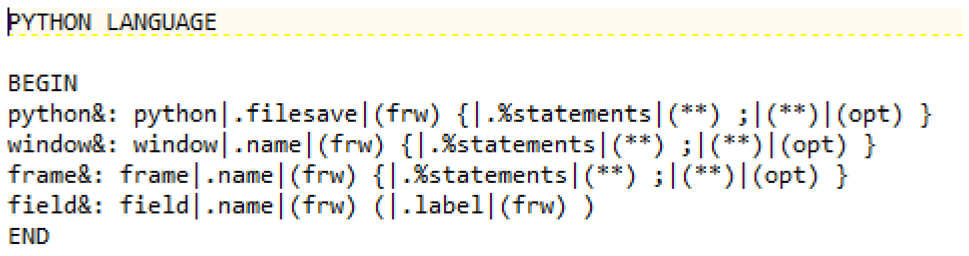
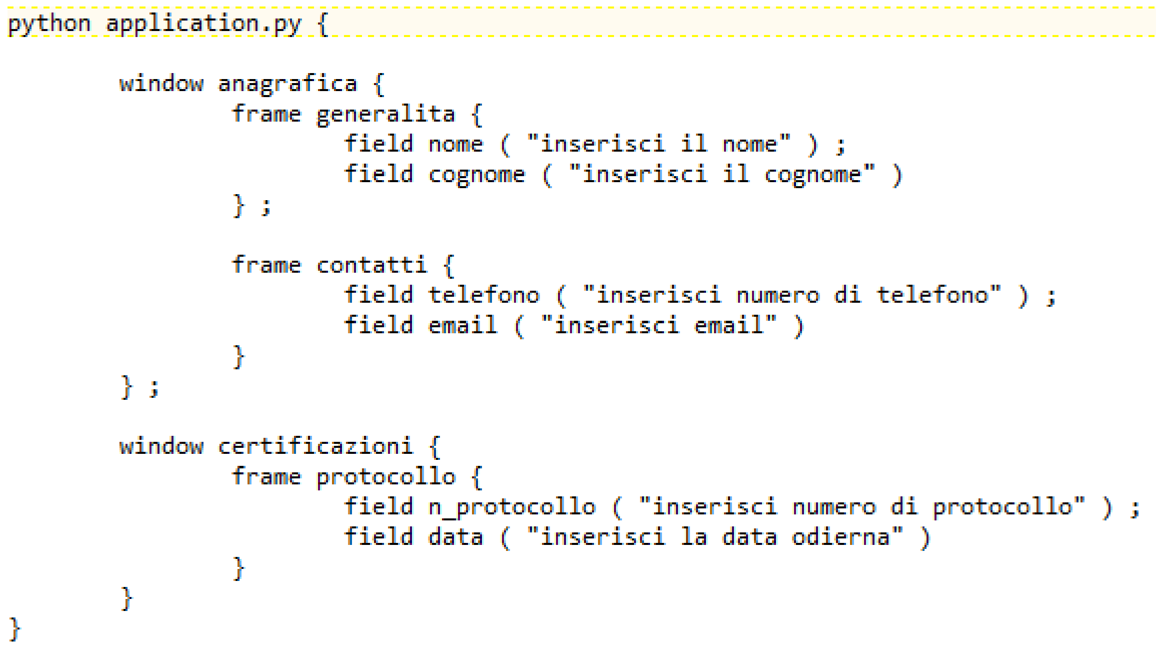
Core 3.0 An example using Python

This simple example just wants to show how it is possible, starting from class models, to have an instantiation and a compilation of these classes which are then composed for the purpose expressed in the special language we want to create. Briefly we can say that we are dealing with the creation of graphical interfaces with fields to fill in. And the example deals with window, frame, and field (text field) components.
We therefore instruct the Parser to recognize the following types of statements:



```
PYTHON LANGUAGE

BEGIN
python&: python|.filesave|(frw) {|.%statements|(**) ;|(**)|(opt) }
window&: window|.name|(frw) {|.%statements|(**) ;|(**)|(opt) }
frame&: frame|.name|(frw) {|.%statements|(**) ;|(**)|(opt) }
field&: field|.name|(frw) (|.label|(frw) )
END
```

