tipo,i,@by)
                print i & " Byte "& by
            case PyUbyte ' ubyte
                myTuple.getitem(tipo,i,@uby)
                print i & " UByte "& uby
            case PyShort ' short
                myTuple.getitem(tipo,i,@sh)
                print i & " Short "& sh
            case PyUShort ' Ushort
                myTuple.getitem(tipo,i,@ush)
                print i & " Ushort "& ush
            case PyLongInt ' LonInt
                myTuple.getitem(tipo,i,@li)
                print i & " LongInt "& li
            case PyULongInt ' ULongInt
                myTuple.getitem(tipo,i,@uli)
                print i & " ULongInt "& uli
            case PyFloat ' Single
                myTuple.getitem(tipo,i,@f)
                print i & " Single "& f
```



```

case PyDouble ' Double
    myTuple.getitem(tipo,i,@d)
    print i & " Double " &d
case PyBytes ' Zstring
    myTuple.getitem(tipo,i,None)
    print i & " Zstring "& *myTuple.Zbuff
case PyStr ' Wstring
    myTuple.getitem(tipo,i,None)
    print i & " Wstring " & *myTuple.Wbuff
case PyList ' List
    myTuple.getitem(tipo,i,None)
    print i & " List " & *myTuple.Wbuff
case PyTuple ' Tuple visualizzo solo il contenuto senza creare l'oggetto
    myTuple.getitem(tipo,i,None)
    print i & " Tuple " & *myTuple.Wbuff
case PyDict ' Dict
    myTuple.getitem(tipo,i,None) ' visualizzo solo il contenuto senza creare l'oggetto
    print i & " Dict " & *myTuple.Wbuff
case PySet ' Set
    myTuple.getitem(tipo,i,None) ' visualizzo solo il contenuto senza creare l'oggetto
    print i & " Set " & *myTuple.Wbuff
case PyComplex ' Complex
    myTuple.getitem(tipo,i,None) ' visualizzo solo il contenuto l'oggetto non è implementato
    print i & " Complex " & *myTuple.Wbuff
case else
    print i & " tipo sconosciuto"
end select
next i
print
end sub

```

Un esempio completo dell'uso di getitem in Freebasic con alcune anticipazioni di altri metodi.

File pybase.py: *Questa classe implementa il magic method `__getitem__` per il Freebasic:*

```

class setitem:
def __init__(self,obj,py2fbStruct,tipoInt):
    """ Wrapper al magic method __getitem__ """
    self.let=obj #metodo let
    self.obj=self.let.value #oggetto in let
    self.pystrb=pyStrBuffer() # wstring buffer
    self.pybytb=pyBytesBuffer() # zstring buffer
    self.py2fbStruct=py2fbStruct #struttura dove agganciare il metodo
    self.tipoInt=tipoInt
    if tipoInt: #list o tuple
        ass=CFUNCTYPE(None,c_int,c_int,c_void_p)
    else: #dizionario
        ass=CFUNCTYPE(None,c_int,c_wchar_p,c_void_p)
    self.py2fbStruct.setitem=cast(ass(self._setitem_),POINTER(c_uint))
def _setitem_(self,tipo,index,ptr):
    self.obj=self.let.value
    p=ptr_convert(tipo,ptr) #converte il puntatore nel tipo adeguato
    if tipo in (0,1,2,10,11,12,13,14,15,16,17): #assegnamento numerico
        self.obj[index]=p[0]
    elif tipo in (3,4): #assegnamento alfanumerico (stringa o bytes)
        self.obj[index]=p.value

```

elif tipo in (5,6,7,8,9): *#assegnamento oggetto list, tuple, dict, set, complex*
self.obj[index]=eval(p.value)

Vediamo le due forme di questa funzione

Tuple liste e simili	Dizionari
setitem as sub cdecl(byval tipo as integer,byval index as integer,byval pt as any ptr)	setitem as sub cdecl(byval tipo as integer,byval key as wstring ptr,byval pt as any ptr)

Vediamo un esempio dell'uso di setitem in Freebasic:

```
dim key as wstring *100 ,tmp as wstring *100 ,iv as long
print "-----setitem test-----"
print "Aggiungo delle nuove chiavi al dizionario"
key="gatto":tmp="Kitty"
myDict.setitem(PyStr,@key,@tmp)
iv=999:key="numero"
myDict.setitem(pyInt,@key,@iv) 'intero
key="lista2":tmp="[1,2,3]"
myDict.setitem(PyList,@key,@tmp) ' con wstring buffer
print "Mostro tutti gli elementi della lista"
myDict.str()
print *myDict.Wbuff
print "Elimino la voce lista2"
myDict.delitem(@key)
print "Mostro il nuovo contenuto"
myDict.str()
```

File pybase.py: Questa classe implementa il magic method __delitem__ per il Freebasic:

```
class delitem:
def __init__(self,obj,py2fbStruct,tipoInt):
    """ Wrapper al magic method __delitem__ """
    self.let=obj
    self.obj=self.let.value
    self.py2fbStruct=py2fbStruct
    if tipoInt: # se tipoInt List
        d=CFUNCTYPE(None,c_int)
    else: # Dict
        d=CFUNCTYPE(None,c_wchar_p)
    self.py2fbStruct.delitem=cast(d(self._delitem_),POINTER(c_uint))
def _delitem_(self,index):
    self.obj=self.let.value
    del(self.obj[index])
```

Vediamo le due forme di questa funzione

Liste e simili	Dizionari
delitem as sub cdecl(byval index as integer)	delitem as sub cdecl(byval key as wstring ptr)
myList.delitem(5) <i>‘cancella l’item numero 5 della lista</i> myDict.delitem(‘Cognome’) <i>‘Cancella la chiave “Cognome” del dizionario</i>	

File pybase.py: Questa classe implementa i metodi copy e clear per il Freebasic:

```

class copy_clear: # usato da list,dict, set e simili
def __init__(self,obj,py2fbStruct,myClass,ClassPtr,mem):
    """ Wrapper ai metodi copy e clear """
    self.let=obj
    self.obj=self.let.value
    self.py2fbStruct=py2fbStruct
    self.myClass=myClass
    self.ClassPtr=ClassPtr
    self.mem=mem
    #-----Copy-----
    ass=CFUNCTYPE(None,POINTER(self.ClassPtr))
    self.py2fbStruct.copy=cast(ass(self._copy_),POINTER(c_uint)) # associa la il metodo _copy_
    #-----Clear-----
    clear=CFUNCTYPE(None) # utilizza il metodo originale
    self.py2fbStruct.clear=cast(clear(self.obj.clear),POINTER(c_uint)) #utilizza il metodo originale

def _copy_(self,buffer):
    self.obj=self.let.value
    d=self.obj.copy();key=len(self.mem)
    self.mem[key]=self.myClass(d) # creo un nuovo oggetto
    memmove(buffer,byref(self.mem[key].getObj),sizeof(self.py2fbStruct)) #copio il descrittore

```

Vediamo un esempio dell’uso di copy e clear in Freebasic:

copy	clear
Copy as sub cdecl(byref buffer as py2fbList) Copy as sub cdecl(byref buffer as py2fbDict) Copy as sub cdecl(byref buffer as py2fbSet) Copy as sub cdecl(byref buffer as py2fbTuple)	clear as sub cdecl() <i>Azzera il contenuto dell’oggetto</i>
Ritorna una copia dell’oggetto	
dim myList2 as Py2fbList <i>‘ Creo il descrittore</i> myList.copy(myList2) <i>‘ ora myList2 è una copia myList</i> myList.clear() <i>‘azzero il contenuto di myList</i>	

File pybase.py: Questa classe implementa il magic method `__contains__` per il Freebasic:

```
class contains: # in per i valori usato da tuple,list,set e dict
    def __init__(self,obj,py2fbStruct,tipoInt):
        """ Wrapper al magic method __contains__ """
        self.let=obj
        self.obj=self.let.value
        self.py2fbStruct=py2fbStruct
        if tipoInt:
            ass=CFUNCTYPE(c_bool,c_int,c_void_p) #Liste,Tuple,Set e simili
            self.py2fbStruct.In=cast(ass(self._contains_),POINTER(c_uint))
        else:
            ass=CFUNCTYPE(c_bool,c_wchar_p) #Dizionari
            self.py2fbStruct.In=cast(ass(self._contains2_),POINTER(c_uint))
    def _contains_(self,tipo,ptr): #liste tuple e set
        self.obj=self.let.value #acquisisce il contenuto dell'oggetto
        p=ptr_convert(tipo,ptr) #converte il puntatore nel tipo adeguato
        if tipo in (0,1,2,10,11,12,13,14,15,16,17):
            return p[0] in self.obj
        elif tipo in (3,4):
            return p.value in self.obj
        elif tipo in (5,6,7,8,9):
            return (eval(p.value) in self.obj)
    def _contains2_(self,key): #dizionari
        self.obj=self.let.value
        return key in self.obj
```

Metodo in per valore	Metodo in per le chiavi
in as function cdecl(byval tipo as integer,byval key as any ptr) as Boolean	in as function cdecl(key as wstring ptr) as Boolean
myList.in(tipo,value) ‘ Controlla se il valore è presente	
myDict.in(key) ‘ Controlla se il chiave è presente	

File pybase.py: *Questa classe implementa gli slice del magic method `__getitem__`*

```
class Slice:
    def __init__(self,obj,py2fbStruct,myClass,mem):
        """ Wrapper al slice del magic method __getitem__ """
        self.let=obj
        self.obj=self.let.value
        self.py2fbStruct=py2fbStruct
        self.myClass=myClass
        self.mem=mem
        ass=CFUNCTYPE(None,POINTER(py2fbList),c_int,c_int,c_int)
        self.py2fbStruct.slice=cast(ass(self._slice_),POINTER(c_uint))
    def _slice_(self,buffer,start,end,step):
        self.obj=self.let.value #acquisisce il contenuto dell'oggetto
        t=self.obj[start:end:step] #frazione dell'oggetto
        key=len(self.mem) #nuova chiave dalla lunghezza del dizionario oggetti
        self.mem[key]=self.myClass(t) #Crea un nuovo oggetto
        memmove(buffer,byref(self.mem[key].getObj()),sizeof(self.py2fbStruct)) #copia il descrittore
```

Slice

```
slice as sub cdecl(byref buffer as py2fbTuple,byval start as integer,byval end as integer,byval stop_ as integer)
slice as sub cdecl(byref buffer as py2fbList byval start as integer,byval end as integer,byval stop_ as integer)
```

```
Dim myList2 as Py2fbList
myList.slice(myList2,start,,end,step) ' Crea una sub lista
```

File pybase.py: *Questa classe implementa il metodo newObj per il Freebasic:*

```
class newObj:
    def __init__(self,obj,py2fbStruct,myClass,mytype,ClassPtr,mem):
        """ Wrapper per la creazione di un nuovo oggetto da Freebasic """
        self.obj=obj #let
        self.py2fbStruct=py2fbStruct #struttura di aggancio
        self.myClass=myClass #classe Contains da utilizzare, PyListExport, PyTupleExport, PyDictExport, PySetExport
        self.mytype=mytype #classe Python appropriata alla classe Contains, list tuple, dict, set
        self.mem=mem # dizionario per la persistenza dell'oggetto
        ass=CFUNCTYPE(None,ClassPtr,c_wchar_p)
        self.py2fbStruct.newObj=cast(ass(self._newObj_),POINTER(c_uint)) #aggancia il metodo alla struttura
    def _newObj_(self,buffer,elenco):
        t=eval(elenco) #valuta ls stringa
        if not isinstance(t,self.mytype):t=mytype() # oggetto vuoto se tipo errato
        key=len(self.mem);self.mem[key]=self.myClass(t) #crea l'oggetto
        memmove(buffer,byref(self.mem[key].getObj()),sizeof(self.py2fbStruct)) # copio il descrittore
```

Vediamo un esempio di uso del metodo newObj in Freebasic:

newObj questo metodo permette a Freebasic di creare un nuovo oggetto dello stesso tipo
newObj as sub cdecl(byref buffer as py2fbList,byval lista as wstring ptr)
newObj as sub cdecl(byref buffer as py2fbTuple,byval tupla as wstring ptr)
newObj as sub cdecl(byref buffer as py2fbDict,byval dict as wstring ptr)
newObj as sub cdecl(byref buffer as py2fbSet,byval set as wstring ptr)
Dim myList2 as Py2fbList <i>‘Crea una nuova struttura</i>
myList.newObj(myList2, "[11,13,'Cane']") <i>‘Crea una nuova lista 2 interi una stringa</i>
myList.newObj(myList2, "[2**x for x in range(1,11)]") <i>‘crea una nuova lista usando la list comprehensions</i>
Crea una lista con le potenze di 2 fino ad esponente 10 [2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]

File pybase.py: *Questa classe implementa i magic method __str__ e __repr__ per il Freebasic:*

```

class toString:
    def __init__(self,obj,py2fbStruct,buffer):
        """ Wrapper ai magic metod __str__ e __repr__ """
        self.let=obj #classe let
        self.obj=self.let.value #oggetto in let
        self.py2fbStruct=py2fbStruct #struttura di aggancio
        self.buffer=buffer #buffer bytes e unicode
        s=CFUNCTYPE(None)
        self.py2fbStruct.str=cast(s(self.__str__),POINTER(c_uint)) #aggancia il metodo str
        s=CFUNCTYPE(None)
        self.py2fbStruct.repr=cast(s(self.__repr__),POINTER(c_uint)) #aggancia il metodo repr
    def __str__(self): # Il contenuto dell'oggetto viene ritornato sia come stringa di byte che unicode
        self.obj=self.let.value #ritorna l'oggetto attuale
        v=str(self.obj) #converti in stringa
        self.buffer.strbuff=v #wstring ritorna in wbuff
        self.buffer.bytbuff=v.encode() #zstring ritorna in zbuff
    def __repr__(self):
        self.obj=self.let.value #ritorna l'oggetto attuale
        v=repr(self.obj) #repr dell'oggetto
        self.buffer.strbuff=v #wstring ritorna in wbuff
        self.buffer.bytbuff=v.encode() # ritorna in zbuff

```

Vediamo un esempio l'uso del metodo Str in Freebasic:

Str e repr
str as sub cdecl()
repr as sub cdecl()
myList.str() <i>‘contenuto della lista in formato wstring e zstring nei buffer</i>
print *myList.wbuff <i>‘stampa il contenuto del buffer in formato unicode</i>
print *myList.zbuff <i>‘stampa il contenuto del buffer in formato bytes</i>
<i>Python automaticamente li salva in entrambi i buffer</i>

Per gli oggetti convenzionali non c'è nessuna differenza tra str e repr, ma in altri oggetti ad esempio datetime la cosa cambia repr restituisce un formato compatibile con la funzione eval di Python.

File pybase.py: Questa classe implementa il metodo Del per il Freebasic:

```
class Del:
    def __init__(self,mem,py2fbStruct):
        """ Wrapper al metodo per la cancellazione dell'oggetto Python libera la memoria,
            e permette di riutilizzare un descrittore Freebasic assegnato ad un oggetto precedente"""
        self.py2fbStruct=py2fbStruct #struttura di aggancio
        d=CFUNCTYPE(None,c_int)
        self.py2fbStruct.Del=cast(d(self._Del_),POINTER(c_uint))
        self.mem=mem #dizionario per la persistenza dell'oggetto
    def _Del_(self,index): #non è permesso cancellare l'oggetto principale mandato da Python
        if index>=0: del(self.mem[index]) #oggetto principale index= -1
```

Vediamo un esempio di uso del metodo del in Freebasic:

Del questo metodo permette a Freebasic di eliminare l'oggetto Python

del as sub cdecl(byval t as integer) 'list

myList.del(myList.id) 'Elimina l'oggetto con identificativo id contenuto nella struttura

File pybase.py: Questa classe implementa il metodo len per il Freebasic:

```
class Len:
    def __init__(self,obj,py2fbStruct):
        """ Wrapper al magic metod __len__ """
        self.let=obj
        self.obj=self.let.value
        self.py2fbStruct=py2fbStruct
        d=CFUNCTYPE(c_int)
        self.py2fbStruct.len=cast(d(self._len_),POINTER(c_uint))
    def _len_(self):
        self.obj=self.let.value #legge l'oggetto
        return len(self.obj) #ritorna il numero di elementi
```

Vediamo un esempio di uso del metodo newObj in Freebasic:

len questo ritorna il numero di elementi dell'oggetto

len as sub cdecl(byval t as integer) 'list

myList.len()

File pybase.py: Questa classe implementa i metodi count e index per il Freebasic:

```
class count_index:
    def __init__(self,obj,py2fbStruct):
        """ Wrapper ai metodi count e index """
        self.let=obj
        self.obj=self.let.value
        self.py2fbStruct=py2fbStruct
        index=CFUNCTYPE(c_int,c_int,c_void_p,c_int,c_int)
        self.py2fbStruct.index=cast(index(self._index_),POINTER(c_uint))
        count=CFUNCTYPE(c_int,c_int,c_void_p)
        self.py2fbStruct.count=cast(count(self._count_),POINTER(c_uint))

    def _index_(self,tipo,ptr,start,stop):
        self.obj=self.let.value # acquisisce il contenuto dell'oggetto
        p=ptr_convert(tipo,ptr) # converte il puntatore
        if tipo in (0,1,2,10,11,12,13,14,15,16,17): #tipi numerici
            return self.obj.index(p[0],start,stop)
        elif tipo in (3,4): #stringhe di bytes e unicode
            return self.obj.index(p.value,start,stop)
        elif tipo in (5,6,7,8,9): #oggetti
            return self.obj.index(eval(p.value),start,stop)

    def _count_(self,tipo,ptr):
        self.obj=self.let.value
        p=ptr_convert(tipo,ptr)
        if tipo in (0,1,2,10,11,12,13,14,15,16,17):
            p=cast(ptr,POINTER(c_float)) #cast puntatore
            return self.obj.count(p[0])
        elif tipo in (3,4):
            return self.obj.count(p.value)
        elif tipo in (5,6,7,8,9):
            return self.obj.count(eval(p.value))
```

Vediamo un esempio di uso dei metodi count e index in Freebasic:

Count conta il numero di occorrenze	Index ritorna la posizione di un valore
count as function cdecl(byval tipo as integer, byval pt as any ptr) as integer	index as function cdecl(byval tipo as integer,byval pt as any ptr,byval start as integer,byval stop_ as integer) as integer
<pre>dim as long i=4 myTuple.let(("1,2,3,4,5,4,3")) ' assegno un nuovo valore ad una tupla print "il numero 4 si ripete per " & myTuple.count(pyLong,@i) & " volte" ' stampa il numero 4 si ripete per 2 volte ' ----- Dim l as long=myTuple.len() Print "il primo 4 si trova nella posizione " myTuple.index(pyLong,@i,0,l) ' stampa il primo 4 che trova, posizione 3</pre>	

File pybase.py: Questa classe implementa i buffer dati wbuff e zbuff

```
class buffer:
    def __init__(self,obj,py2fbStruct):
        """ Wrapper per creare i buffer di memoria str e bytes """
        self.let=obj
        self.obj=self.let.value
        self.py2fbStruct=py2fbStruct
        self.pystrb=pyStrBuffer() # wstring buffer
        self.pybytb=pyBytesBuffer() # zstring buffer
        p=self.pybytb.pBuffer
        self.py2fbStruct.zbuff=cast(p,POINTER(c_uint)) #aggancia il Bytes buffer
        p2=self.pystrb.pBuffer
        self.py2fbStruct.wbuff=cast(p2,POINTER(c_uint)) #aggancia l'unicode buffer
    @property
    def strbuff(self): # ritorna il valore del wstring buffer
        return self.pystrb.buffer
    @strbuff.setter
    def strbuff(self,value): # setta un valore nel wstring buffer
        self.pystrb.buffer=value
    @property
    def bytbuff(self): # ritorna il valore del zstring buffer
        return self.pybytb.buffer
    @bytbuff.setter
    def bytbuff(self,value): # setta un valore nel zstring buffer
        self.pybytb.buffer=value
```

In questi due buffer transita la maggior parte dei dati da Python a Freebasic, stringhe di bytes e unicode, liste tuple, dizionari, set e complex e qualsiasi altro oggetto non numerico passa in formato repl passa da qui.

Ne fanno uso in Freebasic str, repr e getitem.

```
Dim myList2 as Py2fbList ' nuovo descrittore
myList.let("[3,[2,45,78],'Cane']") ' nuovo assegnamento alla lista principale
Dim tmp as wstring ptr
myList.getitem(PyList, 1) ' mette una copia della lista in formato repl nel buffer
tmp=*myList.wbuff ' salva il contenuto del buffer
myList.newObj(myList2, tmp) ' nuova lista creata con i valori di quella precedente
```

File list_method.py: *Questo modulo implementa i metodi che valgono esclusivamente per le liste.*

```
from ctypes import *
from fblib.PyObj.PyStruct import *
from fblib.PyObj.pybase import ptr_convert
class list_method:
    def __init__(self,obj,py2fbStruct,Buffer,myClass):
        self.let=obj
        self.obj=self.let.value
        self.buffer=Buffer          # wstring buffer e zstring buffer
        self.py2fbStruct=py2fbStruct #struttura di aggancio
        self.myClass=myClass #classe PyListExport
        #-----sort-----
        sort=CFUNCTYPE(None,c_bool) #Prototype
        self.py2fbStruct.sort=cast(sort(self._sort_),POINTER(c_uint)) #aggancia i wrapper alla struttura
        #-----Append-----
        append=CFUNCTYPE(None,c_int,c_void_p)
        self.py2fbStruct.append=cast(append(self._append_),POINTER(c_uint))
        #-----pop-----
        pop=CFUNCTYPE(None,c_int,c_int,c_void_p)
        self.py2fbStruct.pop=cast(pop(self._pop_),POINTER(c_uint))
        #-----Remove-----
        remove=CFUNCTYPE(None,c_int,c_void_p)
        self.py2fbStruct.remove=cast(remove(self._remove_),POINTER(c_uint))
        #-----Insert-----
        insert=CFUNCTYPE(None,c_int,c_int,c_void_p)
        self.py2fbStruct.insert=cast(insert(self._insert_),POINTER(c_uint))
        #-----Reverse-----
        reverse=CFUNCTYPE(None)
        self.py2fbStruct.reverse=cast(reverse(self._reverse_),POINTER(c_uint))
    def _sort_(self,reverse): # sort as sub cdecl(byval reverse as Boolean)
        self.obj=self.let.value
        self.obj.sort(reverse=reverse)
    def _append_(self,tipo,ptr): # append as sub cdecl(byval tipo as integer,byval pt as any ptr)
        self.obj=self.let.value
        p=ptr_convert(tipo,ptr)
        if tipo in (0,1,2,10,11,12,13,14,15,16,17):
            self.obj.append(p[0])
        elif tipo in (3,4):
            self.obj.append(p.value)
        elif tipo in (5,6,7,8,9):
            self.obj.append(eval(p.value))
    def _pop_(self,tipo,index,ptr): # pop as sub cdecl(byval tipo as integer,byval index as integer,byval pt as any ptr)
        self.obj=self.let.value
        p=ptr_convert(tipo,ptr)
        if tipo in (0,1,2,10,11,12,13,14,15,16,17):
            p[0]=self.obj.pop(index)
        elif tipo in (3,4):
            self.buffer.strbuff=self.obj.pop(index)
            self.buffer.bytbuff=self.obj.pop(index)
        elif tipo in (5,6,7,8,9):
            self.buffer.strbuff=repr(self.obj.pop(index))
            self.buffer.bytbuff=repr(self.obj.pop(index))
    def _remove_(self,tipo,ptr): # remove as sub cdecl(byval tipo as integer,byval pt as any ptr)
        self.obj=self.let.value
```

```

p=ptr_convert(tipo,ptr)
if tipo in (0,1,2,10,11,12,13,14,15,16,17):
    self.obj.remove(p[0])
elif tipo in (3,4):
    self.obj.remove(p.value)
elif tipo in (5,6,7,8,9):
    self.obj.remove(eval(p.value))
def _insert_(self,tipo,index,ptr):#insert as sub cdecl(byval tipo as integer,byval index as integer,byval pt as any ptr)
    self.obj=self.let.value
    p=ptr_convert(tipo,ptr)
    if tipo in (0,1,2,10,11,12,13,14,15,16,17):
        self.obj.insert(index,p[0])
    elif tipo in (3,4):
        self.obj.insert(index,p.value)
    elif tipo in (5,6,7,8,9):
        self.obj.insert(index,eval(p.value))
def _reverse_(self):
    self.obj=self.let.value
    self.obj.reverse()

```

Esempi:

```

myList.sort(false)  'false sort ascendente true discendente
myList.reverse()    'inverte l'ordine della lista
f=2345.9921:myList.append(pyFloat, @f) 'inserimento di un float
myList.remove(PyInt, 2, @i) 'rimuove un intero su l'indice 2
myList.pop(PyInt, 0, @i) 'elimina un intero dalla posizione 0
dim as Uinteger ui=32899:myList.insert(PyUInt, 4, @ui) 'inserisce il nuovo dato nella posizione 4

```

File dict_method.py: Questo modulo implementa i metodi che valgono esclusivamente per i dizionari.