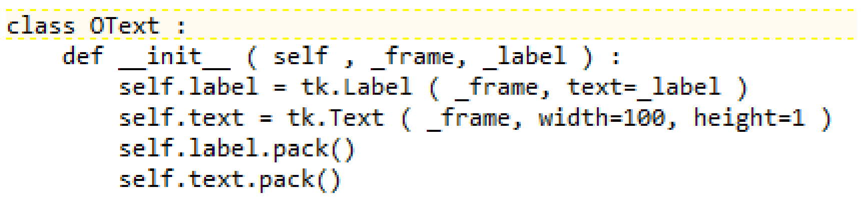
Which will allow us to create applications with a very simple language. The demonstration application in question provides this source:



```
python application.py {

        window anagrafica {
                frame generalita {
                        field nome ( "inserisci il nome" ) ;
                        field cognome ( "inserisci il cognome" )
                } ;

                frame contatti {
                        field telefono ( "inserisci numero di telefono" ) ;
                        field email ( "inserisci email" )
                }
        } ;

        window certificazioni {
                frame protocollo {
                        field n_protocollo ( "inserisci numero di protocollo" ) ;
                        field data ( "inserisci la data odierna" )
                }
        }
}
```

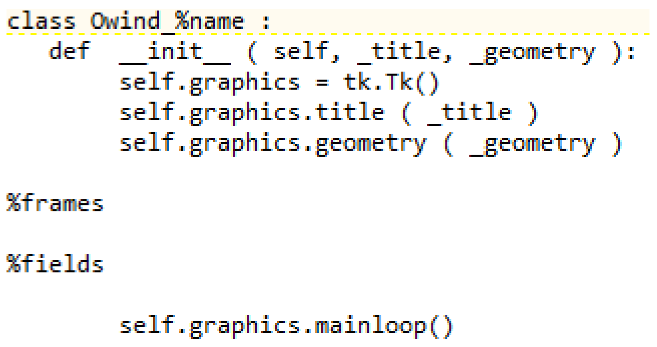
We want to create the graphical application “application.py” made up of two windows, and each window will be multi-step, that is, frames will be displayed, going forward and backward.

The first thing to do is support engineering. Which includes static classes and "template" classes. The static class is the one used for "field" fields and is called Otext.py. The source of which is the following:

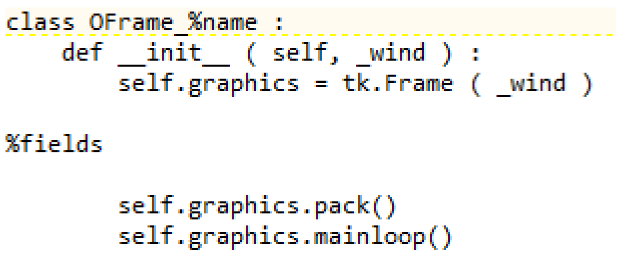


```
class OText :
    def __init__ ( self , _frame, _label ) :
        self.label = tk.Label ( _frame, text=_label )
        self.text = tk.Text ( _frame, width=100, height=1 )
        self.label.pack()
        self.text.pack()
```

While the other two Classes "Window" and "Frame" will be Templates:

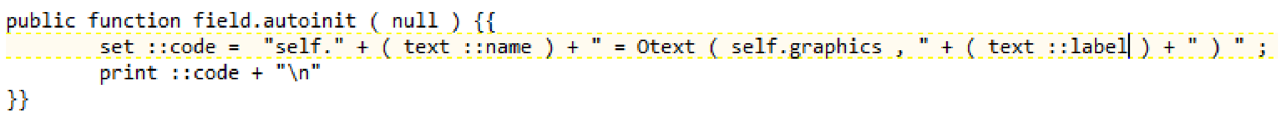


```
class Owind_%name :
   def  __init__ ( self, _title, _geometry ):
        self.graphics = tk.Tk()
        self.graphics.title ( _title )
        self.graphics.geometry ( _geometry )

%frames

%fields

        self.graphics.mainloop()
```