```

# metodi dizionario
from ctypes import *
from fblib.PyObj.pybase import *
from fblib.PyObj. PyStruct import *
from fblib.PyObj.pyList import *
class dict_method:
def __init__(self,obj,py2fbStruct,Buffer,myClass):
    self.let=obj
    self.obj=self.let.value
    self.buffer=Buffer # wstring e zstring buffer
    self.py2fbStruct=py2fbStruct
    self.myClass=myClass #classe PyDictExport
    self.mem=[] # utilizzata per la persistenza delle liste create dai metodi values, keys, items
#-----pop-----
    pop=CFUNCTYPE(None,c_int,c_wchar_p,c_void_p,c_void_p)
    self.py2fbStruct.pop=cast(pop(self._pop_),POINTER(c_uint))
#-----popitem-----
    popitem=CFUNCTYPE(None)
    self.py2fbStruct.popitem=cast(popitem(self._popitem_),POINTER(c_uint))
#-----get-----
    get=CFUNCTYPE(None,c_int,c_wchar_p,c_void_p,c_void_p)

```

```

self.py2fbStruct.get=cast(get(self._get_),POINTER(c_uint))

#-----setdefault-----
setdefault=CFUNCTYPE(None,c_wchar_p,c_int,c_void_p,c_void_p)
self.py2fbStruct.setdefault=cast(setdefault(self._setdefault_),POINTER(c_uint))

#-----update-----
update=CFUNCTYPE(None,c_wchar_p)
self.py2fbStruct.update=cast(update(self._update_),POINTER(c_uint))

#-----keys-----
keys=CFUNCTYPE(None,POINTER(py2fbList))
self.py2fbStruct.keys=cast(keys(self._keys_),POINTER(c_uint))

#-----values-----
values=CFUNCTYPE(None,POINTER(py2fbList))
self.py2fbStruct.values=cast(values(self._values_),POINTER(c_uint))

#-----items-----
items=CFUNCTYPE(None,POINTER(py2fbList))
self.py2fbStruct.items=cast(items(self._items_),POINTER(c_uint))

#-----Clear-----
clear=CFUNCTYPE(None)
self.py2fbStruct.clear=cast(clear(self.obj.clear),POINTER(c_uint))

#-----fromkeys-----
fromkeys=CFUNCTYPE(None,POINTER(py2fbDict),c_wchar_p)
self.py2fbStruct.fromkeys=cast(fromkeys(self._fromkeys_),POINTER(c_uint))

#-----
def _pop_(self,tipo,key,default,ptr):
    """elimina la chiave se presente e ritorna il valore altrimenti ritorna default"""
    self.obj=self.let.value
    p=ptr_convert(tipo,ptr)
    pdef=ptr_convert(tipo,default)
    if tipo in (0,1,2,10,11,12,13,14,15,16,17):
        p[0]=self.obj.pop(key,pdef[0])
    elif tipo in (3,4):
        v=self.obj.pop(key,pdef.value)
        self.buffer.strbuff=v # wstring ritorna in wbuff
        self.buffer.bytbuff=v.encode() # wstring ritorna in wbuff
    elif tipo in (5,6,7,8,9):
        v=repr(self.obj.pop(key,pdef.value))
        self.buffer.strbuff=v # wstring ritorna in wbuff
        self.buffer.bytbuff=v.encode() # wstring ritorna in wbuff

def _popitem_(self):
    """elimina una chiave ritorna una tupla (chiave,valore in formato wstring e zstring)"""
    self.obj=self.let.value
    t=self.obj.popitem()
    v=repr(t)
    self.buffer.strbuff=v # wstring ritorna in wbuff
    self.buffer.bytbuff=v.encode() # zstring ritorna in zbuff

def _get_(self,tipo,key,default,ptr):
    """ritorna il valore della chiave se esiste altrimenti ritorna default"""
    self.obj=self.let.value
    p=ptr_convert(tipo,ptr)
    pdef=ptr_convert(tipo,default)
    if tipo in (0,1,2,10,11,12,13,14,15,16,17):
        p[0]=self.obj.get(key,pdef[0])

```

```

elif tipo in (3,4):
    v=self.obj.get(key,pdef.value)
    self.buffer.strbuff=v # wstring ritorna in wbuff
    self.buffer.bytbuff=v.encode() # zstring ritorna in zbuff
elif tipo in (5,6,7,8,9):
    v=repr(self.obj.get(key,pdef.value))
    self.buffer.strbuff=v # wstring ritorna in wbuff
    self.buffer.bytbuff=v.encode() # zstring ritorna in zbuff
def _setdefault_(self,key,tipo,default,ptr): # inserisce una chiave e valore, ritorna default se la chiave non esiste
    self.obj=self.let.value
    p=ptr_convert(tipo,ptr)
    pdef=ptr_convert(tipo,default)
    if tipo in (0,1,2,10,11,12,13,14,15,16,17):
        p[0]=self.obj.setdefault(key,pdef[0])
    elif tipo in (3,4):
        v=self.obj.setdefault(key,pdef.value)
        self.buffer.strbuff=v # wstring ritorna in wbuff
        self.buffer.bytbuff=v.encode() # zstring ritorna in zbuff
    elif tipo in (5,6,7,8,9):
        v=repr(self.obj.setdefault(key,eval(pdef.value)))
        self.buffer.strbuff=v # wstring ritorna in wbuff
        self.buffer.bytbuff=v.encode() # zstring ritorna in zbuff
def _update_(self,iteratore): #update chiavi e valori
    self.obj=self.let.value
    t=eval(iteratore)
    self.obj.update(t)
def _fromkeys_(self,buffer,iteratore): #crea un nuovo dizionario da un interable
    self.obj=self.let.value
    t=eval(iteratore)
    d=self.obj.fromkeys(*t)
    self.mem.append(self.myClass(d)) # creo un nuovo oggetto
    memmove(buffer,byref(self.mem[-1].getObj()),sizeof(py2fbDict))#copio il descrittore
def _keys_(self,buffer):
    self.obj=self.let.value
    t=list(self.obj.keys())
    self.mem.append(PyListExport(t)) # creo un nuovo oggetto
    memmove(buffer,byref(self.mem[-1].getObj()),sizeof(py2fbList))#copio il descrittore
def _values_(self,buffer):
    self.obj=self.let.value
    t=list(self.obj.values())
    self.mem.append(PyListExport(t)) # creo un nuovo oggetto
    memmove(buffer,byref(self.mem[-1].getObj()),sizeof(py2fbList))#copio il descrittore
def _items_(self,buffer):
    self.obj=self.let.value
    t=list(self.obj.items())
    self.mem.append(PyListExport(t)) # creo un nuovo oggetto
    memmove(buffer,byref(self.mem[-1].getObj()),sizeof(py2fbList))#copio il descrittore
def _contains_(self,key): # versione per dizionari
    self.obj=self.let.value
    return key in self.obj

```

Esempi su i metodi dizionario

```
print "-----get test-----"
Dim as long iv=0:i=-888
myDict.get(pyInt,"numero",@i,@iv)
print "Valore da get per chiave numerica esistente "&iv
myDict.get(pyInt,"numero2",@i,@iv)
print "Valore da get per chiave numerica inesistente "&iv
Dim tmp as wstring *50 tmp=>"999"
myDict.get(pyStr,"Cane",@tmp,None)
print "Valore da get per chiave di stringa esistente "&*myDict.wbuff
tmp="Gigio":myDict.get(pyStr,"Topo",@tmp,None)
print "Valore da get per chiave di stringa inesistente "&*myDict.wbuff
print "-----pop test-----"
tmp="Pluto":myDict.pop(pyStr,"Cane",@tmp,None)
print "Elimina una chiave con pop ritorna il suo valore "&*myDict.wbuff
print "-----popitem test-----"
print "Elimina una coppia chiave-valore utilizzando popitem"
myDict.popitem():tmp=*myDict.wbuff
print "risultato una tupla in formato wstring "&*tmp ' stampa la coppia estratta
dim myTuple2 as py2fbTuple:myTuple.newObj(@tmp) ' Creo una nuova tuple con la coppia chiave valore
```

```
print "-----update test-----"
print "Update del dizionario"
myDict.update({"Cane':'Mastino','Topo':'Gigio','Città':'Roma'})
myDict.str(): print "dizionario update "&*myDict.wbuff
print
print "-----setdefault test-----"
print "setdefault esempio " :iv=20
myDict.setdefault("Eta",pyInt,@iv,@i)
print "valore ritornato "&i
myDict.str():print "voce update "&*myDict.wbuff
print
print "-----fromkeys test-----"
print "Creo un nuovo dizionario con fromkeys assegnando lo stesso valore di default"
dim NewDict as py2fbDict
myDict.fromkeys(NewDict,("Pane','Latte','Uova'),5")
NewDict.str()
print "dizionario creato "&*NewDict.wbuff
print
print "-----copy test-----"
dim NewDict2 as py2fbDict
myDict.copy(NewDict2)
NewDict2.str()
print "copia del dizionario creata "&*NewDict2.wbuff
print "-----in test-----"
print wstr("Controllo se una chiave è presente nel dizionario")
print wstr("Pane è presente ") & newDict.in("Pane")
print "-----keys test-----"
Dim as py2fbList KeysList,ValuesList,ItemsList
NewDict2.keys(KeysList) ' lista contenente le chiavi del dizionario
NewDict2.values(ValuesList) ' lista contenente i valori del dizionario
NewDict2.items(ItemsList) ' lista contenente la coppia chiavi valori del dizionario
```

Le chiavi dei dizionari in Python possono essere di un qualsiasi tipo hashable stringhe di bytes e unicode, tuple, interi, float e numeri complessi, la versione per Freebasic accetta solo stringhe unicode, il tipo maggiormente utilizzato.

File set_method.py

```
from ctypes import *
from fblib.PyObj. PyStruct import *
from fblib. PyObj. pybase import ptr_convert
class set_method:
    def __init__(self,obj,py2fbStruct,Buffer,myClass,mytype,ClassPtr):
        self.let=obj
        self.obj=self.let.value
        self.buffer=Buffer # wstring buffer e zstring buffer
        self.py2fbStruct=py2fbStruct
        self.myClass=myClass #PySetExport
        self.mytype=mytype #set
        self.ClassPtr=ClassPtr #puntatore alla struttura
        self.mem=[] #lista persistenza dei set creati dal freebasic
        #-----pop-----
        pop=CFUNCTYPE(None,c_int,c_int,c_void_p)
        self.py2fbStruct.pop=cast(pop(self._pop_),POINTER(c_uint))
        #-----add-----
        add=CFUNCTYPE(None,c_int,c_void_p)
        self.py2fbStruct.add=cast(add(self._add_),POINTER(c_uint))
        #-----discard-----
        discard=CFUNCTYPE(None,c_int,c_void_p)
        self.py2fbStruct.discard=cast(discard(self._discard_),POINTER(c_uint))
        #-----remove-----
        remove=CFUNCTYPE(None,c_int,c_void_p)
        self.py2fbStruct.remove=cast(remove(self._remove_),POINTER(c_uint))
        #-----union-----
        union=CFUNCTYPE(None,POINTER(py2fbSet),c_wchar_p)
        self.py2fbStruct.union=cast(union(self._union_),POINTER(c_uint))
        #-----difference-----
        difference=CFUNCTYPE(None,POINTER(py2fbSet),c_wchar_p)
        self.py2fbStruct.difference=cast(difference(self._difference_),POINTER(c_uint))
        #-----intersection-----
        intersection=CFUNCTYPE(None,POINTER(py2fbSet),c_wchar_p)
        self.py2fbStruct.intersection=cast(intersection(self._intersection_),POINTER(c_uint))
        #-----symmetric_difference-----
        symmetric_difference=CFUNCTYPE(None,POINTER(py2fbSet),c_wchar_p)
        self.py2fbStruct.symmetric_difference=cast(symmetric_difference(self._symmetric_difference_),\
        POINTER(c_uint))
        #-----isubset-----
        isubset=CFUNCTYPE(c_bool,c_wchar_p)
        self.py2fbStruct.isubset=cast(isubset(self._isubset_),POINTER(c_uint))
        #-----issuperset-----
        issuperset=CFUNCTYPE(c_bool,c_wchar_p)
        self.py2fbStruct.issuperset=cast(issuperset(self._issuperset_),POINTER(c_uint))
        #-----isdisjoint-----
        isdisjoint=CFUNCTYPE(c_bool,c_wchar_p)
        self.py2fbStruct.isdisjoint=cast(isdisjoint(self._isdisjoint_),POINTER(c_uint))
```



```

#-----symmetric_difference_update-----
symmetric_difference_update=CFUNCTYPE(None,c_wchar_p)
self.py2fbStruct.symmetric_difference_update=\
cast(symmetric_difference_update(self._symmetric_difference_update_),POINTER(c_uint))
#-----symmetric_update-----
symmetric_update=CFUNCTYPE(None,c_wchar_p)
self.py2fbStruct.symmetric_update=cast(symmetric_update(self._symmetric_update_),POINTER(c_uint))
#-----update-----
update=CFUNCTYPE(None,c_wchar_p)
self.py2fbStruct.update=cast(update(self._update_),POINTER(c_uint))
def _pop_(self,tipo,ptr):
    self.obj=self.let.value
    p=ptr_convert(tipo,ptr)
    if tipo in (0,1,2,10,11,12,13,14,15,16,17):
        p[0]=self.obj.pop(index)
    elif tipo in (3,4):
        self.buffer.strbuff=self.obj.pop(index)
        self.buffer.bytbuff=self.obj.pop(index)
    elif tipo in (5,6,7,8,9):
        self.buffer.strbuff=repr(self.obj.pop(index))
        self.buffer.bytbuff=repr(self.obj.pop(index))
def _add_(self,tipo,ptr):
    self.obj=self.let.value
    p=ptr_convert(tipo,ptr)
    if tipo in (0,1,2,10,11,12,13,14,15,16,17):
        self.obj.add(p[0])
    elif tipo in (3,4):
        self.obj.add(p.value)
    elif tipo in (5,6,7,8,9):
        self.obj.add(eval(p.value))
def _discard_(self,tipo,ptr):
    self.obj=self.let.value
    p=ptr_convert(tipo,ptr)
    if tipo in (0,1,2,10,11,12,13,14,15,16,17):
        self.obj.discard(p[0])
    elif tipo in (3,4):
        self.obj.discard(p.value)
    elif tipo in (5,6,7,8,9):
        self.obj.discard(eval(p.value))
def _remove_(self,tipo,ptr):
    self.obj=self.let.value
    p=ptr_convert(tipo,ptr)
    # Controlla prima se l'elemento è presente per evitare errori
    if tipo in (0,1,2,10,11,12,13,14,15,16,17):
        if not p[0] in self.obj: return
        t=p[0]
    elif tipo in (3,4):
        if not p.value in self.obj: return
        t=p.value
    elif tipo in (5,6,7,8,9):
        t=eval(p.value)
        if not t in self.obj: return
    self.obj.remove(t)
def _union_(self,buffer,ptr):

```



```

self.obj=self.let.value
t=eval(ptr) #valuta ls stringa
t2=self.obj.union(t)
self.mem.append(self.myClass(t2))
memmove(buffer,byref(self.mem[-1].getObj),sizeof(self.py2fbStruct)) # copio il descrittore
def _difference_(self,buffer,ptr):
    self.obj=self.let.value
    t=eval(ptr) #valuta la stringa
    t2=self.obj.difference(t)
    self.mem.append(self.myClass(t2))
    memmove(buffer,byref(self.mem[-1].getObj),sizeof(self.py2fbStruct)) # copio il descrittore
def _intersection_(self,buffer,ptr):
    self.obj=self.let.value
    t=eval(ptr) #valuta ls stringa
    t2=self.obj.intersection(t)
    self.mem.append(self.myClass(t2))
    memmove(buffer,byref(self.mem[-1].getObj),sizeof(self.py2fbStruct)) # copio il descrittore
def _symmetric_difference_(self,buffer,ptr):
    self.obj=self.let.value
    t=eval(ptr) #valuta ls stringa
    t2=self.obj.symmetric_difference(t)
    self.mem.append(self.myClass(t2))
    memmove(buffer,byref(self.mem[-1].getObj),sizeof(self.py2fbStruct)) # copio il descrittore
def _isubset_(self,ptr):
    self.obj=self.let.value
    s=eval(ptr)
    return self.obj.isubset(s)
def _issuperset_(self,ptr):
    self.obj=self.let.value
    s=eval(ptr)
    return self.obj.issuperset(s)
def _isdisjoint_(self,ptr):
    self.obj=self.let.value
    s=eval(ptr)
    return self.obj.isdisjoint(s)
def _symmetric_difference_update_(self,ptr):
    self.obj=self.let.value
    s=eval(ptr)
    self.obj.symmetric_difference_update(s)
def _symmetric_update_(self,ptr):
    self.obj=self.let.value
    s=eval(ptr)
    self.obj.symmetric_update(s)
def _update_(self,ptr):
    self.obj=self.let.value
    s=eval(ptr)
    self.obj.update(s)

```

Un esempio completo su i set (vedremo dopo le classi ambiente PySetExport, PyTuple, PyList, PyDict)

```

from PyObj.pyTuple import *
from PyObj.pySet import *
from PyObj.PyStruct import *
PATHLIB=r'.\esempio_set_1.dll'

```

Autore: Marco Salvati
Licenza Tutorial: CC BY 3.0

Email: marcosalvati61@gmail.com
Licenza Software: GPL V.3

```

s1={1,2,4} # crea un set
pte=PySetExport(s1) #oggetto set condiviso con freebasic
PATHLIB=r'.\esempio_set_1.dll'
lib=CDLL(PATHLIB)
pyt=pte.getObj # ricevi il descrittore
lib.init.argtypes=[POINTER(py2fbSet)]
lib.init(pyt)
lib.test()
print("\n\nControllo l'inserimento da Python")
print(s1)

```

Esempio_set_1.bas

```

declare sub init cdecl alias "init" (byref s as py2fbSet)
declare sub test cdecl alias "test"()
dim shared myset as py2fbSet
sub init cdecl alias "init" (byref s as py2fbSet) export
    mySet=s
end sub
sub test cdecl alias "test"() export
    print "-----Len e Str test-----"
    print "Numero di elementi nel set originale "&mySet.len()
    mySet.str():print *mySet.Wbuff :print
    dim i as long,f as Single,w as wstring*100
    print "-----add test-----"
    print "Aggiungo al set un intero un float e una stringa unicode e una tupla"
    i=777:f=6.75:w="By By"
    mySet.add(PyInt,@i)
    mySet.add(PyFloat,@f)
    mySet.add(PyStr,@w)
    w="(1,2,3,'cane',5.5)":mySet.add(PyTuple,@w)
    print "Numero di elementi nel set "&mySet.len()
    mySet.str():print *mySet.Wbuff
    print "-----In test-----"
    print "Controllo la presenza di un elemento":i=777
    print wstr("Valore 777 è presente ")&mySet.in(PyInt,@i)
    print "Controllo la presenza di un elemento":f=88.7
    print wstr("Valore 88.7 è presente ")&mySet.in(PyFloat,@f)
    print "Controllo la presenza di un elemento":w="(1,2,3,'cane',5.5)"
    print wstr("Tupla (1,2,3,'cane',5.5) è presente ")&mySet.in(PyTuple,@w)
    print "-----discard remove test-----"
    print "-----Elimino il valore 1 dal set con remove-----"
    i=1:mySet.remove(PyInt,@i)
    print "-----Elimino il valore 'By By' dal set con discard-----"
    w="By By":mySet.discard(PyStr,@w)
    print "Numero di elementi nel set "&mySet.len()
    mySet.str():print *mySet.Wbuff
    print "-----union test-----"
    dim mySet2 as py2fbSet
    print "unione con l'insieme {3,6.75,8.8,(7+7.4j),'By By'}"
    mySet.union(myset2,"{3,6.75,8.8,(7+7.4j),'By By'}")

```

```

mySet2.str():print:print *mySet2.Wbuff
dim mySet3 as py2fbSet
print "-----difference test-----"
print "difference con l'insieme {3,6.75,8.8,(7+7.4j),'By By'}"
mySet.difference(myset3,"{3,6.75,8.8,(7+7.4j),'By By'}")
mySet3.str():print *mySet3.Wbuff
dim mySet4 as py2fbSet
print "-----intersection test-----"
print "intersezione con l'insieme {3,6.75,8.8,(7+7.4j),'By By'}"
mySet.intersection(myset4,"{3,6.75,8.8,(7+7.4j),'By By'}")
mySet4.str():print *mySet4.Wbuff
dim mySet5 as py2fbSet
print "-----symmetric difference test-----"
print "symmetric_difference con l'insieme {3,6.75,8.8,(7+7.4j),'By By'}"
mySet.symmetric_difference(myset5,"{3,6.75,8.8,(7+7.4j),'By By'}")
mySet5.str():print *mySet5.Wbuff
end sub

```

Le classi contenitori dei wrapper

Questa è la classe madre degli ambienti viene ereditata da tutti i contenitori.

File contains.py classe base per gli ambienti virtuali

```

from fbllib.PyObj.pybase import *

class Contains:
    def __init__(self,value,mytype,py2fbStruct,myClass,mem):
        if not isinstance(value,mytype):value=mytype() # validazione del tipo
        self.mytype=mytype # salva il tipo dato
        self._mem=mem # dizionario per la persistenza degli oggetti
        self.py2fbStruct=py2fbStruct() # crea la struttura contenitore
        if len(self._mem)>0: # se il dizionario contiene oggetti
            self.py2fbStruct.id=len(self._mem)-1 # identificativo del nuovo oggetto
        else: # se prima istanza
            self.py2fbStruct.id= -1
        self.myClass=myClass
        self.let=let(value,self.py2fbStruct) # oggetto let
        self.ClassPtr=POINTER(py2fbStruct) # puntatore alla struttura
        self.buffer=buffer(self.let,self.py2fbStruct) # buffer str e bytes

    @property
    def getObj(self): # ritorna l'oggetto Freebasic
        return self.py2fbStruct

    @property
    def value(self): # ritorna il valore attuale dell'oggetto padre
        return self.let.value

    @value.setter
    def value(self,value): # modifica l'oggetto da Python
        self.let.value=value

```

```

def __len__(self):          # numero di elementi creati da Freebasic
    return len(self._mem)
def __getitem__(self, item): # ritorna un oggetto creato da Freebasic
    return self._mem[item]
def keys(self):             #itera sulle le chiavi del dizionario degli oggetti
    return self._mem.keys()
def values(self):           #itera su gli oggetti creati
    return self._mem.values()
def items(self):            #itera sulle chiavi e gli oggetti creati
    return self._mem.items()

```

File *pyTuple.py*: Ambiente per la creazione e persistenza delle tuple

```

# Tuple envelope
from fblib.PyObj.pybase import *
from fblib.PyObj.contains import *
class fblib. PyTupleExport(Contains):
    _mem={ }
    def __init__(self,tupla=()):
        super().__init__(tupla,tuple,py2fbTuple,PyTupleExport,self._mem)
        #-----
        self.is_type =is_type(self.let,self.py2fbStruct,True) # aggancio i wrapper
        self.getitem=getitem(self.let,self.py2fbStruct,self.buffer,True)
        self.count_index=count_index(self.let,self.py2fbStruct)
        self.contains=contains(self.let,self.py2fbStruct)
        self.count_index=count_index(self.let,self.py2fbStruct)
        self.slice=Slice(self.let,self.py2fbStruct,PyTupleExport,self._mem)
        self.toString=toString(self.let,self.py2fbStruct,self.buffer)
        self.len=Len(self.let,self.py2fbStruct)
        self.newObj=newObj(self.let,self.py2fbStruct,PyTupleExport,tuple,self.ClassPtr,self._mem)
        self.Del=Del(self._mem,self.py2fbStruct)

```

File *pyDict.py*: Ambiente per la creazione e persistenza dei dizionari

```
from fblib.PyObj.pybase import *
from fblib.PyObj.contains import *
from fblib.PyObj.dict_method import *

class PyDictExport(Contains):
    _mem={ }
    def __init__(self,Dict={ }):
        super().__init__(Dict,dict,py2fbDict,PyDictExport,self._mem)
        #-----
        self.is_type =is_type(self.let,self.py2fbStruct,False) # aggancio i wrapper
        self.getitem=getitem(self.let,self.py2fbStruct,self.buffer,False)
        self.setitem=setitem(self.let,self.py2fbStruct,False)
        self.delitem=delitem(self.let,self.py2fbStruct,False)
        self.toString=toString(self.let,self.py2fbStruct,self.buffer)
        self.contains=contains(self.let,self.py2fbStruct,False)
        self.len=Len(self.let,self.py2fbStruct)
        self.copy_clear=copy_clear(self.let,self.py2fbStruct,PyDictExport,self.ClassPtr,self._mem)
        self.dict_method=dict_method(self.let,self.py2fbStruct,self.buffer,PyDictExport)
        self.newObj=newObj(self.let,self.py2fbStruct,PyDictExport,dict,self.ClassPtr,self._mem)
        self.Del=Del(self._mem,self.py2fbStruct)
```

File *pySet.py*: Ambiente per la creazione e persistenza dei Set

```
from fblib.PyObj.pybase import *
from fblib.PyObj.contains import *
from fblib.PyObj.set_method import *
class fblib.PyObj.PySetExport(Contains):
    _mem={ }
    def __init__(self,Set=set()):
        super().__init__(Set,set,py2fbSet,PySetExport,self._mem)
        #-----
        self.toString=toString(self.let,self.py2fbStruct,self.buffer) # aggancio i wrapper
        self.len=Len(self.let,self.py2fbStruct)
        self.contains=contains(self.let,self.py2fbStruct,True)
        self.copy_clear=copy_clear(self.let,self.py2fbStruct,PySetExport,self.ClassPtr,self._mem)
        self.set_method=set_method(self.let,self.py2fbStruct,self.buffer,PySetExport,set,self.ClassPtr)
        self.newObj=newObj(self.let,self.py2fbStruct,PySetExport,set,self.ClassPtr,self._mem)
        self.Del=Del(self._mem,self.py2fbStruct)
```

Non c'è bisogno di molte spiegazioni, le classi contenitori raggruppano i vari wrapper che si autoinstallano sulle strutture desiderate creando un descrittore per il Freebasic.

File *pyList.py*: Ambiente per la creazione e persistenza delle Liste

```
from fbllib.PyObj.pybase import *
from fbllib.PyObj.list_method import *
from fbllib.PyObj.contains import *
class fbllib.PyListExport(Contains):
    _mem={ }
    def __init__(self,lista=[]):
        super().__init__(lista,list,py2fbList,PyListExport,self._mem)
        #-----
        self.is_type =is_type(self.let,self.py2fbStruct,True) # aggancio i wrapper
        self.getitem=getitem(self.let,self.py2fbStruct,self.buffer,True)
        self.setitem=setitem(self.let,self.py2fbStruct,True)
        self.delitem=delitem(self.let,self.py2fbStruct,True)
        self.count_index=count_index(self.let,self.py2fbStruct)
        self.contains=contains(self.let,self.py2fbStruct)
        self.slice=Slice(self.let,self.py2fbStruct,PyListExport,self._mem)
        self.toString=toString(self.let,self.py2fbStruct,self.buffer)
        self.len=Len(self.let,self.py2fbStruct)
        self.list_method=list_method(self.let,self.py2fbStruct,self.buffer,PyListExport)
        self.copy_clear=copy_clear(self.let,self.py2fbStruct,PyListExport,self.ClassPtr,self._mem)
        self.newObj=newObj(self.let,self.py2fbStruct,PyListExport,list,self.ClassPtr,self._mem)
        self.Del=Del(self._mem,self.py2fbStruct)
```

File *pyobj_esempio_1.py* Esempio completo delle classi contenitori

```
from fbllib.PyObj.pyTuple import *
from fbllib.PyObj.pyList import *
from fbllib.PyObj.PyStruct import *

l1=[1,3,(5,6.5,'Hei'),[45,67,'Sole nero'],(7+7j)]
t1=(77,99,(55,'cane'),(8+3j),[77,'By By'])
ple=PyListExport(l1)
pte=PyTupleExport(t1)
LIBPATH=r'.\lib\pyobj_esempio_1.dll'
lib=CDLL(LIBPATH) # aggancio la libreria
pyl=ple.getObj # ricevo il descrittore della lista
pyt=pte.getObj # ricevo il descrittore della tupla

lib.init.argtypes=[POINTER(py2fbList),POINTER(py2fbTuple)] # setto i parametri
lib.init(pyl,pyt) # inizializzazione
lib.test() # start
print("\n\nControllo l'inserimento da Python\n")
print(l1)

print("\n\nliste create da Freebasic\n")
for i in ple.values():
    print(i.value)
    print(f"id={i.getObj.id}")

print("\n\ntuple create da Freebasic\n")
for i in pte.values():
    print(i.value)
    print(f"id={i.getObj.id}")
```

File *pyObj_ esempio_1.bas*

```
#include once "pyObj.bi"

dim shared myList as py2fbList, myTuple as py2fbTuple

declare sub init cdecl alias "init" (byref l as py2fbList,byref t as py2fbTuple)
declare sub test cdecl alias "test"()
sub init cdecl alias "init" (byref l as py2fbList,byref t as py2fbTuple) export
    myList=l : myTuple=t
end sub

sub test cdecl alias "test"() export
    print"-----Len e Str test-----"
    print "Numero di elementi nella lista originale "&myList.len()
    myList.str():print *myList.Wbuff
    print
    myTuple.str()
    print "Numero di elementi nella Tupla originale "&myTuple.len()
    print *myTuple.Wbuff
    print
    print"-----Getitem test-----"
    print "Mostro tutti gli elementi della lista"
    dim i as long, tipo as long
    dim b as Boolean, iv as long, f as single
    dim z as zstring ptr, w as wstring ptr
    dim None as any ptr, tmp as wstring *200, tmp2 as zstring ptr
    print "Nella lista ci sono "&myList.len() &" elementi"
    for i=0 to myList.len()-1
        tipo=myList.isType(i)
        select case tipo
            case PyBool ' Boolean
                myList.getitem(tipo,i,@b)
                print i & " Boolean "&b
            case PyInt,pyByte,pyUbyte,pyshort,pyUshort,pyUint ' uno qualsiasi di questi tipi
                myList.getitem(tipo,i,@iv)
                print i & " Integer "&iv
            case PyFloat ' Single
                myList.getitem(tipo,i,@f)
                print i & " Single " &f
            case PyBytes ' Zstring
                myList.getitem(tipo,i,None)
                print i & " Zstring "&*myList.Zbuff
            case PyStr ' Wstring
                myList.getitem(tipo,i,None)
                print i & " Wstring " &*myList.Wbuff
            case PyList ' List creo l'oggetto
                dim myList2 as py2fbList
                myList.getitem(tipo,i,None)
                tmp=*myList.Wbuff
                myList.newObj(myList2,@tmp)
                myList2.str()
                print i & " List " &*myList2.Wbuff
```

```

case PyTuple 'Tuple creo un oggetto
    dim myTuple2 as py2fbTuple
    myList.getItem(tipo,i,None)
    tmp=*myList.Wbuff
    myTuple.newObj(myTuple2,@tmp)
    myTuple2.str()
    print i & " Tuple " & *myTuple2.Wbuff
case PyDict 'Dict
    myList.getItem(tipo,i,None) ' visualizzo solo il contenuto senza creare l'oggetto
    print i & " Dict " & *myList.Wbuff
case PySet 'Set
    myList.getItem(tipo,i,None) ' visualizzo solo il contenuto senza creare l'oggetto
    print i & " Set " & *myList.Wbuff
case PyComplex 'Complex
    myList.getItem(tipo,i,None) ' visualizzo solo il contenuto l'oggetto non è implementato
    print i & " Complex " & *myList.Wbuff
case else
    print i & " tipo non sconosciuto"
end select
next i
print
print "Premi un tasto per continuare"
sleep
print "Nella Tupla ci sono "&myTuple.len() & " elementi"
for i=0 to myTuple.len()-1
    tipo=myTuple.isType(i)
    select case tipo
    case 0 ' Boolean
        myTuple.getItem(tipo,i,@b)
        print i & " Boolean "& b
    case PyInt,pyByte,pyUbyte,pyshort,pyUshort,pyUint ' uno qualsiasi di questi tipi
        myTuple.getItem(tipo,i,@iv)
        print i & " Integer "& iv
    case 2 ' Single
        myTuple.getItem(tipo,i,@f)
        print i & " Single "& f
    case 3 ' Zstring
        myTuple.getItem(tipo,i,None)
        print i & " Zstring "& *myTuple.Zbuff
    case 4 ' Wstring
        myTuple.getItem(tipo,i,None)
        print i & " Wstring " & *myTuple.Wbuff
    case 5 ' List
        dim myList3 as py2fbList
        myTuple.getItem(tipo,i,None) #preleva la lista dalla tuple in format stringa
        tmp=*myTuple.Wbuff # dal buffer
        myList.newObj(myList3,@tmp) # e creo una nuova lista
        myList3.str()
        print i & " List " & *myList3.Wbuff
    case 6 'Tuple
        dim myTuple3 as py2fbTuple
        myTuple.getItem(tipo,i,None)
        tmp=*myTuple.wbuff
        myTuple.newObj(myTuple3,@tmp) #creo una nuova tupla

```



```

    myTuple3.str()
    print i & " Tuple " & *myTuple3.Wbuff #stampo il contenuto della tupla
case 7 'Dict
    myTuple.getitem(tipo,i,None) ' visualizzo solo il contenuto senza creare l'oggetto
    print i & " Dict " & *myList.Wbuff
case 8 'Set
    myTuple.getitem(tipo,i,None) ' visualizzo solo il contenuto l'oggetto non è implementato
    print i & " Set " & *myTuple.Wbuff
case 9 'Complex
    myTuple.getitem(tipo,i,None) ' visualizzo solo il contenuto l'oggetto non è implementato
    print i & " Complex " & *myTuple.Wbuff
case else
    print i & " tipo sconosciuto"
end select
next i
print "premi un tasto":sleep
print "-----Slice test-----"
dim myList4 as py2fbList
myList.Slice(myList4,1,-1,1)
myList4.str()
print "nuova lista " & *myList4.Wbuff
print
dim myTuple4 as py2fbTuple
myTuple.Slice(myTuple4,1,-1,1)
myTuple4.str()
print "nuova Tupla " & *myTuple4.Wbuff
print "test append premi un tasto":sleep
myList.str()
print "prima "& *myList.wbuff
b=false:print "inserisco un bool "& b
myList.append(0,@b)
iv=2345: print "inserisco un intero "&iv
myList.append(1,@iv)
f=2345.9921:print "inserisco un float "&f

myList.append(2,@f)
z=allocate(50)
*z="qwerty":print "inserisco una zstring "& *z
myList.append(3,z)
w=allocate(100)
*w ="ciao ciao":print "inserisco una wstring "& *w
myList.append(4,w)

*w="[2,3,5,rosso]":print "inserisco una lista "& *w
myList.append(5,w)
*w="(22,23,35,verde)": print "inserisco una tupla "& *w
myList.append(6,w)

*w="{ 'A':2,'B':3,'C':5,'D':rosso}": print "inserisco una dizionario "& *w
myList.append(7,w)
*w="{ 'A','B','C','D'}":myList.append(8,w)
print "inserisco una set" &*w
*w="(4.6+7.5j)":myList.append(9,w):print "inserisco un numero complesso "&*w

```

```

myList.append(9,w)
myList.str()
print"premi un tasto per continuare":sleep
print "visualizzo da Freebasic"
print *myList.wbuff
print:myTuple4.str()
print "Elimino l'ultima tupla creata id "&myTuple4.id & " " & *myTuple4.wbuff

myTuple4.del(myTuple4.id)
print "Creo una nuova tupla usando la vecchia etichetta"
myTuple.newObj(myTuple4,"(11,13,17,19)")
myTuple4.str():print "la nuova tupla id "& myTuple4.id & " contiene 4 numeri primi"&*myTuple4.wbuff
end sub

```

Output completo

-----Len e Str test-----

Numero di elementi nella lista originale 5

[1, 3, (5, 6.5, 'Hei'), [45, 67, 'Sole nero'], (7+7j)]

Numero di elementi nella Tupla originale 5

(77, 99, (55, 'cane'), (8+3j), [77, 'By By'])

-----Getitem test-----

Mostro tutti gli elementi della lista

Nella lista ci sono 5 elementi

0 Integer 1

1 Integer 3

2 Tuple (5, 6.5, 'Hei')

3 List [45, 67, 'Sole nero']

4 Complex (7+7j)

Premi un tasto per continuare

Nella Tupla ci sono 5 elementi

0 Integer 77

1 Integer 99

2 Tuple (55, 'cane')

3 Complex (8+3j)

4 List [77, 'By By']

premi un tasto

-----Slice test-----

Nuova lista [3, (5, 6.5, 'Hei'), [45, 67, 'Sole nero']]

Nuova Tupla (99, (55, 'cane'), (8+3j))

Test append premi un tasto

Prima [1, 3, (5, 6.5, 'Hei'), [45, 67, 'Sole nero'], (7+7j)]

Inserisco un bool false

Inserisco un intero 2345

Inserisco un float 2345.992

Inserisco una zstring qwerty

Inserisco una wstring ciao ciao

Inserisco una lista [2,3,5,'rosso']

Autore: Marco Salvati

Licenza Tutorial: CC BY 3.0

Email: marcosalvati61@gmail.com

Licenza Software: GPL V.3

```

Inserisco una tupla (22,23,35,'verde')
Inserisco una dizionario {'A':2,'B':3,'C':5,'D': 'rosso'}
Inserisco una set{'A','B','C','D'}
Inserisco un numero complesso (4.6+7.5j)
Premi un tasto per continuare
Visualizzo da Freebasic
[1, 3, (5, 6.5, 'Hei'), [45, 67, 'Sole nero'], (7+7j), False, 2345, 2345.9921875, 'qwerty', 'ciao ciao', [2, 3, 5,
'rosso'], (22, 23, 35, 'verde'), {'A': 2, 'B': 3, 'C': 5, 'D': 'rosso'}, {'A', 'C', 'B', 'D'}, (4.6+7.5j)]

Elimino l'ultima tupla creata id 1 (99, (55, 'cane'), (8+3j))
Creo una nuova tupla usando la vecchia etichetta
la nuova tupla id 1 contiene 4 numeri primi (11, 13, 17, 19)

Controllo l'inserimento da Python
[1, 3, (5, 6.5, 'Hei'), [45, 67, 'Sole nero'], (7+7j), False, 2345, 2345.9921875, b'qwerty', 'ciao ciao', [2, 3, 5,
'rosso'], (22, 23, 35, 'verde'), {'A': 2, 'B': 3, 'C': 5, 'D': 'rosso'}, {'A', 'C', 'B', 'D'}, (4.6+7.5j)]

Liste create da Freebasic

[45, 67, 'Sole nero']
id= -1
[77, 'By By']
id=0
[3, (5, 6.5, 'Hei'), [45, 67, 'Sole nero']]
id=1

tuple create da Freebasic

(5, 6.5, 'Hei')
id= -1
(11, 13, 17, 19)
id=1

```

Come si può notare la classe contenitore include l'oggetto Python, restituisce un suo descrittore e funge da ambiente virtuale per Freebasic.
Al contrario delle Python/C Api che ci costringono a adeguare il nostro codice al loro protocollo, i descrittori di classe permettono al Freebasic di lavorare in modo modo naturale senza alcun wrapper, semplificando la creazione di codice.
Inoltre l'esiguo header utilizzato da Freebasic può essere facilmente convertito per qualsiasi linguaggio compilato, che supporti strutture, puntatori a funzioni e il passaggio di parametri stile C.

Continua in appendice

Un mini framework per Console

- Passo 01: Utilizziamo Freebasic per creare le primitive per la gestione video da console.
- Passo 02: Creazione di un wrapper per l'utilizzo delle primitive
- Passo 03: Creazione della classe WinBase, base per la gestione delle finestre
- Passo 04: Creazione classe Screen derivata Winbase
- Passo 05: Creazione della classe Window derivata da WinBase
- Passo 06: Creazione della classe Write scrittura in una finestra
- Passo 07: Creazione della classe Draw disegno di box nella finestra
- Passo 08: Creazione della classe Shadow ombra di una finestra o di un box al suo interno
- Passo 09: Creazione della classe Widgets
- Passo 10: Il widget Label
- Passo 11: Il widget Button
- Passo 12: Il widget Entry
- Passo 13: La classe Medit gestore di multi edit
- Passo 14: La classe Event gestore di eventi mouse e tastiera
- Passo 15: La classe Vmenu, crea dei menu verticali
- Passo 16: La classe Hmenu, crea dei menu orizzontali

Per le primitive passo 01, utilizzerò due file uno di dichiarazione delle funzioni e le strutture utilizzate che chiamerò fbConsole.bi, e uno con i sorgenti delle funzioni e commenti per documentarle che chiamerò fbConsole.bas.

Per il passo 02, avremo 3 file constant.py per le dichiarazioni delle costanti, e fbTypes.py, già utilizzato nella prima parte di questo tutorial, per le definizioni di tipi e per le strutture utilizzate i file sono tutti adeguatamente commentati.

Il file della libreria Freebasic, i file di importazione Python, e il wrapper sono piuttosto lunghi, ma facilmente comprensibili. Quindi dopo la presentazione, prima di passare al Passo 3, eseguiremo dei test dimostrativi.

File fbConsole.bi : file di definizione dati e dichiarazioni delle funzioni Passo 01

type Video *'Struttura della memoria video*

 attr as Long *'attributo di colore*

 char as Long *'carattere*

end type

type Area *'descrittore del puntatore del buffer di memoria video*

 id as ubyte *'identifica la Struttura se contiene un puntatore valido id=65 valido*

 height as Long *'altezza dell'area*

 width as Long *'larghezza dell'area*

 buff as Video ptr *'puntatore all'area della memoria video salvata*

end type

'Prototype delle funzioni e delle subroutine in fbConsole.bas

Declare Function fbGetMouse cdecl alias "fbGetMouse"(ByRef x As long, ByRef y As long, ByRef wheel As long, ByRef buttons As long , ByRef clip As long) As long

Declare Function fbSetMouse cdecl alias "fbSetMouse"(ByVal x As long , ByVal y As long, ByVal visibility As long, ByVal clip As long) As long

Declare Sub fbCls cdecl alias "fbCls" (Byval mode as long)

Declare Sub fbWidthSet cdecl alias "fbWidthSet" (Byval columns as long, Byval row as long)

Declare Function fbWidthGet cdecl alias "fbWidthGet" () as long

Declare Sub fbColor cdecl alias "fbColor" (ByVal foreground As long , ByVal background As long)

Declare Function fbGetColor cdecl alias "fbGetColor"() as Ulong

Declare Function fbCursorY cdecl alias "fbCursorY" () As long

Declare Function fbCursorX cdecl alias "fbCursorX" () As long

Declare Sub fbLocate cdecl alias "fbLocate" (byval row as long,byval column as long,cursorState as long)

Declare Function fbScreen cdecl alias "fbScreen"(ByVal row As long, ByVal column As long, ByVal colorflag As long = 0) As long

Declare Sub fbWPrint cdecl alias "fbWPrint" (Byval s as wstring ptr)

Declare Sub fbLWPrint cdecl alias "fbWLPrint"(Byval text as wstring ptr)

Declare Sub fbSleep cdecl alias "fbSleep" (ByVal amount As long,flag as long)

Declare Function fbInkey cdecl alias "fbInkey"() As long

Declare Function fbGetKey cdecl alias "fbGetKey"() As long

Declare Function fbMultiKey cdecl alias "fbMultiKey"(ByVal scancode As long) As long

Declare Sub fbPCopy cdecl alias "fbPCopy"(ByVal source As long, ByVal destination As long)

Declare Function fbHiByte cdecl alias "fbHiByte" (ByVal value as long) as long

Declare Function fbLoByte cdecl alias "fbLoByte" (ByVal value as long) as long

Declare Function fbHiWord cdecl alias "fbHiWord" (ByVal value as long) as long

Declare Function fbLoWord cdecl alias "fbLoWord" (ByVal value as long) as long

Declare Function newArea cdecl alias "newArea"(byval h as long,byval w as long) as Area ptr

Declare Function fbGet cdecl alias "fbGet"(byval s as Area ptr,byval row as long,byval col as long) as long

Declare Function fbPut cdecl alias "fbPut"(byval s as Area ptr,byval row as long,byval col as long) as long

Declare Function fbAllocate cdecl alias "fbAllocate"(ByVal count As Ulong) As Any Ptr

Declare Sub fbDeallocate cdecl alias "fbDeallocate"(ByVal pt As Any Ptr)

Declare Sub fbDeallocateArea cdecl alias "fbDeallocateArea"(ByVal pt As Area Ptr)

Declare Sub fbClearArea cdecl alias "fbClearArea" (byval row as long, byval col as long,byval h as long,byval w as long,byval fg as Ulong,byval bg as Ulong)

Declare Sub fbFill cdecl alias "fbFill" (byval row as long, byval col as long,byval h as long,byval w as long,byval char as Ulong,byval fg as Ulong,byval bg as Ulong)

Declare Sub fbColorArea cdecl alias "fbColorArea" (byval row as long, byval col as long,byval h as long,byval w as long,byval fg as Ulong,byval bg as Ulong)

```
#include once "fbConsole.bi" ' include le definizioni
'----- Gestione del mouse-----
Function fbGetMouse cdecl alias "fbGetMouse" ( ByRef x As long, ByRef y As long, ByRef wheel As long, ByRef buttons As long, ByRef clip As long ) As long export 'legge lo stato del mouse
    return GetMouse ( x , y , wheel,buttons , clip)
End Function

Function fbSetMouse cdecl alias "fbSetMouse" ( ByVal x As long, ByVal y As long , ByVal visibility As long, ByVal clip As long ) As long export 'posiziona il mouse
    return SetMouse ( x , y, visibility,clip)
End Function

'-----Gestione Video-----
Sub fbCls cdecl alias "fbCls" (ByVal mode as long) export
    'mode=0 cancella l'intero schermo
    'mode=1 cancella la graphic viewport se definita, altrimenti cancella la text viewport
    'mode=2 cancella la text viewport
    Cls mode
End sub

Sub fbWidthSet cdecl alias "fbWidthSet" (Byval columns as long, Byval row as long) export
    'setta il numero di righe e colonne dello schermo
    Width columns, row
End sub

Function fbWidthGet cdecl alias "fbWidthGet" () as long export
    'ritorna il numero di righe e colonne dello schermo
    return Width() 'HiWord=row - LoWord=cols
End function

Sub fbColor cdecl alias "fbColor" ( ByVal foreground As long , ByVal background As long ) export
    Color (foreground, background) 'setta I colori attivi
End sub

Function fbGetColor cdecl alias "fbGetColor"()as Ulong export
    return color() 'ritorna I colori in uso HiWord=bg - LoWord=fg
End function

Function fbCursorY cdecl alias "fbCursorY" () As long export
    ' ritorna la riga in cui si trova il cursore
    return CsrLin
End function

Function fbCursorX cdecl alias "fbCursorX" () As long export
    ' ritorna la colonna in cui si trova il cursore
    return Pos()
end function

Sub fbLocate cdecl alias "fbLocate" (row as long, column as long,cursorState as long) export
    'posiziona il cursore e attiva ho disattiva la visibilità del cursore
    if row<1 and column <1 then 'nessuna variazione setta o resetta il cursorState
        Locate ,,cursor,State
    elseif row <1 then ' cambia solo la colonna
        Locate ,column,cursor,State
    elseif column<1 then ' cambia solo la riga
        Locate row,cursor,State
    else 'cambiano entrambi
        Locate row,column,cursorState
    end if
End sub
```

```

Function fbScreen cdecl alias "fbScreen"( ByVal row As long, ByVal column As long, ByVal colorflag As long ) As long export
    ' ritorna il carattere o l'attributo alla posizione riga e colonna desiderato colorflag=0 ascii code, colorflag=1 attributo
    return Screen( row, column , colorflag)
End Function

Sub fbWPrint cdecl alias "fbWPrint" ( ByVal s as wstring ptr) export
    'scrive un stringa unicode
    Print *s
End sub

'-----Stampante-----
Sub fbLWPrint cdecl alias "fbWLPrint"(Byval text as wstring ptr) export
    ' Send some text to the Windows printer on LPT1:, using driver text imaging.
    Open Lpt "LPT1:EMU=TTY" For Output As #1
    Print #1, *text
    Close
End sub

'-----Keyboard-----
Sub fbSleep cdecl alias "fbSleep" ( ByVal amount As Long,flag as Long) export
    'attende fintanto che un specificato tempo passi o che un tasto sia premuto
    if amount = -1 then 'attesa pressione di un tasto
        Sleep
    else
        Sleep amount,flag 'se flag
        'attesa pressione di un tasto o un certo numero di millisecondi ,flag=0 normale 1 attende la fine del del tempo richiesto
    end if
End sub

Function fbInkey cdecl alias "fbInkey"() As Long export
    'ritorna un intero rappresentante la prima chiave in attesa nel buffer di tastiera
    dim a as Long
    s=Inkey() # legge come stringa e ritorna come intero
    if len(s)=2 then
        a=asc(right(s,1))
        return (a Shl 8)+asc(left(s,1))
    end if
    return asc(s)
End Function

Function fbGetKey cdecl alias "fbGetKey"() As Long export ' ritorna l'ascii code della prima chiave nel buffer di tastiera
    return GetKey()
End Function

Function fbMultiKey cdecl alias "fbMultiKey"( ByVal scancode As Long ) As Long export
    'determina lo stato dei tasti dai codici di scansione della tastiera
    return MultiKey (scancode)
End Function

'-----Copia buffer video-----
sub fbPCopy cdecl alias "fbPCopy"( ByVal source As Long, ByVal destination As Long r) export
    'Copia una pagina grafica o di testo in un altra
    PCopy source,destination
End sub

'-----HI-LO-----
function fbHiByte cdecl alias "fbHiByte" (Byval value as Long) as Long export
    return HiByte(value) 'ritorna il secondo byte dell'operando.
End function

Function fbLoByte cdecl alias "fbLoByte" (Byval value as Long) as Long export

```



```
    return LoByte(value) 'ritorna il primo byte dell'operando.  
End function
```