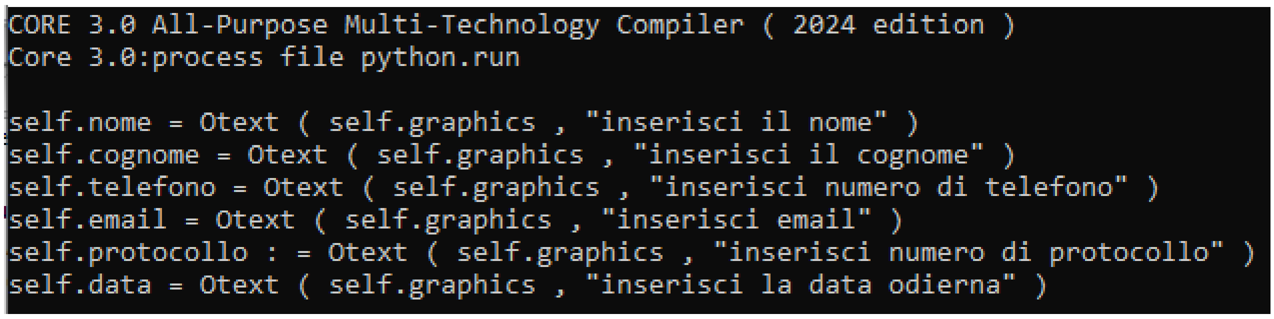


```
class OFrame_%name :
    def __init__ ( self, _wind ) :
        self.graphics = tk.Frame ( _wind )

%fields

        self.graphics.pack()
        self.graphics.mainloop()
```

Let's start and start from the code associated with the fields:



```
public function field.autoinit ( null ) {{
        set ::code =  "self." + ( text ::name ) + " = Otext ( self.graphics , " + ( text ::label ) + " ) " ;
        print ::code + "\n"
}}
```



```
CORE 3.0 All-Purpose Multi-Technology Compiler ( 2024 edition )
Core 3.0:process file python.run

self.nome = Otext ( self.graphics , "inserisci il nome" )
self.cognome = Otext ( self.graphics , "inserisci il cognome" )
self.telefono = Otext ( self.graphics , "inserisci numero di telefono" )
self.email = Otext ( self.graphics , "inserisci email" )
self.protocollo : = Otext ( self.graphics , "inserisci numero di protocollo" )
self.data = Otext ( self.graphics , "inserisci la data odierna" )
```

These are all the fields of all the frames and all the windows, but clearly each line of code must be inserted in its own frame, and the frame code in its own window.

We load into memory the static class and template files that must be instantiated, compiled and correctly composed.

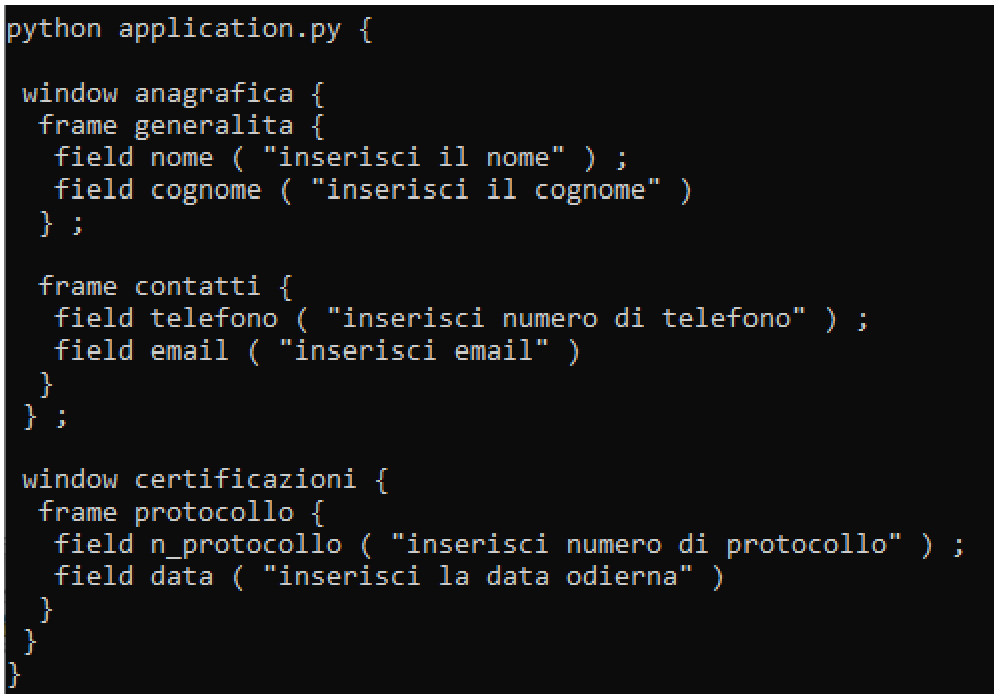
```
public function python.autoinit ( null ) {{

        file read "support\\Oframe.py" , ..frametemplate ;
        file read "support\\Owind.py" , ..windowtemplate

        print ..frametemplate ;
        print ..windowtemplate

}} ;
```

An instance class of your template will be created for each frame and for each window, to which we will add those lines of code we just saw. This is possible in the Core 3.0 Compiler Language using these few lines of code:

```
public function window.autoexec ( null ) {{

        set ::class = ( ..windowtemplate ) ;
        ::class[ "%name" ] = text ::name ;
        ::class[ "%frames" ] = "\+\+\n" + ::frames + "\-\-\n" ;
        ::class[ "%fields" ] = "\+\+\n" + ::fields + "\-\-\n" ;

        print ::class + "\n"

}} ;
```

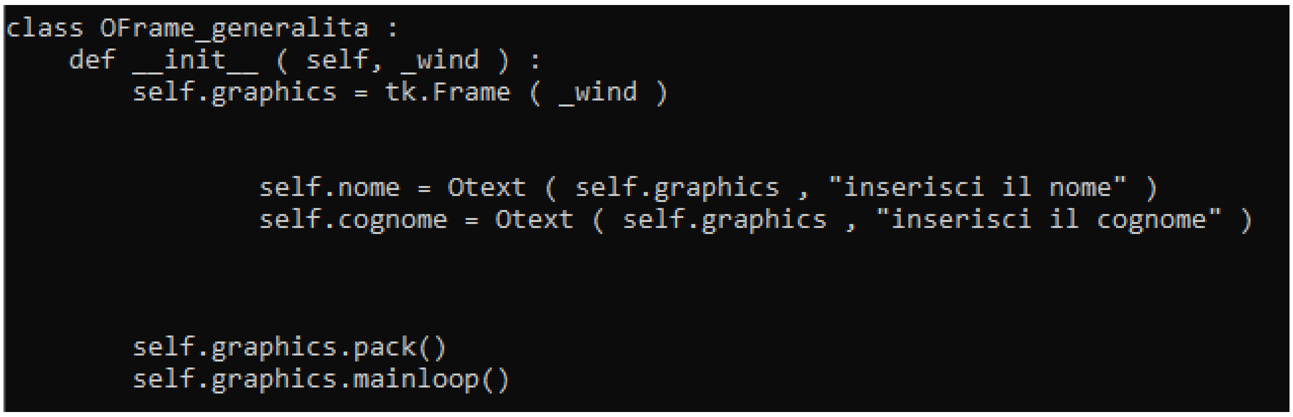
```
public function frame.autoexec ( null ) {{
        ::__CALLER.frames = ::__CALLER.frames + ::code ;

        set ::class = ( ..frametemplate ) ;
        ::class[ "%name" ] = text ::name ;
        ::class[ "%fields" ] = "\+\+\n" + ::fields + "\-\-\n" ;

        print ::class + "\n"
}} ;
```

And the result is to have, in output, from this source:

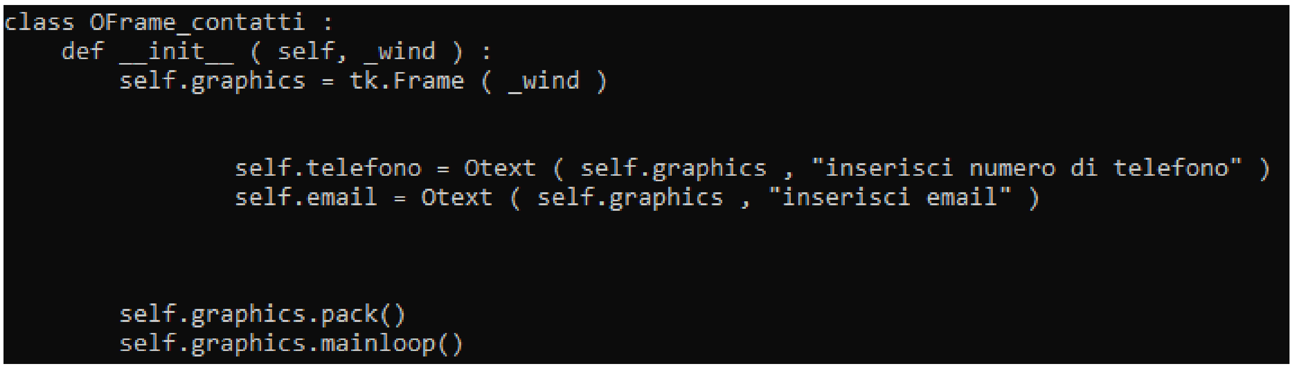


```
python application.py {

 window anagrafica {
  frame generalita {
   field nome ( "inserisci il nome" ) ;
   field cognome ( "inserisci il cognome" )
  } ;

  frame contatti {
   field telefono ( "inserisci numero di telefono" ) ;
   field email ( "inserisci email" )
  }
 } ;

 window certificazioni {
  frame protocollo {
   field n_protocollo ( "inserisci numero di protocollo" ) ;
   field data ( "inserisci la data odierna" )
  }
 }
}
```

The various classes for frames:

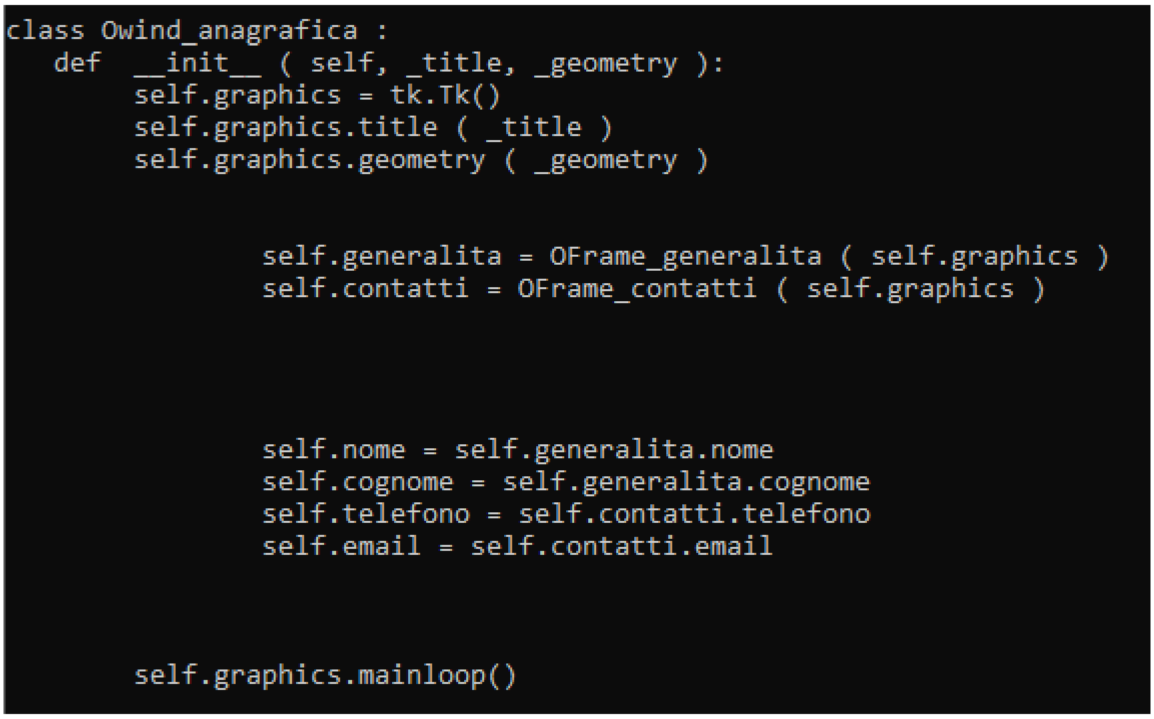


```
class OFrame_generalita :
    def __init__ ( self, _wind ) :
        self.graphics = tk.Frame ( _wind )


                self.nome = Otext ( self.graphics , "inserisci il nome" )
                self.cognome = Otext ( self.graphics , "inserisci il cognome" )



        self.graphics.pack()
        self.graphics.mainloop()
```