```
Function fbHiWord cdecl alias "fbHiWord" (ByVal value as Long) as Long export  
    return HiWord(value) 'ritorna la seconda word dell'operando.  
End function
```

```
Function fbLoWord cdecl alias "fbLoWord" (ByVal value as Long) as Long export  
    return LoWord(value) 'ritorna la prima word dell'operando.  
End Function
```

----- Gestione finestre-----

```
Function newArea cdecl alias "newArea"(byval h as Long r,byval w as Long) as Area ptr export  
'alloca l'area per una finestra  
dim s as Area ptr=Allocate(sizeof(Area))  
s->buff=Allocate(h*w*sizeof(Video)*2)  
if (s->buff=0) then  
    s->id=0 'Null non allocato  
else  
    s->id=65 'A buffer allocato  
end if  
s->height=h  
s->width=w    return s  
End Function
```

```
Sub fbDeallocateArea cdecl alias "fbDeallocateArea" ( ByVal pt As Area Ptr ) export  
    Deallocate(pt->buff) 'elimina il buffer della finestra  
    Deallocate(pt)  
End Sub
```

```
Function fbGet cdecl alias "fbGet"(byval s as Area ptr,byval row as Long,byval col as Long) as Long export  
'copia una area video nel buffer  
dim as Long r,c,i=0  
dim v as Video ptr=>s->buff  
if s->id=65 then ' se è un puntatore valido  
    for r=row to row+s->height  
        for c=col to col+s->width  
            v[i].attr=Screen(r,c,1)    'prende l'attributo  
            v[i].char=Screen(r,c,0)    'prende il carattere  
            i+=1  
        next c  
    next r  
    return false 'ok  
else  
    return true 'errore  
end if  
End Function
```

```
Function fbPut cdecl alias "fbPut"(byval s as Area ptr,byval row as Long,byval col as Long) as Long export  
'scrive il buffer sullo schermo  
dim as Long r,c,i=0,ct= -1  
dim v as Video ptr=>s->buff  
if s->id=65 then ' se è un puntatore valido  
    for r=row to row+s->height  
        for c=col to col+s->width  
            if ct<> v[i].attr then # se l'attributo è cambiato  
                Color v[i].attr and &hf,v[i].attr shr 4 ' setta l'attributo  
                ct=v[i].attr 'salva l'attributo attuale  
            end if  
            v[i].attr=Screen(r,c,1)  
            v[i].char=Screen(r,c,0)  
            i+=1  
        next c  
    next r  
    return true  
else  
    return false  
end if  
End Function
```



```

        end if
        locate r,c:print chr(v[i].char)  ' scrive il carattere
        i+=1
    next c
next r
return false 'ok
else
    return true 'error
end if
End Function

Sub fbClearArea cdecl alias "fbClearArea" (byval row as Long, byval col as Long,byval h as Long r,byval w as Long,byval fg as
ULong r,byval bg as ULong) export
    ' cancella un area dello schermo con un dato colore
    fbFill(row,col,h,w,32,fg,bg)
End Sub





Sub fbFill cdecl alias "fbFill" (byval row as Long, byval col as Long,byval h as Long,byval w as Long,byval char as long,byval
fg as ULong,byval bg as ULong) export
    ' riempie un area dello schermo con un carattere e un colore desiderato
    dim as Long r,c,i=0
    color fg,bg
    for r=0 to h-1
        locate row+r,col
        print wString(w,char)
    next r
End Sub

Sub fbColorArea cdecl alias "fbColorArea" (byval row as Long, byval col as Long,byval h as Long,byval w as Long,byval fg as
ULong,byval bg as ULong) export
    ' Cambia il colore ad un area dello schermo lasciando inalterato il carattere contenuto
    dim tmp as Area PTR=>newArea(h,w)
    dim as Long r,c,i=0,co=bg*16+fg ' attributo
    dim valido as boolean
    dim v as Video ptr=>tmp->buff
    if tmp->id=65 then ' se è un puntatore valido
        valido=fbGet(tmp,row,col)
        for r=row to row+tmp->height
            for c=col to col+tmp->width
                v[i].attr=co ' nuovo attributo
                i+=1
            next c
        next r
        fbPut(tmp,row,col) ' scrive l'area
        fbDeallocateArea(tmp) ' cancella il buffer
    end if
End Sub

'-----Gestione memoria-----
Sub fbDeallocate cdecl alias "fbDeallocate" ( ByVal pt As Any Ptr ) export ' dealloca la memoria allocata
    Deallocate(pt)
End Sub

Function fbAllocate cdecl alias "fbAllocate"( ByVal count As ULong ) As Any Ptr
    return Allocate(count) ' alloca count blocchi di memoria
End Function

```

File Costant.py: Questo file contiene le costanti utilizzate		Passo 02
# Colori BLACK= 0 BLUE= 1 GREEN= 2 CYAN= 3 RED= 4 PURPLE= 5 BROWN= 6 LGREY= 7 DGREY= 8 LBLUE= 9 LGREEN= 10 LCYAN= 11 LRED= 12 LPURPLE= 13 YELLOW= 14 WHITE= 15 # Shadow BLOCK0= chr(0x2588) #  BLOCK1= chr(0x2591) #  BLOCK2= chr(0x2592) #  BLOCK3= chr(0x2593) #  SPACE= ' ' # Keyboard scancodes returned by MULTIKEY SC_ESCAPE =0x01 SC_1 =0x02 SC_2 =0x03 SC_3 =0x04 SC_4 =0x05 SC_5 =0x06 SC_6 =0x07 SC_7 =0x08 SC_8 =0x09 SC_9 =0x0A SC_0 =0x0B SC_MINUS =0x0C SC_EQUALS =0x0D SC_BACKSPACE=0x0E SC_TAB =0x0F SC_Q =0x10 SC_W =0x11 SC_E =0x12 SC_R =0x13 SC_T =0x14 SC_Y =0x15 SC_U =0x16 SC_I =0x17 SC_O =0x18 SC_P =0x19 SC_LEFTBRACKET =0x1A SC_RIGHTBRACKET =0x1B SC_ENTER =0x1C SC_CONTROL=0x1D	SC_A =0x1E SC_S =0x1F SC_D =0x20 SC_F =0x21 SC_G =0x22 SC_H =0x23 SC_J =0x24 SC_K =0x25 SC_L =0x26 SC_SEMICOLON =0x27 SC_QUOTE =0x28 SC_TILDE =0x29 SC_LSHIFT =0x2A SC_BACKSLASH =0x2B SC_Z =0x2C SC_X =0x2D SC_C =0x2E SC_V =0x2F SC_B =0x30 SC_N =0x31 SC_M =0x32 SC_COMMA =0x33 SC_PERIOD =0x34 SC_SLASH =0x35 SC_RSHIFT =0x36 SC_MULTIPLY =0x37 SC_ALT =0x38 SC_SPACE =0x39 SC_CAPSLOCK=0x3A SC_F1 =0x3B SC_F2 =0x3C SC_F3 =0x3D SC_F4 =0x3E SC_F5 =0x3F SC_F6 =0x40 SC_F7 =0x41 SC_F8 =0x42 SC_F9 =0x43 SC_F10 =0x44 SC_NUMLOCK =0x45 SC_SCROLLLOCK =0x46 SC_HOME =0x47 SC_UP =0x48 SC_PAGEUP =0x49 SC_LEFT =0x4B SC_RIGHT =0x4D SC_PLUS =0x4E SC_END =0x4F SC_DOWN =0x50 SC_PAGEDOWN =0x51 SC_INSERT =0x52 SC_DELETE =0x53 SC_F11 =0x57 SC_F12 =0x58	# Extra scancodes not compatible with DOS scancodes SC_LWIN =0x5B SC_RWIN =0x5C SC_MENU =0x5D # codici speciali riportati da inkey EXTCHAR=chr(255) # codice esteso o Null I_ESCAPE=chr(27) I_SPACE=chr(32) I_BACKSPACE=chr(8) I_TAB=chr(9) I_ENTER=chr(13) I_HOME=EXTCHAR+'G' I_UP=EXTCHAR+'H' I_PGUP=EXTCHAR+'I' I_LEFT=EXTCHAR+'K' I_CENTER=EXTCHAR+'L' I_RIGHT=EXTCHAR+'M' I_END=EXTCHAR+'O' I_DOWN=EXTCHAR+'P' I_PGDN=EXTCHAR+'Q' I_INS=EXTCHAR+'R' I_CANC=EXTCHAR+'S' I_F1=EXTCHAR+chr(59) I_F2=EXTCHAR+chr(60) I_F3=EXTCHAR+chr(61) I_F4=EXTCHAR+chr(62) I_F5=EXTCHAR+chr(63) I_F6=EXTCHAR+chr(64) I_F7=EXTCHAR+chr(65) I_F8=EXTCHAR+chr(66) I_F9=EXTCHAR+chr(67) I_F10=EXTCHAR+chr(68) I_F11=EXTCHAR+chr(132) I_F12=EXTCHAR+chr(134)

```

from ctypes import *
_FB64_=True # impostare a True se il compilatore freebasic è a 64 bit altrimenti a False se a 32 bit
class FBSTRING(Structure): # Descrittore Freebasic String
    _fields_=[('pointer',c_char_p),
              ('len',c_ssize_t),
              ('size',c_ssize_t)]

FB_MAXDIMENSIONS=4 # Massimo numero di dimensioni di un array utilizzate 3
FBARRAY_FLAGS_DIMENSIONS = 0x0000000f # number of entries allocated in arraydim ()
FBARRAY_FLAGS_FIXED_DIM   = 0x00000010 # array has fixed number of dimensions
FBARRAY_FLAGS_FIXED_LEN   = 0x00000020 # array points to fixed-length memory
FBARRAY_FLAGS_RESERVED    = 0xffffffffc0 # reserved, do not use

# tipi interi
BYTE=c_byte; UBYTE=c_ubyte; SHORT=c_short; USHORT=c_ushort
if _FB64_:
    INTEGER=c_longlong
    UINTEGER=c_ulonglong
else:
    INTEGER=c_long
    UINTEGER=c_ulong
LONG=c_long; ULONG=c_ulong; LONGINT=c_longlong; ULONGINT=c_ulonglong

# virgola mobile
SINGLE=c_float; DOUBLE=c_double

# ptr
ZSTRING=c_char_p; WSTRING=c_wchar_p ; STRING=POINTER(FBSTRING); ANY=c_void_p; PTR=POINTER
class Video(Structure): #text video struct
    _fields_=[('attr',c_int),
              ('char',c_int)]

class Area(Structure): #Area video struct
    _fields_=[('id',c_ubyte),
              ('height',c_int),
              ('width',c_int),
              ('buff',POINTER(Video))]

class FBArrayDim(Structure): #Array data dimensioni
    _fields_ = [("elements", c_size_t),
                ("lbound", c_ssize_t),
                ("ubound", c_ssize_t)]

class FBArray(Structure): #array descrittore struct
    _fields_ = [("data", c_void_p),
                ("ptr", c_void_p),
                ("size", c_size_t),
                ("element_len", c_size_t),
                ("dimensions", c_size_t),
                ("flags", c_ssize_t),
                ('arraydim',(FBArrayDim*FB_MAXDIMENSIONS))]

def CadrI(array,startIndex,Ctype): # deprecata, create un array con fbDim evitando il lavoro manuale
    ..... la funzione è già stata presentata
def fb_ArrayCalcDiff(*args): #calcola lo spostamento degli indici deprecata utilizzate fbDim
    ..... la funzione è già stata presentata
class fbDim: #Questa classe è già stata presentata nella sezione array

```

```

#-----
# Name:      fbConsole.py
# Purpose:   Wrapper FreeBasic Console Library
# Author:    Marco Salvati
# Created:   14/04/2019
# Copyright: (c) Marco Salvati 2019
# Licence:   GPL v.3
# Versione:  1.9
#-----

from fbTypes import *
from Costant import *
import os
FBCONSOLE_PATH=r'C:\myDll\fbConsole.dll' #Inserire qui il path assoluto della VS libreria
class fbConsole:
    def __init__(self):
        self._fbl=cdll.LoadLibrary(FBCONSOLE_PATH)
        self._pos=()
        self._fg,self._bg= LGREY,BLACK # colori standart
        # mouse info
        self._X,self._Y,self._Buttons,self._Wheel,self._visibility,self._Clip=c_int(0),c_int(0),c_int(0),c_int(0),\
        c_int(0),c_int(0)
        self.actual=(LGREY,BLACK) #colori attuali
        self.set_color(LGREY,BLACK) #setta i colori attuali
        self.cornici={'SINGLE': '┌─┐└─┐└─┐└─┐','DOUBLE': '═┐┌═┐┌┐┐┐┐'} # cornici delle finestre
        self._width=self.width() # acquisisce le dimensioni dello schermo
    def cls(self,mode=0): #Cancella lo schermo
        self._fbl.fbCls.argtypes=[c_int] #utilizza per default l'opzione zero intero schermo l'unica che ci interessa
        self._fbl.fbCls(c_int(mode))
    def LoByte(self,n): #ritorna il primo byte dell'operando
        self._fbl.fbLoByte.argtypes=[c_int]
        self._fbl.fbLoByte.restype=c_int
        return self._fbl.fbLoByte(c_int(n))
    def HiByte(self,n): #ritorna il secondo byte dell'operando
        self._fbl.fbLHiByte.argtypes=[c_int]
        self._fbl.fbHiByte.restype=c_int
        return self._fbl.fbLowByte(c_int(n))
    def LoWord(self,n): #ritorna la prima word dell'operando
        self._fbl.fbLoWord.argtypes=[c_int]
        self._fbl.fbLoWord.restype=c_int
        return self._fbl.fbLoWord(c_int(n))
    def HiWord(self,n): #ritorna la seconda word dell'operando.
        self._fbl.fbHiWord.argtypes=[c_int]
        self._fbl.fbHiWord.restype=c_int
        return self._fbl.fbHiWord(c_int(n))
    def width(self,col=-1,row=-1): #setta o ritorna le dimensioni dello schermo
        self._fbl.fbWidthGet.restype = c_int
        rc=self._fbl.fbWidthGet() #legge il numero di righe e colonne attuali dello schermo
        if col > -1 and row > -1: #setta le nuove dimensioni dello schermo se presenti
            self._fbl.fbWidthSet.argtypes=[c_int,c_int]
            self._fbl.fbWidthSet(c_int(col),c_int( row))
            self._width=(col,row)
        return
        r=self.HiWord(rc) #ricava il numero di righe

```

```

c=self.LoWord(rc) #ricava il numero colonne
self._width=(c,r) #salva le nuove dimensioni settate
return (c,r) # ritorna una tuple (colonne,righe)
def columns(self): # ritorna il numero di colonne settate con width
    return self._width[0]
def lines(self): # ritorna il numero di righe settate con width
    return self._width[1]
@property
def cursorY(self): #ritorna la riga in cui si trova il cursore
    self._fbl.fbCursorY.restype = c_int
    return self._fbl.fbCursorY()
@property
def cursorX(self): #ritorna la colonna in cui si trova il cursore
    self._fbl.fbCursorX.restype = c_int
    return self._fbl.fbCursorX()
def locate(self,row,col,cursorState=1): #Posizione del cursore
    self._fbl.fbLocate.argtypes=[c_int,c_int,c_int]
    self._fbl.fbLocate(c_int(row),c_int(col),c_int(cursorState))
def save_pos(self): #salva la posizione attuale del cursore
    self._pos=(self.cursorY(),self.cursorX())
def restore_pos(self): #ripristina la posizione del cursore
    self.Locate(self._pos[0],self._pos[1])
def reset(self): #resetta i colori
    self.set_color(LGREY,BLACK)
def set_color(self, fg=None, bg=None): #setta colori in uso
    self._fbl.fbColor.argtypes=[c_int,c_int]
    fg=fg if fg is not None else self.actual[0]
    bg=bg if bg is not None else self.actual[1]
    self._fbl.fbColor(c_int(fg),c_int(bg))
    self.actual=(fg,bg) #colori attuali
def color(self): #ritorna i colori attuali utilizzati
    return self.actual
def get_color(self): #ritorna i colori attuali utilizzando Freebasic deprecata utilizzare color
    self._fbl.fbGetColor.restype=c_uint
    c=self._fbl.fbGetColor()
    return (self.HiWord(c),self.LoWord(c)) # tupla (foreground,background)
def screen(self,row,col,colorflag=0): #ritorna il carattere o il suo colore alle coordinate video indicate
    self._fbl.fbScreen.restype = c_int
    self._fbl.fbScreen.argtypes=[c_int,c_int,c_int]
    rc=self._fbl.fbScreen(row,col,colorflag)
    if colorflag>0: # colorflag : 0 codice ascii,1 colore ritorna tupla stile Python (fg,bg)
        rc= (rc & 15,rc >> 4)
    return rc
def sleep(self,amount=-1,flag=0): #attende pressione di un tasto,oppure attende per un certo numero di
millisecondi
    self._fbl.fbSleep.argtypes=[c_int,c_int]
    self._fbl.fbSleep(c_int(amount),c_int(flag))
def print(self,text): #stampa una stringa unicode sullo schermo
    if isinstance(text,bytes): #se bytes converti in string
        text=text.decode()
    elif not isinstance(text,str): # se non string converti in str
        text=str(text)
    t=c_wchar_p(str(text))
    self._fbl.fbWPrint.argtypes=[c_wchar_p]

```

```

self._fbl.fbWPrint(t)
def print_at(self,row,col,text):    #stampa un testo alle coordinate desiderate
    self.locate(row,col)
    self.print(text)
def cprint(self,fg,bg,text): #stampa un testo con i colori desiderati alla posizione attuale del cursore
    self.set_color(fg,bg)
    self.print(text)
def lprint(self, text):            #invia un testo alla stampante
    if isinstance(text,bytes): #converti se non Unicode
        t=text.decode()
    elif not isinstance(text,str): t=str(text) # converti in stringa
    else: t=text #unicode
    t=c_wchar_p(t)
    self._fbl.fbWPrint.argtypes=[c_wchar_p]
    self._fbl.fbWPrint(t)          #invia alla stampante
def move_left(self,c=1):          #muovi il cursore indietro di di c posizioni
    row=self.cursorY()
    col=self.cursorX()
    self.Locate(row,col-c)
def move_right(self,c=1):         #muovi il cursore avanti di di c posizioni
    row=self.cursorY()
    col=self.cursorX()
    self.Locate(row,col+c)
def move_up(self,c=1):           #muovi il cursore sopra di di c posizioni
    row=self.cursorY()
    col=self.cursorX()
    self.Locate(row-c,col)
def move_down(self,c=1):         #muovi il cursore sotto di di c posizioni
    row=self.cursorY()
    col=self.cursorX()
    self.Locate(row+c,col)
def cursor_on(self):             #visualizza il cursore
    self.locate(-1,-1,1)
def cursor_off(self):            #nasconde il cursore
    self.locate(-1,-1,0)
def reverse(self):               #inverte i colori
    tmp=self.color()
    self.color(tmp[1],tmp[0])
def getmouse(self):              # legge lo stato del mouse
    self._fbl.fbGetMouse.argtypes = [POINTER(pMouse)]
    self._fbl.fbGetMouse.restype = c_int
    return self._fbl.fbGetMouse(self.data)
def setmouse(self,x=-1,y=-1,visibility=1,clip=1): #setta la posizione del mouse
    self._fbl.fbSetMouse.argtypes = [c_int,c_int,c_int,c_int]
    self._fbl.fbSetMouse.restype = c_int
    self._X.value,self._Y.value,self._visibility.value,self._Clip.value=x,y,visibility,clip
    return self._fbl.fbSetMouse(self._X.value,self._Y.value,self._visibility.value,self._Clip.value)
@property
def mouseX(self):                #ritorna l'ultima posizione X del mouse letta
    return self.data.x
@property
def mouseY(self):                #ritorna l'ultima posizione Y del mouse letta
    return self.data.y
@property

```



```

def wheel(self): #ritorna l'ultima posizione della rotella del mouse letta nella modalità grafica
    return self.data.wheel # in modalità testo non ritorna nulla ma effettua uno scrol del video
@property
def btLeft(self): #ritorna lo stato del pulsante di sinistra del mouse nell'ultima lettura
    return self.data.btLeft==1
@property
def btRight(self): #ritorna lo stato del pulsante di destra del mouse nell'ultima lettura
    return self.data.btRight==2
@property
def btMiddle(self): #ritorna lo stato del pulsante centrale del mouse nell'ultima lettura
    return self.data.btMiddle==4
@property
def clip(self):
    return self.data.clip #stato del mouse in modalità grafica 1 nella finestra 0 fuori
def mouse_over(self,r,c,h,w):
    # ritorna True se il mouse è nelle coordinate desiderate
    return (self.mouseY in range(r,r+h))and(self.mouseX in range(c,c+w))
def mouse_click(self,button,r,c,h,w): # ritorna True se il mouse è cliccato alle coordinate desiderate
    select={ 1:self.btLeft,2:self.btRight,3:self.btMiddle }
    if select[button]:
        return self.mouse_over(r,c,h,w)
    else:
        return False
def inkey(self): #legge da tastiera ritorna una stringa
    self._fbl.fbInkey.restype = c_int
    r=self._fbl.fbInkey()
    if r==0:
        return EXTCHAR #nessun tasto è stato premuto
    elif r > 255: #tasto speciale
        a=r>>8
        return EXTCHAR+ chr(a) #tasto speciale
    else:
        return chr(r) #ascii code
def getkey(self): #legge da tastiera codice codificato in un intero
    self._fbl.fbGetKey.restype = c_int
    t=self._fbl.fbGetKey()
    return t
def getch(self): #attesa della pressione di un tasto ritorna un intero
    a=0
    while a==0:a=self.getkey()
    return a
def getchar(self): #attesa della pressione di un tasto ritorna una stringa
    a=EXTCHAR
    while a==EXTCHAR : a=self.inkey()
    return a
def multikey(self,code): #determina lo stato dei tasti dai codici di scansione della tastiera
    self._fbl.fbMultiKey.restype = c_int
    self._fbl.fbMultiKey.argtypes =[c_int]
    return self._fbl.fbMultiKey(code)
def reverse(self): #inverte I colori
    c=self.color()
    self.set_color(c[1],c[0])
def reset_colors(self): #resetta I colori
    self.reset()

```

```

def fill(self,row,col,h,w,char,fg=LGREY,bg=BLACK): #riempie un area con un carattere e il suo colore
    tmp=self.color()
    o=ord(char)
    self._fbl.fbFill.argtypes=[c_int,c_int,c_int,c_int,c_uint,c_uint,c_uint]
    self._fbl.fbFill(c_int(row),c_int(col),c_int(h),c_int(w),c_uint(o),c_uint(fg),c_uint(bg))
    self.set_color(tmp[0],tmp[1])
def clear_area(self,row,col,h,w,fg=LGREY,bg=BLACK): #cancella un area dello schermo con un colore
    self._fbl.fbClearArea.argtypes=[c_int,c_int,c_int,c_int,c_uint,c_uint]
    self._fbl.fbClearArea(c_int(row),c_int(col),c_int(h),c_int(w),c_uint(fg),c_uint(bg))
def color_area(self,row,col,h,w,fg=LGREY,bg=BLACK): #cambia il colore ad un area dello schermo
    self._fbl.fbColorArea.argtypes=[c_int,c_int,c_int,c_int,c_uint,c_uint]
    self._fbl.fbColorArea(c_int(row),c_int(col),c_int(h),c_int(w),c_uint(fg),c_uint(bg))
def vseparator(self,row,col,h,relief='SINGLE'): #separatoro verticale di un box
    relief=relief.upper()
    up=self.cornici[relief][9]
    down= self.cornici[relief][8]
    middle= self.cornici[relief][2]
    self.print_at(row,col,up)
    for i in range(1,h): self.print_at(row+i,col,middle)
    self.print_at(row+h,col,down)
def hseparator(self,row,col,w,relief='SINGLE'): #separatoro orrizzontale di un box
    relief=relief.upper()
    left= self.cornici[relief][6]
    right= self.cornici[relief][7]
    middle= self.cornici[relief][3]
    self.print_at(row,col,left)
    tmp=left+middle*(w-1)+right
    self.print_at(row,col,tmp)
def cseparator(self,row,col,relief='SINGLE'): #unisce un separatore verticale con uno orrizzontale
    relief=relief.upper()
    cross= self.cornici[relief][10]
    self.print_at(row,col,cross)
def box(self,row,col,h,w,relief='SINGLE',title=None): #Crea una cornice
    relief=relief.upper()
    bo= self.cornici[relief]
    up=bo[0]+bo[3]*(w-1)+bo[1]
    down=bo[4]+bo[3]*(w-1)+bo[5]
    self.print_at(row,col,up)
    for i in range(1,h):
        self.print_at(row+i,col,bo[2])
        self.print_at(row+i,col+w,bo[2])
    self.print_at(row+h,col,down)
    if title is not None:
        l=(w-len(title))//2
        self.print_at(row,col+l,title)
def shadow(self,row,col,h,w,fg=None,bg=None,syb=BLOCK3): #ombra di un area
    self.set_color(fg,bg)
    col2=col+w+1
    row2=row+h+1
    for i in range(1,h+1):
        self.print_at(row+i,col2,syb)
    for i in range(1,w+2):
        self.print_at(row2,col+i,syb)
def pcopy(self,source , destination): #salva o ripristina una schermata video,0 pagina visibile,1,2,3 buffer

```



```

self._fbl.fbPCopy.argtypes=[c_int,c_int] # il numero dei buffer dipende dal Sistema operativo
self._fbl.fbPCopy(c_int(source),c_int(destination))
def ajust(self,s,width,justify): # giustifica una stringa rispetto a width
    if justify=='left': return s.ljust(width,' ')
    if justify=='right': return s.rjust(width,' ')
    if justify=='center': m=(width-len(s))/2;return ' '*m + s + ' '*m
def newArea(self,h,w): #crea un buffer per un area video
    self._fbl.newArea.argtypes=[c_int,c_int]
    self._fbl.newArea.restype=POINTER(Area)
    return self._fbl.newArea(c_int(h),c_int(w))
def put(self,buff,h,w): #scrive il buffer nella posizione video indicata
    self._fbl.fbPut.argtypes=[POINTER(Area),c_int,c_int]
    self._fbl.fbPut.restype=c_int
    return self._fbl.fbPut(buff,c_int(h),c_int(w))
def get(self,buff,h,w): #copia un area video nel buffer
    self._fbl.fbGet.argtypes=[POINTER(Area),c_int,c_int]
    self._fbl.fbGet.restype=c_int
    return self._fbl.fbGet(buff,c_int(h),c_int(w))
def deallocateArea(self,pt): #Elimina il buffer
    self._fbl.fbDeallocateArea.argtypes=[POINTER(Area)]
    self._fbl.fbDeallocateArea(pt)
def allocate(self, count): #alloca count posti di memoria
    self._fbl.fbAllocate.argtypes=[c_uint]
    self._fbl.fbAllocate.restype=c_void_p
    return self._fbl.fbAllocate(c_uint(count))
def deallocate(self,pt): #deallocata la memoria creata con allocate
    self._fbl.fbDeallocate.argtypes=[c_void_p]
    self._fbl.fbDeallocate(pt)

```

Ora vediamo un paio di esempi per provare le potenzialità della libreria

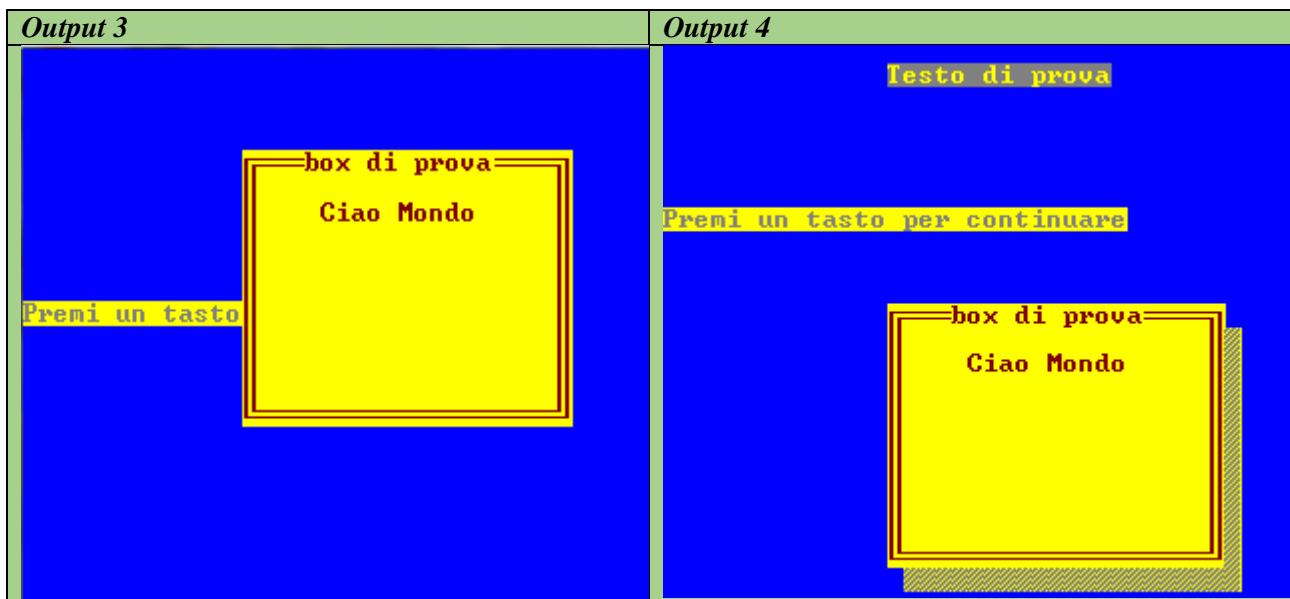
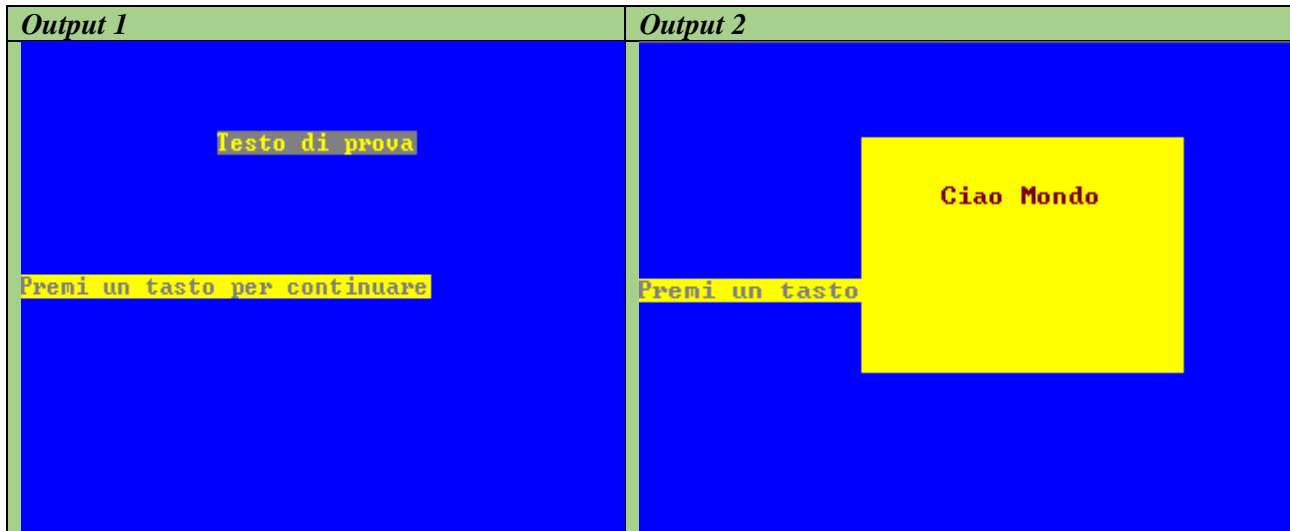
File test_console 01

```

from fblib.fbConsole. fbConsole import *
from fblib.Costant import *
term=fbConsole() #Crea un terminale
term.width(120,50) # setta una console 120 colonne x 50 righe
term.set_color(WHITE,LBLUE) # setta bianco su blu brillante
term.cls() #cancella lo schermo con i nuovi colori
term.locate(5,15) #posiziona il cursore
term.cprint(YELLOW,DGREY,'Testo di prova') #scrive un testo con i nuovi colori giallo su grigio scuro
term.reverse() # inverte i colori
term.move_down(5) # sposta il cursore in basso di 5 righe
term.print('Premi un tasto per continuare')
term.sleep() # attesa pressione di un tasto
p1=term.newArea(10,20) # crea un buffer 10 righe x 20 colonne
term.get(p1,5,15) # salva l'area
term.clear_area(5,15,10,20,fg=RED,bg=YELLOW) #cancella l'area con i colori rosso su giallo
term.print_at(7,20,'Ciao Mondo') #scrivi a un testo nella posizione voluta
term.sleep(500) # ritardo
term.box(5,15,10,20,relief='DOUBLE',title='box di prova') #crea un box intorno all'area
term.sleep(500) # ritardo
p2=term.newArea(10,20) # crea un buffer 10 righe x 20 colonne
term.get(p2,5,15) # salva l'area
term.put(p1,5,15) #ripristina la vecchia area
term.put(p2,15,15) #riscrive l'area in una altra posizione

```

```
term.shadow(15,15,10,20) #crea l'ombra della finestra con i colori e il carattere di default
term.deallocateArea(p1) # cancella i buffer's
term.deallocateArea(p2) #Elimina la memoria allocata
term.sleep(500)          # ritardo
```

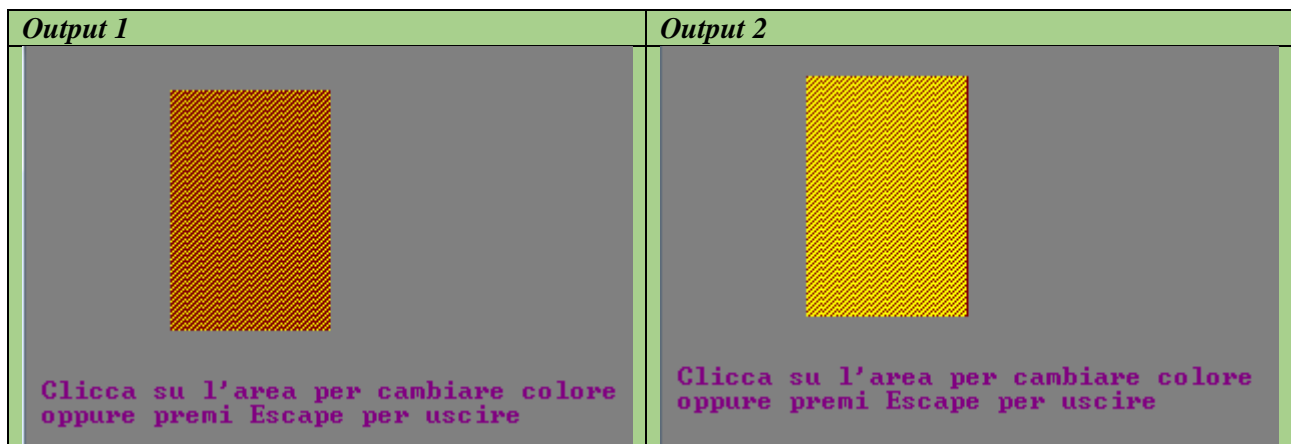


Un esempio semplice ma già con queste poche righe si nota la potenza delle primitive in freebasic.

File test_console 02.py

```
from fplib.fbConsole.fbConsole import *
from fplib..Costant import *

term=fbConsole()    # terminale
term.width(120,50)  # setta una console 120 colonne x 50 righe
term.set_color (PURPLE, DGREY) # setta viola su grigio scuro
term.cls()          #cancella lo schermo con i nuovi colori
term.fill(10,10,10,10,BLOCK3,fg=RED,bg=YELLOW) #riempie un area con un carattere e colore
term.print_at(22,1,"Clicca su l'area per cambiare colore ")
term.print-at(23,2,'oppure premi Escape per uscire')
Loop=True
while Loop: #Creo un gestore di eventi elementare
    s=term.inkey() #legge la tastiera
    m=term.getmouse() #legge il mouse
    if s==I_ESCAPE: #escape per uscire
        Loop=False
    elif term.mouse_click(1,10,10,10,10): #pulsante sinistro premuto sopra l'area creata
        term.reverse() #inverti i colori
        c=term.color() #legge i nuovi colori
        term.fill(10,10,10,10,BLOCK3,fg=c[0],bg=c[1]) #riempie l' area con un carattere e il nuovo colore
```



Ogni volta che premiamo sull'area colorata, i colori si invertono

File test pcopy.py: Esempio copia del buffer video (valido per windows)

```
from fplib.fbConsole.fbConsole import *
con=fbConsole()
con.width(120, 50) # setta la console 50 righe x 120 colonne
row=con.lines() #righe del buffer video
col=con.columns() #colonne del buffer video
i=0

con.cls() #cancella lo schermo
con.locate(1,1)
con.set_color(RED, BLACK)
con.print('A'*(row*col))
con.pcopy(0,1) #salva la pagina nel buffer 1
for x in range(10000000):pass #ritardo
con.set_color(YELLOW, BLACK)
con.cls()
con.locate(1,1)
con.print('B'*(row*col))
#0 pagina attiva – 1,2,3 buffer

con.pcopy(0,2) # salva la pagina nel buffer 2
for x in range(10000000):pass #ritardo
con.set_color(BLUE, BLACK)
con.cls()
con.locate(1,1)
con.print('C'*(row*col))
con.pcopy(0,3) # salva la pagina nel buffer 3
for x in range(8000000):pass #ritardo
con.set_color(LPURPLE, BLACK)
con.cls()
con.locate(1,1)
con.print('D'*(row*col))
con.pcopy(0,4) #prova a salvare la pagina nel buffer 4
for x in range(8000000):pass #ritardo
con.cls()
con.pcopy(1,0) #ripristina la pagina 1
for x in range(8000000):pass #ritardo
con.cls()
con.pcopy(2,0) #ripristina la pagina 2
for x in range(8000000):pass
con.cls()
con.pcopy(3,0) #ripristina la pagina 3
for x in range(8000000):pass
con.cls()
con.pcopy(4,0) # tentativo fallito pagina non abilitata
for x in range(8000000):pass #ritardo
```

[illegible]

Piattaforma differenti

- *Il Massimo numero di pagine di testo in Windows è 4.*
- *Il Massimo numero di pagine di testo in DOS è 8.*
- *Il Massimo numero di pagine di testo in tutti gli altri sistemi è 1.*

Il Massimo numero di pagine grafiche dipende da ciò che viene specificato alla chiamata Screen statement o in Screenres. (Questa opzione non ci interessa)

WinBase Classe dei metodi comuni alle finestre e all'oggetto Screen

La classe WinBase, contiene primitive comuni per la gestione delle finestre.
Questa classe sarà ereditata dalle classi Screen e Window che andremo in seguito a creare.

File Winbase :Classe base per le finestre

Passo 03

classe finestra per console

```
from fbllib.fbConsole.fbConsole import *  
from fbllib.Costant import *
```

class WinBase:

```
def __init__(self,term,row,col,height,width,title=" ,relief='SINGLE',fg=LGREY,bg=BLACK ,repaint=None):
```

```
    self._term=term
```

```
    self._repaint=repaint
```

```
    self._var={'row':1,'col':1,'height':15,'width':50,'title':" \ #valori di default  
                'bg':BLACK,'fg':LGREY,'active_bg':BLACK,'active_fg':LGREY,\  
                'relief':'SINGLE','cursorX':1,'cursorY':1 }
```

```
    self._var['row']=row; self._var['col']=col #aggiorna I valori di default
```

```
    self._var['height']=height; self._var['width']=width; self._var['title']=title
```

```
    self._var['relief']=relief; self._var['bg']=bg; self._var['fg']=fg
```

```
@property
```

```
def term(self): #ritorna il terminale
```

```
    return self._term
```

```
def paint(self):
```

```
    pass # non implementato
```

```
def repaint(self,*args,**kargs):
```

```
    if self._repaint is not None:self._repaint(*args,**kargs) # chiama la funzione di repaint se esiste
```

```
def __getitem__(self,key): #ritorna il valore di una variabile interna
```

```
    if key in self._var: return self._var[key]
```

```
    else: return None # chiave non presente
```

```
def __contains__(self,key): return key in self._var #controlla se una chiave è presente
```

```
def __setitem__(self,key,value): #cambia il valore di una variabile interna o ne crea una con il suo valore
```

```
    self._var[key]=value
```

```
def __delitem__(self,key):
```

```
    if key in self._var: del(self._var[key]) # cancella la chiave se presente
```

Autore: Marco Salvati

Licenza Tutorial: CC BY 3.0

Email: marcosalvati61@gmail.com

Licenza Software: GPL V.3


```

def wcalc(self,r,c): # controllo sulla nuova posizione del cursore
    if r in range(1,self._var['height']):
        row=self._var['row']+r
    else:
        row=self._var['row']+self._var['height']-1
    if c in range(1,self._var['width']):
        col=self._var['col']+c
    else:
        col=self._var['col']+self._var['width']-1
    char=self._var['width']-c # numero di caratteri scrivibili
    self._var['cursorY']=r;self._var['cursorX']=c # salva la nuova posizione del cursore
    return (row,col,char)

def valid_area(self,row,col,h,w): #controllo se le coordinate sono nel range giusto
    r1=(row>0 and row+h in range(self._var['height']+1))
    c1= (col>0 and col+w in range(self._var['width']+1))
    if r1 and c1: return True
    return False

def border(self): #cambia il bordo della finestra
    self._term.set_color(self._var['fg'],self._var['bg'])
    tmp= 'DOUBLE'if self._var['relief']== 'SINGLE' else 'SINGLE'
    self._var['relief']=tmp
    self._term.box(self._var['row'],self._var['col'],self._var['height'],self._var['width'],\
        relief=self._var['relief'],title=self._var['title'])

def mouse_over(self,r,c,h,w):
    # ritorna True se il mouse è nelle coordinate desiderate altrimenti False
    return (self._term.mouseY in range(r-1,r+h))and(self._term.mouseX in range(c,c+w))

def mouse_click(self,button,r,c,h,w):
    #ritorna True se il mouse è stato cliccato nell'area indicata
    row=self._var['row']+r-1
    col=self._var['col']+c
    select={ 1:self._term.btLeft,2:self._term.btRight,3:self._term.btMiddle }
    if select[button]:
        return self.mouse_over(row,col,h,w)
    else:
        return False

```

Screen, questa classe è necessaria se vogliamo utilizzare dei widgets sulla finestra principale.

File Screen : Finestra principale intero schermo

Passo 04

```

from fblib.fbConsole.fbConsole import *
from fblib. Costant import *
from fblib.fbConsole.WinBase import *
class Screen(WinBase):
    def __init__(self,term,title=",relief='SINGLE',fg=LGREY,bg=BLACK ,repaint=None):
        super().__init__(term,1,1,term.lines()-1,term.columns()-1,title=title, relief=relief, fg=fg, bg=bg, repaint=repaint)
        term.set_color(self._var['fg'],self._var['bg'])
        term.cls(0)
        def border(self): # soprascrive il metodo per rientrare nei limiti dello schermo
            self._term.set_color(self._var['fg'],self._var['bg'])
            tmp= 'DOUBLE' if self._var['relief']== 'SINGLE' else 'SINGLE'
            self._var['relief']=tmp
            self._term.box(self._var['row'],self._var['col'],self._var['height']-1,self._var['width']-1,relief= self._var['relief'],\
                title=self._var['title'])

```

Window, questa classe gestisce le primitive e i widgets al suo interno

File Window.py **questa classe gestisce le primitive e i widgets al suo interno** **Passo 05**

```
from fblib.fbConsole.fbConsole import *
from fblib.Costant import *
from fblib.fbConsole.WinBase import *

class Window(WinBase):
    def __init__(self,term,row,col,height,width,title='',relief='SINGLE',fg=LGREY,bg=BLACK,repaint=None):
        super().__init__(term,row,col,height,width,title=title,relief=relief,fg=fg,bg=bg,repaint=repaint)
        if 'visible' not in self._var : self._var['visible']=True
        self._buffer=self._term.newArea(self._var['height'],self._var['width']) # crea il buffer per la finestra
        self._buffer2=None
        t=self._term.get(self._buffer,self._var['row'],self._var['col']) # salva l'area destinata alla finestra
        self._repaint=repaint
        self.paint()
    def __deallocArea__(self): # magic method non standart utilizzato da kill per deallocare la finestra
        if self.isVisible(): #se la finestra è visibile
            t=self._term.put(self._buffer,self._var['row'],self._var['col']) #ripristina l'area
        else:
            self._term.deallocateArea(self._buffer2) # dealloca il buffer temporaneo
            self._term.deallocateArea(self._buffer) #dealloca il buffer
    def hide(self): # nasconde la finestra e ripristina l'area
        if self.isVisible():
            self._buffer2=self._term.newArea(self._var['height'],self._var['width']) # nuovo buffer
            t=self._term.get(self._buffer2,self._var['row'],self._var['col']) # salva la finestra
            t=self._term.put(self._buffer,self._var['row'],self._var['col']) # ripristina lo sfondo
            self._var['visible']=False
    def show(self): # mostra la finestra
        if not self.isVisible():
            t=self._term.put(self._buffer2,self._var['row'],self._var['col']) # ripristina la finestra
            self._term.deallocateArea(self._buffer2) # cancella il buffer
            self._buffer2=None
            self._var['visible']=True
    def move(self,row,col): #sposta la finestra alle nuove coordinate
        self.hide()
        self._var['row']=row;self._var['col']=col # nuova posizione
        t=self._term.get(self._buffer,self._var['row'],self._var['col']) # salva l'area destinata alla finestra
        self.show() # mostra la finestra nella nuova posizione
    def isVisible(self): # True se la finestra è visibile
        return self._var['visible']
    def paint(self): #disegna la finestra
        self._term.clear_area(self._var['row'],self._var['col'],self._var['height'],self._var['width'], fg=self._var['fg'],\
            bg=self._var['bg'])
        self._term.box(self._var['row'],self._var['col'],self._var['height'],self._var['width'],relief=self._var['relief'],\
            title=self._var['title'])
    def is_mouse_over(self): # controlla se il mouse è nella finestra
        row=self._parent['row']+self._var['row']-1
        col=self._parent['col']+self._var['col']
        return self._term.mouse_over(row,col,self._var['height'],self._var['width'])
    def is_mouse_click(self,button): # controlla se il mouse è stato cliccato nella finestra
        row=self._parent['row']+self._var['row']-1
```



```

col=self._parent['col']+self._var['col']
return self._term.mouse_click(button,row,col,self._var['height'],self._var['width'])
# dealloca il buffer della finestra e cancella il riferimento
def kill(obj):
    b=dir(obj) #Controllo la presenza del magic method
    if '__deallocateArea__' in b: #dealloca il buffer
        obj.__deallocateArea__()
    del(obj) # elimina l'oggetto

```

La classe è ben leggibile i metodi documentati, interessante è l'uso del magic method **non standart** e della funzione kill, questa prima controlla se è presente il metodo `__deallocateArea__`. Se si ripristina l'area salvata e dealloca il buffer quindi elimina l'oggetto. Se no elimina l'oggetto, possiamo considerarla un del avanzato.

Un esempio di una semplice finestra e dell'uso della funzione repaint.

File test_window 01

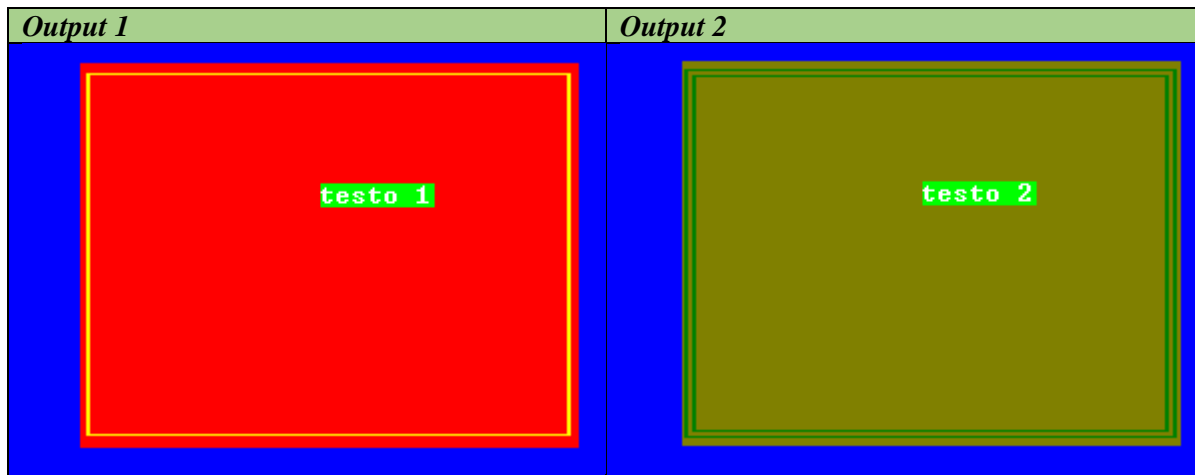
```

from fblib.fbConsole.Window import *
from fblib.Costant import *
term=fbConsole()
term.set_color(RED,LBLUE)
term.cls()

def repaint(fg,bg,text):
    win['fg']=fg # nuovo foreground
    win['bg']=bg # nuovobackground
    win.paint() # aggiorna la finestra
    win.border() # cambia il bordo della finestra
    term.locate(10,20) #posiziona il cursore
    term.cprint(WHITE,LGREEN,text) #scrive il testo con I nuovi colori

win=Window(term,5,5,15,30,fg=YELLOW,bg=LBLUE,repaint=repaint)
term.locate(10,20)
term.cprint(WHITE,LGREEN,'testo 1')
term.sleep(600) # ritardo
win.repaint(GREEN,BROWN,'testo 2') #chiamo la funzione di repaint
term.sleep(600) # ritardo

```



Semplice esempio sull'uso della funzione di repaint della finestra

<i>File Write.py : Scrive in una finestra con le coordinate relative</i>	<i>passo 6</i>
<pre> from fbllib.fbConsole.fbConsole import * from fbllib.fbConsole.Window import * from fbllib.fbTypes import * from fbllib.Costant import * class Write: def __init__(self, win): self._win=win #salva il parent self._term=win.term # salva il terminale self._cstack=[] #color stack unico per ogni finestra def print(self,text): #funzione print adatta alla finestra self.print_at(self._win['cursorY'],self._win['cursorX'],text) def cprint(self,fg,bg,text): #funzione cprint adatta alla finestra r=self._win.wcalc(self._win['cursorY'],self._win['cursorX']) self._term.locate(r[0],r[1]) self._term.cprint(fg,bg,text[:r[2]]) self._win['active_fg']=fg;self._win['active_bg']=bg self._win['cursorX']+=len(text[:r[2]]) # aggiorna per la lunghezza del testo scritto def print_at(self,row,col,text): #funzione print_at adatta alla finestra if '\n' in text: t=text.split('\n') #testo multi riga i=0 for l in t: r=self._win.wcalc(row+i,col) self._term.print_at(r[0],r[1],t[i][:r[2]]) i+=1 self._win['cursorX']+=len(t[i-1][:r[2]]) # aggiorna per la lunghezza del testo scritto else: </pre>	