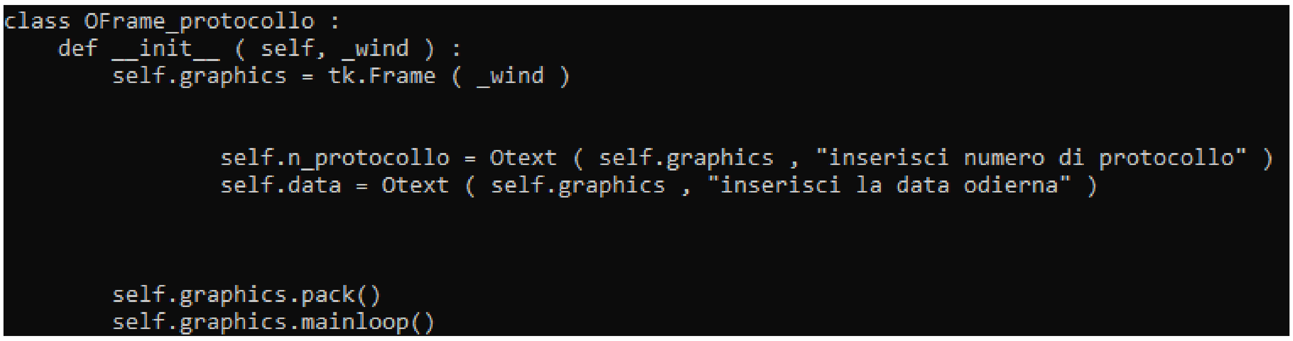


```
class OFrame_contatti :
    def __init__ ( self, _wind ) :
        self.graphics = tk.Frame ( _wind )


                self.telefono = Otext ( self.graphics , "inserisci numero di telefono" )
                self.email = Otext ( self.graphics , "inserisci email" )



        self.graphics.pack()
        self.graphics.mainloop()
```