```

r=self._win.wcalc(row,col)
self._term.print_at(r[0],r[1],text[:r[2]])
self._win['cursorX']+=len(text[:2]) # aggiorna per la lunghezza del testo scritto

def set_color(self, fg=None, bg=None): # setta i colori della finestra
    fg=fg if fg is not None else self._win['active_fg']
    bg=bg if bg is not None else self._win['active_bg']
    self._term.set_color(fg,bg)
    self._win['active_fg']=fg;self._win['active_bg']=bg
def color(self): #ritorna i colori in uso
    return (self._win['active_fg'],self._win['active_bg'])
def screen(self,row,col,colorflag=0): # ritorna il carattere o il suo colore alle coordinate indicate
    return self._term.screen(self._win['row']+row,self._win['col']+col,colorflag)
def locate(self,row,col): #Posiziona il cursore
    r=self._win.wcalc(row,col)
    self._term.locate(r[0],r[1])
def reverse(self): #inverte i colori
    tmp=self.color()
    self.set_color(tmp[1],tmp[0])
def reset(self):#ripristina i colori iniziali
    self.set_color(self._win['fg'],self._win['bg'])
    self._win['active_fg']=self._win['fg'];self._win['active_bg']=self._win['bg']
def clear_area(self,row,col,h,w,fg=None,bg=None): #cancella un area della finestra con i colori desiderati
    self.fill(row,col,h,w,' ',fg,bg)
def cls(self): #cancella il contenuto della finestra con i colori default
    self.clear_area(1,1,self._win['height'],self._win['width'],fg=self._win['fg'],bg=self._win['bg'])
def fill(self,row,col,h,w,char,fg=BLACK,bg=None):#riempie un area della finestra con un carattere e i colori
    fg=fg if fg is not None else self._win['fg'] #desiderati
    bg=bg if bg is not None else self._win['bg']
    if self._win.valid_area(row,col,h,w): # se rientra nella finestra
        self._term.fill(self._win['row']+row,self._win['col']+col,h,w,char,fg=fg,bg=bg)
def color_area(self,row,col,h,w,fg=None,bg=None): #cambia il colore contenuto in un buffer area
    fg=fg if fg is not None else self._win['fg']
    bg=bg if bg is not None else self._win['bg']
    if self._win.valid_area(row,col,h,w): # se rientra nella finestra
        self._term.color_area(self._win['row']+row,self._win['col']+col,h-1,w-1,fg=fg,bg=bg)
def cpush(self): #salva i colori attuali
    self._cstack.append((self._win['active_fg'],self._win['active_bg']))
def cpop(self): #ripristina i colori
    if len(self._cstack)>0:
        c=self._cstack.pop()
        fg,bg=c
    else:
        fg,bg=LGREY,BLACK # stack vuoto colori di default
    self.set_color(self._win['fg'],self._win['bg']) # ripristina i colori
    self._win['active_fg']=fg; self._win['active_bg']=bg

```

Questa classe si aggancia agli oggetti Screen e Window e permette di utilizzare le coordinate relative alla finestra usata e con uno stack colori per ogni finestra.

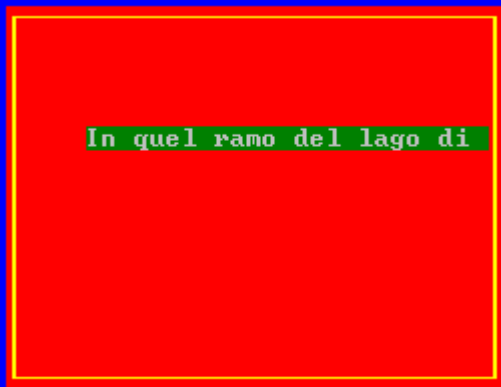
File test_window 02 : Esempio d'uso modulo Write della funzioni di repaint

```
from fbllib.fbConsole.Window import *
from fbllib.Costant import *
from fbllib.fbConsole.Write import *
term=fbConsole() #terminale
term.set_color(RED,LBLUE) #rosso su blu chiaro
term.cls() #cancella lo schermo

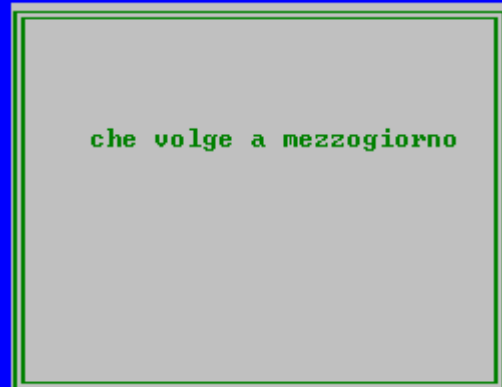
def repaint(fg,bg,text): #funzione di repaint della finestra
    win['fg']=fg # nuovo foreground
    win['bg']=bg #nuovo background
    win.paint() #ridisegna la finestra
    win.border() #cambia il tratto del bordo
    con.locate(5,10) #riscrivi il nuovo testo
    con.cprint(fg,bg,text)

win=Window(term,5,5,15,30,fg=YELLOW,bg=LRED,repaint=repaint)
con=Write(win)
#Utilizza le coordinate relative alla finestra
con.locate(5,10)
term.cprint(WHITE,LGREEN, "In quell ramo del lago di Como")
term.sleep(600) # ritardo
win.repaint(GREEN,WHITE,"che volge a mezzogiorno")
term.sleep(600)
```

Output 1



Output 2



Come si può notare nel primo output la frase è troppo lunga e quindi viene troncata, nel secondo caso la frase viene scritta per intero. Il modulo write tiene conto della dimensione della finestra.

File Draw.py : disegna cornici all'interno della finestra**passo 7**

```
from fblib.fbConsole import *
from fblib.fbConsole.Window import *
from fblib.fbTypes import *
from fblib.Costant import *
class Draw:
    def __init__(self, win):
        self._win=win
        self._term=win.term
    def box(self,row,col,h,w,relief='SINGLE',title=None): #disegna una cornice con il tratto desiderato
        if self._win.valid_area(row,col,h,w): # se rientra nella finestra
            self._term.box(self._win['row']+row,self._win['col']+col,h,w,relief=relief,title=title)
    def vseparator(self,row,col,h,relief='SINGLE'): #separatore verticale
        if (row+h) in range(1,self._win['height']):
            self._term.vseparator(self._win['row']+row,self._win['col']+col,h,relief=relief)
    def hseparator(self,row,col,w,relief='SINGLE'): #separatore orizzontale
        if (col+w) in range(1,self._win['width']):
            self._term.hseparator(self._win['row']+row,self._win['col']+col,w,relief=relief)
    def cseparator(self,row,col,relief='SINGLE'): #separatore centrale
        r1= row in range(1,self._win['height']);c1 =col in range(1,self._win['width'])
        if r1 and c1:
            self._term.cseparator(self._win['row']+row,self._win['col']+col,relief=relief)
    def rbox(self,row,col,nrow,width,fg=WHITE,bg=DGREY,relief='SINGLE',title="):
        r=nrow*2 ; h=r+2 #box con separatori orizzontali
        self._term.set_color(fg,bg)
        self.box(row,col,h,width,relief=relief,title=title)
        for i in range(2,h,2):
            self.hseparator(row+i,col,width,relief=relief)
```

File test_draw.py:

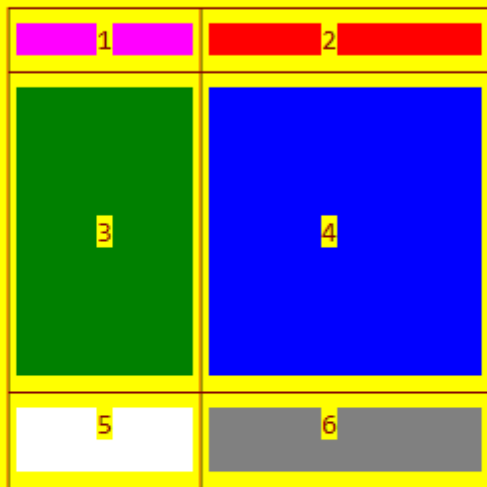
```
from Costant import *
from fblib.fbConsole.Draw import *
from fblib.fbConsole.Window import *
from fblib.fbConsole.Write import *

def test(relief):
    draw.box(5,5,15,30,relief=relief) # disegna il box
    draw.hseparator(7,5,30,relief=relief) # disegna il primo separatore orrizzontale
    draw.vseparator(5,17,15,relief=relief) # disegna il separatore verticale
    draw.cseparator(7,17,relief=relief) ) # disegna il primo incrocio
    draw.hseparator(17,5,30,relief=relief) # disegna il secondo separatore orrizzontale
    draw.cseparator(17,17,relief=relief) # disegna il secondo incrocio
    con.cpush() #salva I colori attuali
    con.clear_area(6, 6, 1,11,fg=YELLOW,bg=LPURPLE) # riempie un area con i colori indicati
    con.print_at(6,11,'1') #numera l'area
    con.clear_area(6,18 ,1,17,fg=WHITE,bg=LRED) # riempie un area con i colori indicati
    con.print_at(6,25,'2') #numera l'area
    con.clear_area(8, 6 ,9,11,fg=YELLOW,bg=GREEN) # riempie un area con i colori indicati
    con.print_at(12,11,'3') #numera l'area
    con.clear_area(8,18 ,9,17,fg=YELLOW,bg=LBLUE) # riempie un area con i colori indicati
    con.print_at(12,25,'4') #numera l'area
    con.clear_area(18,6, 2,11,fg=LGREY,bg=WHITE) # riempie un area con i colori indicati
    con.print_at(18,11,'5') #numera l'area
```

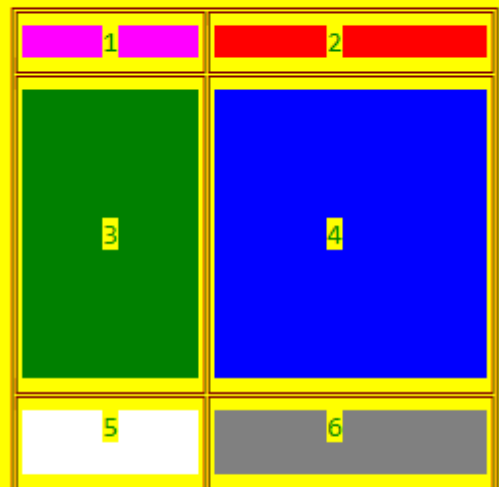
```
con.clear_area(18,18,2,17,fg=YELLOW,bg=DGREY) # riempie un area con i colori indicati
con.print_at(18,25,'6') #numera l'area
con.cpop() #ripristina i colori
```

```
term=fbConsole() # terminale
term.width(120, 50) # schermo 120 colonne x 50 righe
term.set_color(GREEN, PURPLE) # colori verde su porpora
term.cls() #cancella lo schermo
win=Window(term, 2, 10, 30, 40, g=RED, bg=YELLOW) # crea una finestra
c=con.cpush() #salva i colori
test('SINGLE') #chiamata 1 cornice tratto singolo
term.sleep(500) # attesa di un tasto o che passino 500 millisecondi
test('DOUBLE') #chiamata 2 cornice tratto doppio
term.sleep(500) #attesa di un tasto o che passino 500 millisecondi
con.cpop() #ripristina i colori
kill(win) #elimina la finestra
# eseguire da console
```

Esempio dimostrativo dei moduli Draw e Write: Output 1



Output 2



File Shadow.py: Ombra della finestra e delle cornici al suo interno passo 8

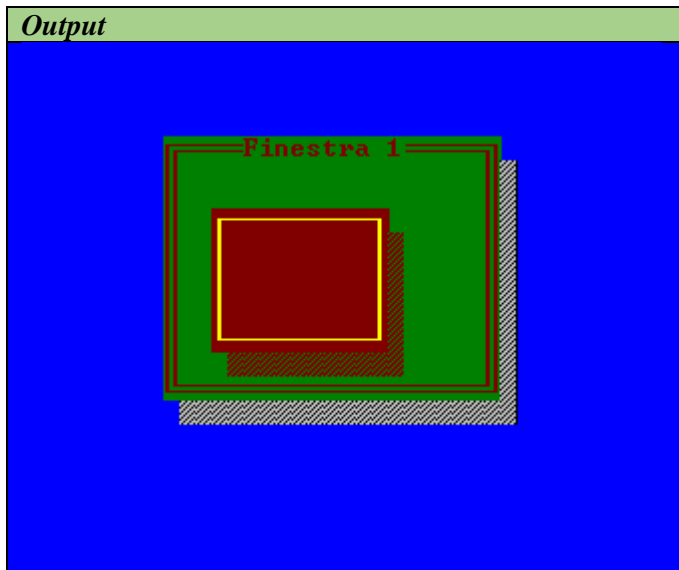
```
from fbllib.fbConsole.fbConsole import *
from fbllib.fbConsole.Window import *
from fbllib.fbTypes import *

class Shadow:
    def __init__(self,win):
        self._win=win
        self._term=win.term
    def shadow(self,row,col,h,w,fg=None,bg=None,syb=BLOCK3): # ombra all'interno della finestra
        fg=fg if fg is not None else self._win['fg']
        bg=bg if bg is not None else self._win['bg']
        c=self._term.color() # salva i colori attuali
        if self._win.valid_area(row,col,h,w): # se rientra nella finestra
            self._term.shadow(self._win['row']+row,self._win['col']+col,h,w,fg=fg,bg=bg,syb=syb)
        self._term.set_color(c[0],c[1]) # ripristina i colori
    def wshadow(self,fg=LGREY,bg=BLACK,syb=BLOCK3): # ombra della finestra
        c=self._term.color() # salva i colori attuali
        self._term.shadow(self._win['row'],self._win['col'],self._win['height'],self._win['width'],fg=fg,bg=bg,syb=syb)
        self._term.set_color(c[0],c[1]) # ripristina i colori
```

File test_shadow 01 .py: Esempio su ombreggiature della finestra

```
from fbllib.Costant import *
from fbllib.fbConsole.Draw import *
from fbllib.fbConsole.Shadow import *
from fbllib.fbConsole.Window import *
from fbllib.fbConsole.Write import *
term.set_color(WHITE,LBLUE)
term.cls()

win=Window(term,5,10,10,20,fg=RED,bg=GREEN,relief='DOUBLE',title='Finestra 1')
draw=Draw(win)
sh=Shadow(win)
con=Write(win)
con.clear_area(3,3,5,10,fg=YELLOW,bg=RED)
draw.box(3,3,5,10) #creo un box all'interno della finestra
sh.shadow(3,3,5,10) # ombra drl box
sh.wshadow() # ombra della finestra
term.sleep()
```



Ora andiamo a implementare la classe Widgets che ci permetterà di creare Label, Button e altro.

File Widgets.py

Passo 09

```
from fblib.fbConsole.fbConsole import *
from fblib.Costant import *
from fblib.fbConsole.WinBase import *
from fblib.fbConsole.Write import * from string import ascii_lowercase,ascii_uppercase,digits,punctuation
from abc import ABCMeta, abstractmethod

class Widgets(metaclass=ABCMeta):
    def __init__(self,parent,row,col,**kargs):
        self._parent=parent
        self._term=self._parent.term
        self._row,self._col=row,col
        self._var=kargs
        self._var['row']=row; self._var['col']=col #salva la posizione del widgets nel dizionario interno solo dopo,
                                                obbligando così a passarle come parametri formali

        if 'height' not in self._var: self._var['height']= 2 #se l'altezza non è stata dichiarata default 2
        if 'width' not in self._var : self._var['width']=10 #se la larghezza non è stata dichiarata default 10
        if 'text' not in self._var : self._var['text']=" #default txt nullo
        if 'fg' not in self._var : self._var['fg']=LGREY #foreground di default
        if 'bg' not in self._var : self._var['bg']=BLACK #background di default
        if 'active_fg' not in self._var : self._var['active_fg']=self._var['fg'] #active foreground di default
        if 'active_bg' not in self._var : self._var['active_bg']=self._var['bg'] #active background di default
        if 'visible' not in self._var : self._var['visible']=True #se visible non è indicato default True
        if 'relief' not in self._var: self._var['relief']='SINGLE' #cornice di default singolo tratto
        if 'command' not in self._var: # se la funzione di command non è indicata None
            self._command=None
        else: #registra la action funzione ed elimina la voce command dal dizionario
            self._command=self._var['command']; del(self._var['command'])
        if 'repaint' not in self._var: # se la funzione di repaint non è indicata None
            self._repaint=None
        else: #registra la funzione di repaint ed elimina la voce command dal dizionario
            self._repaint=self._var['repaint']; del(self._var['repaint'])
        self._buffer=self._term.newArea(self._var['height'],self._var['width']) # crea il buffer per il Widgets
        t=self._term.get(self._buffer, self._row_,self._col_) # salva l'area destinata alla finestra
```

Autore: Marco Salvati

Licenza Tutorial: CC BY 3.0

Email: marcosalvati61@gmail.com

Licenza Software: GPL V.3


```

self._buffer2=None
@property
def term(self): #ritorna il terminale
    return self._term
def __deallocateArea__(self):
    if self._var['visible']:
        t=self._term.put(self._buffer, self._row_,self._col_) #ripristina l'area
    else:
        self._term.deallocateArea(self._buffer2) #dealloca il buffer secondario
        self._term.deallocateArea(self._buffer) #dealloca il buffer primario
def hide(self): #nascondi il widget
    if self._isVisible():
        self._buffer2=self._term.newArea(self._var['height'],self._var['width']) # nuovo buffer
        t=self._term.get(self._buffer2, self._row_,self._col_) # salva la finestra
        t=self._term.put(self._buffer, self._row_,self._col_) # ripristina lo sfondo
        self._var['visible']=False
def show(self): #mostra il widget
    if not self._isVisible():
        t=self._term.put(self._buffer2, self._row_,self._col_) # ripristina la finestra
        self._term.deallocateArea(self._buffer2) # cancella il buffer
        self._buffer2=None
        self._var['visible']=True
def move(self,row,col): #sposta il widget
    self.hide()
    self._var['row']=row;self._var['col']=col # nuova posizione
    t=self._term.get(self._buffer, self._row_,self._col_) # salva l'area destinata al widgets
    self.show()
def isVisible(self): #Controlla se il widget è visibile o meno
    return self._var['visible']
def isDisable(self):
    return self._var['disable'] #Controlla se il widget è disabilitato
def paint(self): #disegna il widget
    row=self._row_-1
    col=self._col_-1
    self._term.clear_area(row,col,self._var['height'],self._var['width'],fg=self._var['fg'],bg=self._var['bg'])
    self._term.box(row,col,self._var['height'],self._var['width'],relief=self._var['relief'])
def repaint(self,*args,**kargs):
    if self._repaint is not None:self._repaint(*args,**kargs) # chiama la funzione di repaint
def action(self,*args,**kargs): # azione svolta dal widgets
    if self._command is not None:self._command(*args,**kargs) # chiama la funzione di associata
def __getitem__(self,key): #ritorna il valore di una variabile interna
    if key in self._var:
        return self._var[key]
    else:
        raise KeyError('Chiave non presente')
def __setitem__(self,key,value): #setta una variabile interna
    self._var[key]=value
def __delitem__(self,key):
    if key in self._var: del(self._var[key]) # cancella la chiave se presente
def border(self): # cambia il bordo del widget
    self._var['relief']='DOUBLE' if self._var['relief']=='SINGLE' else 'SINGLE'
    self.paint()
def is_mouse_over(self): # se il mouse è sopra il widget ritorna True
    row=self._parent['row']+self._var['row']-1

```

```

col=self._parent['col']+self._var['col']-1
return self._term.mouse_over(row,col,self._var['height'],self._var['width'])
def is_mouse_click(self,button): # se il mouse è cliccato sul widget ritorna True
row=self._row_-2
col=self._col_
return self._term.mouse_click(button,row,col,self._var['height'],self._var['width'])
@property
def _row_(self): #Uso interno posizione riga reale del widget dello schermo
return self._parent['row']+self._var['row']
@property
def _col_(self): #Uso interno posizione colonna reale del widget dello schermo
return self._parent['col']+self._var['col']
def reverse(self): #inverti I colori del widget
if not self.isDisable() # se non disabilitata
tmp,self._var['active_fg']=self._var['active_bg'],self._var['active_bg']
self._var['active_bg']=tmp
self.paint()
def restore(self): #ripristina I colori di default
self._var['active_bg']=self._var['bg']
self._var['active_fg']=self._var['fg']

```

Ora andiamo a introdurre I widgets più utilizzati.

- Label
- Button
- Entry
- Menu

Il Widget Label

File Label.py:

Passo 10

```
from fblib.fbConsole.Widgets import *
class Label(Widgets):
    def __init__(self,parent,row,col,**kargs):
        super().__init__(parent,row,col,**kargs)
        self._con=Write(self._parent) #associa il modulo Write al parent (Screen o Window)
        self._border=False; self._buff=None #default border False buffer area bordo None
    def paint(self): #disegna il widget si sovrappone al metodo della classe madre
        if self._isVisible(): #se la label è visibile
            fg,bg=self._color_() #ritorna i colori memorizzati
            self._con.set_color(fg,bg) #attiva i colori
            txt=self._var['text'] #ritorna il testo della label
            #stampa il testo con max larghezza pari a width
            self._con.print_at(self._var['row'],self._var['col'],txt[:self._var['width']])

    def border(self): #disegna un bordo intorno alla Label si sovrappone al metodo della classe madre
        if not self._border:
            self._buff=self._term.newArea(self._var['height'],self._var['width']+1)
            self._term.get(self._buff,self._row-1,self._col-1) #salva l'area
            self._border=True
        self._var['relief']='DOUBLE' if self._var['relief']=='SINGLE' else 'SINGLE'
        self._term.box(self._row-1,self._col-1,self._var['height'],self._var['width']+1,relief=self._var['relief'])
        self.paint()

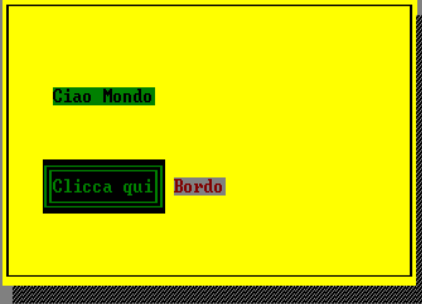
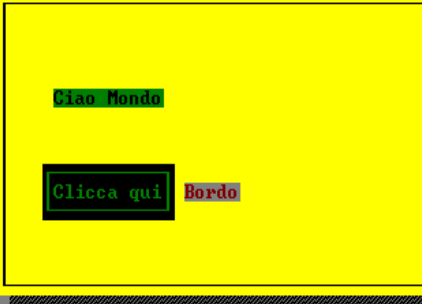
    def clear_border(self): #cancella il bordo della label
        if self._border:
            self._term.put(self._buff,self._row-1,self._col-1) #ripristina l'area
            self._term.deallocateArea(self._buff) #elimina il buffer per il bordo
            self._border=False #nessun bordo presente
            self.paint() #ridisegna la Label
```

#Label esempio

```
from fbllib.Costant import *
from fbllib.fbConsole.fbConsole import *
from fbllib.fbConsole.Draw import *
from fbllib.fbConsole.Label import Label
from fbllib.fbConsole.Write import *
from fbllib.fbConsole.Screen import Screen
from fbllib.fbConsole.Shadow import *
term=fbConsole()
scr=Screen(term,fg=RED,bg=DGREY) #Oggetto Screen
win=Window(term,5,20,15,40,fg=BLACK,bg=YELLOW) #Finestra
con=Write(win) #strumenti Scrittura per la finestra
con2=Write(scr) # strumenti Scrittura per l'oggetto Screen
con2.set_color(BLACK,GREEN) #colori attivi
sh=Shadow(win) #strumento per creare ombreggiature intorno o nella finestra
sh.wshadow(fg=BLACK,bg=DGREY) #crea un ombra intorno alla finestra
con.locate(5,5);con.cprint(BLACK,GREEN,'Ciao Mondo') # Scrive qualcosa nella finestra
term.cursor_off() # cursore invisibile
l1=Label(win,10,5,text="Clicca qui",fg=BLACK,bg=GREEN) #etichetta nella finestra
l2=Label(scr,24,20,text='Premi un tasto per nascondere la label',width=37,fg=BLACK,bg=GREEN) #etichetta sullo schermo
term.sleep() #ritardo
l1.hide() #nasconde l'etichetta nella finestra
l2['text']='Premi un tasto per farla riapparire ' #cambia il testo nell'etichetta sullo schermo
l2.paint() #ridisegna l'etichetta fuori la finestra
term.sleep()
kill(l2) #elimina l'etichetta sullo schermo
l1.show() #mostra l'etichetta nella finestra
con.set_color(RED,DGREY) #colori della scritta Bordo
con.print_at(10,17,'Bordo')
con2.print_at(24,20,"Passa il mouse qui sopra"); s=''
while True: #mainloop
    c=term.getmouse() #leggi il mouse
    s=term.inkey() #leggi la tastiera
    if s==I_ENTER. # controllo keyboard se s==Enter esci
        term.cursor_on() #cursore visibile
        break
    elif s==I_LEFT: #tasto left cambia il bordo della finestra
        win.border()
    elif s==I_RIGHT: #tasto rigth cambia il bordo della finestra
        win.border()
    elif scr.mouse_over(24,20,1,25): #mouse su un area dell'oggetto Screen
        con2.reverse();con2.print_at(24,20,"Passa il mouse qui sopra")
    else:
        if c==0:
            if l1.is_mouse_click(1): #mouse click pulsante 1 sulla Label 1
                l1.reverse() #inverte i colori della Label
            elif win.mouse_click(2,10,17,3,10): #mouse click pulsante destro (2) sul testo Bordo
                l1.border() #cambia il bordo della Label se non presente lo crea
            elif win.mouse_click(1,10,17,3,10): #mouse click pulsante sinistro (1) sul testo Bordo
                l1.clear_border():
```

Label Esempio completo

<p><i>01 Per nascondere l'etichetta premere un tasto</i></p>	<p><i>02 Per rivisualizzare l'etichetta premere un tasto</i></p>
 <p>Premi un tasto per nascondere la label</p>	 <p>Premi un tasto per farla riapparire</p>
<p><i>03 Posizionati sulla scritta passa il mouse qui sopra</i></p>	<p><i>04 La scritta inverte i colori ogni volta</i></p>
 <p>Passa il mouse qui sopra</p>	 <p>Passa il mouse qui sopra</p>
<p><i>05 Click sinistro sulla scritta clicca qui</i></p>	<p><i>06 Click pulsante destro sul testo Bordo</i></p>
 <p>Passa il mouse qui sopra</p>	 <p>Passa il mouse qui sopra</p>

07 Click pulsante destro sul testo Bordo cambia cornice	08 Click pulsante sinistro sulla Label clicca qui
	
09 Click pulsante destro sulla Label Bordo	10 Click pulsante sinistro sulla Label Bordo
	
11 Premendo Left o Right cambia il bordo della finestra	12 Cliccando enter si esce dal loop
	

La classe Button

La classe Button si limita a disegnare dei pulsanti, la classe madre Widget ereditata fornisce i metodi per l'utilizzo.

File Button.py	Passo 11
<pre>From fbllib.fbConsole.Widgets import * class Button(Widgets): def __init__(self,parent,row,col,**kargs): super().__init__(parent,row,col,**kargs) self._con=Write(self._parent) if not 'justify' in self._var:self._var['justify']='center' <i>#default testo giustificato al centro</i> self.paint() def paint(self): super().paint() <i>#chiama il metodo della classe madre per disegnare il box</i> t=self._parent.ajust(self._var['text'],self._var['width']-3,self._var['justify']) <i>#giustifica il testo</i> fg,bg=self._color_() <i>#ritorna i colori adeguati</i> self._con.set_color(fg,bg) <i>#li attiva</i> self._con.print_at(self._var['row'],self._var['col']+1,t) <i>#scrive il testo</i></pre>	

Esempio completo.


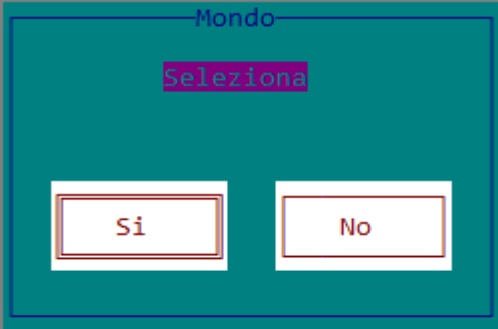

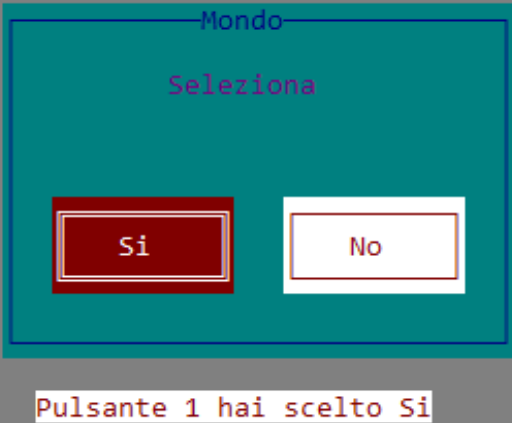
File test_button 01.py
<pre>from fbllib.fbConsole.fbConsole import * from fbllib.fbConsole.Window import * from fbllib.fbConsole.Widgets import * from fbllib.Costant import * from fbllib.fbConsole.Label import * from fbllib.fbConsole.Button import * term=fbConsole() def bt1(): <i>#azione button1</i> term.print_at(30,2, 'Pulsante 1 hai scelto Si') b1.reverse() def bt2(): <i>#azione button2</i> term.print_at(30,2, 'Pulsante 2 hai scelto No') b2.reverse() term.cursor_off() <i># cursore invisibile</i> win=Window(term,10,10,10,30,fg=BLUE,bg=CYAN,repaint=None,title='Mondo') l1=Label(win,2,10,text='Ciao Mondo',fg=PURPLE,bg=CYAN) b1=Button(win,7,4,text='Si',fg=RED,bg=WHITE,command=bt1) b1.border() <i># bordo doppio opzione di default</i> b2=Button(win,7,18,text='No',fg=RED,bg=WHITE,command=bt2) x=1 while True: <i># un semplice gestore mouse e tastiera</i> s=term.inkey() <i># controllo keyboard</i> m=term.getmouse() <i>#controllo il mouse</i> if s==I_ENTER: <i>#tasto enter</i> term.cursor_on() <i>#cursore visibile</i></pre>


```

    break #esci
elif s==I_LEFT or s==I_RIGHT: #tasto left o right
    if x==1: #cambia pulsante
        x += 1
        b1.border() #inverte i bordi
        b2.border()
    elif x==2: #cambia pulsante
        x -= 1
        b1.border() #inverte i bordi
        b2.border()
elif l1.is_mouse_over(): #se il mouse è sopra la Label
    l1.reverse() #inverte i colori della Label
else: #altri controlli del mouse
    if m==0:
        if b1.is_mouse_click(1): #click su Button 1
            if x==2:
                b1.border() #inverte I bordi
                b2.border()
                b1.action() #chiama la funzione associata
                x=1
                break
            elif b2.is_mouse_click(1): #click su Button 2
                if x==1:
                    b2.border() #inverte I bordi
                    b1.border()
                    b2.action() #chiama la funzione associata
                    x=2
                    break
    term.sleep() #necessario solo per la visualizzazione del messaggio

```

Vediamo il test output :

Passate il mouse sulla scritta seleziona	Il testo inverte il colore
	
I tasti left e right cambiano la selezione	Click del mouse o enter selezionano
	

Ora vediamo il classe Entry per l' input da tastiera:

File Entry.py	passo 12
<pre> from fbllib.fbConsole.Widgets import * class Entry(Widgets): def __init__(self,parent,row,col,**kargs): super().__init__(parent,row,col,**kargs) self._con=Write(parent) <i>#associa il modulo write al parent</i> self._index=0 if "relief" not in self._var: self._var["relief"]=None <i>#default nessun bordo</i> if not 'multi_edit' in self._var: self._var['multi_edit']=False <i># default single edit</i> self._row2=self._var['row'] if self._var['relief'] is None else self._var['row']+1 self._con.set_color(self._var['fg'],self._var['bg']) self._con.print_at(self._var['row'],self._var['col'],'*self._var['width']) l=len(self._var['text']) if l>0:self._con.print_at(self._var['row'],self._var['col'],self._var['text']);self._index=l-1 def paint(): <i>#riscrive il metodo paint per adeguarlo</i> if self._var["relief"] is not None: row=self._row_-1 col=self._col_-1 fg,bg=self._color_() self._term.clear_area(row,col,self._var['height'],self._var['width']+1,fg=fg,bg=bg) self._term.box(row,col,self._var['height'],self._var['width']+1,relief=self._var['relief']) def _delFirstchar(self): <i>#cancella il primo carattere</i> </pre>	

Autore: Marco Salvati
Licenza Tutorial: CC BY 3.0

Email: marcosalvati61@gmail.com
Licenza Software: GPL V.3

```

self._var['text']=self._var['text'] [1:]
def _delLastchar(self): #cancella l'ultimo carattere
    self._var['text']=self._var['text'][:-1]
def _delMiddlechar(self): #cancella il carattere nella posizione index-1
    l=self._var['text'][:self._index-1]
    r=self._var['text'][self._index:]
    self._var['text']=l+r
def _delActualPos(self): #cancella il carattere nella posizione attuale
    l=self._var['text'][:self._index]
    r=self._var['text'][self._index+1:]
    self._var['text']=l+r
def _insertFirstchar(self,s): #inserisci un carattere nella prima posizione
    if len(self._var['text'])< self._var['width']:
        if self._command is not None: #funzione di controllo
            s=self._command(s)
        self._var['text']=s+self._var['text']
def _insertLastchar(self,s): #inserisci un carattere nell'ultima posizione
    if len(self._var['text'])< self._var['width']:
        if self._command is not None: #funzione di controllo
            s=self._command(s)
        self._var['text']+=s
def _insertMiddlechar(self,s): #inserisci un carattere in posizione centrale
    if len(self._var['text'])< self._var['width']:
        if self._command is not None: #funzione di controllo
            s=self._command(s)
        l=self._var['text'][:self._index]
        r=self._var['text'][self._index:]
        self._var['text']=l+s+r
def input(self): #esegue l'input
    fg,bg=self._color_()
    self._con.set_color(fg,bg)
    self._con.print_at(self._var['row'],self._var['col'],'*self._var['width'])
    self._con.print_at(self._var['row'],self._var['col'],self._var['text'])
    mode=I_INS
    char=ascii_lowercase+ascii_uppercase+digits
    while True:
        i= self._index if self._index <= self._var['width'] else self._var['width']
        self._con.locate(self._var['row'],self._var['col']+i)
        s=self._term.getchar() # attende la pressione di un tasto
        if not self.isDisable():
            if (s in char or s in punctuation ):
                if mode==I_INS: #inserimento
                    if self._index==0 and len(self._var['text'])<=self._var['width']: #in testa
                        self._insertFirstchar(s)
                        self._index+=1
                    elif self._index==len(self._var['text'])and len(self._var['text'])<self._var['width']: #in coda
                        self._insertLastchar(s)
                        self._index+=1
                    elif self._index<len(self._var['text']) and len(self._var['text'])<self._var['width']: # al centro
                        self._insertMiddlechar(s)
                        self._index+=1
                elif mode=='NoIns':
                    if self._index==0:
                        self._delFirstchar()

```

```

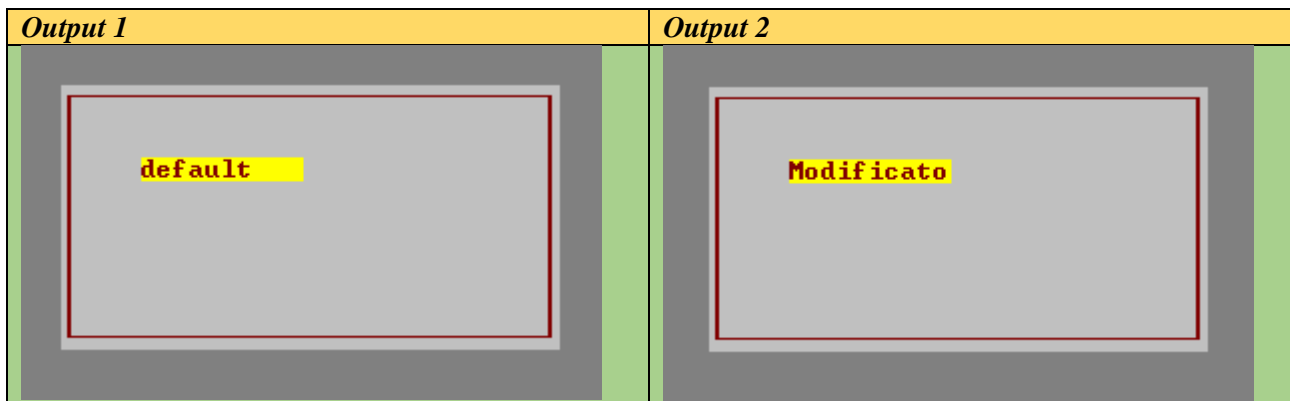
        self._insertFirstchar(s)
    elif self._index==len(self._var['text']):
        self._delLastchar()
        self._insertLastchar(s)
    else:
        self._delMiddlechar()
        self._insertMiddlechar(s)
elif s==I_INS: ### INS
    mode=I_INS if mode!= I_INS else 'NoIns'
elif s==I_BACKSPACE: ### BACKSPACE
    if self._index==0:
        self._delFirstchar()
        self._index=1
    elif self._index==len(self._var['text']):
        self._delLastchar()
        self._index-=1
    else:
        self._delActualPos()
        self._index-=1
elif s==I_CANC: ### CANC
    if self._index==0:
        self._delFirstchar()
    elif self._index==len(self._var['text']):
        self._delLastchar()
    elif self._index <len(self._var['text']):
        self._delActualPos()
elif s==I_LEFT: ### LEFT
    if self._index in range(1,self._var['width']+1): self._index-=1
elif s==I_RIGHT : ### RIGHT
    if self._index in range(self._var['width']+2): self._index+=1
elif s==I_HOME: ### HOME
    self._index=0
elif s==I_END: ### END
    self._index=len(self._var['text'])-1
if s==I_ENTER: ### ENTER esci input terminato
    return I_ENTER
elif s==I_TAB: # TAB esci widget successive se multi_edit
    if self._var['multi_edit']: return I_TAB
elif s==I_UP: # Up esci input precedente se multi_edit
    if self._var['multi_edit']: return I_UP
elif s==I_DOWN: # Down esci input successive se multi_edit
    if self._var['multi_edit']: return I_UP
elif s==I_ESCAPE: # I_ESCAPE
    if self._var['multi_edit']: return I_ESCAPE #esci se multiedit
self._con.print_at(self._var['row'],self._var['col'],' '*self._var['width'])
self._con.print_at(self._var['row'],self._var['col'],self._var['text'][:self._var['width']])

```

Questa classe implementa una funzione di input di lunghezza massima voluta, la modalità di input può essere a campo singolo o multi input ciò permetterebbe di scorrere tra vari i entry.

Esempi di uso della classe Entry a input singolo

```
from fblib.fbConsole import *
from fblib.fbConsole.Window import *
from fblib.fbConsole.Entry import *
from fblib.fbConsole.Write import *
term=fbConsole()
win=Window(term,5,5,10,30,fg=RED,bg=LGREY)
e1=Entry(win,3,5,text='default',fg=RED,bg=YELLOW)
e1.input()
term.sleep()
```



Ora andiamo a creare una classe Medit che ci permetta di gestire input multipli.

Medit è un gestore di eventi per tastiera e mouse per Entry multipli , funge anche da sotto gestore per la classe Event che motrata in seguito..

```

from fbllib.fbConsole.Entry import *

class MEdit:
    def __init__(self):
        self._stack,self._i=[],0
    def add(self,entry):
        entry['multi_edit']=True #setta la modalita multi edit
        self._stack.append(entry) #memorizza l'entry
    def is_mouse_over(self): #ritorna True se il mouse è sopra su una delle Entry
        for i,o in enumerate(self._stack):
            if o.is_mouse_over(): self._i=i;return True
        return False
    def is_mouse_click(self,button): #ritorna True se il mouse è stato cliccato sopra su una delle Entry
        for i,o in enumerate(self._stack):
            if o.is_mouse_click(button): self._i=i;return True
        return False
    def action(self): #avvia il loop
        self._loop=True
        while self._loop:
            c=self._stack[self._i].input()
            if c==I_UP: #Entry precedente
                if self._i==0: self._i=len(self._stack)-1
                else: self._i-=1
            elif c==I_DOWN or c==I_TAB: #Entry successivo
                if self._i==len(self._stack)-1:self._i=0
                else: self._i+=1
            elif c==I_ESCAPE or c==I_ENTER: #uscita dal loop
                self._loop=False
                return c #codice di uscita

```

Prima di mostrare un esempio introduciamo la classe Event per la gestione di eventi mouse e tastiera

```

EV_OVER = 0          #Mouse sopra la finestra
EV_IS_OVER= 1        # Mouse sopra wigets
EV_CLICK= 2          #Click mouse sopra la finestra
EV_IS_CLICK=3        #Click mouse sopra il widget
EV_KEYBOARD=4        #Pressione hotkey

from fbConsole import *
from threading import Thread

class Event:
    def __init__(self,term):
        self._term=term
        self._t=Thread(target=self._loop_)
        self._running = True
        self._stack=[]
        self._index=0
    def terminate(self): #esci dal loop
        self._running = False
    def mouse_over(self,obj,row,col,h,w,*args,**kargs): #posizione del mouse su un area della finestra
        self._stack.append((EV_OVER,obj,row,col,h,w,args,kargs))
    def is_mouse_over(self,obj,*args,**kargs): #posizione del mouse sull'oggetto
        self._stack.append((EV_IS_OVER,obj,args,kargs))
    def mouse_click(self,obj,but,row,col,h,w,*args,**kargs): #click del mouse su un area della finestra
        self._stack.append((EV_CLICK,obj,but,row,col,h,w,args,kargs))
    def is_mouse_click(self,obj,but,*args,**kargs): #click del mouse sull' oggetto
        self._stack.append((EV_IS_CLICK,obj,but,args,kargs))
    def keyboard(self,obj,sel,key,*args,**kargs): # scorciatoie da tastiera Selettore + tasto ( esempio alt-F)
        self._stack.append((EV_KEYBOARD,obj,sel,key,args,kargs))
    def mainloop(self): #avvia il gestore degli eventi
        self._t.start()
    def _loop_(self): #loop principale
        while self._running:
            self._term.getmouse()
            op=self._stack[self._index]
            if op[0]== EV_OVER:
                if op[1].mouse_over(op[2],op[3],op[4],op[5]): #se è sopra l'area
                    op[1].repaint(*op[6],**op[7])
            elif op[0]== EV_IS_OVER: #se è sopra al widget
                if op[1].is_mouse_over():
                    op[1].action(*op[2],**op[3])
            elif op[0]== EV_CLICK:
                if op[1].mouse_click(op[2],op[3],op[4],op[5],op[6]): #click sopra l'area
                    op[1].repaint(*op[6],**op[7])
            elif op[0]== EV_IS_CLICK: #click sull'oggetto
                if op[1].is_mouse_click(op[2]):
                    op[1].action(*op[3],**op[4])
            elif op[0]== EV_KEYBOARD: #hotkey
                if self._term.multikey(op[2]) and self._term.multikey(op[3]):
                    op[1].action(*op[4],**op[5])
            self._index+=1
            if self._index<0: self._index=len(self._stack)-1
            elif self._index>=len(self._stack): self._index=0

```


File test_event.py

```
from fbConsole import *
from Screen import *
from Window import *
from Costant import *
from Label import *
from Button import *
from Entry import *
from MEdit import *
from Event import *

def change_color(fg,bg): # Cambia I colori alla Label l1
    l1['fg']=fg
    l1['bg']=bg
    l1.paint()

def reverse(): # inverte i colori alla Label l2
    l2.reverse()

def reverse2(): # inverte i colori alla Label l1 repaint dell'oggetto Screen
    l1.reverse()

def bt(but): # chiamata dalla pressione dei pulsanti
    term.print_at(20,10,f'Pulsante {but}')

def esci(): #Chiamata dal Pulsante Esci
    ev.terminate()

term=fbConsole()
ev=Event(term) #definisce il gestore di eventi
le=MEdit() # definisce il gestore di Entry
scr=Screen(term,fg=WHITE,bg=LBLUE,repaint=reverse2) #oggetto Screen
l1=Label(scr,5,5,text='Hello Word',fg=RED,bg=GREEN,command=change_color,visible=True)
l2=Label(scr,5,35,text='Ciao ciao',fg=RED,bg=GREEN,command=reverse,visible=True)
b1=Button(scr,10,5,width=15,text='Pulsante 1',fg=BLACK,bg=GREEN,command=bt)
b2=Button(scr,10,30,width=15,text='Pulsante 2',fg=BLACK,bg=GREEN,command=bt)
b3=Button(scr,10,50,width=15,text='Esci',fg=BLACK,bg=GREEN,command=esci)
e1=Entry(scr,14,5,text='Pippo',fg=RED,bg=YELLOW,relief=None) #Crea una Entry con un testo di default
le.add(e1) #associa l'entry al gestoreMEdit
e2=Entry(scr,16,5,text='De Pippis',fg=RED,bg=YELLOW relief=None) #Crea una Entry con un testo di default
le.add(e2) #associa l'entry al gestoreMEdit
ev.keyboard(l1,SC_CONTROL,SC_L ,fg=YELLOW,bg=RED) # cambia colore alla label control-L
ev.keyboard(b1,SC_CONTROL,SC_1 ,1) # scorciatoia pulsante 1 control-1
ev.keyboard(b2,SC_CONTROL,SC_2 ,2) # scorciatoia pulsante 2 control-2
ev.keyboard(b3,SC_CONTROL,SC_E) # scorciatoia pulsante Esci control-E
ev.is_mouse_click(b1,1,1) #click pulsante 1
ev.is_mouse_click(b2,1,2) #click pulsante 1
ev.is_mouse_over(l2) #mouse sopra all'etichetta 2 reverse
ev.is_mouse_click(b3,1) #click pulsante Esci
ev.mouse_over(scr,5,5,2,10) #mouse sopra all'etichetta 2 reverse posizione assoluta
ev.is_mouse_click(le,1) #click su un Entry nel contenitore
ev.mainloop() #event loop
```

Output 1

Hello Word Ciao ciao

Pulsante 1 Pulsante 2 Esci

Pippo

De Pippis

Testiamo I vari eventi del mouse e di tastiera

Output 2 premendo il Pulsante 1

Hello Word Ciao ciao

Pulsante 1 Pulsante 2 Esci

Pippo

De Pippis

Pulsante 1

*Click su Pulsante 1
O premendo control-1*

Output 3 premendo il Pulsante 2

Hello Word Ciao ciao

Pulsante 1 Pulsante 2 Esci

Pippo

De Pippis

Pulsante 2

*Click su Pulsante 2
O premendo control-2*

Output 4 Mouse over		
<div> <div>Hello Word</div> <div>Ciao ciao</div> <div> <div>Pulsante 1</div> <div>Pulsante 2</div> <div>Esci</div> </div> <div> <div>Pippo</div> <div>De Pippis</div> </div> <div>Pulsante 2</div> </div>	<div> <div>Posizionando il mouse sopra le Label invertiamo i colori.</div> </div>	

Output 5 Eventi da tastiera		
<div> <div>Hello Word</div> <div>Ciao ciao</div> <div> <div>Pulsante 1</div> <div>Pulsante 2</div> <div>Esci</div> </div> <div> <div>Pippo</div> <div>De Pippis</div> </div> <div>Pulsante 2</div> </div>	<div> <div>Premendo control-L Settiamo i nuovi colori Della Label 1.</div> </div>	

Output 6 Entry		
<div> <div>Hello Word</div> <div>Ciao ciao</div> <div> <div>Pulsante 1</div> <div>Pulsante 2</div> <div>Esci</div> </div> <div> <div>Mickey</div> <div>Mouse</div> </div> <div>Pulsante 2</div> </div>	<div> <div>Click su un entry per editare, tasti UP entry precedente, DOWN o TAB successivo, ENTER o ESCAPE per uscire e tornare al gestore principale</div> <div>Clickando su Esci o con control-E si esce dal loop principale e termina.</div> </div>	

Ora andiamo a creare una classe che gestisca menu verticali.

File Vmenu.py: crea un menu verticale

passo 15