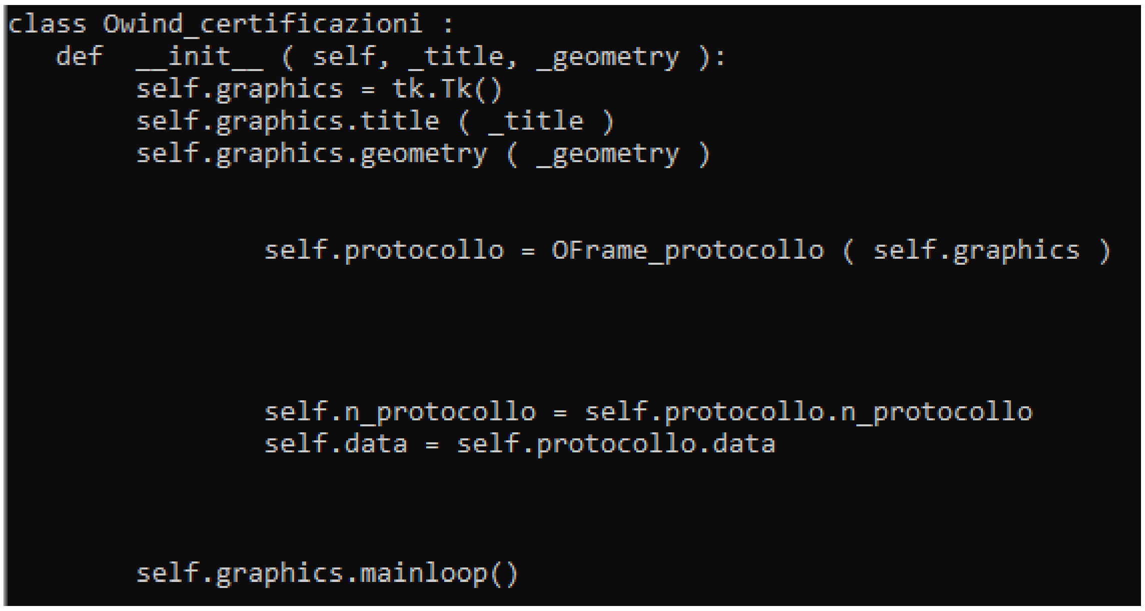
And the related windows that use instances of the template classes:



```
class Owind_anagrafica :
   def  __init__ ( self, _title, _geometry ):
        self.graphics = tk.Tk()
        self.graphics.title ( _title )
        self.graphics.geometry ( _geometry )


                self.generalita = OFrame_generalita ( self.graphics )
                self.contatti = OFrame_contatti ( self.graphics )




                self.nome = self.generalita.nome
                self.cognome = self.generalita.cognome
                self.telefono = self.contatti.telefono
                self.email = self.contatti.email



        self.graphics.mainloop()
```



```
class OFrame_protocollo :
    def __init__ ( self, _wind ) :
        self.graphics = tk.Frame ( _wind )


                self.n_protocollo = Otext ( self.graphics , "inserisci numero di protocollo" )
                self.data = Otext ( self.graphics , "inserisci la data odierna" )



        self.graphics.pack()
        self.graphics.mainloop()
```



```
class Owind_certificazioni :
   def  __init__ ( self, _title, _geometry ):
        self.graphics = tk.Tk()
        self.graphics.title ( _title )
        self.graphics.geometry ( _geometry )


                self.protocollo = OFrame_protocollo ( self.graphics )




                self.n_protocollo = self.protocollo.n_protocollo
                self.data = self.protocollo.data



        self.graphics.mainloop()
```

The Source code in the Core 3.0 Compiler Language Ed. 2024 is the following:

```
file read "python.src" , ..source ;
print ..source + "\n" ;

public function python.autoinit ( null ) {{
      file read "support\\Oframe.py" , ..frametemplate ;
      file read "support\\Owind.py" , ..windowtemplate
}} ;

public function window.autoinit ( null ) {{
      set ::frames = "" ;
      set ::fields = ""
}} ;

public function window.autoexec ( null ) {{

      set ::class = ( ..windowtemplate ) ;
      ::class[ "%name" ] = text ::name ;
      ::class[ "%frames" ] = "\+\+\n" + ::frames + "\-\-\n" ;
      ::class[ "%fields" ] = "\+\+\n" + ::fields + "\-\-\n" ;

      print ::class + "\n"

}} ;

public function frame.autoinit ( null ) {{
      set ::code =  "self." + ( text ::name ) + " = OFrame_" + ( text ::name ) +
" ( self.graphics )\n" ;
      set ::fields = ""
}} ;

public function frame.autoexec ( null ) {{
      ::__CALLER.frames = ::__CALLER.frames + ::code ;

      set ::class = ( ..frametemplate ) ;
      ::class[ "%name" ] = text ::name ;
      ::class[ "%fields" ] = "\+\+\n" + ::fields + "\-\-\n" ;

      print ::class + "\n"
}} ;

public function field.autoinit ( null ) {{
      set ::code =  "self." + ( text ::name ) + " = Otext ( self.graphics , " +
( text ::label ) + " )\n" ;
      set ::field_ref = "self." + ( text ::name ) + " = self." + ( text ::__CALLER.name ) + "." + ( text ::name ) + "\n"
}} ;

public function field.autoexec ( null ) {{
      ::__CALLER.fields = ::__CALLER.fields + ::code ;
      ::__CALLER.__CALLER.fields = ::__CALLER.__CALLER.fields + ::field_ref
}}
```