```
from fblib.fbConsole.Screen import Screen
from fblib.fbConsole.Write import *
from fblib.fbTypes import *
from fblib.Costant import *
from fblib.fbConsole.Widgets import *
from fblib.fbConsole.Label import *
class Vmenu:
    def __init__(self,parent,row,col,**kargs):
        self._stack,self._ind=[],0
        self._parent=parent
        self._row=row
        self._col=col
        self._term=parent.term
        self._var=kargs
        self._var['visible']=False #forza non visibile
        if "escape" not in self._var: self._var["escape"]=False
    def is_mouse_over(self): #se il mouse è su un etichetta
        for i,o in enumerate(self._stack):
            if o.is_mouse_over(): return True
        return False
    def is_mouse_click(self,button): #click su un etichetta
        for i,o in enumerate(self._stack):
            if o.is_mouse_click(button):
                self._stack[self._ind].reverse()
                o.reverse()
                self._ind=i
                return True
        return False
    def choice(self,label,func): #creazione di una voce
        self._var['text']=label
        self._var['command']=func
        self._stack.append(Label(self._parent,self._row+self._ind,self._col,**self._var))
        self._ind+=1
    def space(self):
        self._ind+=1
    def show(self):
        for l in self._stack:
            l.show()
    def hide(self):
        for l in self._stack:
            l.hide()
    def paint(self):
        for l in self._stack:
            l['visible']=True;
            l.paint()
    def select(self,dummy=0):
        self.paint()
        self._ind=0 #resetta indice
        self._stack[self._ind].reverse()
        loop=True
```

Autore: Marco Salvati

Licenza Tutorial: CC BY 3.0

Email: marcosalvati61@gmail.com

Licenza Software: GPL V.3

```

while loop:
    m=self._term.getmouse() # controllo del mouse
    s=self._term.inkey() # controllo keyboard
    if s==I_ESCAPE: # se escape è abilitato esci dal menu senza alcuna selezione
        if self._var["escape"]==True:
            loop=False
            self.hide()
    elif s==I_ENTER: # conferma con enter
        self._stack[self._ind].reverse()
        self._stack[self._ind].action(self._ind) # esegui
        self._stack[self._ind].reverse()
        if self._var['escape']==True:
            self._stack[self._ind].reverse()
            self.hide()
            loop=False
    elif s==I_LEFT or s==I_RIGHT: # se escape è abilitato con left o right esci senza alcuna selezione
        if self._var['escape']==True:
            self._stack[self._ind].reverse()
            self.hide()
            loop=False
    elif s==I_UP: # up voce precedente
        if self._ind > 0:
            self._stack[self._ind].reverse()
            self._ind-=1
            self._stack[self._ind].reverse()
    elif s==I_DOWN: # up voce successiva
        if self._ind < len(self._stack):
            self._stack[self._ind].reverse()
            self._ind+=1
            if self._ind==len(self._stack):
                self._ind=0
                self._stack[self._ind].reverse()
            else:
                self._stack[self._ind].reverse()
    elif s==I_HOME: # home prima voce
        self._stack[self._ind].reverse()
        self._ind=0
        self._stack[self._ind].reverse()
    elif s==I_END: # end ultima voce
        self._stack[self._ind].restore()
        self._ind=len(self._stack)- 1
        self._stack[self._ind].reverse()
    else:
        if self.is_mouse_click(1): # se click del mouse su un etichetta
            self._stack[self._ind].action(self._ind)
            if self._var['escape']==True:
                self._stack[self._ind].reverse()
                self.hide()
                loop=False

```

Testiamo la classe Vmenu.

File test Vmenu.py

```
from fplib.fbConsole.Screen import Screen
from fplib.Costant import *
from fplib.fbConsole.Widgets import *
from fplib.fbConsole.Draw import *
from fplib.fbConsole.Vmenu import *

def test1(but): # funzione chiamata dal click o da enter sulla voce
    term.print_at(24,10,f'Selezionato la Voce n.{but+1}')

term=fbConsole() # terminale
scr=Screen(term,fg=WHITE,bg=LBLUE) #Oggetto schermo

draw=Draw(scr) # classe di disegno per l'oggetto Screeen
draw.rbox(9,9,5,11,fg=RED,bg=LBLUE,relief='DOUBLE')) # cornice delle voci del menu

m=Vmenu(scr,10,12,fg=RED,bg=YELLOW,escape=True) #definisce un menu con escape si può terminare
m.choise('Voce 1',test1) # definisce una voce
m.space() # salta di una riga
m.choise('Voce 2',test1)
m.space()
m.choise('Voce 3',test1)
m.space()
m.choise('Voce 4',test1)
m.space()
m.choise('Voce 5',test1)
m.space()
m.choise('Voce 6',test1)
m.end() #fine inserimento voci
m.select() #attiva la selezione
```

Output 1	Output 2
	 <p>Selezionato la Voce n.2</p>

Da tastiera, UP voce precedente
Down voce successiva, Enter o mouse click sulla voce conferma.
Se Escape è True esci dal menu annullando la scelta.

Di conseguenza, la classe Hmenu gestisce menu orizzontali.

```

from fblib.fbConsole.Screen import Screen
from fblib.fbConsole.Write import *
from fblib.fbTypes import *
from fblib.Costant import *
from fblib.fbConsole.Widgets import *
from fblib.fbConsole.Label import *

class Hmenu:
    def __init__(self,parent,row,col,**kargs):
        self._stack, self._ind=[],0
        self._parent=parent
        self._row=row
        self._col=col
        self._term=parent.term
        self._var=kargs
        self._var['visible']=False # forza non visibile
        if "escape" not in self._var: self._var["escape"]=False
    def is_mouse_over(self): # se il mouse è su un etichetta
        for i,o in enumerate(self._stack):
            if o.is_mouse_over(): return True
        return False
    def is_mouse_click(self,button): # click su un etichetta
        for i,o in enumerate(self._stack):
            if o.is_mouse_click(button):
                self._stack[self._ind].reverse()
                o.reverse()
                self._ind=i
                return True
        return False
    def choise(self,label,func): # creazione voci
        self._var['text']=label
        self._var['command']=func
        self._var['width']=len(label)
        self._stack.append(Label(self._parent,self._row,self._col+self._ind,**self._var))
        self._ind+=len(label)+2
    def space(self): # salta di un posto
        self._ind+=1
    def paint(self): # visualizza tutte le label
        for l in self._stack:
            l['visible']=True;
            l.paint()
    def select(self,dummy=0): # inizio selezione
        self.paint()
        self._ind=0 # indice
        self._stack[self._ind].reverse()
        loop=True
        while loop:
            m=self._term.getmouse() # controllo del mouse
            s=self._term.inkey() # controllo keyboard
            if s==I_ESCAPE: # se escape è abilitato esci senza selezione
                if self._var["escape"]==True:
                    loop=False
            elif s==I_ENTER: # conferma con enter

```



```

        self._stack[self._ind].reverse()
        self._stack[self._ind].action(self._ind) #esegui
        self._stack[self._ind].reverse()
    elif s==I_DOWN: # se down esegui come se fosse premuto enter ma lasciando selezionata
        self._stack[self._ind].action(self._ind) #esegui
    elif s==I_LEFT: #left voce precedente
        if self._ind > 0:
            self._stack[self._ind].reverse()
            self._ind-=1
            self._stack[self._ind].reverse()
        else:
            self._stack[self._ind].reverse()
            self._ind=len(self._stack)-1
            self._stack[self._ind].reverse()
    elif s==I_RIGHT: #right voce successiva
        if self._ind < len(self._stack):
            self._stack[self._ind].reverse()
            self._ind+=1
            if self._ind==len(self._stack):
                self._ind=0
                self._stack[self._ind].reverse()
            else:
                self._stack[self._ind].reverse()
    elif s==I_HOME: #home prima voce
        self._stack[self._ind].reverse()
        self._ind=0
        self._stack[self._ind].reverse()
    elif s==I_END: #end ultima voce
        self._stack[self._ind].reverse()
        self._ind=len(self._stack)- 1
        self._stack[self._ind].reverse()

```

File test Hmenu.py

```

from fblib.fbConsole.Screen import Screen
from fblib.fbConsole.Write import *
from fblib.Costant import *
from fblib.fbConsole.Widgets import *
from fblib.fbConsole.Label import *
from fblib.fbConsole.Hmenu import *
def test1(but):
    term.print_at(24,10,f'Selezionato la Voce n.{but+1}')
term=fbConsole()
scr=Screen(term,fg=WHITE,bg=LBLUE)
m=Hmenu(scr,10,10,fg=RED,bg=YELLOW,escape=True)
m.choise('Voce 1',test1)
m.choise('Voce 2',test1)
m.choise('Voce 3',test1)
m.choise('Voce 4',test1)
m.choise('Voce 5',test1)
m.choise('Voce 6',test1)
m.end()
m.select()

```

<p>Voce 1 Voce 2 Voce 3 Voce 4 Voce 5 Voce 6</p>	<p><i>Da tastiera Left precedente, Right seguente, Enter conferma. Escape se True permette di uscire dal menu senza selezione.</i></p>
<p>Voce 1 Voce 2 Voce 3 Voce 4 Voce 5 Voce 6</p> <p>Selezionato la Voce n.6</p>	<p><i>Click del mouse sulla voce.</i></p>
<p>Voce 1 Voce 2 Voce 3 Voce 4 Voce 5 Voce 6</p> <p>Selezionato la Voce n.4</p>	<p><i>Premendo enter.</i></p>
<p>Hmenu esempio</p>	

Ora un esempio per un menu classico.

<p>File test menu completo.py</p> <pre> from fblib.fbConsole.Screen import Screen from fblib.Costant import * from fblib.fbConsole.Hmenu import * from fblib.fbConsole.Vmenu import * def test1(but): <i>#funzione chiamata dai vari menu a cascata</i> term.print_at(15,10,f'Selezionato la Voce n.{but+1}') term=fbConsole() <i>#terminale</i> scr=Screen(term,fg=WHITE,bg=LBLUE) <i>#definisce lo schermo</i> v1=Vmenu(scr,3,2,fg=RED,bg=YELLOW,escape=True) <i>#menu 1</i> v1.choise('Nuovo',test1) <i>#voci del menu 1</i> v1.choise('Apri',test1) v1.choise('Salva',test1) v1.choise('Salva con nome',test1) v1.choise('Stampa',test1) v1.choise('Esci',test1) v2=Vmenu(scr,3,9,fg=RED,bg=YELLOW,escape=True) <i>#menu 2</i> v2.choise('Voce 1',test1) v2.choise('Voce 2',test1) v2.choise('Voce 3',test1) v2.choise('Voce 4',test1) v2.choise('Voce 5',test1) v3=Vmenu(scr,3,19,fg=RED,bg=YELLOW,escape=True) <i>#menu 3</i> v3.choise('Voce 1',test1) </pre>

```

v3.choise('Voce 2',test1)
v3.choise('Voce 3',test1)
v4=Vmenu(scr,3,29,fg=RED,bg=YELLOW,escape=True)
v4.choise('Voce 1',test1)
v4.choise('Voce 2',test1)
v4.choise('Voce 3',test1)
v4.choise('Voce 4',test1)
v4.choise('Voce 5',test1)
v4.choise('Voce 6',test1)
v5=Vmenu(scr,3,38,fg=RED,bg=YELLOW,escape=True)
v5.choise('Voce 1',test1)
v5.choise('Voce 2',test1)
v5.choise('Voce 3',test1)
v5.choise('Voce 4',test1)
v5.choise('Voce 5',test1)
v5.choise('Voce 6',test1)
m=Hmenu(scr,1,2,fg=RED,bg=YELLOW)
m.choise('File',v1.select)
m.choise('Modifica',v2.select)
m.choise('Formato',v3.select)
m.choise('Visualizza',v4.select)
m.choise(' ? ',v5.select)
m.select()

```

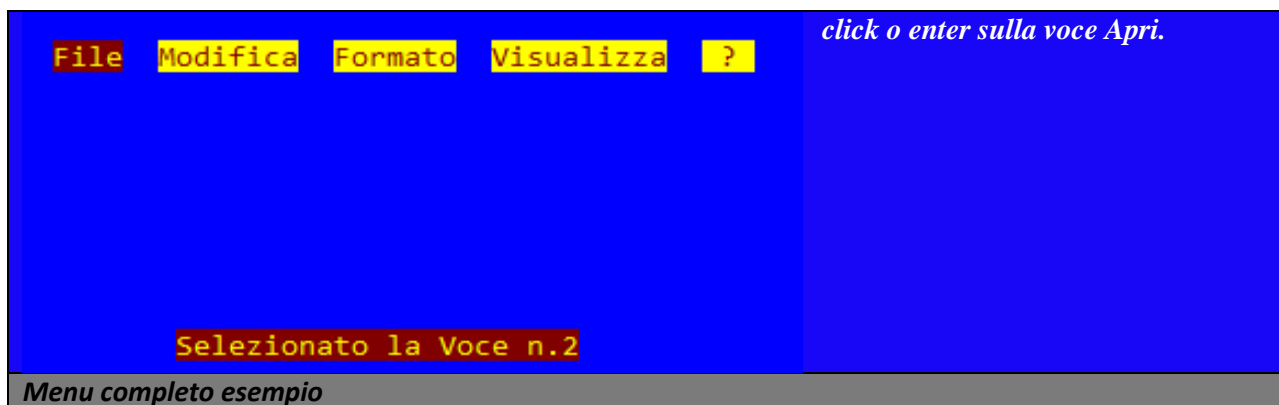
File **Modifica** **Formato** **Visualizza** **?**

*Da tastiera left voce precedente, right voce successiva ,end ultima voce, home prima voce, enter o down seleziona il menu a cascata.
O click sulla voce*

File **Modifica** **Formato** **Visualizza** **?**

Nuovo
Apri
Salva
Salva con
Stampa
Esci

*Menu a cascata da tastiera, up voce precedente,down voce successiva, home prima voce, end ultima voce, enter o click del mouse conferma la selezione.
Left, right,escape esci dal menu a cascata senza selezione.*



Terminiamo con questo esempio la creazione del mini framework per console, potrei continuare a creare altri widgets, ListBox, RadioButton, Slice, barre di scorrimento e altro, ma non era questo lo scopo.

Intendevo dimostrare il potenziale connubio tra Python e Freebasic, un modo semplice ed elegante per estendere Python senza utilizzare i classici C e C++.

Inoltre volevo mostrarvi che le PyApi non sono l'unica via per manipolare oggetti Python da altri linguaggi, i descrittori degli oggetti permettono a un qualsiasi linguaggio di manipolare un oggetto Python facilmente con la sintassi standard del linguaggio.

La libreria di wrapper che ho creato può essere utilizzata per una qualsiasi classe Python da condividere con un altro linguaggio, creando un descrittore creiamo un ambiente virtuale che ci permette di accedere con naturalezza all'oggetto Python contenuto e ci permette di creare altri oggetti dello stesso tipo, possiamo definirlo come una stretta di mano tra i due linguaggi.

Appendice:

Qui troverete degli esempi completi e approfondimenti su gli argomenti trattati:

Array:

Questo esempio completo mostra tutta la potenza della classe fbDim e come sia semplice condividere un array con il Freebasic e come accedervi con una sintassi simile.

La classe è completamente autonoma dalla presenza del Freebasic, questa sicuramente è una cosa molto utile, ci permette di testare il codice direttamente in Python e solo se è necessaria una maggiore velocità di calcolo riscriverlo in Freebasic.

Creare un array in Python e dopo aver assegnato dei valori, passate l'array ad una subroutine in Freebasic al ritorno a Python l'array dovrà contenere i suoi quadrati stampate i risultati.

Esempio completo sull'uso dell'array:

<pre>declare sub Quadrati cdecl alias "Quadrati" (ar() as long) sub Quadrati cdecl alias "Quadrati" (ar() as long) export dim as long i,j for i=Lbound(ar) to Ubound(ar) for j=Lbound(ar,2) to Ubound(ar,2) ar(i,j)=ar(i,j) ^2 next j next i end sub</pre>	<pre>from fbllib.fbTypes import * PATHLIB=r'.\quadrati.dll' lib=CDLL(PATHLIB) data=(2,3,4,5, 6,7,8,9, 10,11,12,13, 14,15,16,17) ar=fbDim(LONG,(1,4),(1,4),data=data) desc=ar.getDesc #array descrittore lib.Quadrati.argtypes=[POINTER(FBArray)] lib.Quadrati(desc) #chiama Freebasic #stampa i risultati for i in range(ar.Lbound(1), ar.Ubound(1)+1): for j in range(ar.Lbound(2), ar.Ubound(2)+1): print(ar[(i,j)],end='t') input('\nPremi Enter per uscire')</pre>
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Output:	Eseguire da console														
4	9	16	25	36	49	64	81	100	121	144	169	196	225	256	289

Python/Fb API:

Le Api sono state già analizzate, tranne la parte dei dizionari che riguarda i metodi keys, values e items.

Vediamo un esempio completo sull'uso di questi metodi.

File pydict_esempio_2.bas: Uso dei dizionari iteratori

```
#include once "pyObj.bi"
declare sub test cdecl alias "test" (byref myDict as py2fbDict)
sub test cdecl alias "test" (byref myDict as py2fbDict)export
  dim t1 as py2fbList,t2 as py2fbList,i as long,v as long,tip as long
  dim b as boolean, iv as longInt, w as wstring ptr=Allocate(50)
  dim f as Single, key as wstring ptr=Allocate(50) 'buffer chiave temporanea
  print "-----Ciclo sulle chiavi-----"
  myDict.keys(t1) 'crea una lista delle chiavi
  for i=0 to t1.len()-1
    t1.getitem(PyStr,i,None) 'legge la chiave dalla lista
    key=t1.wbuff ' la chiave si trova in wbuff e zbuff utilizziamo wbuff
    print "Chiave " & i & " " & *key
  next i
  print
  t1.del(t1.id) 'elimina la lista
  print "-----Ciclo su i valori-----"
  myDict.values(t1) 'crea una lista dei valori
  for i=0 to t1.len()-1
    tipo=t1.isType(i) 'Testa su i tipi di dato contenuti nella lista
    Select case tipo
      case PyBool
        t1.getitem(tipo,i,@b)
        print "Indice="& i & " Boolean "& b
      case PyInt,PyByte,PyUbyte,Pyshort,PyUshort,PyLong,PyLongInt ' uno qualsiasi di questi tipi
        t1.getitem(tipo,i,@iv)
        print "Indice="& i & " Integer "& iv
      case PyFloat,PyDouble,PyUbyte ' uno qualsiasi di questi tipi
        t1.getitem(tipo,i,@f)
        print "Indice="& i & " Single "& f
      case PyBytes,PyStr ' uno qualsiasi di questi tipi
        t1.getitem(tipo,i,None)
        print "Indice="& i & " String "& *t1.wbuff ' *t1.zbuff se utilizzate il buffer di bytes
      case else
        print "Item "& i & " ignorato"
    end select
  next i
  print
  print "Ciclo su chiavi e valori"
```

```

t1.newobj(t2,"[]") 'crea una lista vuota
t1.del(t1.id) 'elimina la vecchia lista
myDict.items(t1) 'crea una lista chiavi-valore
for i=0 to t1.len()-1 'cicla su tutta la lista
  t1.getitem(PyTuple,i,None) 'legge la chiave

  *w="list(" + *t1.wbuff + ")" 'prende la tupla chiave-valore **]
  t2.let(w) 'trasforma la tupla in lista e associa un nuovo valore alla lista t2
  t2.getitem(PyStr,0,None) 'prende la chiave
  *w=*t2.wbuff
  tipo=t2.isType(1)
  Select case tipo
  case PyBool
    t2.getitem(tipo,1,@b)
    print "chiave=" & *w & " value=" & b
  case PyInt,PyByte,PyUbyte,Pyshort,PyUshort,PyLong,PyLongInt 'uno qualsiasi di questi tipi
    t2.getitem(tipo,1,@iv)
    print "chiave=" & *w & " value=" & iv
  case PyFloat,PyDouble,PyUbyte 'uno qualsiasi di questi tipi
    t2.getitem(tipo,1,@f)
    print "chiave=" & *w & " value=" & f
  case PyBytes,PyStr 'uno qualsiasi di questi tipi
    t2.getitem(tipo,1,None)
    print "chiave=" & *w & " value=" & *t2.wbuff 't1.zbuff se utilizzate il buffer di bytes
  case PyTuple,PyList,PySet,PyComplex
    t2.getitem(tipo,1,None)
    print "chiave=" & *w & " valore non convertito " & *t2.wbuff 'non tiene conto di questo tipo
  end select
next i
end sub

```

L'esempio è molto semplice e ben documentato, si può notare come tra Python e Freebasic ci sia un continuo scambio, e come Python lavori in background al Freebasic.

```

**]      *w="list(" + *t1.wbuff + ")" 'prende la tupla chiave-valore **]
        t2.let(w) 'trasforma la tupla in lista e associa un nuovo valore alla lista t2

```

il metodo items ritorna una nuova lista contenente una serie di tuple (chiave,valore), non avendo inviato un secondo parametro di tipo tuple, ho creato una stringa di conversione tramite la classe list di Python.

Nell'esempio Python utilizzo il dizionario:

```
d={'Cane':'Doberman','Eta':10,'Byte':55,'Float':4.5,'Tuple':(5,6.5,'Hei')}
```

Per accedere a gli elementi della tupla devo creare una nuova tupla e se l'oggetto non è disponibile trasformarla in lista per potervi accedere.

File pydict_esempio_2.py: Uso dei dizionari iteratori

```
from fblib.PyObj.pyDict import *
from fblib.PyObj.PyStruct import *
LIBPATH=r'.\pydict_esempio_2.dll'

d={'Cane':'Doberman','Eta':10,'Byte':55,'Float':4.5,'Tuple':(5,6.5,'Hei')}
pyd=PyDictExport(d)
lib=CDLL(LIBPATH)
pyo=pyd.getObj # ricevi il descrittore della lista
lib.test.argtypes=[POINTER(py2fbDict)]
lib.test(pyo)
print("\n")
input('Premi enter per uscire')
```

Output: Questo esempio va eseguito da console

Ciclo sulle chiavi

Chiave 0 Cane

Chiave 1 Eta

Chiave 2 Byte

Chiave 3 Float

Chiave 4 Tuple

Ciclo su i valori

Indice=0 String Doberman

Indice=1 Integer 10

Indice=2 Integer 55

Indice=3 Single 4.5

Item 4 ignorato

Ciclo su chiavi e valori

chiave=Cane value=Doberman

chiave=Eta value=10

chiave=Byte value=55

chiave=Float value= 4.5

chiave=Tuple valore non convertito (5, 6.5, 'Hei')

Premi enter per uscire

Mini framework per console:

In questo contesto daremo uno sguardo alle primitive in fbConsole non menzionate, ho solo accennate.


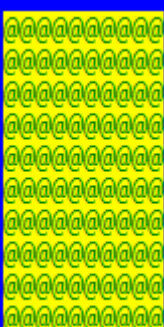
HiByte	ritorna il secondo byte dell'operando	hb= term.HiByte(n)
LoByte	ritorna il primo byte dell'operando	lb= term.LoByte(n)
HiWord	ritorna la seconda Word dell'operando	hb= term.Hi Word (n)
LoWord	ritorna la prima Word dell'operando	lb=term.LoWord (n)
save_pos	salva la posizione del cursore	term.save_pos()
restore_pos	ripristina la posizione del cursore	term.restore_pos()
lprint manda un testo alla stampante		
move_left	sposta il cursore di n posizioni a sinistra	term.move_left(n)
move_righ	sposta il cursore di n posizioni a destra	term.move_right(n)
move_down	sposta il cursore di n posizioni in in basso	term.move_down(n)
move_up	sposta il cursore di n posizioni in alto	term.move_up(n)
cursor_off	cursore invisibile	term.cursor_off()
cursor_on	cursore visibile	term.cursor_on()
inkey	ritorna una stringa rapresentate la prima chiave in attesa nel buffer di tastiera	k= term.inkey()
getkey	ritorna il codice ascii della prima chiave in attesa nel buffer di tastiera	k= term.getkey()
getch	attende la pressione di un tasto e ne ritorna il codice asci	k= term.getch()
getchar	attende la pressione di un tasto e ne ritorna una stringa	k= term.getchar()
multikey	determina lo stato dei tasti in base al codice di scansione	k= term.multikey(code)
color_area	cambia il colore ad un area del video senza toccare l'attributo	
term.color_area(row , col, h, w, fg=col, bg=col)		
allocate	alloca un blocco di memoria	buffer=allocate(integerCount * sizeof(c_int))
deallocate	dealloca un blocco di memoria	deallocate(buffer)
Ajust	giustifica una stringa rispetto a width	st= ajust(s,width,justify)

Esempio d'uso del metodo color_area contenuto nel modulo Write che si serve di color_area in fbConsole.

```
from fblib.fbConsole.fbConsole import *
from fblib.fbConsole.Screen import *
from fblib.fbConsole.Widgets import *
from fblib.Costant import *
from fblib.fbConsole.Write import *

term=fbConsole()
scr=Screen(term,fg=RED, bg=LBLUE)
con=Write(scr)
con.fill(3,3,10,10,'@',fg=WHITE,bg=RED) #riempie l'area con il carattere @
k=term.getch() #attende un tasto ritorna un intero

con.color_area(3,3,10,10,fg=GREEN,bg=YELLOW) #cambia il colore all'area
k=term.getchar() #attende un tasto e ritorna una stringa
```

Output 1	Output 2
	

Il metodo `color_area` cambia il colore ad un area senza dover riscrivere il testo contenuto.

Temina qui questo tutorial dedicato all'utilizzo del Freebasic per estendere Python, spero di avervi incuriosito abbastanza, le potenzialità di questo sottovalutato linguaggio sono notevoli.

Per l'uso che se ne può fare per estendere Python, e la semplicità con cui possiamo manipolare i suoi oggetti grazie ai descrittori, possiamo considerare il Freebasic come ad un'ottima alternativa ai soliti C e C++.